

Н. К. Смоленцев

Создание Windows-приложений с использованием математических процедур MATLAB

*Рекомендовано научно-методическим советом
по математике и механике УМО
по классическому университетскому образованию РФ
в качестве учебного пособия для студентов высших учебных заведений,
обучающихся по группе математических и механических направлений
и специальностей*



УДК 519.68
ББК 32.913
С51

С51 Смоленцев Н. К.

Создание Windows-приложений с использованием математических процедур MATLAB. – М.: ДМК-Пресс, 2008. – 456 с.: ил.

ISBN 5-94074-122-3***

Данная книга посвящена изложению методов использования математических процедур MATLAB® при создании Windows-приложений, работающих независимо от MATLAB. Книга содержит введение в MATLAB и описание пакетов расширения MATLAB, позволяющих создавать компоненты, которые могут быть использованы при программировании на C++, Borland JBuilder, VBA в Excel и Visual Studio 2005. Кратко изложены необходимые сведения по языкам программирования Java и C#. Подробно рассматриваются примеры создания программ на Borland JBuilder, дополнений к Excel и программ на Visual C#, которые используют математические процедуры, разработанные на MATLAB. Освоение технологии использования математических возможностей MATLAB в других языках программирования позволит создавать полноценные Windows-приложения с развитой графической средой, в которых возможна реализация сложных математических алгоритмов.

Книга предназначена студентам и преподавателям ВУЗов по специальностям, близким к прикладной математике, профессиональным программистам, которые сталкиваются с проблемами реализации математических алгоритмов, и MATLAB-программистам, которые хотят использовать другие языки программирования для реализации алгоритмов MATLAB в виде законченных и независимых от MATLAB приложений.

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

УДК 519.68
ББК 32.913

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 5-94074-122-3***

© Смоленцев Н. К., 2008

© Оформление, издание, ДМК-Пресс, 2008

Краткое содержание

Предисловие	16
Глава 1. ОСНОВЫ РАБОТЫ В СИСТЕМЕ MATLAB®	19
Глава 2. КОМПИЛЯТОР MATLAB® ВЕРСИИ 4.6	85
Глава 3. СОЗДАНИЕ КОМПОНЕНТОВ ДЛЯ JAVA ПРИ ПОМОЩИ JAVA BUILDER	171
Глава 4. MATLAB BUILDER ДЛЯ EXCEL	291
Глава 5. СОЗДАНИЕ КОМПОНЕНТОВ ДЛЯ .NET ПРИ ПОМОЩИ .NET BUILDER	341
Предметный указатель	450
Содержание компакт-диска	451
Литература	455

Содержание

Предисловие	16
--------------------------	-----------

Глава 1. Основы работы в системе MATLAB®	19
---	-----------

1.1. Система компьютерной математики MATLAB®	20
--	----

1.1.1. Основные компоненты системы MATLAB	21
---	----

1.1.2. Инструментальные средства рабочего стола MATLAB	21
---	----

1.1.3. Константы и системные переменные MATLAB	27
--	----

1.1.4. Типы данных MATLAB	28
---------------------------------	----

1.2. Основы работы с MATLAB®	31
------------------------------------	----

1.2.1. Запуск MATLAB и начало работы	31
--	----

1.2.2. Задание массивов	33
-------------------------------	----

Задание одномерных массивов	33
-----------------------------------	----

Задание двумерных массивов	34
----------------------------------	----

1.2.3. Операции над массивами	36
-------------------------------------	----

1.2.4. Решение систем линейных уравнений	39
--	----

Символьная математика пакета расширения Symbolic Math	40
--	----

1.2.5. М-файлы	42
----------------------	----

1.2.6. Чтение и запись текстовых файлов	44
---	----

1.2.7. Операции с рабочей областью и текстом сессии	47
--	----

1.3. Массивы символов	49
-----------------------------	----

1.3.1. Задание массива символов	49
---------------------------------------	----

1.3.2. Общие функции	49
----------------------------	----

1.3.3. Проверка строк	51
-----------------------------	----

1.3.4. Операции над строками	51
1.3.5. Преобразование чисел в символы и обратно	52
1.3.6. Функции преобразования систем счисления	54
1.3.7. Вычисление строковых выражений	55
1.4. Массивы ячеек	55
1.4.1. Создание массивов ячеек	56
1.4.2. Доступ к данным в ячейках	58
1.4.3. Вложенные массивы ячеек	60
1.4.4. Массивы ячеек, содержащих структуры	61
1.4.5. Многомерные массивы ячеек	62
1.5. Массивы структур	62
1.5.1. Построение структур	63
1.5.2. Доступ к полям и данным структуры	64
1.5.3. Многомерные массивы структур	67
1.6. Программирование в среде MATLAB	67
1.6.1. М-функции	67
1.6.2. Операторы системы MATLAB	73
1.6.3. Управление последовательностью исполнения операторов	76
1.6.4. Вычисление символьных выражений	80
1.6.5. Ошибки и предупреждения	81
1.6.6. Повышение эффективности обработки М-файлов ...	82

Глава 2. Компилятор MATLAB®

версии 4.6

2.1. Основы работы с Компилятором MATLAB®	86
2.1.1. Назначение Компилятора MATLAB	86
2.1.2. Инсталляция и конфигурирование	87
2.1.3. Пример использования Компилятора	88

Среда разработки Deployment Tool	88
Создание приложения	90
Использование команды msc	93
2.1.4. Среда выполнения компоненты MATLAB, библиотека MCR	94
2.1.5. Файлы, создаваемые Компилятором	95
Технологический файл компоненты (CTF)	96
Файлы обертки	96
2.2. Процесс создания компонента MATLAB®	97
2.2.1. Процесс создания компонента	97
2.2.2. Управление путями при компиляции	98
2.3. Работа с msc и mbuild	100
2.3.1. Работа с msc	101
Обычное использование msc	101
Опции msc	102
Порядок использования опций	105
Использование файлов групп	106
Создание файлов обертки	107
2.3.2. Использование псевдокомментариев	108
2.3.3. Несколько полезных замечаний	109
2.3.4. Функция mbuild	110
2.4. Примеры создания автономных приложений и библиотек	111
2.4.1. Библиотеки совместного использования	111
Библиотека совместного использования C	111
Функции, создаваемые из m-файлов	118
Использование varargin и varargout в интерфейсе m-функции	119
C++ библиотека совместного использования	119
2.4.2. Создание автономных приложений	122
Создание кода только из m-файлов	123
Объединение M-файлов и кода C или C++	124

2.5. Классы C++ Компилятора 4.6 MATLAB®	127
2.5.1. Основные типы данных	127
2.5.2. Класс mxArray	128
Конструкторы	128
Методы копирования	130
Методы получения информации о массиве	130
Методы сравнения	132
Методы доступа к элементам массива mxArray	132
Операторы	134
Статические методы	135
2.5.3. Класс mwString	136
Конструкторы	136
Методы	136
Операторы	136
2.5.4. Класс mxArrayException	137
Конструкторы	137
Методы	138
Операторы	138
2.6. Внешние интерфейсы	138
2.6.1. Процедуры доступа к MAT-файлам	139
2.6.2. Операции с массивами mxArray	139
2.7. Передача значений между C/C++ double, mxArray и mxArray	141
2.7.1 Преобразование значений между C/C++ double и mxArray	142
Преобразование скаляров	142
Преобразование векторов	142
Преобразование матриц	143
2.7.2 Преобразование значений из C/C++ double в mxArray	143
Преобразование скаляров	143
Преобразование векторов	143
Преобразование матриц	143

2.7.3 Преобразование значений из mxArray в C/C++ double	144
Преобразование скаляров	145
Преобразование векторов	145
Преобразование матриц	145
2.7.4. Вспомогательные функции преобразования данных ..	145
Преобразование значений из C/C++ double в mxArray	146
Преобразование значений из mxArray в C/C++ double ..	147
Преобразование из C/C++ double в mxArray	149
Преобразование mxArray в C/C++ double	149
Пример создания заголовочного файла	150
2.8. Математическая библиотека C++ MATLAB® 6.5	151
2.8.1. Расположение файлов математической библиотеки C++	152
2.8.2. Документация Математической библиотеки MATLAB C++	153
2.8.3. Знакомство с Математической библиотекой MATLAB C++	154
2.8.4. Работа с массивами mxArray	155
Числовые массивы	156
2.8.5. Подключение математических библиотек к Borland C++ Builder	160
2.8.6. Примеры приложений использующих математические библиотеки	161
Чтение, обработка и запись данных	162
Построение графиков данных mxArray	166

Глава 3. Создание компонентов для Java при помощи Java Builder

3.1. Язык программирования Java	172
3.1.1. Основные элементы программирования на Java ...	173
Первая программа на Java	173
Комментарии и имена	175
Константы	175

Типы данных	176
Операции	182
Операторы	184
Массивы	187
3.1.2. Классы в Java	189
Понятие класса	189
Как описать класс и подкласс	191
Окончательные члены и классы	193
Класс Object	193
Опертор new	194
Конструкторы класса	194
Статические члены класса	195
Метод main()	196
Где видны переменные	196
Вложенные классы	197
Пакеты и интерфейсы	197
Структура Java-файла	200
3.2. Введение в Java Builder	201
3.2.1. Общие сведения о MATLAB Builder для Java	201
3.2.2. Графический интерфейс пользователя MATLAB Builder для Java	203
3.2.3. Создание компонента Java	205
3.2.4. Использование командной строки для создания компонента	208
3.2.5. Разработка приложения, использующего компонент	210
3.2.6. Обсуждение примера магического квадрата	213
3.3. Массивы MATLAB в Java	214
3.3.1. Использование методов класса MArray	215
Построение MArray	216
Методы получения информации о MArray	216
Методы получения и задания данных в MArray	218
Методы копирования, преобразования и сравнения массивов MArray	220

Методы для использования на разреженных массивах <code>MWArray</code>	221
3.3.2. Использование <code>MWNumericArray</code>	222
Построение различных типов числовых массивов	223
Методы уничтожения <code>MWNumericArray</code>	227
Методы для получения информации о <code>MWNumericArray</code>	227
Методы доступа к элементам и задания элементов <code>MWNumericArray</code>	228
Методы копирования, преобразования и сравнения массивов <code>MWNumericArray</code>	232
Методы возвращения значений специальных констант	233
Методы <code>toArray</code> и <code>getTypeArray</code> преобразования массивов данных	234
Методы работы с разреженными массивами <code>MWNumericArray</code>	235
3.3.3. Работа с логическими, символьными и массивами ячеек	237
3.3.4. Использование <code>MWClassID</code>	239
Поля <code>MWClassID</code>	240
Методы класса <code>MWClassID</code>	240
3.3.5. Использование класса <code>MWComplexity</code>	240
3.4. Примеры приложений Java	241
3.4.1. Пример спектрального анализа	241
Построение компонента	241
Разработка приложения, использующего компонент	244
3.4.2. Пример матричной математики	248
Построение компонента	251
Разработка приложения, использующего компонент	250
3.5. Некоторые вопросы программирования	255
3.5.1. Импорт классов и создание экземпляра класса	255

3.5.2. Правила обращения к методам Java Builder	255
Стандартный интерфейс	256
Интерфейс <code>mlx</code>	257
3.5.3. Правила преобразования данных MATLAB и Java ...	258
Автоматическое преобразование в тип MATLAB	259
Преобразование типов данных вручную	260
3.5.4. Аргументы методов Java Builder	262
Передача неопределенного числа параметров	262
Получение информации о результатах методов	264
Передача объектов Java по ссылке	266
3.5.5. Обработка ошибок	266
Обработка исключений <code>MWException</code>	266
Обработка общих исключений	268
3.5.6. Управление собственными ресурсами	269
Использование «сборки мусора» JVM	269
Использование метода <code>dispose</code>	269
3.6. Среда проектирования JBuilder	271
3.7. Примеры создания приложений	
с использованием классов Java Builder	277
3.7.1. Объем n -мерного шара	
и площадь $(n-1)$ -мерной сферы	278
Создание компонента Java Builder	278
Создание приложения JBuilder	279
Создание пакета	
для распространения приложения	283
3.7.2. Магический квадрат	285
Глава 4. MATLAB Builder для Excel	291
4.1. Введение	292
4.1.1. Создание компонента для Excel	294
4.1.2. Установка компонента на другие машины	297

4.1.3. Мастер функций	298
4.1.4. Работа с компонентами в Excel	303
4.2. Общие вопросы создания компонент Excel Builder	305
4.2.1. Процедура создания компонента	305
4.2.2. Регистрация компонента	306
4.2.3. Разработка новых версий	307
4.3. Пример создания дополнения для спектрального анализа	308
4.3.1. Построение компонента	308
4.3.2. Подключение компонента к Excel с использованием VBA	311
4.3.3. Создание формы Visual Basic	314
4.3.4. Добавление пункта меню Spectral Analysis в Excel	317
4.3.5. Тестирование дополнения	319
4.3.6. Упаковка и распространение дополнения	320
4.3.7. Обсуждение программы VBA	321
4.3.8. Использование флагов	324
4.4. Библиотека утилит Excel Builder	326
4.4.1. Функции MATLAB Builder для Excel	326
4.4.2. Библиотека утилит Excel Builder	327
Класс MWUtil	328
Класс MWFlags	328
Class MWStruct	330
Класс MWField	330
Класс MWComplex	330
Class MWSparse	331
Класс MWArg	331
Перечисления	332
4.5. Справка по VBA	333

Глава 5. Создание компонентов для .NET при помощи .NET Builder	341
5.1. Среда разработки Microsoft .NET	342
5.1.1. Основные элементы платформы Microsoft .NET	342
Новые понятия	343
5.1.2. Среда выполнения .NET Framework	346
5.1.3. Стандартная система типов	347
5.1.4. Общая спецификация языков программирования	349
5.2. Основы языка C#	349
5.2.1. Элементы синтаксиса языка C#	350
Алфавит и слова C#	350
Структура программы C#	351
Переменные и константы C#	352
Объявление переменных. Область видимости и время жизни	353
5.2.2. Система типов	354
Значимые и ссылочные типы	355
Системные встроенные типы	356
Приведение типов	357
Логический тип	359
Строковые и символьные типы	359
Перечисления	360
Организация системы типов	361
5.2.3. Массивы	362
5.2.4. Операции и выражения	365
5.2.5. Управление последовательностью выполнения операторов	367
Оператор if...else условного перехода	367
Оператор switch	368
Оператор цикла while	369
Оператор цикла do... while	369

Оператор цикла for	370
Операторы break и continue	370
5.2.6. Класс и структура	371
Классы	371
Структуры	375
Интерфейсы	375
5.2.7. Отражение	376
5.3. Введение в .NET Builder	377
5.3.1. Библиотека классов .NET MWArray	379
5.3.2. Правила преобразования данных	381
5.3.3. Интерфейсы, создаваемые .NET Builder	384
5.3.4. Задание сборки компонента и пространства имен ...	387
5.4. Создание консольных приложений	387
5.4.1. Пример магического квадрата	388
Создание .NET компонента	388
Использование компонента в приложении	391
5.4.2. Пример матричной математики	395
Создание .NET компонента	395
Использование компонента в приложении	396
5.4.3. Использование командной строки для создания компоненты .NET	400
5.5. Некоторые вопросы программирования с компонентами .NET Builder	402
5.5.1. Обязательные элементы программы	402
5.5.2. Передача входных параметров	404
Примеры передачи входных параметров	405
Передача массива вводов	406
Обработка глобальных переменных MATLAB	407
Обработка возвращаемых значений	407
Использование запросов MWArray	408
5.5.3. Обработка ошибок	409
5.5.4. Управление родными ресурсами	410

5.5.5. Преобразования между типами C# и MWNumericArray	412
Преобразование скаляров	412
Преобразование векторов	413
Преобразование матриц	414
5.6. Среда разработки Visual Studio 2005	415
5.6.1. Создание нового проекта	419
5.7. Программирование на Visual Studio 2005 с использованием математических процедур MATLAB	420
5.7.1. Вычисление интегралов	421
Создание .NET компонента	421
Создание приложения	422
5.7.2. Решение обыкновенных дифференциальных уравнений	429
Создание .NETкомпонента ODE	429
Создание Windows-приложения	432
5.7.3. Открытие, обработка и сохранение файлов	440
Создание .NETкомпонента	440
Создание приложения	442
Предметный указатель	450
Содержание компакт-диска	451
Литература	455

Предисловие

Как известно, система MATLAB® является одной из наиболее мощных универсальных систем компьютерной математики. Возможности системы MATLAB уникальны. Список основных функций MATLAB (не включая специализированных функций пакетов расширений) содержит более 1000 наименований. Кроме встроенных процедур, система MATLAB имеет чрезвычайно легкий в использовании язык программирования высокого уровня, основанный на таких мощных типах данных, как многомерные числовые массивы, массивы символов, ячеек и структур MATLAB. Программы, написанные на m-языке MATLAB работают только в среде MATLAB, однако в системе MATLAB предусмотрены возможности создания приложений на других языках программирования, которые используют процедуры, написанные на m-языке MATLAB. До выпуска MATLAB 6.5 для этих целей предназначались математические библиотеки C/C++ MATLAB, которые позволяли создавать автономные C/C++ приложения, использующие функции MATLAB.

Начиная с выпуска MATLAB 7, корпорация MathWorks отказалась от дальнейшего использования математических библиотек C/C++, существенно изменив и расширив возможности пакета расширения MATLAB Compiler – Компилятора MATLAB. При этом были разработаны такие расширения MATLAB Compiler, как: MATLAB Builder for Java – пакет расширения для создания и использования компонентов для языка Java; MATLAB Builder for Excel – пакет расширения для создания и использования дополнений (Add-Ins) Excel; MATLAB Builder for .NET – пакет расширения для создания и использования компонентов в среде .NET Framework. Для обеспечения работы компонентов, созданных Компилятором MATLAB, разработана универсальная среда MCR исполнения компонентов MATLAB. Программа, созданная на других языках программирования и использующая скомпилированные функции MATLAB, выполняется только с MCR. Сама система MATLAB для работы приложения не требуется. Созданные компоненты MATLAB и приложения, их использующие, могут свободно распространяться вместе со средой исполнения MCR.

В данной книге рассматривается использование Компилятора MATLAB и его расширений: MATLAB Builder for Java, MATLAB Builder for Excel и MATLAB Builder for .NET. Данное издание является продолжением книги [ППС], в которой изложены математические библиотеки C/C++ MATLAB и показано их использование для создания Windows-приложений на Borland C++ Builder. Особенности программирования для систем, отличных от Windows, можно найти в документации MATLAB®.

Рассмотрим кратко содержание данной книги по главам.

Первая глава содержит первоначальные сведения о системе MATLAB. Она предназначена для читателей, которые владеют программированием, но не на MATLAB. Глава содержит описание работы с числовыми массивами, массива-

ми символов, ячеек и структур, а также основы программирования в среде MATLAB.

Вторая глава посвящена описанию пакета расширения MATLAB Compiler версии 4.6 (для MATLAB R2007a). Возможности Компилятора огромны. Компилятор MATLAB поддерживает почти все функциональные возможности MATLAB. Компилятор MATLAB из m-файлов MATLAB может создать C или C++ автономные консольные приложения и библиотеки общего доступа (dll). Изложение материала сопровождается обсуждением тестовых примеров MATLAB (эти примеры входят в инсталляционный пакет MATLAB Compiler). В конце главы дано краткое описание математических библиотек C/C++ MATLAB 6.5 и приведены примеры их использования.

Глава 3 посвящена созданию компонентов для Java и приложений Java, которые используют компоненты MATLAB. Вначале кратко излагаются необходимые сведения о языке Java. Подробно на примерах рассматривается создание компонентов и консольных Java-приложений, которые используют созданные компоненты (учебные примеры MATLAB). В конце главы подробно рассмотрено создание Windows-приложений на Borland JBuilder, которые используют упакованные в компоненты процедуры MATLAB.

В главе 4 рассматривается создание компонентов для Excel и VBA-приложений, которые используют эти компоненты MATLAB. Система MATLAB предлагает свое, фирменное, дополнение к Excel для использования при работе в Excel скомпилированных функций MATLAB. Это дополнение называется «Мастер функций». Рассмотрена работа с Мастером функций. Кроме того, рассмотрено создание на VBA собственных дополнений для решения различных математических задач с данными Excel.

Глава 5 посвящена созданию .NET-компонентов и приложений .NET, которые используют компоненты MATLAB. Вначале кратко излагаются необходимые сведения о .NET Framework и языке программирования C#. Подробно на примерах рассматривается создание компонентов и консольных C#-приложений, которые используют созданные компоненты (учебные примеры MATLAB). В конце главы подробно рассмотрено создание Windows-приложений на Visual Studio 2005, которые используют процедуры MATLAB.

Книга имеет приложение в виде компакт-диска с исходными текстами примеров программ, рассматриваемых в данной книге.

Книга предназначена студентам и преподавателям ВУЗов по специальностям, близким к прикладной математике, профессиональным программистам, которые сталкиваются с проблемами реализации математических алгоритмов и MATLAB-программистам, которые хотят использовать другие языки программирования для реализации алгоритмов MATLAB в виде законченных и независимых от MATLAB приложений.

Освоение технологии использования колоссальных математических возможностей MATLAB в других языках программирования позволит создавать полноценные Windows-приложения с развитой графической средой, в которых возможна

реализация сложных математических алгоритмов для решения научно-технических задач.

Книга написана при содействии корпорации MathWorks в соответствии с программой MathWorks поддержки книг, посвященных MATLAB®. Автор выражает благодарность компании MathWorks за предоставленную возможность использования документации и лицензионного программного обеспечения MATLAB® R2007a для написания этой книги.

Основы работы в системе MATLAB®

1.1. Система компьютерной математики MATLAB®	20
1.2. Основы работы с MATLAB®	31
1.3. Массивы символов	49
1.4. Массивы ячеек	55
1.5. Массивы структур	62
1.6. Программирование в среде MATLAB	67

MATLAB® – это одна из старейших систем компьютерной математики, построенная на применении матричных операций. Название MATLAB происходит от слов *matrix laboratory* (матричная лаборатория). Матрицы широко применяются в сложных математических расчетах. Однако в настоящее время MATLAB далеко вышла за пределы специализированной матричной системы и стала одной из наиболее мощных универсальных систем компьютерной математики. В MATLAB используются такие мощные типы данных, как многомерные числовые массивы, массивы символов, ячеек и структур, что открывает широкие возможности применения системы во многих областях науки и техники. В данной главе мы кратко рассмотрим некоторые вопросы работы в системе MATLAB.

Описание системы MATLAB® и ее применения к решению различных задач математического анализа, обработки данных, решения дифференциальных уравнений и к графике можно найти в *Help MATLAB* и в любом руководстве по MATLAB, см. например [ККШ], [Пот], [ЧЖИ], [Кр], [Ма], [Ко], [Д], [ГЦ], [Ан] и [ППС]. Отметим также интернет-ресурсы [W].

1.1. Система компьютерной математики MATLAB®

Система MATLAB® была разработана в конце 70-х гг. и широко использовалась на больших ЭВМ. В дальнейшем были созданы версии системы MATLAB для персональных компьютеров с различными операционными системами и платформами. К расширению системы были привлечены крупнейшие научные школы мира в области математики, программирования и естествознания. Одной из основных задач системы является предоставление пользователям мощного языка программирования высокого уровня, ориентированного на математические расчеты и способного превзойти возможности традиционных языков программирования для реализации численных методов.

Система MATLAB® объединяет вычисление, визуализацию и программирование в удобной для работы окружающей среде, где задачи и решения выражаются в привычном математическом виде. Обычные области использования MATLAB: математика и вычисления, разработка алгоритмов, моделирование, анализ данных и визуализация, научная и техническая графика, разработка приложений. В университетских кругах MATLAB® – это стандартный учебный инструмент для вводных и продвинутых курсов в математике, в прикладных исследованиях и науке. В промышленности, MATLAB – это инструмент высокой производительности для исследований, анализа и разработки приложений.

Поразительная легкость модификации системы и возможность ее адаптации к решению специфических задач науки и техники привели к созданию десятков пакетов прикладных программ (Toolboxes), намного расширивших сферы применения системы. Пакеты расширений представляют собой обширные библиотеки функций MATLAB® (m-файлы), которые созданы для использования MATLAB в решении специальных задач. Пакеты расширения (их число более 50) включают такие инте-

ресные области, как обработка сигналов, системы управления, нейронные сети, нечеткая логика, биоинформатика, вейвлеты, моделирование и много других.

Возможности системы MATLAB® уникальны. Список основных функций MATLAB® (не включая специализированных функций пакетов расширений) содержит более 1000 наименований.

1.1.1. Основные компоненты системы MATLAB

Система MATLAB состоит из пяти главных частей.

Среда разработки. Это набор инструментов и средств обслуживания, которые помогают использовать функции и файлы MATLAB. Многие из этих инструментов – графические пользовательские интерфейсы. Среда разработки включает рабочий стол MATLAB и командное окно, окно истории команд, редактор-отладчик, и браузеры для просмотра помощи, рабочего пространства, файлов и путей поиска.

Библиотека математических функций MATLAB. Это обширное собрание вычислительных алгоритмов от элементарных функций типа суммы, синуса, косинуса и комплексной арифметики, до более сложных функций типа транспонирования, обращения матриц, нахождения собственных значений матриц, функций Бесселя и быстрого преобразования Фурье.

Язык MATLAB. Это язык высокого уровня, основанный на работе с матричными массивами, с функциями управления потоками, структурами данных, вводом/выводом и объектно-ориентированным программированием. Он позволяет быстро и легко освоить создание небольших программ, а также имеется возможность создания полных и сложных прикладных программ.

Графика. MATLAB имеет обширные средства для графического отображения векторов и матриц, а также создания аннотаций и печати этих графиков. Графика MATLAB включает функции высокого уровня для двумерной и трехмерной визуализации данных, обработки изображений, анимации, и презентационной графики. Графика MATLAB также включает функции низкого уровня, которые позволяют полностью настроить вид графики и создавать законченные графические интерфейсы пользователя на ваших приложениях MATLAB.

MATLAB API (Application Program Interface, интерфейс прикладного программирования). Это библиотека, которая позволяет писать программы C и Fortran совместно с MATLAB. API включает средства для вызова подпрограмм из MATLAB (динамическая связь), вызывая MATLAB как вычислительный механизм, и для чтения и записи MAT-файлов.

1.1.2. Инструментальные средства рабочего стола MATLAB

При запуске MATLAB, появляется *рабочий стол* MATLAB. Он содержит инструменты (графические пользовательские интерфейсы) для управления файлами, переменными и приложениями, связанными с MATLAB. Рабочий стол MATLAB имеет вид как на рис. 1.1.1.

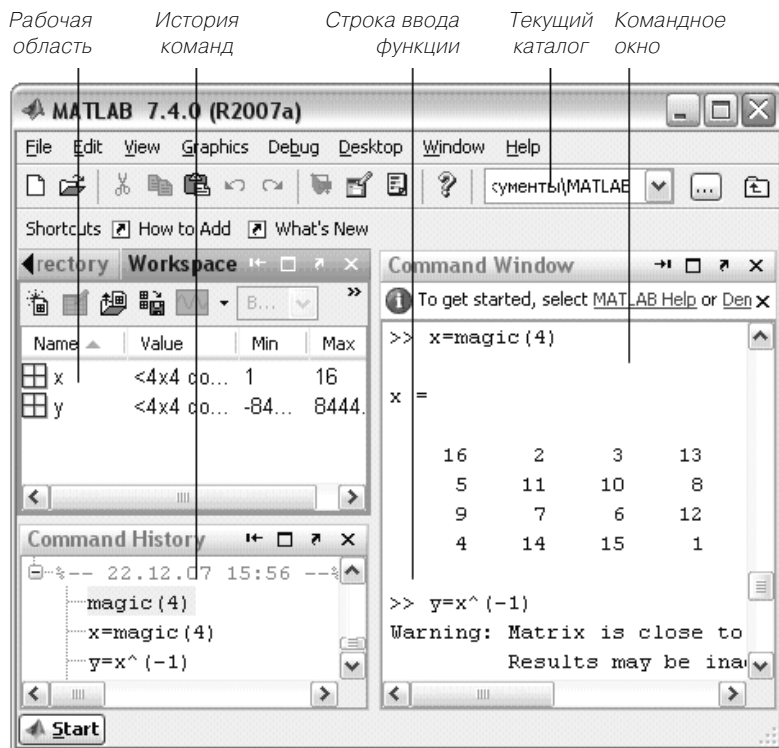


Рис. 1.1.1. Рабочий стол MATLAB

Инструментальные средства рабочего стола MATLAB включают следующие компоненты:

- командное окно (Command Window);
- браузер рабочей области (Workspace Browser);
- редактор массива (Array Editor);
- история команд (Command History);
- браузер текущего каталога (Current Directory Browser);
- кнопка старта (Start);
- браузер справки (Help Browser);
- редактор/отладчик (Editor/Debugger);
- профилировщик (Profiler).

Замечание 1. Некоторые характеристики для настольных инструментальных средств можно определить, выбирая **Preferences** из меню **File**.

Рассмотрим подробнее инструментальные средства рабочего стола.

Командное окно (Command Window). Используется для ввода команд, переменных и выполнения функции и m-файлов. Команду можно вызвать в строке ввода – это последняя строка с символом приглашения (**>>**). Выполненная команда перестает быть активной, она недоступна для редактирования. Ранее ис-

полненные команды можно ввести в командную строку либо из окна истории команд, либо пролистывая их в командной строке клавишами «стрелка вверх/вниз».

Браузер рабочей области (Workspace Browser). Рабочая область MATLAB состоит из набора переменных (массивов) созданных в течение сеанса MATLAB и сохраненных в памяти (см. рис. 1.1.1). Переменные добавляются к рабочей области в результате выполнения функций, m-файлов, или при загрузке сохраненных ранее рабочих областей. В рабочей области содержится информация о каждой переменной, см. рис. 1.1.1. Содержимое этой области можно просмотреть также из командной строки с помощью команд `who` и `whos`. Команда `who` выводит только имена переменных, а команда `whos` – информацию о размерах массивов и типе переменной.

Чтобы удалить переменные из рабочей области, достаточно выбрать переменную и выполнить **Delete** в меню **Edit**, либо в, открывающемся правой кнопкой мыши, контекстном меню. Чтобы сохранить рабочую область в файле, который может быть загружен в следующем сеансе MATLAB, достаточно выбрать **Save Workspace As** в меню **File**, или использовать функцию `save`. Рабочая область сохраняется в бинарном MAT-файле. Чтобы прочитать данные из MAT-файла, нужно выбрать **Import Data** из меню **File**.

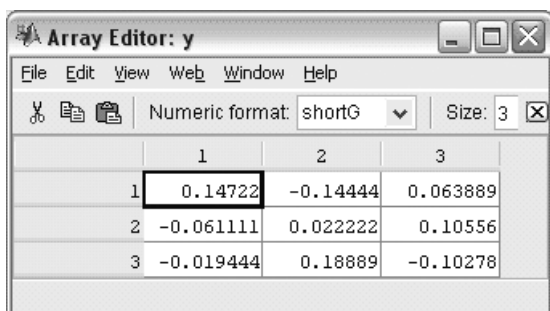


Рис. 1.1.2. Редактор массива

Редактор массивов. Если дважды щелкнуть мышкой по переменной в рабочей области, то эта переменная отобразится в *редакторе массива*. Он используется для визуального просмотра и редактирования одно- или двумерных числовых массивов, массивов строк и массивов ячеек строк, которые находятся в рабочей области.

История команд (Command History). Инструкции и команды, которые вводятся командном окне, регистрируются в окне истории команд. Можно рассмотреть ранее выполненные команды, копировать и выполнить выбранные команды. Чтобы сохранить вводы и выводы сессии MATLAB в файл используется функция `diary`.

Браузер текущего каталога (Current Directory). М-файл, который можно выполнить в командном окне, должен находиться или в *текущем* каталоге или на пути поиска файлов. Для быстрого изменения текущего каталога можно использовать поле **Current Directory** в инструментальной панели рабочего стола. Для просмотра содержания текущего каталога используется браузер текущего каталога. Он позволяет также менять каталог, искать файлы, открывать файлы и делать изменения.

Кнопка Start. Обеспечивает свободный доступ к инструментальным средствам, демонстрационным версиям, и документации.

Браузер справки (Help). MATLAB имеет обширную и прекрасно организованную документацию, состоящую из описания функций и серии электронных книг для более глубокого изучения методов, используемых в MATLAB. Справочный материал и электронные книги созданы в формате html, поэтому доступ к ним возможен как в среде MATLAB, так и независимо. Для поиска и изучения документации и демонстрационных версий для всех программ в среде MATLAB используется *Help-браузер* MATLAB. Он открывается из меню **Help**, или нажатием кнопки справки «?» в инструментальной панели, или из командной строки командой `helpbrowser`.

Браузер справки состоит из двух панелей, Навигатор (Help Navigator), который используется для поиска, и правая панель, где отображается выбранная информация.

Навигатор справки содержит оглавление документации в раскрывающихся списках. После выбора темы появляется следующий раскрывающийся список с содержанием документации по данной теме. При этом следует обратить внимание, что значок двух синих страниц обозначает руководство пользователя по данной теме, а значок двух желтых страниц обозначает справку по функциям. Например, на рис. 1.1.3, Using the Symbolic Math Toolbox – это руководство пользователя (электронная книга) по пакету символьной математики, а Function Reference – справка по функциям пакета. Help-навигатор имеет следующие возможности:

- **Product filter** (Фильтр программ) – устанавливается для того, чтобы показывать документацию только для заданных продуктов системы MATLAB;
- **Contents** (Содержание) – отражает заголовки и оглавления документации;
- **Index** (Индекс) – справка по ключевым словам в алфавитном порядке;
- **Demos** (Демонстрационные примеры) – представляет и выполняет демонстрации многих продуктов MATLAB;
- **Search** (Поиск) – поиск по определенному слову или фразе в документации;
- **Favorites** (Избранное) – показывает список документов, которые предварительно определены как фавориты.

В правой панели отображается содержание найденной документации. Данное окно также имеет ряд дополнительных возможностей поиска и печати (гиперссылки на близкие темы, переход на следующую страницу, в самой нижней строке отображается путь и название файла справки). Отметим, что при выборе темы в

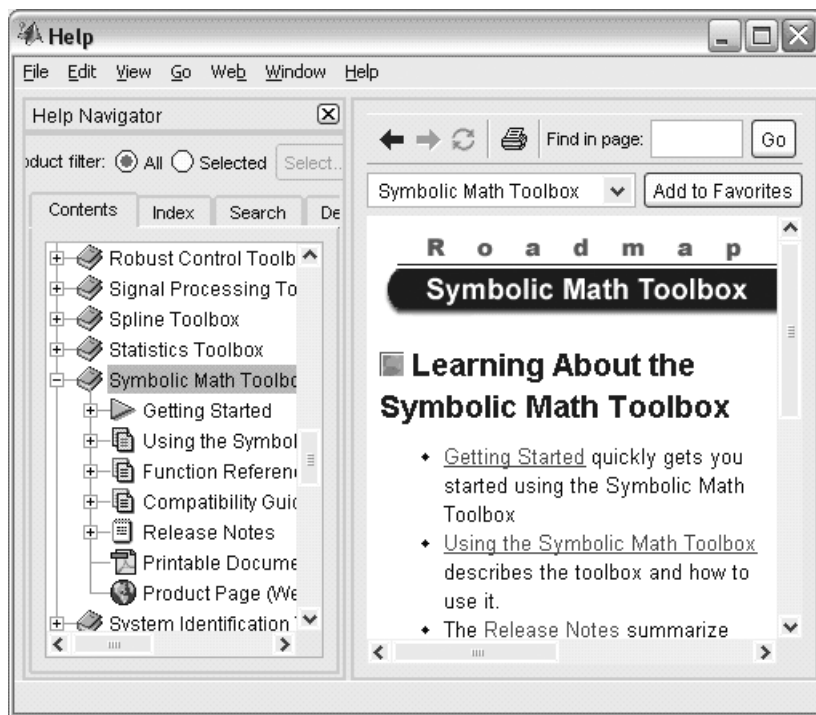


Рис. 1.1.3. Help-браузер MATLAB

Help-навигаторе, в правой панели также отражается содержание данной темы с комментариями.

Возможен прямой доступ к документации вне среды MATLAB. Для этого достаточно открыть каталог справки `C:\MATLAB\help\` и в нем открыть файл `begin_here.html`, либо в каталоге `C:\MATLAB\help\techdoc\` открыть `matlab_product_page.html`. Для справки по пакету расширения нужно открыть соответствующий каталог, например, `C:\MATLAB\help\toolbox\symbolic\` и в нем файл `symbolic.html` или `symbolic_product_page.html`.

Доступ к справке имеется и из командной строки MATLAB. Это наиболее быстрый способ выяснить синтаксис и особенности применения m-функции. Для этого используются команды `help <имя m-функции>` в командной строке. Соответствующая информация появляется непосредственно в командном окне. Например, команда `help magic` выведет в командное окно следующую информацию

```
help magic
```

```
MAGIC Magic square.
```

```
MAGIC(N) is an N-by-N matrix constructed from the integers
1 through N^2 with equal row, column, and diagonal sums.
```

```
Produces valid magic squares for all N > 0 except N = 2.
```

Все функции системы MATLAB организованы в логические группы, структура каталогов основана на этой организации. Например, все функции линейной алгебры находятся в каталоге `matfun`. Можно распечатать все функции этого каталога с короткими пояснениями, если использовать команду

```
help matfun
```

Команда `help` сама по себе выводит на экран список каталогов. Команда `lookfor` позволяет выполнить поиск `m`-функции по ключевому слову, при этом анализируется первая строка комментария, и она же выводится на экран, если в ней встретилось ключевое слово. Например, команда `lookfor inverse` выводит на экран большой список, начало которого представлено ниже

```
lookfor inverse
INVHILB Inverse Hilbert matrix.
IPERMUTE Inverse permute array dimensions.
ACOS    Inverse cosine.
ACOSH   Inverse hyperbolic cosine.
ACOT    Inverse cotangent.
ACOTH   Inverse hyperbolic cotangent.
ACSC    Inverse cosecant.
ACSCH   Inverse hyperbolic cosecant.
ASEC    Inverse secant.
ASECH   Inverse hyperbolic secant.
ASIN    Inverse sine. lookfor inverse
```

Дополнительные команды справочной системы. Укажем еще ряд команд, при помощи которых можно получить справочные данные в командном режиме:

- `computer` – выводит сообщение о типе компьютера, на котором установлена текущая версия MATLAB;
- `info` – выводит информацию о фирме Math Works с указанием адресов электронной почты;
- `ver` – выводит информацию о версиях установленной системы MATLAB и ее пакетах расширений;
- `version` – выводит краткую информацию об установленной версии MATLAB;
- `what` – выводит имена файлов текущего каталога;
- `what name` – выводит имена файлов каталога, заданного именем `name`;
- `whatsnew name` – выводит на экран содержимое файлов `readme` заданного именем `name` класса для знакомства с последними изменениями в системе и в пакетах прикладных программ;
- `which name` – выводит путь доступа к функции с данным именем;
- `help demos` – выводит весь список примеров в справочной системе MATLAB;
- `bench` – тест на быстродействие компьютера. Результаты теста представляются в виде таблицы и диаграммы сравнения с другими типами компьютеров.

Редактор/отладчик. Он используется для создания и отладки `m`-файлов, т.е. программ, написанных на языке MATLAB. *Редактор/отладчик* представляет со-

бой текстовый редактор с возможностями запуска и отладки программы m-файла. Редактор/отладчик вызывается либо из меню **File** \Rightarrow **New**, либо по кнопкам «новый документ», «открыть» в инструментальной панели MATLAB, либо двойным щелчком по соответствующему m-файлу. Если в редакторе/отладчике открыт m-файл из текущего каталога, он может быть запущен в MATLAB прямо из редактора по кнопке «Run». Эта кнопка записывает файл в текущий каталог и затем запускает его. Как видно на рис. 1.1.4, знак (%) означает начало текста комментария. Этот знак действует только в пределах одной строки. Возможности редактора/отладчика достаточно большие, однако с ними лучше познакомиться практически, записывая и запуская m-файлы.

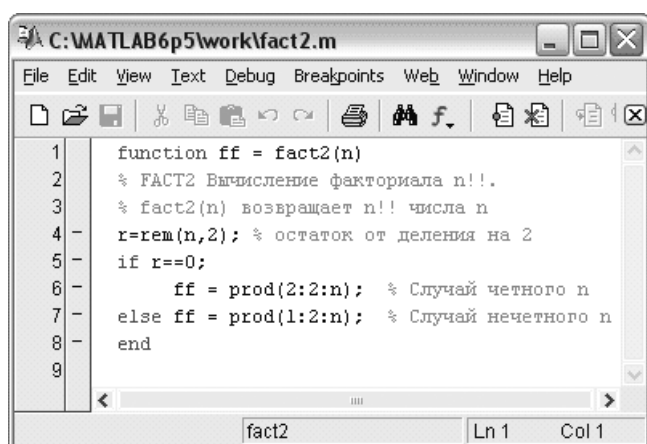


Рис. 1.1.4. Редактор/отладчик

Профилировщик (Profiler). Он представляет собой графический интерфейс пользователя, помогающий улучшать работу m-файла. Для открытия профилировщика нужно исполнить команду `profile viewer` в командной строке MATLAB.

1.1.3. Константы и системные переменные MATLAB

Это следующие специальные числовые и системные константы:

- **i** или **j** – мнимая единица (корень квадратный из -1);
- **pi** – число $\pi = 3.141592653589793\text{e}+000$;
- **eps** – погрешность операций над числами с плавающей точкой, это расстояние от единицы до ближайшего большего числа, $\text{eps} = 2.220446049250313\text{e}-016$, или 2^{-52} ;

- **realmin** – наименьшее число с плавающей точкой, $\text{realmin} = 2.225073858507202\text{e-}308$, или 2^{-1022} ;
- **realmax** – наибольшее число с плавающей точкой, $\text{realmax} = 1.797693134862316\text{e}+308$, или 2^{1023} ;
- **inf** – значение машинной бесконечности;
- **ans** – переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея;
- **NaN** – неопределенность, нечисловое значение (Not-a-Number), например $0/0$.

1.1.4. Типы данных MATLAB

В MATLAB существует 15 основных типов данных (или классов). Каждый из этих типов данных находится в форме массива, вообще говоря, многомерного. Все основные типы данных показаны на рисунке 1.1.5. Дополнительные типы данных *user classes* и *java classes* могут быть определены пользователем как подклассы структур, или созданы из классов Java.

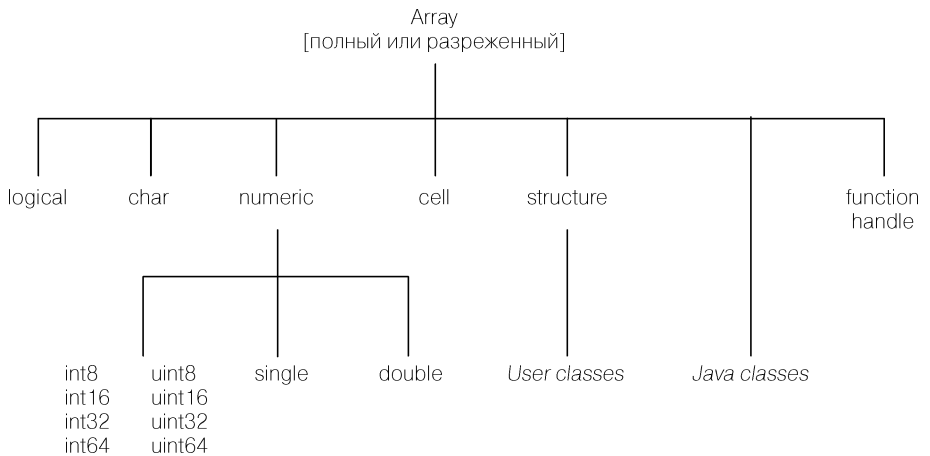


Рис. 1.1.5. Типы данных MATLAB

Типы переменных в MATLAB заранее не декларируются. Тип переменной *numeric* или *array* в MATLAB не задается. Эти типы служат только для того, чтобы сгруппировать переменные, которые имеют общие атрибуты. Матрицы типа *double* и *logical* могут быть как полными, так и разреженными. Разреженная форма матрицы используется в том случае, когда матрица имеет небольшое количество отличных от нуля элементов. Разреженная матрица, требует для хранения меньше памяти, поскольку можно хранить только отличные от нуля элементы и их индексы. Операции с разреженными матрицами требуют специальных методов.

Тип данных `logical`. *Логический массив.* Он представляет значения логических переменных `true` или `false`, используя логическую единицу (1, истина) и логический ноль (0, ложь), соответственно. Логические матрицы могут быть разреженными. MATLAB возвращает логические значения из отношений (например, `>`, `~ =`) и логических операций и функций. Например, следующая команда

```
x = magic(4) > 10
```

создает логический массив 4-на-4 из единиц и нулей, в соответствии с тем, больше элемент матрицы `magic(4)` числа 10, или нет.

Тип данных `char`. Массив символов (каждый символ 2 байта). Такой массив называют также строкой. Символьная строка – это просто массив 1-на-*n* символов. Можно создать массив *m*-на-*n* строк, если каждая строка в массиве имеет одну и ту же длину. Для создания массива строк неравной длины, используется массив ячеек. Массив символов может быть задан в командной строке в одинарных кавычках, например,

```
x = 'Привет!'
```

Числовые типы данных `numeric`. Это массивы чисел с плавающей запятой одинарной точности (`single`), массивы чисел с плавающей запятой двойной точности (`double`), массивы целых чисел со знаком (`int8`, ... , `int64`) и без знака (`uint8`, ... , `uint64`), которые имеют длину в 8, 16, 32, и 64 бита. Для числовых типов данных в MATLAB отметим следующее:

- все вычисления MATLAB делаются с двойной точностью;
- чтобы выполнять математические операции на целочисленных или массивах одинарной точности, нужно преобразовать их к двойной точности, используя функцию `double`.

Тип данных `int*`. Он содержит следующие типы:

- **`int8`** – массив 8-разрядных целых чисел со знаком (1 байт на одно число). Он позволяет хранить целые числа в диапазоне от -128 до 127;
- **`int16`** – массив 16-разрядных целых чисел со знаком (2 байта на одно число). Он позволяет хранить целые числа в диапазоне от -32 768 до 32 767;
- **`int32`** – массив 32-разрядных целых чисел со знаком (4 байта на одно число). Он позволяет хранить целые числа в диапазоне от -2 147 483 648 до 2 147 483 647;
- **`int64`** – массив 64-разрядных целых чисел со знаком (8 байт на одно число). Он позволяет хранить целые числа в диапазоне от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

Тип данных `uint*`. Он содержит следующие типы:

- **`uint8`** – массив 8-разрядных целых чисел без знака (1 байт на одно число). Он позволяет хранить целые числа в диапазоне от 0 до 255;
- **`uint16`** – массив 16-разрядных целых чисел без знака (2 байта на одно число). Он позволяет хранить целые числа в диапазоне от 0 до 65 535;
- **`uint32`** – массив 32-разрядных целых чисел без знака (4 байта на одно число). Он позволяет хранить целые числа в диапазоне от 0 до 4 294 967 295;

- **uint64** – массив 64-разрядных целых чисел без знака (8 байта на одно число). Он позволяет хранить целые числа в диапазоне от 0 до 18 446 744 073 709 551 615.

Большинство операций, которые управляют массивами, не изменяя их элементы, определены для целочисленных типов. Однако математические операции не определены для объектов `int*` из-за неопределенности значений, которые выходят за пределы диапазона.

Функция для преобразования числового массива в целый тип со знаком имеет вид `ix = int(x)`. Переменная `x` может быть любым числовым объектом, например `double`. Если значение `x` выше или ниже диапазона для класса, то результат будет равен соответствующему конечному значению диапазона. Пример использования,

```
y = uint8(magic(3)) % массив целых чисел типа uint8
```

Тип данных single. Массив чисел с плавающей запятой одинарной точности (8 знаков). Класс *single* предназначен для более экономного хранения данных. Величины одинарной точности требуют меньшего количества памяти (4 байта на одно число) для хранения, чем величины с двойной точностью (8 байт на одно число), но имеют меньше точности и меньший диапазон. Большинство операций, которые управляют массивами, не изменяя их элементы, определено для `single`. Математические операции не определены для объектов `single`. Функция преобразования в тип с одинарной точностью имеет вид `B = single(A)`.

Тип данных double. Массив чисел с плавающей запятой двойной точности (16 знаков). Это самый общий тип переменной MATLAB. Определены все операции.

Массив ячеек, cell array. В ячейках массива можно сохранить массивы различных типов и/или размеров. Обращение к данным в массиве ячеек использует матричную индексацию, как и в других MATLAB матрицах и массивах. Массивы ячеек рассмотрим в дальнейшем более подробно.

Тип данных структура, structure. Он подобен массиву ячеек и также позволяет сохранять несходные виды данных. Но в этом случае данные хранятся в полях, а не в ячейках. Это дает возможность присвоить названия группам данных, которые сохраняются в структуре. Обращение к данным в структуре использует имена полей. Массивы структур рассмотрим в дальнейшем более подробно.

Дескриптор функции, function handle. *Описатель (дескриптор) функции* содержит (в виде структуры) всю информацию о функции, которая используется в ссылках на функцию и которая необходима для определения местонахождения, дальнейшего выполнения, или оценивания (`evaluate`). Как правило, дескриптор функции передается в списке параметров к другим функциям. Это используется вместе с `feval` для вычисления функции, которая соответствует дескриптору. Пример получения дескриптора функции `sin`

```
z=functions(@sin) %получаем массив 1-на-1 типа структура
z =
    function: 'sin'
    type: 'simple'
    file: 'MATLAB built-in function'
```

Типы данных MATLAB реализованы в виде классов. Можно также создавать собственные классы (user classes) MATLAB как подклассы структуры. MATLAB

обеспечивает интерфейс с языком программирования Java, который дает возможность создавать объекты из классов Java и вызывать методы Java на этих объектах. Класс Java есть тип данных MATLAB. Есть встроенные и сторонние классы, которые являются доступными через интерфейс MATLAB. Каждому типу данных можно соотнести свои функции и операторы обработки, или другими словами, методы.

1.2. Основы работы с MATLAB®

Здесь мы рассмотрим те вопросы, которые возникают в начале работы с MATLAB: как запустить систему и начать работу, как задать массив и выполнить операции над массивами, как загрузить данные и сохранить результаты работы.

1.2.1. Запуск MATLAB и начало работы

После запуска MATLAB на экране появляется рабочий стол системы MATLAB (см. рис. 1.1.1). Система готова к проведению вычислений в командном режиме. Сеанс работы с MATLAB называется сессией (session). Окно справа называется командным. Именно в нем происходит задание команд и выводятся результаты вычислений. Команды можно ввести в строку приглашения, которая отмечены символом «>>>» и положением курсора. В этой строке можно ввести арифметическую операцию, функцию, или оператор присвоения, затем нажать клавишу исполнения Enter и результат появляется также в командном окне. При этом строка ввода будет самой нижней строкой окна, а текст выше – недоступен для редактирования.

Пример 1. Рассмотрим создание *магического* квадрата порядка 3 и присвоения его переменной *x*. Это числовая матрица порядка 3, обладающая тем свойством, что сумма элементов по строкам, по столбцам и по диагоналям одинакова. Для создания такой матрицы в MATLAB имеется функция *magic(n)*.

```
x=magic(3)
```

```
x =
```

8	1	6
3	5	7
4	9	2

Имя переменной (ее идентификатор) может содержать до 63-х символов. Имя любой переменной не должно совпадать с именами функций и процедур системы. Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания *_*. Недопустимо включать в имена переменных пробелы и специальные знаки, например *+, -, *, /* и т. д. MATLAB не допускает использование кириллицы в именах файлов и именах переменных.

Замечание 1. Если строка команд слишком длинная и не входит в видимую часть командного окна, ее можно перенести на следующую строку, используя оператор многоточия «...»:

```
x=magic(3)+magic(3)^2+magic(3)^3+...
+magic(3)^4
```

Замечание 2. Ранее исполненные команды можно ввести в командную строку клавишами «стрелка вверх» и «стрелка вниз». Они используются для их исправления, дублирования или дополнения ранее введенных команд.

Все операции над числами MATLAB выполняет в формате двойной точности `double`, т.е. 16 знаков для числа. Однако в командном окне числа могут отображаться в различных видах. Для выбора *формата* представления числа используется функция `format`. Отметим еще раз, что функция `format` меняет только представление чисел на экране, но не меняет вычисления MATLAB с двойной точностью. Команда `format type` изменяет формат на указанный в строке `type`. Укажем основные допустимые значения для `type`.

- **short** – короткий формат (по умолчанию). Целая часть (по модулю) менее 1000, после запятой содержит 4 знака, например $x = 112.1416$. Если модуль целой части больше 1000, то применяется `short e`. Для целого числа отображается 9 знаков;
- **short e** – короткий формат числа с плавающей запятой с 5 знаками. Например, $x = 1.1214e+002$. Для целого числа отображается 9 знаков;
- **long** – длинный формат, 16 знаков. Целая часть (по модулю) менее 100, остальные 14, или 15 знаков – после запятой, например $e^\pi = 23.14069263277927$. Если модуль целой части больше 100, то применяется `long e`. Для целого числа отображается 9 знаков;
- **long e** – длинный формат числа с плавающей запятой с 16 знаками. Целая часть (по модулю) менее 10, остальные 15 знаков – после запятой, например $e^\pi = 2.314069263277927e+001$. Для целого числа отображается 9 знаков;
- **rat** – представление числа в виде рациональной дроби, например, $\pi = 355/113$;
- **hex** – шестнадцатеричное представление числа с двойной точностью, например, $\pi = 400921fb54442d18$.

Пример 2. Изменим короткий формат по умолчанию на длинный и вычислим число π ,

```
format long
pi^(exp(1))
ans =
22.45915771836105
```

Для обработки чисел используются следующие функции.

- `round(x)` – округление до ближайшего целого;
- `fix(x)` – обнуление всех знаков после запятой;
- `floor(x)` – целая часть числа, $[x]$, наибольшее целое, не превосходящее данное x ;
- `ceil(x)` – наименьшее целое, большее или равное x ;
- `sign(x)` – знак числа, принимает значения -1, 0, +1;
- `rem(x,y)` – остаток от деления, $x - n \cdot y$, где $n = \text{fix}(x./y)$.

Комплексные числа. Для обозначения мнимой единицы комплексных чисел используются символы i и j . Комплексное число $z = a + bi$ можно задать в командной строке одним из следующих способов:

$$z = a+bi = a + i b = a + i * b = a + b * i = a + b j = \dots$$

Для работы с комплексными числами используются следующие функции:

- $\text{abs}(z)$ – модуль комплексного числа, $|z|$;
- $\text{conj}(z)$ – комплексно сопряженное число, $a-bi$;
- $\text{imag}(z)$ – мнимая часть числа;
- $\text{real}(z)$ – вещественная часть числа;
- $\text{angle}(z)$ – аргумент числа;
- $\text{isreal}(z)$ – дает логическую 1, если число действительное и логический 0 – в случае комплексного.

1.2.2. Задание массивов

Как известно, все переменные MATLAB являются массивами. Числовые массивы по умолчанию имеют тип `double`. В частности, одно число считается массивом типа `double` размерности 1-на-1. Например, если задать переменную $x=1.5$, то ее можно вызвать просто как x , либо как одномерный массив, $x(1)$, либо как двумерный, $x(1,1)$.

Положение элементов массивов определяется индексами. *Индексация* в MATLAB начинается с единицы – это, так называемая 1-базовая индексация. Кроме того, даже для многомерных массивов MATLAB поддерживает еще их одномерную индексацию, сохраняя данные в постолбцовом порядке, эта традиция происходит от ФОРТРАН-а.

Задание одномерных массивов

Вектор-строку можно задать непосредственно в командной строке, используя оператор *объединения* `[]`. Например, команда

```
x=[1,2,3,4]
```

создает вектор $x=(1,2,3,4)$. Элементы вектора в выражении $x=[1,2,3,4]$ можно также отделять пробелами: $x=[1\ 2\ 3\ 4]$. Выражение $y=[x,5]$ добавляет к вектору x еще один элемент 5.

Элементы массива можно задать (изменить) указывая прямо значение с соответствующим одномерным индексом. Например, команда $x(8)=-1$ создает вектор x длины 8, на восьмое место ставится число -1, остальные недостающие элементы являются нулями

```
x(8)=-1
x =
     1     2     3     4     0     0     0    -1
```

Одномерный массив можно также задать как диапазон значений. Например, команда

```
x=1:0.001:5;
```

создает массив чисел от 1 до 5 с шагом 0.001. Число элементов этого массива равно 4001, поскольку граничные значения включаются. Длину массива можно найти командой

```
length(x)
ans =
    4001
```

Замечание 3. Точка с запятой в конце команды предотвращает вывод результатов вычислений в командное окно. Например, если бы в команде $x=1:0.001:5$ не поставить в конце знак «;», то в рабочее поле были бы выведены все 4001 значений массива.

Замечание 4. Индексация элементов массива MATLAB начинается с единицы, а не с нуля, как в C++. Индексы элементов массива указываются в круглых скобках.

К любому элементу массива можно обратиться, указав его индекс. Например, команда

```
z=x(124)
z = 1.1230
```

присваивает переменной z значение $x(124)$ массива x .

Приведем несколько команд для создания *одномерных* массивов.

- `linspace(a,b)` – массив из 100 равноотстоящих чисел между a и b , с включением конечных значений a и b ;
- `linspace(a,b,n)` – массив из n равноотстоящих чисел на отрезке $[a, b]$ с включением конечных значений a и b ;
- `logspace(a,b,n)` – массив из n чисел на отрезке $[10^a, 10^b]$, равноотстоящих в логарифмическом масштабе с включением конечных значений 10^a и 10^b .

Задание двумерных массивов

Двумерный массив (матрицу) можно задать непосредственно в командной строке, используя оператор объединения `[]`. Например, команда

```
x=[1,2,3,4;5,6,7,8]
```

создает матрицу из двух строк (1,2,3,4) и (5,6,7,8). Правила для создания такого массива следующие: сначала записываются элементы первой строки, окончание строки и переход на следующую строку отмечается точкой с запятой (;). Элементы строк можно отделять либо запятыми, либо пробелами.

Элементы массива можно задать (изменить) указывая прямо значение с соответствующим двумерным индексом. Например, команда

```
x(3,1)=-1
```

добавляет третью строку, в которой на первом месте стоит число -1, а остальные недостающие элементы являются нулями. Команда `z(5,10)=1.5` создает новый двумерный массив размера 5-на-10, причем `z(5,10)=1.5`, а остальные элементы являются нулями.

В случае если имеются несколько строк (или столбцов) одинаковой длины, их можно объединить в матрицу оператором объединения `[]`. Например, пусть даны 3 строки u , v , w одной длины n , тогда команды

```
X=[u;v;w]
```

```
Y=[u',v',w']
```

создают, соответственно, матрицу 3-на- n из трех строк u , v , w и матрицу n -на-3 из трех столбцов u' , v' , w' (транспонированные строки).

Оператор `[]` объединяет не только строки и столбцы, но и матрицы при естественном требовании совместимости по количеству строк или столбцов. Для команды горизонтального объединения

```
X=[A,B]
```

требуется равенство строк матриц A и B , а для команды вертикального объединения

```
Y=[C;D]
```

требуется равенство столбцов матриц C и D .

Замечание 5. Оператор `[]` горизонтального объединения имеет функциональную версию в виде функции *horzcat*. Например, команды $V=[A,B]$ и $V=\text{horzcat}(A,B)$ дают одинаковый результат. Аналогично, оператор `[:]` вертикального объединения может употребляться в виде функции *vertcat*. Например, команды $V=[A;B]$ и $V=\text{vertcat}(A,B)$ дают одинаковый результат.

Замечание 6. Пустой массив задается символом `[]`. Он используется также для удаления элементов и массивов.

Замечание 7. В двумерной индексации $z(n,m)$ первый индекс – это номер строки, а второй индекс – это номер столбца массива z . Для матриц используется также и одномерная индексация, когда элементы нумеруются по столбцам в порядке их следования (в отличие от построчной индексации в других языках программирования).

Индексация позволяет обратиться к отдельным элементам матрицы. Если мы хотим выделить целую строку или столбец, не нужно создавать цикл. Для этого имеется оператор *двоеточия* `(:)`. Например, обратиться ко всей p -ой строке матрицы A можно следующим образом

```
y=A(p,:);
```

а для обращения ко всему q -му столбцу матрицы A достаточно записать

```
z=A(:,q);
```

Команда $B=A(:, :)$ обращается ко всем элементам матрицы, т.е. создает копию B матрицы A .

Поскольку мы можем обратиться к строкам и к столбцам матрицы, то можно их удалять, или переставлять. Удаление производится присваиванием выбранной строке или столбцу пустого множества, например

```
A(p,:)=[];
```

Пусть A – матрица порядка n -на- m и $(s(1), \dots, s(n))$ – перестановка чисел индексов $(1, \dots, n)$ строк. Тогда следующая команда задает перестановку строк матрицы.

```
B=[A(s(1),:); A(s(2),:); ... A(s(n),:)];
```

Перестановка столбцов матрицы производится аналогично.

Пустые массивы. Многомерный массив, у которого хотя бы одна размерность нулевая, называется пустым. Например, массивы с размерами 1-на-0 или 10-на-0-на-20

определяются как пустые. Квадратные скобки [] обозначают массив 0-на-0. Для проверки, является ли массив *A* пустым, применяется функция `isempty(A)`. Основное назначение пустых массивов состоит в том, чтобы любая операция, которая определена для массива (матрицы) размера *m*-на-*n* и которая дает результат в виде функции от *m* и *n*, давала бы правильный результат для случая, когда *m* или *n* равно нулю.

Элементарные матрицы. Приведем несколько команд для создания некоторых стандартных матриц.

- `zeros(n,m)` – матрица из нулей размера *n*-на-*m*;
- `ones(n,m)` – матрица из единиц размера *n*-на-*m*;
- `rand(n,m)` – матрица случайных чисел размера *n*-на-*m*;
- `eye(n)` – единичная матрица порядка *n*;
- `eye(n,m)` – матрица из единиц на главной диагонали размера *n*-на-*m*;
- `magic(n)` – магическая матрица порядка *n*.

Система MATLAB имеет также большую серию специальных матриц, таких как матрица Гильберта (создается функцией `hilb(n)`), матрица Адамара (функция `hadamard(n)`), матрица Уилкинсона (функция `wilkinson(n)`), матрица Ганкеля (функция `hankel(n)`),

1.2.3. Операции над массивами

Система MATLAB автоматически определяет размерность массива. При выполнении функций и арифметических операций требуется только, чтобы не возникало противоречий. А именно при сложении требуется, чтобы размерности массивов совпадали, для умножения матриц требуется, чтобы количество столбцов первой матрицы было равно количеству строк второй матрицы. Многие математические функции, аргумент которых традиционно считается скалярным, определены и на массивах. Например, если *A* – это массив *n*-на-*m*, то функция `sin(A)` вычислит синусы всех элементов матрицы *A*. Однако есть чисто матричные операции, такие как произведение матриц *A***B*. В этом случае нужно указывать дополнительно, является ли это умножение матричным, или поэлементным. Матричные операции обозначаются стандартно, а для того, чтобы отметить, что производится поэлементная операция, ставится точка перед соответствующей матричной операцией. Каждой матричной операции соответствует функция.

Таблица 1.2.1. Арифметические операции

Операция	Функция	Описание
<i>A</i> + <i>B</i>	<code>plus(A,B)</code>	Сложение массивов одинаковой размерности
<i>A</i> - <i>B</i>	<code>minus(A,B)</code>	Вычитание массивов одинаковой размерности
<i>A</i> * <i>B</i>	<code>mtimes(A,B)</code>	Матричное умножение
<i>A</i> / <i>B</i> (<i>A</i> \ <i>B</i>)	<code>mrdivide(A,B)</code> <code>mldivide(A,B)</code>	Матричное деление (правое и левое) <i>A</i> / <i>B</i> = <i>A</i> *inv(<i>B</i>), <i>A</i> \ <i>B</i> = inv(<i>A</i>)* <i>B</i> (! для более точного определения см. справку MATLAB по арифметическим операциям +, -, /, \)

Таблица 1.2.1. Арифметические операции (окончание)

Операция	Функция	Описание
A'	<code>ctranspose(A, B)</code>	Комплексное сопряжение и транспонирование
$A.*B$	<code>times(A, B)</code>	Перемножение элементов матрицы, $A.*B = (a_{ij} * b_{ij})$
$A./B$ ($A.\backslash B$)	<code>rdivide(A, B)</code> <code>ldivide(A, B)</code>	Деление элементов матрицы, $A./B = (a_{ij} / b_{ij})$, $A.\backslash B = (b_{ij} / a_{ij})$
$A.'$	<code>transpose(A)</code>	Транспонирование матрицы (без сопряжения)
$A.^B$	<code>power(A, B)</code>	Матрица, состоящая из элементов a_{ij}^b

Кроме того, нужно иметь в виду, что операции вида $2*A$ или $A+1$ означают умножение всех элементов матрицы A на число 2 и, соответственно, прибавление числа 1 ко всем элементам матрицы A . В системе MATLAB принято, что скаляр расширяется до размеров второго операнда и заданная операция применяется к каждому элементу.

Пример 3. Введем матрицу $A = [1, 2; 3, 4]$ и вычислим A^2 и $A.^2$.

```
A^2
ans =
     7     10
    15     22

A.^2
ans =
     1     4
     9    16
```

Как уже упоминалось, многие математические функции MATLAB являются векторизованными, т.е. вычисляют значения на каждом элементе массива. Однако есть функции, которые допускают матричный аргумент, например матричная экспонента

$$e^A = 1 + A + \frac{1}{2!}A^2 + \dots + \frac{1}{n!}A^n + \dots$$

Матрица A должна быть квадратной. Обращение к такой функции $F = \text{fun}(A)$ производится по правилу, $F = \text{funm}(A, @\text{fun})$. Например, матричный синус $\sin(A)$ может быть вычислен так:

```
f=funm(A,@sin)
```

Для матричной экспоненты, матричного логарифма и квадратного корня матрицы используются специальные функции:

```
expm(A) ,      logm(A) ,      sqrtm(A)
```

Отметим некоторые функции, относящиеся к матрицам.

Таблица 1.2.2. Функции матриц

Функция	Описание
<code>det(A)</code>	Определитель матрицы
<code>B = inv(A)</code>	Обратная матрица
<code>[n,m] = size(A)</code>	Размерность матрицы

Таблица 1.2.2. Функции матриц (окончание)

Функция	Описание
$S = \text{length}(A)$	Максимальный размер матрицы A , $s = \max(\text{size}(A))$
$\text{trace}(A)$	След матрицы, сумма диагональных элементов, матрица может быть не квадратной
$\text{sum}(A)$	Вектор, состоящий из сумм элементов столбцов
$\text{prod}(A)$	Вектор, состоящий из произведений элементов столбцов
$V = \text{diag}(A)$	Вектор-столбец элементов главной диагонали
$A = \text{diag}(V)$	Диагональная матрица с вектором V элементов главной диагонали
$U = \text{triu}(A)$	Верхняя треугольная часть матрицы
$U = \text{tril}(A)$	Нижняя треугольная часть матрицы
$B = \text{orth}(A)$	Столбцы матрицы B образуют ортонормированный базис пространства $\text{Im}(A)$, натянутого на столбцы матрицы A
$V = \text{null}(A)$	Столбцы матрицы V образуют ортонормированный базис пространства $\text{Ker}(A)$, нулевого пространства оператора A
$p = \text{poly}(A)$	Характеристический полином матрицы A
$J = \text{Jordan}(A)$	Жорданова форма A

Следующие функции требуют дополнительных комментариев.

Собственные числа матрицы. Применяются следующие команды

- $d = \text{eig}(A)$ вычисляет вектор *собственных* чисел матрицы A ;
- $[V, D] = \text{eig}(A)$ находит собственные векторы и собственные числа A . Столбцы матрицы V есть собственные векторы единичной нормы, а матрица D – диагональная, с собственными значениями на диагонали.

Сингулярные числа матрицы. Вычисляются для любых матриц, не только для квадратных. Матрица A может быть приведена к диагональному виду S при помощи двух унитарных матриц U и V следующим образом: $A = U * S * V^{-1}$. Матрица S диагональная с положительными значениями на диагонали, упорядоченными по убыванию. Эти диагональные элементы и называются *сингулярными числами* матрицы A . Матрицы U , S и V находятся командой *svd*,

$[U, S, V] = \text{svd}(A)$

Команда $s = \text{svd}(A)$ вычисляет вектор сингулярных чисел матрицы A . Сингулярные числа – это квадратные корни из собственных значений симметричной матрицы $A * A^t$. Термин «сингулярные числа» объясняется тем, что эти числа являются критическими значениями функции $f(x) = \|Ax\|$ на единичной сфере $\|x\| = 1$. Сингулярные числа отличаются от собственных чисел матрицы. Например, для матрицы A вида

$$\begin{pmatrix} 1 & 2 \\ 0 & 4 \end{pmatrix}$$

собственные числа есть 1 и 4. Матрица $B = A * A'$ имеет вид

$$B = \begin{pmatrix} 5 & 8 \\ 8 & 16 \end{pmatrix}$$

ее собственные числа равны 0.7918 и 20.2082, а сингулярные числа матрицы A есть 4.4954 и 0.8898.

Ранг матрицы (rank). Число ненулевых сингулярных значений называется рангом матрицы. Поскольку MATLAB не оперирует в своих вычислениях с целыми числами, то он не может определить точное равенство нулю, а только с некоторой точностью. Поэтому ранг матрицы в MATLAB зависит от точности вычислений. Функция

`R=rank(A)`

определяет ранг матрицы как число сингулярных значений, которые больше, чем $\max(\text{size}(A)) \cdot \text{norm}(A) \cdot \text{eps}$. Функция

`R=rank(A, tol)`

определяет ранг матрицы как число сингулярных значений, которые больше, чем `tol`.

Норма матрицы (norm). Она определяется как максимальное сингулярное значение $\text{norm}(A) = \max(\text{svd}(A))$. Это обычное определение нормы. Имеется несколько других определений нормы матрицы в виде $\text{norm}(A, p)$, где $p = 1, 2, \text{inf}, \text{'fro'}$:

- $\text{norm}(A, 1) = \max(\text{sum}(\text{abs}(A)))$ – максимальное значение сумм модулей элементов столбцов;
- $\text{norm}(A, 2) = \text{norm}(A) = \max(\text{svd}(A))$ – максимальное сингулярное значение;
- $\text{norm}(A, \text{inf}) = \max(\text{sum}(\text{abs}(A')))$ – максимальное значение сумм модулей элементов строк;
- $\text{norm}(A, \text{'fro'}) = \sqrt{\text{sum}(\text{diag}(A' * A))}$ – евклидова, или норма Фробениуса;

В случае вектора X норма определяется следующим образом:

- $\text{norm}(X, p) = \text{sum}(\text{abs}(X) . ^p) ^{1/p}$, для любого $1 \leq p < \infty$;
- $\text{norm}(X) = \text{norm}(X, 2)$;
- $\text{norm}(A, \text{inf}) = \max(\text{abs}(X))$ – максимальное значение из модулей элементов вектора;
- $\text{norm}(A, -\text{inf}) = \min(\text{abs}(X))$ – минимальное значение из модулей элементов вектора.

1.2.4. Решение систем линейных уравнений

Рассмотрим систему линейных уравнений из m уравнений с n неизвестными.

$$\sum_{i=1}^n a_{ij} x_i = b_j, \quad j=1, 2, \dots, m, \quad a_{ij}, b_i \in \mathbf{R} \text{ (C)}.$$

В матричном виде, $A * x = b$. Матрица A имеет размерность m -на- n .

1. Случай $m=n$. Матрица A квадратная. Если матрица невырожденная, то решение находится следующим образом

$$x = A \backslash b;$$

В случае вырожденной матрицы вместо обратной матрицы $\text{inv}(A)$ можно попробовать применить псевдообратную матрицу $\text{pinv}(A)$, $x = \text{pinv}(A) * b$. Эта матрица $\text{pinv}(A)$ обладает некоторыми свойствами обратной матрицы $\text{inv}(A)$ (см. справку по функции `pinv`).

2. Случай $m > n$. Переопределенная система, уравнений больше, чем переменных. Решения может не существовать. Тогда команда $x = A \backslash b$ ищет такое x , которое минимизирует $\|A^*x - b\|^2$. Это значение, вообще говоря, решением не будет.

Пример 4. Рассмотрим систему (очевидно несовместную) вида

$$\begin{cases} x_1 &= 1 \\ x_1 &= 0. \\ x_1 + x_2 &= 0 \end{cases}$$

Тогда команда $x = A \backslash b$ дает следующее решение

```
x =
    -0.5000
     0.5000
```

Проверка, $b = A^*x$, приводит к следующему результату

```
b =
     0.5000
     0.5000
     0.0000
```

3. Случай $m < n$. Недоопределенная система, переменных больше, чем уравнений. Тогда команда $x_0 = A \backslash b$ ищет такое частное решение x_0 , которое имеет не более m ненулевых компонент. Для нахождения общего решения нужно определить $V = \text{null}(A)$ ортонормированный базис пространства $\text{Ker}(A)$. Тогда общее решение записывается в виде $x = x_0 + V^*C$, где C есть вектор-столбец произвольных констант, $C' = (C_1, \dots, C_k)$.

В случае, когда ранг основной матрицы не равен рангу расширенной, выдается предупреждение о том, что с указанной точностью MATLAB определил недостаточность ранга основной матрицы

```
Warning: Rank deficient, rank = 2    tol =    2.1756e-015.
```

В этом случае найденное решение может быть неправильным.

Применение системы MATLAB к решению других задач, например, математического анализа, обработки данных, решения дифференциальных уравнений и к графике можно найти в Help MATLAB и в любом руководстве по MATLAB, см. [ККШ], [По], [ЧЖИ], [Кр], [Ma], [Ко], [Д], [ГЦ] и [Ан]. Рассмотрим кратко возможности символьных вычислений MATLAB.

Символьная математика пакета расширения *Symbolic Math*

Пакет Symbolic Math позволяет пользоваться символьными математическими операциями. Он включает вычислительное ядро системы Maple. Пакет Extended Symbolic Math Toolbox, входящий в Symbolic Math Toolbox, предоставляет дополнительные возможности программирования Maple и дает доступ к специализированным библиотекам Maple.

Команда `help symbolic` вызывает перечень команд и функций пакета. Список всех функций ядра Maple можно получить командой `mfunlist`. Для вызова справки по конкретной функции достаточно в командной строке выполнить `mhhelp <function>`.

Команда `funtool` вызывает графическое окно для выполнения основных операций над символьными функциями и для построения графиков функций.

Для работы с символьной математикой определяется специальный тип объектов `sym`. Конструктор объектов может быть вызван двумя способами, которые показаны на следующих примерах:

```
Expr = sym('2*x+3*y'); %задание символьного выражения (138 байт);
x = sym('x');          %задание символьной переменной x (126 байт);
pi = sym('pi');         %задание символьного числа pi (128 байт);
syms y z t;             %задание символьных переменных y, z, t.
```

В пределах пакета `Symbolic Math` над переменными типа `sym` можно выполнять гигантское количество операций и применять к ним множество функций символьной математики.

Алгебраические операции. Это различные операции с многочленами и матрицами: разложение на множители (`factor`), раскрытие выражений (`expand`), упрощение выражений (`simplify`), нахождение корней многочлена (`solve`), нахождение определителя (`det`), решение систем линейных уравнений (`solve`) и многие другие. Например, найти решение уравнения $x^2 + 2x - 10 = 0$ можно следующим образом:

```
solve(x^2+2*x-10,x)
ans =
[ -1+11^(1/2) ]
[ -1-11^(1/2) ]
```

Операции математического анализа. Это дифференцирование (`diff`), интегрирование (`int`), нахождение пределов (`lim`), суммирование и разложение в ряд (`symsum` и `taylor`), нахождение точных решений дифференциальных уравнений (`dsolve`) и многие другие. Например, найдем интеграл $\int \sqrt{2-x^2} dx$,

```
I=int(sqrt(2-x^2),x)
I = 1/2*x*(2-x^2)^(1/2)+asin(1/2*2^(1/2)*x)
```

Преобразование форматов чисел. Для преобразования числа или матрицы в символьную форму `sym` используется команда

```
S = sym(A, flag)
```

где параметр `A` является числом или матрицей, `flag` может быть `'f'`, `'r'`, `'e'` или `'d'`. По умолчанию берется `'r'`. Другие значения флага:

- `'f'` – для формата с плавающей запятой. Значения представляются в виде `'1.F'*2^(e)` или `'-1.F'*2^(e)` где `F` есть строка из 13 шестнадцатеричных разрядов и `e` – целое. Например,

```
>> S = sym(1/11,'f')
S =
'1.745d1745d1746'*2^(-4)
```

- `'r'` – для рациональной формы числа в виде p/q , $p\pi/q$, \sqrt{p} , 2^q , and 10^q с целыми p и q . Если невозможно представить значение с плавающей точкой в простом рациональном виде с точностью ошибки округления, то берется выражение вида $p*2^q$ с большими целыми числами p и q .

Например,

```
>> sym(1+sqrt(3), 'r')
ans =
6152031499462229*2^(-51)
```

- 'd' – для десятичной формы числа. Число цифр устанавливается пользователем командой `digits`. Например,

```
>> digits(20);
>> sym(pi, 'd')
ans =
3.1415926535897931160
```

Команда `double(S)` делает обратное преобразование в формат `double`.

Построение графиков. Имеется достаточно большой набор функций для построения различных видов графиков функций. График функции одной переменной может быть построен при помощи функции `ezplot`, для графика функции двух переменных может быть использована функция `ezsurf`.

Пример 5. График функции $y = \sin(x)/x$ на промежутке $[-10, 10]$ и график функции $z = x^2 - y^2$, на области $x \in [-10, 10]$ и $y \in [-5, 5]$:

```
syms x y
ezplot(sin(x)/x, [-10 10])
ezsurf(x^2-y^2, [-10 10 -5 5])
```

Интересной дополнительной возможностью являются вычисления с заданной произвольно точностью. Для этих целей пакет Symbolic Math предоставляет две функции **digits** и **vpa**. Первая функция `digits` устанавливает число значащих цифр, а вторая, `vpa`, осуществляет вычисления с заданной точностью.

Пример 6. Вычисление чисел e и π с точностью до 45 верных знаков,

```
digits(40)
vpa(exp(1))
ans =
2.718281828459045534884808148490265011787
vpa(pi)
ans =
3.141592653589793238462643383279502884197
```

1.2.5. М-файлы

В командной строке можно выполнить небольшое количество простых команд. Достаточно большой набор команд MATLAB правильнее оформить и записать в виде отдельного файла, так называемого М-файла. Для создания М-файла может быть использован любой текстовый редактор, поддерживающий формат ASCII. В MATLAB имеется свой редактор/отладчик для создания и отладки м-файлов, т.е. программ, написанных на языке MATLAB. Редактор/отладчик вызывается либо из меню **File** \Rightarrow **New**, либо по кнопкам «новый документ» или «открыть» в инструментальной панели MATLAB. М-файл, созданный в редакторе/отладчике записывается в текущий каталог и имеет расширение `.m`.

Существует два типа М-файлов: М-сценарии (m-файлы скрипты) и М-функции со следующими характеристиками.

М-сценарий. Представляет просто последовательность команд MATLAB без входных и выходных параметров. *Сценарий* оперирует с данными из рабочей области. Результаты выполнения М-сценария сохраняются в рабочей области после завершения сценария и могут быть использованы для дальнейших вычислений.

Пример 7. Вычисление спектра Фурье сигнала.

```
%Вычисление спектра Фурье сигнала и вывод на график
%Открытие файла Nes_4.txt из текущего каталога
v=fopen('Nes_4.txt','rt');
S=fscanf(v,'%g',[1 inf]); %Считывание данных из файла Nes_4.txt
L=length(S); %длина сигнала
F=fft(S'); %Преобразование Фурье строк сигнала
F1=F'; P=F1.*conj(F1)/L; % Вычисление спектра
plot(P); axis([0 length(P) min(P) max(P)]); % График
fclose(v); %заккрытие файла
```

После выполнения этого сценария в рабочей области остались доступными для дальнейших вычислений следующие массивы S, L, F, F1 и P. Для М-сценария полезно писать комментарии. Они открываются символом % и служат для разъяснения смысла выполняемых команд.

М-функции. Это новые функции MATLAB, которые расширяют возможности системы. *М-функции* используют входные и выходные аргументы. Имеют внутренние локальные переменные. Каждая М-функция имеет следующую структуру:

- строка определения функции. Она задает имя функции и количество входных и выходных аргументов, их локальные имена. Например
function y = function_name(u,v,w)
- первая строка комментария определяет назначение функции. Она выводится на экран с помощью команд lookfor или help < имя функции >;
- основной комментарий;
- тело функции – это программный код, который реализует вычисления и присваивает значения выходным аргументам.

Если выходных параметров больше, то они указываются в квадратных скобках после слова function, например,

```
function [x, y, z] = sphere(theta, phi, rho)
```

М-функция записывается в файл с тем же названием, что и функция и с расширением m.

Пример 8. Функция n!!. Напомним, что в случае четного $n=2k$, $n!!$ есть произведение четных чисел от 2 до $2k$, а в случае нечетного $n=2k-1$, $n!!$ есть произведение нечетных чисел от 1 до $2k-1$.

```
function ff = fact2(n)
% FACT2 Вычисление факториала n!!.
% fact2(n) возвращает n!! числа n
r=rem(n,2); % остаток от деления на 2
if r==0;
```

```

ff = prod(2:2:n); % Случай четного n
else ff = prod(1:2:n); % Случай нечетного n
end

```

После создания этого кода, он записывается в файл с названием функции и расширением `m`, т.е. как `fact2.m` в текущий каталог. Теперь функция может быть вызвана из командной строки MATLAB,

```

fact2(6)
ans =
    48

```

Справка по этой функции вызывается так:

```

>> help fact2
FACT2 Вычисление факториала n!!.
Fact2(n) возвращает n!! числа n

```

Команда `what` выводит на экран имена `m`-файлов текущего каталога, среди которых находится и наша функция `fact2.m`. Команда `type fact2` выводит на экран полный текст `m`-файла `fact2.m`.

1.2.6. Чтение и запись текстовых файлов

Система MATLAB имеет ряд команд для работы с файлами вида `*.txt`, `*.html`, `*.m` и `*.mat`.

Команда *fopen*. Она дает доступ к файлу типа `*.txt`, `*.html`, `*.m` и `*.mat`. Формат команды

```
Fid=fopen('имя файла') ['мода'].
```

Переменная `Fid` называется файловым идентификатором. Она может иметь любое имя, разное для разных файлов. Переменная `Fid` принимает значение 1, если доступ открыт и значение -1, если доступ к файлу невозможен. Команда `fopen` применяется как для уже существующих файлов, так и для файлов, которые будут записаны. Мода может быть следующей.

Таблица 1.2.3. Мода открытия файла

Мода	Описание
'rt'	Открытие файла для чтения (по умолчанию)
'wt'	Открытие файла, или создание нового для записи. Если файл существует, то он будет удален без предупреждения, вместо него создается новый пустой файл с тем же именем
'at'	Открытие файла, или создание нового для записи. Если файл существует, то добавление данных в конец файла
'rt+'	Открытие файла для чтения и записи. Если файл существует, то новые данные будут записаны сначала, но место старых данных
'wt+'	Открытие файла для чтения и записи. Если файл существует, то он будет удален без предупреждения, вместо него создается новый пустой файл с тем же именем
'at+'	Открытие файла, или создание нового для чтения и записи, добавление данных в конец файла

Файл `filename` должен быть либо в текущем каталоге, либо на путях MATLAB, либо должен быть указан полный путь. По умолчанию новый файл записывается в текущий каталог. Команда `foren` открывает и бинарные файлы `*.mat`, в этом случае мода не содержит буквы `'t'`.

Команда *fclose*. Закрывает доступ к открытому ранее файлу, `fclose(fid)` – закрытие файла с идентификатором `fid`, `fclose(all)`) – закрытие всех открытых файлов.

Команда *fscanf*. Чтение форматированных данных из файла, к которому открыт доступ командой `foren`. Формат команды

```
A = fscanf(fid, 'format', size)
```

Параметр `size` определяет размерность массива в MATLAB, который будет создан при чтении данных из файла с идентификатором `fid`. Параметр `size` может иметь вид:

- `n` – чтение `n` элементов в столбец;
- `inf` – чтение элементов в столбец до конца;
- `[n m]` – чтение в матрицу размера `n`-на-`m`.

Отметим, что команда `fscanf` читает данные из файла по строкам, а записывает их в массив MATLAB по столбцам так, как указано в `size`. MATLAB читает данные из указанного файла в соответствии с заданным форматом. Параметр `format` может быть следующим:

Таблица 1.2.4. Формат чтения

Формат	Описание
<code>%c</code>	Последовательность символов
<code>%d</code>	Десятичные числа
<code>%e, %f, %g</code>	Числа с плавающей запятой (экспоненциальный, с фиксированной запятой и компактный вид)
<code>%i</code>	Целое число со знаком
<code>%o</code>	Восьмеричное целое число со знаком
<code>%s</code>	Ряд символов без пробелов
<code>%u</code>	Десятичное целое число со знаком
<code>%x</code>	Шестнадцатеричное целое число со знаком

Команда *fprint*. Запись форматированных данных в файл, к которому открыт доступ командой `foren`. Формат команды

```
fprintf(fid, 'format', A)
```

Здесь `fid` есть идентификатор открытого ранее файла. Имя файла указано при его открытии. `A` – массив, который будет записан в файл. Отметим, что команда `fprint` читает данные из массива MATLAB по столбцам, а пишет их в файл по строкам.

MATLAB пишет данные из указанного массива в файл в соответствии с заданным форматом. Строка формата указывает к какому виду следует преобразовать данные для записи. Строка формата записи начинается с символа `(%)` и содержит следующие необходимые и дополнительные и элементы:

- флаги (дополнительно);
- поля ширины и точности (дополнительно);
- символ преобразования (необходим).

Например, в записи «%-12.5e» знак «минус» есть флаг, число 12 определяет ширину поля (общее количество цифр), число 5 – это количество знаков после запятой и, наконец, буква «e» определяет, к какому типу будут преобразованы данные для записи.

Кроме того, применяются символы, которые управляют процессом вывода:

Таблица 1.2.5. Формат вывода данных

Формат	Описание
\n	Переход на новую строку
\t	Горизонтальная табуляция
\b	Возврат назад на один символ
	Пробелы в строке формата записываются как пробелы

Возможные флаги указаны в табл. 1.2.6.

Таблица 1.2.6. Флаги команды *fprint*

Символ	Описание	Пример
Знак минус (-)	Левое выравнивание преобразованных параметров	%-5.2d
Знак плюс (+)	Всегда печатать символ знака (+ или-)	%+5.2d
Нуль (0)	Замещение нулями вместо пробелов	%05.2d

Символ преобразования может быть следующим:

Таблица 1.2.7. Символы преобразования команды *fprint*

Формат	Описание
%c	Отдельный символ
%d	Десятичное представление чисел (со знаком)
%e	Экспоненциальное представление чисел (как в 3.1415e+00)
%f	С фиксированной точкой
%g	Компактный вид, без лишних нулей
%i	Десятичное представление чисел (со знаком)
%o	Восьмеричное (без знака)
%s	Строка символов
%u	Десятичное представление чисел (без знака)
%x	Шестнадцатеричное представление (с использованием символов нижнего регистра a-f)
%X	Шестнадцатеричное представление (символы верхнего регистра A-F)

Пример 9. Открытие файла *dat.txt* для записи и запись в него данных из массива А. Данные из массива А читаются по столбцам, а запись ведется числами из 6 цифр с фиксированной запятой в 5 столбцов с горизонтальной табуляцией.

```
fid=fopen('dat.txt','wt');
fprintf(fid,'%6.4f\t%6.4f\t%6.4f\t%6.4f\t%6.4f\n',A);
fclose(fid);
```

Если команда `fprintf` используется без идентификатора файла (вместо него – цифра 1), то вывод идет на дисплей.

Пример 10. Следующая команда

```
B = [8.8 7.7; 8800 7700]
fprintf(1,'X is %6.2f meters or %8.3f mm\n',9.9,9900,B)
```

выводит на дисплей строки:

```
X is 9.90 meters or 9900.000 mm
X is 8.80 meters or 8800.000 mm
X is 7.70 meters or 7700.000 mm
```

1.2.7. Операции с рабочей областью и текстом сессии

При работе с MATLAB могут быть получены интересные данные, которые желательно сохранить для следующей сессии. Кроме того, имеет смысл оптимизации памяти для ускорения работы. В MATLAB имеются средства для решения данных задач.

Дефрагментация. По мере задания одних переменных и стирания других рабочая область перестает быть непрерывной и начинает занимать много места. Это может привести к ухудшению работы системы или даже к нехватке оперативной памяти. Подобная ситуация возможна при работе с достаточно большими массивами данных. Во избежание непроизводительных потерь памяти при работе с большими массивами данных следует делать дефрагментацию рабочей области командой `pack`. Эта команда переписывает все переменные рабочей области на жесткий диск, очищает рабочую область и затем заново «непрерывно» считывает все переменные в рабочую область.

Сохранение рабочей области сессии. Система MATLAB позволяет сохранять значения переменных рабочей области в виде бинарных файлов с расширением `.mat`. Для этого служит команда `save`, которая может использоваться в следующем виде

```
save [имя файла] [переменные] [опции]
```

Она применяется в следующих формах:

- `save filename` – записывается рабочая область всех переменных в файле бинарного формата с именем `filename.mat`;
- `save filename X` – записывает только значение переменной `X`;
- `save filename XYZ -option` – записывает значения переменных `X`, `Y` и `Z`.

Ключи `-option`, уточняющие формат записи файлов, могут быть следующие.

Таблица 1.2.8. Опции команды `save`

Ключи	Результат
<code>-append</code>	Добавление в конец существующего MAT-файла
<code>-ascii</code>	ASCII-формат единичной точности (8 цифр, построчное сохранение)

Таблица 1.2.8. Опции команды *save* (окончание)

Ключи	Результат
-ascii -double	ASCII-формат двойной точности (16 цифр)
-ascii -tabs	Формат данных, разделенных табуляцией.
-ascii -double -tabs	Формат данных, разделенных табуляцией
-mat	Двоичный MAT-формат (используется по умолчанию)

Команда сохранения может применяться в виде функции, например,
`save('d:\myfile.txt','X','Y','-ASCII')`

Для загрузки рабочей области ранее проведенной сессии (если она была сохранена) можно использовать команду *load*, с аналогичными опциями, что и для *save*:

- `load filename XYZ -option` – загрузка массивов X, Y, Z, вместе с именами переменных, сохраненных в файле `filename.mat` с опциями (включая ключ `-mat` для загрузки файлов с расширением `.mat` обычного бинарного MAT-формата по умолчанию);
- `load('fname')` – загрузка в форме функции переменных файла `fname.mat`.

Если команда (или функция) *load* используется в ходе проведения сессии, то произойдет замена текущих значений переменных значениями из считываемого MAT-файла.

Для задания имен загружаемых файлов может использоваться знак `*`, означающий загрузку всех файлов с определенными признаками. Например, `load demo*.mat` означает загрузку всех файлов с началом имени `demo`, например `demo1`, `demo2`, `demoa`, `demob` и т. д. Имена загружаемых файлов можно формировать с помощью операций над строковыми выражениями.

Ведение дневника. Если есть необходимость записи команд всей сессии на диск, то можно воспользоваться специальной командой для ведения дневника сессии:

- `diary filename.txt` – ведет запись на диск всех команд в строках ввода и полученных результатов в виде текстового файла с указанным именем;
- `diary off` – приостанавливает запись в файл;
- `diary on` – вновь начинает запись в файл.

Таким образом, чередуя команды `diary off` и `diary on`, можно сохранять нужные фрагменты сессии. Команду *diary* можно задать и в виде функции `diary('filename')`, где строка `'filename'` задает имя файла. Просмотреть файл дневника сессии можно командой

```
type filename
```

Для завершения работы с системой можно использовать команды `exit`, `quit` (которые сохраняют содержимое рабочей области и выполняет другие действия в соответствии с файлом завершения `finish.m`) или комбинацию клавиш `Ctrl+Q`. Если необходимо сохранить значения всех переменных (векторов, матриц) системы, то перед вводом команды `exit` следует дать команду *save* нужной формы.

Команда `load` после загрузки системы считывает значения этих переменных и позволяет начать работу с системой с того момента, когда она была прервана.

1.3. Массивы символов

Функции обработки *массивов символов* (строк) имеют большое значение в системе MATLAB. Строковое представление данных лежит в основе символьной математики, арифметики произвольной точности и многочисленных программных конструкций, не говоря уже о том, что оно широко применяется в базах данных и массивах ячеек. В этом параграфе мы рассмотрим возможности MATLAB для обработки символьных переменных и выражений.

1.3.1. Задание массива символов

В основе представления символов в строках лежит их кодирование с помощью таблиц, которые устанавливают взаимно однозначное соответствие между символами и целыми числами от 0 до 255 (код). Вектор, содержащий строку символов, в системе MATLAB задается в командной строке в одинарных кавычках, например

```
S= 'Массив символов'
```

Это вектор длины 15, компонентами которого являются числовые коды, соответствующие символам. Первые 127 чисел — это коды ASCII, представляющие буквы латинского языка, цифры и спецзнаки. Они образуют основную таблицу кодов. Вторая таблица (коды от 128 до 255) является дополнительной и может использоваться для представления символов других языков, например русского. Длина вектора `S` соответствует числу символов в строке, включая пробелы. Апостроф внутри строки символов должен вводиться как два апострофа `' '`. Символьная матрица размера 3-на-4 может быть задана так:

```
S=['abc ';'defg';'hi  ']
```

Обратите внимание, что для выравнивания числа элементов в строках используются пробелы в конце символов (символу пробела соответствует число 32 в кодовой таблице).

1.3.2. Общие функции

Приведем описание основных функций *массивов символов* с примерами их применения.

Функция `char(X)`. Преобразует массив `X` положительных целых чисел (числовых кодов от 0 до 65 535) в массив символов системы MATLAB (причем только первые 127 кодов — английский набор ASCII, со 128 до 255 — расширенный набор ASCII).

Например,

```
char([33:38])
ans =
! "# $ % &
```

На числа, большие, чем 255 функция продолжается по периодичности, т.е. учитывается только остаток $\text{rem}(X, 256)$ от деления на 256, а в случае нецелого – $\text{fix}(\text{rem}(X, 256))$.

Функция `char(C)`. Преобразует массив `C` ячеек строк в символьный массив текстовых строк. При необходимости добавляются пробелы, для того, чтобы в каждой строке было одинаковое число символов. Обратная операция есть `cellstr`. Например,

```
C = [{ 'Алгебра' }; 'Геометрия'; { 'ТФКП' } ];
char(C)
ans =
Алгебра
Геометрия
ТФКП
```

В результате получился символьный массив размера 3-на-9 с добавленными пробелами в первой и третьей строках.

Функция `char(t1, t2, t3)`. Образует массив символов, в котором строками являются текстовые строки `t1`, `t2`, `t3`. В случае необходимости добавляются пробелы для выравнивания. Например,

```
t1='Алгебра';
t2='Геометрия';
t3='ТФКП';
char(t1,t2,t3)
ans =
Алгебра
Геометрия
ТФКП
```

В результате получился символьный массив размера 3-на-9 с добавленными пробелами.

Функция `double(S)`. Преобразует символы строки `S` в числовые коды `double`. Например,

```
double('Математика')
ans =
204    224    242    229    236    224    242    232    234    224
```

Функция `cellstr(S)`. Создает массив ячеек строк символов из символьного массива `S`. Каждая строка массива `S` помещается в отдельную ячейку. При этом пробелы выравнивания, имеющиеся в символьном массиве ликвидируются.

Функция `blanks(n)`. Создает строку из `n` пробелов. Например,

```
disp(['xxx' blanks(20) 'yyy'])
xxx                               yyy
```

Функция `deblank(str)`. Удаляет пробелы в конце символьной строки `str`. В случае массива `S` ячеек строк функция `deblank(C)` действует на строку в каждой ячейке.

1.3.3. Проверка строк

Функция `ischar(S)`. Возвращает логическую единицу, если `S` является символьным массивом, и логический ноль в противном случае.

Функция `iscellstr(C)`. Возвращает логическую единицу, если `C` является массивом ячеек строк символов, и логический ноль в противном случае.

Функции `isletter(S)` и `isspace(S)`. Определяют количество букв и количество пробелов в символьном массиве `S`.

1.3.4. Операции над строками

Рассмотрим операции вертикального объединения, сравнения строк, выравнивания строк, поиска и замены символов в строке и преобразование регистров.

Функция `strcat(s1,s2,s3,...)`. Выполняет горизонтальное объединение массивов символов `s1`, `s2`, `s3`, ..., причем пробелы в конце каждого отдельного массива отбрасываются. Все входные массивы должны иметь одинаковое число строк.

Функция `strvcat(t1,t2,t3,...)`. Выполняет вертикальное объединение строк `t1`, `t2`, `t3`,... в массив символов `S` с автоматическим выравниванием пробелами.

Функция `strcmp('str1','str2')`. Сравнение символьных массивов. Возвращает логическую единицу, если две сравниваемые строки `str1` и `str2` идентичны, и логический ноль в противном случае. Если `S` и `T` — массивы ячеек строк, то `TF = strcmp(S,T)` — возвращает массив того же размера, что и `S,T`, содержащий единицы для идентичных элементов массивов `S` и `T` и нули для всех остальных. Функция `strcmpi` работает так же, но не различает строчные и прописные буквы латинского алфавита.

Функция `strncmp('strT','str2',n)`. Сравнение символьных массивов по первым `n` символам строк. Функция `strncmpi('strT','str2',n)` работает так же, но не различает строчные и прописные буквы латинского алфавита.

Функция `findstr(str1,str2)`. Находит начальные индексы более короткой строки внутри более длинной и возвращает вектор этих индексов. Индексы указывают положение первого символа более короткой строки в более длинной строке. Например,

```
s = 'Find the starting indices of the shorter string.';
findstr(s,'the')
ans =
     6     30
findstr('the',s)
ans =
     6     30
```

Функция `strjust(S)`. Выравнивает вправо массив символов (т. е. перемещает пробелы в конце символов массива, если они есть, в начало тех же строк). Функция `strjust(S, 'left')` выравнивает влево массив символов, а `strjust(S, 'center')` выравнивает по центру массив символов.

Функция `strmatch('str', STRS)`. Просматривает массив символов или строковый массив ячеек `STRS` по строкам, находит строки символов, начинающиеся с строки `str`, и возвращает соответствующие индексы строк. Функция `strmatch('str', STRS, 'exact')` возвращает только индексы тех строк символов массива `STRS`, которые точно совпадают со строкой символов `str`;

Функция `strrep(str1, str2, str3)`. Поиск строки и замена. Заменяет все подстроки `str2`, найденные внутри строки символов `str1` на строку `str3`. Например,

```
s1 = 'This is a good example.';
str = strrep(s1, 'good', 'great')
str =
This is a great example.
```

Функция `strtok('str', d)`. Выдает начало строки до первого разделителя. Использует символ-разделитель по умолчанию (пробел). Символами-разделителями являются также символ табуляции (ASCII-код `d=9`), символ возврата каретки (ASCII-код `d=13`) и пробел (ASCII-код `d=32`).

Функция `[token, rem]=strtok(...)`. Возвращает остаток `rem` исходной строки после символа разделителя. Например,

```
str='This is a good example for me.':
[token, rem] = strtok(str)
token =
This
rem =
is a good example for me.
```

Функция `upper('str')`. Возвращает строку символов `str`, в которой все символы нижнего регистра переводятся в верхний регистр, а все остальные символы остаются без изменений.

Функция `lower('str')`. Переводит все символы верхнего регистра строки `str` в нижний регистр, а все остальные символы остаются без изменений.

1.3.5. Преобразование чисел в символы и обратно

Здесь представлены функции, которые преобразуют числа в обычном формате `double` в их *символьное* представление (запись чисел) в формате `char`.

Функция `num2str(A)`. Преобразование числового массива `A` в символьный массив, представляющий эти числа в MATLAB с точностью до четырех десятичных разрядов и экспоненциальным представлением, если требуется. Обычно используется при выводе графиков совместно с функциями `title`, `xlabel`, `ylabel` или `text`.

num2str(A,precision). Выполняет преобразование массива A в строку символов с точностью, определенной аргументом precision. Аргумент precision определяет число разрядов в выходной строке.

num2str(A,format). Выполняет преобразование массива чисел A, используя заданный формат format. По умолчанию принимается формат, который использует четыре разряда после десятичной точки для чисел с фиксированной или плавающей точкой. Например,

```
A=rand(2,3)
A =
    0.4057    0.9169    0.8936
    0.9355    0.4103    0.0579
str = num2str(A,2)
str =
           % массив типа char размера 2-на-22
0.41    0.92    0.89
0.94    0.41    0.058
```

Функция int2str(X). Преобразование массива X целых чисел в массив символов (цифр) целых чисел. Элементы массива X не целые, то они округляются до целых чисел и строится массив символов, содержащий символьные представления округленных целых чисел. Аргумент X может быть скаляром, вектором или матрицей. Например,

```
>> int2str(133.3)
ans =
133
```

Ответ является массивом типа char размера 1-на-3.

Функция mat2str(A). Преобразует матрицу A в одну символьную строку так, как она задается в MATLAB, подходящую для ввода функции eval. При этом числа преобразуются с полной точностью. Если элемент матрицы не скаляр, то он заменяется на [].

Функция mat2str(A,n) преобразует матрицу A в строку символов, используя точность до n цифр после десятичной точки. Функция eval(str) осуществляет обратное преобразование. Например,

```
mat2str(A,3)
ans =
[0.406 0.917 0.894;0.935 0.41 0.0579] % массив типа char 1x36
```

Функция str2num(s). Выполняет преобразование массива символов числа в ASCII-символах, в числовой массив double. Например,

```
str2num('3.14159e0')
ans =
3.1416
```

Выходной массив ans имеет тип double. Обратите особое внимание, что при этом можно вводить знаки + и – в любом месте строки. MathWorks рекомендует использовать str2num с осторожностью и по возможности заменять ее на str2double.

Функция `str2double('str')`. Выполняет преобразование строки числа `str`, которая представлена в ASCII-символах, в число с двойной точностью. При этом `+` и `-` могут быть только в начале строки. Отметим, что данная функция обеспечивает переход от символьного представления математических выражений в MATLAB к их числовым значениям;

1.3.6. Функции преобразования систем счисления

Некоторые строковые функции служат для *преобразования систем счисления*. Ниже представлен набор этих функций.

Функция `bin2dec('binarystr')`. Преобразование двоичного числа в десятичное представление. Например,

```
bin2dec('101')
ans =
5
```

Функция `dec2bin(d)`. Преобразование десятичного числа `d` в строку (типа `char`) двоичных символов (0 и 1). Аргумент `d` должен быть неотрицательным целым числом, меньшим чем 252 (в случае необходимости применяется округление).

Функция `dec2bin(d,n)` возвращает строку двоичных символов, содержащую по меньшей мере `n` разрядов. Например,

```
dec2bin(111.3, 9)
ans =
001101111
```

Функция `dec2base(d,base)`. Преобразует десятичное число `d` в строку символов, представляющих это число в системе счисления с основанием `base`. Основание `base` быть целым числом в пределах от 2 до 36.

Функция `dec2base(d,base,n)` дает строку символов, представляющих число `d` в системе счисления с основанием `base`, содержащую по меньшей мере `n` знаков. Например,

```
dec2base(365, 21, 5)
ans =
000H8
```

Функция `base2dec(S, B)`. Преобразует строку символов `S`, представляющих число в системе счисления по основанию `B`, в десятичное представление числа в формате `double`. Например,

```
d = base2dec('4D2', 16)
d =
1234
```

Функция `dec2hex(d)`. Преобразует десятичное число `d` в шестнадцатеричную строку символов, представляющих это число в системе счисления с основанием 16.

Функция `hex2dec('hex_value')`. Преобразует шестнадцатеричную строку символов `hex_value` (она содержит символы 0 – 9 и A – F) в десятичное (целое) представление числа в формате `double`. Например,

```
d = hex2dec('10FE3')
d =
    69603
```

1.3.7. Вычисление строковых выражений

Строковые выражения обычно не вычисляются. Однако строка, представляющая математическое выражение, может быть вычислена с помощью функции `eval('строковое выражение')`. Например,

```
eval('2*sin(pi/3)+(1/3)^(1/5)')
ans =
    2.5348
```

Еще один пример. Сначала задаются значения переменных, а затем вычисляется символьное выражение, содержащее эти переменные,

```
a=2; b=4;
eval('a^2 - sqrt(b) + a*b - a/b')
ans =
    9.5000
```

Функция `feval(@имя_функции, x1, x2, ...)` позволяет передавать в вычисляемую функцию список ее аргументов. При этом вычисляемая функция задается только своим именем, например,

```
feval(@prod, [1 2 3])
ans =
    6
```

1.4. Массивы ячеек

Массив ячеек – это массив, элементами которого являются ячейки, содержащие массивы любого типа, в том числе и массивы ячеек. Массивы ячеек позволяют хранить массивы с элементами разных типов и разных размерностей. К примеру, одна из ячеек может содержать действительную матрицу, другая – массив текстовых строк, третья – вектор комплексных чисел (рис. 1.4.1). Можно создавать массивы ячеек любой размерности.

При работе с массивами ячеек можно использовать следующие функции.

- **`{}`**, **`cell`** – создание массив ячеек;
- **`cellstr`** – создание массива ячеек строк из символьного массива;
- **`cellfun`** – применение функции к каждому элементу в массиве ячеек;
- **`celldisp`** – показ содержимого массива ячеек;
- **`cellplot`** – показ графической структуры массива ячеек;
- **`deal`** – обмен данными;

- **num2cel** – преобразование числового массива в массив ячеек;
- **cell2mat** – преобразование массива ячеек в отдельную матрицу;
- **mat2cell** – разбиение матрицы на массив ячеек матриц;
- **cell2struct** – преобразование массива ячеек в структуру;
- **struct2cell** – преобразование структуры в массив ячеек;
- **iscell** – определяет, является ли введенная переменная массивом ячеек.

1.4.1. Создание массивов ячеек

Для создания массивов ячеек используются *конструкторы* `{}` и `cell` и некоторые функции для работы с ячейками. Конструктор `{}` действует подобно оператору `[]` для числовых массивов. Он объединяет данные в ячейки.

Пример 1. Образование массива из четырех ячеек,

```
C = {1 2 3 4}
C = [1]    [2]    [3]    [4]
```

Индекс в круглых скобках `C(j)` определяет отдельную ячейку. А индекс в фигурных скобках `C{j}` обозначает содержимое соответствующей ячейки. В приведенном примере `C{1}`, `C{2}`, `C{3}`, `C{4}` есть числа. Следующая операция объединяет их в один вектор

```
A = [C{:}]
A = 1    2    3    4
```

Для образования массива ячеек можно использовать привычные операторы горизонтального и вертикального объединения. Например, следующая команда создает массив 2-на-2 ячеек строк символов

```
V = {'МатАнализ', {'Геометрия'}; {'Алгебра'}, {'ТФКП'}}
V =
    'МатАнализ'    'Геометрия'
    'Алгебра'      'ТФКП'
```

Можно построить массив ячеек, присваивая данные отдельным ячейкам. Система MATLAB автоматически строит массив по мере ввода данных.

Задание ячеек с использованием индексации. Для индексов ячейки массива используются круглые скобки (стандартные обозначения для массива). Содержимое ячейки в правой части оператора присваивания заключается в фигурные скобки `{}`.

Пример 2. Создание массива ячеек `A` размера 2-на-2, который содержит матрицу, строку символов, число и вектор.

```
A(1, 1) = {[1 4 3; 0 5 8]};
A(1, 2) = {'Математика'};
A(2, 1) = {3+7i};
A(2, 2) = {-2:2:6}
A =
    [2x3 double]    'Математика'
    [3.0000+ 7.0000i]    [1x5 double]
```


Замечание 1. Символ { } соответствует пустому массиву ячеек точно так же, как [] соответствует пустому числовому массиву. Фигурные скобки { } являются конструктором массива ячеек, а квадратные [] – конструктором числового массива. Фигурные скобки аналогичны квадратным скобкам, за исключением того, что они могут быть еще и вложенными.

Задание содержимого с использованием индексации. Обращение к содержанию ячейки массива производится с использованием индексов ячейки в фигурных скобках. Содержимое ячейки указывается в правой части оператора присваивания, как это показано на следующем примере.

Пример 3.

```
A{1, 1} = [1 4 3; 0 5 8];
A{1, 2} = 'Математика';
A{2, 1} = 3+7i;
A{2, 2} = -2:2:6;
A =
    [2x3 double]    'Математика'
    [3.0000+ 7.0000i]    [1x5 double]
```

Замечание 2. Система MATLAB не очищает массив ячеек при выполнении оператора присваивания. Могут остаться старые данные в незаполненных ячейках. Полезно удалять массив перед выполнением оператора присваивания.

Замечание 3. Нельзя называть одним именем разные массивы, даже, если один из них числовой массив, а другой – массив ячеек.

Система MATLAB отображает массив ячеек в сжатой форме. В частности, если содержимое ячейки есть массив, то отображается только размерность и тип (см. пример выше). Для отображения содержимого ячеек следует использовать функцию *celldisp*:

```
celldisp(A)
A{1,1} =
    1      4      3
    0      5      8
A{2, 1} = 3.0000+ 7.0000i
A{1,2} = Математика
A{2, 2} =      -2      0      2      4      6
```

Для отображения структуры массива ячеек в виде графического изображения (рис. 1.4.1) предназначена функция *cellplot*:

```
cellplot(A)
```

Так же, как и в случае числового массива, если данные присваиваются ячейке, которая находится вне пределов текущего массива, MATLAB автоматически расширяет массив ячеек. При этом ячейки, которым не присвоено значений, заполняются пустыми массивами.

Применение функции cell. Функция *cell* позволяет создать шаблон массива ячеек, заполняя его пустыми ячейками. Например, для создания пустого массива ячеек В размера 2x3 можно следующей командой

```
B = cell(2, 3)
```

Используя оператор присваивания можно заполнить ячейки массива В.

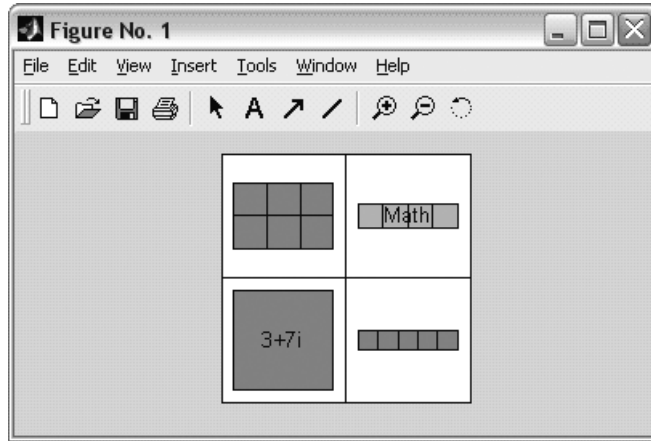


Рис. 1.4.1. Массив ячеек размера 2-на-2

1.4.2. Доступ к данным в ячейках

Существует два способа извлечения данных из массива ячеек:

- доступ к содержимому ячейки, используя индексацию содержимого;
- доступ к подмножеству ячеек, используя индексацию ячеек.

Доступ к содержимому ячеек. Используя индексацию ячеек и содержимого ячеек можно получить доступ к некоторым или всем данным в ячейке. Индексы ячейки указываются в фигурных скобках, а индексы массива в ячейке – в круглых. При этом извлекается содержимое ячеек, а не сами ячейки.

Пример 4. Рассмотрим массив ячеек A размера 2-на-2, определенный ранее. Строку, находящуюся в ячейке $A\{1, 2\}$ можно извлечь следующим образом:

```
c = A{1, 2}
c = Математика      %массив типа char
```

Пример 5. Извлечение элемента с индексами (2,2) из числового массива ячеек $A\{1, 1\}$:

```
d = A{1, 1}(2, 2)
d = 5
```

Доступ к подмножеству ячеек. Используя индексацию в массиве ячеек (круглые скобки), можно получить доступ к подмножествам ячеек внутри массива ячеек. Результат будет массивом ячеек. Например, следующая команда выбирает первую строку в массиве ячеек.

```
B=A(1,:)
B =
[2x3 double]      'Математика'
```

Таким же образом можно удалить ячейки из массива. При этом, удалять можно либо целую строку, либо столбец. Например, следующая команда удаляет первую строку

```
A(1, :) = []
A =
    3.0000+ 7.0000i    [1x5 double]
```

Массив ячеек стал одномерным. Команда

```
A(2) = []
```

удаляет вторую ячейку.

Отметим еще раз, что фигурные скобки используются для обозначения содержимого ячейки. MATLAB обрабатывает содержимое каждой ячейки как отдельную переменную.

Пример 6. Определим массив ячеек C, содержащих векторы одинаковой длины.

```
C(1) = {[1 2 3]}; C(2) = {[4 5 6]}; C(3) = {[7 8 9]};
```

Выведем на экран векторы из ячеек

```
C{1:3}
ans =
     1     2     3
ans =
     4     5     6
ans =
     7     8     9
```

Можно сформировать новый числовой массив, используя следующий оператор присваивания

```
B = [C{1}; C{2}; C{3}]
B =
     1     2     3
     4     5     6
     7     8     9
```

Аналогичным образом, используя фигурные скобки, можно создать новый массив ячеек, используя в качестве ячеек выходные переменные функций. Например, команда

```
[D{1:2}] = eig(B)
D =
 [3x3 double] [3x3 double]
```

создает массив, первая ячейка которого есть первая выходная переменная функции eig, а вторая ячейка – для второй выходной переменной,

Можно вывести на экран матрицы правых собственных векторов и собственных значений, используя ячейки D{1} и D{2}, соответственно.

```
D{1}
ans =
 -0.2320    -0.7858     0.4082
 -0.5253    -0.0868    -0.8165
 -0.8187     0.6123     0.4082
```

```
D{2}
ans =
    16.1168         0         0
         0    -1.1168         0
         0         0    -0.0000
```

1.4.3. Вложенные массивы ячеек

Допускается, что ячейка может содержать массив ячеек и даже массив массивов ячеек. Массивы, составленные из таких ячеек, называются вложенными. Сформировать вложенные массивы ячеек можно с помощью последовательности фигурных скобок, функции `cell` или операторов присваивания. Для уже созданных массивов можно получить доступ к отдельным ячейкам, подмассивам ячеек или элементам самих ячеек.

Применение фигурных скобок. Для создания вложенных массивов ячеек можно применять фигурные скобки, как показано на следующем примере.

```
clear A
A(1, 1) = {magic(5)};
A(1, 2) = {[5 2 8; 7 3 0; 6 7 3] 'Test 1'; [2-4i 5+7i] {17 []}};
cellplot(A)
```

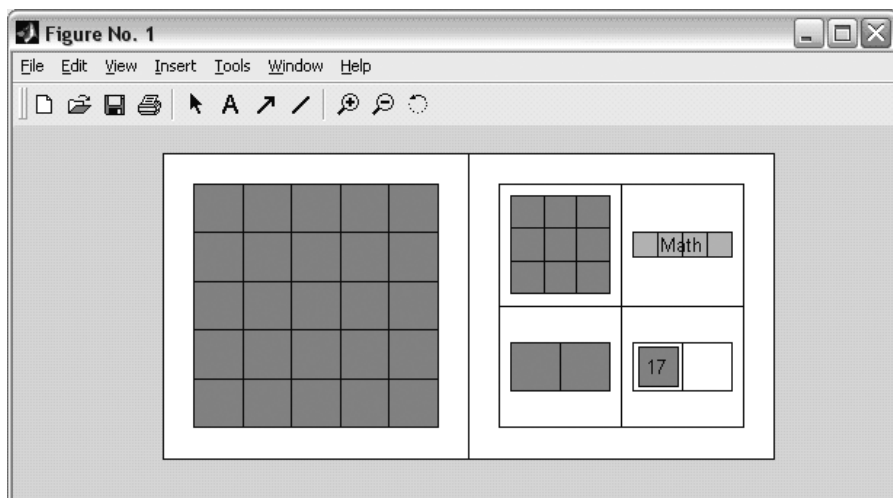


Рис. 1.4.2. Вложенные массивы ячеек

Заметим, что в правой части последнего оператора присваивания использовано 3 пары фигурных скобок: первая пара определяет ячейку `A(1, 2)` массива `A`, вторая задает внутренний массив ячеек размера 2-на-2, который, в свою очередь, содержит ячейку `{17 []}`.

Применение функции `cell`. Для создания вложенного массива ячеек вышеприведенного примера с помощью функции `cell` можно следовать следующей схеме:

1. Создать пустой массив ячеек размера 1x2,

```
A = cell(1, 2);
```

2. Создать пустой массив ячеек A(1, 2) размера 2x2 внутри массива A,

```
A(1, 2) = {cell(2, 2)};
```

3. Заполнить массив A, включая вложенный массив, с помощью операторов присваивания,

```
A(1, 1) = {magic(5)};
A{1, 2}(1, 1) = {[5 2 8; 7 3 0; 6 7 3]};
A{1, 2}(1, 2) = {'Test 1'};
A{1, 2}(2, 1) = {[2-4i 5+7i]};
A{1, 2}(2, 2) = {cell(1,2)}
A{1, 2}{2, 2}(1) = {17};
```

Обратите внимание на использование фигурных скобок для последнего уровня вложенности. Это обусловлено тем, что необходимо обратиться к содержанию ячейки внутри массива ячеек.

Можно также сформировать вложенные массивы ячеек простым присваиванием значений его элементам, как это сделано выше на последнем шаг.

Индексация вложенных ячеек. Для того чтобы индексировать вложенные ячейки, необходимо использовать объединение индексов. Первое множество индексов определяет доступ к верхнему уровню ячеек, а последующие индексные выражения, заключенные в фигурные скобки, задают доступ к более глубоким уровням. Отметим, что приведенный выше массив ячеек A имеет 3 уровня вложенности:

- для доступа к числовому массиву размера 5-на-5 в ячейке (1,1) надо использовать обращение A{1, 1};
- для доступа к массиву ячеек размера 2x2 в ячейке (1, 2) надо использовать обращение A{1, 2};
- для доступа к числовому массиву размера 3x3 в позиции (1, 1) ячейки (1, 2) надо использовать обращение A{1, 2}{1, 1};
- для доступа к элементу (2, 2) предыдущего числового массива надо использовать обращение A{1, 2}{1, 1}(2, 2);
- для доступа к пустой ячейке в позиции (1, 2) ячейки (2, 2), вложенной в ячейку A(1, 2), надо использовать обращение A{1, 2}{2, 2}{1, 2}.

1.4.4. Массивы ячеек, содержащих структуры

Для того чтобы объединить структуры с разными архитектурами полей, удобно использовать массивы ячеек. Создание такого массива рассмотрим на примере.

Пример 7.

```
cel_str = cell(1,2)
cel_str{1}.label = '19.09.05';
cel_str{2}.xdat = [-0.03 0.41 1.98 2.12 17.11];
```

```

cel_str{2}.ydat = [-3 5 8 0 9];
celldisp(c_str)
cel_str{1} =
label: '19.09.05'
cel_str{2} =
xdat: [-0.0300 0.4100 1.9800 2.1200 17.1100]
ydat: [-3 5 8 0 9]

```

Ячейка {1} массива `cel_str` содержит структуру из одного поля `label` – строка, ячейка {2} – два поля с числовыми векторами.

При построении массивов ячеек, включающих структуры, необходимо использовать контекстную индексацию вида

```
cell_array{index}.field
```

Для доступа к содержимому структур внутри ячеек используется такая же индексация. Например, чтобы получить доступ к полю `label` ячейки {1} приведенной выше следующей структуры, необходимо использовать обращение `cel_str{1}.label`.

1.4.5. Многомерные массивы ячеек

Для образования многомерного массива ячеек можно просто использовать функцию `cat`. Например, создадим следующий 3-мерный массив ячеек `C`, размера 2-на-2-на-2, объединяющий 2-мерные массивы ячеек `A` и `B`:

```

A{1, 1} = 'Name';
A{1, 2} = [4 2; 1 5];
A{2, 1} = 2-4i;
A{2, 2} = 7;
B{1, 1} = 'Name2';
B{1, 2} = [ 3 5 ]';
B{2, 1} = 0:1:3;
B{2, 2} = 3;
C = cat(3, A, B);

```

1.5. Массивы структур

Структура – это массив записей с именованными полями, предназначенными для хранения различных данных. Каждое поле может содержать данные любого типа. MATLAB имеет следующие функции при работе с массивами структур:

- **struct** – создание массива структур;
- **fieldnames** – получение имен полей;
- **getfield** – получение содержимого поля;
- **setfield** – установка содержимого поля;
- **rmfield** – удаление поля;
- **isfield** – истинно, если это поле массива структур;
- **isstruct** – истинно, если это массив структур;
- **struct2cel** – преобразование массива структур в массив ячеек.

1.5.1. Построение структур

Структуру можно построить с использованием операторов присваивания и с использованием функции `struct`.

Применение оператора присваивания. Для того чтобы создать простейшую структуру размера 1-на-1, необходимо присвоить данные соответствующим полям. Система MATLAB автоматически формирует структуру по мере ее заполнения.

Пример 1. Предположим, что формируется база данных фотографий. Тогда можно создать структуру `summer` размера 1-на-1 с тремя полями: само изображение, описание фотографии и дата. Следующий код MATLAB создает одну структуру:

```
summer.image = 'image1';
summer.description = 'На берегу Томи в Подъяково';
summer.date.year = 2007;
summer.date.month = 07;
summer.date.day = 20;
```

Структура `summer` содержит три поля: `image`, `description` и `date`. Поле `date` – это самостоятельная структура и содержит три дополнительных поля: `year`, `month` и `day`. Обратите внимание, что структуры могут содержать различные типы данных, изображения содержат матрицы (изображения), строки (описание) и другие структуры (дата).

Если теперь введем в командной строке имя структуры

```
summer,
то получим описание структуры:
>> summer
summer =
    image: 'image1'
description: 'На берегу Томи в Подъяково'
    date: [1x1 struct]
```

Таким образом, `summer` – это пока массив из одной записи с тремя полями. Для того чтобы расширить его, достаточно добавить индекс в имени структуры.

Пример 2. Создадим вторую запись в структуре `summer`.

```
summer(2).image = 'image2';
summer(2).description = 'Вечер у костра';
summer(2).date.year = 2007;
summer(2).date.month = 07;
summer(2).date.day = 22;
```

Теперь структура `summer` имеет размер 1-на-2. Заметим, что когда структура содержит более одной записи, при ее запросе, содержимое полей не выводится, а выводится только обобщенная информация о структуре в следующем виде:

```
>> summer
summer =
1x2 struct array with fields:
```

```
image  
description  
date
```

Для получения этой же информации можно использовать функцию *fieldnames*, которая возвращает массив ячеек, содержащий строки с именами полей.

При расширении структуры система MATLAB заполняет неприсвоенные поля пустыми массивами. При этом все элементы массива структур имеют одинаковое количество полей и все имена полей одинаковы. Размеры полей могут быть разными для разных записей. Для структуры *summer* поле *description* может иметь строки различной длины, поля *image* могут содержать массивы разных размеров и так далее.

Применение функции *struct*. Функция *struct* имеет следующий синтаксис:

```
str_array = struct('<имя_поля1>', '<значение>',  
                  '<имя_поля2>', '<значение>', ...).
```

Пример 3. Воспользуемся функцией *struct*, чтобы создать структуру *summer* размера 1-на-1, содержащую вложенную структуру:

```
summer = struct('image', 'image2', 'description', 'Вечер у костра',  
               'date', struct('year', 2007, 'month', 07, 'day', 22))  
summer =  
    image: 'image2'  
description: 'Вечер у костра'  
    date: [1x1 struct]
```

1.5.2. Доступ к полям и данным структуры

Используя индексацию, можно легко определить значение любого поля или элемента структуры. Точно так же можно присвоить значение любому полю или элементу поля. Чтобы обратиться к некоторому полю, необходимо ввести точку (.) после имени структуры, за которым должно следовать имя поля. Например,

```
str = summer(2).description  
str =  
Вечер у костра
```

Чтобы обратиться к элементам поля, надо использовать индексацию поля в правой части оператора присваивания. Другими словами, если содержание поля является числовым массивом, то использовать индексы массива; если поле – массив ячеек, использовать индексы массива ячеек и т. п. Например, если поле *image* содержит массив чисел *n*-на-*m*, то можно выбрать некоторый элемент этого массива с индексами 2 и 3:

```
n = summer(2).image(2,3)
```

Используя тот же подход, можно присваивать значения элементам поля в левой части оператора присваивания. Получить значение некоторого поля для всех элементов структуры нельзя, это можно сделать только для отдельной записи.

Пример 4. Для вывода всех значений поля *image* необходимо организовать цикл:


```
for i = 1 : length(summer)
disp(summer(i).image)
end
image1
image2
```

Чтобы получить доступ к элементу, необходимо указать соответствующий индекс в массиве структур.

Пример 5. Результатом выполнения нижеследующего оператора является структура размера 1-на-1, которая соответствует второй фотографии структуры `summer`,

```
B = summer(2)
```

Функции `setfield` и `getfield`. Непосредственная индексация – это, как правило, наиболее эффективный способ определить или присвоить значение полю записи. Однако, если использовалась функция `fieldnames` и известно имя поля, то можно воспользоваться функциями `setfield` и `getfield`.

Функция `getfield` позволяет определить значение поля или элемента поля:

```
f = getfield(array, {array_index}, 'field', {field_index})
```

где аргументы `array_index` и `field_index` задают индексы для структуры и поля; они не являются обязательными для структуры размера 1-на-1. Результат применения функции `getfield` соответствует элементу следующей структуры

```
f = array(array_index).field(field_index);
```

Пример 6. Чтобы получить доступ к полю `description` второй записи структуры `summer`, необходимо использовать функцию `getfield` в следующей форме

```
str = getfield(summer, {2}, 'description')
```

```
str =
```

```
Вечер у костра
```

Аналогично, функция `setfield` позволяет присваивать значения полям, используя обращение следующего вида

```
f = setfield(array, {array_index}, 'field', {field_index}, value)
```

Применение функции `size`. Функция `size` позволяет получить размер массива структур или любого его поля. Задавая в качестве аргумента имя структуры, функция `size` возвращает ее размеры. При задании аргумента в форме `array(n) . field` функция `size` возвращает размеры поля. Например, функция

```
size(summer)
```

для структуры `summer` размера 1x2 возвращает вектор

```
ans = 1      2
```

Обращение

```
size(summer(2).description)
```

возвращает размер поля `description` для структуры `summer(2)`

```
ans = 1      14
```

Для того чтобы добавить новое поле к структуре, достаточно добавить поле к одной записи. Для удаления поля из структуры предназначена функция `rmfield`, которая имеет следующий синтаксис

```
struc2 = rmfield(array, 'field'),
```

где `array` – имя структуры, а `'field'` – имя поля, которое подлежит удалению. Для удаления поля `name` в структуре `patient` надо использовать оператор

```
summer = rmfield(summer, 'day');
```

Замечание. Выполнение операций с полями и элементами полей производит-ся совершенно аналогично операциям с элементами обычного числового массива. В обоих случаях надо использовать индексные выражения.

Вложенные структуры. Поле структуры может само включать другую струк-туру или даже массив структур. Как только структура создана, с помощью опера-торов присваивания или функции `struct` можно вложить структуры в существую-щие поля. Эту процедуру мы уже применяли в примере 3 создания структуры `summer` размера 1-на-1, содержащую вложенную структуру:

```
summer = struct('image', 'image2', 'description', 'Вечер у костра',  
'date', struct('year', 2007, 'month', 07, 'day', 22))
```

Просто делается обращение к функции `struct` внутри функции `struct`. Приве-дем еще один пример.

Пример 7. Допустим, что требуется создать структуру размера 1-на-1. Органи-зуем следующий вложенный вызов функции `struct`:

```
A = struct('data', [3 4 7; 8 0 1], 'nest', ...  
struct('testnum', 'Test 1', 'xdata', [4 2 8], 'ydata', [7 1 6]))  
A =  
data: [2x3 double]  
nest: [1x1 struct]
```

Запись `A(1)` содержит требуемые значения, благодаря вызову внешней функ-ции `struct`. Следующая последовательность операторов производит результат, аналогичный предыдущему для массива структур 1-на-1:

```
A(1).data = [3 4 7; 8 0 1];  
A(1).nest.testnum = 'Test 1';  
A(1).nest.xdata = [4 2 8];  
A(1).nest.ydata = [7 1 6];  
A(2).data = [9 3 2; 7 6 5];  
A(2).nest.testnum = 'Test 2';  
A(2).nest.xdata = [3 4 2];  
A(2).nest.ydata = [5 0 9]  
A =  
1x2 struct array with fields:  
data  
nest
```

Вложенные массивы структур можно также создавать с использованием опе-раторов присваивания.

Индексация вложенных структур. Для индексации структуры нужно добавить имена вложенных полей, используя в качестве разделителя точку (.). Первая текстовая строка индексного выражения определяет имя структуры, а последующие имена полей, содержащих другие структуры. Например, вышеописанный массив `A` имеет 2 уровня вложенности:

- для получения доступа к вложенной структуре внутри `A(1)` надо использовать команду `A(1).nest`;
- для получения доступа к полю `xdata` вложенной структуры внутри `A(1)` надо использовать команду `A(2).nest.xdata`;
- для получения доступа к элементу 2 поля `ydata` вложенной структуры внутри `A(1)` надо использовать команду `A(1).nest.ydata(2)`.

1.5.3. Многомерные массивы структур

Многомерные массивы структур рассматриваются как расширение двумерных массивов структур. По аналогии с другими типами многомерных массивов их можно создавать, либо используя индексацию и операторы присваивания, либо функцию `cat`.

Пример 8. Сформируем многомерный массив структур размера 1-на-2-на-2 (два листа с двумя фотографиями), используя операторы присваивания. Достаточно задать значения полей только последней структуре:

```
summer(1,2,2).image = 'image4';  
summer(1,2,2).description = 'Утро на Томи';  
summer(1,2,2).date.year = 2007;  
summer(1,2,2).date.month = 07;  
summer(1,2,2).date.day = 24;
```

Для применения функций к многомерным массивам структур надо использовать индексный подход, чтобы получить доступ к полям записи и элементам полей.

1.6. Программирование в среде MATLAB

В этом параграфе мы рассмотрим дополнительные вопросы относительно М-функций, основные операторы программирования М-языка, управление памятью и обработку ошибок.

1.6.1. М-функции

Файлы, которые содержат коды языка MATLAB, называются М-файлами. Для создания М-файла используется текстовый редактор (редактор/отладчик MATLAB). Существует два типа М-файлов: М-сценарии (м-файлы скрипты) и М-функции.

М-сценарий представляет просто последовательность команд MATLAB без входных и выходных параметров. Сценарий оперирует с данными из рабочей

области. Результаты выполнения М-сценария сохраняются в рабочей области после завершения сценария и могут быть использованы для дальнейших вычислений.

М-функции – это новые функции MATLAB, которые расширяют возможности системы. М-функции используют входные и выходные аргументы, внутренние локальные переменные. Напомним, что каждая М-функция имеет следующую структуру:

- строка определения функции. Она задает имя функции и количество входных и выходных аргументов, их локальные имена. Например
`function y = function_name(u,v,w)`
- первая строка комментария определяет назначение функции. Она выводится на экран с помощью команд `lookfor` или `help < имя функции >`;
- основной комментарий. Он выводится на экран вместе с первой строкой при использовании команды `help < имя функции >`. Основной комментарий начинается со второй строки комментария и заканчивается либо пустой строкой, либо началом программного кода;
- тело функции – это программный код, который реализует вычисления и присваивает значения выходным аргументам.

Если выходных параметров больше, то они указываются в квадратных скобках после слова `function`, например,

```
function [x, y, z] = sphere(theta, phi, rho)
```

Имена входных переменных не обязаны совпадать с именами, указанными в строке определения функции. М-функция записывается в файл с тем же названием, что и функция и с расширением `m`.

Подфункции. М-функции могут содержать коды для более, чем одной функции. Первая функция в файле – это основная функция, вызываемая по имени М-файла. Другие функции внутри файла – это *подфункции*, которые являются видимыми только для основной функции и других подфункций этого же файла. Каждая подфункция имеет свой собственный заголовок. Подфункции следуют друг за другом непрерывно. Подфункции могут вызываться в любом порядке, в то время как основная функция выполняется первой.

Пример 1. Следующая функция находит среднее значение и медиану для элементов вектора `u`, используя встроенную функцию `n = length(u)` и подфункции `avg = mean(u,n)` и `med = median(u,n)`.

```
function [avg,med] = newstats(u) % Основная функция
n = length(u);
avg = mean(u,n);
med = median(u,n);
```

```
function a = mean(v,n)          % Подфункция
% Вычисление среднего
a = sum(v)/n;
```

```
function m = median(v,n)       % Подфункция
```

```
% Вычисление медианы
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

Когда функция вызывается из М-файла, то MATLAB сначала проверяет, является ли вызванная функция подфункцией М-файла. Затем ищет частную (private) функцию с тем же именем и, наконец, ищет обычный М-файл на пути поиска файлов. Поэтому нет необходимости заботиться о том, чтобы имя подфункции не совпало с именем существующей функции MATLAB.

Частные функции. *Частный* каталог представляет собой подкаталог с именем private родительского каталога. М-файлы частного каталога доступны только М-файлам родительского каталога. Поскольку файлы частного каталога не видны вне родительского каталога, они могут иметь имена совпадающие, с именами файлов других каталогов системы MATLAB. Это удобно в тех случаях, когда пользователь создает собственные версии некоторой функции, сохраняя оригинал в другом каталоге. Поскольку MATLAB просматривает частный каталог раньше каталогов стандартных функций системы MATLAB он в первую очередь использует функцию из частного каталога.

Вызов функции. М-функцию можно вызвать из командной строки системы MATLAB или из других М-файлов, обязательно указав все необходимые атрибуты – входные аргументы в круглых скобках, выходные аргументы в квадратных скобках. Когда появляется новое имя функции, или переменной, система MATLAB проверяет:

- является ли новое имя именем переменной;
- является ли это имя именем подфункции данного М-файла;
- является ли оно именем частной функции, размещаемой в каталоге private;
- является ли оно именем функции в пути доступа системы MATLAB.

В случае дублирования имен система MATLAB использует первое имя в соответствии с вышеприведенной 4-уровневой иерархией. Следует отметить, что в системе MATLAB допускается переопределять функцию по правилам объектно-ориентированного программирования.

При вызове М-функции, система MATLAB транслирует функцию в *псевдокод* и загружает в память. Это позволяет избежать повторного синтаксического анализа. Псевдокод остается в памяти до тех пор пока не будет использована команда *clear* или завершен сеанс работы. Команда *clear* применяется следующим образом:

- `clear <имя_функции>` – удаление указанной функции из рабочей области;
- `clear functions` – удаление всех откомпилированных программ;
- `clear all` – удаление программ и данных.

Откомпилированные М-функции или М-сценарии можно сохранить для последующих сеансов, используя команду `rcode` в форме:

```
rcode myfunc
```

Эта команда выполняет синтаксический анализ М-файла `myfunc.m` и сохраняет результирующий псевдокод (р-код) в файле с именем `myfunc.p`. Это позволяет избежать повторного разбора во время нового сеанса работы. При удалении М-файла `myfunc.m` система работает с Р-кодом `myfunc.p`. Однако справка об этой функции уже недоступна. Применение Р-кода целесообразно в двух случаях:

- когда требуется выполнять синтаксический анализ большого числа М-файлов, необходимых для визуализации графических объектов в приложениях, связанных с разработкой графического интерфейса пользователя;
- когда пользователь хочет скрыть алгоритмы, реализованные в М-файле.

Рабочая область функции. Каждой М-функции выделяется дополнительная область памяти, не пересекающаяся с рабочей областью системы MATLAB. Такая область называется рабочей областью функции. При работе с системой MATLAB можно получить доступ только к переменным, размещенным в рабочей области системы или в рабочей области функции. Если переменная объявлена глобальной, то ее можно рассматривать как бы принадлежащей нескольким рабочим областям.

Проверка количества аргументов. Функции `nargin` и `nargout` позволяют определить количество входных и выходных аргументов вызываемой функции. Эту информацию в дальнейшем можно использовать в условных операторах для изменения хода вычислений. Например,

```
function c = testarg1(a,b)
if (nargin == 1)
    c = a.^2;
elseif (nargin == 2)
    c = a + b;
end
```

При задании единственного входного аргумента функция вычисляет квадрат входной переменной; при задании двух аргументов выполняется операция сложения.

Заметим, что порядок следования аргументов в выходном списке имеет важное значение. Если при обращении к М-функции выходной аргумент не указан, по умолчанию выводится первый аргумент. Для вывода последующих аргументов требуется соответствующее обращение к М-функции.

Произвольное количество аргументов. В MATLAB имеются функции, которые могут иметь меняющееся число входных аргументов и меняющееся число выходных параметров. Например, функция `S=svd(A)` вычисления сингулярных чисел матрицы A . Она может применяться в виде `[U, S, V]=svd(A)`, когда требуется большее число выходных параметров. Другим примером такой функции может служить функция `cat(A,B)` горизонтального объединения массивов A и B . Она может иметь произвольное число входных массивов, `cat(A1, A2, A3, A4)`.

Для создания таких функций, использующих неопределенной количество аргументов, в список аргументов вставляют переменные `varargin` и `varargout`

(variable argument input, variable argument output). Переменная `varargin` должна быть последней в списке входных аргументов, после всех обязательных. Переменная `varargout` должна быть последней в списке выходных переменных. Функции `varargin` и `varargout` позволяют передавать произвольное количество входных и выходных аргументов. Тогда система MATLAB упаковывает входные и выходные аргументы в массивы ячеек `varargin` и `varargout`. Каждая ячейка может содержать любой тип и любое количество данных.

При обращении к такой функции переменные, число которых может меняться, вызываются так же, как и обязательные переменные.

Пример 2. Функция `testvar` допускает в качестве входных аргументов любое количество векторов из двух элементов и выводит на экран линии, их соединяющие.

```
function testvar(varargin)
for k = 1:length(varargin)
    x(k) = varargin{k}(1);    % Выбор координат k-го вектора
    y(k) = varargin{k}(2);    % из k-ой ячейки
end
xmin = min(0,min(x));
ymin = min(0,min(y));
axis([xmin fix(max(x))+3 ymin fix(max(y))+3])
plot(x,y)
```

Таким образом, функция `testvar` может работать с входными списками разной длины, например,

```
testvar([2 3],[1 5],[4 8],[6 5],[4 2],[2 3])
testvar([-1 0],[3 -5],[4 2],[1 1])
```

Формирование входного массива `varargin`. Поскольку список `varargin` хранит входные аргументы в массиве ячеек, то необходимо использовать индексы ячеек для извлечения данных. Индекс ячейки состоит из двух компонентов, например, `y(i) = varargin{i}(2);`

Здесь индекс в фигурных скобках `{i}` указывает доступ к содержанию *i*-ой ячейки массива `varargin`, а индекс в круглых скобках `(2)` указывает на второй элемент массива в ячейке.

Формирование выходного массива `varargout`. При произвольном количестве выходных аргументов их необходимо упаковать в массив ячеек `varargout`. Чтобы определить количество выходных аргументов функции, надо использовать функцию `nargout`.

Пример 3. Следующая функция использует в качестве входа массив из двух столбцов, где первый столбец – множество значений координаты *x*, а второй – множество значений координаты *y*. Функция разбивает массив на отдельные векторы-строки, которые могут быть переданы в функцию `testvar` в качестве входов,

```
function [varargout] = testvar2(arrayin)
for k = 1:nargout
    varargout{k} = arrayin(k,:) % Запись значений в массив ячеек
end
```

Отметим, что оператор присваивания в цикле `for` использует синтаксис массивов ячеек. А именно, фигурные скобки указывают, что данные в виде строки массива присваиваются ячейке. Вызвать функцию `testvar2` можно следующим образом:

```
a = {1 2;3 4;5 6;7 8;9 0};
[p1,p2,p3,p4,p5] = testvar2(a);
```

При использовании массивов ячеек в списках аргументов, массивы ячеек `varargin` и `varargout` должны быть последними в соответствующих списках аргументов. Например, приведенные ниже обращения к функциям показывают правильное использование списков `varargin` и `varargout`:

```
function[out1, out2] = example1(a,b,varargin)
function[i,j,varargout] = example2(x1,y1,x2,y2,flag)
```

Локальные и глобальные переменные. Использование переменных в М-файле ничем не отличается от использования переменных в командной строке, а именно:

- переменные не требуют объявления; прежде чем переменной присвоить значение;
- любая операция присваивания создает переменную, или изменяет значение существующей переменной;
- имена переменных начинаются с буквы, за которой следует любое количество букв, цифр и подчеркиваний; система MATLAB не поддерживает кириллицу и различает символы верхнего и нижнего регистров;
- имя переменной не должно превышать 31 символа. Более точно, имя может быть и длиннее, но система MATLAB принимает во внимание только первые 31 символ.

Обычно каждая М-функция, задаваемая в виде М-файла, имеет собственные *локальные* переменные, которые отличны от переменных других функций и переменных рабочей области. Однако, если несколько функций и рабочая область объявляют некоторую переменную глобальной, то все они используют единственную копию этой переменной. Любое присваивание этой переменной распространяется на все функции, где она объявлена глобальной.

Пример 4. Допустим, требуется исследовать влияние коэффициентов α и β для модели хищник-жертва, описываемой уравнениями Лотке-Вольтерра:

$$\begin{aligned}\dot{y}_1 &= y_1 - \alpha y_1 y_2, \\ \dot{y}_2 &= -y_2 + \beta y_1 y_2.\end{aligned}$$

Создадим М-файл `lotka.m`, который является векторной функцией правой части данной системы уравнений.

```
function yp = lotka(t, y)
%ЛОТКА уравнения Лотке-Вольтерра для модели хищник-жертва
global ALPHA BETA
yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

Затем через командную строку введем переменные, которые должны быть приняты функцией `lotka.m` (*глобальные* переменные), решим систему и построим графики решений в одном окне.


```
global ALPHA BETA
ALPHA = 0.01;
BETA = 0.02;
[t,y] = ode23('lotka',[0 10],[1; 1]);
plot(t,y)
```

Команда `global` объявляет переменные `ALPHA` и `BETA` глобальными и следовательно, доступными в функции `lotka.m`. Таким образом, они могут быть изменены из командной строки, а новые решения будут получены без редактирования М-файла `lotka.m`. Для работы с глобальными переменными необходимо:

- объявить переменную как глобальную в каждой М-функции, которой необходима эта переменная. Для того чтобы переменная рабочей области была глобальной, необходимо объявить ее как глобальную из командной строки;
- в каждой функции использовать команду `global` перед первым появлением переменной; рекомендуется указывать команду `global` в начале М-файла.

1.6.2. Операторы системы MATLAB

Операторы системы MATLAB делятся на три категории:

- арифметические операторы. Они позволяют конструировать арифметические выражения и выполнять числовые вычисления;
- операторы отношения. Они позволяют сравнивать аргументы;
- логические операторы позволяют строить логические выражения.

Логические операторы имеют самый низкий приоритет относительно операторов отношения и арифметических операторов.

Арифметические операторы. Это операция сложения, вычитания, умножения (матричного и поэлементного), деления (матричного и поэлементного), возведения в степень (матричного и поэлементного), транспонирования и сопряжения и оператор двоеточия (`:`). При работе с массивом чисел установлены обычные уровни приоритета среди арифметических операций.

Первый уровень. Поэлементное транспонирование (`.'`), поэлементное возведение в степень (`.^`), сопряжение матрицы (`'`), возведение матрицы в степень (`^`).

Второй уровень. Унарный плюс (`+`) и унарный минус (`-`).

Третий уровень. Умножение массивов (`*`), правое деление (`./`), левое деление массивов (`.\`), умножение матриц (`*`), решение систем линейных уравнений – операция (`/`), операция (`\`).

Четвертый уровень. Сложение (`+`) и вычитание (`-`) массивов.

Пятый уровень. Оператор двоеточия (`:`).

Внутри каждого уровня операторы имеют равный приоритет и вычисляются в порядке следования слева направо. Заданный по умолчанию порядок следования может быть изменен с помощью круглых скобок.

Замечание. Арифметические операторы системы MATLAB работают с массивами одинаковых размеров, за исключением единственного случая, когда один из них – скаляр. Если один из операндов скалярный, а другой нет, в системе

MATLAB принято, что скаляр расширяется до размеров второго операнда и заданная операция применяется к каждому элементу. Такая операция называется расширением скаляра.

Операторы отношения. В системе MATLAB определено 6 следующих операторов отношения.

Таблица 1.6.1. Операторы сравнения

Оператор	Функция	Описание	Пример
<	lt ()	Меньше	A<B; lt (A, B) ;
<=	le ()	Меньше или равно	A<=B; le (A, B) ;
>	gt ()	Больше	A>B; gt (A, B) ;
>=	ge ()	Больше или равно	A>=B; ge (A, B) ;
==	eq ()	Равно	A==B; eq (A, B) ;
~=	ne ()	Не равно	A~=B; ne (A, B) ;

Операторы отношения выполняют поэлементное сравнение двух массивов равных размерностей. Для векторов и прямоугольных массивов, оба операнда должны быть одинакового размера, за исключением случая когда один из них скаляр. В этом случае MATLAB сравнивает скаляр с каждым элементом другого операнда. Позиции, где это соотношение истинно, получают логическое значение 1, где ложно – логическое значение 0.

Операторы отношения обычно применяются для изменения последовательности выполнения операторов программы. Поэтому они чаще всего используются в теле операторов if, for, while, switch. При вычислении арифметических выражений операторы отношения имеют более низкий приоритет, чем арифметические, но более высокий, чем логические операторы.

Логические операторы. В состав *логических* операторов системы MATLAB входят следующие три оператора. В примерах следующей таблицы используются следующие массивы (они не обязаны быть целочисленными):

A = [0 1 1 0 1];

B = [1 1 0 0 1];

Таблица 1.6.2. Логические операторы

Оператор	Функция	Описание	Пример
&	and ()	Создает логический массив, в котором 1 – для каждого местоположения, в котором оба элемента имеют значение true (отличны от нуля) и 0 – для всех других элементов	A & B = 01001
	or ()	Возвращает 1 для каждого местоположения, в котором хотя бы в один из элементов имеет значение true (отличен от нуля) и 0 для всех других элементов	A B = 11101
~	not ()	Логическое отрицание для каждого элемента входного массива, A	~A = 10010

Таблица 1.6.2. Логические операторы (окончание)

Оператор	Функция	Описание	Пример
<code>xor</code>		Возвращает 1 для каждого местоположения, в котором только один элемент является true (отлично от нуля) и 0 для всех других элементов	<code>xor(A, B) = 10100</code>

Логические операторы реализуют поэлементное сравнение массивов одинаковых размерностей. Для векторов и прямоугольных массивов оба операнда должны быть одинакового размера, за исключением случая, когда один из них скаляр. В последнем случае MATLAB сравнивает скаляр с каждым элементом другого операнда. Позиции, где это соотношение истинно, получают значение 1, где ложно – 0.

Отметим также логические операторы, действующие по короткой схеме (short-circuit). Их применение позволяет получить результат по одному из аргументов, без оценки второго.

Таблица 1.6.3. Логические операторы укороченной схемы

Оператор	Описание
<code>&&</code>	Возвращает true (1), если оба ввода есть true, и false (0), если не так
<code> </code>	Возвращает true (1), если или один аргумент, или оба имеют значение true, и false (0), если не так

Пусть, например, применяется команда

```
A && B
```

Если A равняется нулю, то полное выражение оценивается как false, независимо от значения B. При этих обстоятельствах, нет необходимости оценивать B, потому что результат уже известен.

Логические функции. В дополнение к логическим операторам в состав системы MATLAB включено ряд логических функций.

Функция `xor(a, b)`. Результат есть TRUE, если один из операндов имеет значение TRUE, а другой FALSE. Для числовых выражений, функция возвращает 1, если один из операндов отличен от нуля, а другой – нуль.

Функция `all`. Она возвращает 1, если все элементы истинны (отличны от нуля). Например, пусть задан вектор `u` и требуется проверить его на условие «все ли элементы меньше 3?».

```
u = [1 2 3 4];
v = all(u < 3)
v =
0
```

В случае массивов функция `all` проверяет столбцы, то есть является ориентированной по столбцам.

Функция any. Она возвращает 1, если хотя бы один из элементов аргумента отличен от нуля; иначе, возвращается 0. В случае массивов функция any применяется к столбцам.

Функции isnan и isinf. Возвращают 1 для NaN и Inf, соответственно. Функция isfinite истинна только для величин, которые не имеют значения inf или NaN.

Функция find. Определяет индексы элементов массива, которые удовлетворяют заданному логическому условию. Как правило, она используется для создания шаблонов для сравнения и создания массивов индексов. В наиболее употребительной форме функция `i = find(x <условие>)` возвращает вектор индексов тех элементов, которые удовлетворяет заданному условию.

Пример 5. Построим матрицу и поставим значение 100 вместо каждого элемента, который больше 6.

```
>> A = magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
>> i = find(A > 6);
A(i) = 100
A =
    100     1     6
     3     5    100
     4    100     2
```

Функция вида `[i, j] = find(x)` позволяет получить индексы ненулевых элементов прямоугольного массива. Функция вида `[i, j, s] = find(x)` возвращает кроме того и их значения в виде вектора `s`.

Полный список логических функций системы MATLAB содержится в каталоге `<matlab>\toolbox\matlab\ops\`.

1.6.3. Управление последовательностью исполнения операторов

Существуют восемь операторов управления последовательностью исполнения инструкций:

- **if** – оператор условия, в сочетании с оператором `else` и `elseif` выполняет группу инструкций в соответствии с некоторыми логическими условиями;
- **switch** – оператор переключения, в сочетании с операторами `case` и `otherwise` выполняет различные группы инструкций в зависимости от значения некоторого логического условия;
- **while** – оператор условия, выполняет группу инструкций неопределенное число раз, в соответствии с некоторым логическим условием завершения;
- **for** – оператор цикла, выполняет группу инструкций фиксированное число раз;

- **continue** – передает управление к следующей итерации цикла `for` или `while`, пропуская оставшиеся инструкции в теле цикла;
- **break** – прерывает выполнение цикла `for` или `while`;
- **try...catch** – изменяет управление потоком данных при обнаружении ошибки во время выполнения;
- **return** – возвращение к функции вызова.

Все операторы управления включают оператор **end**, чтобы указать конец блока, в котором действует этот оператор управления.

Оператор условия `if...else...elseif...end`. Применяется в трех формах.

Первая форма:

```
if логическое_выражение
    команда
end
```

Вторая форма:

```
if логическое_выражение
    команда
else
    команда
end
```

Третья форма:

```
if логическое_выражение
    команда
elseif логическое_выражение
    команда
else
    команда
end
```

Описание. Оператор условия `if ... end` вычисляет некоторое логическое выражение и выполняет соответствующую группу инструкций в зависимости от значения этого выражения. Если логическое выражение истинно, то MATLAB выполнит все инструкции между `if` и `end`, а затем продолжит выполнение программы в строке после `end`. Если условие ложно, то MATLAB пропускает все утверждения между `if` и `end` и продолжит выполнение в строке после `end`. Аналогично работают операторы условия `if ... else ... end` и `if ... elseif ... else ... end`. Отметим следующие особенности.

Если в операторе `if` условное выражение обращается к пустым массивом, то такое условие ложно.

Оператор `else` не содержит логического условия. Инструкции, связанные с ним, выполняются, если предшествующий оператор `if` (и возможно `elseif`) ложны.

Оператор `elseif` содержит логическое условие, которое вычисляется, если предшествующий оператор `if` (и возможно `elseif`) ложны. Инструкции, связанные с оператором `elseif` выполняются, если соответствующее логическое условие истинно. Оператор `elseif` может многократно использоваться внутри оператора условия `if`.

Пример 6. Определение четности числа. Проверяется равенство нулю остатка от деления числа на 2.

```
if rem(a,2) == 0
    disp('a is even')
    b = a/2;
end
```

Пример 7. Определение четности числа. Проверяется равенство нулю остатка от деления числа на 2.

```
if rem(a,2) == 0
    disp('a is even')
    b = a/2;
else
    disp('a is odd')
    b = (a+1)/2;
end
```

Оператор переключения *switch...case...otherwise...end*. Синтаксис.

```
switch выражение (скаляр или строка)
case value1
    команды % Исполняются, если выражение есть value1
case value2
    команды % Исполняются, если выражение есть value2
...
otherwise
    команды % Исполняются, если не обработана ни одна
              % из предыдущих групп case
End
```

Описание. Оператор `switch ... case 1 ... case k ... otherwise ... end` выполняет ветвления, в зависимости от значений некоторой переменной или выражения. Оператор переключения включает:

- заголовок `switch`, за которым следует вычисляемое выражение (скаляр или строка);
- произвольное количество групп `case`. Заголовок группы состоит из слова `case`, за которым следует возможное значение выражения, расположенное на одной строке. Последующие строки содержат инструкции, которые выполняются для данного значения выражения. Выполнение продолжается до тех пор, пока не встретится следующий оператор `case` или оператор `otherwise`. На этом выполнение блока `switch` завершается;
- группа `otherwise`. Заголовок включает только слово `otherwise`, начиная со следующей строки размещаются инструкции, которые выполняются, если значение выражения оказалось не обработанным ни одной из групп `case`;
- Оператор `end` является последним в блоке переключателя.

Оператор `switch` работает, сравнивая значение вычисленного выражения со значениями групп `case`. Для строковых выражений, оператор `case` истинен, если функция сравнения строк `strcmp(значение, выражение)` дает истинное значение.

Пример 8. Функция $n!!$. Напомним, что в случае четного $n=2k$, $n!!$ есть произведение четных чисел от 2 до $2k$, а в случае нечетного $n=2k-1$, $n!!$ есть произведение нечетных чисел от 1 до $2k-1$. Вычисляется остаток $\text{rem}(n,2)$ от деления числа n на 2. Если число четное (остаток $=0$), то вычисляется произведение четных чисел. Если число нечетное (остаток $=1$), то вычисляется произведение нечетных чисел.

```
function ff = fact2(n)
% FACT2 Вычисление факториала n!!.
% fact2(n) возвращает n!! числа n
switch rem(n,2)
case 0
    ff = prod(2:2:n);
case 1
    ff = prod(1:2:n);
otherwise
end
```

В данной программе не использован оператор `otherwise`. Не будет считаться ошибкой, если его совсем опустить.

Оператор цикла с неопределенным числом операций *while...end*. Синтаксис:

```
while expression
    statements
end
```

Описание. Оператор цикла с неопределенным числом операций `while ... end` многократно выполняет инструкцию или группу инструкций, пока управляющее выражение истинно. Если выражение использует массив, то все его элементы должны быть истинны для продолжения выполнения. Можно использовать функции `any` и `all`.

Пример 9. Этот цикл с неопределенным числом операций находит первое целое число n , для которого $n!$ – записывается числом, содержащим 100 знаков:

```
n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end
```

Выход из `while`-цикла может быть реализован с помощью оператора `break`. Если в операторе `while`, управляющее условие является пустым массивом, то такое условие ложно.

Оператор цикла с определенным числом операций *for...end*. Синтаксис:

```
for index = start:increment:end
    statements
end
```

Описание. Оператор цикла `for ... end` выполняет инструкцию или группу инструкций определенное число раз. По умолчанию приращение равно 1. Можно задавать любое приращение, в том числе отрицательное. Для положительных индексов выполнение завершается, когда значение индекса превышает конечное значение; для отрицательных приращений выполнение завершается, когда

индекс становится меньше чем конечное значение. Возможны вложенные циклы, например

```
for i = 1:m
    for j = 1:n
        A(i,j) = 1/(i + j - 1);
    end
end
```

В качестве переменной цикла `for` могут использоваться массивы, например, следующее условие определяет, какие значения может принять переменная цикла `for p=[5,6,9,10,17,18,16];`

1.6.4. Вычисление символьных выражений

В MATLAB имеется возможность исполнения символьных выражений. Кроме того можно обращаться по имени к ранее написанным функциям и вызывать их в зависимости от ситуации.

Функция *eval*. Вычисляет символьное выражение. В простейшей форме имеет вид

```
eval('string')
```

Описание. Функция `eval('string')` интерпретирует и вычисляет выражение в строке `string`, которое может быть либо арифметическим выражением, либо командой, либо обращением к функции. Например, вычисление текущего времени `t`,

```
format rational
eval('t = clock')
t =
    2005     9    21    17    16   1983/40
```

Пример 10. Следующий программный код позволяет создать матрицу Гильберта порядка `n`:

```
t = '1/(i + j-1)';
n = 4;
for i = 1:n
    for j = 1:n
        G(i,j) = eval(t);
    end
end
format rational
G
G =
     1          1/2          1/3          1/4
    1/2          1/3          1/4          1/5
    1/3          1/4          1/5          1/6
    1/4          1/5          1/6          1/7
```

Функция *feval*. Вычисление функции по заданному имени. Имеет синтаксис:
`[y1,y2,...] = feval(function,x1,...,xn)`

Если function есть строка, содержащая имя функции (m-файл), то feval(function, x1, ..., xn) вычисляет эту функцию при данных значениях аргументов. Параметр function должен быть простым именем функции без информации о пути. Например, следующие команды эквивалентны.

```
[V,D] = eig(A)
[V,D] = feval(@eig,A)
```

Пример 11. Пусть задан некоторый список функций fun. Требуется выбрать по номеру функцию из списка и вычислить ее для значения x, которое вводится из командной строки.

```
fun = [@sin; @cos; @log];
k = input('Choose function number: ');
x = input('Enter value: ');
feval(fun(k),x)
```

1.6.5. Ошибки и предупреждения

Независимо от того, как тщательно проверяется программа, она не всегда работает так гладко, как хотелось бы. Поэтому желательно включить проверку ошибок в программы, чтобы гарантировать выполнение операции при всех условиях. Во многих случаях желательно предпринять определенные действия при возникновении ошибок. Например, можно потребовать ввода недостающих аргументов, или повторить вычисление, используя значения по умолчанию. Возможности обработки ошибок в MATLAB позволяют приложению проверять условия ошибки и выполнять соответствующий код в зависимости от ситуации.

Когда в коде имеются инструкции, которые могут генерировать нежелательные результаты, то нужно помещать эти инструкции в блок *try-catch*, который захватывает любые ошибки и обрабатывает их соответственно. Следующий показывает блок try-catch в пределах обычной функции, которая умножает две матрицы:

```
function matrix_multiply(A,B)
try
    X = A*B
catch
    disp '** Error multiplying A*B'
end
```

Блок try-catch разделен на два раздела. Первый начинается с try и второй с catch. Заканчивается блок символом end.

Все инструкции в части try выполняются обычно, так если бы они были в обычном коде. Но если любая из этих операций приводит к ошибке, MATLAB пропускает остальные инструкции в try и переходит к разделу catch блока.

Сегмент catch обрабатывает ошибку. В приведенном примере – это отображение общего сообщения об ошибках. Если есть различные виды ошибок, то можно уточнить, какая ошибка была захвачена и ответить на ту определенную ошибку. Можно также попробовать избавиться от ошибки в разделе catch.

Можно также вложить блоки try-catch, как показано ниже. Это можно использовать для исправления ошибки (выбор второго варианта действий), захваченной в первом разделе try.

```
try
    statement1
catch
    try
        statement2
    catch
        disp 'Operation failed'
    end
end
```

Функция lasterror дает информацию о последней ошибке во время выполнения программы.

Предупреждения системы MATLAB аналогичны сообщениям об ошибках, за исключением того, что выполнение программы не прекращается. Для вывода на экран предупреждающих сообщений предназначена функция warning, имеющая следующий синтаксис:

```
warning('<строка_предупреждения>')
```

1.6.6. Повышение эффективности обработки М-файлов

Этот раздел описывает некоторые методы повышения быстродействия при выполнении программы. MATLAB – это язык, специально разработанный для обработки массивов и выполнения матричных операций. Всюду, где это возможно, пользователь должен учитывать это обстоятельство.

Векторизация циклов. Под векторизацией понимается преобразование циклов for и while к эквивалентным векторным или матричным выражениям. При векторизации алгоритма ускоряется выполнение М-файла.

Пример 12. Один из способов вычисления 1001 значения функции синуса на интервале [0, 10] может использовать оператор цикла,

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

Эквивалентная векторизованная форма имеет вид

```
t = 0:.01:10;
y = sin(t);
```

В этом случае вычисления выполняются намного быстрее, и такой подход в системе MATLAB является предпочтительным. Время выполнения этих М-файлов можно оценить, используя команды tic и toc.

Предварительное выделение памяти. В системе MATLAB есть возможность для существенного сокращения времени выполнения программы за счет предварительного размещения массивов для выходных данных. Предварительное распределение избавляет от необходимости изменять массив при увеличении его размеров. Например, сделаем предварительное выделение памяти для числового массива,

```
y = zeros(1, 100)
for i = 1:100
    y(i) = det(X^i);
end
```

Предварительное выделение памяти позволяет избежать фрагментации памяти при работе с большими матрицами. В ходе сеанса работы системы MATLAB, память может стать фрагментированной из-за работы механизмов динамического распределения и освобождения памяти. Это может привести к появлению большого количества фрагментов свободной памяти, тогда непрерывного пространства памяти может оказаться недостаточно для хранения какого-либо большого массива. Предварительное выделение памяти позволяет определить непрерывную область, достаточную для проведения всех вычислений.

Функции управления памятью. Существует несколько подходов к повышению эффективности использования памяти, рассмотренные ниже. В системе MATLAB предусмотрено пять функций для работы с *памятью*:

- **clear** – удаление переменных из оперативной памяти;
- **pack** – запись текущих переменных на диск и последующей их загрузкой;
- **quit** – по мере необходимости выход системы MATLAB с освобождением всей памяти;
- **save** – сохранение переменных в файле.
- **load** – считывание данных из файла.

Память выделяется для переменной при ее возникновении. Для экономии памяти надо:

- избегать использовать одни и те же переменные в качестве входных и выходных аргументов функции, поскольку они будут передаваться ссылкой;
- после использования переменной целесообразно либо присвоить ей пустой массив, либо удалить с помощью команды `clear` имя переменной;
- стремиться использовать переменные повторно.

Глобальные переменные. При объявлении глобальной переменной в таблицу переменных просто помещается флаг. При этом не требуется дополнительной памяти. Например, последовательность операторов

```
a = 5;
global a
```

определяет переменную `a` как глобальную и формируется дополнительная копия этой переменной. Функция `clear a` удаляет переменную `a` из рабочей области системы MATLAB, но сохраняет ее в области глобальных переменных. Функция `clear global a` удаляет переменную `a` из области глобальных переменных.

Компилятор MATLAB® версии 4.6

2.1. Основы работы с Компилятором MATLAB®	86
2.2. Процесс создания компонента MATLAB®	97
2.3. Работа с mcs и mbuild	100
2.4. Примеры создания автономных приложений и библиотек	111
2.5. Классы C++ Компилятора 4.6 MATLAB®	127
2.6. Внешние интерфейсы	138
2.7. Передача значений между C/C++ double, mxArray и mxArray	143
2.8. Математическая библиотека C++ MATLAB® 6.5	153

Компилятор MATLAB® позволяет из m-функций создавать автономные приложения, C и C++ библиотеки совместного использования. Расширения Компилятора могут создавать дополнения к Excel, компоненты .NET и Java. Продукты, которые производит Компилятор и его расширения называются компонентами MATLAB. В данной главе рассмотрим использование Компилятора для создания C и C++ библиотек совместного использования и независимых от MATLAB консольных приложений. Возможности его расширений будут рассмотрены в следующих главах.

Рассматриваемый Компилятор 4.6 является усовершенствованной версией Компилятора MATLAB версии 4 пакета MATLAB® R14. В более ранних выпусках, MATLAB® R12 и MATLAB® R13, использовался Компилятор MATLAB версии 3, который работал совершенно на других принципах, в частности, он использовал математическую библиотеку C/C++ MATLAB и из m-функций создавал полноценные исходные коды C/C++. Подробнее о работе этого Компилятора версии 3 см. в [ППС]. Начиная с выпуска MATLAB® R14 математическая библиотека C/C++ была заменена на среду выполнения компонентов MATLAB MCR (MATLAB Component Runtime). Среда исполнения компонентов MATLAB более универсальна, поскольку поддерживает работу компонентов MATLAB, созданных для использования в языках программирования C++, Java, VBA для Excel, C# и других допустимых языках платформы .Net Framework.

Математическая библиотека MATLAB специально создавались для использования в C/C++, поэтому она для языка C++ пока удобнее универсальной среды выполнения MCR. В последнем параграфе данной главы изложены основы работы с математической библиотекой C++ при программировании на Borland C++ Builder.

Предполагается, что читатель знаком с основами языка C++. Необходимые сведения о программировании на C++ можно найти в книгах [ППС], [ДХ] или [Хо].

2.1. Основы работы с Компилятором MATLAB®

В данном разделе кратко излагаются первоначальные сведения о Компиляторе MATLAB версии 4.6 на примере создания простого приложения.

2.1.1. Назначение Компилятора MATLAB

Компилятор MATLAB® используется для преобразования программ MATLAB в приложения и библиотеки, которые могут работать независимо от системы MATLAB. Можно компилировать m-файлы, MEX-файлы и другие коды MATLAB. Компилятор MATLAB поддерживает все особенности MATLAB, включая объекты, частные функции и методы. Компилятор MATLAB используется для создания:

- автономных C и C++ приложений на платформах Windows, UNIX и Macintosh;
- C и C++ библиотек совместного использования (динамически подключаемых библиотек, или dll, на Windows).

Отметим, что функции некоторых пакетов расширения (toolboxes) MATLAB недоступны для Компилятора MATLAB. Для получения точной информации об этом лучше обратиться на сайт www.mathworks.com, MATLAB Compiler product page.

2.1.2. Инсталляция и конфигурирование

Компилятор MATLAB устанавливается вместе с MATLAB®. Для этого следует выбрать установку компоненты MATLAB Compiler. Компилятор не налагает особых требований к операционной системе, памяти и дисковому пространству. Для работы Компилятора MATLAB требуется, чтобы на системе был установлен внешний ANSI C или C++ компилятор, поддерживаемый MATLAB. Для MATLAB R2007a можно использовать один из следующих 32-разрядных C/C++ компиляторов:

- Lcc C версии 2.4.1 (включен в MATLAB), это – только C компилятор, но не C++;
- Borland C++ версии 5.5 и 5.6 (эти компиляторы использует Borland C++ Builder версии 5.0, и 6.0.);
- Microsoft Visual C/C++ (MSVC) версии 6.0, 7.1 и 8.0.

Отметим, что единственный компилятор, который поддерживает создание COM объектов и дополнений к Excel – это Microsoft Visual C/C++ (версии 6.0, 7.1 и 8.0). Единственный компилятор, который поддерживает создание .NET объектов – это компилятор Microsoft Visual C# для .NET Framework версии 1.1 и 2.0.

Компилятор MATLAB поддерживает системные компиляторы Solaris. На Linux, Linux x86-64, и Mac OS X Компилятор MATLAB поддерживает gcc и g++.

Перечень поддерживаемых компиляторов может меняться. Последний список всех поддерживаемых компиляторов см. на сайте MathWorks <http://www.mathworks.com/support/tech-notes/1600/1601.shtml>

Конфигурирование. Внешний компилятор ANSI C или C++ необходимо сконфигурировать для работы с Компилятором MATLAB. Для этого имеется утилита mbuild MATLAB. Она обеспечивает простое решение следующих задач:

- выбор внешнего компилятора для MATLAB и задание параметров настройки компоновщика;
- замена компилятора или его параметров настройки;
- создание приложения.

Для выбора компилятора в командной строке MATLAB используется команда:

```
mbuild -setup
```

При выполнении этой команды MATLAB определяет список всех имеющихся на системе компиляторов C/C++ и предлагает выбрать один из списка. Выбран-

ный компилятор становится компилятором по умолчанию. Для замены компилятора нужно снова выполнить `mbuild -setup`.

Конфигурирование внешнего компилятора происходит автоматически при выполнении команды `mbuild -setup`. Для выбранного компилятора создается файл опций `comports.bat`, который сохраняется в пользовательском (user profile) каталоге `C:\Documents and Settings\UserName\Application Data\MathWorks\MATLAB\R2007a`. Файл опций содержит параметры настройки и флаги, которые управляют работой внешнего C/C++ компилятора. Для создания файла опций система MATLAB имеет готовые сконфигурированные файлы опций, которые приведены ниже в табл. 2.1.1 (они находятся в каталоге `<Matlab_root>\bin\win32\mbuildopts\`).

Таблица 2.1.1. *Файлы опций*

Файл опций	Компилятор
<code>lcccomp.bat</code>	Lcc C, Version 2.4.1 (включен в MATLAB)
<code>msvc60comp.bat</code>	Microsoft Visual C/C++, Version 6.0
<code>msvc71comp.bat</code>	Microsoft Visual C/C++, Version 7.1
<code>msvc80comp.bat</code>	Microsoft Visual C/C++, Version 8.0
<code>bcc55freecomp.bat</code>	Borland C/C++ (free command-line tools) Version 5.5
<code>bcc55comp.bat</code>	Borland C++ Builder 5
<code>bcc56comp.bat</code>	Borland C++ Builder 6

В процессе работы `mbuild -setup` берется один из этих сконфигурированных файлов опций, в нем указывается местонахождение внешнего компилятора и после этого файл опций сохраняется под именем `comports.bat` в указанном выше пользовательском каталоге профилей. При замене компилятора происходит замена файла опций. Вообще говоря, файл опций допускает пользовательское редактирование, однако обычно в этом нет необходимости.

2.1.3. Пример использования Компилятора

Для вызова Компилятора MATLAB используется команда `mcc`. Однако, начиная с версии MATLAB® R2006b можно использовать графический интерфейс пользователя Компилятора MATLAB – Deployment Tool.

Среда разработки Deployment Tool

Команда MATLAB

`deploytool`

вызывает среду разработки, которая позволяет создать проект, добавить к проекту файлы, создать приложение или библиотеку и сделать инсталляционный пакет для распространения пользователю.

Среда разработки открывается как присоединяемое окно справа от командного окна MATLAB (рис 2.1.1) и к строке меню MATLAB добавляется элемент меню **Project**. Это окно можно сделать и отдельным. Для этого достаточно взять его правой кнопкой мыши и перенести на удобное для работы место.

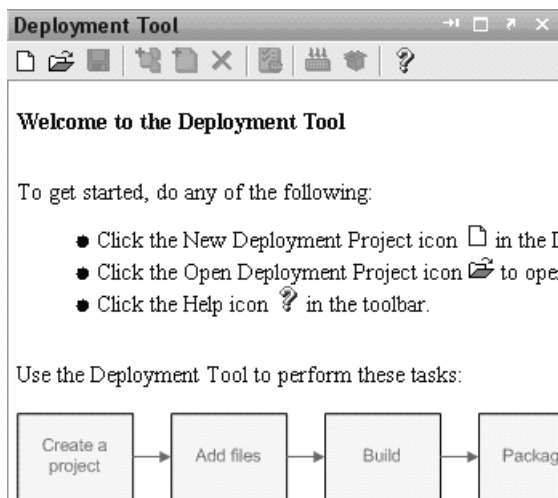


Рис. 2.1.1. Рабочее окно среды разработки Компилятора MATLAB

Инструментальная панель Deployment Tool имеет следующие кнопки (рис. 2.1.1):

- **New Project** – создание нового проекта;
- **Open Project** – просмотр проектов и выбор одного из них;
- **Save Project** – сохранение текущего проекта, включая все файлы и параметры настройки;
- **Add Class** – открытие диалогового окна Add Class, где можно определить название нового класса, который будет создан как часть текущего проекта (если компонент содержит классы);
- **Add File** – добавление файлов в папки проекта;
- **Remove** – удаление выбранной папки класса или выбранных файлов проекта;
- **Build** – построение компонента, определенного проектом с отображением процесса создания в окне вывода;
- **Package** – создание самоизвлекающегося exe-файла для Windows, или zip-файла для UNIX, который содержит файлы, нужные для использования компонента в приложении;
- **Settings** – изменение настроек проекта;
- **Help** – справка по использованию Deployment Tool.

При работе Deployment Tool внизу основного окна MATLAB открывается новое поле, в котором отражается информация о процедуре построения. Эти данные записываются также в файл build.log проекта. Данное окно вывода имеет дополнительные функциональные возможности, доступные через правую кнопку мыши. В частности, имеется возможность управлять действиями, зарегистрированными в окне вывода, при помощи опций **Back** и **Forward**, обновлять окно

вывода и печатать его содержание. Опции **Selection** позволяют, после выбора определенного текста в окне вывода, получить следующее:

- **Evaluate Selection** – выполнить отмеченный текст, как будто это была команда, введенная в MATLAB;
- **Open Selection** – открыть выбранный файл, если отмеченный текст содержит правильный путь;
- **Help on Selection** – открыть справку MATLAB для выбранного текста, если этот текст есть документированная функция MATLAB.

Создание приложения

Покажем на примере использование среды разработки для создания независимого приложения, которое получается компиляцией m-файла `magicsquare.m`, вычисляющего магический квадрат. Это квадратная матрица, которая имеет одинаковые суммы элементов строк, столбцов и диагоналей. М-файл `magicsquare.m` находится в каталоге примеров `matlabroot\extern\examples\compiler`, где `matlabroot` – есть корневой каталог MATLAB. Текст m-файла `magicsquare.m`:

```
function m = magicsquare(n)
if ischar(n)
    n=str2num(n);
end
m = magic(n)
```

Приведем пошаговую процедуру создания приложения `MagicExample.exe` и инсталляционного пакета.

1. Создание файлов и каталога проекта. Каталог для разработки данного проекта может быть любым. Пусть, например это будет подкаталог `MagicExample` в рабочем каталоге MATLAB `D:\R2007a\work\MagicExample\`. Файл `magicsquare.m` для проекта уже имеется. Достаточно скопировать его из каталога примеров MATLAB `matlabroot\extern\examples\compiler\` в созданный каталог проекта `D:\R2007a\work\MagicExample`. Каталог проекта необходимо сделать текущим каталогом MATLAB.

2. Запуск среды разработки Компилятора MATLAB. Для этого в командной строке MATLAB достаточно выполнить следующую команду:

```
deploytool
```

3. Создание проекта. В меню **File**, или в инструментальной панели выбираем **New Deployment Project**. Открывается диалоговое окно (рис.2.1.2) в котором выбираем процедуру создания автономного приложения, **Standalone Application**. В поле **Name** диалогового окна определяем название проекта, пусть это будет `MagicExample.prj`. Кроме того, можно выбрать каталог проекта, хотя по умолчанию указывается текущий рабочий каталог MATLAB. Нажимаем кнопку **OK**. В окне среды разработки Компилятор MATLAB отображает структура проекта `MagicExample.prj`. Проект содержит три папки (рис. 2.1.3), которые пока являются пустыми. Используя меню **Project** добавляем к нашему проекту необходимые m-файлы. В данном проекте это будет один m-файл `magicsquare.m`.

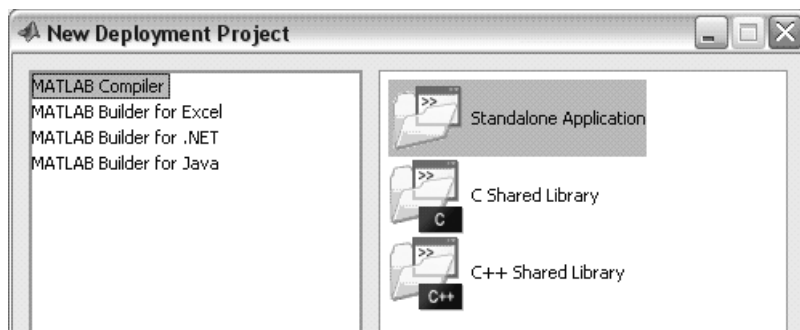


Рис. 2.1.2. Выбор типа проекта Компилятора MATLAB

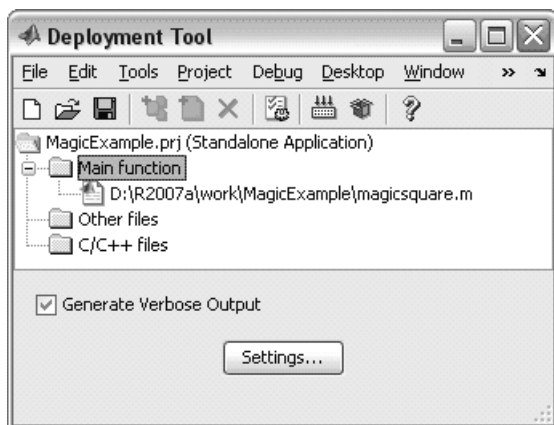


Рис. 2.1.3. Проект MagicExample

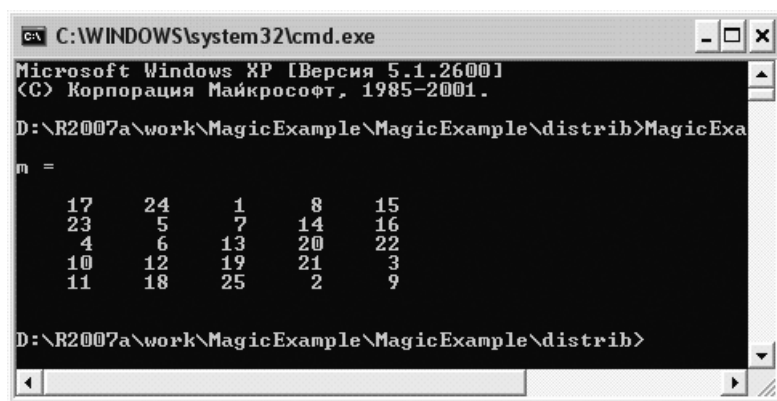
4. **Создание приложения.** В меню **Tools** выбираем **Build** – построение проекта (для этого имеется и кнопка на инструментальной панели). Начинается процесс построения приложения. Он может быть достаточно долгим. При этом в окне MATLAB открывается новое поле, в котором отражается информация о процедуре построения. Эти данные записываются также в файл build.log проекта.

По окончании процедуры построения Компилятор MATLAB помещает полученные файлы приложения в двух созданных им подкаталогах, **src** и **distrib** каталога проекта MagicExample. Каталог **distrib** содержит файлы MagicExample.exe и MagicExample.ctf, которые предназначены для инсталляционного пакета приложения. Каталог **src** содержит более полный набор файлов, в частности он содержит два log-файла и достаточно подробный файл readme.txt. Этот файл указывает системные и другие требования, которые нужно учесть при установке созданного компонента на другую машину. Подробнее о создаваемых файлах см. в разделе 2.1.5.

Файл MagicExample.exe запускается из строки DOS. Для получения магического квадрата, например, порядка 5, нужно исполнить команду

MagicExample.exe 5

из каталога D:\R2007a\work\MagicExample\MagicExample\src\, где находится приложение. Результаты изображены на рис. 2.1.4.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

D:\R2007a\work\MagicExample\MagicExample\distrib>MagicExa
m =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

D:\R2007a\work\MagicExample\MagicExample\distrib>
```

Рис. 2.1.4. Результаты работы приложения MagicExample

5. Создание инсталляционного пакета. В меню **Tools** выбираем **Package** – упаковку приложения (для этого имеется и кнопка на инструментальной панели). Компилятор MATLAB создает пакет в подкаталоге **distrib**. На Windows это самораспаковывающийся пакет MagicExample_pkg.exe. На других платформах – это zip-файл. Инсталляционный пакет MagicExample_pkg.exe содержит файлы MagicExample.exe, MagicExample.ctf и файл _install.bat – для запуска установки среды MCR, необходимой для работы приложения без MATLAB.

6. Установка приложения на другую машину. Созданное приложение можно установить на другую машину, которая имеет ту же самую операционную систему, что и машина разработки. Кроме созданного программного обеспечения, в состав пакета для установки должен входить файл MCRIInstaller.exe, который расположен в следующем каталоге matlabroot\toolbox\compiler\deploy\win32. Этот файл есть архив, содержащий все библиотеки MATLAB, необходимые для работы приложения (отметим, что этот файл достаточно большой – более 125 Мб, а после распаковки он занимает более 270 Мб на диске). В случае, когда используется другая операционная система, файл MCRIInstaller.exe заменяется на файл MCRIInstaller.zip, который можно создать в MATLAB командой buildmcr.

Таким образом, для установки приложения на другую машину необходимо сделать следующее:

- запустить MCRInstaller.exe для установки библиотек MCR, необходимых для работы приложения без MATLAB. По умолчанию MCR устанавливается в каталог C:\Program Files\MATLAB\MATLAB Component Runtime. Библиотека MCR устанавливается на машине один раз и используется всеми установленными компонентами, созданными Компилятором MATLAB;
- распаковать инсталляционный пакет с созданным программным обеспечением (MagicExample_pkg.exe) в выбранный каталог (например, в C:\approot);
- установить архив STF. Он содержит необходимые приложению функции, m-файлы и MEX-файлы. Эти файлы нужно извлечь из архива до выполнения программы. Архив STF автоматически разворачивается в первый раз, когда запускается приложение. Распаковать архив STF, не запуская приложение, можно утилитой extractSTF.exe, она находится в каталоге <matlabroot>/toolbox/compiler/deploy/win32. Эта утилита разворачивает архив в текущий рабочий каталог.
- установить пути так, чтобы система могла поддерживать программу во время выполнения. Для Windows необходимо установить следующий путь: <mcr_root>\<ver>\runtime\win32 (например: C:\MATLAB Component Runtime\v76\runtime\win32). Отметим, что на Windows XP этот каталог автоматически добавляется к путям, установка путей на других системах указана в документации Компилятора MATLAB.

Обратите внимание, что если машина развертывания имеет инсталляцию MATLAB, то для работы установленного приложения, каталоги <mcr_root> должны быть первыми на путях, а для запуска и работы MATLAB каталоги <matlabroot> должны быть первыми на путях.

Использование команды msc

Вместо графического интерфейса пользователя можно также использовать команду msc для запуска Компилятора MATLAB. Ниже перечислены типовые команды msc для создания автономного приложения или общедоступной библиотеки:

- автономное приложение из m-файла mymfunction,
`msc -m mymfunction.m`
- общедоступная библиотека C из m-файлов file1.m, file2.m и file3.m,
`msc -l file1.m file2.m file3.m`
- общедоступная библиотека C++ из m-файлов file1.m, file2.m и file3.m,
`msc -l file1.m file2.m file3.m -W cpplib -T link:lib`

Здесь пакетная опция -l заменяет последовательность опций: -W lib -T link:lib.

Команда может msc быть запущена как из командной строки MATLAB, так и DOS. В дальнейшем мы рассмотрим работу с msc более подробно.

2.1.4. Среда выполнения компоненты MATLAB, библиотека MCR

Как известно, программа, созданная на каком-либо языке, требует для своего выполнения определенный набор служб – среду выполнения. Например, программа, созданная на Java, требует для своей работы большой набор файлов, входящих в состав виртуальной машины Java. Среду выполнения программы на C++, использующей математические библиотеки MATLAB, составляют библиотеки математических процедур (набор файлов dll). Как уже упоминалось ранее, для Компилятора MATLAB версии 4.6 нет математических библиотек. Вместо них используется *среда выполнения компоненты MATLAB*, называемая MCR (MATLAB Component Runtime), которая содержит автономный набор общедоступных библиотек MATLAB и все необходимое для работы созданного Компилятором приложения или компонента (без установленного на системе MATLAB). Среда MCR обеспечивает полную поддержку всем особенностям языка MATLAB, включая Java. Среду выполнения компоненты MATLAB мы будем иногда называть, для краткости, библиотекой MCR.

Для установки среды выполнения компоненты MATLAB нужно использовать файл **MCRInstaller.exe**, который расположен в следующем каталоге `matlabroot\toolbox\compiler\deploy\win32`. При выполнении этого файла начинается обычный процесс установки Windows-приложения (никаких серийных номеров и регистрации не предполагается). Библиотеки MCR по умолчанию устанавливаются в каталог `C:\Program Files\MATLAB\MATLAB Component Runtime\v76`, где подкаталог «v76» соответствует версии 7.6 среды MCR, другая версия MCR устанавливается независимо в соседний каталог, например, «.\v74» – от MATLAB R2006b. При работе приложения используется та версия MCR, на которой был создан компонент MATLAB. Для перехода на другую версию компонент должен быть перекомпилирован.

При установке MCRInstaller автоматически:

- копирует необходимые файлы в заданный каталог;
- регистрирует библиотеки dll на системе;
- обновляет системный путь, чтобы указать на каталог dll-библиотек MCR, который есть `<target_directory>/<version>/runtime/bin/win32`.

Библиотека MCR устанавливается на машине один раз и используется всеми установленными компонентами, созданными Компилятором MATLAB. Среда исполнения MCR свободно распространяется вместе с легально созданным компонентом. Отметим, что среда выполнения MCR является достаточно большой, например, для MATLAB R2007a, она занимает около 270 Мб на диске и содержит 9931 файл в 270 каталогах. Рекомендуется посмотреть каталоги среды выполнения MCR. Отметим, что все m-файлы, входящие в MCR, представлены в зашифрованном виде.

2.1.5. Файлы, создаваемые Компилятором

Укажем файлы, создаваемые в течение процесса создания приложения и библиотеки.

Автономная выполняемая программа. В приведенном ниже примере Компилятор из m-файла `magicsquare.m` создает следующие файлы автономного приложения:

- **MagicExample.exe** – основной исполняемый файл приложения;
- **MagicExample.ctf** – архив CTF. Этот файл содержит сжатый и зашифрованный архив m-файлов, которые составляют приложение (`magicsquare.m`). Он также содержит другие файлы, от которых зависят основные m-файлы, и еще ряд файлов, необходимые во время выполнения;
- **MagicExample_mcc_component_data.c** – C-файл, содержащий данные, необходимые MCR для инициализации и использования приложения. Эти данные включают ключи шифрования, информацию о путях и другую информацию о создаваемом компоненте для MCR;
- **MagicExample_main.c** – файл обертки C, содержит функцию `main`. Он не содержит полного C-кода приложения. Файл обертки содержит интерфейсы функций, описание процедур инициализации среды MCR MATLAB, обращений к файлу данных `MagicExample_mcc_component_data.c`;
- **readme.txt** – содержит необходимую информацию для инсталляции приложения;
- **build.log, mccExcludedFiles.log** – log-файлы отчетов, в которых записывается информация о процессе компиляции.

В инсталляционный пакет входят файлы: `MagicExample.exe`, `MagicExample.ctf` и `readme.txt`.

Библиотека совместного использования C/C++. Напомним, что динамически компануемая библиотека (Dynamic Link Library) – это участок кода, хранимый в файле с расширением `.dll`. Код может быть использован другими программами, но сама по себе библиотека программой не является. Предположим, что создается C библиотека `libfoo.dll`, содержащая функции, определенные в m-файлах `foo.m` и `bar.m`. Тогда Компилятор создает следующие файлы.

- **libfoo.dll** – файл общедоступной библиотеки, это набор скомпилированных функций, участок кода, хранимый в файле с расширением `.dll`;
- **libfoo.ctf** – архив CTF. Этот файл содержит сжатый и зашифрованный архив m-файлов, которые составляют библиотеку (`foo.m` и `bar.m`) и m-файлы, от которых зависят основные m-файлы, а также еще ряд файлов, необходимые во время выполнения библиотеки;
- **libfoo_mcc_component_data.c** – C-файл, содержащий данные, необходимые MCR для инициализации и использования библиотеки. Эти данные включают ключи шифрования, информацию о путях и другую информацию для MCR о создаваемом компоненте;

- **libfoo.h** – заголовочный файл обертки библиотеки, содержит объявления функций создаваемой библиотеки и функций инициализации среды MCR;
- **libfoo.c** – С-файл библиотечной обертки. Файл обертки содержит интерфейсы функций, описание процедур инициализации среды MCR MATLAB, процедур вызова функций создаваемой библиотеки;
- **libfoo.exports** – файл экспорта, содержит список экспортируемых функций библиотеки;
- **libfoo.lib** – библиотека импорта, содержит описание функций библиотеки dll. Этот файл позволяет компоновщику связать вызовы в приложении с адресами в DLL;
- **readme.txt** – файл одержащий необходимую информацию для инсталляции приложения;
- **build.log, mccExcludedFiles.log** – файлы отчетов, в которых записывается информация о процессе компиляции.

В инсталляционный пакет входят файлы: libfoo.dll, libfoo.ctf, libfoo.h, libfoo.exports, libfoo.lib и readme.txt.

Технологический файл компоненты (CTF)

Компилятор MATLAB генерирует *технологический файл компоненты* (CTF-файл, Component Technology File), который является независимым от типа конечного продукта – автономного приложения или библиотеки – но определенный для каждой операционной системы. Этот архивный файл содержит зашифрованные m-функции, которые составляют приложение и все другие m-функции MATLAB, от которых зависят основные m-функции. Архив содержит все основанное на MATLAB содержание (m-файлы, MEX-файлы, и т. д.) связанное с компонентом. Все m-файлы зашифрованы в архиве CTF используя расширенный стандарт кодирования (AES). Каждый произведенный компонент имеет свой архив CTF. Различные архивы CTF, типа COM, .NET или компонентов Excel, могут сосуществовать в одном пользовательском приложении, но нельзя смешивать и каким-либо образом менять их содержание. Все m-файлы из данного CTF-архива блокированы вместе уникальным криптографическим ключом. М-файлы с различными ключами не будут выполняться если они помещены в тот же самый архив CTF.

Архив CTF автоматически распаковывается при первом запуске программы, или вызове функции библиотеки. Чтобы распаковать архив CTF не выполняя приложение, можно использовать утилиту extractCTF.exe, которая находится в каталоге <matlabroot>/toolbox/compiler/deploy/win32. Эта утилита распаковывает архив CTF в каталог, в котором архив находится постоянно. Например, команда `extractCTF MagicExample.ctf` распаковывает MagicExample.ctf в каталог D:\R2007a\work\MagicExample\MagicExample\src.

Файлы обертки

Для создания бинарного файла компоненты Компилятор MATLAB генерирует один или несколько файлов обертки. Файлы обертки обеспечивают интерфейсы для скомпилированного М-кода. Обертка выполняет следующее:

- выполняет инициализацию и завершение как требуется определенным интерфейсом;
- определяет массивы данных, содержащие информацию о путях, ключи шифрования, и другую информацию, требующуюся MCR;
- обеспечивает необходимый код, чтобы отправить вызовы от функций интерфейса к функциям MATLAB в MCR;
- для приложения, содержит функцию `main`;
- для библиотеки, содержит точки входа для каждой m-функции.

2.2. Процесс создания компонента MATLAB®

Продукт, создаваемый Компилятором MATLAB называется компонентом MATLAB. Практическое создание компонента описано в первом параграфе. В этом параграфе рассмотрим общую схему работы Компилятора MATLAB.

2.2.1. Процесс создания компонента

Процесс создания программных компонентов с Компилятором MATLAB является полностью автоматическим. Например, чтобы создать автономное приложение MATLAB, достаточно представить список m-файлов, которые составляют приложение. Тогда Компилятор выполняет следующие операции:

- анализ зависимости m-файлов;
- генерацию объектного кода;
- создание архива;
- компиляцию;
- компоновку.

Следующий рисунок 2.2.1 иллюстрирует, как Компилятор использует две m-функции с именами `foo.m` и `bar.m` и создает автономную выполняемую программу `foo.exe`.

Рассмотрим подробнее основные этапы работы Компилятора.

Анализ зависимости. На первом шаге определяются все функции, от которых зависят представленные m-файлы, MEX-файлы, и P-файлы. Этот список включает все m-файлы, вызываемые данными файлами а также файлами, которые они вызывают, и так далее. Также включаются все встроенные функции и объекты MATLAB.

Создание кода обертки. На этом шаге создается весь исходный текст, необходимый для создания компоненты, включая:

- код интерфейса C/C++ для m-функций, указанных в командной строке `mcc`. Для библиотек и компонентов, этот файл (`foo_main.c`) включает интерфейсы всех созданных функций;
- файл данных компоненты, который содержит информацию необходимую для выполнения m-кода во время работы. Эти данные включают информа-

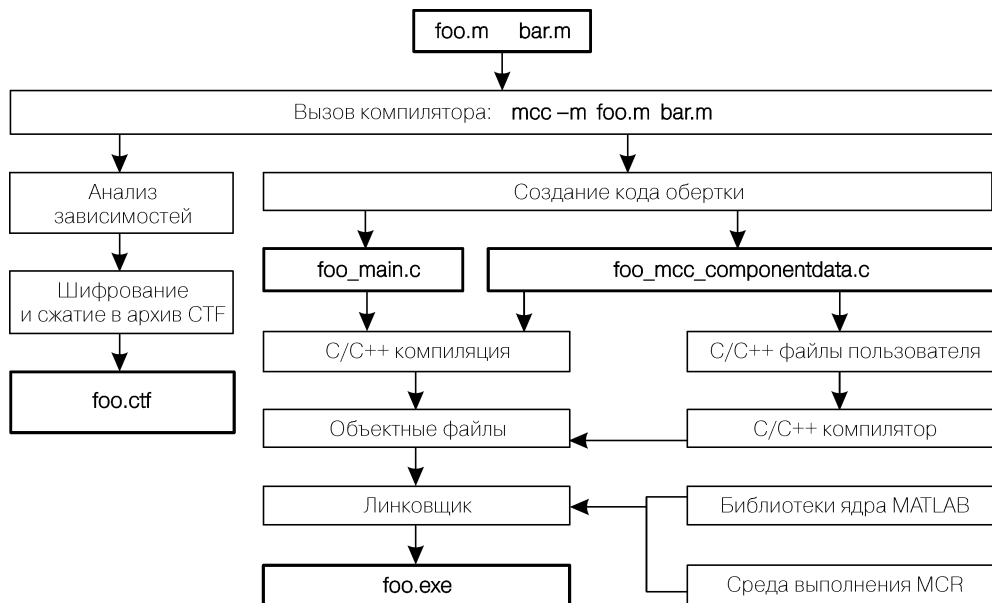


Рис. 2.2.1. Создание автономного приложения

цию о путях и ключи шифрования, необходимые для загрузки m-кода, сохраненного в архиве STF компонента.

Создание архива STF. Список файлов MATLAB (m-файлы и MEX-файлы) созданный в течение анализа зависимости используется для создания архива STF, который содержит файлы, необходимые компоненте во время ее выполнения. Включена также информация о каталогах. Файлы зашифрованы и сжаты в отдельный файл.

С/С++ компиляция. Этот шаг компилирует созданные C/C++ файлы из кода обертки в объектный код.

Соединение (линковка). Заключительный шаг связывает созданные объектные файлы с необходимыми библиотеками MATLAB, чтобы создать законченный компонент. Этапы C/C++ компиляции и линковки используют утилиту `mbuild` Компилятора MATLAB.

2.2.2. Управление путями при компиляции

Компиляция приложения занимает достаточно большое время. Это время работы по анализу зависимости для определения списка необходимых файлов для включения в пакет STF. В некоторых случаях, этот процесс затрагивает слишком большое число файлов, например, когда в компиляцию включены классы объектно-ориентированного программирования MATLAB, и невозможно решить проблемы

с перегруженными методами во времени компиляции. Анализ зависимости – это итерационный процесс, который обрабатывает информацию о включении или исключении файлов при каждом проходе. Следовательно, этот процесс может привести к очень большому архиву STF и, следовательно, к большому времени компиляции для относительно маленьких приложений.

Наиболее эффективный способ уменьшать число файлов состоит в использовании функции `depfun` для ограничения путей MATLAB. Данная функция

```
list = depfun('fun'); или  
[list, builtins, classes] = depfun('fun');
```

позволяет найти список `list` путей к всем файлам (в виде массива ячеек строк), от которых зависит данная функция `fun.m` (`builtins` – список встроенных функций, `classes` – список классов). Найденные пути файлов можно прямо указать Компилятору. В настоящей версии есть три способа взаимодействовать с путями компиляции:

- команды `addpath` и `rmpath` в MATLAB;
- опция `-I <directory>` – передача каталогов в командную строку `mcc`;
- опции `-N` и `-p` – очистка путей и добавление каталогов в командную строку `mcc`.

Если Компилятор запускается из командной строки MATLAB, то можно использовать команды `addpath` и `rmpath`, чтобы изменить пути MATLAB перед выполнением компиляции, например, выполнить команду `addpath('list{:}')`. Следует учитывать два недостатка:

- путь изменяется только для текущей сессии MATLAB;
- если Компилятор выполняется вне MATLAB, это не сработает, если в MATLAB выполнена команда `savepath`.

Можно также использовать опцию `-I`, чтобы добавить каталог к началу пути текущей компиляции. Эта особенность полезна, когда компилируются файлы, которые находятся в каталогах не на путях MATLAB.

Есть две новые опции Компилятора, которые обеспечивают более детальное управление путями. Они действуют подобно «фильтру», применяемому к путям MATLAB для данной трансляции.

Первая новая опция есть `-N`. Передача `-N` в командную строку `mcc` эффективно очищает пути всех каталогов, кроме следующих основных каталогов (этот список может меняться):

```
<matlabroot>/toolbox/matlab  
<matlabroot>/toolbox/local  
<matlabroot>/toolbox/compiler/deploy  
<matlabroot>/toolbox/compiler
```

При этом сохраняются все подкаталоги вышеупомянутого списка, которые появляются на пути MATLAB во времени компиляции. Включение `-N` в командную строку также позволяет заменять каталоги первоначального пути, при сохранении относительного упорядочения включенных каталогов. Все подкаталоги включенных каталогов, которые появляются на первоначальном пути, также включены.

Опция `-p` используется, чтобы добавить каталог к пути трансляции в соответствии с порядком, в котором они находятся на пути MATLAB. Синтаксис:

```
p <directory>
```

где `<directory>` является каталогом, который будет включен. Если `<directory>` – не абсолютный путь, считается, что он находится в текущем рабочем каталоге. Отметим, что опция `-p` требует опции `-N` в командной строке `mcc`.

Пропущенные функции. Когда Компилятор создает автономное приложение, он компилирует `m`-файлы, указанные в командной строке и, кроме того, он компилирует любые другие `m`-файлы, которые данные `m`-файлы вызывают. Компилятор использует анализ зависимости, который определяет все функции, от которых зависит указанный `m`-файл: `m`-файлы, `MEX`-файлы и `P`-файлы. Анализ зависимости не может определить местонахождение функции, если единственное место, где вызывается функция в вашем `m`-файле, есть обращение к функции в строке повторного вызова, или в строке, которую передают как аргумент к функции `feval` или решателю `ODE`. Компилятор не ищет в этих текстовых строках имен функций для компиляции. Чтобы устранить возможную ошибку, создайте список всех функций, которые определены только в строках повторного вызова и передайте эти функции, используя псевдокомментарий `%#function`. Вместо того чтобы использовать псевдокомментарий `%#function`, можно задать имя отсутствующего `m`-файла в командной строке Компилятора, используя опцию `-a`.

Для нахождения функций в приложении, которые должны быть перечислены в псевдокомментарии `%#function`, нужно искать в исходном тексте `m`-файла текстовые строки, указанные как строки повторного вызова или как аргументы функций `feval`, `fminbnd`, `fminsearch`, `funm`, `fzero` или любые решатели `ODE`. Чтобы найти текстовые строки, используемые как строки повторного вызова, ищите символы `"Callback"` или `"fcn"` в вашем `m`-файле.

2.3. Работа с `mcc` и `mbuild`

Среда разработки Deployment Tool позволяет легко создавать необходимые компоненты MATLAB. Однако иногда нужны более широкие возможности управления процессом построения автономных приложений и библиотек. В этом случае можно использовать две утилиты, `mcc` и `mbuild`, с их опциями. Первая утилита, `mcc`, является автономной и может быть вызвана как из командной строки MATLAB, так и из командной строки DOS. В первом случае используется `mcc.m`, а во втором – `mcc.exe`. Программа `mcc` создает все вспомогательные `C`-файлы из `m`-файлов и вызывает `mbuild`. Утилита `mbuild` (`bat`-файл) вызывает внешний компилятор для компиляции `C/C++`-кода, использующего в своей работе функции, определенные в `m`-файлах, либо функции, созданные Компилятором MATLAB, общедоступной библиотеки. Приведем схему работы `mcc` и `mbuild` по созданию автономного приложения из `m`-файлов (`mrank.m`, `printmatrix.m`) и кода `C` (`mrankp.c`), рис. 2.3.1.

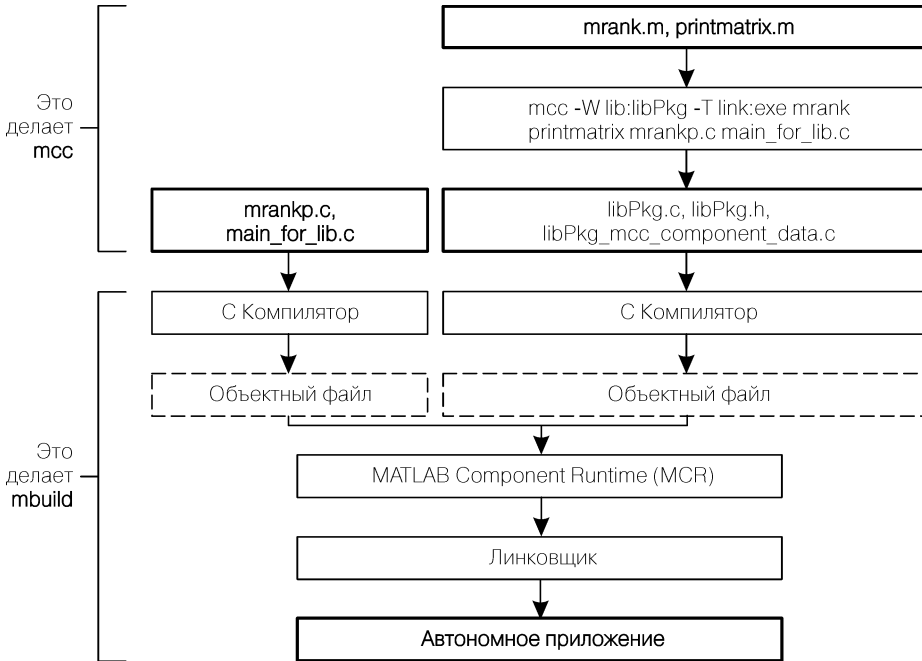


Рис. 2.3.1. Схема работы mcc и mbuild

2.3.1. Работа с mcc

Команда *mcc* создает вспомогательные С-файлы и вызывает Компилятор MATLAB. Синтаксис mcc:

```
mcc [-options] mfile1 [mfile2 ... mfileN] [C/C++file1 ... C/C++fileN]
```

Если в командной строке задано более одного *m*-файла, Компилятор генерирует С или С++ функцию для каждого *m*-файла. Если в командную строку *mcc* включаются имена С или С++ файлов, то эти файлы передаются непосредственно к *mbuild*, вместе с любым С или С++ файлом, созданным Компилятором. При использовании *mcc* рекомендуется указывать расширение файла.

Обычное использование mcc

Покажем это на примерах.

Пример 1. Создание автономного приложения из функций *myfun1.m* и *myfun2.m*,

```
mcc -m myfun1 myfun2
```

Пример 2. Создание автономного приложения для *myfun.m*. Функция *myfun.m* берется из каталога */files/source*, созданные С файлы и выполняемая программы помещается в каталог */files/target*,

```
mcc -m -I /files/source -d /files/target myfun
```

Пример 3. Создание общедоступной динамически подключаемой С-библиотеки с именем `libmatrix` из `m`-функций `addmatrix.m` `multiplymatrix.m`,

```
mcc -W lib:libmatrix -T link:lib addmatrix.m multiplymatrix.m
```

С++ библиотека,

```
mcc -W cpplib:libmatrix -T link:lib addmatrix.m multiplymatrix.m
```

здесь опция `-W` определяет тип обертки, а опция `-T link` определяет тип результата.

Пример 4. Создание приложения `mrnk.exe` из С-кода `mrnk.c` и `main_for_lib.c`, использующего функции из `m`-файлов `mrnk.m` и `printmatrix.m`,

```
mcc -W lib:libPkg -T link:exe mrnk printmatrix mrnk.c  
main_for_lib.c
```

Опции *mcc*

Рассмотрим некоторые опции Компилятора *mcc* подробнее.

Опция –а. Добавление файла к архиву СТЕ. Используется в виде

```
-a filename
```

Для определения файла, который будет непосредственно добавлен к архиву СТЕ, разрешаются кратные опции `-a`. Компилятор ищет эти файлы на путях MATLAB, полное имя пути является дополнительным. Эти файлы не передаются `mbuild`, так что можно включить файлы типа файлов данных.

Опция –b. Создает файл Visual Basic (`.bas`) содержащий интерфейс функции Microsoft Excel для созданного Компилятором компонента. Требуется MATLAB Builder for Excel.

Опция –В. Использование файла группы. Замена файла в командной строке *mcc* на содержание указанного файла. Используется в виде

```
-B filename[:<a1>,<a2>,...,<an>]
```

Файл группы `filename` должен содержать только опции командной строки *mcc* и соответствующие параметры и/или другие имена файлов. Файл может содержать другие опции `-В`. Файл группы может включить параметры замены для опций Компилятора, которые принимают имена и номера версии.

Опция –с. Создание только кода С. Когда используется с макро опцией, то создается С код, но не вызывается `mbuild`, то есть, не создается автономное приложение. Эта опция эквивалентна `-T codegen` помещенной в конце командной строки *mcc*.

Опции –g и –G. Создание информации об отладке. Включение информации отладки для обертки.

Опция –I. Добавляет путь нового каталога в список включенных каталогов. Каждая опция `-I` добавляет каталог в конец текущего пути поиска файлов. Например,

```
-I <directory1> -I <directory2>
```

Устанавливает путь поиска файлов так, что сначала ищутся `m`-файлы из `directory1`, затем из `directory2`. Эта опция важна для автономной компиляции, когда нет путей MATLAB.

Опция -I. Макрокоманда для создания библиотеки функций. Эта опция создает обертку функций библиотеки для каждого m-файла из командной строки и вызывает C компилятор, чтобы образовать общедоступную библиотеку, которая экспортирует эти функции. Название библиотеки берется из названия первого m-файла в командной строке. Эта макрокоманда эквивалентна

```
-W lib -T link:lib
```

Опция -M. Прямая передача опций во время компиляции. Используется в виде

```
-M string
```

Для передачи строки `string` непосредственно к скрипту `mbuild`. Это обеспечивает полезный механизм для определения опций во время компиляции, например, `-M "-Dmacro=value"`. Кратные опции `-M` не используются, учитывается только последняя позиция `-M`.

Опция -N. Очистка путей. Передача опции `-N` эффективно очищает пути всех каталогов кроме следующих основных каталогов (этот список может измениться):

```
<matlabroot>/toolbox/matlab  
<matlabroot>/toolbox/local  
<matlabroot>/toolbox/compiler/deploy  
<matlabroot>/toolbox/compiler
```

Также сохраняются все подкаталоги вышеупомянутого списка. Это позволяет заменить каталоги первоначального пути.

Опция -R. Опция `-R` используется, чтобы отменить опции MCR во время выполнения программы. Используется синтаксис

```
-R option
```

для обеспечения любой из этих опций во время выполнения. Возможные значения `option`:

- `-nojvm` – не использовать виртуальную машину Java (JVM);
- `-nojit` – не использовать MATLAB JIT (создание бинарного кода используется для ускорения выполнения m-файлов).

Опция `-R` доступна только для автономных приложений. Чтобы заменить опции MCR в других продуктах Компилятора MATLAB, используются функции `mclInitializeApplication` и `mclTerminateApplication`.

Примеры правильного использования опции `-R`.

```
mcc -m -R "-nojvm -nojit" -v foo.m  
mcc -m -R "-nojvm" -v -R "-nojit" foo.m  
mcc -m -R -nojvm -R -nojit foo.m  
mcc -m -R -nojvm -v foo.m  
mcc -m -R -nojvm -R -nojit foo.m
```

Пример неправильного использования.

```
mcc -m -R -nojvm -nojit foo.m
```

Опция -T. Определение типа конечного продукта. Используется синтаксис

```
-T target
```

для определения типа вывода. Допустимые значения `target` следующие:

- `codegen` – создает C/C++ файл обертки. Значение по умолчанию есть `codegen`;
- `compile:exe` – то же самое, что и `codegen`, плюс компилирует C/C++ файлы в объектную форму, подходящую для линковки в автономную выполняемую программу;
- `compile:lib` – то же самое, что и `codegen`, плюс компилирует C/C++ файлы в объектную форму, подходящую для линковки в общедоступную библиотеку/DLL;
- `link:exe` – то же самое, что и `compile:exe` плюс линкует объектные файлы в автономную выполняемую программу;
- `link:lib` – то же самое как `compile:lib` плюс линкует объектные файлы в общедоступную библиотеку/DLL.

Опция –v. Подробности. Отображает шаги трансляции, включая номер версии Компилятора, имена исходных файлов после их создания, имена созданных выходных файлов, обращения к `mbuild`

Опция –w. Предупреждающие сообщения. Предупреждающие сообщения дисплеев. Используется в виде

`-w option[:<msg>]`

для управления выводом предупреждений. Допустимые значения:

- `-w list` – создает таблицу, которая отображает значения строк `<string>` предупреждающих сообщений для использования с `enable`, `disable` и `error`. Список сообщений ошибок и предупреждений;
- `-w enable` – допускает все предупреждения;
- `-w disable[:<string>]` – отключает определенное предупреждение, связанное с `<string>`. См. список сообщений ошибок и предупреждений для допустимых значений `<string>`. Отсутствие дополнительной строки `:<string>` применяет действие `disable` ко всем предупреждениям;
- `-w enable[:<string>]` – допускает предупреждение, определенное строкой `<string>`. См. список сообщений ошибок и предупреждений для допустимых значений `<string>`. Отсутствие дополнительной строки `:<string>` применяет действие `enable` ко всем предупреждениям;
- `-w error[:<string>]` – обрабатывает определенное предупреждение, связанное с `<string>` как ошибку. Отсутствие дополнительной строки `:<string>` применяет действие `error` ко всем предупреждениям.

Опция –W. Обертка функции. Управляет созданием оберток функций. Используется в виде

`-W type`

Для управления созданием оберток функций для набора созданных компилятором `m`-файлов. Дается список функций, а Компилятор генерирует функции обертки и любые соответствующие определения глобальных переменных. Допустимые опции:

- `main` – производит POSIX-функцию `main()`;
- `lib:<string>` – производит функции инициализации и завершения для использования при компилировании этого созданного компилятором кода

в большее приложение. Эта опция также производит заголовочный файл, содержащий прототипы для всех общих функций во всех указанных m-файлах. Строка <string> становится базовым названием для созданного файла C/C++ и заголовочного файла. Создает файл .exports, который содержит все нестатические имена функций;

- `com:<component_name>, <class_name>, <version>` – производит COM-объект из m-файла;
- `none` – не производит файл обертки. Значение по умолчанию есть `none`.

Опция -z. Определение пути. Определение пути для библиотеки и для включаемых файлов. Используется в виде

`-z path`

задание `path` для использования библиотек компилятора и для включения файлов, вместо пути, возвращаемого `matlabroot`.

Порядок использования опций

Для настройки процесса компиляции используется одна или более опций `mcc`. Применение опции отмечается одной чертой перед символом опции. В случае нескольких опций их можно группировать с единственной чертой (-), например,

```
mcc -m -g myfun или mcc -mg myfun
```

Опции с параметрами не могут объединяться кроме случая, когда опция с параметрами стоит последней в списке. Например, эти форматы правильные,

```
mcc -v -W main -T link:exe myfun
mcc -vW main -T link:exe myfun
```

Следующая запись ошибочна.

```
mcc -Wv main -T link:exe myfun
```

Если используются противоречивые опции, то Компилятор разрешает их слева направо, т.е. последняя опция имеет приоритет. Рассмотрим, например, две эквивалентные макро опции:

```
mcc -m -W none test.m
mcc -W main -T link:exe -W none test.m
```

В этом примере, есть два конфликта опции `-w`. После обработки слева направо, Компилятор решает, что последняя опция `-W none` имеет приоритет и Компилятор не создает обертку.

Установка опций по умолчанию, файл `mccstartup`. Если имеется набор опций командной строки, которые необходимо всегда передавать `mcc`, то их можно записать в файл `mccstartup`. Это обычный текстовый файл, содержащий необходимые опции командной строки. Размещается этот файл в один из двух каталогов:

- текущий рабочий каталог;
- пользовательский каталог настроек `C:\Documents and Settings\UserName\Application Data\MathWorks\MATLAB\R2007a`

Во время работы mcs ищет файл mccstartup в этих двух каталогах в указанном выше порядке. Если mcs находит файл mccstartup, то читает и обрабатывает опции, записанные в файле так, как будто они появились на командной строке mcs перед любыми фактическими опциями командной строки. Файл mccstartup и опция -v обрабатываются одним и тем же самым способом.

Использование макросов. Макросы, соответствующие опциям Компилятора упрощают наиболее общие задачи компиляции. Вместо того, чтобы вручную группировать несколько опций вместе для задания определенного типа компиляции, можно использовать простую макро-опцию. Макрос группирует несколько опций вместе, чтобы выполнить определенный тип трансляции. Макро опции представлены в табл. 2.3.1.

Таблица 2.3.1. Макро опции mcs

Макроопция	Файл группы	Создается	Эквивалентные опции
-l	macro_option_l	Библиотека	-W lib -T link:lib
-m	macro_option_m	Автономное С приложение	-W main -T link:exe

Опции, которые составляют, например, макрокоманду -m:

- -W main – создание файла обертки для автономного приложения;
- -T link:exe – создание выполнимой программы.

Можно изменить значение макро опции, редактируя соответствующий файл группы macro_option, в каталоге <matlabroot>/toolbox/compiler/bundles. Например, чтобы изменить макрокоманду -m, следует редактировать файл macro_option_m.

Использование имен путей. Если определяется полное имя пути к m-файлу в командной строке mcs, то Компилятор MATLAB делит полное имя на две части: соответствующее имя пути и имена файлов (<path> и <file>) и заменяет полное имя пути в списке параметров на "-I <path> <file>". Например,

```
mcs -m /home/user/myfile.m
```

будет обработана как

```
mcs -m -I /home/user myfile.m
```

Иногда это поведение может привести к недоразумению, если имеются файлы с одинаковым именем в разных каталогах. Можно определить опцию -v и посмотреть, какой m-файл анализирует Компилятор. Опция -v печатает полное имя пути m-файла в течение стадии анализа зависимости. Отметим, что Компилятор дает предупреждение (specified_file_mismatch) если файл с полным именем пути включен в командную строку и Компилятор находит его еще где-нибудь.

Использование файлов групп

Файлы групп дают удобный способ для объединения наборов групп опций Компилятора MATLAB и многократного вызова их при необходимости. Синтаксис опций файла групп

```
-B <filename>[:<a1>,<a2>,...,<an>]
```

Опция группы -В заменяет название файла <filename> содержанием указанного файла. Файл должен содержать только опции командной строки msc и соответствующие аргументы и/или другие имена файлов. Файл может содержать другие опции -В. Файл группы может включить параметры замены для опций Компилятора, которые принимают имена и номера версии. Например, файл группы csharedlib для С библиотеки совместного использования состоит из строки

```
-W lib:%1% -T link:lib
```

Тогда для создания С библиотеки совместного использования, можно использовать

```
msc -B csharedlib:mysharedlib myfile.m myfile2.m
```

Вообще, каждый символ %n% в файле группы будет заменен соответствующим аргументом, указанным в строке msc (в предыдущем примере заменяется %1% на mysharedlib).

Замечание. Можно использовать опцию -В замены файла содержанием как в строке DOS, так и в UNIX. Для использования -В в командной строке MATLAB, необходимо включить выражение, которое следует за -В в одинарные кавычки, когда передается больше одного параметра. Например, файл группы cexcel имеет содержание -W excel:%1%, %2%, %3% -T link:lib -b, которое предполагает три аргумента %1%, %2%, %3%. Тогда в командной строке будет следующая строка

```
msc -B 'cexcel:component,class,1.0' weekday data tic calendar to
```

Файлы групп опций можно создавать самостоятельно. В каталоге <matlab>\toolbox\compiler\bundles\ имеются набор файлов групп. В табл. 2.3.2 приведены примеры файлов групп из данного каталога.

Таблица 2.3.2. Файлы групп Компилятора

Имя файла группы	Создание	Содержание
ccom	COM объект	-W com:<component_name>, <class_name>, <version> -T link:lib
cexcel	Excel COM объект	-W excel:<component_name>, <class_name>, <version> -T link:lib -b
cprcom	COM объект	-B ccom:<component_name>, <class_name>, <version>
cprexcel	Excel COM объект	-B cexcel:<component_name>, <class_name>, <version>
cpplib	C++ библиотека	-W cpplib:<library_name> -T link:lib
csharedlib	С библиотека	-W lib:<shared_library_name> -T link:lib

Создание файлов обертки

Файлы обертки определяют связь между кодами, созданными Компилятором MATLAB, и приложением, таким как автономное приложение или библиотека, обеспечивая требуемый интерфейс, который позволяет коду работать в желатель-

ной среде выполнения. Для определения типа обертки при создании используется команда

```
-W <type>
```

где <type> может быть `main`, `lib:libname`, `cpplib:libname` и `com:<component_name>`. Рассмотрим их подробнее.

Main обертка. Опция `-W main` генерирует обертки, необходимые для создания автономного приложения. Эти POSIX `main` обертки принимают строки от оболочки POSIX и возвращают код состояния. Они передают выражения командной строки к `m`-функциям как строки MATLAB.

Замечание. POSIX – Portable Operating System Interface for computer environments, интерфейс переносимой операционной системы (набор стандартов IEEE, описывающих интерфейсы ОС для ЮНИКС).

Обертка библиотеки C. Опция `-l`, или ее эквивалент `-W lib:libname`, создает файл обертки библиотеки C. Эта опция создает общедоступную библиотеку из произвольного набора `m`-файлов. Созданный заголовочный файл содержит объявление функции C для каждой откомпилированной `m`-функции. Экспортный список содержит набор имен функций, которые экспортируются из библиотеки совместного использования C.

Обертка библиотеки C++. Опция `-W cpplib:libname` создает файл обертки библиотеки C++. Созданный заголовочный файл содержит все точки входа для всех откомпилированных `m`-функций.

2.3.2. Использование псевдокомментариев

В `m`-кодах можно использовать два псевдокомментария: `%#external` – для вызова произвольной C/C++ функции из `M`-кода и `%#function` – для включения дополнительных `m`-функций.

Функция `%#external`. Псевдокомментарий для вызова произвольной C/C++ функции из `m`-кода. Он пишется в тексте `m`-функции. *Псевдокомментарий `%#external`* сообщает Компилятору MATLAB что используемая функция будет написанной вручную, а не созданной из `m`-кода. Этот псевдокомментарий затрагивает только ту функцию, в которой он появляется, любая `m`-функция может содержать этот псевдокомментарий. При использовании этого псевдокомментария Компилятор генерирует дополнительный заголовочный файл, называемый `fcn_external.h`, где `fcn` является именем исходной `m`-функции, содержащей псевдокомментарий `%#external`. Этот заголовочный файл будет содержать объявление `extern` функции, которую пользователь должен затем обеспечить. Эта функция должна соответствовать тому же интерфейсу, что и созданный компилятором код. Созданный компилятором файл C или C++ будет включать этот заголовочный файл `fcn_external.h` для объявления данной функции. При компиляции программы, которая содержит псевдокомментарии `%#external`, необходимо явно передать в командную строку `mcc` каждый файл, который содержит этот псевдокомментарий.

Функция `%#function`. Псевдокомментарий

```
%#function <function_name-list>
```

записывается в коде m-функции и сообщает Компилятору MATLAB, что указанная в списке m-функция должна быть включена в компиляцию, независимо от того, обнаруживает ли ее анализ зависимости Компилятора MATLAB. Например, в следующем примере

```
function foo
%#function bar foobar
feval('bar');
feval('foobar');
end %#function foo
```

функции (bar и foobar) включены в компиляцию только потому, что они включены в строку *псевдокомментария* `%#function`. Иначе Компилятор бы их не обнаружил в виде символьных переменных в функции feval. Таким образом, без этого псевдокомментария, анализ зависимости Компилятора MATLAB может не найти все функции и скомпилировать все m-файлы, используемые в вашем приложении. Этот псевдокомментарий добавляет функцию верхнего уровня так же как все подфункции в файле компиляции. Отметим, что нельзя использовать псевдокомментарий `%#function`, чтобы обратиться к функциям, которые не доступны в виде m-кода. Рекомендуется использовать `%#function` в коде везде, где используются инструкции feval.

2.3.3. Несколько полезных замечаний

1. Использование встроенных функций. Компилятор MATLAB не может компилировать встроенные функции MATLAB. Поэтому нужно написать m-функцию, которая вызывает встроенную функцию MATLAB и ее использовать для компиляции. Например, нет возможности компилировать встроенную функцию magic, но можно компилировать следующую m-функцию:

```
function m = magicssquare(n)
if (ischar(n))
    n=str2num(n);
end
m = magic(n);
```

2. Вызов функции из системной командной строки. Функция MATLAB компилируется в консольное автономное приложение. Оно вызывается из командной строки обычным образом. Все параметры, которые передаются к такой функции из системной командной строки являются символьными. Если функция ожидает числовой ввод, то нужно это учесть при написании m-функции и включить функцию преобразования строки в число, как в вышеприведенном примере. Если функция ожидает ввод символа, то ничего делать не надо.

3. Использование MAT-файлов. При необходимости использовать MAT-файл в приложении можно использовать опцию `-a` Компилятора MATLAB для

включения такого файла в архив СТЕ. Кроме того, имеется набор С-функций для работы с МАТ-файлами, см. раздел 2.6 и документацию по использованию mх-процедур «MATLAB/External Interfaces» из общего раздела документации MATLAB.

4. Запуск программы не из каталога приложения. Если программа запускается из другого каталога с указанием полного пути, то она может не работать. Дело в том, что для ее работы нужен архив СТЕ. Нужно либо запускать программу из основного каталога автономного приложения, либо задать системные пути для архива СТЕ.

5. Ввод и вывод переменных. Для передачи входных параметров к созданному Компилятором MATLAB автономному приложению используются те же правила, что и для любого другого консольного приложения. Например, чтобы передать файл по имени helpfile к скомпилированной функции названной filename, используется команда

```
filename helpfile
```

Для передачи чисел или символов (например, 1, 2, и 3), используется

```
filename 1 2 3
```

Параметры не разделяются запятыми. Для передачи матрицы на ввод, используется

```
filename "[1 2 3]" "[4 5 6]"
```

Возвратить значения от скомпилированной программы можно двумя способами: либо отобразить их на экране, либо сохранить в файле. Для отображения полученных данных на экране, можно поступить как в MATLAB – не ставить точку с запятой в конце команды, результаты которой нужно получить на экране, либо использовать команду disp. Можно также переадресовать выводы другим приложениям, используя переназначение вывода (оператор >).

2.3.4. Функция mbuild

Утилита *mbuild* является bat-файлом из каталога C:\R2007a\bin\. Она работает как из командной строки DOS (UNIX), так и из MATLAB. Вызывает внешний компилятор для компилирования и линковки исходных файлов в автономное приложение или общедоступную библиотеку

Синтаксис

```
mbuild [option1 ... optionN] sourcefile1 [... sourcefileN]  
[objectfile1 ... objectfileN] [libraryfile ... libraryfileN]  
[exportfile1 ... exportfileN]
```

Обратите внимание, что поддерживаются следующие типы исходных файлов: .c, .cpp, .idl, .rc.

Пример использования. Создание автономного консольного приложения matrixdriver.exe из С-кода matrixdriver.c и библиотеки libmatrix.dll:

```
mbuild matrixdriver.c libmatrix.lib (на Windows)
```

Предполагается, что файлы `libmatrix.dll`, `libmatrix.h`, `libmatrix.lib` и файл `matrixdriver.c` помещены в один каталог, из которого и выполняется данная команда. Подробнее об этом примере см. в следующем параграфе.

Утилита `mbuild` поддерживает различные опции, позволяющие настраивать создание и линковку кода. Опции `mbuild` можно найти в документации MATLAB, или посмотреть в книге [ППС].

2.4. Примеры создания автономных приложений и библиотек

В этом параграфе рассмотрим более подробно использование Компилятора MATLAB для создания автономных приложений и библиотек. Автономное приложение C может быть создано или полностью из `m`-файлов или некоторой комбинации `m`-файлов, `MEX`-файлов и файлов исходного текста на языке C или C++. Будет показано, как написать программу на C или C++, которая может вызывать функции из созданной Компилятором библиотеки. Поскольку создание автономного приложения уже рассматривалось в первом параграфе, обсудим сначала создание библиотек.

2.4.1. Библиотеки совместного использования

Покажем, как создать библиотеку C или C++ совместного использования и как написать C/C++ код, который вызывает функции библиотеки.

Библиотека совместного использования C

Напомним, что динамически компокуемая библиотека (Dynamic Link Library) – это участок кода, хранимый в файле с расширением `.dll`. Код может быть использован другими программами, но сама по себе библиотека программой не является. DLL может содержать код в двух основных формах. Первая форма – это отдельные функции, которые можно вызвать из главного приложения. Вторая форма кода – это классы C++. Вместо доступа к отдельным функциям, можно создать экземпляр класса и вызывать функции-члены этого класса. Мы будем использовать библиотеки функций. Библиотека DLL содержит экспортируемые функции. Описание этих функций находится в специальном файле, называемом файлом библиотеки импорта (`import library file`). Файл библиотеки импорта имеет имя, совпадающее с именем DLL и с расширением `.lib`. Этот файл позволяет компоновщику связать вызовы в приложении с адресами в DLL. При статической загрузке, DLL автоматически загружается при запуске использующего ее приложения. Для использования статической загрузки нужно подключить к проекту `lib`-файл на этапе компоновки. Добавление `lib`-файла к проекту легко делается через менеджер проектов. Динамическая загрузка означает, что DLL загружается по мере необходимости и выгружается по окончании ее использования.

Файл библиотеки импорта создается утилитой `implib` в строке DOS:

```
C:\CBuilder\bin\implib MyDll.lib MyDll.dll
```

При создании библиотеки DLL Компилятором MATLAB, создается и DLL, и остальные файлы: файл обертки, заголовочный файл, файл библиотеки импорта и список экспорта. При создании приложения Компилятором MATLAB, использующего DLL, подключение `lib`-файла не требуется, но нужно, чтобы он был в одном каталоге с приложением.

Рассмотрим пример создания C библиотеки совместного использования из нескольких `m`-файлов и пример программы на C создания автономного приложения, которое вызывает функции общедоступной библиотеки. Выберем рабочий каталог для библиотеки `C:\R2007a\work\Lib_matrix_C`. Для начала работы нужно скопировать следующие файлы примеров из `<matlabroot>/extern/examples/compiler` в рабочий каталог: `addmatrix.m`, `multiplymatrix.m`, `eigmatrix.m` и `matrixdriver.c`. М-файлы содержат процедуры сложения матриц, матричного умножения и нахождения собственных чисел матрицы. Последний файл `matrixdriver.c` содержит функцию `main` автономного приложения, которая вызывает функции библиотеки. Сначала создадим библиотеку, а затем – приложение.

Создание библиотеки. Процедура создания библиотеки была уже рассмотрена в первом параграфе. Для создания библиотеки мы используем среду разработки `Deployment Tool`, которая вызывается командой:

```
deploytool
```

Выберем создание библиотеки C и назовем проект именем `libmatrix`, поскольку такое имя библиотеки используется далее в коде приложения `matrixdriver.c`. Тогда файлы компонента будут в двух подкаталогах **`distrib`** и **`src`** каталога **`libmatrix`**.

Можно также исполнить команду

```
mcc -B csharedlib:libmatrix addmatrix.m multiplymatrix.m  
eigmatrix.m -v
```

из командной строки MATLAB. Здесь опция `-B csharedlib` есть группа опций, которая расширяется в

```
-W lib:<libname> -T link:lib
```

Опция `-W lib:<libname>` указывает Компилятору MATLAB создать функцию обертки для общедоступной библиотеки и назвать ее `libname`. Опция `-T link:lib` определяет конечный вывод как общедоступную библиотеку. При этом создаются все необходимые файлы библиотеки: `libmatrix.c`, `libmatrix.h`, `libmatrix_mcc_component_data.c`, `libmatrix.ctf`, `libmatrix.exports`, `libmatrix.lib`, `libmatrix.dll`, `libmatrix.prj` и `readme.txt`.

Создание приложения. Для приложения зададим подкаталог **`matrixdriver`** в рабочем каталоге. Поместим файлы `libmatrix.dll`, `libmatrix.h` и `libmatrix.lib` подкаталога **`distrib`**, полученные при создании библиотеки и файл `matrixdriver.c` в каталог `matrixdriver`. Для компиляции кода приложения `matrixdriver.c`, используется внешний C/C++ компилятор, который вызывается следующей командой `mbuild` (можно в строке DOS из каталога `matrixdriver`):

```
mbuild matrixdriver.c libmatrix.lib (на Windows)
```

Данная команда создает автономное консольное приложение `matrixdriver.exe`.

Еще раз обращаем внимание, что эта команда предполагает, что общедоступная библиотека `dll`, соответствующий заголовочный файл и `lib`-файл, созданы и находятся в текущем каталоге **matrixdriver**, там же находится и файл `matrixdriver.c`. Если это не так, то нужно определить, используя опцию `-I`, полный путь к `libmatrix.lib` и заголовочному файлу. Файл `libmatrix.ctf` нужен будет на этапе запуска приложения.

Правила написания кода приложения. Для использования созданной Компилятором MATLAB общедоступной библиотеки в приложении, код программы C должен включать следующую структуру:

1. Включение в приложение созданного заголовочного файла для каждой библиотеки `<lib-name>.h`.
2. Объявление переменных.
3. Вызов функции `mclInitializeApplication` для инициализации среды исполнения MCR MATLAB для работы приложения. Необходимо вызвать эту функцию один раз в приложении, и это нужно сделать перед вызовом любых других MATLAB функций и функций из общедоступной библиотеки. Это дает возможность образования экземпляров MCR. Функция `mclInitializeApplication` позволяет установить глобальные опции MCR. Они применяются одинаково ко всем экземплярам MCR.
4. Вызов функции `<lib-name>Initialize()` инициализации библиотеки для каждой общедоступной библиотеки созданной Компилятором MATLAB, которая включается в приложение. Эта функция исполняет несколько библиотечных локальных инициализаций, типа распаковки архива CTF, и старта экземпляра MCR с необходимой информацией, чтобы выполнить код в этом архиве. Эта функция возвращает значение `true` при успешной инициализации и `false` – в случае отказа.
5. Вызов экспортируемых функций библиотеки при необходимости (это – основная часть программы). При этом следует использовать `mx`-интерфейс C, чтобы обработать параметры ввода и вывода для этих функций. Отметим также, что если приложение отображает фигуру в окне MATLAB, то нужно включить вызов `mclWaitForFiguresToDie(NULL)` перед вызовом функций `Terminate` и `mclTerminateApplication`.
6. Вызов функции `<lib-name>Terminate()` завершения библиотеки, когда приложение больше не нуждается в данной библиотеке. Эта функция освобождает ресурсы, связанные с ее экземпляром MCR.
7. Вызов функции `mclTerminateApplication`, когда приложение больше не должно вызывать никакой библиотеки, созданной Компилятором MATLAB. Эта функция освобождает ресурсы на уровне приложения, используемые MCR.
8. Освобождение переменных, закрытие файлов и т.д., выход.

Эта структура кода хорошо видна на примере основной части кода приложения `matrixdriver.c`:


```

#include <stdio.h>
#ifdef __APPLE_CC__
#include <CoreFoundation/CoreFoundation.h>
#endif
/* Включение заголовочного файла MCR и заголовочных файла библиотек
 * порожденных MATLAB Compiler */
#include "libmatrix.h"

/* Эта функция используется для вывода на экран матрицы,
 * хранящейся в mxArray (см. Полный текст файла matrixdriver.c) */
void display(const mxArray* in);

void *run_main(void *x)
{
    int *err = x;
    mxArray *in1, *in2; /* Определение входных и выходных */
    mxArray *out = NULL; /* параметров для функций библиотеки */

    double data[] = {1,2,3,4,5,6,7,8,9}; /* Элементы матрицы */

    /* Вызов процедуры the mclInitializeApplication и проверка,
     * что приложение был инициализировано должным образом. Эта
     * инициализация должна быть сделана перед вызовом любого API
     * MATLAB, или вызовом любой созданной Компилятором функции
     * общедоступной библиотеки. */
    if( !mclInitializeApplication(NULL,0) )
    {
        fprintf(stderr, «Could not initialize the application.\n»);
        *err = -1;
        return(x);
    }
    /* Создание входных данных */
    in1 = mxCreateDoubleMatrix(3,3,mxREAL);
    in2 = mxCreateDoubleMatrix(3,3,mxREAL);
    memcpy(mxGetPr(in1), data, 9*sizeof(double)); /* in1 = data */
    memcpy(mxGetPr(in2), data, 9*sizeof(double)); /* in2 = data */

    /* Вызов процедуры инициализации библиотеки и проверка,
     * что инициализация прошла нормально. */
    if (!libmatrixInitialize()){
        fprintf(stderr, "Could not initialize the library.\n");
        *err = -2;
    }
    else
    {
        /* Вызов функций библиотеки */
        mlfAddmatrix(1, &out, in1, in2);

        /* Отображение на экран полученных функциями значений */

```

```

printf("The value of added matrix is:\n");
display(out);

/* Уничтожение возвращаемых значений переменных, чтобы переменные
 * могли использоваться многократно в следующих обращениях
 * к функциям библиотеки. */
mxDestroyArray(out); out=0;

mlfMultiplymatrix(1, &out, in1, in2);
printf("The value of the multiplied matrix is:\n");
display(out);
mxDestroyArray(out); out=0;

mlfEigmatrix(1, &out, in1);
printf("The eigenvalues of the first matrix are:\n");
display(out);
mxDestroyArray(out); out=0;

/* Вызов процедуры завершения библиотеки */
libmatrixTerminate();

/* Освобождение памяти */
mxDestroyArray(in1); in1=0;
mxDestroyArray(in2); in2 = 0;
}
#ifdef __APPLE_CC__
mclSetExitCode(*err);
#endif
mclTerminateApplication();
return 0;
}

```

Тестирование приложения. Перед запуском автономного приложения добавьте к путям каталог, содержащий созданную общедоступную библиотеку. Кроме того, необходимо добавить каталог <matlabroot>\bin\win32 к путям библиотек MCR. Файл libmatrix.ctf должен либо находиться в каталоге запуска приложения, либо к нему должны быть определены пути. Запустите приложения из командной строки DOS:

matrixdriver.exe

Результаты отображаются как на рис. 2.4.1.

Распространение автономных приложений, которые вызывают общедоступные библиотеки на основе Компилятора. Для этого нужно собрать и упаковать следующие файлы и затем установить их на машине развертывания:

- MCRInstaller.exe – среда исполнения компонентов MATLAB MCR. На Linux – это MCRInstaller.zip и файл unzip – для разархивирования;
- matrixdriver.ctf – архив CTF. Должен соответствовать платформе конечного пользователя;

```

C:\WINDOWS\system32\cmd.exe
Extracting CTF archive. This may take a few seconds,
size of your application. Please wait...
...CTF archive extraction complete.
The value of added matrix is:
2.00 8.00 14.00
4.00 10.00 16.00
6.00 12.00 18.00

The value of the multiplied matrix is:
30.00 66.00 102.00
36.00 81.00 126.00
42.00 96.00 150.00

The eigenvalues of the first matrix are:
16.12 -1.12 -0.00

```

Рис. 2.4.1. Результаты работы приложения *matrixdriver.exe*

- *matrixdriver.exe* – приложение;
- *libmatrix.dll* – общедоступная библиотека.

Распространение общедоступных библиотек, которые используются другими проектами. Чтобы распределять общедоступную библиотеку для использования с внешними приложениями, нужно установить следующее:

- *MCRInstaller.exe* – среда исполнения компонентов MATLAB MCR. На Linux – это *MCRInstaller.zip* и файл *unzip* – для разархивирования;
- *libmatrix.ctf* – архив CTF. Должен соответствовать платформе конечного пользователя;
- *libmatrix.dll* – общедоступная библиотека;
- *libmatrix.h* – заголовочный файл библиотек;
- *libmatrix.exports* – файл экспорта;
- *libmatrix.lib* – библиотека импорта.

Вызов общедоступной библиотеки. Во время выполнения, создается экземпляр MCR, связанный с каждой индивидуальной общедоступной библиотекой. Следовательно, если приложение линкуется с двумя общедоступными библиотеками, созданными компилятором MATLAB, будут два экземпляра MCR, созданные во время выполнения. Можно управлять поведением каждого экземпляра MCR, используя опции MCR. Есть два класса MCR опций – глобальные и локальные. Глобальные опции MCR одинаковы для каждого экземпляра MCR в приложении. Локальные опции MCR могут отличаться для разных экземпляров MCR. Для вызова и завершения MCR необходимо использовать следующие функции:

```

mclInitializeApplication
mclTerminateApplication

```

Функция *mclInitializeApplication* инициализирует среду исполнения MCR MATLAB и позволяет установить глобальные опции MCR. Они применяются одинаково ко всем экземплярам MCR. Необходимо установить эти опции перед созданием первого экземпляра MCR. Данные функции имеют вид:

```
bool mclInitializeApplication(const char **options, int count);  
bool mclTerminateApplication(void);
```

Функция `mclInitializeApplication` берет массив строк пользовательских установок опций (это те же самые опции, которые могут быть даны msc через опцию `-R`) и значение числа опций (длина массива опций). Возвращает `true` в случае успеха и `false` в случае неудачи.

Функция `mclTerminateApplication` не берет никаких параметров и может только вызываться после того, как все экземпляры MCR были закрыты. Возвращает `true` в случае успеха и `false` в случае неудачи. Обратите внимание, что после вызова `mclTerminateApplication` нельзя вызвать снова `mclInitializeApplication`. Никакие функции MathWorks также нельзя вызвать после `mclTerminateApplication`.

Таким образом, нужно вызвать функцию `mclInitializeApplication` один раз в приложении, и это нужно сделать перед вызовом любых других функций API MATLAB, типа `mx`-функций C или функций C для `mat`-файлов, или любых функций из созданной Компилятором MATLAB общедоступной библиотеки.

Для каждой библиотеки, созданной Компилятором MATLAB, которая включается в приложение, вызывается функция инициализации библиотеки. Эта функция выполняет несколько инициализаций: распаковка архива CTF, старт экземпляра MCR и передача ему необходимой информации для выполнения кода в архиве CTF. Функция инициализации библиотеки называется по имени библиотеки:

```
<libname>Initialize().
```

Есть две формы функции инициализации библиотеки:

```
bool libmatrixInitialize(void);  
bool libmatrixInitializeWithHandlers(  
    mclOutputHandlerFcn error_handler,  
    mclOutputHandlerFcn print_handler);
```

Более простая из двух функций инициализации не имеет никаких параметров. Эта функция создает экземпляр MCR, используя настройки по умолчанию для печати, обработки ошибок и другую информацию, созданную в течение процесса компиляции.

Однако, если Вы хотите иметь больше контроля над выводом на печать и над обработкой сообщений об ошибках, то можно вызвать вторую форму функции, которая берет два параметра. Вызывая эту функцию, Вы можете использовать ваши собственные версии подпрограмм печати и обработки ошибок, вызываемые MCR, подробнее об этом см. в `Print and Error Handling Functions`.

Завершение библиотеки производится функцией

```
void <libname>Terminate(void)
```

Эта функция вызывается один раз для каждой библиотеки перед вызовом `mclTerminateApplication`.

Функции, создаваемые из m-файлов

Для каждого m-файла, указанного в командной строке Компилятора MATLAB, создаются две функции: `mlx`-функция и `mlf`-функция. Каждая из этих функций исполняет одно и то же самое действие – вызывают функцию C-библиотеки, полученную из m-файла. Эти две функции представляют различные интерфейсы. Имя каждой функции основано на имени первой функции в m-файле, например, в рассмотренном выше примере – это функции `mlxAddmatrix` и `mlfAddmatrix`, `mlxMultiplymatrix` и `mlfMultiplymatrix`, `mlxEigmatrix` и `mlfEigmatrix`. Отметим, что для C++ библиотеки совместного использования, Компилятор MATLAB создает вместо `mlf`-функции, функцию, которая пишется без префикса и использует в качестве параметров тип `mwArray` вместо `mxArray`.

Поскольку функция в C/C++ могут иметь только единственный параметр вывода, а функция MATLAB обычно имеют много выводов, то m-функция транслируется в функцию C/C++, которая не имеют вывода, а все переменные вывода функции MATLAB, являются изменяемыми аргументами C/C++ функции, что допустимо в C/C++.

mlx функция интерфейса. Функция, которая начинается с префикса `mlx` использует те же типы и число параметров, что и MEX-функция MATLAB (см. документацию относительно MEX-функций), например:

```
extern void mlxAddmatrix(int nlhs, mxArray *plhs[],  
                        int nrhs, mxArray *prhs[]);
```

Первый параметр, `nlhs`, является числом параметров вывода, и второй параметр, `plhs`, является указателем на массив, который функция заполнит требуемым числом возвращаемых значений (буквы «lhs» в этих названиях параметров – это краткая запись «left-hand side», «левая сторона», т.к. переменные вывода в выражении MATLAB слева от оператора назначения). Третий и следующие параметры – число вводов и массив, содержащий входные переменные.

mlf функция интерфейса. Вторая из созданных функций начинается с префикса `mlf`,

```
extern void mlfAddmatrix(int nargout, mxArray** a, mxArray* a1,  
                        mxArray* a2);
```

Эта функция ожидает, что ее параметры ввода и вывода будут переданы как индивидуальные переменные, а не упакованы в массивы. Первый параметр – это число выводов функции.

Обратите внимание, что в обоих случаях, созданные функции распределяют память для возвращаемых значений. Если не удалять эту память (через `mxDestroyArray`) как делалось с переменными вывода, программа будет допускать расход памяти.

Использование `mlf`-функции удобнее, поскольку позволяет избежать управления дополнительными массивами, требуемыми формой `mlx`. Вызов в приведенном выше примере программы:

```
mlfAddmatrix(1, &out, in1, in2);
```

Замечание. Если переменные вывода передаются к mlf-функции не как NULL, то функция mlf будет делать попытку к их освобождению используя mxDestroyArray. Это означает, что можно многократно использовать переменные вывода в последовательных запросах к mlf функции, не волнуясь об утечках памяти. Это также подразумевает, что нужно передать все выводы либо NULL, либо как допустимый массив MATLAB, иначе Ваша программа не будет работать, потому что менеджер памяти не делает различия между неинициализированным (недопустимым) указателем массива и допустимым массивом. Он будет пробовать освободить недопустимый указатель, что обычно вызывает ошибку сегментации или подобную неустранимую ошибку. Например, в программе выше мы полагаем:

```
mxArray *out = NULL;
```

Замечание. На платформах Microsoft Windows, Компилятор MATLAB генерирует дополнительную функцию инициализации, стандартную функцию инициализации Microsoft DLL, DllMain.

```
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, void *pv)
```

Созданная функция DllMain выполняет очень важное действие: она определяет местонахождение каталога, в котором общедоступная библиотека сохранена на диске. Эта информация используется, чтобы найти архив CTF, без которого не будет выполняться приложение. Если Вы изменяете созданную DllMain (что мы не рекомендуем делать), удостоверьтесь, что Вы сохраняете эту часть ее функциональных возможностей.

Использование varargin и varargout в интерфейсе m-функции

Если интерфейс m-функции использует аргументы varargin или varargout с переменным числом параметров, то их нужно передавать как массивы ячеек. Например, если имеется N varargin-ов, то нужно создать один массив ячеек размера 1-на-N. Точно так же varargout-ы возвращаются как один массив ячеек. Длина varargout равна числу возвращаемых значений функции, минус число обязательных возвращаемых переменных. Например, рассмотрим интерфейс этого m-файла:

```
[a,b,varargout] = myfun(x,y,z,varargin)
```

Соответствующий C интерфейс для него

```
void mlfMyfun(int numOfRetVars, mxArray **a, mxArray **b,  
             mxArray **varargout, mxArray *x, mxArray *y,  
             mxArray *z, mxArray *varargin)
```

C++ библиотека совместного использования

Библиотеки C++ совместного использования создаются компилятором MATLAB из произвольного набора m-файлов совершенно аналогично. Создадим каталог C:\R2007a\work\CPP_Lib и в него скопируем файлы addmatrix.m, multiplymatrix.m, eigmatrix.m и matrixdriver.cpp из каталога примеров MATLAB <matlabroot>\extern\examples\compiler. Последний файл matrixdriver.cpp содержит функцию

main автономного приложения, которая вызывает функции библиотеки. Сначала создадим библиотеку, а затем – приложение.

Создание библиотеки. Для создания библиотеки C++ мы используем среду разработки Deployment Tool, которая вызывается командой:

```
deploytool
```

Назовем проект именем libmatrixp, поскольку такое имя библиотеки используется далее в коде приложения matrixdriver.cpp. Тогда файлы компонента будут в двух подкаталогах **distrib** и **src** каталога **libmatrixp**.

Можно также исполнить команду:

```
mcc -W cpplib:libmatrixp -T link:lib addmatrix.m multiplymatrix.m  
eigmatrix.m -v
```

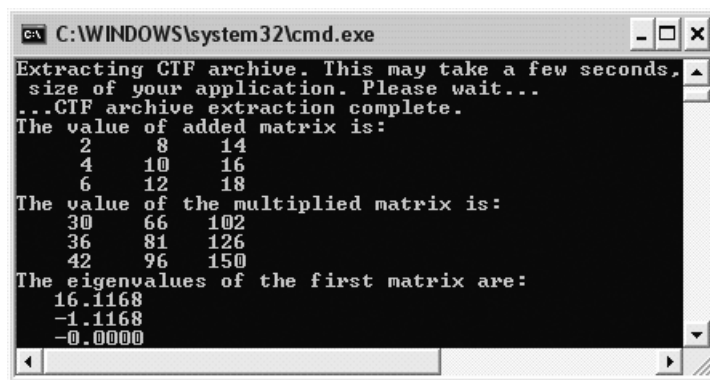
Опция -W cpplib: <libname> указывает Компилятору MATLABу создать функциональную обертку для общедоступной библиотеки и называть ее <libname>. Опция -T link:lib определяет конечный вывод как общедоступную библиотеку. Создаются все необходимые файлы библиотеки: libmatrixp.dll, libmatrixp.ctf, libmatrixp.h, libmatrixp.cpp, libmatrixp_mcc_component_data.c, libmatrixp.exports, libmatrixp.lib, libmatrixp.prj и readme.txt.

Создание приложения. Файл matrixdriver.cpp дает пример кода C++, который использует функции этой библиотеки. Для создания автономного приложения мы создаем подкаталог **matrixdriver** в рабочем каталоге, помещаем в него файлы библиотеки libmatrixp.dll, libmatrixp.ctf, libmatrixp.h, libmatrixp.exports, libmatrixp.lib и файл C++ кода приложения matrixdriver.cpp. Делаем подкаталог matrixdriver текущим каталогом MATLAB и исполняем команду (из строки MATLAB):

```
mbuild matrixdriver.cpp libmatrixp.lib
```

В результате создается файл matrixdriver.exe, работа которого показана на рис. 2.4.2. Приведем часть исходного кода файла matrixdriver.cpp.

```
#ifdef __APPLE_CC__  
#include <CoreFoundation/CoreFoundation.h>
```



```
C:\WINDOWS\system32\cmd.exe  
Extracting CTF archive. This may take a few seconds,  
size of your application. Please wait...  
...CTF archive extraction complete.  
The value of added matrix is:  
2 8 14  
4 10 16  
6 12 18  
The value of the multiplied matrix is:  
30 66 102  
36 81 126  
42 96 150  
The eigenvalues of the first matrix are:  
16.1168  
-1.1168  
-0.0000
```

Рис. 2.4.2. Результаты работы приложения matrixdriver.exe

```
#endif

// Подключение заголовочного файла из библиотек MCR
#include "libmatrixp.h"

void *run_main(void *x)
{
    int *err = (int *)x;
    if (err == NULL) return 0;
    // Вызов процедуры mclInitializeApplication и проверка,
    // что приложение было инициализировано должным образом. Эта
    // инициализация должна быть сделана перед вызовом любого API
    // MATLAB, или вызовом любой созданной Компилятором функции
    // общедоступной библиотеки.

    if (!mclInitializeApplication(NULL,0))
    {
        std::cerr << "could not initialize the application properly"
            << std::endl;
        *err = -1;
        return x;
    }
    if( !libmatrixpInitialize() )
    {
        std::cerr << "could not initialize the library properly"
            << std::endl;
        *err = -1;
    }
    else
    {
        try
        {
            // Создание входных данных
            double data[] = {1,2,3,4,5,6,7,8,9};
            mxArray in1(3, 3, mxDOUBLE_CLASS, mxREAL);
            mxArray in2(3, 3, mxDOUBLE_CLASS, mxREAL);
            in1.SetData(data, 9);
            in2.SetData(data, 9);

            // Создание выходного массива
            mxArray out;

            // Вызов функции библиотеки
            addmatrix(1, out, in1, in2);

            // Вывод на экран результатов
            std::cout << "The value of added matrix is:" << std::endl;
            std::cout << out << std::endl;

            multiplymatrix(1, out, in1, in2);
```



```

        std::cout << "The value of the multiplied matrix is:"
            << std::endl;
        std::cout << out << std::endl;

        eigmatrix(1, out, in1);
        std::cout << "The eigenvalues of the first matrix are:"
            << std::endl;
        std::cout << out << std::endl;
    }
    catch (const mwException& e)
    {
        std::cerr << e.what() << std::endl;
        *err = -2;
    }
    catch (...)
    {
        std::cerr << "Unexpected error thrown" << std::endl;
        *err = -3;
    }
    // Вызов процедур завершения
    libmatrixpTerminate();
}
#ifdef __APPLE_CC__
    mclSetExitCode(*err);
#endif
mclTerminateApplication();
return 0;
}

```

Отметим некоторые особенности случая C++:

- функции интерфейса используют тип `mwArray` для передачи параметров, а не тип `mxArray`, используемый в С библиотеке;
- используются исключения C++ для сообщений об ошибках. Поэтому, все запросы должны быть обернуты в блок try-catch:

```

try
{
    ...
}
catch (const mwException& e)
{
    ...
}

```

2.4.2. Создание автономных приложений

Рассмотрим создание автономных приложений. Сначала приведем пример создания приложения только из m-файлов, а затем – с использованием кода на языке С.

Создание кода только из m-файлов

Это наиболее простой способ создания автономного приложения. Программа сначала разрабатывается в среде MATLAB в виде одного, или нескольких m-файлов. При этом используется вся сила MATLAB и его инструментальных средств. Пример создания приложения из одного m-файла достаточно подробно рассмотрен в первой главе. В случае нескольких m-файлов один из файлов должен быть главным, он выполняет требуемое действие, вызывая для этого остальные m-функции. При использовании среды разработки Deployment Tool главный файл добавляется в папку **Main function**, остальные файлы помещаются в папку **Other files**. В случае использования командной строки и функции msc, главный файл ставится на первое место в командной строке.

Пример 1. Рассмотрим простое приложение, исходный текст которого состоит из двух m-файлов mrank.m и main.m (тестовые файлы из каталога примеров Компилятора <matlab>\extern\examples\compiler\). Функция mrank.m возвращает вектор r рангов магических квадратов порядка от 1 до n. Например, после завершения функции, r(3) содержит ранг магического квадрата размера 3-на-3.

```
function r = mrank(n)
r = zeros(n,1);
for k = 1:n
    r(k) = rank(magic(k));
end
```

Файл main.m содержит «main-подпрограмму», которая вызывает mrank и затем печатает результаты.

```
function main
r = mrank(5)
```

Для компиляции файлов в автономное приложение из командной строки нужно вызвать Компилятор MATLAB следующим образом:

```
msc -m main mrank
```

Создается файл приложения main.exe (по имени главного файла) и сопутствующие ему файлы.

Мы будем использовать среду разработки Deployment Tool. Создадим каталог для проекта Mrank_Appl_1 и скопируем туда файлы mrank.m и main.m. Сделаем этот каталог текущим рабочим каталогом MATLAB и вызовем среду разработки Deployment Tool командой:

```
deploytool
```

Пусть имя проекта – Mrank. В этом случае исполняемый файл имеет имя проекта, Mrank.exe. Главный и остальные m-файлы проекта будут расположены так, как показано на рис. 2.4.3.

После нажатия кнопки **Build** начинается процесс построения. Будет создан каталог проекта **Mrank** и в нем два подкаталога **distrib** и **src**. Подкаталог **distrib** содержит исполняемый файл Mrank.exe и файл Mrank.ctf. При первом запуске приложения архив Mrank.ctf автоматически разворачивается в подкаталог **Mrank_mcr** и затем выполняется приложение. Результат работы программы показан на рис. 2.4.4.

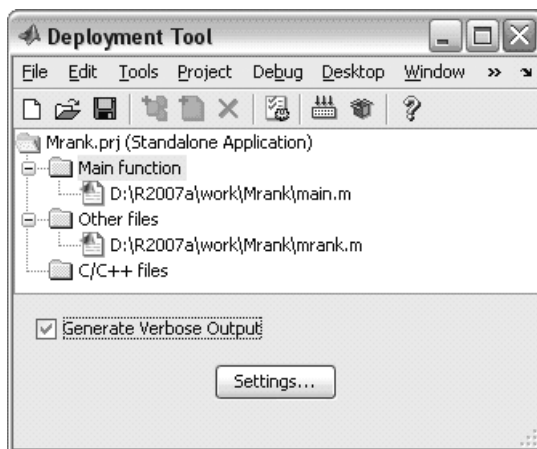


Рис. 2.4.3. Создание приложения из нескольких m-файлов

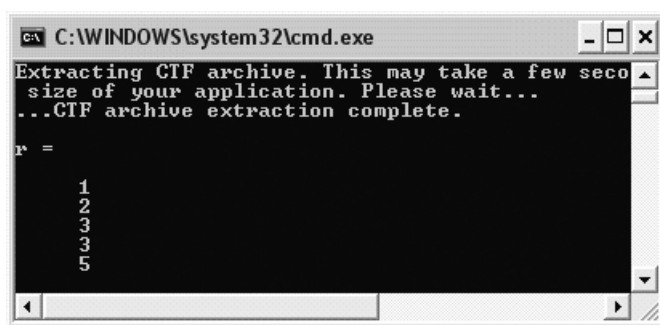


Рис. 2.4.4. Результаты работы приложения matrixdriver.exe

Объединение M-файлов и кода С или С++

Наиболее общий способ создания автономного приложения состоит в том, чтобы часть приложения записать как один или более m-файлов функций, а другие части написать непосредственно в С или С++.

Пример 2. Объединение m-файла и С кода. Рассмотрим простое приложение, исходный текст которого состоит из двух файлов mrnk.m, printmatrix.m и следующих С-кодов: mrnkpr.c, main_for_lib.c, and main_for_lib.h. Эти тестовые файлы копируем из каталога примеров <matlabroot>\extern\examples\compiler\ в текущий каталог проекта, например, <matlabroot>\work\Mrank_Appl_2. Данные файлы имеют следующий смысл:

- mrnk.m – вычисляет ранги магических квадратов;
- printmatrix(m) – выводит на дисплей полученный массив;
- mrnkpr.c – содержит вызов функций mlfMrnk и mlfPrintmatrix;

- `main_for_lib.c` – обертка, содержит функцию `main` ;
- `main_for_lib.h` – является заголовочным для структур, используемых в `main_for_lib.c`

Для создания приложения достаточно использовать команду в MATLAB:

```
mcc -W lib:libPkg -T link:exe mrank printmatrix mrankp.c  
main_for_lib.c
```

Компилятор создает, кроме приложения `mrnk.exe`, следующие файлы С-библиотеки Компилятора MATLAB: `libPkg.c`, `libPkg.ctf`, `libPkg.h`, `libPkg_mcc_component_data.c` и `libpkg.exports`.

Таким образом, фактически из `m`-файлов `mrnk.m`, `printmatrix.m` создается библиотека **libPkg**, содержащая функций `mlfMrank` и `mlfPrintmatrix`, но `dll`-файл не создается. Так же, как и в случае библиотеки, С-код `mrnkp.c` содержит процедуры инициализации:

```
mclInitializeApplication(NULL,0);  
libPkgInitialize();
```

вызова функций `mlfMrank` и `mlfPrintmatrix`

```
mlfMrank(1, &R, N);  
mlfPrintmatrix(R);
```

и завершения

```
libPkgTerminate();  
mclTerminateApplication();
```

В результате компиляции `m`-функций `mrnk.m` и `printmatrix.m` получают функции `mlfMrank` и `mlfPrintmatrix`. Первая функция вычисляет вектор рангов магических квадратов, а вторая отображает вектор, полученный `mlfMrank`. Созданное приложение `mrnk.exe` принимает в качестве параметра максимальное число рангов магических квадратов. Если параметр опущен, то он считается равным 12. Результат работы программы примерно такой же как на рис. 2.4.4.

Листинг и обсуждение С-кода `mrnkp.c` приведено ниже.

Вообще, при включении созданного компилятором кода в большее приложение, нужно использовать опции `-W lib` или `-W srplib`, даже, если общедоступная библиотека не создается.

Файл `mrnkp.c`. Компилятор MATLAB имеет две различные версии `mrnkp.c` в каталоге: `<matlabroot>/extern/examples/compiler`. Это файлы `mrnkp.c` и `mrnkwin.c`. Файл `mrnkp.c` содержит POSIX С `main`-функцию, вызов функции `mlfMrank`, полученной в результате компиляции `m`-файла `mrnk.m` и вызов функции `mlfPrintmatrix`, полученной в результате компиляции `m`-файла `printmatrix.m`. Файл `mrnkwin.c` есть Windows-версию `mrnkp.c`, она содержит `WinMain`-функцию и вызовы функций `mlfMrank` и `mlfPrintmatrix`.

Приведем код `mrnkp.c`:

```
#include <stdio.h>  
#include <math.h>  
#include "libPkg.h"  
#include "main_for_lib.h"
```

```

void *run_main( void *in )
{
    mxArray *N;           /* Матрица, содержащая n. */
    mxArray *R = NULL;    /* результирующая матрица */
    int n;                 /* Целый параметр из командной строки */

    /*Получение параметра из командной строки */
    if (((inputs*)in)->ac >= 2) {
        n = atoi(((inputs*)in)->av[1]);
    } else {
        n = 12;
    }

    /*Вызов процедур инициализации */
    mclInitializeApplication(NULL,0)
    libPkgInitialize()

    /*Создание 1-на-1 матрицы содержащей n. */
    N = mxCreateDoubleScalar(n);

    /*Вызов mlfMrank, скомпилированной версии mrank.m. */
    mlfMrank(1, &R, N);

    mlfPrintmatrix(R);      /* Печать результатов */

    mxDestroyArray(N);      /* Освобождение массивов */
    mxDestroyArray(R);

    libPkgTerminate();      /* Завершение библиотеки m-функций */
    mclTerminateApplication();
}

```

Основа кода mrank.c – это вызов функции mlfMrank, полученной в результате компиляции m-функции mrank.m, и отображение вектора, который возвращает mlfMrank. Сначала, код должен инициализировать MCR и библиотеку libPkg.

```

mclInitializeApplication(NULL,0);
libPkgInitialize();

```

Чтобы понять, как вызвать mlfMrank, рассмотрим ее объявление в заголовочном файле libPkg.h:

```

void mlfMrank(int nargout, mxArray** r, mxArray* n);

```

Как мы видим, функция mlfMrank ожидает один входной параметр и возвращает одно значение. Все параметры ввода и вывода – указатели на тип данных mxArray. Для создания и использования переменных типа mxArray в С-коде можно использовать mx-процедуры, описанные в документации «MATLAB/External Interfaces» из общего раздела документации MATLAB. Например, чтобы создать 1-на-1 вещественную переменную N типа mxArray *, можно использовать С-функцию mxCreateScalarDouble,

```

N = mxCreateDoubleScalar(n);

```

Теперь `mrankp` может вызвать `mlfMrank`, принимая инициализированное `N` как единственный входной параметр:

```
mlfMrank(1, &R, N);
```

Как уже ранее отмечалось, переменные вывода, которым не были назначены значения `mxArray`, должны быть установлены как `NULL`, поэтому переменная `R` должна быть инициализирована как `NULL`. Для вывода на экран значения переменной `R` вызывается функция `mlfPrintmatrix`. Эта функция определена в файле `printmatrix.m`. В конце программа `mrankp` освобождает память и вызывает функции завершения для библиотеки `m-функций libPkg` и для `MCR`.

Хотя можно использовать функции двух типов `mlx` и `mlf`, функции `mlf` удобнее, поскольку позволяют избежать управления дополнительными массивами, требуемыми формой `mlx`.

Замечание. В каталоге `<matlabroot>\extern\examples\compiler\` имеется еще один набор тестовых файлов `multarg.m`, `printmatrix.m` и `multargp.c` для создания еще одного автономного приложения, которое производит некоторые математические операции с матрицами и выводит результаты на дисплей.

2.5. Классы C++ Компилятора 4.6 MATLAB®

Для того, чтобы использовать в C++ методы, созданные Компилятором 4.6, в MATLAB разработаны специальные классы для C++:

- `mwArray` – используется для передачи параметров ввода/вывода к C++ функциям, созданным Компилятором MATLAB;
- `mwString` – класс строковых данных.

Эти классы представляют аналоги в C++ основных типы данных MATLAB. Для обработки исключений, которые происходят при работе с функциями `mwArray`, имеется класс `mwException`. Все эти классы описаны в заголовочных файлах каталога `<matlab>\extern\include`. В данном параграфе дается краткое описание библиотеки классов C++ Компилятора 4.6 MATLAB R2007a.

2.5.1. Основные типы данных

Класс `mwArray` поддерживает в C++ все *основные типы*, которые могут быть сохранены в массиве MATLAB. Эти типы перечислены в табл. 2.5.1.

Таблица 2.5.1. Основные типы

Тип	Описание	mxClassID
<code>mxChar</code>	Символьный тип	<code>mxCHAR_CLASS</code>
<code>mxLogical</code>	Логический тип	<code>mxLOGICAL_CLASS</code>
<code>mxDouble</code>	Числовой тип <code>double</code>	<code>mxDOUBLE_CLASS</code>
<code>mxSingle</code>	Числовой тип одинарной точности	<code>mxSINGLE_CLASS</code>

Таблица 2.5.1. Основные типы (окончание)

Тип	Описание	mxClassID
mxInt8	Целое число со знаком, 1-byte	mxINT8_CLASS
mxUInt8	Целое число без знака, 1-byte	mxUINT8_CLASS
mxInt16	Целое число со знаком, 2-byte	mxINT16_CLASS
mxUInt16	Целое число без знака, 2-byte	mxUINT16_CLASS
mxInt32	Целое число со знаком, 4-byte	mxINT32_CLASS
mxUInt32	Целое число без знака, 4-byte	mxUINT32_CLASS
mxInt64	Целое число со знаком, 8-byte	mxINT64_CLASS
mxUInt64	Целое число без знака, 8-byte	mxUINT64_CLASS

2.5.2. Класс *mwArray*

Он используется для того, чтобы передать параметры ввода/вывода к функциям C++ интерфейса, созданным Компилятором MATLAB. Объекты *mwArray* являются аналогами в C++ массивов MATLAB. Класс *mwArray* обеспечивает необходимые конструкторы, методы, и операторы для создания массива и инициализации, а также для простой индексации. Отметим, что арифметические операторы, такие, как сложение и вычитание, не поддерживаются начиная с версии R14.

Класс *mwArray* определен в файле `mclcppclass.h` каталога `<matlab>\extern\include`. Поэтому конструкторы, методы и операторы требуют подключения этого заголовочного файла:

```
#include "mclcppclass.h"
```

Ниже представлено краткое описание каждого конструктора, метода и оператора и приведены примеры их использования в C++.

Конструкторы

mwArray(). Создание пустого массива типа `mxDOUBLE_CLASS`

```
mwArray a;
```

mwArray(mxClassID mxID). Создание пустого массива указанного типа. Может использоваться любой допустимый `mxClassID`.

```
mwArray a(mxDOUBLE_CLASS);
```

mwArray(int num_rows, int num_cols, mxClassID mxID, mxComplexity cmplx = mxREAL). Создание матрицы указанного типа и измерений. Все элементы инициализированы нулями. Для числовых типов, указание `mxCOMPLEX` последним параметром создает комплексную матрицу. Для матриц ячеек, все элементы инициализированы пустыми ячейками.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
mwArray b(3, 3, mxSINGLE_CLASS, mxCOMPLEX);
```

```
mwArray c(2, 3, mxCELL_CLASS);
```

Аргументы. `num_rows` – число строк; `num_cols` – число столбцов; `mxID` – тип данных типа матрицы; `cmplx` – комплексная матрица (только числовой тип).

mwArray(int num_dims, const int* dims, mxClassID mxID, mxComplexity cmplx = mxREAL). Создание N-мерного массива указанного типа и измерений. Для числовых типов, массив может быть вещественным или комплексным.

```
int dims[3] = {2, 3, 4};
mwArray a(3, dims, mxDOUBLE_CLASS);
mwArray b(3, dims, mxSINGLE_CLASS, mxCOMPLEX);
mwArray c(3, dims, mxCELL_CLASS);
```

mwArray(const char* str). Создание символьного массива 1-на-n типа `mxCHAR_CLASS`, с `n = strlen(str)` и инициализация данных массива символами из снабженной строки.

```
mwArray a("This is a string");
```

Аргументы. **str** – строка, заканчивающаяся символом `NULL`.

mwArray(int num_strings, const char str).** Создание символьной матрицы из списка строк. Созданный массив имеет измерения `m`-на-`max`, где `max` является длиной самой длинной строки в `str`.

```
const char** str = {"String1", "String2", "String3"};
mwArray a(3, str);
```

mwArray(int num_rows, int num_cols, int num_fields, const char fieldnames).** Создание матрицы структур указанных измерений и имен полей.

```
const char** fields = {"a", "b", "c"};
mwArray a(2, 2, 3, fields);
```

Аргументы. `num_rows` – число строк в матрице структур; `num_cols` – число столбцов в матрице структур; `num_fields` – число полей в матрице структур; `fieldnames` – массив строк, заканчивающихся символом `NULL`, представляющих имена полей.

mwArray(int num_dims, const int* dims, int num_fields, const char fieldnames).** Создание n-мерного массива структур указанных измерений и имен полей.

```
const char** fields = {"a", "b", "c"};
int dims[3] = {2, 3, 4};
mwArray a(3, dims, 3, fields);
```

mwArray(const mwArray& arr). Конструктор настоящей копии `mwArray`. Создает новый массив из существующего.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
mwArray b(a);
```

mwArray(<type> re). Создание вещественного скалярного массива типа входного параметра и инициализация данными из значения входного параметра.

```
double x = 5.0;
mwArray a(x);          // Создание массива double 1-на-1 для числа 5.0
```

Этот конструктор используется для создания вещественного скалярного массива. `<type>` может быть любым из следующих: `mxDouble`, `mxSingle`, `mxInt8`,

mxUInt8, mxInt16, mxUInt16, mxInt32, mxUInt32, mxInt64, mxUInt64, или mxLogical. Скалярный массив соответствует типу входного параметра.

mwArray(<type> re, <type> im). Создание комплексного скалярного массива типа входных параметров и инициализация вещественной и мнимой частей данными из значений входных параметров.

```
double re = 5.0;
```

```
double im = 10.0;
```

```
mwArray a(re, im); // Создание комплексного массива 1-на-1 для числа 5+10i
```

Методы копирования

mwArray Clone() const. Возвращает новый массив, представляющий настоящую копию (deep copy) этого массива.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
mwArray b = a.Clone();
```

mwArray SharedCopy() const. Возвращает новый массив, представляющий общедоступную копию этого массива. Новый и первоначальный массивы оба указывают на те же самые данные.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
mwArray b = a.SharedCopy();
```

mwArray Serialize() const. Сериализация данного массива в байтовый массив.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
mwArray s = a.Serialize();
```

Методы получения информации о массиве

mxClassID ClassID() const. Возвращает тип этого массива.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
mxClassID id = a.ClassID(); // Возвращает mxDOUBLE_CLASS
```

int ElementSize() const. Возвращает размер в байтах элемента этого массива.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
int size = a.ElementSize(); // Возвращает sizeof(double)
```

int NumberOfElements() const. Возвращает число элементов в этом массиве.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
int n = a.NumberOfElements(); // Возвращает 4
```

int NumberOfNonZeros() const. Возвращает число элементов, которые могли потенциально быть отличными от нуля в разреженном массиве – это число элементов, для которых выделена память разреженной матрицы. Если основной массив не разрежен, то возвращается значение NumberOfElements().

```
mwArray a(2, 2, mxDOUBLE_CLASS);
```

```
int n = a.NumberOfNonZeros(); // Возвращает 4
```

int MaximumNonZeros() const. Возвращает максимальное число отличных от нуля элементов для разреженного массива. Если основной массив не разрежен, то возвращается то же самое значение как в NumberOfElements().

```
mwArray a(2, 2, mxDOUBLE_CLASS);
int n = a.MaximumNonZeros(); // Возвращает 4
```

int NumberOfDimensions() const. Возвращает число измерений в этом массиве.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
int n = a.NumberOfDimensions(); // Возвращает 2
```

int NumberOfFields() const. Возвращает число полей в массиве struct структур.

```
const char** fields = {"a", "b", "c"};
mwArray a(2, 2, 3, fields);
int n = a.NumberOfFields(); // Возвращает 3
```

mwString GetFieldName(int index). Возвращает строку, представляющую имя i-го поля (отсчитываемого с нуля) в массиве структур.

```
const char** fields = {"a", "b", "c"};
mwArray a(2, 2, 3, fields);
const char* name = (const char*)a.GetFieldName(1); // должен вернуть "b"
```

Аргументы. Index – номер поля, отсчитываемый с нуля.

mwArray GetDimensions() const. Возвращает массив типа mxINT32_CLASS представляющий размеры этого массива.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
mwArray dims = a.GetDimensions();
```

bool IsEmpty() const. Возвращает true, если основной массив пуст.

```
mwArray a;
bool b = a.IsEmpty(); // должен вернуть true
```

bool IsSparse() const. Возвращает true если массив разреженен.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
bool b = a.IsSparse(); // Возвращает false
```

bool IsNumeric() const. Возвращает true, если основной массив является числовым.

bool IsComplex() const. Возвращает true если массив комплексный.

int HashCode() const. Возвращает хэш-код для этого массива.

```
mwArray a(1, 1, mxDOUBLE_CLASS);
int n = a.HashCode();
```

mwString ToString() const. Возвращает строковое представление данного массива.

```
#include <stdio.h>
mwArray a(1, 1, mxDOUBLE_CLASS, mxCOMPLEX);
a.Real() = 1.0;
a.Imag() = 2.0;
printf("%s\n", (const char*)(a.ToString())); // Должно печататься
// "1 + 2i" на экране.
```

mwArray RowIndex() const. Возвращает массив типа mxINT32_CLASS, содержащий индексы строк каждого элемента в этом массиве. Для разреженных массивов возвращаются индексы только ненулевых элементов.

```
mwArray a(1, 1, mxDOUBLE_CLASS);
mwArray rows = a.RowIndex();
```

mwArray ColumnIndex() const. Возвращает массив типа `mxINT32_CLASS`, содержащий индексы столбцов каждого элемента в этом массиве. Для разреженных массивов возвращаются индексы только ненулевых элементов.

```
mwArray a(1, 1, mxDOUBLE_CLASS);
mwArray rows = a.ColumnIndex();
```

void MakeComplex(). Преобразование вещественного числового массива в комплексный

```
double rdata[4] = {1.0, 2.0, 3.0, 4.0};
double idata[4] = {10.0, 20.0, 30.0, 40.0};
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(rdata, 4);
a.MakeComplex();
a.Imag().SetData(idata, 4);
```

Методы сравнения

bool Equals(const mwArray& arr) const. Проверка равенства двух массивов.

```
mwArray a(1, 1, mxDOUBLE_CLASS);
mwArray b(1, 1, mxDOUBLE_CLASS);
a = 1.0;
b = 1.0;
bool b = a.Equals(b); // должен вернуть true.
```

Аргументы. **arr** – массив, что сравнивается с данным массивом.

int CompareTo(const mwArray& arr) const. Сравнение данных двух массивов одного типа относительно порядка.

```
mwArray a(1, 1, mxDOUBLE_CLASS);
mwArray b(1, 1, mxDOUBLE_CLASS);
a = 1.0;
b = 1.0;
int n = a.CompareTo(b); // должен вернуть 0
```

Методы доступа к элементам массива mwArray

mwArray Get(int num_indices, ...). Возвращает отдельный элемент `mwArray` массива, указанный индексами. Первым передается число индексов, далее идет отдельный запятыми список индексов.

```
double data[4] = {1.0, 2.0, 3.0, 4.0};
double x;
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(data, 4);
x = a.Get(1,1);           // x = 1.0
x = a.Get(2, 1, 2);       // x = 3.0
x = a.Get(2, 2, 2);       // x = 4.0
```

mwArray Get(const char* name, int num_indices, ...). Возвращает отдельный элемент с указанным именем поля и индексами в массиве структур. Первым передается поле, далее число индексов и отделенный запятыми список индексов.

```
const char** fields = {"a", "b", "c"};
mwArray a(1, 1, 3, fields);
mwArray b = a.Get("a", 1, 1); // b=a.a(1)
mwArray b = a.Get("b", 2, 1, 1); // b=a.b(1,1)
```

Аргументы: Name – строка, содержащая имя поля; num_indices – число индексов; далее – отделенный запятыми список входных индексов. Число элементов должно равняться num_indices.

mwArray GetA(int num_indices, const int* index). Возвращает отдельный элемент MwArray массива по указанному индексу. Индекс передается как массив индексов.

```
double data[4] = {1.0, 2.0, 3.0, 4.0};
int index[2] = {1, 1};
double x;
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(data, 4);
x = a.GetA(1, index); // x = 1.0
x = a.GetA(2, index); // x = 1.0
index[0] = 2;
index[1] = 2;
x = a.Get(2, index); // x = 4.0
```

mwArray GetA(const char* name, int num_indices, const int* index). Возвращает отдельный элемент с указанным именем поля и индексом в массиве структур. Индекс передается как массив индексов.

```
const char** fields = {"a", "b", "c"};
int index[2] = {1, 1};
mwArray a(1, 1, 3, fields);
mwArray b = a.Get("a", 1, index); // b=a.a(1)
mwArray b = a.Get("b", 2, index); // b=a.b(1,1)
```

mwArray Real(). Возвращает ссылку mwArray на вещественную часть комплексного массива.

```
double rdata[4] = {1.0, 2.0, 3.0, 4.0};
double idata[4] = {10.0, 20.0, 30.0, 40.0};
mwArray a(2, 2, mxDOUBLE_CLASS, mxCOMPLEX);
a.Real().SetData(rdata, 4);
a.Imag().SetData(idata, 4);
```

Этот метод используется для обращения к вещественной части комплексного массива. Возвращенный массив mwArray считается вещественным и имеет ту же самую размерность и тип, как исходный массив.

mwArray Imag(). Возвращает ссылку mwArray на мнимую часть комплексного массива.

```
double rdata[4] = {1.0, 2.0, 3.0, 4.0};
double idata[4] = {10.0, 20.0, 30.0, 40.0};
mwArray a(2, 2, mxDOUBLE_CLASS, mxCOMPLEX);
a.Real().SetData(rdata, 4);
a.Imag().SetData(idata, 4);
```

Этот метод используется для обращения к мнимой части комплексного массива. Возвращенный массив `mwArray` считается мнимым и имеет ту же самую размерность и тип, как исходный массив.

void Set(const mwArray& arr). Назначение общедоступной копии входного массива в указанную ячейку, для массивов типа `mxCELL_CLASS` и `mxSTRUCT_CLASS`. Этот метод используется для создания массивов ячеек и структур.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
mwArray b(2, 2, mxINT16_CLASS);
mwArray c(1, 2, mxCELL_CLASS);
c.Get(1,1).Set(a);           // Назначение c(1) = a
c.Get(1,2).Set(b);           // Назначение c(2) = b
```

void GetData(<numeric-type>* buffer, int len) const,
void SetData(<numeric-type>* buffer, int len) const. Копирование данных из одного массива в другой. Данные копируются в столбцовом порядке. Если основной массив не имеет того же самого типа, как массив назначения, данные преобразуются в этот тип после копирования. Если преобразование не может быть сделано, вызывается `mwException`.

```
double rdata[4] = {1.0, 2.0, 3.0, 4.0};
double data_copy[4];
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(rdata, 4);         // Копирование из rdata в a
a.GetData(data_copy, 4);     // Копирование из a в data_copy
```

Аргументы: `buffer` – массив для получения копии; `len` – максимальная длина буфера.

void GetLogicalData(mxLogical* buffer, int len) const,
void SetLogicalData(mxLogical* buffer, int len) const. Копирование данных из одного логического массива в другой.

```
mxLogical data[4] = {true, false, true, false};
mxLogical data_copy[4];
mwArray a(2, 2, mxLOGICAL_CLASS);
a.SetData(data, 4);
a.GetData(data_copy, 4);
```

void GetCharData(mxChar* buffer, int len) const,
void SetCharData(mxChar* buffer, int len) const. Копирование данных из одного символьного массива в другой.

```
mxChar data[6] = {'H', 'e', 'l', 'l', 'o', '\\0'};
mxChar data_copy[6];
mwArray a(1, 6, mxCHAR_CLASS);
a.SetData(data, 6);          // Копирование из data в a
a.GetData(data_copy, 6);     // Копирование из a в data_copy
```

Операторы

mwArray operator()(int i1, int i2, int i3, ...,). Возвращает отдельный элемент `mwArray` с указанными индексами. Индексы передают как отделенный запятыми список индексов. Поддерживается от 1 до 32 индексов.

```
double data[4] = {1.0, 2.0, 3.0, 4.0};
double x;
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(data, 4);
x = a(1,1);      // x = 1.0
x = a(1,2);      // x = 3.0
x = a(2,2);      // x = 4.0
```

Допустимое число индексов, которые можно передать либо 1 (простая одномерная постолбцовая индексация), или размерность NumberOfDimensions() (многомерная индексация), когда используется список индексов, чтобы обратиться к указанному элементу многомерного массива.

mwArray operator()(const char* name, int i1, int i2, int i3, ...,). Возвращает отдельный элемент mwArray, указанный именем поля и индексами в массиве структур. Поддерживается 1 до 32 индексов.

```
const char** fields = {"a", "b", "c"};
int index[2] = {1, 1};
mwArray a(1, 1, 3, fields);
mwArray b = a("a", 1, 1);
mwArray b = a("b", 1, 1);
```

mwArray& operator=(const <type>& x). Назначение простого скалярного значения элементу массива. Этот оператор применяется для всех числовых и логических типов.

```
mwArray a(2, 2, mxDOUBLE_CLASS);
a(1,2) = 2.0;      // назначение 2.0 элементу (1,2)
a(2,1) = 3.0;      // назначение 3.0 элементу (2,1)
```

operator <type>() const. Выбор отдельного скалярного значения из массива. Этот оператор используется для всех числовых и логических типов.

```
double data[4] = {1.0, 2.0, 3.0, 4.0};
double x;
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(data, 4);
x = (double)a(1,2); // x = 3.0
x = (double)a(2,1); // x = 2.0
```

Статические методы

static mwArray Deserialize(const mwArray& arr). Обратное преобразование массива, который был сериализован mwArray::Serialize

```
double rdata[4] = {1.0, 2.0, 3.0, 4.0};
mwArray a(2, 2, mxDOUBLE_CLASS);
a.SetData(rdata, 4);
mwArray b = a.Serialize();
a = mwArray::Deserialize(b); // а должна содержать те же данные,
                             // что были вначале
```

Аргументы. **arr** – mwArray, который был получен вызовом mwArray::Serialize
static double GetNaN(). Получает значение NaN (неопределенность, 0.0/0.0 или Inf-Inf).

```
double x = mwArray::GetNaN();
```

static double GetEps(). Получает значение eps MATLAB. Эта переменная есть расстояние от 1.0 до следующего большего числа с плавающей запятой.

```
double x = mxArray::GetEps();
```

static double GetInf(). Возвращает значение внутренней переменной Inf MATLAB, положительной бесконечности.

```
double x = mxArray::GetInf();
```

static bool IsFinite(double x). Проверка конечности значения. Число конечно, если оно больше чем -Inf и меньше чем Inf. Возвращается true если значение конечно.

```
bool x = mxArray::IsFinite(1.0);    // Возвращает true
```

static bool IsInf(double x). Проверка значения на бесконечность. Возвращает true если значение бесконечно.

static bool IsNaN(double x). Проверка значения на NaN (неопределенность, 0.0/0.0 или Inf-Inf). Возвращается true если значение NaN.

2.5.3. Класс *mwString*

Это простой *класс строк*, используемый API mxArray для передачи строковых данных как ввод/вывод некоторых методов. Конструкторы, методы и операторы требуют подключения заголовочного файла mclcppclass.h:

```
#include "mclcppclass.h"
```

Ниже представлено краткое описание каждого конструктора, метода и операторы и приведены примеры использования в C++.

Конструкторы

mwString(). Создает пустую строку.

```
mwString str;
```

mwString(const char* str). Создание новой строки и инициализация данных строки из представленного символа (с NULL в конце).

```
mwString str("This is a string");
```

mwString(const mwString& str). Конструктор копии для mwString. Создает новую строку и инициализирует ее данные из представленной mwString.

```
mwString str("This is a string");
```

```
mwString new_str(str);    // new_str содержит копию символов из str.
```

Методы

int Length() const. Возвращает число символов в строке.

```
mwString str("This is a string");
```

```
int len = str.Length();    // len должно быть 16.
```

Операторы

operator const char* () const. Используется, чтобы получить прямой доступ только для чтения к строкам данных.

```
mwString str("This is a string");
const char* pstr = (const char*)str;
```

mwString& operator=(const mwString& str). Оператор назначения mwString. Используется для копирования содержания одной строки в другую.

```
mwString str("This is a string");
mwString new_str = str;    // new_str содержит копию данных из str.
```

mwString& operator=(const char* str). Оператор назначения mwString.

```
const char* pstr = "This is a string";
mwString str = pstr;    // str содержит копию данных из pstr.
```

bool operator==(const mwString& str) const. Проверка на равенство двух строк mwStrings

```
mwString str("This is a string");
mwString str2("This is another string");
bool ret = (str == str2);    // ret должно иметь значение false.
```

bool operator!=(const mwString& str) const. Проверка двух строк mwStrings на неравенство.

bool operator<(const mwString& str) const. Сравнивает входную строку с данной строкой и возвращает true если данная строка - лексикографически меньше чем входная строка.

bool operator<=(const mwString& str) const. Сравнивает входную строку с данной строкой и возвращает true если данная строка - лексикографически меньше или равна входной строки.

bool operator>(const mwString& str) const. Сравнивает входную строку с данной строкой и возвращает true если данная строка - лексикографически больше входной строки.

bool operator>=(const mwString& str) const. Сравнивает входную строку с данной строкой и возвращает true если данная строка - лексикографически больше или равна входной строке.

friend std::ostream& operator<<(std::ostream& os, const mwString& str). Используется для печати содержания mwString в ostream.

```
#include <ostream>
mwString str("This is a string");
std::cout << str << std::endl; // на дисплее должно быть напечатано
                                // "This is a string"
```

2.5.4. Класс *mwException*

Это основной *тип исключений*, используемый API mwArray и функциями C++ интерфейса. Все ошибки, созданные в течение запросов к API mwArray и к функциям интерфейса C++, созданным Компилятором MATLAB, считаются как mwExceptions.

Конструкторы

- mwException();
- mwException(const char* msg);

- `mwException(const mwException& e);`
- `mwException(const std::exception& e).`

Методы

- `const char *what() const throw();`

Операторы

- `mwException& operator=(const mwException& e);`
- `mwException& operator=(const std::exception& e).`

Подробности об этом классе см. в документации Компилятора MATLAB.

2.6. Внешние интерфейсы

MATLAB® обеспечивает *интерфейсы к внешним подпрограммам*, написанным в других языках программирования, данным, которые должны совместно использоваться с внешними подпрограммами, клиентами или серверами, общающимися через COM-объекты или динамический обмен данными (DDE). Эти возможности называют интерфейсами прикладного программирования MATLAB (API), или внешними интерфейсами. В этом параграфе рассмотрим некоторые функции внешних интерфейсов, которые созданы для их использования в языке C. Более подробная информация о внешних интерфейсах содержится в документации, MATLAB/External Interfaces, см. также MATLAB/C and Fortran Functions – By Category.

Напомним, что MATLAB работает с только единственным типом объекта: массивом MATLAB. Все переменные MATLAB, включая скаляры, векторы, матрицы, строки, массивы ячеек, структуры, и объекты, сохраняются как массивы MATLAB. Для взаимодействия языков программирования C и MATLAB для языка C создан соответствующий типу MATLAB тип *данных* `mxArray` C и разработаны методы его использования. Это реализовано в заголовочном файле `matrix.h`, а также в файлах `mcclcppclass.h` и `mwutil.h` из каталога `<MatlabRoot>\extern\include\`. Поэтому использование типа данных `mxArray` и функций в программах на C требует подключения заголовочных файлов **`matrix.h`**, **`mcclcppclass.h`** и **`mwutil.h`**.

В языке C массив MATLAB декларируется как тип `mxArray`. Структура типа `mxArray` содержит следующую информацию: тип массива; его измерения; данные ассоциированные с этим массивом; если массив числовой, является ли он вещественным или комплексным; если массив разрежен, его индексы и максимальное число отличных от нуля элементов; если структура или объект, то число полей и имена полей.

Напомним также некоторые особенности MATLAB. Все данные MATLAB сохраняются в постолбцовом порядке, эта традиция происходит от ФОРТРАН-а. 1-на-1 структура сохраняется тем же самым способом, как 1-на-n массив ячеек, где n является числом полей в структуре. Элементы массив 1-на-n называют полями. Каждое поле имеет имя, сохраненное в `mxArray`. Объекты подобны структурам. В MATLAB, объектами называют структуры с зарегистрированными методами. Вне MATLAB, объект – есть структура, которая содержит память для дополнительного имени класса, который идентифицирует название объекта.

2.6.1. Процедуры доступа к МАТ-файлам

Как известно, для сохранения массивов MATLAB имеется специальный тип файлов – так называемые МАТ-файлы. В МАТ-файле сохраняются не только данные массива MATLAB, но и его структура. Во многих отношениях было бы удобно использовать такой тип хранения в приложениях, разработанных при помощи Компилятора MATLAB. Для использования в языке C такого способа хранения массивов MATLAB имеется ряд функций доступа к МАТ-файлам. Использование этих функций требует подключения внешнего компилятора и использования заголовочного файла `mat.h` из каталога `<MatlabRoot>\extern\include\`:

```
#include "mat.h"
```

Примеры использования этих функций можно найти в документации MATLAB.

Таблица 2.6.1. Процедуры доступа к МАТ-файлам

Функция	Описание
<code>matClose</code>	Закрытие МАТ-файла
<code>matDeleteVariable</code>	Удаление переменной <code>mxArray</code> из <code>mat</code> -файла
<code>matGetDir</code>	Каталог <code>mxArrays</code> в <code>mat</code> -файле
<code>matGetFp</code>	Указатель файла на <code>mat</code> -файл
<code>matGetNextVariable</code>	Чтение следующего <code>mxArray</code> из МАТ-файла
<code>matGetNextVariableInfo</code>	Загрузка только информации заголовка массива
<code>matGetVariable</code>	Чтение <code>mxArrays</code> из МАТ-файлов
<code>matGetVariableInfo</code>	Загрузка только информации заголовка массива
<code>matOpen</code>	Открытие МАТ-файла
<code>matPutVariable</code>	Запись <code>mxArrays</code> в МАТ-файлы
<code>matPutVariableAsGlobal</code>	Помещение <code>mxArrays</code> в МАТ-файлы

2.6.2. Операции с массивами mxArray

Использование этих функций в программах на C/C++ требует подключения заголовочного файла `matrix.h` из каталога `<MatlabRoot>\extern\include\`:

```
#include "matrix.h"
```

Приведем списки функций по категориям для работы с массивами `mxArray`. Примеры использования этих функций даны в следующем параграфе.

Таблица 2.6.2. Функции создания массивов и их удаления

Функция	Описание
<code>mxCreateCellArray</code>	Создание пустого массива <code>mxArray</code> ячеек размерности N
<code>mxCreateCellMatrix</code>	Создание пустого массива <code>mxArray</code> ячеек размерности 2

Таблица 2.6.2. *Функции создания массивов и их удаления (окончание)*

Функция	Описание
<code>mxCreateCharArray</code>	Создание пустого массива <code>mxArray</code> строк размерности <code>N</code>
<code>mxCreateCharMatrixFromStrings</code>	Создание пустого массива строк <code>mxArray</code> размерности 2
<code>mxCreateDoubleMatrix</code>	Создание 2-мерного <code>mxArray double</code> , инициализированного нулями
<code>mxCreateDoubleScalar</code>	Создание скалярного массива <code>double</code> , инициализированного указанным значением
<code>mxCreateLogicalArray</code>	Создание логического <code>mxArray</code> размерности <code>N</code> , инициализированного как <code>false</code>
<code>mxCreateLogicalMatrix</code>	Создание двумерного логического <code>mxArray</code> , инициализированного как <code>false</code>
<code>mxCreateLogicalScalar</code>	Создание скалярного логического <code>mxArray</code> , инициализированного как <code>false</code>
<code>mxCreateNumericArray</code>	Создание пустого числового массива <code>mxArray</code> размерности <code>N</code>
<code>mxCreateNumericMatrix</code>	Создание числовой матрицы <code>mxArray</code> , инициализированной нулями
<code>mxCreateSparse</code>	Создание пустого 2-D разреженного <code>mxArray</code>
<code>mxCreateSparseLogicalMatrix</code>	Создание пустого 2-D разреженного логического <code>mxArray</code>
<code>mxCreateString</code>	Создание 1-на- <code>N</code> <code>mxArray</code> строк, инициализированного указанной строкой
<code>mxCreateStructArray</code>	Создание пустого массива структур <code>mxArray</code> размерности <code>N</code>
<code>mxCreateStructMatrix</code>	Создание пустой матрицы структур <code>mxArray</code>
<code>mxDestroyArray</code>	Освобождение динамической памяти, распределенной <code>mxCreate</code>
<code>mxDuplicateArray</code>	Создание глубокой копии массива
<code>mxRemoveField</code>	Удалите поля из массива структуры

Таблица 2.6.3. *Функции доступа к данным типа `mxArray`*

Функция	Описание
<code>mxGetCell</code>	Содержание ячейки <code>mxArray</code>
<code>mxGetChars</code>	Указатель на символьные данные массива
<code>mxGetClassID</code>	Тип класса <code>mxArray</code>
<code>mxGetClassName</code>	Имя класса <code>mxArray</code> как строка
<code>mxGetData</code>	Указатель на данные
<code>mxGetDimensions</code>	Указатель на массив измерений
<code>mxGetElementSize</code>	Число байтов, требуемых для хранения каждого элемента данных
<code>mxGetEps</code>	Значение <code>eps</code>
<code>mxGetField</code>	Значение поля, данного именем поля и индексом в массиве структур
<code>mxGetFieldByNumber</code>	Значение поля, данного номером поля и индексом в массиве структур

Таблица 2.6.3. Функции доступа к данным типа mxArray (окончание)

Функция	Описание
mxGetFieldNameByNumber	Имя поля, данного номером поля в массиве структур
mxGetFieldNumber	Номер поля, данного именем поля в массиве структур
mxGetImagData	Указатель на мнимые данные mxArray
mxGetInf	Значение бесконечности
mxGetIr	Массив ir разреженной матрицы
mxGetJc	Массив jc разреженной матрицы
mxGetLogicals	Указатель на данные логического массива
mxGetM	Число строк в mxArray
mxGetN	Число столбцов в mxArray
mxGetNaN	Значение неопределенности NaN (Not-a-Number)
mxGetNumberOfDimensions	Число измерений в mxArray
mxGetNumberOfElements	Число элементов в mxArray
mxGetNumberOfFields	Число полей в структуре mxArray
mxGetNzmax	Число элементов в массивах ir, pr и pi
mxGetPi	Мнимые данные элементов mxArray
mxGetPr	Вещественные данные элементов mxArray
mxGetScalar	Вещественная часть первого элемента данных в mxArray
mxGetString	Копирование строки mxArray в строку языка C

Таблица 2.6.4. Функции записи данных в mxArray

Функция	Описание
mxSetCell	Установка значения ячейки mxArray
mxSetClassName	Преобразование массива структур в массив объекта MATLAB
mxSetData	Установка указателя на данные
mxSetDimensions	Изменение числа измерений и размера каждого измерения
mxSetField	Установка поля массива структур по данному имени поля
mxSetFieldByNumber	Установка поля массива структур по данному номеру поля
mxSetImagData	Установка указателя мнимых данных для mxArray
mxSetIr	Задание массива ir разреженного mxArray
mxSetJc	Задание массива jc разреженного mxArray
mxSetM	Задание числа строк в mxArray
mxSetN	Задание числа столбцов в mxArray
mxSetNzmax	Установка размера элементов, отличных от нуля
mxSetPi	Задание новых мнимых данных для mxArray
mxSetPr	Задание новых вещественных данных для mxArray

2.7. Передача значений между C/C++ double, mxArray и mxArray

Компилятор MATLAB имеет два основных типа: mxArray и mxArray. Можно использовать mxArray и mxArray, как новые типы в C/C++ при написании кодов. При создании программ, использующих функции библиотек, созданных Компилятором MATLAB, часто возникает необходимость преобразования значений между C/C++ double, mxArray и mxArray. Для преобразования данных можно использо-

вать конструкторы `mxArray` и `mwArray` и функции `set` и `get` доступа к элементам `mxArray` и `mwArray`. Если функции преобразования данных используются часто, то удобно написать свои функции преобразования в отдельном заголовочном файле и при написании других кодов использовать заготовленные функции, подключая заголовочный файл. В этом параграфе приведем, следуя книге [LePh1], примеры кодов преобразования вещественных массивов. Для комплексных массивов можно написать свои функции, либо обратиться к книге [LePh1].

2.7.1. Преобразование значений между C/C++ *double* и *mxArray*

В этом разделе покажем на примерах как передать значения между типами C/C++ `double` и `mxArray`. В примерах используются следующие функции преобразования данных, которые имеют имена типа:

- `double2mxArray_scalarReal(..)` – преобразование скаляра `double` в скаляр `mxArray`;
- `mxArray2double_scalarReal(..)` – преобразование скаляра `mxArray` в скаляр `double`.

Для преобразования векторов и матриц, нужно вместо символов «`scalarReal`» подставить соответственно: `vectorReal`, `matrixReal`. Коды этих вспомогательных функций приведены в разделе 2.7.4.

Преобразование скаляров

```
double db_scalar = 1.1 ;
mxArray *mx_scalar = NULL ;
mx_scalar = mxCreateDoubleMatrix(1, 1, mxREAL) ;
double2mxArray_scalarReal(db_scalar, mx_scalar) ;
double db_scalarReturn = mxArray2double_scalarReal(mx_scalar) ;
cout << " db_scalarReturn = " << db_scalarReturn << endl ;
mxDestroyArray(mx_scalar) ;
```

Преобразование векторов

```
double db_vector[3] = {1.1, 2.2, 3.3} ;
int vectorSize = 3 ;
/* вектор-строка */
mxArray *mx_vector = NULL ;
mx_vector = mxCreateDoubleMatrix(vectorSize, 1, mxREAL) ;
double2mxArray_vectorReal(db_vector, mx_vector) ;
double *db_vectorReturn = new double [vectorSize] ;
mxArray2double_vectorReal(mx_vector, db_vectorReturn) ;
int i ;
for (i=0; i<vectorSize; i++) {
    cout << db_vectorReturn[i] << endl ;
}
mxDestroyArray(mx_vector) ;
delete [] db_vectorReturn ;
```

Преобразование матриц

```
double db_A[3][3] = {{1.1, 2.2, 3.3}, {4.4, 5.5, 6.6}, {7.7, 8.8, 9.9}};
int row = 3 ;
int col = 3 ;
mxArray *mx_A = NULL ;
    mx_A = mxCreateDoubleMatrix(row, col, mxREAL) ;
    double2mxArray_matrixReal(&db_A[0][0], mx_A) ;
double **db_ReturnA ;
db_ReturnA = new double* [row] ;
int i ;
    for (i=0; i<row; i++) {
        db_ReturnA[i] = new double [col] ;
    }
mxArray2double_matrixReal(mx_A, db_ReturnA) ;
printMatrix(db_ReturnA, row, col) ;
mxDestroyArray(mx_A) ;
delete [] db_ReturnA ;
```

2.7.2. Преобразование значений из C/C++ double в mxArray

В этом разделе покажем на примерах как передать значения от типа C/C++ double к типу mxArray. В примерах используются следующие функции преобразования данных:

- `double2mxArray_matrixReal(..)` – преобразование матрицы double в матрицу mxArray;
- `double2mxArray_matrixComplex(..)` – преобразование комплексной матрицы double в комплексную матрицу mxArray.

Эти функции приведены в разделе 2.7.4.

Преобразование скаляров

```
mxArray mw_scalar(1, 1, mxDOUBLE_CLASS) ;
mw_scalar(1,1) = 1.4 ;
```

или

```
double db_scalar2 = 1.2 ;
mxArray mw_scalar2(1, 1, mxDOUBLE_CLASS) ;
mw_scalar2 = db_scalar2 ;
```

Преобразование комплексных скаляров

```
mxArray mw_scalarComplex(1, 1, mxDOUBLE_CLASS, mxCOMPLEX) ;
mw_scalarComplex(1,1).Real() = 2.2 ;
mw_scalarComplex(1,1).Imag() = 3.3 ;
```

или

```
double db_scalarReal = 4.4 ;
double db_scalarImag = 5.5 ;
mxArray mw_scalarComplex2(1, 1, mxDOUBLE_CLASS, mxCOMPLEX) ;
    mw_scalarComplex2(1,1).Real() = db_scalarReal ;
    mw_scalarComplex2(1,1).Imag() = db_scalarImag ;
```

Преобразование векторов

```
double db_vector[6] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0} ;
int vectorSize = 6 ;
mwArray mw_vector(vectorSize, 1, mxDOUBLE_CLASS) ;
mw_vector.SetData(db_vector, vectorSize) ;
```

Преобразование комплексных векторов

```
double realdata[4] = {1.0, 2.0, 3.0, 4.0};
double imagdata[4] = {10.0, 20.0, 30.0, 40.0};
int aSize = 4 ;
mwArray mw_vectorComplex(aSize, 1, mxDOUBLE_CLASS, mxCOMPLEX);
mw_vectorComplex.Real().SetData(realdata, aSize);
mw_vectorComplex.Imag().SetData(imagdata, aSize);
```

Преобразование матриц

```
int i, j ;
double db_matrix[3][2] = { {1.0, 2.0} , {3.0, 4.0}, {5.0, 6.0} } ;
int arow = 3 ;
int acol = 2 ;
mwArray mw_matrix = double2mwArray_matrixReal(&db_matrix[0][0], arow,
acol) ;
std::cout << mw_matrix << std::endl ;
или, если матрица в двойном указателе,
double **db_matrixA ;
db_matrixA = new double*[arow] ;
for(i=0; i<arow; i++) {
    db_matrixA[i] = new double [acol] ;
}
for (i=0; i<arow; i++) {
    for (j=0; j<acol; j++) {
        db_matrixA[i][j] = 1.2 + i+j ; // assign a number
    }
}
mwArray mw_matrixA = double2mwArray_matrixReal(db_matrixA, arow, acol) ;
std::cout << mw_matrixA << std::endl ;
delete [] db_matrixA ;
```

2.7.3. Преобразование значений из mwArray в C/C++ double

В этом разделе покажем на примерах как передать значения от типа mwArray к типу C/C++ double. В примерах используются следующие функции преобразования данных:

- `mwArray2double_vectorReal(..)` – преобразование вектора `mwArray` в вектор `double`;
- `mwArray2double_matrixReal(..)` – преобразование матрицы `mwArray` в матрицу `double`;

Коды функций приведены в конце параграфа.

Преобразование скаляров

```
mwArray mw_scalar(1, 1, mxDOUBLE_CLASS) ;
mw_scalar(1,1) = 1.4 ;
double db_scalar = (double) mw_scalar(1,1) ;
```

Комплексный скаляр

```
mwArray mw_scalarComplex(1, 1, mxDOUBLE_CLASS, mxCOMPLEX) ;
mw_scalarComplex(1,1).Real() = 2.2 ;
mw_scalarComplex(1,1).Imag() = 3.3 ;
double db_scalarReal = (double) mw_scalarComplex(1,1).Real() ;
double db_scalarImag = (double) mw_scalarComplex(1,1).Imag() ;
```

Преобразование векторов

```
/* предположим, что вектор mw_vector уже имеет значения */
int vectorSize = 6 ;
double *db_vector2 = new double[vectorSize] ;
mwArray2double_vectorReal(mw_vector, db_vector2) ;
for (i=0; i<vectorSize; i++) {
    cout << db_vector2[i] << endl ;
}
delete [] db_vector2 ;
```

Преобразование матриц

```
int arow = 3 ;
int acol = 2 ;
double **db_matrixA = new double* [arow] ;
for (i=0; i<arow; i++) {
    db_matrixA[i] = new double [acol] ;
}
...
/* предположим, что матрица mw_matrix уже имеет значения */
mwArray2double_matrixReal(mw_matrix, db_matrixA) ;
delete [] db_matrixA ;
```

2.7.4. Вспомогательные функции преобразования данных

В этом разделе приведем коды вспомогательных функций преобразования данных, использованные в примерах выше. Как уже отмечалось, эти коды можно записать в заголовочный файл, который можно подключать к основной программе, где будут использоваться данные функции. Приведенные коды требуют подключения следующих заголовочных файлов:


```
#include "mclcppclass.h"
#include "mwutil.h"
```

Преобразование значений из C/C++ double в mxArray

Преобразование скаляра C/C++ double в вещественный mxArray

```
void double2mxArray_scalarReal (double cpp, mxArray* mx_pointer) {
double db_bufx[1] ;
db_bufx[0] = cpp ;
    memcpy( mxGetPr(mx_pointer), db_bufx, 1*sizeof(double) ) ;
}
```

Преобразование скаляров C/C++ double в комплексный mxArray

```
void double2mxArray_scalarComplex (double cppReal, double cppImag,
mxArray* mx_pointer) {
double db_bufReal[1] ;
double db_bufImag[1] ;
db_bufReal[0] = cppReal ;
db_bufImag[0] = cppImag ;
    memcpy( mxGetPr(mx_pointer), db_bufReal, 1*sizeof(double) ) ;
    memcpy( mxGetPi(mx_pointer), db_bufImag, 1*sizeof(double) ) ;
}
```

Преобразование вектора C/C++ double в вещественный mxArray

```
void double2mxArray_vectorReal (double* db_vector, mxArray*
mx_pointer)
{
int row = (int)mxGetM(mx_pointer) ; /* число строк */
int col = (int)mxGetN(mx_pointer) ; /* число столбцов */
int vectorSize ;
    if ( row > col ) { vectorSize = row ;}
    else { vectorSize = col ;}
    memcpy(mxGetPr(mx_pointer), db_vector, vectorSize*sizeof(double));
}
```

Преобразование C/C++ double матрицы в вещественный mxArray

```
void double2mxArray_matrixReal(double** db_matrix, mxArray* mx_pointer) {
    int row = (int)mxGetM(mx_pointer) ; /* число строк */
    int col = (int)mxGetN(mx_pointer) ; /* число столбцов */
double* db_vector;
db_vector = new double[row*col];
int i, j, index ;
    for(j=0; j<col; j++) {
        for(i=0; i<row; i++) {
            index = j*row + i;
            db_vector[index] = db_matrix[i][j];
        }
    }
}
```

```
memcpy(mxGetPr(mx_pointer), db_vector, row*col*sizeof(double));
delete[] db_vector ;
}
```

Преобразование матрицы C/C++ double в вещественный mxArray

```
void double2mxArray_matrixReal(double* addressMatrix00, mxArray*
mx_pointer) {
int row = (int)mxGetM(mx_pointer); /* число строк */
int col = (int)mxGetN(mx_pointer); /* число столбцов */
/* назначение памяти для буфера */
int i, j ;
double **db_matrixbuf ;
db_matrixbuf = new double*[row] ;
    for(i=0; i<row; i++) {
        db_matrixbuf[i] = new double [col] ;
    }
/*установка адреса для строк */
    for(i=0; i<row; i++) {
        db_matrixbuf[i] = addressMatrix00 + i*col ;
    }
double* db_vector ;
db_vector = new double [row*col] ;
int index ;
    for(j=0; j<col; j++) {
        for(i=0; i<row; i++) {
            index = j*row + i ;
            db_vector[index] = db_matrixbuf[i][j] ;
        }
    }
memcpy(mxGetPr(mx_pointer), db_vector, row*col*sizeof(double));
delete[] db_vector ;
delete[] db_matrixbuf ;
}
```

Преобразование значений из mxArray в C/C++ double

Преобразование вещественного mxArray в скаляр C/C++

```
double mxArray2double_scalarReal (mxArray* mx_pointer) {
double db_scalar = mxGetScalar(mx_pointer);
return db_scalar ;
}
```

Преобразование комплексного mxArray в скаляры C/C++ double

```
void mxArray2double_scalarComplex (mxArray* mx_pointer,
double &db_scalarReal, double &db_scalarImag) {
double* bufferReal ;
bufferReal = (double *)mxGetPr(mx_pointer) ;
db_scalarReal = bufferReal[0] ;
```

```
double* bufferImag ;
if( mxGetPi(mx_pointer)!= NULL ) {
    bufferImag = (double *)mxGetPi(mx_pointer) ;
    db_scalarImag = bufferImag[0] ;
}
else {
    db_scalarImag = 0 ;
}
}
```

Преобразование вещественного mxArray в вектор C/C++ double

```
void mxArray2double_vectorReal (mxArray* mx_pointer, double* cpp) {
int i ;
int row = (int)mxGetM(mx_pointer) ; /* число строк */
int col = (int)mxGetN(mx_pointer) ; /* число столбцов */
int vectorSize ;
    if ( row > col ) { vectorSize = row ;}
    else { vectorSize = col ;}
double* buffer ;
buffer = mxGetPr(mx_pointer) ;
    for (i=0; i<vectorSize; i++) {
        cpp[i] = buffer[i] ;
    }
}
```

Преобразование вещественного mxArray в матрицу C/C++ double

```
void mxArray2double_matrixReal (mxArray* mx_pointer, double**
db_matrix) {
int i, j, index ;
int row = (int)mxGetM(mx_pointer); /* число строк */
int col = (int)mxGetN(mx_pointer); /* число столбцов */
double* buffer ;
buffer = mxGetPr(mx_pointer) ;
    for(j=0; j<col; j++) {
        for(i=0; i<row; i++) {
            index = j*row + i ;
            db_matrix[i][j] = buffer[index] ;
        }
    }
}

void printMatrix(double** matrix, int row, int col) {
int i, j;
    for (i=0; i<row; i++) {
        for (j=0; j<col; j++) {
            std::cout << matrix[i][j] << «\t» ;
        }
        std::cout << std::endl ;
    }
}
```

Преобразование из C/C++ double в mxArray

В случае mxArray преобразование скаляров производится непосредственно и не требует вспомогательных функций.

Преобразование матрицы C/C++ double в вещественный mxArray

```
mxArray double2mxArray_matrixReal(double** db_matrix, int row, int col) {
    mxArray mw_matrix(row, col, mxDOUBLE_CLASS) ;
    for(int i=0; i<row; i++) {
        for(int j=0; j<col; j++) {
            mw_matrix(i+1, j+1) = db_matrix[i][j] ;
        }
    }
    return mw_matrix ;
}
```

Преобразование матрицы C/C++ double в вещественный mxArray

```
mxArray double2mxArray_matrixReal(double* addressMatrix00, int row,
int col) {

    /*назначение памяти для буфера */
    int i, j ;
    double **db_matrixbuf ;
    db_matrixbuf = new double*[row] ;
    for(i=0; i<row; i++) {
        db_matrixbuf[i] = new double [col] ;
    }
    /*установка адреса для строк */
    for(i=0; i<row; i++) {
        db_matrixbuf[i] = addressMatrix00 + i*col ;
    }
    /*Преобразование в mxArray */
    mxArray mw_matrix(row, col, mxDOUBLE_CLASS) ;
    for(i=0; i<row; i++) {
        for(j=0; j<col; j++) {
            mw_matrix(i+1, j+1) = db_matrixbuf[i][j] ;
        }
    }
    delete[] db_matrixbuf ;
    return mw_matrix ;
}
```

Преобразование mxArray в C/C++ double

Преобразование вещественного mxArray в вектор C/C++ double

```
void mxArray2double_vectorReal (mxArray mw, double* cpp)
{
    int i ;
    int vectorSize = mw.NumberOfElements() ;
```

```

mwArray dim = mw.GetDimensions() ;
int row = (int) dim(1,1) ;
int col = (int) dim(1,2) ;
/*случай строки */
    if ( row > col ) {
        for (i=0; i<vectorSize; i++) {
            cpp[i] = (double)mw(i+1, 1); }
    }
/*случай столбца */
    else {
        for (i=0; i<vectorSize; i++) {
            cpp[i] = (double)mw(1, i+1); }
    }
}

```

Преобразование вещественного mxArray в матрицу C/C++ double

```

void mxArray2double_matrixReal (mwArray mw_matrix, double** cpp) {
int i,j ;
mwArray dim = mw_matrix.GetDimensions() ;
int row = (int) dim(1,1) ;
int col = (int) dim(1,2) ;
    for (i=0; i<row; i++) {
        for (j=0; j<col; j++) {
            cpp[i][j] = (double)mw_matrix(i+1, j+1) ;
        }
    }
}

```

Пример создания заголовочного файла

Для удобства программирования можно записать эти коды преобразования в заголовочный файл, который затем нужно подключать к основной программе, где используются данные функции. Приведем пример создания такого файла mwConvert.h. Для краткости, будут представлены только две функции преобразования скаляров. Можно в него включить все остальные функции или написать свои вспомогательные функции преобразования.

```

/* mwConvert.h */
/* Преобразование значений между C/C++ double и mxArray */
/* **** */
#include "mclcppclass.h"
#include "mwutil.h"
/* **** */

/* Преобразование матрицы C/C++ double в вещественный mxArray */

mwArray double2mwArray_matrixReal(double** db_matrix, int row, int
col) {
mwArray mw_matrix(row, col, mxDOUBLE_CLASS) ;
    for(int i=0; i<row; i++) {

```

```

    for(int j=0; j<col; j++) {
        mw_matrix(i+1, j+1) = db_matrix[i][j] ;
    }
}
return mw_matrix ;
}
/* ***** */

/* Преобразование вещественного mxArray в матрицу C/C++ double */

void mxArray2double_matrixReal (mxArray mw_matrix, double** cpp) {
    int i,j ;
    mxArray dim = mw_matrix.GetDimensions() ;
    int row = (int) dim(1,1) ;
    int col = (int) dim(1,2) ;
    for (i=0; i<row; i++) {
        for (j=0; j<col; j++) {
            cpp[i][j] = (double)mw_matrix(i+1, j+1) ;
        }
    }
}

```

2.8. Математическая библиотека C++ MATLAB® 6.5

Как уже упоминалось, система MATLAB® версии 6.5 имела *математическую библиотеку* C++. В состав математической библиотеки C++ MATLAB входит приблизительно 400 математических функций MATLAB. Для использования функций математической библиотеки C++ создан специальный класс `mxArray`, объекты которого представляют аналоги в C++ основных типы данных MATLAB. Кроме того, разработан синтаксис, подобный синтаксису языка MATLAB. Поэтому программирование с математической библиотекой MATLAB C++ очень похоже на написание m-файлов в MATLAB. Описание класса `mxArray` и его методов представлено в заголовочных файлах библиотеки в каталоге `<matlab>\extern\include`.

Начиная с выпуска MATLAB 7(R14) математическая библиотека C++ отсутствует полностью как самостоятельный пакет, а ее возможности частично включены в значительно расширенный пакет расширения MATLAB Compiler 4.0. Однако это не означает, что математическая библиотека C++ MATLAB канула в прошлое. Следует иметь ввиду, что использование математической библиотеки C++ для создания независимых приложений дает потрясающие математические возможности и необыкновенную легкость программирования на C++. Для использования математической библиотеки C++ не нужно иметь установленную систему MATLAB 6.5, достаточно иметь саму библиотеку файлов `dll` (файл-архив `mginstaller.exe`) и сопутствующие заголовочные и `lib`-файлы математической

библиотеки. При программировании нужно просто подключить ряд файлов математической библиотеки C++ от MATLAB 6.5 к разрабатываемой программе. Отметим также, что математическая библиотека может использоваться при создании Windows-приложений, тогда как Компилятор MATLAB версии R2007a пока позволяет создавать только консольные приложения.

В данном параграфе рассмотрим очень кратко возможности математической библиотеки C++ от MATLAB 6.5. Более подробно об этом см. [ППС].

2.8.1. Расположение файлов математической библиотеки C++

Математическая библиотека C++ MATLAB состоит из трех наборов файлов: dll-файлы, заголовочные h-файлы и lib-файлы. Кроме того, в MATLAB 6.0 имеется документация по библиотеке. Математическая библиотека MATLAB 6.5 устанавливается в каталогах <matlab>\extern и <matlab>\bin.

Каталог <matlab>\bin. Содержит динамические компоненты библиотеки (DLL), требуемые автономными приложениями C++. Эти файлы входят в свободно распространяемый (вместе с приложением) самораспаковывающийся архив **mglinstaller.exe** с математическими библиотеками (dll), необходимыми для работы автономного приложения. Данный набор файлов обеспечивает работу приложения вне MATLAB. Этот каталог <matlab>\bin должен быть прописан в путях для работы приложений.

Каталог <matlab>\extern\include. Содержит заголовочные файлы и def-файлы для разработки автономных C/C++ приложений. Поскольку математическая библиотека MATLAB C++ содержит математическую библиотеку MATLAB C, то заголовочные файлы matlab.h и matrix.h требуются библиотекой C++. В табл. 2.8.1 приведены некоторые из заголовочных файлов и def-файлы, используемые компиляторами Borland и Microsoft Visual C++. Файлы lib*.def используются MSVC++, а файлы _lib*.def используются Borland.

Таблица 2.8.1. Заголовочные и def-файлы каталога include

Файл	Описание
libmatlb.h	Заголовочный файл, содержащий прототипы для функций встроенной математической библиотеки MATLAB
libmmfile.h	Заголовочный файл, содержащий прототипы для функции библиотеки M-файлов MATLAB
matlab.h	Заголовочный файл для Математической библиотеки C
matrix.h	Заголовочный файл, содержащий определение типа mxArray и прототипы функций для процедур доступа к массивам
_libmat.def	Содержит имена функций, экспортируемых из MAT-файлов DLL
libmat.def	
_libmatlb.def	Содержит имена функций, экспортируемых из встроенной математической библиотеки MATLAB DLL
libmatlb.def	

Таблица 2.8.1. Заголовочные и def-файлы каталога include (окончание)

Файл	Описание
_libmmfile.def	Содержит названия функций, экспортируемых из математической библиотеки DLL m-файлов MATLAB
_libmx.def	Содержит имена функций, экспортируемых из libmx.dll
libmx.def	

Каталог <matlab>\extern\include\cpp. Содержит заголовочные файлы C++ для разработки автономных C++ приложений.

Каталог <matlab>\extern\lib\win32. Содержит библиотеки импорта (*.lib) для разных компиляторов. Поскольку разные компоновщики используют разные файловые форматы, предоставлены lib-файлы для компиляторов: Borland, Microsoft Visual C++ и Watcom. В данном каталоге также находится самораспаковывающийся архив **mglinstaller.exe** с математическими библиотеками (dll), необходимыми для работы автономного приложения.

Каталог <matlab>\extern\examples\cppmath. Содержит исходные тексты тестовых примеров на C++, которые используются для тестирования Компилятора MATLAB 6.5. Примеры обсуждаются в справочном руководстве MATLAB C/C++ Math Library и рассматриваются далее в данной главе. Перечень примеров:

- ex1.cpp – создание массивов и ввод-вывод массива;
- ex2.cpp – вызов библиотечных функций;
- ex3.cpp – передача функций в качестве аргументов;
- ex4.cpp – написание простых функций;
- ex5.cpp – управление исключениями;
- ex6.cpp – использование функций ввода/вывода файлов;
- ex7.cpp – использование функций load() и save();
- ex8.cpp – перевод файла m-файла roots.m в код C++.

Организация каталогов на UNIX вполне аналогична Windows. Назначение файлов, входящих в каталоги, такое же. Файлы отличаются расширением. Вместо расширения *.dll, расширения файлов есть: *.a на IBM RS/6000; *.so на Solaris, Alpha, Linux, and SGI; и *.sl on HP 700.

2.8.2. Документация Математической библиотеки MATLAB C++

Подробная документация для библиотеки состоит из двух электронных книг в форматах HTML и в PDF (в MATLAB 6):

1. MATLAB C++ Math Library User's Guide – руководство пользователя, содержит общую информацию о библиотеке. Оно доступно автономно и через Help MATLAB.
2. MATLAB C++ Math Routine Reference – справка с описаниями всех функций Математической библиотеки C++, доступная автономно и из Help MATLAB. Каждая страница справки по функциям представляет C++

синтаксис функции и имеет ссылку на страницу MATLAB Function Reference для описания соответствующей функции MATLAB.

Для доступа к справочной информации не запуская MATLAB, достаточно открыть HTML-файл `<matlab>/help/mathlib/mathlib.html` или `mathlib_product_page.html` и выбрать, например, MATLAB C++ Math Library User's Guide. На этой странице также доступны:

- MATLAB C Math Library User's Guide;
- MATLAB C Math Routine Reference

Файл `helpdesk.html` каталога `<matlab>/help/` открывает доступ ко всем электронным книгам документации. Отметим, что в MATLAB 6.5 документация библиотеки уже отсутствует.

2.8.3. Знакомство с Математической библиотекой MATLAB C++

Этот раздел содержит краткий обзор Математической библиотеки MATLAB C++. Он дает введение в большинство основных понятий библиотеки.

Типы данных. Подобно MATLAB, в математической библиотеке C++ центральный тип данных – массив. Массивы представлены объектами класса *mwArray*. Числа также представалены массивами 1-на-1. Все программы в Математической библиотеке C++ могут обрабатывать целые числа, числа с плавающей запятой двойной точности, или строки так же легко как массивы. Скаляры или строки автоматически конвертируются в массивы перед вызовом функции математической библиотеки.

Каждый объект класса *mwArray* содержит указатель на структуру массива MATLAB. По этой причине, свойства объекта *mwArray* – это множество свойств массива MATLAB. Каждый MATLAB-массив содержит информацию о типе, размере (количество элементов) и форме (число строк, столбцов, и страниц) этого массива. Для комплексных чисел создается два массива данных. Первый массив хранит вещественную часть данных массива, и второй массив хранит мнимую часть. Для массивов без мнимой части, второй массив отсутствует. Данные в массиве размещаются столбцеобразно, а не по строкам.

Класс *mwArray* имеет небольшой интерфейс. Большинство функций математической библиотеки C++ не являются членами класса *mwArray*.

Подобно программам в MATLAB, большинство операторов и функции, вызываемых в математической библиотеке MATLAB C++, возвращают заново назначенный массив.

Операторы. Математическая библиотека поддерживает часть операторов, доступных в MATLAB. Библиотека предоставляет все операторы отношения, арифметические и смешанные операторы, которые не нарушают правила синтаксиса C++. Операторы, которые не доступны как операторы, доступны путем вызова функции. В MATLAB есть два класса арифметических операторов: операторы массива и матричные операторы. В математической библиотеке C++ арифмети-

ческие операторы – это матричные операторы, за исключением + и -. Это означает, что, например, $A*B$ является матричным умножением A на B , а не поэлементным произведением A и B . Все арифметические операторы над массивами также доступны путем вызова функции.

Операторы в математической библиотеке C++ являются векторизованными. Это означает, что можно использовать, например, оператор + для вычисления суммы двух массивов, не используя цикл.

Функции. Математическая библиотека C++ содержит более чем 400 математических функций и набор утилит. Математические функции – это C++ версии функций MATLAB, тогда как утилиты обеспечивают услугами, например, распечатка и управление памятью.

В отличие от функций C++, функции MATLAB могут иметь несколько возвращаемых значений. Математическая библиотека обеспечивает множественные возвращаемые значения, требуя передачи всех возвращаемых значений кроме первого в функцию как выходных параметров. В списке аргументов функции, выходные параметры всегда предшествуют входным параметрам. Например, функция MATLAB `[V, D] = eig(X)` принимает вид в Математической библиотеке C++ как

```
V = eig (&D,X);
```

Большинство встроенных функций MATLAB и операторов векторизованы, т.е. они оперируют полными массивами. Это верно также для всех программ в Математической библиотеке C++. Например, чтобы вычислить квадратный корень всех элементов в массиве, не нужно делать цикл по элементам массива. Вместо этого, нужно вызвать `sqrt()` на весь массив, функция `sqrt()` сама сделает цикл.

Ввод и вывод. MATLAB программы используют утилиты `fscanf()` и `fprintf()` для чтения и записи при вводе/выводе и `load()` и `save()` для чтения и записывания переменных типа массив из `mat`-файла или в `mat`-файл. Математическая библиотека поддерживает MATLAB-стиль функций `fscanf()` и `fprintf()` ввода и вывода наряду с `load()` и `save()`, а также обеспечивает необходимые операторы для C++ потокового ввода и вывода.

Управление памятью. Пользователи MATLAB обычно не волнуются об управлении памятью, потому что интерпретатор MATLAB управляет памятью за них. Этим MATLAB отличается от большинства языков программирования. Математическая библиотека MATLAB C++ также использует схему управления памятью, которая гарантирует отсутствие расхода памяти.

2.8.4. Работа с массивами `mwArray`

Математическая библиотека C++ использует один класс `mwArray` для представления всех типов массивов MATLAB. Чтобы использовать процедуры и функции математической библиотеки C++, необходимо передавать им данные в форме массива `mwArray`. Синтаксис операций с массивами `mwArray` очень близок

к синтаксису в MATLAB. Математическая библиотека C++ поддерживает следующие типы MATLAB-массивов:

- числовые массивы, элементы которых представлены в формате двойной точности. Все арифметические функции математической библиотеки оперируют с числовыми массивами;
- разреженные массивы, когда сохраняются только отличные от нуля элементы массива;
- массивы символов;
- массивы ячеек;
- структуры, это одномерный массив ячеек, где каждой ячейке присвоено имя.

Каждый экземпляр класса `mwArray` содержит информацию о типе массива, его размере (длине) и виде, а также содержит данные, хранящиеся в массиве. Этот класс, подобно любому другому C++ классу, определяет набор конструкторов. Всякий раз, когда создается переменная этого типа, вызывается один из этих конструкторов.

Рассмотрим немного подробнее числовые массивы.

Числовые массивы

Функции и процедуры математической библиотеки C++ для создания числовых массивов и выполнения с ними некоторых задач указаны в табл. 2.8.2. Для более детальной информации об использовании числовых массивов и о любой из процедур библиотеки, см. MATLAB C++ Math Library Reference.

Таблица 2.8.2. Процедуры для работы с числовыми массивами

Функция	Назначение
<code>mwArray A;</code>	Конструктор по умолчанию. Создание неинициализированного массива
<code>empty()</code>	Создание пустого ([]) массива
<code>mwArray(double)</code>	Конструктор скаляра <code>mwArray</code> . Создание инициализированного скалярного массива (1-на-1) из числа с плавающей точкой двойной точности
<code>mwArray(int)</code>	Создание инициализированного скалярного массива (1-на-1) из целого числа
<code>mwArray(int, int, double*, double*)</code>	Матричный конструктор Создание инициализированного массива m-на-n (матрицы) из действительных, целых или беззнаковых коротких данных (unsigned short data)
<code>mwArray(int, int, unsigned short*, unsigned short*)</code>	
<code>mwArray(mxArray *)</code>	Копирование существующего объекта <code>mwArray</code>
<code>mwArray(const mwArray&)</code>	Копирование существующего массива <code>mwArray</code> (возвращаемо-го процедурами Математической библиотеки или процедурами MATLAB API)

Таблица 2.8.2. Процедуры для работы с числовыми массивами (окончание)

Функция	Назначение
<code>mwArray(int, int, int)</code>	Создание 1-на-n равномерно возрастающей последовательности
<code>mwArray(const mwSubArray&)</code>	Создание <code>mwArray</code> из подмассива
<code>horzcat()</code>	Создание массива m-на-n путем объединения существующих массивов
<code>vertcat()</code>	
<code>cat()</code>	Создание массива размерности больше двух (m-на-n-на-p-на-...)
<code>ones()</code>	Создание массивов размерности больше двух (m-на-n-на-p-на-...) из единиц, нулей или случайных чисел
<code>zeros()</code>	
<code>rand(), randn()</code>	Создание единичной матрицы или магического квадрата
<code>eye()</code>	
<code>magic()</code>	

Числовой массив `mwArray` можно создать в C++ программе следующими способами:

- используя конструктор `mwArray`;
- используя процедуру создания массива;
- вызывая арифметическую программу;
- объединяя существующие массивы;
- присваивая значения элементам массива.

Рассмотрим использование каждого из этих механизмов, уделяя особое внимание тем местам, где синтаксис C++ значительно отличается от соответствующего синтаксиса MATLAB.

Создание массивов конструкторами C++. Когда объявляется объект `mwArray`, как например,

```
mwArray A;
```

то вызывается заданный по умолчанию конструктор, который создает неинициализированный массив. Отметим, что неинициализированный массив нельзя передавать в процедуру математической библиотеки C++, ему нужно присвоить значение перед использованием. Отметим, также, что конструктором `mwArray` невозможно создать массив размерности большей двух. Для создания многомерных массивов используется процедура создания массива или объединения `cat()`.

Пример 1. Использование матричного конструктора `mwArray` для создания матрицы 2-на-3, инициализированной значениями C++ массива чисел двойной точности. Можно также использовать массивы C++ целых или коротких без знаковых (`unsigned short`) значений, чтобы инициализировать `mwArray`. Этот конструктор может по выбору брать второй массив C++, для инициализации мнимой части массива комплексных чисел. Пример показывает использование конструктора копий `mwArray` для создания копии массива 2-на-3.

```
double data[] = {1,4,2,5,3,6,7,8};
mwArray C(2, 4, data); // матричный конструктор
mwArray D(C);          // создание копии C
```

Использование процедуры создания массивов. Математическая библиотека MATLAB C++ имеет ряд процедур, которые создают наиболее часто используемые MATLAB массивы:

- `ones()` – массив, заполненный единицами;
- `zeros()` – массив, заполненный нулями;
- `empty()` – пустой массив;
- `eye()` – единичная матрица (размерности 2);
- `rand()` – случайные числа;
- `randn()` – нормально распределенные случайные числа;
- `magic()` – магический квадрат (размерности 2).

При вызове этих процедур задаются размеры массива. При задании n параметров, программы возвращают многомерный массив размерности n . Например, следующий фрагмент кода создает массив нормально распределенных случайных чисел размерности 2-на-3-на-3.

```
mwArray B;  
B = randn(2,3,3); // Создание трехмерного массива
```

Замечание. Массив нулевой размерности считается пустым массивом.

Вызов арифметических процедур MATLAB. Как и в MATLAB, большинство операторов и функций в математической библиотеке C++ создает по крайней мере один новый результирующий массив. Например, когда умножаются два массива, в результате получается новый массив. Следующий код программы демонстрирует как операция умножения массива 4-на-4 из единиц на случайную матрицу 4-на-4 создает новый массив C.

```
mwArray A, B;  
A = ones(4);  
B = randn(4);  
mwArray C = A * B;
```

C – это новый массив, результат умножения.

Использование объединения. В MATLAB, оператор объединения (`[]`) выполняет обе операции *вертикального* и *горизонтального* объединения. Математическая библиотека C++ использует две процедуры, чтобы эмулировать этот оператор:

- `horzcat()` – горизонтальное объединение массивов;
- `vertcat()` – объединение массивов по вертикали.

Например, можно горизонтально объединить скаляры 1, 2, 3 и 4 в вектор:

```
mwArray A;  
A = horzcat( 1, 2, 3, 4 );
```

Чтобы создать матрицу в программе C++, нужно использовать `vertcat()`,

```
mwArray A;  
A = vertcat( horzcat(1, 2), horzcat(3, 4) );
```

Здесь использован вложенный вызов `horzcat()` для создания строк. Этот фрагмент дает:

```
A = [  
    1    2 ;  
    3    4  
    ]
```

Функции `horzcat()` и `vertcat()` работают с векторами и двумерными массивами так же как со скалярами. Например, следующий фрагмент программы объединяет двумерные массивы `A` и `B` для создания двумерного массива `C`.

```
mwArray A = vertcat( horzcat(1, 2), horzcat(3, 4) );
mwArray B = vertcat( horzcat(5, 6), horzcat(7, 8) );
mwArray C = vertcat( A, B );
```

Горизонтально соединяемые массивы должны иметь одинаковое число строк; вертикально соединяемые массивы должны иметь одинаковое число столбцов.

Процедуры `horzcat()` и `vertcat()` нельзя использовать для создания массивов размерности больше двух. Для высших размерностей используется процедура `cat()`. Она имеет следующий синтаксис

```
mwArray B = cat(dim,A1,A2...)
```

где `A1`, `A2`, и так далее – объединяемые массивы `mwArray`, а `dim` является размерностью, вдоль которой объединяются массивы. Например, следующий фрагмент программы создает трехмерный массив в C++:

```
mwArray A = vertcat( horzcat(1, 2, 3), horzcat(4, 5, 6) );
mwArray B = vertcat( horzcat(11, 12, 13), horzcat(14, 15, 16) );
mwArray C = cat( 3, A, B );
```

```
cout << "C =" << C << endl;
```

Эта программа выводит следующее:

```
C = [
(:, :, 1) =
[
1     2     3;
4     5     6
]
(:, :, 2) =
[
11    12    13;
14    15    16
]
]
```

Если число измерений массива, определяемое в `dim`, больше чем число массивов, определяемых в качестве параметров, `cat` автоматически добавляет индекс 1 между необходимыми измерениями.

Использование присваивания. Можно создать скалярные массивы, используя оператор присваивания (`=`) в C++. Например, следующий C++ код программы создает названный массив `A` и присваивает ему величину 5.

```
mwArray A = 5;
```

Результат этого назначения есть 1-на-1 массив (одна строка, один столбец) содержащий единственное число 5.0, представленного в формате двойной точности с плавающей запятой.

Можно создавать многомерные массивы, используя индексные операторы присваивания, задавая значение в заданном месте массива. Библиотека создает массив (или расширяет уже существующий массив) определяя значение в указанном месте и заполняя нулями остальные элементы массив. Например, следующий фрагмент кода C++

```
mwArray H;
H(2,2,2) = 5;
```

создает трехмерный массив, в котором все элементы нулевые, кроме $H(2,2,2) = 5$.

Замечание. Нельзя одновременно объявлять массив и присваивать ему значение. Оператор `mwArray H(2,2,2) = 5` некорректен.

Создание массива из данных. Можно создать массив из данных стандартного C++ массива. Для этого используется матричный конструктор `mwArray`, в котором нужно указать размер и вид массива. Для комплексных чисел мнимая часть передается конструктору в отдельном C++ массиве.

Когда массив `mwArray` инициализируется из стандартного C++ массива, данные массива C++ сохраняются в столбцовом порядке. Например, чтобы создать массив 2-на-3, содержащий 1 2 3 в первой строке и 4 5 6 – во второй, нужно использовать шестиэлементный одномерный массив C++ с элементами, перечисленными по столбцам:

```
static double data[] = { 1, 4, 2, 5, 3, 6 };
mwArray A(2, 3, data);
```

Этот способ не очень удобен и может привести к ошибкам. Предпочтительнее использовать функцию `row2mat()`, которая создает матрицу из строки C++ массива. Например, ту же матрицу `A` можно получить следующим образом:

```
static double data[] = { 1, 2, 3, 4, 5, 6 };
mwArray A = row2mat(2, 3, data);
```

Функция `row2mat` имеет дополнительный четвертый параметр для того, чтобы создавать комплексные массивы. Четвертый параметр указывает на C++ массив, который содержит мнимые части для комплексных чисел `mwArray`.

При использовании MATLAB-оператора `(:)` вместо некоторого индекса в индексном операторе присваивания, MATLAB заполняет все элементы того измерения, присваивая им заданное значение. Следующий пример C++ использует функцию `colon()` в указании индекса. Пример создает трехмерный массив, заполняя вторую страницу массива заданным значением.

```
mwArray A;
A(colon(),colon(),2) = 4;
```

2.8.5. Подключение математических библиотек к Borland C++ Builder

В файле `<matlab>\extern\examples\cppmath\borland\readme` указаны первые шаги, которые нужно сделать для конфигурирования файлов шаблона проекта для Borland C++Builder 3.0, который содержит компилятор Borland C 5.3 и Borland C++Builder 4.0, который содержит Borland C 5.4, для использования

с Математической библиотекой C/C++. Файлы шаблона проекта сконфигурированы по умолчанию для компилирования примера ex1.cpp (создание и ввод/вывод массивов) из каталога демонстрационных примеров <matlab>\extern\examples\cppmath.

Конфигурирование Borland C++Builder 6.0 и математической библиотеки C++. Для конфигурирования Borland C++Builder 6.0 для работы с математическими библиотеками MATLAB C++ нужно сделать следующие шаги:

1. Открыть в текстовом редакторе файл mk_borland54_libs.bat, который находится в каталоге <matlab>\extern\lib\win32. Отредактировать первую строку файла так, чтобы MATLAB переменная среды указывала на местоположение MATLAB на машине. После изменения файла выполните его из строки DOS в каталоге <matlab>\extern\lib\win32. Это создаст библиотеки импорта (*.lib) из def-файлов, необходимые для компиляции автономных программ.
2. Из опускающегося меню **Project** выбрать **Options**. В появившемся окне выбрать вкладку **Directories/Conditionals**. Добавить в поле **Include** пути для заголовочных файлов математической библиотеки:
 <MATLAB6p5>\extern\include;
 <MATLAB6p5>\extern\include\cpp;
а в поле **Library** добавить пути для lib-файлов математической библиотеки
 <MATLAB6p5>\extern\lib\win32;
 <MATLAB6p5>\extern\lib\win32\borland\bc54
3. К создаваемому проекту необходимо добавить lib-файлы математической библиотеки для Borland. Для этого из опускающегося меню **Project** выбрать **Add to Project** и выбрать файлы _libmat.lib, _libmatlb.lib, _libmmfile.lib, _libmx.lib и libmatpb54.lib из каталога <MATLAB6p5>\extern\lib\win32\, кроме того, нужно выбрать все lib-файлы из каталога <MATLAB6p5>\extern\lib\win32\borland\bc54.
4. При работе с Borland C++ Builder 6 следует учитывать следующее обстоятельство. Некоторые заголовочные файлы каталога <matlab>\extern\include\cpp математической библиотеки MATLAB имеют слишком длинные для Borland C++ Builder строки (свыше 1200 знаков). Поэтому они не будут полностью прочитаны и при компиляции будут возникать ошибки. Чтобы решить эту проблему, необходимо в любом текстовом редакторе, который поддерживает длинные строки, разбить длинные строки заголовочных файлов математической библиотеки, например, пополам. Кроме того, в файле mltif.h необходимо заменить слово pascal (в строке 11668) на слово pascal_.
5. Теперь можно начать создание проекта с использованием математической библиотеки MATLAB C++.

2.8.6. Примеры приложений использующих математические библиотеки

Приведем два примера из книги [ППС], в которых используются функции математической библиотеки C++ MATLAB и которые демонстрируют простоту программирования с математическими библиотеками.

Чтение, обработка и запись данных

Рассмотрим создание проекта (Project_1) в среде Borland C++ Builder, в котором предусмотрено чтение числовых данных из форматированного текстового файла в формате ASCII, небольшая обработка данных функциями математической библиотеки C++ MATLAB и запись результатов в другой текстовый ASCII-файл.

В исходном текстовом файле данные представлены в виде нескольких столбцов чисел (каналов), разделенных табуляцией. После чтения для каждого канала производится удаление тренда и нахождение максимального и минимального элемента канала. Полученные массивы записываются в файл. Загрузка файла происходит при выборе пункта **Открыть** из меню **Файл**, а запись файла – при выборе пункта **Сохранить**. Кнопка **Вычислить** будет запускать удаление тренда в каналах и вычисление максимумов и минимумов. Также будет предусмотрено отображение данных об исходном сигнале (длина сигнала и число каналов) и результатов вычислений. Добавлением кнопок быстрого вызова будет организована панель инструментов. Дополнительно будет иметься возможность ручного ввода длины сигнала и количества каналов загружаемого сигнала.

Используемые функции математической библиотеки C++ MATLAB:

- fopen – открытие файла для чтения данных;
- fscanf – чтение данных из файла;
- fclose – закрытие файла;
- fprintf – запись данных в файл;
- extractScalar – преобразование mxArray в число;
- size – размеры массива данных;
- mclGetInf – бесконечность, чтение до конца файла;
- ctranspose – транспонирование массива данных;
- detrend – удаление тренда (среднего значения);
- horzcat – горизонтальное объединение;
- vertcat – вертикальное объединение;

Основной файл, где реализована процедура обработки – это Unit_1.cpp. Его листинг с комментариями приведен ниже. Текст листинга и весь созданный проект (Project_1) находится в каталоге Project_1 на прилагаемом компакт-диске. Тестовые сигналы Test_Sig.txt и Test_SigD.txt также находятся в каталоге Project_1 на прилагаемом компакт-диске. Остальные файлы проекта создаются средой разработки автоматически. Глобальные переменные, которые используются, как в данной программе, так и в дальнейшем ее развитии – это:

- NCh0 = 0 – начальное значение числа каналов, тип mxArray;
- NLen0 = 0 – начальное значение длины сигнала, тип mxArray;
- NCh – число каналов, тип mxArray;
- iNCh – целое значение числа каналов, тип int;
- NLen – длина всего сигнала, тип mxArray;
- iNLen – целое значение длины сигнала, тип int;
- Sig – имя сигнала, тип mxArray;
- SigDt – сигнал с удаленным трендом, тип mxArray;

- MaxY, MinY – максимумы и минимумы по строкам, тип `mwArray`;
- DOut – выходной массив, тип `mwArray`.

Листинг файла Unit_1.cpp с комментариями. Сначала приведем вид формы Form1, рис. 2.8.1.

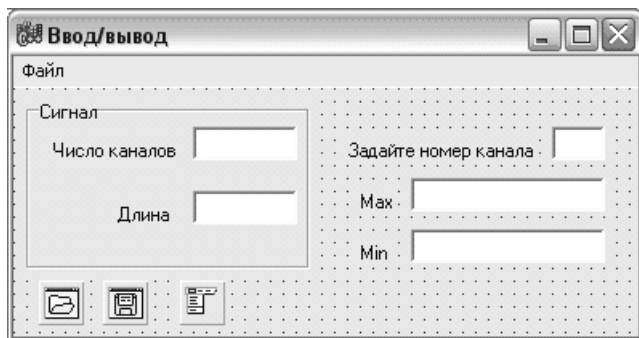


Рис. 2.8.1. Главная форма Form1 проекта Project_1

Начало программы. Первая часть листинга содержит подключение заголовочных файлов `matlab.hpp` и `matrix.h` математических библиотек C/C++ MATLAB.

```
// Unit_1.cpp -----

#include <vcl.h>
#pragma hdrstop
#include "matlab.hpp" //Подключение заголовочных файлов Math
#include "matrix.h"   //математической библиотеки C++

#include "Unit_1.h"
//-----

#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;

// -----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
```

Объявление параметров. В этом разделе объявляются те параметры, которые будут использоваться в программе. Эти переменные уже упоминались ранее. Отметим только еще одну переменную `realdata`, типа `double`, которая будет использоваться для вывода результатов вычислений в текстовые окна формы Edit3 и Edit4.

```

mwArray NCh0 = 0;    // начальное значение числа каналов
mwArray NLen0 = 0;   // начальное значение длины сигнала

mwArray NCh;         // число каналов
int iNCh;            // целое число каналов

mwArray NLen;        // длина всего сигнала
int iNLen;           // целое длина сигнала

mwArray Sig;         // Имя сигнала
mwArray DOut;        // Выходной массив
mwArray Y;
mwArray SigDt;       // Сигнал с удаленным трендом
mwArray MaxY;
mwArray MinY;
double *realdata = new double [2*8]; //Объявление C++ массива для
mwArray

```

Задание числа каналов и длины сигнала. Хотя по умолчанию загружается сигнал из 8 каналов (строк матрицы), в программе предусмотрена загрузка сигналов другой размерности. Для этого нужно в полях Edit1 и Edit2 задать число каналов (строк) и длину сигнала (число столбцов). При этом число каналов (если оно отличается от 8) задавать нужно, а длину – не обязательно. В этом случае сигнал загрузится полностью, до конца. Обратите внимание, что переменная NCh0 имеет тип mwArray и ей присваивается значение, которое получается из функции StrToInt(Edit1->Text).

```

//Задание числа каналов -----

void __fastcall TForm1::Edit1Change(TObject *Sender)
{
    NCh0 = StrToInt(Edit1->Text);
}
// Задание длины сигнала -----
void __fastcall TForm1::Edit2Change(TObject *Sender)
{
    NLen0 = StrToInt(Edit2->Text);
}

```

Открытие файла и чтение данных. Для открытия файла используется диалоговое окно. Сначала из переменных NCh0 и NLen0 типа mwArray извлекаются целые значения iNCh0 и iNLen0 типа int для использования в операторе if, где проверяется, заданы ли эти значения, или принять их по умолчанию: NCh0 = 8, NLen0 = mclGetInf() – чтение до конца. Режим mode чтения файла выбран как «rt» – чтение текстового файла. Имя файла file выбирается через диалоговое окно. Идентификатор файла обозначен переменной v. Чтение файла осуществляется функцией математической библиотеки MATLAB fscanf с флагом %g – числа в компактном виде. После загрузки сигнала вычисляются размерности сигнала, переводятся в целые значения и отображаются в текстовых окнах Edit1 и Edit2 основной формы проекта.

```

void __fastcall TForm1::N2Click(TObject *Sender)
{
    if(OpenDialog1->Execute())
    {
        int iNCh0 = NCh0.ExtractScalar(1); // Извлечение целого из mxArray
        int iNLen0 = NLen0.ExtractScalar(1);

        if (iNCh0 == 0) {NCh0 = 8;}
        if (iNLen0 == 0) {NLen0 = mclGetInf();}

        mxArray mode("rt");
        mxArray file(OpenDialog1->FileName.c_str());

        // Открытие файла и считывание данных
        mxArray v;
        v = fopen(file, mode);
        Sig = fscanf(v, "%g ", horzcat(NCh0, NLen0));
        fclose(v);

        mxArray sizeSig = size(Sig); // Вычисление размерностей сигнала
        NCh = sizeSig(1);             // Число строк
        NLen = sizeSig(2);            // Число столбцов

        iNCh = sizeSig.ExtractScalar(1);
        iNLen = sizeSig.ExtractScalar(2);
        Edit1->Text = IntToStr(iNCh); // Запись в поле Edit1 числа
                                    каналов
        Edit2->Text = IntToStr(iNLen); // Запись в поле Edit2 длины
                                    сигнала
    }
}

```

Отметим, что функция `fscanf` читает данные файла по строкам, а записывает их в массив `mxArray` по столбцам. Поэтому, если первоначальный массив файла (`Test_Sig.txt`) был из 8 столбцов, то загруженный массив `Sig` состоит из 8 строк. Отметим также, что в каталоге проекта имеются несколько тестовых массивов для загрузки: `Test_Sig.txt` и `Test_SigD.txt`.

Применение математических операций. На этом этапе производится удаление тренда (среднего значения) функцией `detrend` математической библиотеки MATLAB. Эта функция удаляет тренд каждого столбца матрицы. Поэтому перед этой операцией делается транспонирование матрицы `Sig`. Функции `max` и `min` также находят максимальной и минимальное значения массива по столбцам. Результат операций `max` и `min` есть две строки по 8 элементов. Эти две строки организуются в матрицу 2-на-8 функцией `vertcat`. Далее из этой матрицы извлекаются вещественные (`double`) значения для возможности их отображения в текстовых окнах `Edit3` и `Edit4` основной формы проекта. Задание номера канала осуществляется в текстовом поле `Edit5`. Максимальное и минимальное значения канала отображается в текстовых окнах `Edit3` и `Edit4` основной формы проекта. Продолжим текст листинга.

```

mxArray X = ctranspose(Sig); // Транспонирование
mxArray Y = detrend(X, "constant"); // Удаление тренда
                                    по столбцам

```

```

mwArray MaxY = max(Y); // Нахождение max элемента по столбцам
mwArray MinY = min(Y); // Нахождение min элемента по столбцам
mwArray SigDt = ctranpose(Y); // Сигнал без тренда

DOut=vertcat(MaxY, MinY); // Образование 2-на-8 матрицы вывода
mwArray TOut = ctranpose(DOut); // Транспонирование для
// преобразования столбцов mwArray в строки C++
TOut.ExtractData(realdata); // Извлечение данных из mwArray
// в realdata
}
}

void __fastcall TForm1::Edit5Change(TObject *Sender)
{
    int k = StrToInt(Edit5->Text);
    Edit3->Text = FloatToStr(realdata[k-1]); // Запись max
    Edit4->Text = FloatToStr(realdata[k+7]); // Запись main
}

```

Запись результатов в файл. Для записи предназначается массив DOut, содержащий максимальные и минимальные значения массива по столбцам. Для записи результата также используется диалоговое окно, в котором можно выбрать имя и каталог записи файла результатов. Режим mode записи файла выбрана как "wt" – запись текстового файла. Имя файла file выбирается через диалоговое окно. Идентификатор файла обозначен переменной fid. Запись файла осуществляется функцией математической библиотеки MATLAB fprintf с флагом %f\t – числа с фиксированной запятой, разделенные табуляцией.

Отметим, что функция fprintf читает данные массива по столбцам, а записывает их в строки, поэтому выходной массив предварительно транспонируется. Далее идет запись по строкам из 8 элементов, разделенных табуляцией.

```

void __fastcall TForm1::N3Click(TObject *Sender)
{
    if(SaveDialog1->Execute())
    {
        mwArray fid;
        mwArray mode("wt");
        mwArray file(SaveDialog1->FileName.c_str());
        fid = fopen(file, mode);
        mwArray TOut = ctranpose(DOut);
        fprintf(fid, "%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t\n", TOut);
        fclose(fid);
    }
}

```

Приведем вид работающей программы Project_1.exe (рис. 2.8.2).

Построение графиков данных mwArray

В данном параграфе мы рассмотрим создание проекта на Borland C++Builder, в котором предусмотрено чтение числовых данных из текстового файла в формате ASCII, математическая обработка и построение графиков полученных число-

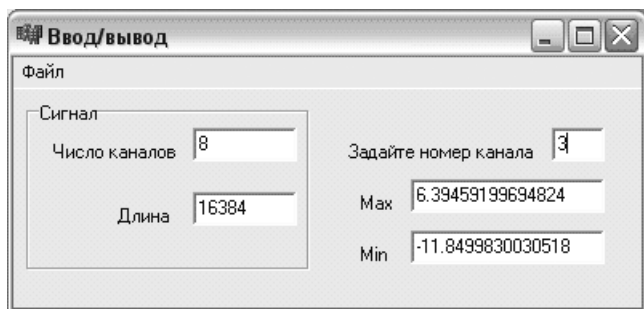


Рис. 2.8.2. Окно программы проекта Project_1

вых массивов. Основная цель параграфа – показать, что графики данных из массива `mwArray` строятся также просто, как и графики на основе массивов C++.

За основу возьмем проект, созданный в предыдущем разделе, поэтому первая часть (загрузка данных, обработка и сохранение результатов) опущена. Основной файл, где реализована процедура, есть `Unit_2.cpp`. Его листинг и весь созданный проект (Project_2) находится в каталоге Project_2 на прилагаемом компакт-диске. Здесь мы рассмотрим ту часть программы `Unit_2.cpp`, где строятся графики каналов сигнала. Тестовые сигналы `Test_Sig.txt` и `Test_SigD.txt` также находятся в каталоге Project_2 на прилагаемом компакт-диске.

Сначала приведем вид формы `Form1` проекта (рис. 2.8.3). Как видно из рисунка, на форму были добавлены три компонента: панель `Panel2` расположенная справа и растянутая на всю оставшуюся клиентскую область, размещенная в нижней части полоса прокрутки `ScrollBar1` и окно рисования `PaintBox1`.

Сначала опишем глобальную логическую переменную готовности графика к построению:

```
// Флаг готовности графика к построению  
bool ready = false;
```

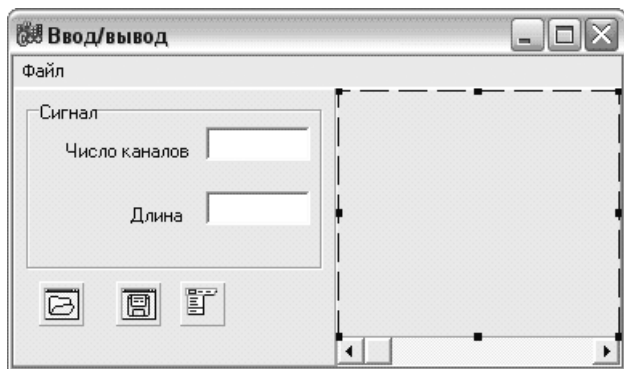


Рис. 2.8.3. Главная форма `Form1` проекта Project_2

Присвоение значения `false` сообщает программе, что первоначальные данные пока не готовы к отображению. В обработчике открытия файла

```
void __fastcall TForm1::N2Click(TObject *Sender)
```

после существующих строк в блоке кода программы условного оператора открытия диалогового окна загрузки

```
if (OpenDialog1->Execute())
{
```

```
    // ... Текст проекта Project_71 ...
```

дополним текст предыдущего проекта нахождением минимальных и максимальных значений загруженных массивов данных, которые потребуются для расчета координат точек в области изображения графика:

```
mwArray SigTr = ctranspose(Sig);
mwArray SigTrDt = detrend(SigTr, "constant");
MaxY = max(SigTrDt); //Нахождение max элемента по столбцам
MinY = min(SigTrDt); // Удаление min элемента по столбцам
SigDt = ctranspose(SigTrDt);
```

Поскольку загруженных данных может оказаться больше чем точек в области рисования, зададим параметры полосы прокрутки, после чего сообщим программе о готовности к изображению графика и обновим окно рисования:

```
if ((iNLen - PaintBox1->Width)<0)
    // если данных оказалось меньше, чем точек помещаемых на экран,
    // максимальное значение положения движка полосы
    // прокрутки равно нулю.
    ScrollBar1->Max = 0;
else
    // иначе, максимальное значение положения движка полосы прокрутки
    // будет равным количеству данных за вычетом тех,
    // которые отображаются на экране. Это необходимо потому,
    // что положение движка определяет начало блока данных
    // для рисования, а индекс не должен выходить за границы массива.
    ScrollBar1->Max = iNLen - PaintBox1->Width;

ready = true;

PaintBox1->Refresh();
}
```

Перейдем к построению графика, задав обработчик события рисования компонента `PaintBox1`:

```
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
```

```
    if (ready) {
        int NN = ScrollBar1->Position;
        // Так как графиков данных будет iNCh штук, то разобьем
        // окно отображения на соответствующее количество частей,
        // не забыв отступить снизу 20 точек для изображения надписей
```

```

// под графиками
int WinH = (PaintBox1->Height-20)/iNCh;

// Так как графиков данных будет iNCh штук, то выбираем поочередно
for (int NCh=1; NCh<=iNCh; NCh++){
    // Выбираем из массива mwArray соответствующие значения
    // максимума и минимума
    mwArray Tmp = MaxY(NCh);
    double mMaxY = Tmp.ExtractScalar(1);
    Tmp = MinY(NCh);
    double mMinY = Tmp.ExtractScalar(1);

    // Делаем соответствующую подпись
    PaintBox1->Canvas->TextOut(2, WinH*(NCh-1),
        "Канал "+IntToStr(NCh));

    // Расчитываем экранные координаты первой изображаемой
    // точки массива и смещаем туда курсор
    Tmp = SigDt(NCh, NN+1);
    int y = floor(WinH*(mMaxY-Tmp.ExtractScalar(1))/(mMaxY-mMinY))+
        WinH*(NCh-1);
    PaintBox1->Canvas->MoveTo(0,y);

    // для оставшихся точек массива расчитываем экранные координаты
    // и проводим замыкающую график линию.
    for(int i=1; i<PaintBox1->Width; i++) {
        Tmp = SigDt(NCh, (NN+i));
        y = floor(WinH*(mMaxY-Tmp.ExtractScalar(1))/(mMaxY-mMinY))+
            WinH*(NCh-1);
        PaintBox1->Canvas->LineTo(i,y);
    }

    // Строим координатные оси
    PaintBox1->Canvas->MoveTo(0,0);
    PaintBox1->Canvas->LineTo(0,PaintBox1->Height-5);
    PaintBox1->Canvas->MoveTo(0,PaintBox1->Height-20);
    PaintBox1->Canvas->LineTo(PaintBox1->Width,
        PaintBox1->Height-20);

    // Ставим подписи на горизонтальной оси
    PaintBox1->Canvas->TextOut(2, PaintBox1->Height-18,
        IntToStr(NN));
    PaintBox1->Canvas->TextOut(PaintBox1->Width-
        PaintBox1->Canvas->TextWidth(IntToStr(NN+
        PaintBox1->Width))-2, PaintBox1->Height-18,
        IntToStr(NN+PaintBox1->Width));
}
}
}

```


Окно программы может быть растянуто, поэтому, по соответствующему событию, необходимо пересчитать заново максимальное значение движка полосы прокрутки.

```
void __fastcall TForm1::Panel2Resize(TObject *Sender)
{
    if ((iNLen - PaintBox1->Width)<0)
        ScrollBar1->Max = 0;
    else
        ScrollBar1->Max = iNLen - PaintBox1->Width;
}
```

Кроме того, нужно сделать обновление окна рисования в случае изменения положения движка полосы прокрутки:

```
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
    PaintBox1->Refresh();
}
```

Приведем вид работающей программы Project_2.exe (рис. 2.8.4).

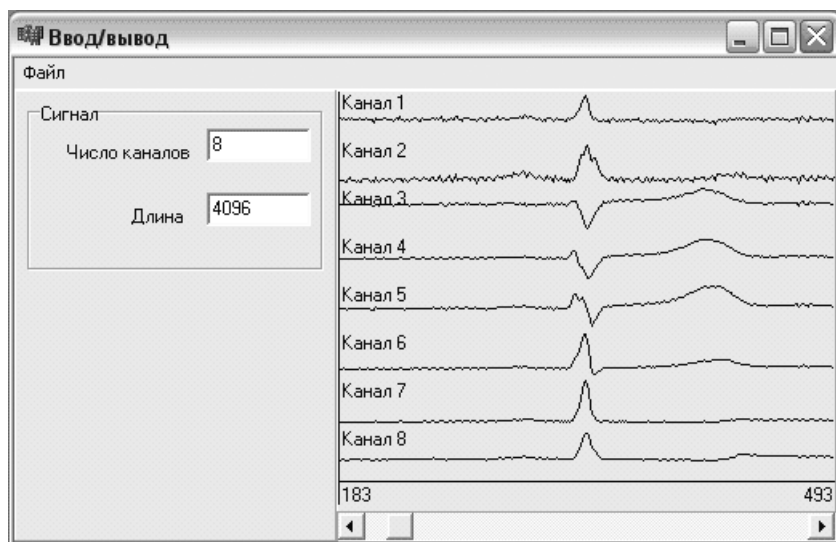


Рис. 2.8.4. Окно программы проекта Project_2

Создание компонентов для Java при помощи Java Builder

3.1. Язык программирования	
Java	172
3.2. Введение в Java Builder	201
3.3. Массивы MATLAB в Java	214
3.4. Примеры приложений	
Java	242
3.5. Некоторые вопросы	
программирования	256
3.6. Среда проектирования	
JBuilder	272
3.7. Примеры создания	
приложений с использованием	
классов Java Builder	278

В данной главе мы рассмотрим создание математических компонентов для Java при помощи пакета Java Builder MATLAB® и приложений, использующих эти компоненты. Сначала мы рассмотрим особенности языка Java. Затем изучим возможности Java Builder, а в конце рассмотрим создание Windows-приложений на JBuilder, в которых используются математические функции, созданные из m-функций MATLAB®. Методы создания приложений для других операционных систем вполне аналогичны, их особенности можно найти в документации MATLAB Builder для Java.

Пакет расширения MATLAB® Builder для Java (также называемый Java Builder) есть расширение пакета MATLAB® Compiler. Java Builder используется для преобразования функций MATLAB® в один или более классов Java, которые составляют компонент Java, или пакет. Каждая функция MATLAB реализуется как метод класса Java и может быть вызвана из приложения Java. Приложения, использующие методы, созданные при помощи Java Builder, при своей работе не требуют установленной системы MATLAB. Однако должна быть установлена MCR – среда выполнения компонентов MATLAB.

3.1. Язык программирования Java

Язык Java создан на основе C++. В некоторых отношениях он является более простым, чем C++. Рассмотрим некоторые особенности этого языка.

Язык Java архитектурно нейтрален, поскольку компилятор генерирует объектный код и делает Java-приложения независимыми от реализации. Это особенно важно для Internet-приложений. Программный код записывается обычным образом в файл с расширением *.java. Для того, чтобы достичь машинной независимости программы, написанной на Java, компилятор языка выполняет перевод программы в промежуточный машинно-независимый код, называемый *байт-кодом* и имеющий расширение *.class. В отличие от exe-приложений, которые выполняются операционной системой Windows, этот байт-код *.class выполняется виртуальной машиной Java (JVM) вне зависимости от платформы. Для выполнения Java-приложения нужно вызвать интерпретатор Java. Например, если консольное приложение имеет байт-код Application1.class, то для его запуска можно выполнить команду

```
java Application1
```

в которой, в случае необходимости нужно добавить аргументы. Одна программа может иметь несколько байт-кодов *.class в соответствии с количеством классов программы.

Таким образом, для работы Java-приложения должна быть установлена (зависимая от платформы) виртуальная машина Java. Она составляет основу среды выполнения Java-программ (Java Runtime Environment, JRE). Загрузка JVM в память для выполнения программы осуществляется утилитой **java** из пакета JDK (Java Development Kit).

Отметим некоторые отличия Java от C++. В Java ликвидировано ручное выделение и освобождение памяти для снижения вероятности ошибок при кодирова-

нии. Нет возможности использовать средства управления памятью C++ для обеспечения быстродействия. В Java отсутствуют арифметические операции с указателями. Массивы Java представляют собой настоящие массивы, а не указатели, как в C++. Используемая в Java модель указателей фактически ликвидирует синтаксис указателей C++. Изменения были внесены для предотвращения случайных нарушений памяти и порчи данных из-за ошибок в арифметических операциях с указателями. Кроме того, размер встроенных типов данных в Java не зависит от компилятора или типа компьютера, как в C++. Типы данных имеют фиксированный размер – скажем, `int` в Java всегда является 32-разрядным числом. Компилятор Java генерирует инструкции байт-кода, которые эффективно преобразуются в набор машинных команд.

Принципиальное отличие между Java и C++ заключается в том, что Java не поддерживает множественного наследования из-за сложностей в управлении иерархиями. Тем не менее, в Java существуют интерфейсы, которые обладают преимуществами множественного наследования без тех затруднений, которые с ним связаны. Классы Java похожи на C++. Тем не менее, все функции в Java (в том числе и `main`) должны принадлежать некоторому классу. В соответствии с требованиями Java для `main` необходимо создать класс-оболочку. В Java нет функций классов, а есть методы, поэтому `main` – это метод, а не функция. Методы Java похожи на функции классов C++, но все же не идентичны им. Например, в Java нет глобальных функций и прототипов функций. Компилятор Java работает в несколько проходов, что позволяет использовать методы до их определения. Более того, функции нельзя передать адрес переменной, поскольку аргументов-указателей и ссылок в Java не существует. Методы Java должны определяться внутри класса. Внешнее определение, как в C++, не допускается.

3.1.1. Основные элементы программирования на Java

В этом разделе рассмотрим очень кратко основы языка Java. Более подробные сведения можно найти в учебниках, например, [Ба], [Пон]. Начнем, по традиции, с простейшей программы.

Первая программа на Java

Отметим сначала особенности языка Java на примере простейшей программы «Hello, World!».

```
class HelloWorld{
public static void main(String[] args){
System.out.println("Hello, World!");
}}
```

Всякая программа представляет собой один или несколько классов, в этом простейшем примере только один класс. Начало класса отмечается служебным словом `class`, за которым следует имя класса, выбираемое произвольно, в данном

случае HelloWorld. Все, что содержится в классе, записывается далее в фигурных скобках и составляет тело класса. Все действия производятся с помощью методов (функций). Один из методов обязательно должен называться `main`, с него начинается выполнение программы. В программе HelloWorld только один метод, а значит, имя ему `main`.

Метод всегда возвращает не более одного значения, тип которого обязательно указывается перед именем метода. Если метод не возвращает никакого значения, играя роль процедуры, как в нашем случае, то вместо типа возвращаемого значения записывается слово **`void`**.

После имени метода в скобках, через запятую, перечисляются аргументы, или параметры метода. Для каждого аргумента указывается его тип и, через пробел, имя. В примере только один аргумент `args`, его тип `String[]` – массив, состоящий из строк символов.

Перед типом возвращаемого методом значения могут быть записаны модификаторы. В примере их два: слово **`public`** означает, что этот метод доступен отовсюду; слово **`static`** обеспечивает возможность вызова метода `main()` в самом начале выполнения программы. Модификаторы вообще необязательны, но для метода `main()` они необходимы.

Тело метода (все, что содержит метод), записывается в фигурных скобках. Единственное действие, которое выполняет метод `main()` в примере, заключается в вызове другого метода с именем `System.out.println` и передаче ему на обработку одного аргумента, текстовой константы "Hello, world!". Текстовые константы записываются в кавычках, которые являются только ограничителями и не входят в состав текста.

Действие метода `System.out.println()` заключается в выводе своего аргумента в выходной поток, связанный обычно с выводом на экран текстового терминала. После вывода курсор переходит на начало следующей строки экрана, на что указывает окончание `ln`, слово `println` – сокращение слов `print line`.

Программа может быть написана в любом текстовом редакторе. Ее надо сохранить в файле, имя которого должно совпадать с именем класса, содержащего метод `main()` и дать имени файла расширение `java`. Система исполнения Java будет находить метод `main()` для начала работы, отыскивая класс, совпадающий с именем файла. В нашем примере, сохраним программу в файле с именем `HelloWorld.java` в текущем каталоге. Затем вызовем компилятор, передавая ему имя файла в качестве аргумента:

```
javac HelloWorld.java
```

Компилятор создает байт-код, файл с именем `HelloWorld.class`, и записывает этот файл в текущий каталог. Осталось вызвать интерпретатор `java`, передав ему в качестве аргумента имя класса (а не файла, расширение `class` при вызове интерпретатора не указывается):

```
java HelloWorld
```

На экране появится:

```
Hello, World!
```

Комментарии и имена

Комментарии вводятся обычным образом: за двумя наклонными чертами подряд `//`, без пробела между ними, начинается *комментарий*, продолжающийся до конца строки; за наклонной чертой и звездочкой `/*` начинается комментарий, который может занимать несколько строк, до звездочки и наклонной черты `*/` (без пробелов между этими знаками).

Для создания документации к JDK в Java введены комментарии третьего типа, а в состав JDK – программа `javadoc`, извлекающая эти комментарии в отдельные файлы формата HTML и создающая гиперссылки между ними. За знаком `/**` начинается комментарий, который может занимать несколько строк и закрывается знаком `*/`. Такой комментарий обрабатывается программой `javadoc`. В него можно вставить указания программе `javadoc`, которые начинаются с символа `@`. Например,

```
/**
 * Начальная программа всех языков программирования
 * @author Неизвестный
 * @version 1.0
 */
class HelloWorld{
public static void main(String[] args){
System.out.println("Hello, World!");
}}
```

Звездочки в начале строк не имеют никакого значения, они написаны просто для выделения комментария.

Имена переменных, классов, методов и других объектов могут быть простыми (идентификаторы) и составными. Идентификаторы в Java состояются из букв и арабских цифр 0–9, причем идентификатор должен начинаться с буквы.

Составное имя – это несколько идентификаторов, разделенных точками, без пробелов, например, уже встречавшееся нам имя `System.out.println`.

Язык Java различает строчные и прописные буквы. Свои имена можно записывать как угодно, но нужно учитывать следующие правила:

- имена классов начинаются с прописной буквы; если имя содержит несколько слов, то каждое слово начинается с прописной буквы;
- имена методов и переменных начинаются со строчной буквы; если имя содержит несколько слов, то каждое следующее слово начинается со строчной буквы;
- имена констант записываются полностью прописными буквами; если имя состоит из нескольких слов, то между ними ставится знак подчеркивания.

Константы

Постоянные величины, которые не изменяются в ходе выполнения программы, называются *константами*. Константы могут быть любого типа, который допустим Java. Перечислим их:

- целые константы, их можно записывать в трех системах счисления: в десятичной, восьмеричной и шестнадцатеричной, хранятся в формате типа `int`;
- действительные константы с фиксированной точкой и с плавающей точкой;

- символьные константы и управляющие символы;
- логические.

Символьные константы. Они представляют собой индексы таблицы символов Unicode. Символьные константы отмечаются апострофами, например, 'A'. Символы хранятся в формате типа char. Управляющие символы записываются в апострофах с обратной наклонной чертой:

- '\n' – символ перевода строки (newline) с кодом ASCII 10;
- '\r' – символ возврата каретки (CR) с кодом 13;
- '\f' – символ перевода страницы (FF) с кодом 12;
- '\b' – символ возврата на шаг (BS) с кодом 8;
- '\t' – символ горизонтальной табуляции (HT) с кодом 9;
- '\\ ' – обратная наклонная черта;
- '\" ' – кавычка;
- '\ ' – апостроф.

Код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его не более чем тремя цифрами в восьмеричной системе счисления в апострофах после обратной наклонной черты, например: '\123' – это буква S в кодировке CP1251. Код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы u ровно четырьмя 16-ричными цифрами: '\u0053' – это буква S.

Замечание. Прописные русские буквы в кодировке Unicode занимают диапазон от '\u0410' – заглавная буква А, до '\u042F' – заглавная Я, строчные буквы от '\u0430' – а, до '\u044F' – я. В какой бы форме ни записывались символы, компилятор переводит их в Unicode, включая и исходный текст программы. Компилятор и исполняющая система Java работают только с кодировкой Unicode.

Машинные константы. В языке Java имеется много различных констант (см. документацию, например JBuilder). Отметим некоторые из них:

- положительная бесконечность POSITIVE_INFINITY, возникающая при переполнении положительного значения, отрицательная бесконечность NEGATIVE_INFINITY для типов double и float;
- неопределенность NaN (Not a Number);
- максимальное и минимальное значения MAX_VALUE, MIN_VALUE для различных числовых типов, для double и float константа MIN_VALUE есть машинная точность;
- число PI и число Эйлера E.

Типы данных

Все типы исходных данных, встроенные в язык Java, делятся на две группы: простые типы (primitive types) и сложные, или ссылочные типы (reference types). *Простые типы* (boolean, short, int, long, char, float и double) принимают единственное число, символ или одно состояние. *Ссылочные типы* предназначены для хранения более одного значения и делятся на массивы (arrays), классы (classes), интерфейсы (interfaces) и строки (String).

Простые типы делятся на логические (boolean) и числовые (numeric).

К числовым типам относятся целые и вещественные типы. Целых типов пять: byte, short, int, long, char. Вещественных типов два: float и double. Отметим, что символы (char) причисляются к целым типам – это значения символов кодировки Unicode.

На рис. 3.1.1 показана иерархия типов данных Java.

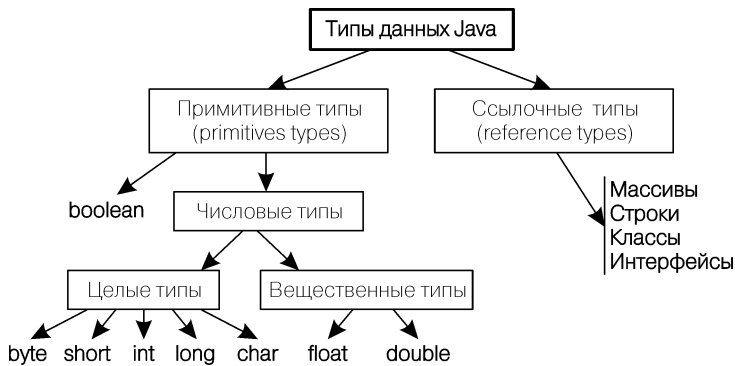


Рис. 3.1.1. Типы данных языка Java

Поскольку по имени переменной невозможно определить ее тип, все переменные обязательно должны быть описаны перед их использованием. Для всех или некоторых переменных можно указать начальные значения после знака равенства, которыми могут служить любые константные выражения того же типа. Описание каждого типа завершается точкой с запятой.

Разберем каждый тип подробнее.

Логический тип (boolean). Логических значений всего два: true (истина) и false (ложь). Значения логического типа boolean возникают в результате различных сравнений, вроде $2 > 3$, и используются чаще всего в условных операторах и операторах циклов. Описание переменных этого типа выглядит так:

```
boolean b = true, bb = false, bool2;
```

Над логическими данными можно выполнять операции присваивания, например, `bool2 = true`, в том числе и составные с логическими операциями, сравнение на равенство `b == bb` и на неравенство `b != bb`, а также логические операции:

- отрицание (NOT) ! (обозначается восклицательным знаком), меняет значение истинности;
- конъюнкция (AND) & (амперсанд), истина, только если оба операнда истинны;
- дизъюнкция (OR) | (вертикальная черта), ложна, только если оба операнда ложны;
- исключающее ИЛИ (XOR) ^ (каре), истинно, только если значения операндов различны.

Кроме перечисленных четырех логических операций есть еще две логические операции сокращенного вычисления:

- сокращенная конъюнкция (conditional-AND) `&&`;
- сокращенная дизъюнкция (conditional-OR) `||`.

Правый операнд сокращенных операций вычисляется только в том случае, если от него зависит результат операции, т. е. если левый операнд конъюнкции имеет значение `true`, или левый операнд дизъюнкции имеет значение `false`. Например, можно записывать выражения `(n != 0) && (m/n > 0.001)` или `(n == 0) || (m/n > 0.001)` не опасаясь деления на нуль.

Строки. Строки символов заключаются в кавычки. Строки могут располагаться только на одной строке исходного кода, нельзя открывающую кавычку поставить на одной строке, а закрывающую – на следующей. Управляющие символы и коды записываются в строках точно так же, с обратной наклонной чертой, но, разумеется, без апострофов, и оказывают то же действие. Для строковых констант определена операция сцепления, обозначаемая плюсом.

" Сцепление " + "строка"

дает в результате строку "Сцепление строк".

Целые типы. Спецификация языка Java, JLS, определяет разрядность (количество байтов, выделяемых для хранения значений типа в оперативной памяти) и диапазон значений каждого типа. Для целых типов они приведены в табл. 3.1.1.

Таблица 3.1.1. Целые типы

Тип	Байт	Диапазон
byte	1	От -128 до 127
short	2	От -32768 до 32767
int	4	От -2147483648 до 2147483647
long	8	От -9223372036854775808 до 9223372036854775807
char	2	От '\u0000' до '\uFFFF', в десятичной форме от 0 до 65535

Хотя тип `char` занимает два байта, в арифметических вычислениях он участвует как тип `int`, ему выделяется 4 байта, два старших байта заполняются нулями.

Приведение типов. Если операнды арифметической операции имеют разные типы, то происходит повышение меньшего типа операнда и результат будет иметь высший тип операндов. Если такое действие не устраивает, можно выполнить явное *приведение типа*. Например, если `b1` и `b2` имеют тип `byte`, а желателен результат типа `short`, то можно использовать код:

```
short k = (short) (b1 + b2) ;
```

Сужение осуществляется просто отбрасыванием старших битов, что необходимо учитывать для больших значений. Например, определение

```
byte b = (byte) 300;
```

даст переменной `b` значение 44. Действительно, в двоичном представлении числа 300, равном 100101100, отбрасывается старший бит и получается 00101100.

Таким же образом можно произвести и явное расширение (widening) типа, если в этом есть необходимость.

Результат арифметической операции над целыми типами имеет тип `int`, кроме того случая, когда один из операндов типа `long`. В этом случае результат будет типа `long`. Перед выполнением арифметической операции всегда происходит повышение типов `byte`, `short`, `char`. Они преобразуются в тип `int`, а может быть, и в тип `long`, если другой операнд типа `long`. Если результат целой операции выходит за диапазон своего типа `int` или `long`, то автоматически происходит приведение по модулю, равному длине этого диапазона, и вычисления продолжают, переполнение никак не отмечается.

Укажем некоторые правила преобразования простых типов в строку и наоборот (более подробно правила преобразования представлены в документации `StringBuilder`).

Преобразование простых типов в строку `String`. Следующие примеры показывают правила преобразования:

- Преобразование из **short** или **int** `n` в **String** `gg`:

```
gg = Integer.toString(n); или gg = String.valueOf(n);
```

Соответственно можно использовать вместо `toString` следующее
`toBinaryString`, `toOctalString`, `toHexString`

Когда используется другая база чисел, например, 7, то используется команда:

```
gg = Integer.toString(b, 7);
```

- Преобразование из **char** `c`, в **String** `gg`:

```
gg = String.valueOf(c);
```

- Преобразование из **long** `n` в **String** `gg`:

```
gg = Long.toString(n); или gg = String.valueOf(n);
```

Можно заменить в случае необходимости `toString` на следующее:

```
toBinaryString, toOctalString, toHexString
```

Когда используется отличное от 10 или 2 основание системы чисел (типа 7), то:

```
gg = Integer.toString(n, 7);
```

- Преобразование из **float** `f` в **String** `gg`:

```
gg = Float.toString(f); или gg = String.valueOf(f);
```

- Преобразование из **double** `d` в **String** `gg`:

```
gg = Double.toString(d); или gg = String.valueOf(d);
```

Для сохранения десятичного формата разделения групп разрядов запятой или экспоненциального формата используются следующие приведения (для `float` – так же). Двойная точность:

```
java.text.DecimalFormat df2  
= new java.text.DecimalFormat("###,##0.00");  
gg = df2.format(d);
```

Экспоненциальный формат (JDK 1.2.x и выше):

```
java.text.DecimalFormat de
= new java.text.DecimalFormat("0.000000E00");
gg = de.format(d);
```

Преобразование строки String в простые типы. Следующие примеры показывают правила преобразования:

- Преобразование из **String** gg в **short** s:

```
try {
    s = (short) Integer.parseInt(gg.trim());
}
catch (NumberFormatException e) {
    ...
}
```

- Преобразование из **String** gg в **char** c:

```
try {
    c = (char) Integer.parseInt(gg.trim());
}
catch (NumberFormatException e) {
    ...
}
```

- Преобразование из **String** gg в **int** i

```
try {
    i = Integer.parseInt(gg.trim());
}
catch (NumberFormatException e) {
    ...
}
```

- Преобразование из **String** gg в **long** n:

```
try {
    n = Long.parseLong(gg.trim());
}
catch (NumberFormatException e) {
    ...
}
```

- Преобразование из **String** gg в **float** f:

```
try {
    f = Float.valueOf(gg.trim()).floatValue;
}
catch (NumberFormatException e) {
    ...
}
```

Лучше использовать так:

```
try {
    f = Float.parseFloat(gg.trim());
```

```

    }
    catch (NumberFormatException e) {
        ...
    }

```

- Преобразование из **String** `gg` в **double** `d`:

```

try {
    d = Double.valueOf(gg.trim()).doubleValue();
}
catch (NumberFormatException e) {
    ...
}

```

Лучше использовать так:

```

try {
    d = Double.parseDouble(gg.trim());
}
catch (NumberFormatException e) {
    ...
}

```

Замечание. Для преобразований **String** `gg` в **short**, **char** и **int**, если используется отличное от 10 основание системы чисел, например 7, то тужно использовать следующее преобразование (например, для **short**):

```

try {
    s = (short) Integer.parseInt(gg.trim(), 7);
}
catch (NumberFormatException e) {
    ...
}

```

Замечание. Для преобразований **String** `gg` в **long**, **float** и **double**, если значение `gg` является пустым указателем, то `trim()` вызывает `NullPointerException`. Если Вы не используете `trim()`, удостоверьтесь, что нет замыкающих пробелов.

Вещественные типы `float` и `double`. Они характеризуются разрядностью, диапазоном значений и точностью представления. К обычным вещественным числам добавляются еще три значения:

1. Положительная бесконечность `POSITIVE_INFINITY`, возникающая при переполнении положительного значения, например, в результате операции умножения `3.0*6e307`.
2. Отрицательная бесконечность `NEGATIVE_INFINITY`.
3. Неопределенность `NaN` (Not a Number), возникающее при делении вещественного числа на нуль или умножении нуля на бесконечность.

Кроме того, стандарт различает положительный и отрицательный нуль, возникающий при делении на бесконечность соответствующего знака, хотя сравнение `0/0 == -0/0` дает `true`.

Характеристики вещественных типов приведены в табл. 3.1.2.

Замечание. В языке Java взятие остатка от деления `%`, инкремент `++` и декремент `--` применяются и к вещественным типам.

Таблица 3.1.2. Вещественные типы

Тип	Разрядность	Диапазон	Точность
float	4	$3,4e-38 < x < 3,4e38$	7–8 цифр
double	8	$1,7e-308 < x < 1,7e308$	17 цифр

Каждому простому типу соответствует класс, который обортытывает значение примитивного типа в объект. Этот объект содержит единственное поле, тип которого является типом соответствующего примитива. Кроме того, класс обеспечивает несколько методов преобразования, имеет константы и методы, полезные для работы с соответствующим простым типом. Например, класс `Double` обортытывает значение примитивного типа `double` в объект. Объект типа `Double` содержит единственное поле, тип которой является `double`. Кроме того, этот класс обеспечивает несколько методов преобразования `double` в `String` и `String` к `double`, а также константы и другие методы, полезные для работы с `double`.

Простым типам `boolean`, `short`, `int`, `long`, `char`, `float` и `double` соответствуют классы: `Boolean`, `Character`, `Integer`, `Long`, `Character`, `Float`, `Double`. Приведем правила преобразования простых типов в объекты соответствующих классов и обратно:

- Преобразование из **long**, **float** или **double** `f` в объект **Integer** `ii`:

```
ii = new Integer((int) f);
```
- Преобразование из **short**, **char**, **int**, **long**, **float**, или **double** `n` в объект **Float** `ff` (или **Double**):

```
Ff = new Float(n);    Dd = new Double(n);
```
- Преобразование из **Integer**, **Long**, **Float**, или **Double** `nn` в простой тип **int** `i`:

```
i = nn.intValue();
```
- Преобразование из **Integer**, **Long**, **Float**, или **Double** `nn` в простой тип **float** `f` (или **double**):

```
f = nn.floatValue();    d = nn.doubleValue();
```

Операции

Рассмотрим здесь основные операции языка Java.

Операции над целыми типами. Все операции, которые производятся над целыми числами, можно разделить на следующие группы.

Арифметические операции над целыми типами. Это обычные операции сложения, вычитания, умножения, и деления. Кроме того, имеются еще три операции:

- взятие остатка от деления (деление по модулю): `%`;
- инкремент (увеличение на единицу): `++`;
- декремент (уменьшение на единицу): `--`.

Отметим, что в Java принято целочисленное деление. Это правило применяется, когда оба операнда имеют один и тот же целый тип, тогда и результат имеет тот же тип. Однако в случаях `5/2.0` или `5.0/2` или `5.0/2.0` получается `2.5` – как результат деления вещественных чисел. Операция деление по модулю определяется так: `a%b = a - (a/b) * b`; например, `5%(-3)` даст в результате `2`, так как `5 = (-3) * (-1) + 2`.

Операции инкремент и декремент означают увеличение или уменьшение значения переменной на единицу и применяются только к переменным, но не к константам или выражениям. Интересно, что эти операции можно записать и перед переменной: `++i`, `--j`. Разница проявится только в выражениях: при первой форме записи (постфиксной) в выражении участвует старое значение переменной и только потом происходит увеличение или уменьшение ее значения. При второй форме записи (префиксной) сначала изменится переменная и ее новое значение будет участвовать в выражении.

Операции сравнения целых чисел. В языке Java шесть обычных операций сравнения целых чисел по величине: больше `>`; меньше `<`; больше или равно `>=`; меньше или равно `<=`; равно `==`; не равно `!=`.

Например, проверку неравенства вида $a < x < b$ следует записать так:

```
(a < x) && (x < b);
```

Замечание. Имеются еще побитовые операции над целыми типами, когда приходится изменять значения отдельных битов в целых данных. Подробнее об этом см. документацию Java.

Операции присваивания. Простая операция присваивания записывается знаком равенства `=`, слева от которого стоит переменная, а справа выражение, совместимое с типом переменной:

```
x = 3.5, y = 2 * (x - 0.567) / (x + 2), b = x < y, bb = x >= y && b.
```

Операция присваивания действует так: выражение, стоящее после знака равенства, вычисляется и приводится к типу переменной, стоящей слева от знака равенства. Результатом операции будет приведенное значение правой части.

В операции присваивания левая и правая части неравноправны, нельзя написать $3.5 = x$. После операции `x = y` изменится переменная `x`, став равной `y`, а после `y = x` изменится `y`.

Кроме простой операции присваивания есть еще 11 составных операций присваивания:

```
+=, -=, *=, /=, % =, & =, | =, ^ =, << =, >> = ; >>> =.
```

Условная операция. Эта операция имеет три операнда. Вначале записывается произвольное логическое выражение, т. е. имеющее в результате `true` или `false`, затем знак вопроса, потом два произвольных выражения, разделенных двоеточием, например,

```
x < 0 ? 0 : x
x > y ? x-y : x+y
```

Условная операция выполняется так. Сначала вычисляется логическое выражение. Если получилось значение `true`, то вычисляется первое выражение после вопросительного знака `?` и его значение будет результатом всей операции. Последнее выражение при этом не вычисляется. Если же получилось значение `false`, то вычисляется только последнее выражение, его значение будет результатом операции.

Выражения. Из констант и переменных, операций над ними, вызовов методов и скобок составляются выражения. При вычислении выражения выполняются четыре правила:

1. Операции одного приоритета вычисляются слева направо: $x+y+z$ вычисляется как $(x+y)+z$. Исключение: операции присваивания вычисляются справа налево: $x=y=z$ вычисляется как $x=(y=z)$.
2. Левый операнд вычисляется раньше правого.
3. Операнды полностью вычисляются перед выполнением операции.
4. Перед выполнением составной операции присваивания значение левой части сохраняется для использования в правой части.

Выражения могут иметь сложный и запутанный вид. В таких случаях возникает вопрос о приоритете операций, о том, какие операции будут выполнены в первую очередь.

Приоритет операций. Операции перечислены в порядке убывания приоритета. Операции на одной строке имеют одинаковый приоритет.

1. Постфиксные операции ++ и --.
2. Префиксные операции ++ и --, дополнение ~ и отрицание !.
3. Приведение типа.
4. Умножение *, деление / и взятие остатка %.
5. Сложение + и вычитание -.
6. Сдвиги <<, >>, >>>.
7. Сравнения >, <, >=, <=.
8. Сравнения ==, !=.
9. Побитовая конъюнкция &.
10. Побитовое исключающее ИЛИ ^.
11. Побитовая дизъюнкция |.
12. Конъюнкция &&.
13. Дизъюнкция ||.
14. Условная операция ?:.
15. Присваивания =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=.

Операторы

Набор операторов языка Java включает:

- операторы описания переменных и других объектов (они были рассмотрены выше);
- операторы-выражения;
- операторы присваивания;
- условный оператор if;
- три оператора цикла while, do-while, for;
- оператор варианта switch;
- операторы перехода break, continue и return;
- блок {} – часть программы заключенная в фигурные скобки;
- пустой оператор – просто точка с запятой.

Всякий оператор завершается точкой с запятой. Можно поставить точку с запятой в конце любого выражения, и оно станет оператором. Точка с запятой в Java не разделяет операторы, а является частью оператора.

Операторы присваивания. Точка с запятой в конце любой операции присваивания превращает ее в оператор присваивания. Разница между операцией и оператором присваивания носит лишь теоретический характер. Присваивание чаще используется как оператор, а не операция.

Операторы управления последовательностью исполнения инструкций. К таким операторам относятся:

- оператор условия `if-else`;
- операторы цикла `while`, `do-while` и `for`;
- оператор `continue` и метки;
- оператор `break`;
- оператор варианта `switch`;
- оператор `return` прекращения исполнения.

Дадим их краткое описание.

Оператор условия `if`. Он допускает и расширенную форму: `if – else if – else`. Действие оператора демонстрируется примером,

```
if (n == 0){  
    sign = 0;  
} else if (n < 0){  
    sign = -1;  
} else {  
    sign = 1;  
}
```

Оператор условия может быть сокращенным (`if-then statement`):

```
if (логическое выражение) оператор ;
```

Тогда в случае `false` не выполняется ничего.

Операторы цикла `while`, `do-while` и `for`. Оператор `while` применяется в виде:

```
while (логическое выражение) оператор ;
```

Сначала вычисляется логическое выражение, если его значение `true`, то выполняется оператор, образующий цикл. Затем снова вычисляется логическое и действует оператор, и так до тех пор, пока не получится значение `false`. Если логическое выражение изначально равняется `false`, то оператор не будет выполнен ни разу. Если в цикл надо включить несколько операторов, то следует образовать блок.

Второй оператор цикла – оператор `do-while` – имеет вид

```
do оператор while (логическое выражение) ;
```

Здесь сначала выполняется оператор, а потом происходит вычисление логического выражения. Цикл выполняется, пока логическое выражение остается равным `true`. В цикле `do-while` проверяется условие продолжения, а не окончания цикла. Различие между этими двумя операторами цикла только в том, что в цикле `do-while` оператор обязательно выполнится хотя бы один раз.

Третий оператор цикла – оператор `for` – выглядит так:

```
for (список_выр1 ; ЛогВыр; список_выр2) оператор ;
```


Перед выполнением цикла вычисляется список выражений `список_выр1`. Это нуль или несколько выражений, перечисленных через запятую. Они вычисляются слева направо, и в следующем выражении уже можно использовать результат предыдущего выражения. Как правило, здесь задаются начальные значения переменным цикла. Затем вычисляется логическое выражение. Если оно истинно, `true`, то действует оператор, потом вычисляются слева направо выражения из списка выражений `список_выр2`. Далее снова проверяется логическое выражение. Если оно истинно, то выполняется оператор и `список_выр2` и т. д. Как только логическое выражение станет равным `false`, выполнение цикла заканчивается. Действие оператора `for` хорошо видно на примере вычисления суммы квадратов первых `N` натуральных чисел:

```
int s=0;
for (int k = 1; k <= N; k++) s += k*k;
```

Оператор `continue` и метки. Оператор `continue` используется только в операторах цикла. Он имеет две формы. Первая форма состоит только из слова `continue` и осуществляет немедленный переход к следующей итерации цикла. В очередном фрагменте кода оператор `continue` позволяет обойти деление на нуль:

```
for (int i = 0; i < N; i++){
    if (i == j) continue;
    s += 1.0 / (i - j);
}
```

Вторая форма содержит метку:

```
continue метка ;
```

Метка записывается, как все идентификаторы и не требует никакого описания. Метка ставится перед оператором или открывающей фигурной скобкой и отделяется от них двоеточием. Так получается помеченный оператор или помеченный блок. Вторая форма используется только в случае нескольких вложенных циклов для немедленного перехода к очередной итерации одного из объемлющих циклов, а именно, помеченного цикла.

Оператор `break`. Он используется в операторах цикла и операторе варианта для немедленного выхода из этих конструкций в следующей форме:

```
if (что-то случилось) break M2;
```

Здесь `M2` – это метка блока, куда нужно передать исполнение.

Оператор варианта `switch`. Оператор варианта `switch` организует разветвление по нескольким направлениям. Каждая ветвь отмечается константой или константным выражением какого-либо целого типа (кроме `long`) и выбирается, если значение определенного выражения совпадет с этой константой. Вся конструкция выглядит так.

```
switch (Выражение){
case значение1: оператор1 ;
case значение2: оператор2 ;
. . . . .
case значениеN: операторN ;
```

```
default: операторDef ;  
}
```

Стоящее в скобках выражение и значения оператора **case** должны быть типа `byte`, `short`, `int`, `char`. Все значения выражения вычисляются заранее, на этапе компиляции, и должны быть различными.

Оператор варианта выполняется так. Сначала вычисляется целочисленное выражение в скобках. Если оно совпадает с одним из значений `case`, то выполняется оператор, отмеченный этим значением. Затем выполняются все следующие операторы и работа заканчивается.

Если же ни одна константа не равна значению выражения, то выполняется `операторDef` и все следующие за ним операторы. Поэтому ветвь `default` должна записываться последней. Ветвь `default` может отсутствовать, тогда в этой ситуации оператор варианта вообще ничего не делает.

Чаще всего необходимо «пройти» только одну ветвь операторов. В таком случае используется оператор **break**, сразу же прекращающий выполнение оператора **switch**. Может понадобиться выполнить один и тот же оператор в разных ветвях `case`. В этом случае ставим несколько меток `case` подряд. Например,

```
switch(dayOfWeek) {  
case 1: case 2: case 3: case 4: case 5:  
System.out.println("Week-day");, break;  
case 6: case 7:  
System.out.println("Week-end"); break;  
default:  
System.out.println("Unknown day");  
}
```

Оператор return. Он используется для прекращения исполнения текущей подпрограммы и передачи управления вызывающей программе. Может быть поставлен в любом месте в виде

```
if (true) return;
```

Массивы

Массив – это совокупность переменных одного типа, хранящихся в смежных ячейках оперативной памяти. *Массивы* в языке Java относятся к ссылочным типам, их описание производится в несколько этапов.

Сначала делается объявление массива. Указывается тип массива, квадратными скобками указывается, что объявляется ссылка на массив и перечисляются имена переменных, например,

```
double[] a, b;
```

Здесь определены две переменные – ссылки `a` и `b` на одномерные массивы типа `double`. Можно поставить квадратные скобки и после имени переменной:

```
int i = 0, arr[], k = -1;
```

Здесь определены две переменные целого типа `i` и `k`, и объявлена ссылка на целочисленный массив `arr`. В скобках можно указать размер массива. Пустые

скобки говорят компилятору, что размер массива не ограничен и память для него будет выделяться в процессе выполнения программы.

Затем указывается количество элементов массива, для того, чтобы выделить память под массив, переменная-ссылка получает адрес массива. Эти действия производятся операцией **new**. Например,

```
a = new double[5];
b = new double[100];
arr = new int[50];
```

Отметим, что индексы массивов всегда начинаются с нуля. Индексы можно задавать любыми целочисленными выражениями, кроме типа `long`, например, `a[i+j]`, `a[i%5]`, `a[++i]`. Исполняющая система Java следит за тем, чтобы значения этих выражений не выходили за границы длины массива.

На последнем этапе производится инициализация массива, элементы массива получают начальные значения. Например,

```
a[0] = 0.01; a[1] = -3.4; a[2] = 2.89; a[3] = 4.5; a[4] = -6.7;
for (int i = 0; i < 100; i++) b[i] = 1.0 / i;
for (int i = 0; i < 50; i++) arr[i] = 2 * i + 1;
```

Первые два этапа можно совместить:

```
Double[] a = new double[5], b = new double[100];
int i = 0, arr[] = new int[50], k = -1;
```

Можно сразу задать и начальные значения, записав их в фигурных скобках через запятую в виде констант или константных выражений. При этом необязательно указывать количество элементов массива, оно будет равно количеству начальных значений;

```
double[] a = {0.01, -3.4, 2.89, 4.5, -6.7};
```

Можно совместить второй и третий этап:

```
a = new double[] {0.1, 0.2, -0.3, 0.45, -0.02};
```

Ссылка на массив не является частью описанного массива, ее можно перебросить на другой массив того же типа операцией присваивания. Например, после присваивания `a = b` обе ссылки `a` и `b` указывают на один и тот же массив из 100 вещественных переменных типа `double` и содержат один и тот же адрес.

Ссылка может присвоить «пустое» значение `null`, не указывающее ни на какой адрес оперативной памяти:

```
arr = null;
```

После этого массив, на который указывала данная ссылка, теряется, если на него не было других ссылок.

Кроме простой операции присваивания, со ссылками можно производить еще только сравнения на равенство, например, `a == b`, и неравенство, `a != b`. При этом сопоставляются адреса, содержащиеся в ссылках, мы можем узнать, не ссылаются ли они на один и тот же массив.

Кроме ссылки на массив, для каждого массива автоматически определяется длина массива как целая константа с именем `length`. Для каждого массива имя этой константы уточняется именем массива через точку. Например, константа `a.length` равна 5. Последний элемент массива `a` можно записать так: `a[a.length - 1]`.

Массив символов в Java не является строкой, даже если он заканчивается нуль-символом `'\u0000'`.

Многомерные массивы. В Java они реализованы как массивы массивов. Элементы массива первого уровня снова являются массивами, причем не требуется, чтобы длины массивов второго уровня были бы одинаковы. Двумерный массив в Java не обязан быть прямоугольным. Многомерный массив можно объявить таким образом:

```
char[][] c;
```

или

```
char c[][];
```

Затем определяем внешний массив (первого уровня):

```
c = new char[3][];
```

Тогда `c` – массив, состоящий из трех элементов-массивов. Теперь определяем его элементы-массивы длины, соответственно, 2, 4 и 3:

```
c[0] = new char[2];
```

```
c[1] = new char[4];
```

```
c[2] = new char[3];
```

Теперь можно задать начальные значения `c[0][0] = 'a', c[0][1] = 'r', c[1][0] = 'r', c[1][1] = 'a', c[1][2] = 'y'` и т.д.

Описания можно сократить:

```
int[][] d = new int[3][4];
```

а начальные значения задать так:

```
int[][] inds = {{1, 2, 3}, {4, 5, 6}};
```

3.1.2. Классы в Java

В этом параграфе приведем краткий обзор объектно-ориентированного программирования на Java.

Понятие класса

Основная идея объектно-ориентированного программирования (ООП) заключается в том, чтобы разбить программу на модули так, чтобы она превратилась в совокупность взаимодействующих объектов. Каждый объект представлен в виде модуля. Автономность модулей позволяет создавать и библиотеки модулей, чтобы потом использовать их в качестве строительных блоков для программы. Для того чтобы обеспечить максимальную независимость модулей друг от друга, надо четко отделить процедуры, которые будут вызываться другими

модулями, это – открытые (public) процедуры, от вспомогательных – закрытых (private) процедур. Данные, занесенные в модуль, тоже делятся на открытые, указанные в интерфейсе и доступные для других модулей и закрытые, доступные только для процедур того же модуля.

Класс можно считать проектом, шаблоном, по которому затем будут создаваться конкретные объекты.

Члены класса. Класс содержит описание переменных и констант, характеризующих объект. Они называются полями класса. Процедуры, описывающие поведение объекта, называются методами класса. Внутри класса можно описать и вложенные классы (nested classes) и вложенные интерфейсы. Поля, методы и вложенные классы первого уровня являются членами класса (class members). Отметим, что в Java нет вложенных процедур и функций, в теле метода нельзя описать другой метод.

Инкапсуляция (incapsulation). Это сокрытие данных и методов их обработки. Инкапсуляция преследует две основные цели. Первая – обеспечить безопасность использования класса, вынести в интерфейс, сделать общедоступными только те методы обработки информации, которые не могут испортить или удалить исходные данные. Вторая цель – упростить, скрыв ненужные детали реализации. Члены класса, к которым не планируется обращение извне, должны быть инкапсулированы. В языке Java инкапсуляция достигается добавлением модификатора **private** к описанию члена класса.

Объекты. После того как описание класса закончено, можно создавать конкретные объекты, *экземпляры* (instances) описанного класса. Объект – это реализация класса, либо массив. Создание экземпляров производится в три этапа, подобно описанию массивов. Сначала объявляются ссылки на объекты: записывается имя класса, и через пробел перечисляются экземпляры класса, точнее, ссылки на них. Например, если создан класс MyClass, то экземпляры A1, A1 и A3 этого класса объявляются так:

```
MyClass A1, A2;
```

Затем операцией **new** определяются сами объекты, под них выделяется оперативная память, ссылка получает адрес этого участка памяти в качестве своего значения.

```
A1 = new MyClass();
```

```
A2 = new MyClass();
```

На третьем этапе происходит инициализация объектов, задаются их начальные значения. Этот этап, как правило, совмещается со вторым, именно для этого в операции **new** в имени класса MyClass() в скобках можно задать начальные значения параметров.

Таким образом, каждый объект имеет определенные характеристики и набор определенных действий (процедур). Например, окно на экране дисплея – это объект, имеющий ширину width и высоту height, расположение на экране, описываемое обычно координатами (x, y) левого верхнего угла окна, а также шрифт, которым в окно выводится текст, цвет фона color и другие характеристики. Дей-

ствия: окно может перемещаться по экрану методом `move()`, увеличиваться или уменьшаться в размерах методом `size()`, сворачиваться в ярлык методом `iconify()`, реагировать на действия мыши и нажатия клавиш. Кнопки, полосы прокрутки и прочие элементы окна – это тоже объекты со своими размерами, шрифтами, перемещениями.

Иерархия классов. Она заключается в том, что для данного, достаточно общего класса, можно образовать подклассы, которые включают свойства и методы исходного класса, но имеют свои особенности. Такая организация классов напоминает классификацию в биологии. Отметим, что на каждом следующем уровне иерархии в класс добавляются новые свойства, но ни одно общее свойство не пропадает. Поэтому новый, более частный класс называется продолжением (*extension*) класса, или подклассом. Также говорят о наследовании (*inheritance*) классов. Более широкий класс при этом называется суперклассом (*superclass*). Часто используют и такую терминологию: надкласс, родительский класс, дочерний класс, класс-потомок, класс-предок.

Права доступа к членам класса. Как уже отмечалось, члены класса, к которым не планируется обращение извне, должны быть инкапсулированы добавлением модификатора `private` к описанию члена класса. Тогда эти члены классов становятся закрытыми, ими могут пользоваться только экземпляры того же самого класса. В противоположность закрытости можно объявить некоторые члены класса открытыми, записав вместо слова `private` модификатор `public`. К таким членам может обратиться любой объект любого класса. Когда надо разрешить доступ только наследникам класса, тогда в Java используется защищенный (*protected*) доступ, отмечаемый модификатором `protected`.

В языке Java словами `private`, `public` и `protected` отмечается каждый член класса в отдельности.

Принцип модульности предписывает открывать члены класса только в случае необходимости. Если же надо обратиться к полю класса, то рекомендуется включить в класс специальные методы доступа, отдельно для чтения этого поля (метод **get**) и для записи в это поле (метод **set**). Имена методов доступа рекомендуется начинать со слов `get` и `set`, добавляя к этим словам имя поля.

Как описать класс и подкласс

Для создания нового класса необходимо создать файл, в котором будет описание класса. Имя файла должно совпадать с именем содержащегося в нем класса и иметь расширение `java`.

Описание класса начинается со слова `class`, после которого записывается имя класса. Рекомендуется начинать имя класса с заглавной буквы. Перед словом `class` можно записать модификаторы класса (*class modifiers*). Это одно из слов `public`, `abstract`, `final`, `strictfp`.

Перед именем вложенного класса можно поставить, кроме того, модификаторы `protected`, `private`, `static`. Тело класса, в котором в любом порядке перечисляются поля, методы, вложенные классы и интерфейсы, заключается в фигурные скобки.

При описании поля указывается его тип, затем, через пробел, имя и, может быть, начальное значение после знака равенства, которое можно записать константным выражением.

Описание поля может начинаться с одного или нескольких необязательных модификаторов `public`, `protected`, `private`, `static`, `final`. Если надо поставить несколько модификаторов, то перечислять их JLS рекомендует в указанном порядке, поскольку некоторые компиляторы требуют определенного порядка записи модификаторов.

При описании метода указывается тип возвращаемого им значения или слово `void`, затем, через пробел, имя метода, потом, в скобках, список параметров. После этого в фигурных скобках пишется процедура метода.

Описание метода может начинаться с модификаторов `public`, `protected`, `private`, `abstract`, `static`, `final`, `synchronized`, `native`, `strictfp`.

В списке параметров через запятую перечисляются тип и имя каждого параметра. Перед типом какого-либо параметра может стоять модификатор `final`. Такой параметр нельзя изменять внутри метода. Список параметров может отсутствовать, но скобки сохраняются.

Перед началом работы метода для каждого параметра выделяется ячейка оперативной памяти, в которую копируется значение параметра, заданное при обращении к методу. Такой способ называется передачей параметров по значению.

Пример. Нахождение корня нелинейного уравнения делением отрезка пополам.

```
class Bisection2
{
    private static double final EPS = 1e-8; // Константа
    private double a = 0.0, b = 1.5, root; // Закрытые поля
    public double getRoot(){return root;} // Метод доступа
    private double f(double x)
    {
        return x*x*x - 3*x*x + 3; // Или что-то другое
    }
    private void bisect(){ // Параметров нет —
        // метод работает с полями экземпляра
        double y = 0.0; // Локальная переменная — не поле
        do{
            root = 0.5 * (a + b); y = f(root);
            if (Math.abs(y) < EPS) break;
            // Корень найден. Выходим из цикла
            // Если на концах отрезка [a; root]
            // функция имеет разные знаки:
            if (f(a) * y < 0.0) b = root;
            // значит, корень здесь
            // Переносим точку b в точку root
            // В противном случае:
            else a = root;
            // переносим точку a в точку root
            // Продолжаем, пока [a; b] не станет мал
        } while(Math.abs(b-a) >= EPS);
    }
}
```

```
}
public static void main(String[] args){
    Bisection2 b2 = new Bisection2();
    b2.bisect();
    System.out.println("x = " +
        b2.getRoot() +      // Обращаемся к корню через метод доступа
        ", f() = " +b2.f(b2.getRoot()));
    }
}
```

В описании метода `f()` сохранен старый, процедурный стиль: метод получает аргумент, обрабатывает его и возвращает результат. Описание метода `bisect` выполнено в духе ООП: метод активен, он сам обращается к полям экземпляра `b2` и сам заносит результат в нужное поле. Метод `bisect()` – это внутренний механизм класса `Bisection2`, поэтому он закрыт (`private`).

Имя метода, число и типы параметров образуют *сигнатуру* (signature) метода. Компилятор различает методы не по их именам, а по сигнатурам. Это позволяет записывать разные методы с одинаковыми именами, различающиеся числом и/или типами параметров. Такое дублирование методов называется перегрузкой методов. Тип возвращаемого значения не входит в сигнатуру метода, значит, методы не могут различаться только типом результата их работы.

Окончательные члены и классы

Пометив метод модификатором **final**, можно запретить его переопределение в подклассах. Это удобно в целях безопасности для уверенности, что метод выполняет те действия, которые вы задали. Именно так определены математические функции `sin()`, `cos()` и прочие в классе `Math`. Для полной безопасности, поля, обрабатываемые окончательными методами, следует сделать закрытыми (`private`).

Если же пометить модификатором **final** весь класс, то его вообще нельзя будет расширить. Так определен, например, класс `Math`:

```
public final class Math{ . . . }
```

Для переменных модификатор **final** имеет совершенно другой смысл. Если пометить модификатором **final** описание переменной, то ее значение (а оно должно быть обязательно задано или здесь же, или в блоке инициализации или в конструкторе) нельзя изменить ни в подклассах, ни в самом классе. Переменная превращается в константу. Именно так в языке Java определяются константы:

```
public final int MIN_VALUE = -1, MAX_VALUE = 9999;
```

Напомним, что константы принято записывать прописными буквами, а слова в них разделяются знаком подчеркивания.

Класс Object

На самой вершине иерархии классов Java стоит класс **Object**. Если при описании класса мы не указываем никакого расширения, т. е. не пишем слово `extends` и имя класса за ним, то Java считает этот класс расширением класса `Object`, и компилятор дописывает это за нас:

```
class MyClass extends Object{ . . . }
```


Можно записать это расширение и явно. Сам же класс `Object` не является ничьим наследником, от него начинается иерархия любых классов Java. В частности, все массивы – прямые наследники класса `Object`. Поскольку такой класс может содержать только общие свойства всех классов, в него включено лишь несколько самых общих методов. Например, метод `equals()`, сравнивающий данный объект на равенство с объектом, заданным в аргументе, и возвращающий логическое значение; метод `toString()`, который преобразует содержимое объекта в строку символов и возвращает объект класса `string`. Класс `Object` входит в базовый пакет `java.lang` языка Java.

Оператор new

Он применяется для выделения памяти массивам и объектам. В первом случае в качестве операнда указывается тип элементов массива и количество его элементов в квадратных скобках, например:

```
double a[] = new double[100];
```

Во втором случае операндом служит конструктор класса. Если конструктора в классе нет, то вызывается конструктор по умолчанию. Числовые поля класса получают нулевые значения, логические поля – значение `false`, ссылки – значение `null`. Результатом операции `new` будет ссылка на созданный объект. Эта ссылка может быть присвоена переменной:

```
Dog k9 = new Dog() ;
```

но может использоваться и непосредственно

```
new Dog().voice();
```

Здесь после создания безымянного объекта сразу выполняется его метод `voice()`. Такая странная запись встречается в программах, написанных на Java, на каждом шагу.

Конструкторы класса

Конструктор – это метод класса, который инициализирует новый объект сразу после его создания. Имя конструктора совпадает с именем соответствующего класса. Конструктор выполняется автоматически при создании экземпляра класса. Конструктор не возвращает никакого значения. Поэтому в его описании не пишется даже слово `void`, но можно задать один из трех модификаторов `public`, `protected` или `private`.

Тело конструктора может начинаться:

- с вызова одного из конструкторов суперкласса, для этого записывается слово `super()` с параметрами в скобках, если они нужны;
- с вызова другого конструктора того же класса, для этого записывается слово `this()` с параметрами в скобках, если они нужны.

В классе может быть несколько конструкторов. Поскольку у них одно и то же имя, совпадающее с именем класса, то они должны отличаться типом и/или коли-

чеством параметров. Если конструктора в классе нет, то вызывается конструктор по умолчанию. Числовые поля класса получают нулевые значения, логические поля – значение `false`, ссылки – значение `null`. Если в классе имеется конструктор класса, например, `MyClass(int a, int b, boolean c)`, то для создания экземпляра `C1` класса он вызывается, например, в виде

```
MyClass C1 = new MyClass(11, 15, false);
```

Статические члены класса

Разные экземпляры одного класса имеют совершенно независимые друг от друга поля, принимающие разные значения. Изменение поля в одном экземпляре никак не влияет на то же поле в другом экземпляре. В каждом экземпляре для таких полей выделяется своя ячейка памяти. Поэтому такие поля называются *переменными экземпляра* класса (*instance variables*) или переменными объекта.

Иногда надо определить поле, общее для всего класса, изменение которого в одном экземпляре повлечет изменение того же поля во всех экземплярах. Такие поля называются *переменными класса*. Для переменных класса выделяется только одна ячейка памяти, общая для всех экземпляров. Переменные класса образуются в Java модификатором **static**. К статическим переменным можно обращаться с именем класса, а не только с именем экземпляра, причем это можно делать, даже если не создан ни один экземпляр класса.

Для работы с такими статическими переменными обычно создаются *статические методы*, помеченные модификатором `static`. Для методов слово `static` имеет совсем другой смысл. Исполняющая система Java всегда создает в памяти только одну копию машинного кода метода, разделяемую всеми экземплярами, независимо от того, статический это метод или нет.

Основная особенность статических методов – они выполняются сразу во всех экземплярах класса. Более того, они могут выполняться, *даже если не создан ни один экземпляр класса*. Достаточно уточнить имя метода именем класса (а не именем объекта), чтобы метод мог работать. Именно так можно использовать методы класса `Math`, не создавая его экземпляры, а просто записывая `Math.abs(x)`, `Math.sqrt(x)`.

Поэтому статические методы называются методами класса (*class methods*), в отличие от нестатических методов, называемых методами экземпляра (*instance methods*). Отсюда вытекают другие особенности статических методов:

- в статическом методе нельзя использовать ссылки `this` и `super`;
- в статическом методе нельзя прямо, не создавая экземпляров, ссылаться на нестатические поля и методы;
- статические методы не могут быть абстрактными;
- статические методы переопределяются в подклассах только как статические.

Статические переменные инициализируются еще до начала работы конструктора, но при инициализации можно использовать только константные выражения. Если же инициализация требует сложных вычислений, например, циклов для задания значений элементам статических массивов или обращений к мето-

дам, то эти вычисления заключают в блок, помеченный словом `static`, который тоже будет выполнен до запуска конструктора:

```
static int[] a = new a[10];
static {
    for(int k = 0; k < a.length; k++)
        a[k] = k * k;
}
```

Операторы, заключенные в такой блок, выполняются только один раз, при первой загрузке класса, а не при создании каждого экземпляра.

Метод *main()*

Всякая программа, оформленная как приложение, должна содержать метод с именем *main*. Он может быть один на все приложение или содержаться в некоторых классах этого приложения, а может находиться и в каждом классе.

Метод `main()` записывается как обычный метод, может содержать любые описания и действия, но он обязательно должен быть открытым (`public`), статическим (`static`), не иметь возвращаемого значения (`void`). Например,

```
class HelloWorld{
    public static void main(String[] args)
    {System.out.println("Hello, World!"); }
}
```

Метод `main()` вызывается автоматически исполняющей системой Java в самом начале выполнения приложения. При вызове интерпретатора `java` указывается класс, где записан метод `main()`, с которого надо начать выполнение. Аргументом метода `main()` является массив строк (`String[]`). По традиции этот массив называют `args`, хотя имя может быть любым. При вызове интерпретатора `java` можно передать в метод `main()` несколько параметров, которые интерпретатор заносит в массив строк. Эти параметры перечисляются в командной строке вызова `java` через пробел сразу после имени файла (класса).

Где видны переменные

В языке Java нестатические переменные можно объявлять в любом месте кода между операторами. Статические переменные могут быть только полями класса, а значит, не могут объявляться внутри методов и блоков. Переменным класса и экземпляра неявно присваиваются нулевые значения. Символы неявно получают значение `"\u0000"`, логические переменные – значение `false`, ссылки получают неявно значение `null`.

Локальные же переменные неявно не инициализируются. Им должны либо явно присваиваться значения, либо они обязаны определяться до первого использования. Компилятор замечает неопределенные локальные переменные и сообщает о них.

Блок инициализации экземпляра (*instance initialization*) – это просто блок операторов в фигурных скобках, но записывается он вне всякого метода, прямо в теле

класса. Этот блок выполняется при создании каждого экземпляра, после инициализации при объявлении переменных, но до выполнения конструктора. Он играет такую же роль, как и `static`-блок для статических переменных.

Вложенные классы

В этом параграфе уже несколько раз упоминалось, что в теле класса можно сделать описание другого, *вложенного* (nested) класса. А во вложенном классе можно снова описать вложенный, внутренний (inner) класс и т. д. Из вложенного класса можно обратиться к членам внешнего класса. Для того, чтобы определить экземпляр вложенного класса, необходимо определить экземпляр внешнего класса. Может оказаться, что экземпляров внешнего класса несколько, тогда имя экземпляра вложенного класса уточняется именем связанного с ним экземпляра внешнего класса. Более того, при создании вложенного экземпляра операция `new` тоже уточняется именем внешнего экземпляра.

Все вложенные классы можно разделить на вложенные классы-члены класса (member classes), описанные вне методов, и вложенные локальные классы (local classes), описанные внутри методов и/или блоков. Локальные классы, как и все локальные переменные, не являются членами класса.

Классы-члены могут быть объявлены статическим модификатором `static`. Поведение статических классов-членов ничем не отличается от поведения обычных классов, отличается только обращение к таким классам. Поэтому они называются вложенными классами верхнего уровня (nested top-level classes), хотя статические классы-члены можно вкладывать друг в друга. В них можно объявлять статические члены. Используются они обычно для того, чтобы сгруппировать вспомогательные классы вместе с основным классом.

Все нестатические вложенные классы называются внутренними (inner). В них нельзя объявлять статические члены.

Локальные классы, как и все локальные переменные, известны только в блоке, в котором они определены. Они могут быть безымянными.

Пакеты и интерфейсы

Все классы Java распределяются по пакетам (packages). Кроме классов пакеты могут включать в себя интерфейсы и вложенные подпакеты (subpackages). Распределение по пакетам аналогично распределению файлов по каталогам и подкаталогам. Имена классов, интерфейсов в разных пакетах могут совпадать. Если надо использовать два класса с одинаковыми именами из разных пакетов, то имя класса уточняется именем пакета: `пакет.класс`. Такое уточненное имя называется полным именем класса. Пакетами пользуются еще и для того, чтобы добавить к уже имеющимся правам доступа к членам класса `private`, `protected` и `public` еще один, «пакетный» уровень доступа. Если член класса не отмечен ни одним из модификаторов `private`, `protected`, `public`, то, по умолчанию, к нему осуществляется пакетный доступ (default access), а именно, к такому члену может обратиться любой метод любого класса из того же пакета, но если класс не помечен модификатором `public`, то все его члены, даже открытые, `public`, не будут видны из других пакетов.

Создание пакета и подпакета. Для создания *пакета* используется оператор **package**, надо просто в первой строке Java-файла с исходным кодом записать следующую строку с именем пакета:

```
package mypack;
```

Тем самым создается пакет с указанным именем *mypack* и все классы, записанные в этом файле, попадут в пакет *mypack*. Повторяя эту строку в начале каждого исходного файла, включаем в пакет новые классы.

Имя подпакета уточняется именем пакета следующим образом:

```
package mypack.subpack;
```

тогда все классы этого файла и всех файлов с такой же первой строкой попадут в подпакет *subpack* пакета *mypack*. Можно создать и подпакет подпакета,

```
package mypack.subpack.sub;
```

Отметим, что строка объявления пакета только одна и это обязательно первая строка файла, поэтому каждый класс попадает только в один пакет или подпакет. Компилятор Java может сам создать каталог с тем же именем *mypack*, а в нем подкаталог *subpack*, и разместить в них class-файлы с байт-кодами. Полные имена классов *A*, *B* будут выглядеть так: *mypack.A*, *mypack.subpack.B*.

Если пакет не создается, то файлы с откомпилированными классами попадают в безымянный пакет, которому соответствует текущий каталог файловой системы. Тогда class-файл оказывался в том же каталоге, что и соответствующий Java-файл. Большие проекты лучше хранить в пакетах.

Импорт классов и пакетов. Компилятор будет искать классы только в одном пакете, именно, в том, что указан в первой строке файла. Для классов из другого пакета надо каждый раз указывать полные имена. Если полные имена длинные, а используются классы часто, то экономнее использовать оператора **import** для того, чтобы подключить необходимые пакеты. Правила использования оператора **import** очень просты: пишется слово **import** и, через пробел, полное имя класса, завершенное точкой с запятой. Сколько классов надо указать, столько операторов **import** и пишется. В дальнейшем они уже используются без полного пути. Во второй форме оператора **import** указывается имя пакета или подпакета, а вместо короткого имени класса ставится звездочка *****. Этой записью компилятору предписывается подключить все классы пакета,

```
import mypack.*;
```

Напомним, что импортировать можно только открытые классы, помеченные модификатором **public**. Оператор **import** аналогичен директиве препроцессора **include** в C++ и аналогичен оператору **with** подключения пакетов в Maple.

Замечание. Пакет *java.lang* стандартной библиотеки Java импортировать не обязательно. Он подключен по умолчанию. Этот пакет содержит фундаментальные классы и интерфейсы языка Java. В частности, он содержит наиболее важные классы: *Object*, *Class*, *Void*, *Compiler*, *Double* и др.

Интерфейсы. В Java запрещено множественное наследование. При расширении класса после слова **extends** можно написать только одно имя суперкласса. С помощью уточнения **super** можно обратиться только к членам непосредственно

го суперкласса. Интерфейс (interface) Java решает проблему множественного наследования.

Интерфейс, в отличие от класса, содержит только константы и заголовки методов, без их реализации. Интерфейсы размещаются в тех же пакетах и подпакетах, что и классы, и компилируются тоже в class-файлы.

Определение интерфейса сходно с определением класса. Главное отличие состоит в том, что в интерфейсе у методов отсутствуют операторы тела {}. Описание интерфейса начинается со слова interface, перед которым может стоять модификатор public, означающий, что интерфейс доступен всюду. Если же модификатора public нет, интерфейс будет виден только в своем пакете.

После оператора interface записывается имя интерфейса, потом может стоять слово extends и список интерфейсов-предков через запятую. Таким образом, интерфейсы могут порождаться от интерфейсов, образуя свою, независимую от классов, иерархию, причем в ней допускается множественное наследование интерфейсов. В этой иерархии нет корня, общего предка.

Затем, в фигурных скобках, записываются в любом порядке константы и заголовки методов. Можно сказать, что в интерфейсе все методы абстрактные, но слово abstract писать не надо. Константы всегда статические, но слова static и final указывать не нужно. Все константы и методы в интерфейсах всегда открыты, не надо даже указывать модификатор public. Общий синтаксис объявления интерфейса следующий:

```
interface Имя
{
    тип_результата имя_метода1(список параметров);
    тип имя_переменной1 = значение;
}
```

Таким образом, интерфейс – это только схема. В нем указано, что делать, но не указано, как это делать. Для использования интерфейса создается класс, который реализует интерфейс. *Реализация (implementation) интерфейса* – это класс, в котором содержатся методы и константы, объявленные в одном или нескольких интерфейсах.

Для задания реализации интерфейсов необходимо в заголовке класса после его имени записывать оператор **implements** и, через запятую, перечислить имена интерфейсов. Объявленные в интерфейсе переменные неявно считаются как final-переменные. Это означает, что класс реализации не может изменить их значения. Поэтому интерфейсы можно использовать для импорта в различные классы совместно используемых констант.

Пример. Система управления светодором

```
interface Lights{
    int RED = 0;
    int YELLOW = 1;
    int GREEN = 2;
    int ERROR = -1;
}

class Timer implements Lights{
    private int delay;
```

```

private static int light = RED;
Timer(int sec) (delay = 1000 * sec; }
public int shift() {
    int count = (light++) % 3;
    try {
        switch (count) {
            case RED: Thread.sleep(delay); break;
            case YELLOW: Thread.sleep(delay/3); break;
            case GREEN: Thread.sleep(delay/2); break;
        }
    } catch (Exception e) { return ERROR; }
    return count;
}
}

class TrafficRegulator {
private static Timer t = new Timer(1);
public static void main(String[] args) {
    for (int k = -0; k < 10; k++)
        switch (t.shift()) {
            case Lights.RED: System.out.println(«Stop!»); break;
            case Lights.YELLOW: System.out.println(«Wait!»); break;
            case Lights.GREEN: System.out.println(«Go!»); break;
            case Lights.ERROR: System.err.println(«Time Error»); break;
            default: System.err.println(«Unknown light.»); return;
        }
    }
}
}

```

Здесь, в интерфейсе `Lights`, определены константы, общие для всего проекта. Класс `Timer` реализует этот интерфейс и использует константы напрямую как свои собственные. Метод `shift` этого класса подает сигналы переключения светофору с разной задержкой в зависимости от цвета. Задержку осуществляет метод `sleep()` класса `Thread` из стандартной библиотеки, которому передается время задержки в миллисекундах. Этот метод нуждается в обработке исключений `try{} catch() {}`.

Класс `TrafficRegulator` не реализует интерфейс `Lights` и пользуется полными именами `Lights.RED` и т.д. Это возможно потому, что константы `RED`, `YELLOW` и `GREEN` по умолчанию являются статическими.

Структура Java-файла

Теперь можно описать структуру исходного файла с текстом программы на языке Java:

- в первой строке файла может быть необязательный оператор `package`;
- в следующих строках могут быть необязательные операторы `import`;
- далее идут описания классов и интерфейсов;
- среди классов файла может быть только один открытый `public`-класс;
- имя файла должно совпадать с именем открытого класса, если последний существует. Отсюда следует, что, если в проекте есть несколько открытых

классов, то они должны находиться в разных файлах. Рекомендуется открытый класс, который, если он имеется в файле, нужно описывать первым.

3.2. Введение в Java Builder

В данном параграфе будет рассмотрено краткое описание работы пакета MATLAB Builder для Java по созданию компонента, который инкапсулирует коды MATLAB® и будет представлен пример простого приложения Java, который вызывает методы, созданные из m-функций MATLAB.

Компонент, созданный MATLAB® Builder для Java – это автономный пакет Java (.jar файл, package). Пакет содержит один или более классов Java, которые инкапсулируют m-коды. Эти классы имеют методы, которые вызываются непосредственно из кода Java. При работе с Java Builder создается проект, который включает необходимые m-коды. Java Builder преобразовывает эти m-функции MATLAB® в методы класса Java.

3.2.1. Общие сведения о MATLAB Builder для Java

Пакет расширения MATLAB Builder для Java (также называемый Java Builder) есть расширение пакета MATLAB Compiler. Java Builder используется для преобразования функций MATLAB в один или более классов Java, которые составляют компонент Java, или пакет. Каждая функция MATLAB реализуется как метод класса Java и может быть вызвана из приложения Java. Приложения, созданные при помощи Java Builder, при своей работе не требуют установленной системы MATLAB. Однако должны быть включены файлы поддержки, созданные Java Builder, а также среда MCR выполнения компонент MATLAB.

Чтобы дать возможность приложениям Java обмениваться данными с методами MATLAB, которые они вызывают, Java Builder обеспечивает пакет `com.mathworks.toolbox.javabuilder.MWArray`. Этот пакет имеет набор классов преобразования данных, полученных из абстрактного класса `MWArray`. Каждый класс представляет тип данных MATLAB. Более подробно это обсуждается в разделе «Использование классов `MWArray`», см. также `com.mathworks.toolbox.javabuilder`.

Установка MATLAB Builder для Java. Пакет MATLAB Builder для Java устанавливается обычным путем: при установке MATLAB нужно выбрать этот компонент вместе с MATLAB Compiler. Для работы MATLAB Builder для Java необходимо следующее:

- среда разработки Java (Java Development Kit, JDK) версии 1.4 или более поздней;
- оперативные средства управления работой программы Java (Java Runtime Environment, JRE), которые используются MATLAB и MCR. Рекомендуется использовать JRE из MATLAB, каталог `matlabroot\sys\java\jre\Win32\jre.cfg`.

Замечание. Поддерживаемая версия JRE есть 1.5.0. Чтобы узнать, какая JRE используется, достаточно исполнить команду `version -java` в MATLAB, или обратиться к файлу `jre.cfg` в `matlabroot/sys/java/jre/win32` или `MCRroot/sys/java/jre/win32`. В дальнейшем предполагается, что на системе установлены JBuilder 2006.

Настройка системы. Перед началом работы нужно обязательно сделать настройки среды для совместимости MATLAB Builder для Java с Java SDK. Для этого необходимо определить следующие переменные среды:

- переменная `JAVA_HOME`;
- Java переменная `CLASSPATH`;
- пути для библиотек.

Java Builder использует переменную `JAVA_HOME`, чтобы определить местонахождение комплекта разработки программного обеспечения Java (Software Development Kit, SDK) на системе. Java Builder также использует эту переменную для того, чтобы найти файлы `javac.exe` и `jar.exe`, используемые в течение процесса компоновки.

Установка `JAVA_HOME` на Windows (машина разработки). При работе на Windows переменная `JAVA_HOME` устанавливается следующей командой в строке окна DOS. (В этом примере считается, что Java SDK установлен в каталоге `C:\Borland\JBuilder2006\jdk1.5\`)

```
set JAVA_HOME=C:\Borland\JBuilder2006\jdk1.5
```

Другой способ заключается в том, чтобы указать пути к необходимым каталогам Java. Для этого достаточно в командной строке DOS исполнить, например, следующую команду (удобно записать ее в bat-файл):

```
PATH C:\Borland\JBuilder2006\jdk1.5\;  
C:\Borland\JBuilder2006\jdk1.5\bin;  
C:\Borland\JBuilder2006\jdk1.5\jre;
```

Установка переменной `CLASSPATH` Java. Для создания и выполнения приложения Java, которое инкапсулирует функции MATLAB, система должна найти `.jar` файлы, содержащие библиотеки MATLAB, а также классы и определения методов, которые разработаны и созданы при помощи Java Builder. Для этого необходимо определить `classpath` или в команде `javac`, или в системных переменных среды. Переменная `CLASSPATH`, «путь класса», содержит каталоги, где постоянно находятся все `.class` и/или `.jar` файлы, необходимые для программы. Эти `.jar` файлы содержат любые классы, от которых зависит ваш класс Java.

При компиляции класса Java, который использует классы, содержащиеся в пакете `com.mathworks.toolbox.javabuilder`, необходимо включить файл **javabuilder.jar** в пути классов Java. Этот файл поставляется с Java Builder. Его можно найти в одном из следующих каталогов:

- `matlabroot/toolbox/javabuilder/jar` (машина разработки);
- `MCRroot/toolbox/javabuilder/jar` (машина конечного пользователя),

где `MCRroot` есть корневой каталог библиотек MCR. Java Builder автоматически включает этот `.jar` файл в пути класса при создании компонента. Для создания и

использования класса, созданного Java Builder, нужно добавить к путям класса следующее:

```
matlabroot/toolbox/javabuilder/jar/javabuilder.jar
```

На конечной машине (где нет MATLAB, но есть MCR) необходимо добавить путь:

```
PATH <mcr_root>\toolbox\javabuilder\jar;
```

Кроме того, нужно добавить к путям класса jar-файл, созданный Java Builder для ваших скомпилированных class-файлов.

Пример. Установка CLASSPATH на Windows. Предположим, что jar-файл созданного компонента mycomponent.jar находятся в C:\mycomponent. Для установки переменной CLASSPATH на машине разработки достаточно исполнить следующую команду в строке DOS (в одной строке!):

```
set CLASSPATH=.;C:\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;  
C:\mycomponent\mycomponent.jar
```

Другой способ заключается в том, чтобы указать пути класса в командной строке Java следующим образом. При этом не допускаются пробелы между именами путей. Например,

```
javac  
-classpath .;C:\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;  
C:\mycomponent\mycomponent.jar usemyclass.java
```

где usemyclass.java – это файл, который будет компилироваться. Отметим, что это одна строка и не должно быть пробела между javabuilder.jar; и c:\mycomponent\mycomponent.jar в этом примере.

Замечание. Команды настройки DOS являются достаточно длинными, поэтому их удобно поместить в bat-файл и исполнить его. Можно также добавить часто используемые пути класса к системной переменной CLASSPATH через панель управления Windows.

Пути для библиотек. При установке приложения на другую машину должны быть установлены пути для библиотек MATLAB, необходимых для работы созданного класса Java. В частности, для тестирования приложения должен быть установлен путь:

```
PATH <matlabroot>\bin\win32;
```

а для развертывания на другой машине – путь:

```
PATH <mcr_root>\<ver>\runtime\win32;
```

3.2.2. Графический интерфейс пользователя MATLAB Builder для Java

Графический интерфейс разработки проектов Java Builder предназначен для облегчения создания классов и компонент Java при помощи MATLAB Builder для Java. Графический интерфейс используется для создания классов Java, которые инкапсулируют функции m-файлов, для построения компонента Java и создания инсталляционного пакета для установки компонента на другую машину.

Для открытия графического интерфейса разработки достаточно выполнить следующую команду MATLAB:

```
deploytool
```

В результате открывается присоединяемое окно (рис. 3.2.1) справа от командного окна MATLAB и к строке меню MATLAB добавляется элемент меню **Project**. Это окно можно сделать и отдельным (рис. 3.2.2).

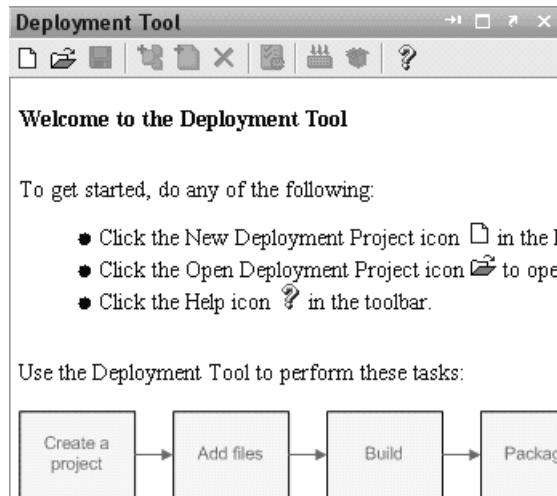


Рис. 3.2.1. Графический интерфейс Компилятора MATLAB

Инструментальная панель Deployment Tool имеет следующие кнопки:

- **New Project** – создание нового проекта;
- **Open Project** – просмотр проектов и выбор одного из них;
- **Save Project** – сохранение текущего проекта, включая все файлы и параметры настройки;
- **Add Class** – открытие диалогового окна **Add Class**, где можно определить название нового класса, который будет создан как часть текущего проекта (если компонент содержит классы);
- **Add File** – добавление файлов в папки проекта;
- **Remove** – удаление выбранной папки класса или выбранных файлов проекта.
- **Build** – построение компонента, определенного проектом с отображением процесса создания в окне вывода;
- **Package** – создание самоизвлекающегося .exe (Windows) или .zip файла (UNIX), который содержит файлы, нужны для использования компонента в приложении;
- **Settings** – изменение настроек проекта;
- **Help** – справка по использованию Deployment Tool.

При построении компонента внизу основного окна MATLAB открывается новое поле, в котором отражается информация о процедуре построения. Эти данные записываются также в файл **build.log** проекта. Данное окно вывода имеет дополнительные функциональные возможности, доступные через правую кнопку мыши. В частности, имеется возможность управлять действиями, зарегистрированными в окне вывода, при помощи опций **Back** и **Forward**, обновлять окно вывода и печатать его содержание. Опции **Selection** позволяют, после выбора определенного текста в окне вывода, получить следующее:

- **Evaluate Selection** – выполнить отмеченный текст, как будто это была команда, введенная в MATLAB;
- **Open Selection** – открыть выбранный файл, если отмеченный текст содержит правильный путь;
- **Help on Selection** – открыть справку MATLAB для выбранного текста, если этот текст есть документированная функция MATLAB.

3.2.3. Создание компонента Java

Для создания компонента нужно написать m-код, создать проект в MATLAB Builder для Java, который инкапсулирует этот код в классы Java и построить компонент.

Процесс построения. MATLAB Builder для Java использует переменную `JAVA_HOME`, чтобы определить местонахождение комплекта разработки программного обеспечения Java (SDK) на системе. Компилятор использует эту переменную, чтобы установить версию команды `javac.exe`, которая используется в течение компиляции. При создании компонента `mycomponent`, Java Builder делает следующее:

- генерирует код Java для создания компонента. Это следующие файлы:
 - `myclass.java` – содержит класс Java с описанием методов, инкапсулирующих m-функции проекта этого класса;
 - `mycomponentMCR.java` – содержит ключи расшифровки для файла CTF и код для инициализации MCR.
- компилирует код Java, созданный на первом шаге;
- создает подкаталоги `/distrib` и `/src`;
- создает технологический файл компонентный (`.ctf`) который содержит зашифрованные файлы MATLAB, созданные Java Builder;
- вызывает утилиту `Jar`, чтобы упаковать файлы классов Java в файл архива Java (`mycomponent.jar`).

Рассмотрим процедуру создания компонента на учебном примере MATLAB (примеры MATLAB Builder для Java находятся в каталоге `matlabroot\toolbox\javabuilder\Examples`).

Магический квадрат. Этот пример показывает, как создать компонент Java (**magicsquare**), который содержит класс **magic**, `jar`-файл, `ctf`-файл и другие файлы, необходимые для развертывания приложения. Этот класс инкапсулирует функцию MATLAB, **makesqr**, которая вычисляет магический квадрат. Дается также

пример приложения, **getmagic**, созданного на Java, которое в качестве результата отображает массив, полученный методом `makesqr` созданного класса `magic`. Приведем пошаговую процедуру создания приложения.

Напомним, что магический квадрат – это целочисленная матрица, обладающая следующим интересным свойством: суммы элементов каждой строки, каждого столбца и главных диагоналей равны.

Напомним также, что работая с Java нужно заботиться об установках переменных среды (см. раздел 3.2.1).

1. Подготовка к созданию проекта. Выберем для проекта следующий каталог: `D:\javabuilder_examples\magic_square`. Затем нужно выбрать `m`-функции, из которых будут создаваться классы и методы Java. Поскольку мы рассматриваем уже готовый учебный пример MATLAB, то просто скопируем содержание следующего каталога MATLAB: `malabroot\toolbox\javabuilder\Examples\MagicSquareExample` в каталог проекта `D:\javabuilder_examples\magic_square`. После копирования в рабочем каталоге проекта будут еще два подкаталога:

- `MagicDemoComp` – содержит `m`-функцию `makesqr.m`, которая вычисляет магический квадрат;
- `MagicDemoJavaApp` – содержит код Java, который будет использоваться для создания приложения.

Устанавливаем в качестве текущего каталога MATLAB новый подкаталог проекта `D:\javabuilder_examples\magic_square`.

Перед началом работы нужно также установить переменную среды `JAVA_HOME`, как было описано выше. Для этого в командной строке DOS (находясь в каталоге проекта) нужно исполнить, одну из следующих команд (для быстрого выполнения установок рекомендуется сохранять длинные команды в `bat`-файле):

```
set JAVA_HOME=C:\Borland\JBuilder2006\jdk1.5
PATH C:\Borland\JBuilder2006\jdk1.5%;
C:\Borland\JBuilder2006\jdk1.5\bin;
C:\Borland\JBuilder2006\jdk1.5\jre;
```

2. Создание нового проекта. Будем использовать графический интерфейс разработки `Deployment Tool`. Он запускается из MATLAB следующей командой:

```
deploytool
```

Для создания нового проекта нужно сделать несколько простых действий:

- выбрать создание нового проекта, это можно сделать из меню **File** \Rightarrow **New Deployment Project**, или кнопкой **New Deployment Project** в инструментальной панели;
- в открывшейся навигационной области окна, выбрать **MATLAB Builder for Java** и из списка компонентов, выбрать **Java Package** как тип компонента, который предполагается создать, в нижней части окна напечатать имя проекта **magicsquare.prj** (вместо `untitled1.prj`), проверить каталог, где создается проект и нажать **ОК**. Проект содержит две папки: **magicsquareclasses** и **Other files**;

- добавить файл `makesqr.m` в папку `magicsquareclasses` проекта. Для этого нужно эту папку активизировать и добавить в нее файл используя либо меню **Project**, либо кнопку инструментальной панели, либо правую кнопку мыши;
- сохранить проект.

Опция **Setting** позволяет задать параметры настройки проекта.

По умолчанию, имя проекта есть имя компонента. При создании нового проекта, диалоговое окно графического интерфейса разработки показывает папки, которые являются частью проекта. По умолчанию главная папка **magicsquareclasses** представляет класс компонента Java. По умолчанию имя класса – то же самое, что и имя проекта. Имя класса можно изменить используя либо меню **Project** ⇒ **Rename Class**, либо используя правую кнопку мыши при активизированном основном каталоге. Переименуем имя класса `magicsquareclasses` на **magic** и сохраним проект (рис. 3.2.2).

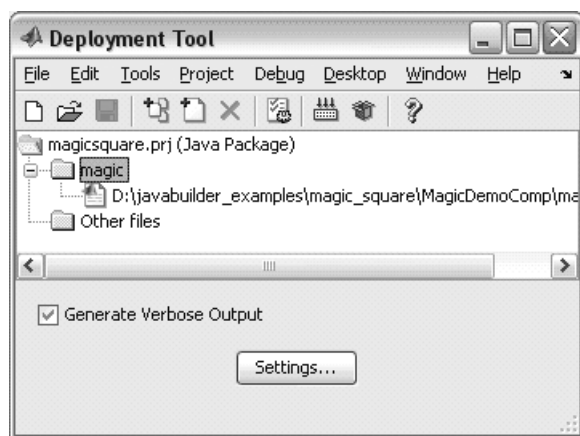


Рис. 3.2.2. Проект *magicsquare*

3. Построение пакета Java. Для построения компонента, инкапсулирующего функцию MATLAB `makesqr.m`, нужно исполнить команду **Build** из меню **Tools**, либо нажать кнопку построения (**Build**) на инструментальной панели. Начинается процесс построения и создается log-файл регистрации процесса построения, в котором записываются все операции и ошибки при построении компонента. В случае успешного исхода в каталоге проекта создается подкаталог **magicsquare**, содержащий два подкаталога **distrib** и **src**, в которые помещаются созданные файлы компонента.

Подкаталог **distrib** содержит файлы `magicsquare.ctf` и `magicsquare.jar`, которые предназначены для создания инсталляционного пакета для распространения. Напомним, что технологический файл компоненты `magicsquare.ctf` содержит за-

шифрованные m-функции, которые составляют компонент и все другие m-функции MATLAB, от которых зависят основные m-функции. Архив содержит все, основанное на MATLAB, содержание (m-файлы, MEX-файлы, и т.д.) связанное с компонентом. Файл `magicsquare.jar` есть архив, содержащий созданный пакет Java с байт-кодами `magic$1.class`, `magic.class` и `magicsquareMCR.class`.

Подкаталог **src** содержит копии файлов `magicsquare.ctf` и `magicsquare.jar`, log-файлы регистрации процесса построения, а также созданный файл `readme.txt`, который содержит полезную информацию для распространения пакета. Кроме того, в подкаталоге **classes** содержатся созданные классы компонента, а в подкаталоге **magicsquare** содержатся интерфейсные Java файлы.

4. Упаковка компонента Java. Для распространения созданного компонента удобно создать инсталляционный пакет. Это можно сделать командой **Package** из меню **Tools**, либо нажать кнопку упаковки (**Package**) на инструментальной панели **Deployment Tool**. В результате создается самораспаковывающийся архив **magicsquare_pkg.exe**, содержащий файлы `magicsquare.ctf`, `magicsquare.jar` и файл `_install.bat` для запуска установки библиотек MCR MATLAB на машине, где устанавливается компонент. Сам файл `MCRInstaller.exe` может быть также включен в инсталляционный пакет. Для этого нужно выбрать соответствующую опцию в **Setting**.

3.2.4. Использование командной строки для создания компонента

Для создания пакетов Java можно использовать интерфейс командной строки MATLAB (или командной строки операционной системы) вместо графического интерфейса пользователя. Для этого используется команда `mcc` с опциями. В этом случае проект не создается и подкаталоги `src` и `distrib` также не создаются.

Дадим краткий обзор некоторых опций `mcc`, связанных с созданием компонентов Java, вместе с синтаксисом и примерами их использования.

1. Создание класса, инкапсулирующего один или более М-файлов. Используемая опция `mcc`

`-W java:`

Описание. Указывает Java Builder создать компонент Java, который содержит класс, инкапсулирующий указанные файлы.

Синтаксис.

```
mcc -W 'java:component_name[,class_name]' file1 [file2...fileN]
```

Имя `component_name` является полным названием пакета для создаваемого компонента. Это имя есть разделенная точками строка.

Имя `class_name` является названием для класса Java, который будет создан. По умолчанию `class_name` является последним элементом в строке, которая определяет `component_name`.

Файлы `file1 [file2...fileN]` являются М-файлами, которые будут инкапсулированы как методы в `class_name`.

Пример.

```
mcc -W 'java:com.mycompany.mycomponent,myclass' foo.m bar.m
```

Пример создает Java компонент, который имеет полное название пакета `com.mycompany.mycomponent`. Этот компонент содержит единственный класс `Java, myclass`, который содержит методы `foo` и `bar`.

Для использования класса `myclass` в приложении нужно его импортировать в коде приложения следующим образом:

```
import com.mycompany.mycomponent.myclass;
```

2. Добавление дополнительных классов к компоненту Java. Используемая опция `mcc`

```
class{...}
```

Описание. Используется с `-W java:.`. Указывает Java Builder создать `class_name`, который инкапсулирует один или более М-файлов, которые определены в списке, разделенном запятыми.

Синтаксис.

```
class{class_name:file1 [file2...fileN]}
```

Пример.

```
mcc -W 'java:com.mycompany.mycomponent,myclass' foo.m
                                     bar.m class{myclass2:foo2.m,bar2.m}
```

Пример создает компонент Java, названный `mycomponent` с двумя классами:

- `myclass` имеет методы `foo` и `bar`;
- `myclass2` имеет методы `foo2` и `bar2`.

3. Упрощенный ввод командной строки. Используемая опция `mcc -B`.

Описание. Указывает Java Builder заменить указанный файл информацией командной строки, которую он содержит.

Синтаксис.

```
mcc -B 'bundlefile'[:arg1, arg2, ..., argN]
```

Пример. Предположим, что файл `myoptions` содержит строку

```
-W 'java:mycomponent,myclass'
```

В этом случае,

```
mcc -B 'myoptions' foo.m bar.m
```

производит то же самое, что и

```
mcc -W 'java:[mycomponent,myclass]' foo.m bar.m
```

4. Управление использованием MCR. Применяется опция `mcc -S`.

Описание. Предписывает Java Builder создавать единственный экземпляр MCR, когда возникает первый экземпляр класса Java. Этот MCR многократно используется и доступен для всех последующих экземпляров класса в пределах компонента, что приводит к более эффективному использованию памяти и уменьшает затраты запуска MCR для каждого последующего экземпляра класса. По умолчанию, для каждого экземпляра класса Java в компоненте создается новый экземпляр MCR. Использование `-S` изменяет значение по умолчанию. При ис-

пользовании `-S`, все экземпляры класса совместно используют единственное рабочее пространство MATLAB и совместно используют глобальные переменные в М-файлах, используемых для создания компонента. Это заставляет свойства класса Java вести себя в виде статических свойств, вместо экземплярных свойств.

Пример. Предположим, что файл `myoptions` содержит

```
mcc -S 'java:mycomponent,myclass' foo.m bar.m
```

Пример создает компонент Java, названный `mycomponent` содержащий единственный класс Java, названный `myclass` с методами `foo` и `bar`. (См. первый пример). Если в приложении возникают кратные экземпляры `myclass`, тогда инициализируется только один MCR и он доступен всем экземплярам `myclass`.

5. Определение каталога для вывода. Используемая опция `mcc`

`-ddirectoryname`

Описание. Предписывает Java Builder создать каталог и копировать выходные файлы в него (если используется `mcc` вместо GUI, то каталоги `project_directory\src` и `project_directory\distrib` автоматически не создаются).

Замечание. Все, указанные выше примеры использования командной строки производят следующие файлы:

- `mycomponent.jar` – архив классов Java компонента;
- `mycomponent.ctf` – технологический файл компонента.

Обратите внимание, что составляющее название при создании этих файлы получено из последнего элемента в разделенной точками строке, которая определяет полное составное имя класса.

3.2.5. Разработка приложения, использующего компонент

Использование компонента (`mycomponent.jar` и `mycomponent.ctf`) для разработки приложения Java может быть проведено как на исходной машине разработки, так и на конечной машине пользователя, без MATLAB. Предварительно среда исполнения MCR должна быть установлена на конечной машине.

Если компонент еще не установлен на машине, где создается приложение, то распаковка и установка компонента производится обычным образом. Нужно распаковать пакет, который был создан на последнем шаге создания компонента, в выбранный каталог на машине развития. Понятно, что это делать не нужно, если приложение разрабатывается на той же самой машине, где создан компонент Java.

Перечислим основные шаги по разработке приложения Java, использующего функции данного компонента.

1. Написание кода приложения. В этом коде Java нужно импортировать библиотеки MATLAB и классы компонента функцией Java `import`. Для создания экземпляра каждого класса, который предполагается использовать в приложении, нужно использовать в коде Java функцию `new`. Методы класса вызываются, так, как это делается с любым классом Java.
2. Установка переменных среды, которые требуются на машине развития.

3. Построение и тестирование приложения Java, как обычного приложения.
4. Установка приложения на машине развертывания.

Рассмотрим все эти этапы на примере создания приложения, использующего классы компонента `magicsquare`.

1. Создание кода приложения Java. Типовое приложение для этого примера находится в каталоге `MagicSquareExample\MagicDemoJavaApp\getmagic.java`. Приведем листинг этого файла.

```
/* Импорт необходимых пакетов */
import com.mathworks.toolbox.javabuilder.*;
import magicsquare.*;

/* класс getmagic вычисляет магический квадрат порядка n. Это
 * натуральное число n передается из командной строки */
class getmagic
{
    public static void main(String[] args)
    {
        MWNumericArray n = null; /* Входное значение */
        Object[] result = null; /* Результат, тип Object[] */
        magic theMagic = null; /* Объявление экземпляра класса magic */
try
    {
        /* Если нет параметра входа, exit */
        if (args.length == 0)
        {
            System.out.println("Error: must input a positive integer");
            return;
        }
        /* Преобразование входного значения в MWNumericArray */
        n = new MWNumericArray(Double.valueOf(args[0]),MWClassID.DOUBLE);

        /* Преобразование входного значения в String и его печать */
        System.out.println("Magic square of order " + n.toString());

        /* Создание нового объекта magic */
        theMagic = new magic();

        /* Вычисление магического квадрата и печать результата */
        result = theMagic.makesqr(1, n);
        System.out.println(result[0]);
    }
    catch (Exception e)
    {
        System.out.println("Exception: " + e.toString());
    }
    finally
```

```

    {
/* Освобождение своих ресурсов */
        MWArray.disposeArray(n);
        MWArray.disposeArray(result);
        if (theMagic != null)
            theMagic.dispose();
    }
}
}

```

В вызове метода, первый входной параметр определяет число выводов, которые метод должен вернуть. Этот ввод эквивалентен в вызываемом методе аргументу `argout` в функции `MATLAB`. Второй входной параметр есть ввод, определенный в объявлении функции в `m`-файле функции `makesqr`. Обратите внимание, что результат метода `makesqr(1, n)` возвращается как тип `Object[]` – это массив из одного элемента `result[0]`, который и содержит магический квадрат.

2. Установка переменных среды и компиляция приложения. Необходимо установить переменные `JAVA_HOME`, `CLASSPATH` и пути, как указано в разделе 3.2.1. В строке `DOS` нужно выполнить одну из следующих команд:

```

PATH C:\Borland\JBuilder2006\jdk1.5\;
C:\Borland\JBuilder2006\jdk1.5\jre;C:\Borland\JBuilder2006\jdk1.5\bin;
set JAVA_HOME=C:\Borland\JBuilder2006\jdk1.5

```

Для установки переменной `CLASSPATH` на машине разработки достаточно исполнить следующую команду в строке `DOS`:

```

set CLASSPATH=.;C:\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;
C:\mycomponent\mycomponent.jar

```

Другой способ заключается в том, чтобы указать пути класса в командной строке `Java` при создании класса из кода `usemyclass.java` следующим образом:

```

javac
    -classpath .;C:\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;
    C:\mycomponent\mycomponent.jar usemyclass.java

```

Например, находясь в каталоге проекта `D:\javabuilder_examples\magic_square`, для создания приложения `getmagic` из указанного выше кода **getmagic.java**, в строке `DOS` нужно выполнить следующую команду:

```

%JAVA_HOME%\bin/javac -classpath
matlabroot\toolbox\javabuilder\jar\javabuilder.jar;
.\magicsquare\distrib\magicsquare.jar
.\MagicDemoJavaApp\getmagic.java

```

Компиляция производит файл **getmagic.class** и помещает его в каталог `MagicDemoJavaApp`. Напомним, что если компонент создан с использованием `mcc`, то `Java Builder` не создает каталог `distrib`, поэтому пути должны быть скорректированы.

При использовании компонента на другой машине (без `MATLAB`) вместо подкаталога `MATLAB matlabroot/toolbox/javabuilder/jar` нужно указывать подкаталог `MCR`: `MCRroot/toolbox/javabuilder/jar`. Также должен быть указан путь `<mcr_root>\<ver>\runtime\win32` к библиотекам `MCR`.

3. Запуск приложения. Если для создания getmagic.class использовался графический интерфейс, то для запуска приложения getmagic нужно выполнить следующее (см. файл Run_M.bat):

- поместить файл getmagic.class в каталог запуска приложения D:\javabuilder_examples\magic_square\;
- определить пути для библиотек MATLAB (или для MCR)

```
PATH <matlabroot>\bin\win32;
```

или

```
PATH <mcr_root>\v76\runtime\win32;
```

- выполнить следующую команду в строке DOS (нужно указывать полный путь в командной строке к команде java):

```
matlabroot\sys\java\jre\win32\jre1.5.0_07\bin\java -classpath .;  
matlabroot\toolbox\javabuilder\jar\javabuilder.jar;  
magicsquare\distrib\magicsquare.jar getmagic 5
```

Для выполнения программы передается параметр, представляющий размерность магического квадрата. В этом примере, значение размерности есть 5. Программа конвертирует передаваемое в командной строке число в скалярное значение double, создает экземпляр класса magic, и вызывает метод makesqr на этом объекте. Этот метод вычисляет квадрат, используя функцию MATLAB magic. Программа getmagic отображает следующий вывод:

```
Magic square of order      5  
17  24   1         8      15  
23   5   7        14      16  
 4   6  13       20      22  
10  12  19       21       3  
11  18  25        2       9
```

4. Упаковка и распространение приложения Java. При упаковке и распределении компонента или приложения пользователям, нужно в инсталляционный пакет включить файлы поддержки, созданные Java Builder, а также библиотеки MCR MATLAB. Нужно также правильно установить пути и переменные среды. В частности должны быть установлены пути:

```
PATH <mcr_root>\toolbox\javabuilder\jar;<mcr_root>\<ver>\runtime\win32
```

При запуске приложения на другой машине (без MATLAB) вместо подкаталога MATLAB matlabroot/toolbox/javabuilder/jar нужно указывать подкаталог MCR, MCRroot/toolbox/javabuilder/jar. Также должен быть скорректированы пути для программы java и для jar-файла компонента.

3.2.6. Обсуждение примера магического квадрата

Пример магического квадрата показывает следующие аспекты написания приложения, используя компоненты, созданные MATLAB Builder для Java: импорт

классов, создание экземпляра класса, вызов методов класса из Java. Рассмотрим их подробнее.

Импорт классов. Необходимо импортировать библиотеки MATLAB и созданные классы компонента Java в код приложения. Для этого используется функция `Java import`.

```
import com.mathworks.toolbox.javabuilder.*;  
import componentname.classname; или import componentname.*;
```

Создание экземпляра класса. Как со всеми классами Java, нужно использовать функцию `new`, чтобы создать экземпляр класса. Для создания объекта (`theMagic`) из класса `magic`, пример приложения использует следующий код:

```
theMagic = new magic();
```

Вызов методов класса из Java. Как только создан экземпляр класса, можно вызвать метод класса, как это делается с любым объектом Java. В примере магического квадрата, метод `makesqr` вызывается строкой:

```
result = theMagic.makesqr(1, n);
```

где `n` является экземпляром класса `MWArray`. Он объявлен как `MWNumericArray` и преобразовывается к этому типу из строки ввода `args[0]` параметра следующей командой:

```
n = new MWNumericArray(Double.valueOf(args[0],  
                                     MWClassID.DOUBLE));
```

Когда вызывается метод компонента Java Builder, входные параметры, полученные методом должны быть во внутреннем формате массива MATLAB. Можно или (вручную) преобразовать их непосредственно в пределах программы запроса, или передать параметры как типы данных Java. Если данные передаются как типы данных Java, они преобразуются автоматически. Чтобы вручную преобразовать данные в один из стандартных типов данных MATLAB, используются классы `MWArray` в пакете `com.mathworks.toolbox.javabuilder`. Подробнее об этом см. разделе «Использование классов `MWArray`».

Отметим, что результат метода `makesqr` имеет тип `Java Object[]` – это массив из одного элемента `result[0]`, который и содержит магический квадрат.

Замечание. Поддержка особенностей MATLAB в Java. Java Builder обеспечивает устойчивое преобразование данных, индексацию и форматирование массивов для сохранения гибкости среды MATLAB при вызове из кода Java. Чтобы поддерживать типы данных MATLAB, Java Builder обеспечивает иерархию классов `MWArray`. Можно использовать `MWArray` и другие элементы класса Java в приложении для конвертации собственных массивов в массивы MATLAB и наоборот. Java Builder также обеспечивает автоматическое преобразование данных для того, чтобы передать параметры, которые являются типами Java.

3.3. Массивы MATLAB в Java

Хотя приложение, разработанное при помощи Java Builder может работать независимо от MATLAB, оно использует среду MCR выполнения компонентов MATLAB. Для того, чтобы из программы на Java можно было обратиться к библиотеке MCR

MATLAB и методам и данным, основанным на Java Builder, имеется специальный класс `Java MWArray` из пакета `com.mathworks.toolbox.javabuilder.MWArray`. Этот класс `MWArray` разработан для обеспечения связи Java с MATLAB. Класс `MWArray` содержит массив MATLAB и имеет набор методов для обращения к свойствам и данным массива. Объекты класса `MWArray` – это аналоги массивов MATLAB в Java. Для краткости будем их в дальнейшем называть массивами MATLAB. Класс `MWArray` также имеет методы для преобразования массивов MATLAB в стандартные типы Java.

Java Builder имеет иерархию классов, которые представляют главные типы массивов MATLAB. Корневой класс есть `MWArray`, который имеет следующие подклассы:

- `MWNumericArray` – для работы с числовыми массивами;
- `MWLogicalArray` – для работы с логическими массивами;
- `MWCharArray` – для работы с символьными массивами;
- `MWCellArray` – для работы с массивами ячеек;
- `MWStructArray` – для работы с массивами структур;

Эти подклассы имеют конструкторы и методы для создания новых массивов MATLAB из стандартных типов и объектов Java. Массивы MATLAB можно использовать как аргументы при вызове в Java метода, созданного Java Builder.

В этом параграфе мы рассмотрим класс `MWArray` и его подкласс `MWNumericArray`. Использование других классов: `MWLogicalArray`, `MWCharArray`, `MWStructArray` и `MWCellArray` аналогично, их описание можно найти в документации MATLAB Builder для Java.

3.3.1. Использование методов класса `MWArray`

Класс `MWArray` обеспечивает набор методов для обращения к свойствам массива и данным MATLAB. Класс `MWArray` также имеет методы для преобразования массивов MATLAB в стандартные типы Java.

В данном разделе дадим описание использования методов классов `MWArray`. Рассмотрим вопросы построения, создания и разрушения `MWArray`, методы доступа к данным `MWArray` и методы копирования, преобразования и сравнения массивов `MWArray`.

В результате подмены методов разные классы могут иметь методы с одинаковыми именами, но выполняющие различные действия. Для интеграции с программой Java, класс `MWArray` обеспечивает подмены для методов `java.lang.Object` и необходимые средства, требуемые интерфейсом Java. Например, класс `MWArray` имеет следующие подмены методов:

- **`equals`** – подмена метода `Object.equals`. Обеспечивает проверку логического равенства двух `MWArrays`. Этот метод делает побайтовое сравнение. Поэтому, два экземпляра `MWArray` логически равны, когда они имеют тот же самый тип MATLAB и имеют идентичный размер, форму, и содержание;

- **hashCode** – подмена метода `Object.hashCode`;
- **toString** – подмена метода `Object.toString` такая, чтобы объекты `MWArray` печатались должным образом. Этот метод формирует новую `java.lang.String` из лежащего в основе массива `MATLAB` так, чтобы вызовы `System.out.println` с аргументом `MWArray` давали бы тот же самый вывод как при выводе массива в `MATLAB`.
- **finalize** – подмена метода `Object.finalize`. Для разрушения лежащего в основе массива `MATLAB` при сборке мусора. Этот метод имеет защищенный доступ.

Класс `MWArray` также обеспечивает ряд методов базового класса, которые являются общими для всех подклассов `MWArray`. Рассмотрим эти методы подробнее.

Построение и удаление `MWArray`

Для создания пустого двумерного объекта `MWArray` используется конструктор **`MWArray()`**. Тип, данный этому объекту, есть `MWClassID.UNKNOWN`.

Пример. Создание пустого объекта `MWArray`:

```
MWArray A = new MWArray();
```

Для удаления объекта класса `MWArray` или любого из его дочерних классов используются методы **`dispose`** и **`disposeArray`**.

Метод `dispose`. Этот метод удаляет массив `MATLAB`, который содержит объект `MWArray` и освобождает память, занятую массивом. Например, создание и затем разрушение объекта `MWArray`:

```
MWArray A = new MWArray();
A.dispose();
```

Метод `disposeArray`. Этот метод удаляет любые массивы `MATLAB`, содержащиеся во входном объекте и освобождает память, занятую ими. Это статический метод класса. Прототипом для метода `disposeArray` является:

```
public static void disposeArray(Object arr)
```

Метод имеет один входной параметр: **`arr`** – объект для удаления. Если этот входной объект представляет единственный экземпляр `MWArray`, тогда этот экземпляр удаляется при вызове метода `dispose()`. Если входной объект представляет массив экземпляров `MWArray`, то удаляется каждый объект в массиве. Если входной объект представляет массив `Object` или многомерный массив, то массив рекурсивно обрабатывается для освобождения каждого `MWArray` содержащийся в массиве. Например, создание и затем удаление массива числовых объектов:

```
MWArray[] MArr = new MWArray[10];
for (int i = 0; i < 10; i++)
    MArr[i] = new MWNumericArray();
MArr.disposeArray(MArr);
```

Методы получения информации о `MWArray`

Для получения информации об объекте класса `MWArray` или любого из его дочерних классов используются методы: **`classID`**, **`getDimensions`**, **`isEmpty`**, **`numberOfDimensions`**, **`numberOfElements`**.

Примеры ниже используют объект A, являющийся массивом 3-на-6 `MWNumericArray`, созданным кодом Java:

```
int[][] Adata = {{ 1, 2, 3, 4, 5, 6},
                 { 7, 8, 9, 10, 11, 12},
                 {13, 14, 15, 16, 17, 18}};
MWNumericArray A = new MWNumericArray(Adata, MWClassID.INT32);
```

Метод `classID`. Возвращает тип MATLAB объекта `MWArray`. Тип возвращения – поле, определенное классом `MWClassID`. Например, получение `ClassID` для объекта `MWNumericArray`, созданного выше:

```
System.out.println("Class of A is " + A.classID());
```

После выполнения пример отображает этот вывод:

```
Class of A is int32
```

Метод `getDimensions`. Этот метод возвращает одномерный `int` массив, содержащий размер каждого измерения объекта `MWArray`. Например, получение измерений массива `MWArray`:

```
int[] dimA = A.getDimensions();
System.out.println("Dimensions of A are " +
                  dimA[0] + " x " + dimA[1]);
```

После выполнения пример отображает этот вывод:

```
Dimensions of A are 3 x 6
```

Метод `isEmpty`. Этот метод возвращает `true` если массив объекта не содержит никаких элементов, и `false` иначе. Например, проверка на пустоту `MWArray`. Отображает сообщение если объект массив A является пустым массивом. Иначе, отображает содержание A:

```
if (A.isEmpty())
    System.out.println("Matrix A is empty");
else
    System.out.println("A = " + A.toString());
```

После выполнения пример отображает содержание A:

```
A = 1    2    3    4    5    6
     7    8    9   10   11   12
    13   14   15   16   17   18
```

Метод `numberOfDimensions`. Этот метод возвращает число измерений массива объекта. Например, получение число измерений `MWArray`:

```
System.out.println("Matrix A has " + A.numberOfDimensions() +
                  "dimensions");
```

После выполнения пример отображает следующее:

```
Matrix A has 2 dimensions
```

Метод `numberOfElements`. Возвращает общее количество элементов в массиве объекта. Например, получение числа элементов `MWArray`:

```
System.out.println("Matrix A has " + A.numberOfElements() +
                  "elements");
```


После выполнения пример отображает следующее:

```
Matrix A has 18 elements
```

Методы получения и задания данных в MWArray

Доступ к элементам массивов. Обратиться непосредственно к данным лежащего в основе массива MATLAB невозможно. Вместо этого используются методы **set** и **get** чтения или изменения элемента массива. Методы **set** и **get** поддерживают как простую индексацию через единственный индекс, так и многомерную – можно указать набор **int** индексов требуемых значений. В случае массивов структур также поддерживается индексация по имени поля.

Для получения и установки значения в объекте класса **MWArray** или любом из его дочерних классов используются методы: **get**, **getData**, **set**, **toArray**.

Метод get. Этот метод возвращает элемент, указанный индексом, или элемент, соответствующий многомерному индексу массива. Элемент возвращается как **Object**. Метод используется так:

```
public Object get(int index)
public Object get(int[] index)
```

Первый синтаксис (**int index**) используется для возвращения элемента указанного 1-мерной индексацией MATLAB (напомним, что в MATLAB индексация начинается с 1 и элементы нумеруются в постолбцовом порядке). Второй синтаксис (**int[] index**) используется для возвращения элемента многомерного массива по указанному многомерному массиву индексов. Первый синтаксис работает лучше, чем второй. В случае, где **index** имеет тип **int[]**, каждый элемент вектора **index** есть индекс по одному измерению объект **MWArray**. Правильный диапазон для любого индекса $1 \leq \text{index}[i] \leq N[i]$, где **N[i]** является размером **i**-го измерения.

Приведем пример получения значения **MWArray** методом **get**. Поиск элемента (2, 4) из массива объекта **A**:

```
int[] index = {2, 4};
Object d_out = A.get(index);
System.out.println("Data read from A(2,4) is " + d_out.toString());
```

После выполнения пример отображает следующее:

```
Data read from A(2,4) is 10
```

Метод getData. Этот метод возвращает все элементы объекта **MWArray**. Элементы возвращаются в одномерном массиве типа **Object**, в постолбцовом порядке. Элементы возвращенного массива преобразуются согласно правилам преобразования значения по умолчанию. Если основной массив MATLAB – комплексный числовой тип, **getData** возвращает вещественную часть. Если основной массив разрежен, то возвращается массив, содержащий элементы отличные от нуля. Если основной массив – массив ячеек или структур, тогда **toArray** рекурсивно вызывается на каждом элементе.

Пример. Получение данных из объекта **MWArray A**, с приведением типа от **Object** к **int**:

```
System.out.println("Data read from matrix A is:");
```

```
int[] x = (int[]) A.getData();
for (int i = 0; i < x.length; i++)
    System.out.print(" " + x[i]);
System.out.println();
```

После выполнения пример отображает следующее:

```
Data read from matrix A is:
1 7 13 2 8 14 3 9 15 4 10 16 5 11 17 6 12 18
```

Метод set. Этот метод заменяет элемент по указанному индексу в объекте `MWArray` на входной элемент. Метод используется так:

```
public void set(int index, Object element)
public void set(int[] index, Object element)
```

Первый синтаксис (`int index`) используется для замены элемента указанного одномерным индексом MATLAB (в столбцовом порядке). Второй синтаксис (`int[] index`) используется для возвращения элемента многомерного массива по указанному многомерному индексу массива. Первый синтаксис работает лучше, чем второй.

Входные параметры:

- `element` – новый элемент для замены в `index`;
- `index` – индекс нужного элемента в `MWArray`.

Пример. Изменение данных в элементе (2, 4) из объекта `MWArray A`:

```
int[] index = {2, 4};
A.set(index, 555);
Object d_out = A.get(index);
System.out.println("Data read from A(2,4) is " + d_out.toString());
```

После выполнения пример отображает следующее:

```
Data read from A(2,4) is 555
```

Метод toArray. Этот метод создает массив Java, содержащий копию данных из лежащего в основе массива MATLAB. Возвращенный массив имеет ту же самую размерность как основной массив MATLAB. Прототипом для метода `toArray` является `public Object[] toArray()`.

Элементы возвращенного массива преобразуются согласно правилам преобразования значения по умолчанию. Если основной массив MATLAB – комплексный числовой тип, `toArray` возвращает вещественную часть. Если основной массив разрежен, то возвращается полное представление массива. Если основной массив – массив ячеек или структур, тогда `toArray` рекурсивно вызывается на каждом элементе.

Пример. Создание и вывод копии объекта `MWArray A`:

```
int[][] x = (int[][]) A.toArray();
int[] dimA = A.getDimensions();

System.out.println("Matrix A is:");
for (int i = 0; i < dimA[0]; i++)
{
    for (int j = 0; j < dimA[1]; j++)
```

```

        System.out.print(" " + x[i][j]);
        System.out.println();
    }

```

После выполнения пример отображает следующее:

```

Matrix A is:
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18

```

Замечание. Метод `toArray` может применяться и в виде `toArray` для получения массива определенного типа. Например, `toArray` – для получения массива типа `double`, `toArray` – для получения массива типа `int`.

Методы копирования, преобразования и сравнения массивов `MWArray`

Для копирования, преобразования и сравнения объектов класса `MWArray` или любого из его дочерних классов используются методы: **`clone`, `compareTo`, `equals`, `hashCode`, `sharedCopy`, `toString`**.

Метод `clone`. Этот метод создает и возвращает реальную (deep) копию объекта `MWArray`. Поскольку `clone` создает новый массив, любые изменения, сделанные в этом новом массиве не отражаются в оригинале. Например, создание копии объекта `A` `MWArray`:

```
Object C = A.clone();
```

Метод `compareTo`. Этот метод сравнивает `MWArray` объект со входным объектом. Он возвращает отрицательное целое число, нуль, или положительное целое число если `MWArray` объект – меньше, равный, или больше, чем указанный объект, соответственно.

Пример. Создание общедоступной копии объекта `MWArray` и затем сравнение ее с оригинальным объектом. Нулевое возвращаемое значение указывает, что два объекта равны:

```

Object S = A.sharedCopy();
if (A.compareTo(S) == 0)
    System.out.println("Matrix S is equal to matrix A");

```

После выполнения пример отображает следующее:

```
Matrix S is equal to matrix A
```

Метод `equals`. Этот метод указывает, равен ли объект `MWArray` входному объекту. Метод `equals` класса `MWArray` подменяет метод `equals` класса `Object`.

Пример. Создание общедоступной копии объекта `MWArray` и затем сравнение ее с оригинальным объектом. Возвращаемое значение `true` указывает, что два объекта равны:

```

Object S = A.sharedCopy();
if (A.equals(S))
    System.out.println("Matrix S is equal to matrix A");

```

Метод hashCode. Этот метод возвращает значение хеш-кода для объекта `MWArray`. Метод `hashCode` класса `MWArray` подменяет метод `hashCode` класса `Object`. Например, получение хеш-кода для `MWArray` объекта `A`:

```
System.out.println("Hash code for matrix A is " + A.hashCode());
```

После выполнения пример отображает следующее:

```
Hash code for matrix A is 456687478
```

Метод sharedCopy. Этот метод создает и возвращает общедоступную копию массива. Общедоступная копия есть указатель на лежащий в основе оригинальный массив MATLAB. Любые изменения, сделанные в копии отражаются в оригинале. Например, создание общедоступной копии `MWArray` объекта `A`:

```
Object S = A.sharedCopy();
```

Метод toString. Этот метод возвращает строковое представление массива. Метод `toString` класса `MWArray` подменяет метод `toString` класса `Object`.

Пример. Отображение содержания `MWArray` объекта `A`:

```
System.out.println("A = " + A.toString());
```

После выполнения пример отображает следующее содержание `A`:

```
A = 1   2   3   4   5   6
     7   8   9  10  11  12
    13  14  15  16  17  18
```

Методы для использования на разреженных массивах `MWArray`

Для получения информации относительно разреженных массивов типа `MWArray` или любого из его дочерних классов используются методы: **`isSparse`**, **`columnIndex`**, **`rowIndex`**, **`maximumNonZeros`**, **`numberOfNonZeros`**.

В следующих ниже примерах используется разреженный объект `MWArray`, созданный с использованием метода **`newSparse`** класса `MWNumericArray`:

```
double[] Adata = { 0, 10, 0, 0, 40, 50, 60, 0, 0, 90};
int[] ri = {1, 1, 1, 1, 1, 2, 2, 2, 2};
int[] ci = {1, 2, 3, 4, 5, 1, 2, 3, 4, 5};
MWNumericArray A = MWNumericArray.newSparse(ri, ci, Adata,
                                             MWClassID.DOUBLE);
System.out.println(A.toString()); // Содержание разреженного MWArray
(2,1)      50
(1,2)      10
(2,2)      60
(1,5)      40
(2,5)      90
```

Метод isSparse. Проверка разреженности массива. Метод возвращает `true` если `MWArray` объект разрежен, и `false` иначе. Например, проверка на разреженность созданного выше объекта `A` `MWArray`:

```
if (A.isSparse())
    System.out.println("Matrix A is sparse");
```

После выполнения пример отображает следующее:

Matrix A is sparse

Метод columnIndex. Этот метод возвращает массив, содержащий индекс столбца каждого элемента в основном массиве MATLAB. Например, получение индексов столбцов разреженного массива A MWArray.

```
System.out.print("Column indices are: ");
int[] colidx = A.columnIndex();
for (int i = 0; i < 5; i++)
    System.out.print(colidx[i] + " ");
System.out.println();
```

После выполнения пример отображает следующее:

Column indices are: 1 2 2 5 5

Метод rowIndex. Этот метод возвращает массив, содержащий индексы строк каждого элемента в основном массиве MATLAB. Например,

```
int[] rowidx = A.rowIndex();
```

Метод maximumNonZeros. Этот метод возвращает вместимость разреженного массива. Если основной массив неразрезан, этот метод возвращает число элементов. Например, получение максимального числа ненулевых элементов в массиве A MWArray:

```
System.out.println("Maximum number of nonzeros for matrix A is "
    + A.maximumNonZeros());
```

После выполнения пример отображает следующее:

Maximum number of nonzeros for matrix A is 10

Метод numberOfNonZeros. Этот метод возвращает число отличных от нуля элементов в разреженном массиве. Если основной массив неразрезан, этот метод возвращает число всех элементов. Например, получение числа ненулевых элементов в A:

```
System.out.println("The number of nonzeros for matrix A is " +
    A.numberOfNonZeros());
```

После выполнения пример отображает следующее:

The number of nonzeros for matrix A is 5

3.3.2. Использование MWNumericArray

В данном разделе дадим описание использования методов классов MWNumericArray. Рассмотрим вопросы построения, создания и разрушения MWNumericArray, методы доступа к данным MWArray и методы копирования, преобразования и сравнения массивов MWArray, методы для разреженных массивов и специальные константы.

Класс MWNumericArray обеспечивает интерфейс Java для числового массива MATLAB. Экземпляр этого класса может хранить массив MATLAB типа: double, single, int8, uint8, int16, int32, uint32, int64, и uint64. Экземпляр класса MWNumericArray

может быть вещественным или комплексным, плотным или разреженным (разреженный формат поддерживается только для типа double).

Класс `MWNumericArray` поддерживает следующие простые типы Java: `double`, `float`, `byte`, `short`, `int`, `long`, `boolean`. Поддерживаются также типы объектов подклассов `java.lang.Number`, `java.lang.String` и `java.lang.Boolean`. Также поддерживаются общие *N*-мерные массивы каждого типа.

Построение различных типов числовых массивов

Для построения массивов типа `MWNumericArray` можно использовать конструкторы и статический метод `newInstance`.

Использование конструкторов. Конструктор класса `MWNumericArray` имеет вид `MWNumericArray()`. В случае отсутствия аргументов создается пустой массив `double`, а при наличии аргументов – различные типы класса `MWNumericArray`:

- `MWNumericArray()` – создание пустого массив типа `double`;
- `MWNumericArray(MWClassID)` – пустой массив типа, определенного указанием `MWClassID`;
- `MWNumericArray(type)` – вещественный массив типа, определенного правилами преобразования по умолчанию;
- `MWNumericArray(javatype, MWClassID)` – вещественный массив типа, определенного указанием `MWClassID`;
- `MWNumericArray(javatype, javatype)` – комплексный массив типа, определенного правилами преобразования по умолчанию.
- `MWNumericArray(javatype, javatype, MWClassID)` – комплексный массив типа, определенного указанием `MWClassID`;

Если тип **MWClassID** возвращаемого `MWNumericArray` не указан, то он определяется правилами преобразования по умолчанию, как показано в табл. 3.3.1.

Таблица 3.3.1. Тип возвращаемого конструктором массива `MWNumericArray`

Ввод <code>javatype</code>	Класс ID <code>MWNumericArray</code>
<code>double</code>	<code>MWClassID.DOUBLE</code>
<code>float</code>	<code>MWClassID.SINGLE</code>
<code>long</code>	<code>MWClassID.INT64</code>
<code>int</code>	<code>MWClassID.INT32</code>
<code>short</code>	<code>MWClassID.INT16</code>
<code>byte</code>	<code>MWClassID.INT8</code>

В случае явного указания типа `MWClassID` возвращаемого `MWNumericArray`, тип Java **javatype** входных значений может быть любым из следующих: `double`, `float`, `long`, `int`, `short`, `byte`, `String`, `boolean`, `Object`.

Приведем некоторые примеры, показывающие, как создать различные типы числовых массивов с различными формами конструктора `MWNumericArray`.

Пример. Построение пустого числового скаляра типа `int64`:

```
MWNumericArray A = new MWNumericArray(MWClassID.INT64);
System.out.println("A = " + A);
```

После выполнения пример отображает следующее:

```
A = []
```

Пример. Создание скалярного числового массива типа MWClassID.INT16:

```
double AReal = 24;
MWNumericArray A = new MWNumericArray(AReal, MWClassID.INT16);
System.out.println("Array A of type " + A.classID() + " = \n" + A);
```

После выполнения пример отображает следующее:

```
Array A of type int16 = 24
```

Пример. Создание комплексного числового скаляра:

```
double AReal = 24;
double AImag = 5;
MWNumericArray A = new MWNumericArray(AReal, AImag);
System.out.println("Array A of type " + A.classID() + " = \n" + A);
```

Результат выполнения этого примера:

```
Array A of type double =
                24.0000 + 5.0000i
```

Пример. Создание 3-на-6 вещественного массива типа MWClassID.SINGLE:

```
double[][] AData = {{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9, 10, 11, 12},
                    {13, 14, 15, 16, 17, 18}};
MWNumericArray A = new MWNumericArray(AData, MWClassID.SINGLE);
System.out.println("Array A = \n" + A);
```

После выполнения пример отображает следующее:

```
A = 1   2   3   4   5   6
    7   8   9   10  11  12
    13  14  15  16  17  18
```

Пример. Создание 2-на-2 матрицы типа double со следующими значениями: [1 2; 3 4] из значений Java разных типов.

```
double[][] x1 = {{1.0, 2.0}, {3.0, 4.0}};
int[][] x2 = {{1, 2}, {3, 4}};
Double[][] x3 = {{new Double(1.0), new Double(2.0)},
                 {new Double(3.0), new Double(4.0)}};
String[][] x4 = {{ "1.0", "2.0"}, {"3.0", "4.0"}};
```

```
MWNumericArray a1 = new MWNumericArray(x1, MWClassID.DOUBLE);
MWNumericArray a2 = new MWNumericArray(x2, MWClassID.DOUBLE);
MWNumericArray a3 = new MWNumericArray(x3, MWClassID.DOUBLE);
MWNumericArray a4 = new MWNumericArray(x4, MWClassID.DOUBLE);
```

Пример. Создание комплексного скаляра типа int32 массив со значением 1+2i из значений Java разных типов:

```
MWNumericArray a1 = new MWNumericArray(1, 2);
MWNumericArray a2 = new MWNumericArray(1.0, 2.0, MWClassID.INT32);
MWNumericArray a3 = new MWNumericArray(new Double(1.0),
```

```

        New Integer(2), MWClassID.INT32);
MWNumericArray a4 = new MWNumericArray("1.0", "2.0",
        MWClassID.INT32);

```

Пример. Создание 1-на-3 комплексного массива MWClassID.DOUBLE:

```

double[] AReal = {24.2, -7, 113};
double[] AImag = {5, 31, 27};
MWNumericArray A = new MWNumericArray(AReal, AImag, MWClassID.DOUBLE);
System.out.println("Array A of type " + A.classID() + " = \n" + A);

```

После выполнения пример отображает следующее:

```

Array A of type double =
1.0e+002 *
    0.2420 + 0.0500i  -0.0700 + 0.3100i   1.1300 + 0.2700i

```

Построение N-мерных массивов. Конструкторы MWNumericArray также поддерживают многомерные массивы всех поддерживаемых типов. Например, можно создать 2-на-2-на-3 массив double следующими двумя инструкциями:

```

Double[][][] x1 = { { {1.0, 2.0, 3.0}, {4.0, 5.0, 6.0} },
                    { {7.0, 8.0, 9.0}, {10.0, 11.0, 12.0} } };
MWNumericArray a1 = new MWNumericArray(x1);

```

Построение зубчатых (Jagged) массивов. Предыдущие примеры создавали прямоугольные массивы Java и использовали эти массивы для инициализации массивов MATLAB. Многомерные массивы в Java являются массивами массивов, что означает возможность создания массива Java, в котором каждая строка может иметь различное число столбцов. Такие массивы обычно называют зубчатыми массивами.

Конструктор MWNumericArray поддерживает построение зубчатых массивов, создавая прямоугольный массив и дополняя нулями недостающие элементы. Окончательный массив MATLAB будет иметь размер столбца равный первому размеру зубчатого массива. Например, следующие инструкции создают 5-на-5 матрицу double из следующего зубчатого массива:

```

double[][] pascalsTriangle = {
    {1.0},
    {1.0, 1.0},
    {1.0, 2.0, 1.0},
    {1.0, 3.0, 3.0, 1.0},
    {1.0, 4.0, 6.0, 4.0, 1.0}
};
MWNumericArray a1 = new MWNumericArray(pascalsTriangle);

```

Результирующий массив MATLAB имеет следующую структуру:

```

[1 0 0 0 0
 1 1 0 0 0
 1 2 1 0 0
 1 3 3 1 0
 1 4 6 4 1]

```


Использование метода newInstance. Другим способом создания числовых массивов является использование метода newInstance класса MWNumericArray. Метод создает вещественный или комплексный массив указанных измерений, типа и комплексности. Это – статический метод класса и, таким образом, не должен вызываться в ссылке на экземпляр класса. Отметим, что этот метод работает лучше, чем конструктор класса. Отметим также, что данные этому методу должны передаваться в виде одномерного массива Java, расположенными в постолбцовом порядке. Метод может использоваться в виде:

- для создания инициализированного нулями вещественного или комплексного числового массива,

```
newInstance(int[] dims, MWClassID classid, MWComplexity cmplx);
```
- для создания и инициализации вещественного числового массива,

```
newInstance(int[] dims, Object rData, MWClassID classid);
```
- для создания и инициализации комплексного числового массива,

```
newInstance(int[] dims, Object rData, Object iData, MWClassID classid);
```

Входные параметры:

- dims – массив неотрицательных размеров измерений;
- classId – представление типа MWClassID массива MATLAB;
- rData – данные для инициализации вещественной части массива. Нужно форматировать массив rData в постолбцовом порядке;
- iData – данные для инициализации мнимой части массива, в постолбцовом порядке.

Правильные типы для realData и imagData следующие: double[], float[], long[], int[], short[], byte[], String[], boolean[], одномерные массивы любого подкласса java.lang.Number и одномерные массивы java.lang.Boolean

Пример. Задание 2-на-2 матрицы MWNumericArray из данных Java разных типов.

```
double[] x1 = {1.0, 3.0, 2.0, 4.0};
int[] x2 = {1, 3, 2, 4};
Double[] x3 = {new Double(1.0),
               new Double(3.0),
               new Double(2.0),
               new Double(4.0)};
String[] x4 = {"1.0", "3.0", "2.0", "4.0"};

int[] dims = {2, 2};
MWNumericArray a1 =
    MWNumericArray.newInstance(dims, x1, MWClassID.DOUBLE);
MWNumericArray a2 =
    MWNumericArray.newInstance(dims, x2, MWClassID.DOUBLE);
MWNumericArray a3 =
    MWNumericArray.newInstance(dims, x3, MWClassID.DOUBLE);
MWNumericArray a4 =
    MWNumericArray.newInstance(dims, x4, MWClassID.DOUBLE);
```

Методы уничтожения *MWNumericArray*

Для освобождения памяти, распределенной под массивы, используются методы *MWNumericArray* **dispose** или **disposeArray**, унаследованные от класса *MWArray* (см. раздел 3.3.1).

Методы для получения информации о *MWNumericArray*

Для получения информации об объекте класса *MWNumericArray* используются методы: **classID**, **complexity**, **getDimensions**, **isEmpty**, **isFinite**, **isInf**, **isNaN**, **numberOfDimensions**, **numberOfElements**.

Метод classID. Возвращает тип массива MATLAB. Этот метод *MWNumericArray* наследует от класса *MWArray*.

Метод complexity. Определение комплексности, или вещественности. Этот метод возвращает свойство комплексности объекта *MWNumericArray* как *MWComplexity.REAL* – для вещественного массива, или *MWComplexity.COMPLEX* – для комплексного массива.

Пример. Определение, является ли матрица *A* вещественной или комплексной:

```
double AReal = 24;
double AImag = 5;
MWNumericArray A = new MWNumericArray(AReal, AImag);
System.out.println("A is a " + A.complexity() + " matrix");
```

После выполнения пример отображает следующий вывод:

```
A is a complex matrix
```

Метод getDimensions. Возвращает массив, содержащий размер каждого измерения массива. Этот метод *MWNumericArray* наследует от класса *MWArray*.

Метод isEmpty. Проверка на пустоту. Этот метод *MWNumericArray* наследует от класса *MWArray*.

Метод isFinite. Проверка конечности значений машинно-независимым способом. Например, проверить *x* на конечность:

```
double x = 25;
if (MWNumericArray.isFinite(x))
    System.out.println("The input value is finite");
```

После выполнения примера получается следующий вывод:

```
The input value is finite
```

Метод isInf. Этот метод проверяет значение на бесконечность машинно-независимым способом. Например, проверим *x* на бесконечность:

```
double x = 1.0 / 0.0;
if (MWNumericArray.isInf(x))
    System.out.println("The input value is infinite");
```

После выполнения примера получается следующий вывод

```
The input value is infinite
```

Метод isNaN. Этот метод проверяет на неопределенность (NaN) машинно-независимым способом. Например, проверим *x* на NaN:

```
double x = 0.0 / 0.0;  
if (MWNumericArray.isNaN(x))  
    System.out.println("The input value is not a number.");
```

Метод `numberOfDimensions`. Возвращает число измерений массива. Этот метод `MWNumericArray` наследует от класса `MWArray`.

Метод `numberOfElements`. Возвращает общее количество элементов в массиве. Этот метод `MWNumericArray` наследует от класса `MWArray`.

Методы доступа к элементам и задания элементов `MWNumericArray`

Класс `MWNumericArray` обеспечивает методы для обращения к данным и изменения данных массива в форме методов **get** и **set**. Ниже перечислены методы `get` и `set`. Напомним, что индексация начинается с единицы, как в MATLAB, а не с нуля, как принято в Java.

Метод `getType(int)`. Возвращает вещественную часть элемента по указанному индексу. Возвращаемое значение имеет определенный тип **type** (например, `getDouble` возвращает `double`).

Метод `getType(int[])`. Возвращает вещественную часть элемента, определенного индексным массивом. Возвращаемое значение имеет определенный тип **type** (например, `getDouble` возвращает `double`).

Метод `getImagtype(int)`. Возвращает мнимую часть элемента по указанному индексу. Возвращаемое значение имеет определенный тип **type** (например, `getImagDouble` возвращает `double`).

Метод `getImagtype(int[])`. Возвращает мнимую часть элемента, определенного индексным массивом. Возвращаемое значение имеет определенный тип **type** (например, `getDouble` возвращает `double`).

Метод `set(int, type)`. Заменяет вещественную часть элемента по указанному индексу поставляемым значением.

Метод `set(int[], type)`. Заменяет вещественную часть элемента, определенного индексным массивом, на указанное значение.

Метод `setImag(int, type)`. Заменяет мнимую часть элемента по указанному индексу указанным значением.

Метод `setImag(int[], type)`. Заменяет мнимую часть элемента, определенного индексным массивом, на указанное значение.

В этих вызовах метода, строка **type** представляет один из следующих поддерживаемых типов Java `MWNumericArray`: `double`, `float`, `byte`, `short`, `int`, `long`, `Boolean`, подклассы `java.lang.Number`, подклассы `java.lang.String`, подклассы `java.lang.Boolean`. Метод, обозначенный выше как **getType**, может быть одним из следующих: `getDouble`, `getFloat`, `getLong`, `getInt`, `getShort`, `getByte`, `getData` и `toArray`.

Рассмотрим эти методы подробнее. Следующий синтаксис применяется ко всем указанным выше методам.

Синтаксис обращения. Чтобы получить элемент, указанный одним индексом или набором индексов, используется одна из следующих команд:

```
public type getType(int index)
public type getType(int[] index)
```

Синтаксис задания. Чтобы задать элемент по указанному одномерному или многомерному индексу, используется одна из следующих команд:

```
public void set(int index, type element)
public void set(int[] index, type element)
```

Первая команда (`int index`) используется для возвращения или задания элемента, указанного индексом одномерной индексацией MATLAB (в постолбцовом порядке). Второй случай (`int[] index`) используется для возвращения или задания элемента многомерного массива по указанному многомерному набору индексов массива. Первый синтаксис работает лучше, чем второй.

Приведем несколько примеров. В каждом из них используется массив `Adata`:

```
short[][] Adata = {{ 1,  2,  3,  4,  5,  6},
                   { 7,  8,  9, 10, 11, 12},
                   {13, 14, 15, 16, 17, 18}};
```

Пример. Получение значения `Short` от числового массива `Adata`.

```
MWNumericArray A = new MWNumericArray(Adata, MWClassID.INT16);
int[] index = {2, 4};
System.out.println("A(2,4) = " + A.getShort(index));
```

После выполнения пример отображает следующее:

```
A(2,4) = 10
```

Методы `get`, `set`, `toArray` и `getData` класса `MWNumericArray` наследуются от класса `MWArray`. Методы `get` и `set` обращаются к единственному элементу по указанному индексу. Индекс передают к этим методам в форме единственного номера или как массив индексов. Приведем примеры.

Пример. Получим и затем изменим значение элемента (2, 3) массива `Adata`:

```
int[] idx = {2, 3};
System.out.println("A(2, 3) is " + A.get(idx).toString());
System.out.println("");
System.out.println("Setting A(2, 3) to a new value ...");
A.set(idx, 555);
System.out.println("");
System.out.println("A(2, 3) is now " + A.get(idx).toString());
```

После выполнения пример отображает следующее:

```
A(2, 3) is 9.0
Setting A(2, 3) to a new value ...
A(2, 3) is now 555.0
```

Пример. Создание 2-на-2 матрицы используя метод `set`. Первый пример использует единственный индекс:

```
int[] dims = {2, 2};
MWNumericArray a =
    MWNumericArray.newInstance(dims, MWClassID.DOUBLE,
                               MWComplexity.REAL);
```

```

int index = 0; // Одномерный индекс (i)
double[] values = {1.0, 3.0, 2.0, 4.0}; // Массив значений

for (int index = 1; index <= 4; index++)
    a.set(index, values[index-1]); // Задание значений a(i)

Тот же самый пример, но на сей раз с использованием индексного массива:

int[] dims = {2, 2};
MWNumericArray a =
    MWNumericArray.newInstance(dims, MWClassID.DOUBLE,
        MWComplexity.REAL); // Матрица 2-на-2
int[] index = new int[2]; // Двумерный индекс (i,j)
int k = 0;
for (index[0] = 1; index[0] <= 2; index[0]++)
{
    for (index[1] = 1; index[1] <= 2; index[1]++)
        a.set(index, ++k); // Задание значений a(i,j)
}

```

Методы доступа и установки мнимых частей MWNumericArray. Символ **getImagType** может принимать следующие значения: **getImagDouble**, **getImagFloat**, **getImagLong**, **getImagInt**, **getImagShort**, **getImagByte**, а также **getImag**, **setImag**, **getImagData** и **toImagArray**. Ко всем перечисленным методам, кроме **getImagData** и **toImagArray**, применяется следующий синтаксис.

Синтаксис обращения. Для получения мнимой части элемента, указанного одним индексом или набором индексов, используется одна из следующих команд:

```

public type getImagType(int index)
public type getImagType(int[] index)

```

Синтаксис задания. Для установки мнимой части элемента по указанному одномерному или многомерному индексу, используется одна из следующих команд:

```

public void setImag(int index, type element)
public void setImag(int[] index, type element)

```

Первая команда (**int index**) используется для возвращения или задания мнимой части элемента указанного индексом одномерной индексацией MATLAB (в постолбцовом порядке). Второй случай (**int[] index**) используется для возвращения или задания мнимой части элемента многомерного массива по указанному многомерному набору индексов массива. Первый синтаксис работает лучше, чем второй. Отметим особенности двух следующих методов.

Метод getImag. Возвращает мнимую часть элемента MWNumericArray, указанного одномерным или многомерным индексом. Тип возвращаемого значения есть Object.

Метод setImag. Этот метод заменяет мнимую часть по указанному одномерному или многомерному индексу в массиве на указанное значение double:

```

public void setImag(int index, javatype element)
public void setImag(int[] index, javatype element)

```

Тип **javatype** может быть любым следующим: **double**, **float**, **long**, **int**, **short**, **byte**, **Object**.

Приведем несколько примеров. Будем использовать следующий комплексный массив A:

```
double[][] Rdata = {{ 2, 3, 4},
                    { 8, 9, 10},
                    {14, 15, 16}};
double[][] Idata = {{ 6, 5, 14},
                    { 7, 1, 23},
                    { 1, 1, 9}};
MWNumericArray A = new MWNumericArray(Rdata, Idata, MWClassID.DOUBLE);
System.out.println("Complex matrix A =");
System.out.println(A.toString());
```

Полученный комплексный массив:

```
2.0000 + 6.0000i   3.0000 + 5.0000i   4.0000 + 14.0000i
8.0000 + 7.0000i   9.0000 + 1.0000i   10.0000 + 23.0000i
14.0000 + 1.0000i  15.0000 + 1.0000i   16.0000 + 9.0000i
```

Пример. Используем `get` и `getImag` для чтения вещественной и мнимой частей элемента с индексами (2, 3):

```
int[] index = {2, 3};
System.out.println("The real part of A(2,3) = " +
                  A.get(index));
System.out.println("The imaginary part of A(2,3) = " +
                  A.getImag(index));
```

После выполнения пример отображает следующее:

```
The real part of A(2,3) = 10.0
The imaginary part of A(2,3) = 23.0
```

Пример. Получение комплексных данных определенного типа (`double`) элемента массива A.

```
int[] index = {2, 3};
System.out.println("The real part of A(2,3) = " +
                  A.getDouble(index));
System.out.println("The imaginary part of A(2,3) = " +
                  A.getImagDouble(index));
```

После выполнения примера получается следующий вывод

```
The real part of A(2,3) = 10.0
The imaginary part of A(2,3) = 23.0
```

Метод `getImagData`. Этот метод возвращает одномерный `MWNumericArray` содержащий копию мнимых частей данных в основном массиве MATLAB. Метод `getImagData` возвращает одномерный массив в постолбцовом порядке. Элементы преобразуются согласно правилам преобразования значений по умолчанию.

Пример. Получение всех вещественных и мнимых частей данных из комплексного массива A.

```
int[] index = {2, 3};
double[] x;
System.out.println("The real data in matrix A is:");
```

```
x = (double[]) A.getData();
for (int i = 0; i < x.length; i++)
    System.out.print(" " + x[i]);
    System.out.println();
System.out.println("The imaginary data in matrix A is:");
x = (double[]) A.getImagData();
for (int i = 0; i < x.length; i++)
    System.out.print(" " + x[i]);
```

После выполнения примера получается следующий вывод

```
The real data in matrix A is:
2.0 8.0 14.0 3.0 9.0 15.0 4.0 10.0 16.0
The imaginary data in matrix A is:
6.0 7.0 1.0 5.0 1.0 1.0 14.0 23.0 9.0
```

Метод toImagArray. Этот метод возвращает массив, содержащий копию мнимых частей данных массива MATLAB.

Пример. Получение массива мнимых частей комплексных данных используя toImagArray.

```
double[][] x = (double[][]) A.toImagArray();
int[] dimA = A.getDimensions();
System.out.println("The imaginary part of matrix A is:");
for (int i = 0; i < dimA[0]; i++)
{
    for (int j = 0; j < dimA[1]; j++)
        System.out.print(" " + x[i][j]);
        System.out.println();
}
```

После выполнения пример отображает следующее:

```
The imaginary part of matrix A is:
6.0 5.0 14.0
7.0 1.0 23.0
1.0 1.0 9.0
```

Методы копирования, преобразования и сравнения массивов MWNumericArray

Для копирования, преобразования и сравнения объектов класса MWNumericArray используются методы: **clone**, **compareTo**, **equals**, **hashCode**, **sharedCopy**, **toString**.

Метод clone. Этот метод создает и возвращает настоящую копию массива. Поскольку clone создает новый массив, любые изменения, сделанные в этом новом массиве не отражаются в оригинале. Прототип для метода clone следующий:

```
public Object clone()
```

Входных параметров нет.

Пример. Создание 3-на-6 массива типа double и преобразование его в массив типа MWNumericArray:

```
double[][] AData = {{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9, 10, 11, 12},
```

```
{13, 14, 15, 16, 17, 18}};
```

```
MWNumericArray A = new MWNumericArray(AData, MWClassID.DOUBLE);
```

Создание копии массива A класса MWNumericArray:

```
Object C = A.clone();
```

```
System.out.println("Clone of matrix A is:");
```

```
System.out.println(C.toString());
```

После выполнения пример отображает следующее:

```
Clone of matrix A is:
```

```
1    2    3    4    5    6
7    8    9   10   11   12
13   14   15   16   17   18
```

Метод compareTo. Этот метод MWNumericArray наследует от класса MWArray.

Метод equals. Этот метод MWNumericArray наследует от класса MWArray.

Метод hashCode. Этот метод MWNumericArray наследует от класса MWArray.

Метод sharedCopy. Этот метод создает и возвращает общедоступную копию объекта MWNumericArray. Общедоступная копия указывает на основной оригинальный массив MATLAB. Любые изменения, сделанные в копии отражаются в оригинале. Метод sharedCopy класса MWNumericArray подменяет метод sharedCopy класса MWArray. Прототип для метода sharedCopy следующий:

```
public Object sharedCopy()
```

Входных параметров нет.

Пример. Создание общедоступной копии числового массива A класса MWArray:

```
Object S = A.sharedCopy();
```

```
System.out.println("Shared copy of matrix A is:");
```

```
System.out.println(S.toString());
```

После выполнения примера получается следующий вывод

```
Shared copy of matrix A is:
```

```
1    2    3    4    5    6
7    8    9   10   11   12
13   14   15   16   17   18
```

Метод toString. Преобразование в строку. Этот метод MWNumericArray наследует от класса MWArray.

Методы возвращения значений специальных констант

Для получения значений констант EPS, Inf и NaN в MATLAB используются следующие методы:

- `getEps` – дает значение EPS (относительная точность с плавающей запятой) в MATLAB;
- `getInf` – представляет значение INF (бесконечность) в MATLAB;
- `getNaN` – представляет значение NaN (неопределенность) в MATLAB.

Метод `getEps`. Этот метод возвращает понятие константы MATLAB EPS, которая представляет относительную точность с плавающей запятой. Прототип для метода `getEps` следующий:

```
public static double getEps()
```

Метод `getInf`. Этот метод возвращает понятие константы MATLAB Inf, которая представляет бесконечность. Прототип для метода `getInf` следующий:

```
public static double getInf()
```

Метод `getNaN`. Этот метод возвращает понятие константы MATLAB NaN, которая представляет «неопределенность». Прототип для метода `getNaN` следующий:

```
public static double getNaN()
```

Методы `toTypeArray` и `getTypeArray` преобразования массивов данных

Для преобразования массива MATLAB в массив указанного примитивного типа данных, такого как `float` или `int`, используются следующие методы **`toTypeArray`**:

```
toByteArray,      toDoubleArray,    toFloatArray,    toIntArray,
toLongArray,      toShortArray,    toImagArray,     toImagByteArray,
toImagDoubleArray, toImagFloatArray, toImagIntArray,  toImagLongArray,
toImagShortArray.
```

Эти методы возвращают массив Java, соответствующий примитивному типу в имени метода. Возвращенный массив имеет ту же самую размерность как и основной массив MATLAB и указанный в методе тип. Например, если вызывается `toShortArray`, то возвращается массив типа `short`, независимо от типа данных в основном массиве. Поэтому при выполнении преобразования возможно усечение или другая потеря точности. Например, если вызывается `toFloatArray` на экземпляре класса `MWArray`, содержащего данные с двойной точностью, значения `double` усекаются до значений `float` – значений с одинарной точностью. Рекомендуемое соответствие типов указано в табл. 3.5.2.

Эти методы могут также быть полезными в определении типов в массиве Java, когда размерность вещественного или комплексного массива `MWArray` известна, но тип данных – нет. Для получения дополнительной информации можно обратиться к документации MATLAB, JavaDoc.

Пример. Следующий код показывает преобразование массива MATLAB в массив указанного примитивного типа тех же измерений.

```
Object results = null;
try {
    // вызов скомпилированной m-функции
    results = myobject.myfunction(2);
    // известно, что первый вывод является числовой матрицей
    MWArray resultA = (MWNumericArray) results[0];
```

```
double[][] a = resultA.toDoubleArray();  
// известно, что второй вывод является 3-мерным числовым массивом  
MArray resultB = (MNumericArray) results[1];  
Int[][][] b = resultB.toIntArray();  
}  
finally {  
    MArray.disposeArray(results);  
}
```

Для преобразования массива MATLAB в одномерный массив указанного примитивного типа данных используются следующие методы **getTypeArray**:

getByteData,	getDoubleData,	getFloatData,	getIntData,
getLongData,	getShortData,	getImagData,	getImagByteData,
getImagDoubleData,	getImagFloatData,	getImagIntData,	getImagLongData,
getImagShortData			

Примеры использования методов см. в следующих разделах.

Методы работы с разреженными массивами

MNumericArray

Операции на разреженных массивах типа `MNumericArray` в настоящее время поддерживаются только для типа `double`. Для получения информации относительно разреженных массивов типа `MNumericArray` используются следующие методы. Все они унаследованы от класса `MArray`.

- `newSparse` – создание вещественной разреженной числовой матрицы с указанным числом строк и столбцов и максимальным числом элементов отличных от нуля, инициализация массива поставляемыми данными;
- `isSparse` – проверка разреженности массива;
- `columnIndex` – возвращает массив, содержащий индексы столбцов каждого ненулевого элемента в основном массиве MATLAB;
- `rowIndex` – возвращает массив, содержащий индексы строк каждого ненулевого элемента в основном массиве MATLAB;
- `maximumNonZeros` – возвращает вместимость разреженного массива. Если основной массив неразрезан, этот метод возвращает то же значение, что и `numberOfElements()`;
- `numberOfNonZeros` – возвращает число ненулевых элементов в разреженном массиве. Если основной массив неразрезан, этот метод возвращает то же, что и `numberOfElements()`.

Эффективный способ создания разреженной матрицы состоит в использовании конструктора **`newSparse`**. Этот метод создает вещественный или комплексный разреженный массив `MNumericArray`, с указанным числом строк, столбцов и максимального числа отличных от нуля элементов и инициализирует массив представленными данными. Это – статический метод класса и таким образом не должен вызываться ссылкой на экземпляр класса.

Приведем примеры построения разреженных массивов.

Пример. Построение разреженной матрицы x со следующими значениями:

```
x = [ 2 -1 0 0
      -1 2 -1 0
        0 -1 2 -1
        0 0 -1 2 ]
```

При вызове `newSparse` передаются три массива: массив матричных данных (x), массив, содержащий индексы строк (`rowindex`) ненулевых элементов и массив индексов столбцов (`colindex`) ненулевых элементов. Число строк (4) и столбцов (4) также передают как тип (`MWClassID.DOUBLE`):

```
double[] x = { 2.0, -1.0, -1.0, 2.0, -1.0, /* Столбеобразное */
              -1.0, 2.0, -1.0, -1.0, 2.0 }; /* перечисление */
int[] rowindex = {1, 2, 1, 2, 3, 2, 3, 4, 3, 4};
int[] colindex = {1, 1, 2, 2, 2, 3, 3, 3, 4, 4};
MWNumericArray a =
    MWNumericArray.newSparse(rowindex, colindex, x, 4, 4,
    MWClassID.DOUBLE);
```

Пример. Построение массива без установки строк и столбцов. Можно передать только массивы строк и столбцов и тогда метод `newSparse` определяет число строк и столбцов разреженной матрицы из максимальных значений `rowindex` и `colindex`:

```
MWNumericArray a = MWNumericArray.newSparse(rowindex, colindex,
x, MWClassID.DOUBLE);
```

Пример. Построение массива из полной матрицы. Можно также создать разреженный массив из полной матрицы используя `newSparse`. Следующий пример переписывает предыдущий пример, используя полную матрицу:

```
double[][] x = {{ 2.0, -1.0, 0.0, 0.0},
                {-1.0, 2.0, -1.0, 0.0},
                { 0.0 -1.0, 2.0, -1.0},
                { 0.0, 0.0, -1.0, 2.0 }};
MWNumericArray a = MWNumericArray.newSparse(x, MWClassID.DOUBLE);
```

Пример. Создание комплексного двумерного разреженного `MWNumericArray` из вещественного и мнимого векторов `double`:

```
double[][] rData = {{ 0, 0, 0, 16, 0}, {71, 63, 32, 0, 0}};
double[][] iData = {{ 0, 0, 0, 41, 0}, { 0, 0, 32, 0, 2}};
MWNumericArray A =
    MWNumericArray.newSparse(rData, iData, MWClassID.DOUBLE);
System.out.println("A = " + A.toString());
```

После выполнения пример отображает следующее:

```
A = (2,1)    71.0000
     (2,2)    63.0000
     (2,3)    32.0000 +32.0000i
     (1,4)    16.0000 +41.0000i
     (2,5)           0 + 2.0000i
```

3.3.3. Работа с логическими, символьными и массивами ячеек

Рассмотрим кратко логические, символьные и массивы ячеек. Работа с ними аналогична работе с числовыми массивами. Более подробную информацию можно найти в документации MATLAB Java Builder.

Логические массивы. Класс `MWLogicalArray` обеспечивает доступ Java к логическому массиву MATLAB. Массив `MWLogicalArrays` может быть плотным или разреженным. Класс `MWLogicalArray` имеет ряд конструкторов и методов для создания логических массивов. Конструкторы:

- `MWLogicalArray()` – пустой логический массив;
- `MWLogicalArray(type)` – массив `Logical` со значениями, инициализированными поставляемыми данными

Здесь **type** представляет поддерживаемые типы Java. Класс `MWLogicalArray` поддерживает следующие примитивные типы Java: `double`, `float`, `byte`, `short`, `int`, `long` и `boolean`, а также объекты подклассов `java.lang.Number`, `java.lang.String` и `java.lang.Boolean`. Поддерживаются общие N-мерные массивы каждого типа.

Когда используются числовые типы, значения в логическом массиве устанавливаются как `true`, если входное значение отлично от нуля, и `false` иначе. Следующие примеры создают скалярный логический массив со значением, инициализированным как `true`:

```
MWLogicalArray a1 = new MWLogicalArray(true);
MWLogicalArray a2 = new MWLogicalArray(1);
MWLogicalArray a3 = new MWLogicalArray("true");
MWLogicalArray a4 = new MWLogicalArray(new Boolean(true));
```

Следующие примеры создают скалярный логический массив, инициализированный как `false`:

```
MWLogicalArray a1 = new MWLogicalArray(false);
MWLogicalArray a2 = new MWLogicalArray(0);
MWLogicalArray a3 = new MWLogicalArray("false");
MWLogicalArray a4 = new MWLogicalArray(new Boolean(false));
```

Для создания массивов `MWLogicalArray` используются следующие статические методы: `newInstance(int[])`, `newInstance(int[], Object)`, `newSparse(int[], int[], Object, int, int, int)`, `newSparse(int[], int[], Object, int, int)`, `newSparse(int[], int[], Object)` и `newSparse(Object)`.

Следующий пример показывает использование конструктора `newInstance` двумерного массива:

```
boolean[] x1 = {true, false, false, true};
int[] x2 = {1, 0, 0, 1};
Boolean[] x3 = {new Boolean(true), new Boolean(false),
                new Boolean(false), new Boolean(true)};
String[] x4 = {"true", "false", "false", "true"};

int[] dims = {2, 2};
```

```
MWLogicalArray a1 = MWLogicalArray.newInstance(dims, x1);
MWLogicalArray a2 = MWLogicalArray.newInstance(dims, x2);
MWLogicalArray a3 = MWLogicalArray.newInstance(dims, x3);
MWLogicalArray a4 = MWLogicalArray.newInstance(dims, x4);
```

Класс `MWLogicalArray` имеет методы для того, чтобы обратиться и изменять данные массива в форме методов **get** и **set**.

Символьные массивы. Класс `MWCharArray` обеспечивает интерфейс Java к массиву `char` MATLAB. Класс `MWCharArray` имеет ряд конструкторов и методов для создания символьных массивов. Конструкторы:

- `MWCharArray()` – пустой массив `char`;
- `MWCharArray(type)` – массив `char` со значениями, инициализированными представленными данными.

Здесь, **type** представляет поддерживаемые типы Java.

Класс `MWCharArray` поддерживает следующие типы Java: `char`, `java.lang.Character` и `java.lang.String`. В дополнение к поддержке скалярных (т.е. 1-на-1) значений перечисленных типов, также поддерживаются общие N-мерные массивы каждого типа.

Следующие примеры создают скалярные массивы `char`:

```
MWCharArray a1 = new MWCharArray('a');
MWCharArray a2 = new MWCharArray(new Character('a'));
```

Построение строк. Вы можете использовать класс `MWCharArray`, чтобы создать строки символов, как показано в этих примерах:

```
char[] x1 = {'A', ' ', 'S', 't', 'r', 'i', 'n', 'g'};
String x2 = "A String";
Character[] x3 = {
    new Character('A'),
    new Character(' '),
    new Character('S'),
    new Character('t'),
    new Character('r'),
    new Character('i'),
    new Character('n'),
    new Character('g')};
MWCharArray a1 = new MWCharArray(x1);
MWCharArray a2 = new MWCharArray(x2);
MWCharArray a3 = new MWCharArray(x3);
```

Для создания массивов `MWCharArray` используются также следующие методы: `newInstance(int[])` и `newInstance(int[] Object)`. Входной массив данных должен быть либо одномерным массивом `char`, либо одномерным массивом `java.lang.Character`, или простым `java.lang.String`.

В приведенном примере можно было бы использовать метод **newInstance**:

```
int[] dims = {1, 8};
MWCharArray a1 = MWCharArray.newInstance(dims, x1);
MWCharArray a2 = MWCharArray.newInstance(dims, x2);
MWCharArray a3 = MWCharArray.newInstance(dims, x3);
```

Класс `MWCharArray` обеспечивает методы доступа к элементам массива `MWCharArray` в форме методов **get** и **set**.

Массивы ячеек. Класс `MWCellArray` обеспечивает интерфейс Java к массиву ячеек MATLAB. Класс `MWCellArray` обеспечивает следующие конструкторы:

- `MWCellArray()` – пустой массив ячеек;
- `MWCellArray(int[])` – новый массив ячеек с указанными измерениями. Все ячейки инициализированы как пустые;
- `MWCellArray(gint, int)` – новая матрица ячеек с указанными числом строк и столбцов.

Построение массива ячеек делается в два шага. Сначала, определяется массив ячеек, использующий один из конструкторов в предыдущей таблице, затем назначаются значения в каждую ячейку, используя один из методов **set**.

Построение `MWCellArray`. Для простых массивов самым удобным подходом является непосредственная передача массива Java. Когда Вы хотите назначить более сложный тип в ячейку (т.е., комплексный массив или другой массив ячеек), Вы должны создать временный `MWArray` для входного значения. После назначения их в ячейку нужно избавиться от любых временных массивов.

Следующий пример создает и инициализирует 2-на-3 массив ячеек `MWCellArray`:

```
int[] cdims = {2, 3};
MWCellArray C = new MWCellArray(cdims);

Integer[] val = new Integer[6];
for (int i = 0; i < 6; i++)
    val[i] = new Integer(i * 15);

for (int i = 0; i < 2; i++)
    for (int j = 0; j < 3; j++)
    {
        int[] idx = {i+1, j+1};
        C.set(idx, val[j + (i * 3)]);
    }

System.out.println(«C = « + C.toString());
```

После выполнения получаем на дисплее следующий вывод:

```
C =      [ 0]      [15]      [30]
      [45]      [60]      [75]
```

Класс `MWCellArray` обеспечивает методы доступа к элементам массива `MWCellArray` в форме методов **get** и **set**.

3.3.4. Использование `MWClassID`

Класс `MWClassID` перечисляет все типы массивов MATLAB. Используется для определения типа массива MATLAB. Этот класс не содержит никаких конструкторов. Обеспечивается набор экземпляров `public static MWClassID`, один для каждого типа массива MATLAB. Класс `MWClassID` расширяет класс `java.lang.Object`.

Поля *MWClassID*

- CELL – представляет тип массива ячеек MATLAB;
- CHAR – представляет тип char массива MATLAB;
- DOUBLE – представляет тип double массива MATLAB;
- INT8 – представляет тип int8 массива MATLAB;
- INT16 – представляет тип int16 массива MATLAB;
- INT32 – представляет тип int32 массива MATLAB;
- INT64 – представляет тип int64 массива MATLAB;
- LOGICAL – представляет тип logical массива MATLAB;
- OPAQUE – представляет тип opaque массива MATLAB;
- SINGLE – представляет тип single массива MATLAB;
- STRUCT – представляет тип массива MATLAB struct.
- UINT8 – представляет тип uint8 массива MATLAB;
- UINT16 – представляет тип uint16 массива MATLAB;
- UINT32 – представляет тип uint32 массива MATLAB;
- UINT64 – представляет тип uint64 массива MATLAB;
- UNKNOWN – представляет тип empty массива MATLAB.

Пример. Определение значения *MWClassID*. Создание скалярного числового массива типа *MWClassID.INT16*:

```
double AReal = 24;  
MWNumericArray A = new MWNumericArray(AReal, MWClassID.INT16);  
System.out.println("Array A of type " + A.classID() + " = \n" + A);
```

После выполнения примера, результаты следующие:

```
Array A of type int16 =  
24
```

Методы класса *MWClassID*

Метод *equals*. Этот метод указывает, является ли некоторый другой класс *MWClassID* равным данному. Метод *equals* класса *MWClassID* подменяет метод *equals* класса *java.lang.Object*.

Метод *getSize*. Этот метод возвращает размер в байтах элемента массива этого типа.

Метод *hashCode*. Этот метод возвращает значение хеш-кода для этого типа. Он подменяет метод *hashCode* класса *java.lang.Object*.

Метод *isNumeric*. Этот метод проверяет, если этот тип является числовым.

Метод *toString*. Этот метод возвращает строковое представление свойства. Метод *toString* класса *MWClassID* подменяет метод *toString* класса *java.lang.Object*.

Метод *toString*. Этот метод возвращает строковое представление. Он подменяет метод *toString* класса *java.lang.Object*.

3.3.5. Использование класса *MWComplexity*

Класс *MWComplexity* класс учитывает свойство вещественности/комплексности массива MATLAB. Этот класс не содержит никаких конструкторов. Обеспечива-

ется набор экземпляров `public static MWComplexity`, один – для представления `real` и один – для `complex`. `MWComplexity` расширяет класс `java.lang.Object`. Поля класса `MWComplexity`:

- `REAL` – представляет вещественное числовое значение. Прототип для `REAL` следующий:

```
public static final MWComplexity REAL
```

- `COMPLEX` – представляет комплексное числовое значение, содержащее и реальную и мнимую части. Прототип для `COMPLEX` следующий:

```
public static final MWComplexity COMPLEX
```

Пример. Определение комплексности массива, является ли матрица `A` вещественной или комплексной. Метод `complexity` класса `MWNumericArray` возвращает описание типа `MWComplexity`.

```
double AReal = 24;
double AImag = 5;
MWNumericArray A = new MWNumericArray(AReal, AImag);
System.out.println("A is a " + A.complexity() + " matrix");
```

После выполнения пример отображает следующий вывод:

```
A is a complex matrix
```

Метод `toString` класса `MWComplexity` возвращает строковое представление. Метод `toString` класса `MWComplexity` подменяет метод `toString` класса `java.lang.Object`.

3.4. Примеры приложений Java

В данном параграфе рассмотрим еще два учебных примера из MATLAB. В первом примере создается приложение, которое делает быстрое преобразование Фурье исходного сигнала и изображает графики самого сигнала и частотного спектра сигнала в графическом окне MATLAB. Второй пример демонстрирует использование матричных функций MATLAB в приложении Java. Учебные примеры Java Builder находятся в каталоге `matlabroot\toolbox\javabuilder\Examples`.

3.4.1. Пример спектрального анализа

Данный пример показывает, как использовать Java Builder для создания компонента (`spectralanalysis`) содержащего класс с двумя методами, как обратиться к компоненту в приложении Java (`powerspect.java`), как создать и выполнить приложение.

Построение компонента

Компонент `spectralanalysis` будет иметь один класс с двумя методами, `computefft` и `plotfft`. Метод `computefft` вычисляет быстрое преобразование Фурье ($y = \text{fft}(x)$), плотность спектра входных данных и вычисляет вектор точек частоты, основанных на длине введенных данных и шаге выборки. Метод `plotfft` выполняет те же

самые операции как `computefft`, но строит еще и графики входных данных и спектра в обычном графическом окне MATLAB. Код MATLAB для этих двух методов находится в двух m-файлах, `computefft.m` и `plotfft.m` (которые можно найти в подкаталоге `\Examples\SpectraExample\SpectraDemoComp\`).

Напомним, что дискретным *преобразованием Фурье* сигнала $\{x_n\}$ конечной длины N называется сигнал $\{y_n\}$, полученный по формуле:

$$y_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi}{N} (k-1)(n-1)}, \quad k=1, 2, \dots, N.$$

Здесь предполагается, что сигнал $\{x_n\}$ получен оцифровкой функции $f(t)$ по формуле $x_n = f((n-1)\Delta t)$, $n = 1, 2, \dots, N$. Параметр t меняется на промежутке $[0, N-1]$, Δt – шаг дискретизации. Величина $F = 1/\Delta t$ называется частотой дискретизации – число отсчетов в единицу времени. Тогда частоты сигнала могут принимать значения от 0 до F . Поэтому массив частот сигнала $\{x_n\}$ вычисляются по формуле $\omega_k = Fk/N = k/(N\Delta t)$, $k = 0, 1, \dots, N-1$. Еще одной характеристикой сигнала является спектр (или мощность спектра), который вычисляется по формуле

$$P_k = \frac{|y_k|}{\sqrt{N}}, \quad k=1, 2, \dots, N.$$

Приведем код функции `computefft.m`, которая имеет входные аргументы:

- `data` – входной сигнал $\{x_n\}$;
- `interval` – шаг дискретизации Δt

и выходные параметры:

- `fftdata` – выходной сигнал $\{y_n\}$;
- `freq` – массив частот;
- `powerspect` – мощность спектра.

Листинг m-функции `computefft.m`.

```
function [fftdata, freq, powerspect] = computefft(data, interval)
%COMPUTEFFT Computes the FFT and power spectral density.
if (isempty(data))
    fftdata = [];
    freq = [];
    powerspect = [];
    return;
end
if (interval <= 0)
    error('Sampling interval must be greater then zero');
    return;
end
fftData = fft(data);
freq = (0:length(fftData)-1)/(length(fftData)*interval);
powerSpect = abs(fftData)/(sqrt(length(fftData)));
```

Функция `plotfft.m` обращается к функции `computefft` для спектрального разложения и выводит графики исходного сигнала $\{x_n\}$ и частотного спектра P_k – до частоты Найквиста $F/2$. Код функции `plotfft.m`:

```
function [fftdata, freq, powerspect] = plotfft(data, interval)
% Обращается к функции computefft для спектрального
% разложения и строит графики данных
[fftdata, freq, powerspect] = computefft(data, interval);
len = length(fftdata);
if (len <= 0)
    return;
end
t = 0:interval:(len-1)*interval;
subplot(2,1,1), plot(t, data)
xlabel('Time'), grid on
title('Time domain signal')
subplot(2,1,2), plot(freq(1:len/2), powerspect(1:len/2))
xlabel('Frequency (Hz)'), grid on
title('Power spectral density')
```

1. Подготовка к созданию проекта. Выберем для проекта следующий каталог: D:\javabuilder_examples\SpectraExample. Затем образуем m-функции, из которых будут создаваться класс и методы Java. Поскольку мы рассматриваем уже готовый учебный пример MATLAB с функциями computefft.m и plotfft.m, то просто скопируем следующий подкаталог примеров MATLAB: .matlabroot\toolbox\javabuilder\Examples\SpectraExample в каталог D:\javabuilder_examples. После копирования в рабочем каталоге проекта будут еще два подкаталога:

- SpectraDemoComp – он содержит m-функции computefft.m и plotfft.m;
- SpectraDemoJavaApp – содержит код Java powerspect.java, который будет использоваться для создания приложения.

Устанавливаем в качестве текущего каталога MATLAB новый подкаталог проекта D:\javabuilder_examples\SpectraExample.

Перед началом работы нужно также установить переменную среды JAVA_HOME, как было описано в разделе 3.2.1. Для этого в командной строке DOS (находясь в каталоге проекта) нужно исполнить, одну из следующих команд (записываем данные команды в файл Java_Home.bat и исполняем его):

```
set JAVA_HOME=C:\Borland\JBuilder2006\jdk1.5
PATH C:\Borland\JBuilder2006\jdk1.5\;
C:\Borland\JBuilder2006\jdk1.5\bin;
C:\Borland\JBuilder2006\jdk1.5\jre;
```

2. Создание нового проекта. Будем использовать графический интерфейс разработки. Он запускается из MATLAB следующей командой:

```
deploytool
```

Создадим новый проект со следующими параметрами настройки: имя компонента – **spectralanalysis**; имя класса – **fourier**; подробный вывод. Добавим к проекту один m-файл plotfft.m. Отметим, что в этом примере, приложение, которое использует класс fourier не должно вызывать computefft непосредственно. Метод computefft требуется только для метода plotfft. Таким образом, при создании компонента не обязательно добавлять в проект файл computefft.m, хотя это и не причиняет вреда (рис. 3.4.1). Сохраним проект.

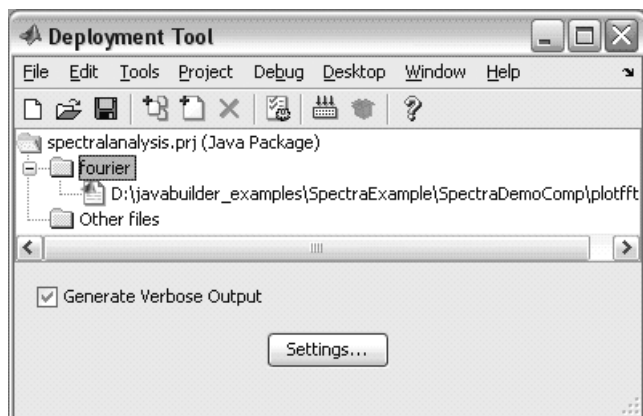


Рис. 3.4.1. Проект spectralanalysis

3. Построение пакета Java. Для построения компонента, инкапсулирующего функции MATLAB `computefft.m` и `plotfft.m`, нужно нажать кнопку построения (**Build**) на инструментальной панели. Начинается процесс построения и создается log-файл регистрации процесса построения, в котором записываются все операции и ошибки при построении компонента. В случае успешного исхода в каталоге проекта создается подкаталог **spectralanalysis**, который содержит два подкаталога **distrib** и **src**, в которые помещаются созданные файлы компонента.

Подкаталог **distrib** содержит файлы `spectralanalysis.ctf` и `spectralanalysis.jar`, которые предназначены для создания инсталляционного пакета для распространения. Напомним, что технологический файл компоненты `spectralanalysis.ctf` содержит зашифрованные m-функции, которые составляют компонент и все другие m-функции MATLAB, от которых зависят основные m-функции. Архив содержит все, основанное на MATLAB содержание (m-файлы, MEX-файлы, и т.д.), связанное с компонентом. Файл `spectralanalysis.jar` есть архив, содержащий созданные байт-коды классов пакета Java.

Подкаталог **src** содержит копии файлов `spectralanalysis.ctf` и `spectralanalysis.jar`, log-файлы регистрации процесса построения, а также созданный файл `readme.txt`, который содержит полезную информацию для распространения пакета. Кроме того, в подкаталоге `classes` содержатся созданные классы компонента, а в подкаталоге `spectralanalysis` содержатся Java интерфейсные файлы.

Разработка приложения, использующего компонент

Рассмотрим приложение, которое обращается к данному компоненту. Основные шаги по разработке приложения Java, использующего функции данного компонента:

- написание кода приложения;
- установка переменных среды, которые требуются на машине развития;
- построение и тестирование приложения Java, как обычного приложения.

Рассмотрим эти этапы на примере создания приложения, использующего класс компонента `spectralanalysis`.

1. Создание кода приложения Java. Пример кода `powerspect.java` приложения для этого примера находится в подкаталоге `SpectraExample\SpectraDemoJavaApp\`. Приведем листинг программы `powerspect.java`.

```
/* Импорт необходимых пакетов */
import com.mathworks.toolbox.javabuilder.*;
import spectralanalysis.*;
/*
 * Определяем класс powerspect, который вычисляет и изображает
 * мощность спектра входного сигнала
 */
class powerspect
{
    public static void main(String[] args)
    {
        double interval = 0.01;      /* Шаг выборки */
        int nSamples = 1001;         /* Число отсчетов */
        MWNumericArray data = null;  /* Входные данные */
        Object[] result = null;      /* Для результатов */
        fourier theFourier = null;   /* Экземпляр класса Fourier */

try
    {
        /*
         * Построение входных данных как sin(2*PI*15*t) +
         * sin(2*PI*40*t) + случайный сигнал
         * продолжительность времени = 10
         * шаг дискретизации, interval = 0.01
         */
        int[] dims = {1, nSamples};
        data = MWNumericArray.newInstance(dims, MWClassID.DOUBLE,
                                          MWComplexity.REAL);

        for (int i = 1; i <= nSamples; i++)
        {
            double t = (i-1)*interval;
            double x = Math.sin(2.0*Math.PI*15.0*t) +
                Math.sin(2.0*Math.PI*40.0*t) +
                Math.random();
            data.set(i, x);
        }

        /* Создание нового объекта fourier */
        theFourier = new fourier();

        /* Вычисление мощности спектра и изображение результатов на графике */
        result = theFourier.plotfft(3, data, new Double(interval));
    }
catch (Exception e)
    {
        System.out.println("Exception: " + e.toString());
    }
finally
```

```

    {
/* Освобождение ресурсов */
        MWArray.disposeArray(data);
        MWArray.disposeArray(result);
        if (theFourier != null)
            theFourier.dispose();
    }
}
}

```

Программа делает следующее:

- создает массив data типа MWNumericArray размеров dims, который содержит данные;
- создает входной массив значений, которые складываются из двух синусоид $\sin(2\pi 15t) + \sin(2\pi 40t)$ с частотами колебаний 15 и 40 Гц и случайного сигнала;
- создает экземпляр объекта fourier;
- вызывает метод plotfft, который вызывает computefft и строит графики данных;
- использует блок try/catch, чтобы обработать исключения;
- освобождает свои ресурсы использованные методами MWArray.

2. Установка переменных среды и компиляция приложения. Скомпилируем приложение powerspect.java используя программу javac. Сначала необходимо установить переменные JAVA_HOME, CLASSPATH и пути, как указано в первом параграфе. Считаем, что JAVA_HOME и пути установлены в предыдущем разделе. Для установки переменной CLASSPATH достаточно исполнить следующую команду в строке DOS:

```

set CLASSPATH=
.;.\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;
D:\javabuilder_examples\SpectraExample\spectralanalysis\distrib\
spectralanalysis.jar

```

Другой способ заключается в том, чтобы указать пути класса в командной строке Java при создании класса из кода powerspect.java следующим образом. Перейти в каталог D:\javabuilder_examples\SpectraExample и исполнить в нем следующую команду DOS (одна строка):

```

javac -classpath
;matlabroot\toolbox\javabuilder\jar\javabuilder.jar;.\spectralanalysis\distrib\
spectralanalysis.jar .\SpectraDemoJavaApp\powerspect.java

```

Эту команду удобно записать в файле Class.bat и исполнить его. При вводе команды имейте ввиду, что не должно быть пробелов между именами пути в параметре matlabroot. Например, не должно быть пробела между javabuilder.jar; и .\spectralanalysis\distrib\spectralanalysis.jar.

Компиляция производит файл powerspect.class и помещает его в подкаталог SpectraDemoJavaApp. Напомним, что если компонент создан с использованием msc, то Java Builder не создает каталог distrib, поэтому пути должны быть скорректированы.

При использовании компонента на другой машине (без MATLAB) вместо подкаталога `matlabroot/toolbox/javabuilder/jar` нужно указывать подкаталог MCR: `MCRroot/toolbox/javabuilder/jar`. Также должен быть путь `<mcr_root>\<ver>\runtime\win32` к библиотекам MCR.

3. Запуск приложения. Для запуска приложения `powerspect` нужно выполнить следующее (см. файл `Run_M.bat`):

- поместить файл `powerspect.class` в каталог запуска приложения `D:\javabuilder_examples\SpectraExample`;
- определить пути для библиотек MATLAB (или для MCR)

```
PATH <matlabroot>\bin\win32;
```

или

```
PATH <mcr_root>\v76\runtime\win32;
```

- выполнить следующую команду в строке DOS (одна строка, нужно указывать полный путь к команде `java`):

```
matlabroot\sys\java\jre\win32\jre1.5.0_07\bin\java -classpath  
.;matlabroot\toolbox\javabuilder\jar\javabuilder.jar.\spectralanalysis\  
distrib\spectralanalysis.jar powerspect
```

Эту команду удобно записать в файле `Run.bat` и исполнить его. Программа `powerspect` должна отобразить вывод в окне **Figure** MATLAB (рис. 3.4.2).

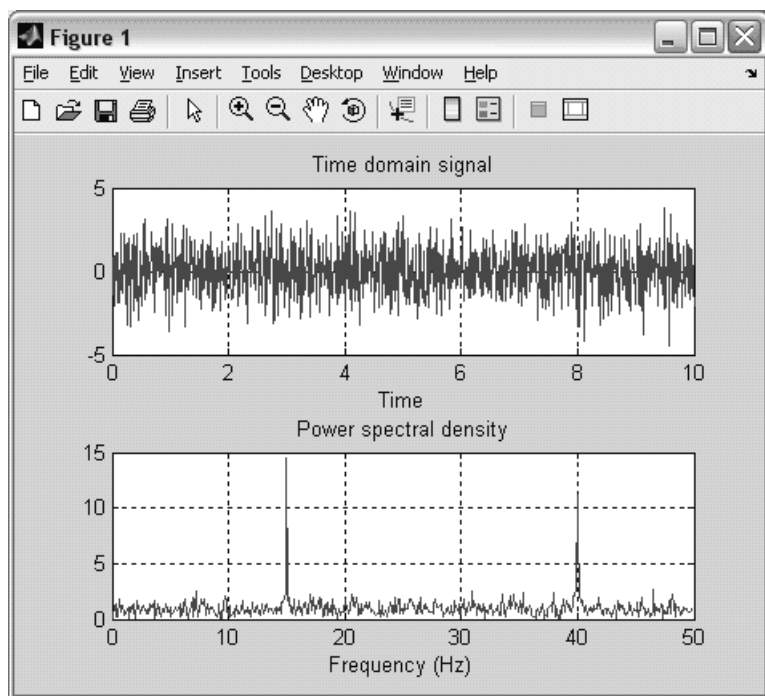


Рис. 3.4.2. Результаты работы программы `powerspect`

3.4.2. Пример матричной математики

На этом примере показывается, как использовать несколько m-функций MATLAB для класса компонента, как вручную управлять памятью и как обратиться к компоненту в приложении Java.

В примере строится компонент Java для выполнения некоторых математических операций над матрицами. Пример создает программу, которая выполняет разложения на множители Холецкого, LU и QR разложения для простой тридиагональной матрицы (матрица конечных разностей) следующего вида:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Напомним, что известно несколько способов разложения матрицы в произведение более простых матриц с целью, быстрее и проще решать системы линейных уравнений при большом числе уравнений.

LU -разложение на множители выражает любую квадратную матрицу как произведение,

$$A = LU,$$

где L – нижняя треугольная матрица с единицами на главной диагонали и, возможно, с переставленными затем строками, U – верхняя треугольная матрица. Матрица U получается в результате преобразования матрицы A по методу Гаусса. Использование такого разложения сводит решение системы уравнений к решению двух систем с треугольными матрицами.

Разложение на множители Холецкого выражает симметрическую положительно определенную матрицу в виде произведения верхней треугольной матрицы R и ее транспонированной:

$$A = R^t \cdot R.$$

Ортогональное, или QR -разложение на множители выражает любую прямоугольную матрицу как произведение ортогональной или унитарной матрицы и верхней треугольной матрицы.

$$A = QR,$$

где Q – ортогональная или унитарная, R – верхняя треугольная матрица. Матрица Q получается при ортогонализации столбцов матрицы методом Грама-Шмидта.

Эти матричные разложения производятся встроенными функциями MATLAB: $L = \text{chol}(A)$; $[L, U] = \text{lu}(A)$; $[Q, R] = \text{qr}(A)$. Учитывая, что Компилятор MATLAB не обрабатывает встроенные функции, нужно будет написать простые m-функции, которые вызывают указанные выше встроенные функции разложения.

Поскольку тип матрицы A для разложения уже выбран, достаточно будет указать размер матрицы в командной строке при вызове приложения. Тогда программа создает матрицу и выполняет эти три разложения на множители. Исходная

матрица A и результаты разложения будут направлены на стандартный вывод. Вторым параметром в командной строке может быть «sparse», что указывает на то, что используются разреженные матрицы.

Построение компонента

Создадим следующие m-функции MATLAB, из которых будут созданы методы создаваемого компонента:

Файл `cholesky.m`:

```
function [L] = cholesky(A)
%CHOLSKY Cholesky factorization of A.
L = chol(A);
```

Файл `ludcomp.m`:

```
function [L,U] = ludcomp(A)
%LUDECOMP LU factorization of A.
[L,U] = lu(A);
```

Файл `qrdecomp.m`:

```
function [Q,R] = qrdecomp(A)
%QRDECOMP QR factorization of A.
[Q,R] = qr(A);
```

1. Подготовка к созданию проекта. Выберем для проекта следующий каталог: `D:\javabuilder_examples\MatrixMathExample`. М-функции, из которых будут создаваться класс и методы Java заданы выше. Но поскольку мы рассматриваем уже готовый учебный пример MATLAB с функциями `cholesky.m`, `ludcomp.m` и `qrdecomp.m`, то просто скопируем следующий подкаталог примеров MATLAB: `.matlabroot\toolbox\javabuilder\Examples\MatrixMathExample` в наш рабочий каталог `D:\javabuilder_examples`. После копирования в рабочем каталоге проекта будут еще два подкаталога:

- `MatrixMathDemoComp` – он содержит m-функции `cholesky.m`, `ludcomp.m` и `qrdecomp.m`;
- `MatrixMathDemoJavaApp` – содержит код Java `getfactor.java`, который будет использоваться для создания приложения.

Устанавливаем в качестве текущего каталога MATLAB новый подкаталог проекта `D:\javabuilder_examples\MatrixMathExample`.

Перед началом работы нужно также установить переменную среды `JAVA_HOME`, как было описано в разделе 3.2.1. Для этого в командной строке DOS (находясь в каталоге проекта) нужно исполнить, одну из следующих команд:

```
set JAVA_HOME=C:\Borland\JBuilder2006\jdk1.5
PATH C:\Borland\JBuilder2006\jdk1.5\;
C:\Borland\JBuilder2006\jdk1.5\bin;
C:\Borland\JBuilder2006\jdk1.5\jre;
```

Для этого записываем данные команды в файл `Java_Home.bat` и исполняем его.

2. Создание нового проекта. Будем использовать графический интерфейс разработки. Он запускается из MATLAB следующей командой:

```
deploytool
```


Создадим новый проект со следующими параметрами настройки: имя компонента – **factormatrix**; имя класса – **factor**; подробный вывод. Добавим к проекту три файла `cholesky.m`, `ludcomp.m` и `qrdecomp.m` (рис. 3.4.3). Сохраним проект.

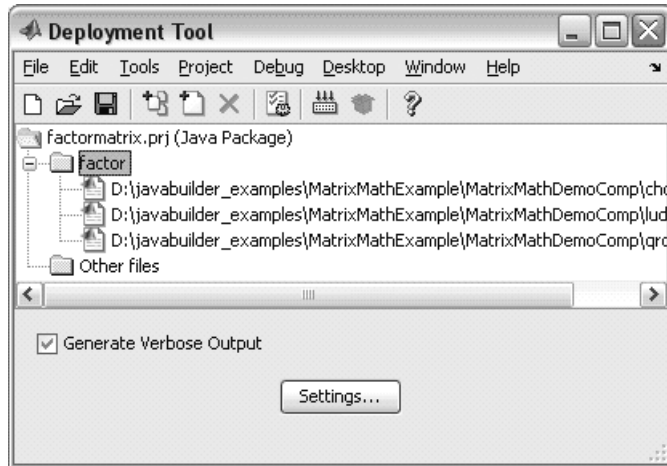


Рис. 3.4.3. Проект *factormatrix*

3. Построение пакета Java. Для построения компонента, инкапсулирующего функции MATLAB `cholesky.m`, `ludcomp.m` и `qrdecomp.m`, нужно нажать кнопку построения (**Build**) на инструментальной панели. В случае успешного исхода в каталоге проекта создается подкаталог **factormatrix**, который содержит два подкаталога **distrib** и **src**, в которые помещаются созданные файлы компонента.

Подкаталог **distrib** содержит файлы `factormatrix.ctf` и `factormatrix.jar`, которые предназначены для создания инсталляционного пакета для распространения.

Подкаталог **src** содержит копии файлов `factormatrix.ctf` и `factormatrix.jar`, log-файлы регистрации процесса построения, а также файл `readme.txt`. Кроме того, в подкаталоге **classes** содержатся созданные классы компонента, а в подкаталоге **factormatrix** содержатся Java интерфейсные файлы.

Разработка приложения, использующего компонент

Создадим приложение, которое обращается к данному компоненту **factormatrix**. Для этого необходимо написать java-код приложения, установить переменные среды, построить и протестировать приложение Java.

1. Создание кода приложения Java. Пример кода `getfactor.java` приложения для этого примера находится в подкаталоге `\MatrixMathExample\MatrixMathDemoJavaApp`. Приведем листинг программы `getfactor.java`.

```
/* Импорт необходимых пакетов */
import com.mathworks.toolbox.javabuilder.*;
```

```
import factormatrix.*;

/*
 * Создание класса getfactor, который вычисляет разложения
 * Холецкого LU и QR, матрицы конечных разностей порядка n.
 * Значение n передается в командной строке
 * Если второй аргумент командной строки передается как «sparse»,
 * то используется разреженная матрица.
 */
class getfactor
{
    public static void main(String[] args)
    {
        MWNumericArray a = null; /* Для матрицы для разложения */
        Object[] result = null; /* Для результата, тип Object[] */
        factor theFactor = null; /* Для экземпляра класса factor */
try
    {
        /* Если нет входных данных, выход */
        if (args.length == 0)
        {
            System.out.println("Error: must input a positive integer");
            return;
        }
        /* Преобразование входного значения из строки в int */
        int n = Integer.valueOf(args[0]).intValue();

        if (n <= 0)
        {
            System.out.println("Error: must input a positive integer");
            return;
        }
    }
    /*
     * Задание типа матрицы.
     * Если второй ввод "sparse", то – разреженная матрица
     */
    int[] dims = {n, n};

    if (args.length > 1 && args[1].equals("sparse"))
        a = MWNumericArray.newSparse(dims[0], dims[1], n+2*(n-1),
                                     MWClassID.DOUBLE, MWComplexity.REAL);
    else
        a = MWNumericArray.newInstance(dims, MWClassID.DOUBLE,
                                       MWComplexity.REAL);

    /* Установка значений элементов матрицы */
    int[] index = {1, 1};

    for (index[0] = 1; index[0] <= dims[0]; index[0]++)
    {
```

```

        for (index[1] = 1; index[1] <= dims[1]; index[1]++)
        {
            if (index[1] == index[0])
                a.set(index, 2.0);
            else if (index[1] == index[0]+1 || index[1] == index[0]-1)
                a.set(index, -1.0);
        }
    }

    /* Создание нового объект factor */
    theFactor = new factor();

    /* Печать исходной матрицы */
    System.out.println("Original matrix:");
    System.out.println(a);

    /* Вычисление разложения Холецкого и печать результатов */
    result = theFactor.cholesky(1, a);
    System.out.println("Cholesky factorization:");
    System.out.println(result[0]);
    MWArray.disposeArray(result);

    /* Вычисление разложения LU и печать результатов */
    result = theFactor.ludecomp(2, a);
    System.out.println("LU factorization:");
    System.out.println("L matrix:");
    System.out.println(result[0]);
    System.out.println("U matrix:");
    System.out.println(result[1]);
    MWArray.disposeArray(result);

    /* Вычисление разложения QR и печать результатов */
    result = theFactor.qrdecomp(2, a);
    System.out.println("QR factorization:");
    System.out.println("Q matrix:");
    System.out.println(result[0]);
    System.out.println("R matrix:");
    System.out.println(result[1]);
    }
    catch (Exception e)
    {
        System.out.println("Exception: " + e.toString());
    }

    finally
    {
        /* Освобождение собственных ресурсов */
        MWArray.disposeArray(a);
        MWArray.disposeArray(result);
        if (theFactor != null)

```

```

        theFactor.dispose();
    }
}
}

```

Обсуждение программы getfactor. Программа состоит из трех частей. Первая часть устанавливает входную матрицу, создает новый объект `factor`, и вызывает методы `cholesky`, `ludcomp`, и `qrdecomp`:

```

theFactor = new factor();
result = theFactor.cholesky(1, a);
result = theFactor.ludcomp(2, a);
result = theFactor.qrdecomp(2, a);

```

Отметим, что результат имеет тип массива `Object[]`. Он является аналогом массива ячеек. Например, после применения операции `ludcomp`, результат состоит из двух «ячеек» `result[0]` и `result[1]`, в первой хранится двумерный числовой массив – матрица L , а во второй, `result[1]` – матрица U .

Эта часть выполняется в блоке `try`. Это сделано для того, чтобы в случае исключения была выполнена передача в блок `catch`. Вторая часть есть блок `catch`. Код печатает сообщение на стандартный вывод, чтобы сообщить пользователю об ошибке. Третья часть есть блок `finally`, чтобы вручную очистить свои ресурсы перед выходом.

2. Установка переменных среды и компиляция приложения. Скомпилируем приложение `getfactor.java` используя программу `javac`. Сначала необходимо установить переменные `JAVA_HOME`, `CLASSPATH` и пути, как указано в первом параграфе. Считаем, что `JAVA_HOME` и пути установлены в предыдущем разделе. Для установки переменной `CLASSPATH` достаточно исполнить следующую команду в строке DOS:

```

set CLASSPATH=
.;\matlabroot\toolbox\javabuilder\jar\javabuilder.jar;
D:\javabuilder_examples\MatrixMathExample\factormatrix\distrib\
factormatrix.jar

```

Другой способ заключается в том, чтобы указать пути класса в командной строке Java при создании класса из кода `powerspect.java` следующим образом. Перейти в каталог `D:\javabuilder_examples\MatrixMathExample` и исполнить в нем следующую команду DOS (одна строка):

```

javac -classpath
;matlabroot\toolbox\javabuilder\jar\javabuilder.jar;.\factormatrix\
distrib\factormatrix.jar .\MatrixMathDemoJavaApp\getfactor.java

```

Эту команду удобно записать в файле `Class.bat` и исполнить его. При вводе команды имейте ввиду, что не должно быть пробелов между `javabuilder.jar`; и `.\factormatrix\distrib\factormatrix.jar`.

Компиляция производит файл `getfactor.class` и помещает его в подкаталог `MatrixMathDemoJavaApp`. Напомним, что если компонент создан с использованием `mcc`, то Java Builder не создает каталог `distrib`, поэтому пути должны быть скорректированы.

При использовании компонента на другой машине (без MATLAB) вместо подкаталога `matlabroot/toolbox/javabuilder/jar` нужно указывать подкаталог MCR: `MCRroot/toolbox/javabuilder/jar`. Также должен быть путь `<mcr_root>\<ver>\runtime\win32` к библиотекам MCR.

3. Запуск приложения. Для запуска приложения **getfactor** нужно выполнить следующее:

- поместить файл `getfactor.class` в каталог запуска приложения `D:\javabuilder_examples\MatrixMathExample`;
- определить пути для библиотек MATLAB (или для MCR)

`PATH <matlabroot>\bin\win32;`

или

`PATH <mcr_root>\v76\runtime\win32;`

- выполнить следующую команду в строке DOS (нужно указывать полный путь в командной строке к команде `java`):

```
matlabroot\sys\java\jre\win32\jre1.5.0_07\bin\java -classpath
.;matlabroot\toolbox\javabuilder\jar\javabuilder.jar.\factormatrix\
distrib\factormatrix.jar getfactor 4
```

Эту команду удобно записать в bat-файле и исполнить его (см. файл `Run_M.bat` на прилагаемом компакт-диске). Программа `getfactor` должна отобразить следующий вывод (рис. 3.4.4):

```
C:\WINDOWS\system32\cmd.exe
D:\javabuilder_examples\MatrixMathExample>D:\R2007a\bin\java -classpath .;D:\R2007a\toolbox\javabuilder\factormatrix\distrib\factormatrix.jar getfactor 4
Original matrix:
  2   -1   0   0
 -1   2  -1   0
  0  -1   2  -1
  0   0  -1   2
Cholesky factorization:
  1.4142  -0.7071   0   0
  0   1.2247  -0.8165   0
  0   0   1.1547  -0.8660
  0   0   0   1.1180
LU factorization:
L matrix:
  1.0000   0   0   0
 -0.5000   1.0000   0   0
  0  -0.6667   1.0000   0
  0   0  -0.7500   1.0000
U matrix:
  2.0000  -1.0000   0   0
  0   1.5000  -1.0000   0
  0   0   1.3333  -1.0000
```

Рис. 3.4.4. Результаты работы программы `getfactor`

Чтобы выполнить программу для разреженной матрицы, используется та же самая команда с добавлением строки «`sparse`» вторым аргументом в командной строке:

```
java (... те же аргументы) getfactor 4 sparse
```

3.5. Некоторые вопросы программирования

Дополнительно к приведенным образцам кодов в предыдущих параграфах, рассмотрим в этом разделе некоторые дополнительные вопросы программирования: импорт классов, создание экземпляра класса, который инкапсулирует код MATLAB, интерфейсы методов, создаваемых Java Builder, согласование типов данных между MATLAB и Java, обработку ошибок, освобождение памяти, используемой классами MWAarray и вызов сигнатуры для передачи параметров и возвращения вывода.

3.5.1. Импорт классов и создание экземпляра класса

Для использования компонента, созданного MATLAB Builder для Java, нужно импортировать библиотеки MATLAB и классы компонента, созданные Java Builder функцией `import Java`, например:

```
import com.mathworks.toolbox.javabuilder.*;  
import com.mathworks.componentname.classname;
```

Строка импорта должна включать полное имя пакета. Можно импортировать все классы компонента, обозначив их звездочкой (`*`).

Замечание. Компоненты Java Builder нужно создавать на той же самой операционной системе, на которой их предполагается установить (машина пользователя). Причины для этого ограничения в том, что `ctf`-файл зависит от платформы.

Как и для любого класса Java, перед использованием класса нужно создать экземпляр класса, образованного при помощи MATLAB Builder для Java. Предположим, что создан компонент с именем **MyComponent** с классом названным **MyClass**. Тогда создание экземпляра `ClassInstance` класса `MyClass` выглядит так:

```
MyClass ClassInstance = new MyClass();
```

Например, в примере предыдущего параграфа, задание экземпляра `theFourier` класса `fourier` и вызов его метода `plotfft` производится следующим образом:

```
fourier theFourier = null;  
theFourier = new fourier();  
result = theFourier.plotfft(3, data, new Double(interval));
```

3.5.2. Правила обращения к методам Java Builder

Java Builder преобразует функции MATLAB в один или более классов Java, которые составляют компонент Java. Каждая функция MATLAB реализуется как метод класса Java и может быть вызвана из приложения Java. Правила вызова метода Java Builder определяются интерфейсом, или сигнатурой метода.

Напомним, что *сигнатуры* (signature) метода образуют имя метода, число и типы параметров. Компилятор различает методы не по их именам, а по сигнатурам. Это позволяет записывать разные методы с одинаковыми именами, различающиеся числом и/или типами параметров. Такое дублирование методов называется *перегрузкой*. Тип возвращаемого значения не входит в сигнатуру метода, значит, методы не могут различаться только типом результата их работы.

Java Builder создает два вида интерфейсов для методов, соответствующих функциям MATLAB: стандартные и `mlx` интерфейсы. При этом Java Builder создает несколько интерфейсов каждого типа – для перегруженных методов, которые отличаются числом аргументов.

Напомним, что при создании компонента Java Builder записывает файлы в два подкаталога **distrib** и **src**. В каталоге **src** создается еще один подкаталог, имеющий имя компонента, например, **factormatrix**. Интерфейсы методов записываются в этот подкаталог в `java`-файл, имеющий имя класса, например, **factor.java**.

Стандартный интерфейс

Этот интерфейс определяет входные параметры для каждого перегруженного метода как массивы класса `MWArray`, или как массивы класса `java.lang.Object` и любого его подкласса (любого поддерживаемого типа Java). Аргументы, передаваемые как типы Java, преобразовываются в массивы MATLAB по правилам преобразования значений по умолчанию.

Стандартный интерфейс определяет возвращаемые значения, если таковые имеются, как массив типа `Object[]`. Стандартный интерфейс вызова возвращает массив из одного или более объектов `Object`. Каждый выходной массив должен быть в конце программы освобожден вызовом метода `dispose()`.

Для общей функции MATLAB

```
function [Out1, Out2, ..., varargout] = foo(In1, In2, ..., InN,
varargin)
```

стандартный интерфейс, в зависимости от числа параметров, имеет вид:

- если нет входных параметров:

```
public Object[] foo(int numArgsOut)
```
- если есть один входной параметр:

```
public Object[] foo(int numArgsOut, Object in1)
```
- если есть от двух до N входных параметров:

```
public Object[] foo(int numArgsOut, Object in1, ... Object inN)
```
- если есть дополнительные параметры, представленные аргументом `varargin`:

```
public Object[] foo(int numArgsOut, Object in1, ..., Object inN,
Object varargin)
```

Здесь `numArgsOut` есть целое число, указывающее число предполагаемых выводов метода. Чтобы не возвращать никаких параметров, нужно опустить этот

параметр. Входные параметры `in1, ... inN` имеют тип `Object[]`. Каждый необходимый ввод может иметь класс `MWArray`, или любой производный из `MWArray` класс. Выходной массив всегда имеет тип `Object[]` длины `numArgsOut`.

Пример. При создании проекта **factormatrix** предыдущего параграфа, Java Builder создает два стандартных интерфейса для метода `cholesky`: без входных параметров и с одним входным параметром (два перегруженных метода, которые отличаются числом аргументов). Все они записываются в файле `factor.java` в каталоге `\MatrixMathExample\factormatrix\src\factormatrix`. Приведем коды этих методов:

```
public Object[] cholesky(int nargout) throws MWException
{
    Object[] lhs = new Object[nargout];
    iMCR.invoke(Arrays.asList(lhs), Collections.EMPTY_LIST,
                scholeskySignature);
    return lhs;
}

public Object[] cholesky(int nargout, Object A) throws MWException
{
    Object[] rhs = {A};
    Object[] lhs = new Object[nargout];
    iMCR.invoke(Arrays.asList(lhs), Arrays.asList(rhs),
                scholeskySignature);
    return lhs;
}
```

Интерфейс `mlx`

Отличается тем, что выходные массивы входят в число аргументов, а сам метод не возвращает никакого значения (тип `void`). Этот интерфейс позволяет пользователю определять входы функции как массив `Object[]`, где каждый элемент массива — один входной параметр. Аналогично пользователь также дает предварительно распределенный массив `Object[]` для выводов функции. Длина массива вывода определяется числом выводов функции. Каждый выходной массив должен быть в конце программы освобожден вызовом метода `dispose()`

Стандартный интерфейс обычно используется, когда нужно вызвать функции MATLAB, которые возвращают единственный массив. В других случаях, возможно, предпочтительнее использовать интерфейс `mlx`.

К интерфейсу `mlx` можно также обратиться, используя контейнеры `java.util.List` вместо массивов `Object` для вводов и выводов. Отметим, что, если используется `java.util.List`, то передаваемый список вывода должен содержать число элементов, равных числу выводов функции.

Для функции со следующей структурой:

```
function [Out1, Out2, ..., varargout] = foo(In1, In2, ..., InN,
varargin)
```


Java Builder генерирует следующие интерфейсы `mlx`:

```
public void foo (List outputs, List inputs) throws MWException;
public void foo (Object[] outputs, Object[] inputs) throws
MWException;
```

Пример. При создании проекта **factormatrix** предыдущего параграфа, Java Builder создает два `mlx` интерфейса: используя контейнеры `java.util.List` и массивы `Object` для вводов и выводов (два перегруженных метода). Приведем коды этих методов:

```
public void cholesky(List lhs, List rhs) throws MWException
{
    iMCR.invoke(lhs, rhs, scholeskySignature);
}

public void cholesky(Object[] lhs, Object[] rhs) throws MWException
{
    iMCR.invoke(Arrays.asList(lhs), Arrays.asList(rhs),
        scholeskySignature);
}
```

Замечание. Как при стандартном, так и при интерфейсе `mlx`, результат получается в форме массива `Object[]`. Каждый элемент этого массива содержит один вывод, который также может быть массивом того типа, который определен соответствующей `m`-функцией. Поэтому, для получения отдельного результата нужно обратиться к элементу массива `Object[]`, а затем производить с ним какие-либо операции.

Например, в случае магического квадрата, сначала находим результат `result` как массив `Object[]`, состоящий из одного элемента:

```
theMagic = new magic(); // Экземпляр класса
result = theMagic.makesqr(1, n); // Вычисление маг. квадрата
```

Затем берем элемент `result[0]` – он содержит матрицу магического квадрата порядка `n`. Чтобы получить эту матрицу, преобразуем элемент `result[0]` в массив `MWNumericArray`. После того, как результат стал массивом MATLAB, преобразуем его в массив Java:

```
result_mw = new MWNumericArray(result[0], MWClassID.DOUBLE);
result_double = (double[][] )res_mw.toArray(); // Перевод в массив
double
```

3.5.3. Правила преобразования данных MATLAB и Java

В этом разделе рассмотрим правила преобразования данных от Java к MATLAB и обратно. Дополнительная информация о методах преобразовании данных имеется в разделах 3.3.1, 3.3.2.

Когда вызывается метод компонента MATLAB Builder для Java, то входные параметры, получаемые методом должны быть во внутреннем формате массива

MATLAB. Можно преобразовать их самостоятельно в пределах вызывающей программы, или передать параметры как типы данных Java, которые тогда автоматически преобразовываются механизмом вызова. Для преобразования данных самостоятельно (вручную), используются методы классов `MWArray`. Обычно используется комбинация ручного и автоматического преобразования.

Автоматическое преобразование в тип MATLAB

Когда параметр передается малое количество раз, то обычно эффективней передать его как примитивный тип или объект Java. В этом случае, вызывающий механизм преобразовывает данные в эквивалентный тип MATLAB, см. табл. 3.5.1. Например, любой из следующих типов Java будет автоматически преобразован в тип MATLAB `double`:

- Java примитив `double`;
- объект класса `java.lang.Double`.

Автоматическое преобразование данных показано в следующем фрагменте кода:

```
result = M.makesqr(1, arg[0]);
```

В этом случае Java `double` передается как `arg[0]`. Приведем еще один пример:

```
result = theFourier.plotfft(3, data, new Double(interval));
```

В этой инструкции Java, третий параметр имеет тип `java.lang.Double`. Согласно правилам преобразования, `java.lang.Double` автоматически преобразовывается в 1-на-1 массив MATLAB `double`.

Преобразование Java к MATLAB. Следующая таблица перечисляет правила автоматического преобразования типов данных Java в соответствующие типы MATLAB. Отметим, что эти правила относятся к скалярам, векторам, матрицам и многомерным массивам перечисленных типов.

Таблица 3.5.1. Правила преобразования Java к MATLAB

Тип Java	Тип MATLAB
<code>double</code>	<code>double</code>
<code>float</code>	<code>single</code>
<code>byte</code>	<code>int8</code>
<code>int</code>	<code>int32</code>
<code>short</code>	<code>int16</code>
<code>long</code>	<code>int64</code>
<code>char</code>	<code>char</code>
<code>boolean</code>	<code>logical</code>
<code>java.lang.Double</code>	<code>double</code>
<code>java.lang.Float</code>	<code>single</code>
<code>java.lang.Byte</code>	<code>int8</code>
<code>java.lang.Integer</code>	<code>int32</code>
<code>java.lang.Long</code>	<code>int64</code>
<code>java.lang.Short</code>	<code>int16</code>
<code>java.lang.Number</code>	<code>double</code>
<code>java.lang.Boolean</code>	<code>logical</code>
<code>java.lang.Character</code>	<code>char</code>
<code>java.lang.String</code>	<code>char</code>

Замечание 1. Напомним, что объекты классов `java.lang.Double` и т.п. являются ссылочными, соответствующим простым типам данных. Эти объекты содержат единственное поле, тип которого является типом соответствующего примитива и имеют несколько методов, полезных для работы с соответствующими простыми типами.

Замечание 2. Строка Java преобразовывается в 1-на-N массив `char` с N, равном длине входной строки. Массив строк Java (`String[]`) преобразовывается M-на-N массив `char`. При этом M равно числу элементов во входном массиве, а N равно максимальной длине строк в массиве, с соответствующим дополнением нулями, когда представленные строки имеют различные длины. Массивы `String` более высокой размерности преобразуются аналогично.

Преобразование типов данных вручную

Преобразование данных из Java в типы MATLAB можно также сделать самостоятельно в пределах программного кода. Для этого используются конструкторы массивов MATLAB, описание которых дано в параграфе 3.5. Преобразование данных из MATLAB в типы Java делается всегда вручную. Для этого имеется ряд методов преобразования, описание которых также приведено в параграфе 3.5.

Преобразование типов Java в типы MATLAB. Покажем на примерах использование конструкторов для получения массива MATLAB из данных Java. В следующем примере кода вызывается конструктор класса `MWNumericArray` для преобразования `double` Java в 16-разрядный целочисленный 1-на-1 массив MATLAB:

```
double Adata = 24;
MWNumericArray A = new MWnumericArray(Adata, MWClassID.INT16);
```

Если не указывать в конструкторе `MWClassID`, то он будет действовать по умолчанию в соответствии с табл. 3.5.1.

Следующий фрагмент кода программы `getmagic.java` раздела 3.2.5 показывает, как параметр класса `java.lang.Double` преобразовывается в тип `double` `MWNumericArray`, который может использоваться `m`-функцией без дальнейшего преобразования.

```
MWNumericArray n = null;
n = new MWNumericArray(Double.valueOf(args[0]), MWClassID.DOUBLE);
result = theMagic.makesqr(1, n);
```

Следующий пример показывает создание 32-разрядного целочисленного массива 3-на-6 типа `MWNumericArray` из целочисленного массива Java:

```
int[][] Adata = {{ 1, 2, 3, 4, 5, 6},
                 { 7, 8, 9, 10, 11, 12},
                 {13, 14, 15, 16, 17, 18}};
MWNumericArray A = new MWNumericArray(Adata, MWClassID.INT32);
```

Преобразование типов MATLAB в типы Java. Для преобразования массива MATLAB в массив указанного примитивного типа данных, такого как `float` или `int`, используются следующие методы `toTypeArray`:

toByteArray,
toLongArray,
toImagDoubleArray,
toImagShortArray.

toDoubleArray,
toShortArray,
toImagFloatArray,

toFloatArray,
toImagArray,
toImagIntArray,

toIntArray,
toImagByteArray,
toImagLongArray,

Эти методы возвращают массив Java, соответствующий примитивному типу в имени метода. Возвращенный массив имеет ту же самую размерность как и основной массив MATLAB и указанный в методе тип. Например, если вызывается toShortArray, то возвращается массив типа short, независимо от типа данных в основном массиве. Поэтому при выполнении преобразования возможно усечение или другая потеря точности. Например, если вызывается toFloatArray на экземпляре класса MWArray, содержащего данные с двойной точностью, значения double усекаются до значений float – значений с одинарной точностью. Рекомендуемое соответствие типов указано в табл. 3.5.2. Эти методы могут также быть полезными в определении типов в массиве Java, когда размерность вещественного или комплексного массива MWArray известна, но тип данных – нет.

Пример. Следующий код показывает преобразование массива MATLAB в массив указанного примитивного типа тех же измерений.

```
// вызов скомпилированной m-функции
results = myobject.myfunction(2);
// известно, что первый вывод является числовой матрицей
MWArray resultA = (MWNumericArray) results[0];
double[][] a = resultA.toDoubleArray();
// известно, что второй вывод является 3-мерным числовым массивом
MWArray resultB = (MWNumericArray) results[1];
Int[][][] b = resultB.toIntArray();
```

Для преобразования массива MATLAB в одномерный массив указанного примитивного типа данных используются следующие методы **getTypeData**:

getBytesData,
getLongData,
getImagDoubleData,
getImagShortData

getDoubleData,
getShortData,
getImagFloatData,

getFloatData,
getImagData,
getImagIntData,

getIntData,
getImagByteData,
getImagLongData,

Для преобразования одного элемента массива MATLAB в примитивный тип Java можно также использовать методы getType(int) и getType(int[]), см. раздел 3.3.2.

Таблица 3.5.2 показывает соответствие типов данных MATLAB и типов Java, которое нужно учитывать при преобразовании. Отметим, что конверсионные правила применяются к скалярам, векторам, матрицам, и многомерным массивам перечисленных типов.

Таблица 3.5.2. Соответствие типов MATLAB и Java

Тип MATLAB	Простой тип Java	Тип Java Object
cell	Нет	Object
structure	Нет	Object
char	char	java.lang.Character

Таблица 3.5.2. Соответствие типов MATLAB и Java (окончание)

Тип MATLAB	Простой тип Java	Тип Java Object
double	double	java.lang.Double
single	float	java.lang.Float
int8	byte	java.lang.Byte
int16	short	java.lang.Short
int32	int	java.lang.Integer
int64	long	java.lang.Long
uint8	byte	java.lang.Byte
uint16	short	java.lang.Short
uint32	int	java.lang.Integer
uint64	long	java.lang.Long
logical	boolean	java.lang.Boolean
Function handle	Не поддерживается	
Java class	Не поддерживается	
User class	Не поддерживается	

Замечание. Массивы ячеек и структур не поддерживаются в Java, однако класс `MWArray`, созданный для Java Builder, поддерживает массивы ячеек `MWCellArray` и структур `MWStructArray`, так же как и другие массивы `MWArray`. Отметим также, что Java не имеет никаких типов без знака для представления типов `uint8`, `uint16`, `uint32`, и `uint64`, используемых в MATLAB. Конструкция и доступ к массивам MATLAB типа без знака требует преобразования. Дополнительную информацию о правилах преобразования см. в `com.mathworks.toolbox.javabuilder`.

3.5.4. Аргументы методов Java Builder

Обсудим некоторые вопросы, связанные с входными и выходными параметрами методов, созданных Java Builder.

Передача неопределенного числа параметров

Рассмотрим примеры использования *m*-функций, которые имеют параметры `varargin` или `varargout`. Для примера определим *m*-функцию, которая вычисляет сумму всех введенных слагаемых:

```
function y = mysum(varargin)
%   MYSUM Returns the sum of the inputs.
y = sum([varargin{:}]);
```

Отметим, что слагаемые аргументы объединяются функцией MATLAB `horzcat`. Поэтому они должны иметь одинаковое число строк, но могут иметь разное число столбцов. В случае числовых и векторных аргументов находится их сумма, в случае матричных аргументов – находится сумма элементов столбцов.

Поскольку слагаемых может быть разное число, то вводы задаются через параметр `varargin`, что означает, что вызывающая программа может определить любое число вводов функции. Результат возвращается как скаляр `double`.

Java Builder генерирует интерфейс Java (стандартный) для этой функции следующим образом:

```
public Object[] mysum(int nargsout, Object varargin) throws MWException
```

Параметр varargin передается как тип Object. Это позволяет любое число вводов в форме массива Object[]. Содержание этого массива передают к откомпилированной m-функции в порядке, в котором они появляются в массиве. Ниже идет пример того, как можно было бы использовать метод mysum в программе Java:

```
public double getsum(double[] vals)
{
    myclass cls = null;
    Object[] x = {vals};
    Object[] y = null;
    try
    {
        cls = new myclass();
        y = cls.mysum(1, x);
        return ((MWNumericArray)y[0]).getDouble(1);
    }
}
```

В этом примере создается массив Object длины 1 и инициализируется ссылкой на поставляемый массив {vals} типа double. Этот параметр передают к методу mysum. Результат, как известно, является скаляром double MATLAB, но возвращается как Object[]. Для получения значения double требуется следующий код:

```
return ((MWNumericArray)y[0]).getDouble(1);
```

Нужно привести возвращаемое значение y[0] к типу MWNumericArray и вызвать метод getDouble(int), чтобы получить первый элемент в массиве как примитивное значение double.

Передача массива вводов. Следующий пример выполняет более общее вычисление:

```
public double getsum(Object[] vals) throws MWException
{
    myclass cls = null;
    Object[] x = null;
    Object[] y = null;
    try
    {
        x = new Object[vals.length];
        for (int i = 0; i < vals.length; i++)
            x[i] = new MWNumericArray(vals[i], MWClassID.DOUBLE);
        cls = new myclass();
        y = cls.mysum(1, x);
        return ((MWNumericArray)y[0]).getDouble(1);
    }
}
```

Эта версия `getsum` берет массив `vals` типа `Object` как ввод и преобразовывает каждое значение `vals[i]` в массив `x[i]` типа `double` MATLAB. Набор массивов MATLAB образует массив типа `Object[]`, который передается к функции `mysum`, где она вычисляет полную сумму входного массива.

Передача переменного числа выводов. Следующий пример кода показывает, что параметры `varargout` обрабатываются таким же образом, как и параметры `varargin`. Рассмотрим следующую `m`-функцию:

```
function varargout = randvectors
for i=1:nargout
    varargout{i} = rand(1, i);
end
```

Эта функция возвращает набор случайных векторов `double` такой, что длина i -го вектора равна i . Вызов в MATLAB этой функции следующий:

```
[y1, y2, y3, y4, y5, y6]=randvectors;
```

где слева можно указать любое число параметров вывода.

Компилятор MATLAB создает стандартный интерфейс Java для этой функции:

```
public Object[] randvectors(int nargout) throws MException
```

Таким образом, для обращения к соответствующему методу достаточно указать число n векторов вывода. Тогда результатом будет массив `y` типа `Object[]` длины n . Приведем пример использования метода `randvectors` в программе Java:

```
public double[][] getrandvectors(int n)
{
    myclass cls = null;
    Object[] y = null;
    cls = new myclass();
    y = cls.randvectors(n);
    double[][] ret = new double[y.length][];

    for (int i = 0; i < y.length; i++)
        ret[i] = (double[]) ((MArray)y[i]).getData();
    return ret;
}
```

Метод `getrandvectors` возвращает двумерный `double` массив с треугольной структурой. Длина i -ой строки равна i . Такие массивы обычно называются зубчатыми массивами. Зубчатые массивы легко поддерживаются в Java, потому что многомерный массив Java – это массив массивов.

Получение информации о результатах методов

Предыдущие примеры использовали известные тип и размерность параметра вывода. В случае, когда эта информация неизвестна, или может измениться (что возможно в `m`-программировании), код, который вызывает метод, возможно, должен сделать запрос о типе и размерности параметров вывода. Есть несколько способа сделать это:

- использовать отражение (reflection), поддерживаемое в языке Java, для того, чтобы выполнить запрос о типе любого объекта;
- использовать методы класса `MWArray` для запроса информации об основном массиве MATLAB;
- сделать приведение к определенному типу, используя методы `toTypeArray`.

Использование отражения Java. Напомним, что отражение дает возможность коду Java обнаружить информацию о полях, методах и конструкторах загруженных классов и использовать эти отраженные поля, методы и конструкторы.

Следующий фрагмент кода вызывает метод `myprimes`, созданный `Lava Builder`, и затем определяет тип, используя отражение (ключевое слово `instanceof` проверяет наследование объекта). Пример предполагает, что вывод возвращается как числовая матрица, но точный числовой тип неизвестен.

```
public void getprimes(int n) throws MWException
{
    myclass cls = null;
    Object[] y = null;
    try
    {
        cls = new myclass();
        y = cls.myprimes(1, new Double((double)n));
        Object a = ((MWArray)y[0]).toArray();
        if (a instanceof double[][])
        {
            double[][] x = (double[][])a;

            /* (некоторые действия с x...) */
        }
        else if (a instanceof int[][])
        {
            int[][] x = (int[][])a;

            /* (некоторые действия с x...) */
        }
        else
        {
            throw new MWException(
                "Bad type returned from myprimes");
        }
    }
}
```

Использование методов `MWArray`. Для определения типа MATLAB объекта `MWArray` можно использовать метод `classID`. Тип возвращения – поле, определенное классом `MWClassID`. Например,

```
cls = new myclass();
y = cls.myprimes(1, new Double((double)n));
MWClassID clsid = ((MWArray)y[0]).classID();
```


Для определения размеров массива MATLAB можно использовать метод `getDimensions`. Этот метод возвращает одномерный `int` массив, содержащий размер каждого измерения объекта `MWArray`, например:

```
int[] dimA = A.getDimensions();
```

Использование методов `toTypeArray`. Следующий фрагмент кода показывает, как можно привести данные к указанному числовому типу, вызывая любой из методов **`toTypeArray`** (см. раздел 3.3.2 для получения дополнительной информации). Эти методы возвращают массивы типов Java, соответствующих примитивному типу, определенному именем вызываемого метода. Отметим, что при использовании этих методов, данные могут усекаться.

```
// вызов скомпилированной m-функции
results = myobject.myfunction(2);
// известно, что первый вывод является числовой матрицей
MWArray resultA = (MWNumericArray) results[0];
double[][] a = resultA.toDoubleArray();
// известно, что второй вывод является 3-мерным числовым массивом
MWArray resultB = (MWNumericArray) results[1];
int[][][] b = resultB.toIntArray();
```

Передача объектов Java по ссылке

`MWJavaObjectRef` – есть специальный подкласс `MWArray`, который может использоваться для создания массивов MATLAB, которые ссылаются на объекты Java. Для более подробной информации об использовании этого класса, конструкторов и соответствующих методов, см. страницу `MWJavaObjectRef` в `JavaDoc` или по поиску `MWJavaObjectRef` в `Help MATLAB`.

3.5.5. Обработка ошибок

Об ошибках, которые происходят в течение выполнения `m`-функции или в течение преобразования данных, сообщает стандартное исключение Java. Оно включает ошибки во время выполнения программы MATLAB, а также ошибки в `m`-коде. Есть два типа исключений: исключения `MWException` и общие исключения.

Обработка исключений `MWException`

Исключения типа `MWException` должны быть объявлены пунктом **`throws`** языка Java. Компоненты Java Builder поддерживают одно исключение с проверкой: `com.mathworks.toolbox.javabuilder.MWException`. Этот класс исключения наследует `java.lang.Exception` и вызывается каждым методом Java, созданным Компилятором MATLAB для сообщения о том, что ошибка произошла в течение вызова. Все нормальные ошибки во время выполнения программы MATLAB, так же как созданные пользователем ошибки (например, ошибка запроса в `m`-коде) проявляются как `MWExceptions`.

Интерфейс Java для каждой `m`-функции содержит объявление о запуске `MWException` использованием пункта `throws`. Например, рассмотренная выше `m`-функция `makesqr`, имеет следующие интерфейсы:

```
public Object[] makesqr(int nargsout, Object x) throws MWException
```

Можно использовать два способа обработки ошибок. Покажем их на примерах.

Обработка исключения в вызванной функции. Фрагмент кода `getprimes` обрабатывает исключение непосредственно и не должен включить пункт `throws` в начале.

```
public double[] getprimes(int n)
{
    myclass cls = null;
    Object[] y = null;

    try
    {
        cls = new myclass();
        y = cls.myprimes(1, new Double((double)n));
        return (double[]) ((MWArray)y[0]).getData();
    }

    catch (MWException e)
    {
        System.out.println("Exception: " + e.toString());
        return new double[0];
    }
}
```

Отметим, что в этом случае программист сам решает, как возвратить что-нибудь разумное из метода в случае ошибки.

Обработка исключения в вызывающей функции. В следующем фрагменте кода, метод, который вызывает `myprimes` объявляет, что он вызывает `MWException`:

```
public double[] getprimes(int n) throws MWException
{
    myclass cls = null;
    Object[] y = null;

    try
    {
        cls = new myclass();
        y = cls.myprimes(1, new Double((double)n));
        return (double[]) ((MWArray)y[0]).getData();
    }

    finally
    {
        MWArray.disposeArray(y);
        if (cls != null)
            cls.dispose();
    }
}
```

Обработка общих исключений

Это исключения Java, которые не объявляются явно пунктом `throws`. Классы API `MWArray` все производят такие исключения. Следующие исключения могут быть вызваны `MWArray`: `java.lang.RuntimeException`, `java.lang.ArrayStoreException`, `java.lang.NullPointerException`, `java.lang.IndexOutOfBoundsException` и `java.lang.NegativeArraySizeException`. Этот список представляет наиболее вероятные исключения. Другие могут быть добавлены в будущем. Для информации относительно исключений, которые могут произойти для каждого метода класса `MWArray` и его подкласса, см. документацию: `Using MWArray Classes`.

Захват общих исключений. Легко модифицировать пример `getprimes` для захвата любого исключения, которое может произойти в течение вызова метода и преобразования данных. Нужно только заменить пункт `catch`, а именно, строку

```
catch (MWException e)
```

заменить на строку

```
catch (Exception e)
```

Захват разных типов исключения. Второй, и более общий, вариант этого примера различает исключения, которые происходят при вызове скомпилированного метода и все другие типы исключений, вводя два пункта `catch` следующим образом:

```
public double[] getprimes(int n)
{
    myclass cls = null;
    Object[] y = null;

    try
    {
        cls = new myclass();
        y = cls.myprimes(1, new Double((double)n));
        return (double[]) ((MWArray)y[0]).getData();
    }

    /* Захватывает исключение, вызванное myprimes */
    catch (MWException e)
    {
        System.out.println("Exception in MATLAB call: " +
            e.toString());
        return new double[0];
    }

    /* Захватывает все другие исключения */
    catch (Exception e)
    {
        System.out.println("Exception: " + e.toString());
        return new double[0];
    }
}
```

Порядок пунктов `catch` здесь важен. Поскольку `MWException` является подклассом `Exception`, пункт `catch` для `MWException` должен быть перед пунктом `catch` для `Exception`. Если порядок будет изменен, то пункт `catch` `MWException` никогда не будет выполняться.

3.5.6. Управление собственными ресурсами

Когда ваш код обращается к классам Java, созданным при помощи Java Builder, ваша программа использует собственные (native) ресурсы, которые существуют вне контроля Виртуальной Машины Java (JVM).

Каждый класс преобразования данных `MWArray` есть интерфейсный класс, который инкапсулирует тип `mxArray` MATLAB. Инкапсулированный массив MATLAB распределяет ресурсы из собственной (native) кучи памяти. Массив `mxArray` может быть большим, но он находится вне контроля JVM, поэтому менеджер памяти JVM может не вызывать сборщика мусора прежде, чем становится исчерпанной или сильно фрагментированной собственная память. Это означает, что собственные массивы должны быть явно освобождены. Можно использовать любую из следующих методик к освобождению памяти: «сборка мусора» JVM; метод `dispose` и метод `Object.Finalize`.

Использование «сборки мусора» JVM

При создании нового экземпляра класса Java, JVM распределяет и инициализирует новый объект. Когда этот объект перестает быть необходимым, или становится недостижимым, он попадает в категорию мусора JVM. Сборщик мусора освобождает память, распределенную для объекта.

При создании экземпляров классов `MWArray`, инкапсулирующих типы MATLAB, распределяется также пространство для собственных ресурсов, но эти ресурсы невидимы для JVM и не могут попасть в категорию мусора JVM.

Ресурсы, распределенные объектам `MWArray` могут быть весьма большими и могут быстро исчерпать доступную память. Чтобы избежать этого, объекты `MWArray` должны быть как можно скорее явно освобождены приложением, которое создает их.

Использование метода *dispose*

Лучшая техника для освобождения ресурсов классов, созданных Java Builder – это явный вызов метода `dispose`. Любой объект Java, включая объект `MWArray`, имеет метод `dispose`.

Классы `MWArray` также имеют метод `finalize`, называемый завершителем (`finalizer`), который вызывает `dispose`. Хотя можно считать завершитель `MWArray` как своего рода сеть безопасности для случаев, когда `dispose` не вызывается явно, нужно иметь в виду, что невозможно определить точно, когда JVM вызывает завершитель и JVM может не обнаружить память, которая должна быть освобождена.

Использование dispose. Следующий фрагмент кода распределяет собственный массив приблизительно на 8 Мбайт. Для JVM размер обернутого объекта – только несколько байтов (размер экземпляра `MWNumericArray`) и это – несущественный размер для вызова сборщика мусора. Пример показывает, что в этом случае необходимо явное освобождение `MWArray`.

```
/* Распределение большого массива */
int[] dims = {1000, 1000};
MWNumericArray a = MWNumericArray.newInstance(dims,
    MWClassID.DOUBLE, MWComplexity.REAL);

    .
    . // использование массива a
    .

/* Освобождение своих ресурсов */
a.dispose();
/* Сделать пригодным для сборки мусора */
a = null;
```

Утверждение `a.dispose()` освобождает память, распределенную и для обертки и для собственного массива MATLAB.

Класс `MWArray` обеспечивает два метода освобождения: `dispose` и `disposeArray`. Метод `disposeArray` является более общим в том, что он избавляется от любого простого `MWArray` или массива массивов типа `MWArray`.

Использование блока try-finally для гарантированного освобождения ресурсов. Лучше вызывать метод `dispose` не из пункта `finally`, а в блоке `try-finally`. Эта методика гарантирует, что все собственные ресурсы освобождаются перед выходом из метода, даже если возникает исключение в некоторый момент перед кодом очистки. Это показывает следующий фрагмент кода:

```
/* Распределение большого массива */
MWNumericArray a;
try
{
    int[] dims = {1000, 1000};
    a = MWNumericArray.newInstance(dims,
        MWClassID.DOUBLE, MWComplexity.REAL);

    .
    . // использование массива a
    .
}

/* Освобождение собственных ресурсов */
finally
{
    a.dispose();
/* Сделать пригодным для сборки мусора */
a = null;
}
```

3.6. Среда проектирования JBuilder

В этом параграфе рассмотрим очень кратко среду проектирования JBuilder 2006. Она построена на тех же принципах, что и аналогичные среды разработки других языков. Поэтому переход на JBuilder не представляет трудностей. Начинаящим программистам полезно обратиться к какому-нибудь руководству, например [Ba], [Пон]. Интересной особенностью JBuilder является то, что виртуальная Java машина постоянно следит за правильностью написания кода и отмечает ошибки сразу, до этапа компиляции.

При запуске JBuilder открывается окно интегрированной среды разработки (IDE) в котором можно выполнить большинство функций разработки: редактирование кода, визуальное проектирование, навигацию, просмотр, компиляцию, отладку, и другие операции. Это окно – рабочее пространство JBuilder и оно состоит из нескольких областей, предназначенных для разработки приложения, рис. 3.6.1.

Строка меню (Menu bar). Включает команды меню для проектирования, разработки, тестирования, развертывания, и управления вашими приложениями Java. Строка меню включает следующие меню: **File, Edit, Search, Refactor, View, Project, Run, Team, Enterprise, Tools, Window** и **Help**.

Основная инструментальная панель (Main toolbar). Находится под строкой меню. Она состоит из меньших инструментальных панелей, сгруппированных по функциональным возможностям: **File, Edit, Search, Build, Run/Debug, Navigate, Help** и **Workspaces**.

Рабочая область (Content pane). Наибольшая область рабочего пространства. Служит для просмотра и редактирования открытых файлов. Имеет несколько вкладок внизу области окна: **Source, Design, Bean, UML, Doc, History**, указывающих режим просмотра файла. Например, вкладка **Source** открывает редактор кода, а вкладка **Design** позволяет перейти в режим визуального проектирования формы приложения, она открывает окно дизайнера формы.

Проектная область (Project pane). Проектная область окна отображает файлы проекта и обеспечивает (при помощи правой кнопки мыши) доступ к командам всего проекта (рис. 3.7.1).

Окно структуры (Structure pane). Отображает структуру файла, в настоящее время активного в рабочей области окна (рис. 3.6.1).

Окно сообщений (Message pane). Отображает вывод информации о различных процессах, типа сообщений компилятора, результатов поиска, отладчик UI, Javadoc, тестирования модуля и т.п. Здесь могут быть и другие компоненты. Например, при запуске сеанса отладки, в области окна сообщений появляется отладчик.

Строки текущего состояния (Status bars). Содержат информацию об открытых файлах и происходящих процессах.

Перед созданием приложения в JBuilder необходимо сначала создать проект. JBuilder использует проектный файл с расширением **.jrx**, чтобы организовать прикладные файлы и поддерживать параметры настройки и свойства проекта.

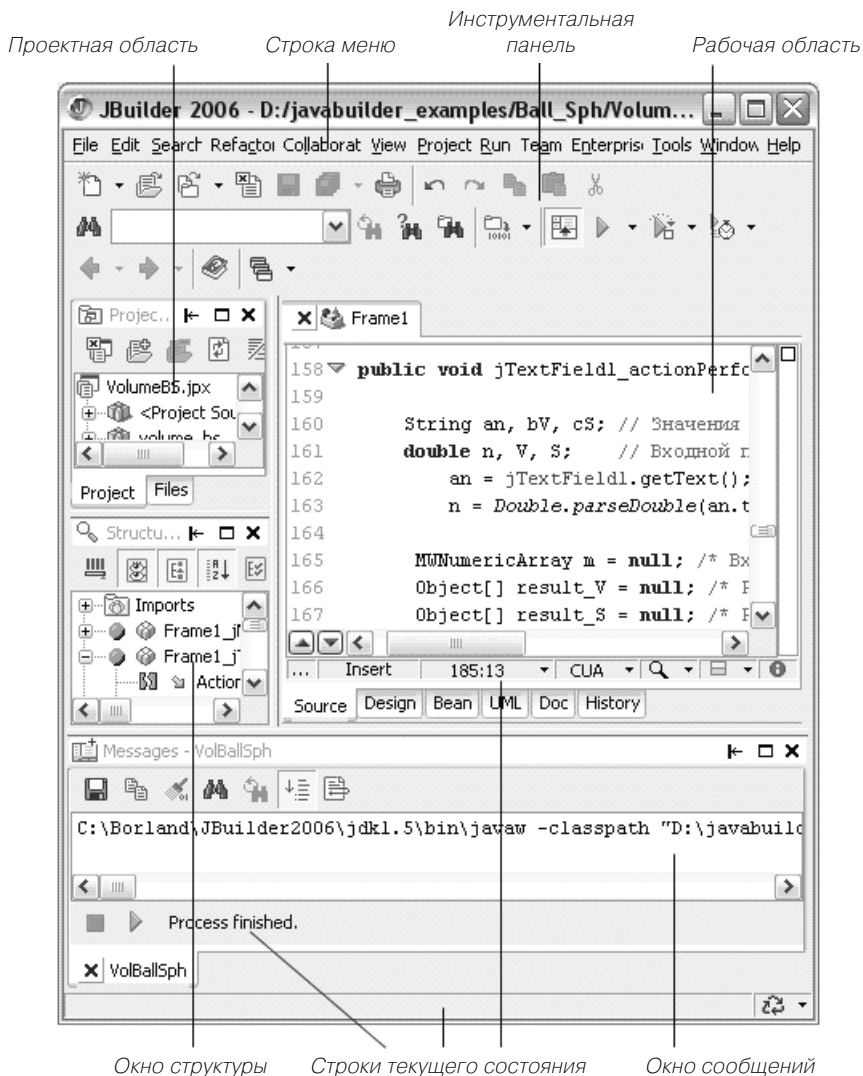


Рис. 3.6.1. Рабочее пространство JBuilder

Каждый проект также включает файл `.jpx.local`, который содержит хронологический список локальных параметров настройки редактора и некоторых действий пользователя, типа отладки. Проектный файл не редактируется непосредственно пользователем, но он меняется всякий раз при добавлении или удалении файлов или установки свойства проекта. Проектный файл является главным узлом дерева проекта в проектной области окна.

Перед созданием нового проекта необходимо закрыть все открытые проекты. Создание приложения на JBuilder начинается выбора меню **File** ⇒ **New**. Открыва-

ющееся диалоговое окно (рис. 3.6.2) содержит большой набор шаблонов приложений. После выбора одного из них (например, **Application**), нужно нажать **OK** для запуска Мастера проекта.

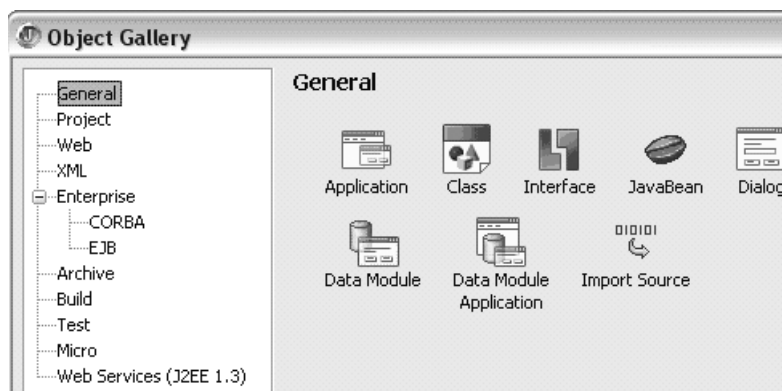


Рис. 3.6.2. Выбор проекта JBuilder

Мастер проекта помогает выбрать имя проекта, каталог, шаблон, необходимые пути и общие установки. Соответствующие диалоговые окна понятны и не требуют дополнительного описания (рис. 3.6.3–3.6.5). Когда печатается название проекта в поле **Name**, то же самое название вводится в поле **Directory**. По умолчанию, JBuilder использует имя проекта для создания имени каталога проекта и названия пакета для классов. Имя пакета будет в нижнем регистре согласно соглашению Java для названий пакетов. Проекты в JBuilder сохраняются по умолчанию в каталоге C:/Documents and Settings/UserName/jbproject/. Можно задать другой каталог.

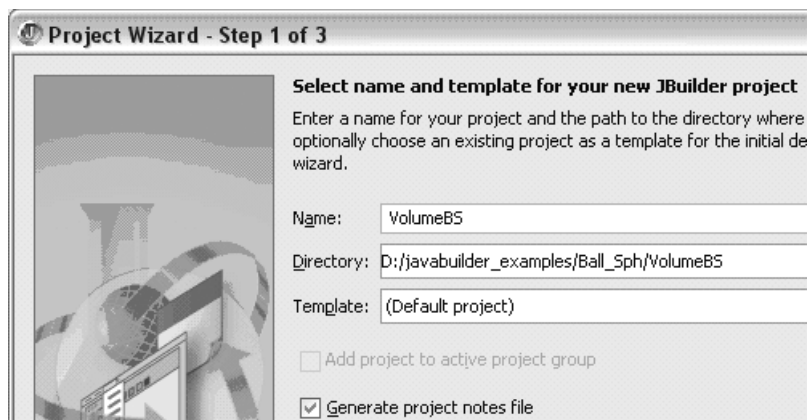


Рис. 3.6.3. Шаг 1. Выбор имени, каталога и шаблона проекта

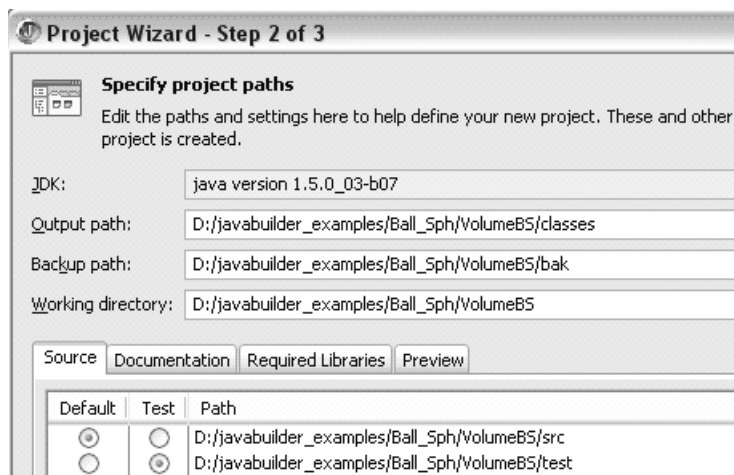


Рис. 3.6.4. Шаг 2. Выбор путей проекта

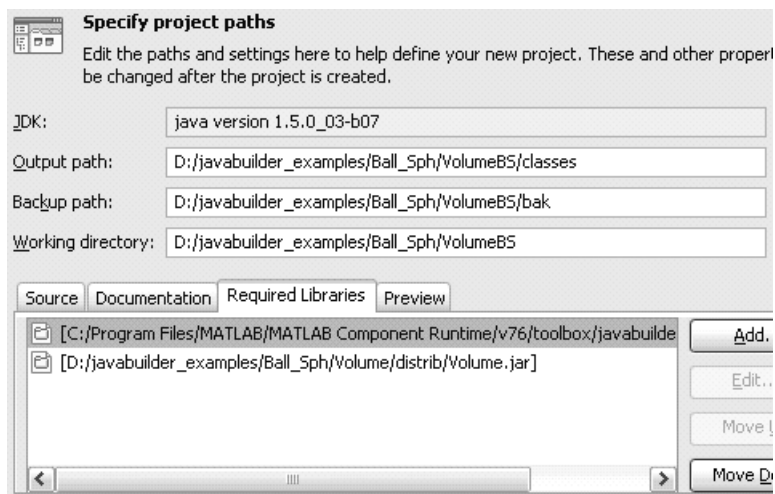


Рис. 3.6.5. Шаг 3. Выбор путей библиотек и архивов, необходимых проекту

На последнем, третьем шаге нажимаем кнопку **Finish** и переходим ко второму этапу – созданию приложения, основанного на шаблоне **Application**.

Запускается Мастер приложений и выбирается имя класса, пакета, имя класса формы приложения, необходимые элементы формы и конфигурация (рис. 3.6.6–3.6.8). Мастер приложений создает .java исходные файлы, которые добавляются к проекту, который только что был создан на первом этапе.

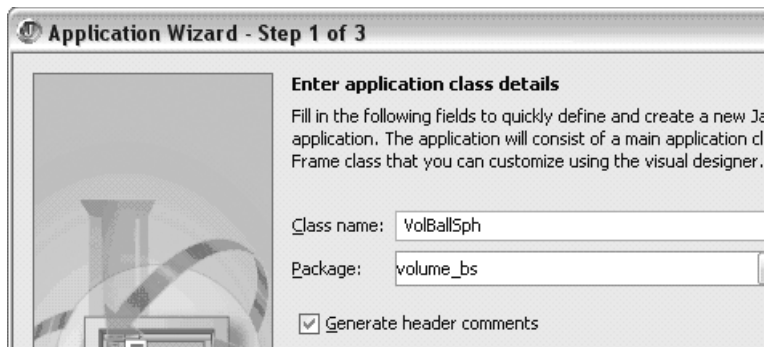


Рис. 3.6.6. Шаг 1. Выбор имени класса, пакета и шаблона проекта



Рис. 3.6.7. Шаг 2. Выбор имени класса формы и необходимых элементов формы

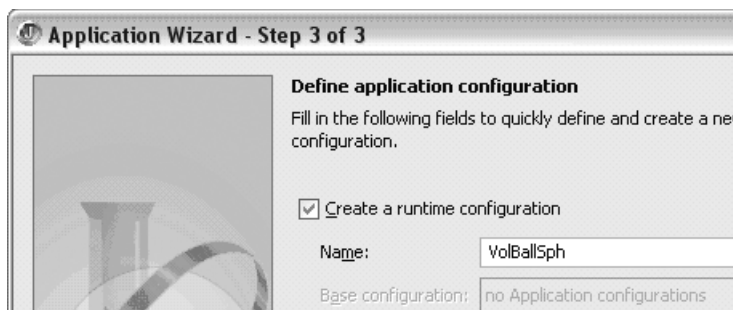


Рис. 3.6.8. Шаг 3. Выбор конфигурации приложения

На последнем, третьем шаге нажимаем кнопку **Finish** и JBuilder создает большой набор файлов проекта выбранного шаблона приложения, включая исходные файлы.

Теперь остается внести в исходные файлы необходимый программный код, выбрать элементы формы приложения и скомпилировать приложение.

Рабочая область приложения по умолчанию открывается на вкладке **Source** и содержит исходный код Java окна приложения **Frame1**, открытый в редакторе кода Java.

Вкладка **Design** открывает окно дизайнера для визуального проектирования приложения. Окно дизайнера состоит из трех частей (рис. 3.6.9):

- палитра компонентов, расположена слева;
- окно приложения, в котором будут размещаться выбранные компоненты;
- окно инспектора, в котором устанавливаются свойства выбранных компонентов.

При выборе вкладки **Design**, JBuilder отображает проектировщика **UI**. Он используется для построения пользовательского интерфейса, который содержат визуальные элементы, типа кнопок и текстовых полей. Можно выбрать другие проектировщики: **Menu** – проектировщик меню, **Data Access** – проектировщик для разработки базы данных и проектировщик Default (рис. 3.6.9).

По умолчанию открыта палитра визуальных компонент библиотеки **Swing** (рис. 3.6.9). Ниже идут другие библиотеки визуальных компонент: Swing Containers, DataExpress, dbSwing, More dbSwing, dbSwing Modules, InternetBeans, XML, EJB, AWT, COBRA, MIDP1, MIDP2, MIDP Screens.

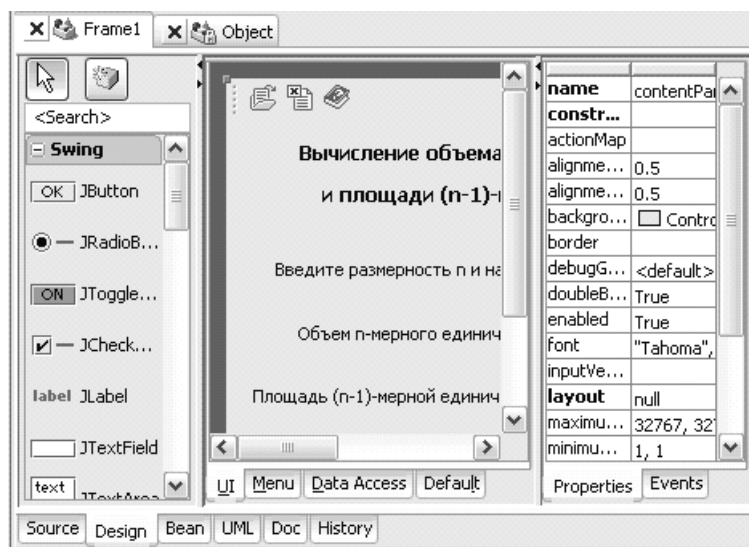


Рис. 3.6.9. Палитра компонент класса Swing и окно инспектора компонент

Визуальные компоненты выбираются левой кнопкой мыши и переносятся на форму обычным образом. При это нужно иметь ввиду, что в JBuilder по умолчанию установлено упорядочивание выбираемых компонент – свойство **Layont** инспектора компонент установлено как **BorderLayont**. Если Вы хотите сами определять размер и положение визуальных компонент, нужно установить свойство **Layont** как **null**.

Проектная область окна отображает файлы проекта и обеспечивает (при помощи правой кнопки мыши) доступ к командам всего проекта (рис. 3.6.10). Окно структуры отображает структуру файла, в настоящее время активного в рабочей области окна (рис. 3.6.10).

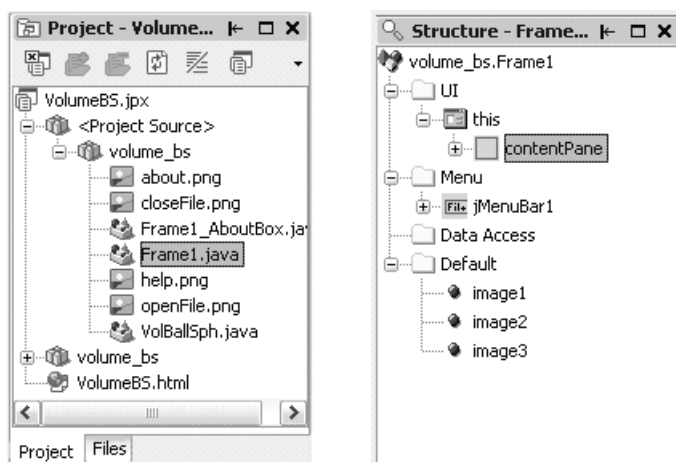


Рис. 3.6.10. Проектная область и окно структуры

Во время разработки проекта можно менять свойства проекта используя меню **Project ⇒ Project Properties**.

В заключение нужно выбрать в меню **File ⇒ Save All** для сохранения исходных файлов и проектного файла в выбранном каталоге.

3.7. Примеры создания приложений с использованием классов Java Builder

Рассмотрим два примера создания оконных приложений, в которых используются классы из компонент, созданных с помощью Java Builder. Исходные тексты примеров находятся на прилагаемом компакт-диске в каталоге Gl_3_javabuilder_examples.

3.7.1. Объем n -мерного шара и площадь $(n-1)$ -мерной сферы

В этом разделе рассмотрим создание приложения на JBuilder для вычисления объема n -мерного шара $B^n(1)$ в \mathbf{R}^n единичного радиуса и площади $(n-1)$ -мерной сферы $S^{n-1}(1)$ в \mathbf{R}^n единичного радиуса.

Из курса математического анализа известно, что объем n -мерного шара $B^n(r)$ в \mathbf{R}^n радиуса r вычисляется по формуле

$$\text{Vol}(B^n(r)) = \frac{\pi^{n/2}}{\Gamma(1+n/2)} r^n,$$

где $\Gamma(s) = \int_0^{+\infty} e^{-t} t^{s-1} dt$ – Γ -функция Эйлера. Площадь $(n-1)$ -мерной сферы $S^{n-1}(r)$ в \mathbf{R}^n радиуса r вычисляется по формуле

$$\text{Vol}(S^{n-1}(r)) = \frac{2\pi^{n/2}}{\Gamma(n/2)} r^{n-1}.$$

Достаточно вычислить объем $B^n(1)$ единичного n -мерного шара и $(n-1)$ -мерной единичной сферы $S^{n-1}(1)$ в \mathbf{R}^n , тогда

$$B^n(r) = B^n(1) \cdot r^n \quad \text{и} \quad S^{n-1}(r) = S^{n-1}(1) \cdot r^{n-1}.$$

Составим m -функции вычисления объема единичного n -мерного шара и $(n-1)$ -мерной единичной сферы в \mathbf{R}^n :

```
function y = Vol_n(n)
y = (pi)^(n/2) / gamma(1+n/2);
```

```
function y = Sph_nmin1(n)
y = 2 * (pi)^(n/2) / gamma(n/2);
```

Создание компонента Java Builder

Поместим созданные выше m -функции в каталог D:\javabuilder_examples\Ball_Sph. Запустим сеанс MATLAB и сделаем этот каталог текущим рабочим каталогом.

Откроем графический интерфейс разработки следующей командой MATLAB: `deploytool`

Создадим новый проект **MATLAB Builder for Java**, тип компонента **Java Package**. Выберем имя проекта **Volume.prj** (вместо `untitled1.prj`), каталог, где создается проект D:\javabuilder_examples\Ball_Sph и нажмем **ОК**. Проект содержит две папки: **Volumeclasses** и **Other files**. Добавим в папку **Volumeclasses** проекта файлы `Vol_n.m` и `Sph_nmin1.m`. Для этого нужно эту папку активизировать и добавить в нее файлы используя либо меню **Project**, либо кнопку инструментальной панели, либо правую кнопку мыши (рис. 3.7.1). Сохраним проект.

Для построения компонента, инкапсулирующего функции MATLAB `Vol_n.m` и `Sph_nmin1.m`, нужно исполнить команду **Build** из меню **Tools**, либо нажать

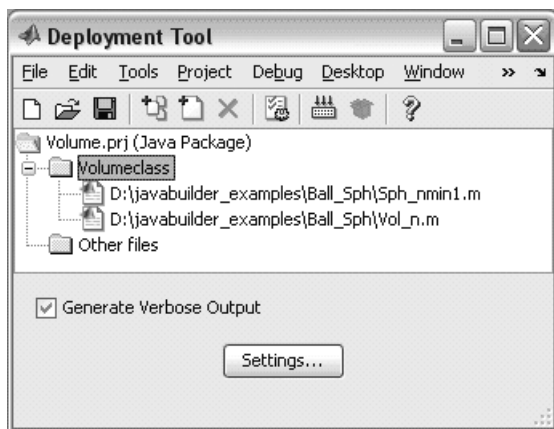


Рис. 3.7.1. Проект Volume Java Builder

кнопку построения (**Build**) на инструментальной панели. В результате процесса построения в каталоге проекта создается подкаталог **Volume**, содержащий два подкаталога **distrib** и **src**, в которые помещаются созданные файлы компонента.

Подкаталог **distrib** содержит файлы **Volume.jar** и **Volume.ctf**. Файл **Volume.jar** – это java-архив созданных классов проекта, а **Volume.ctf** – технологический файл компоненты, содержащий зашифрованные m-функции, которые составляют компонент и все другие m-функции MATLAB, от которых зависят основные m-функции. Подкаталог **src** содержит копии файлов **Volume.jar** и **Volume.ctf**, log-файлы регистрации процесса построения, а также файл **readme.txt**, который содержит полезную информацию для распространения пакета. Кроме того, в подкаталоге **classes** содержатся созданные классы компонента, а в подкаталоге **Volume** содержатся интерфейсные Java-файлы.

Создание приложения JBuilder

Для создания приложения запустим сеанс JBuilder и создадим новый проект с именем **VolumeBS.jrx** в каталоге **D:\javabuilder_examples\Ball_Sph\VolumeBS**, рис. 3.7.2.

Подключение классов MATLAB и классов созданного компонента Volume.

Для создания и работы приложения нужно подключить к проекту файлы **javabuilder.jar** и **Volume.jar**. Первый файл **javabuilder.jar** находится в каталоге **C:\Program Files\MATLAB\MATLAB Component Runtime\v76\toolbox\javabuilder\jar** и содержит классы **MWArray** и подклассы, необходимые для компиляции и работы приложения. Второй файл, **Volume.jar**, находится в каталоге **D:\javabuilder_examples\Ball_Sph\Volume\distrib** и содержит классы созданного компонента.

Для подключения этих файлов есть две возможности. Во-первых, можно использовать диалоговое окно выбора путей (рис. 3.7.3), которое открывается на очередном шаге по созданию нового проекта. В этом диалоговом окне выберем

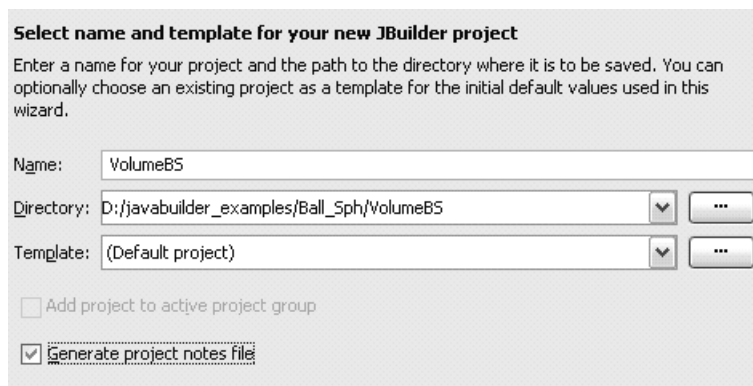


Рис. 3.7.2. Проект VolumeBS JBuilder

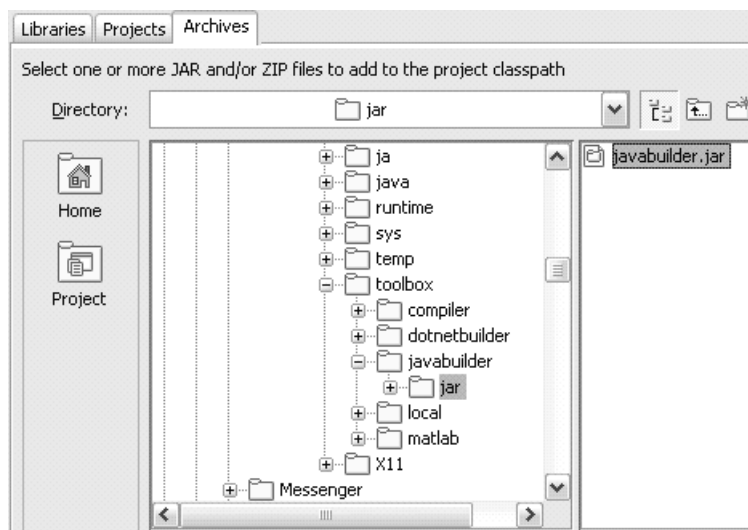


Рис. 3.7.3. Диалоговое окно подключения архивов

вкладку **RequiredLibraries**, нажимаем кнопку **Add**, тогда открывается диалоговое окно задания путей. В этом окне выберем вкладку **Archives** и укажем местонахождение файла `javabuilder.jar`, рис. 3.7.3. Аналогично подключаем архив `Volume.jar`. В результате основное поле вкладки **RequiredLibraries** будет содержать полные пути подключенных файлов.

Если это не сделано на начальном этапе создания нового проекта, то подключение можно провести используя свойства проекта, раздел меню **Project** ⇒ **ProjectProperties** ⇒ **RequiredLibraries** ⇒ **Archives**. В результате открывается то же окно задания путей и архивов (рис. 3.7.3 и 3.7.4).

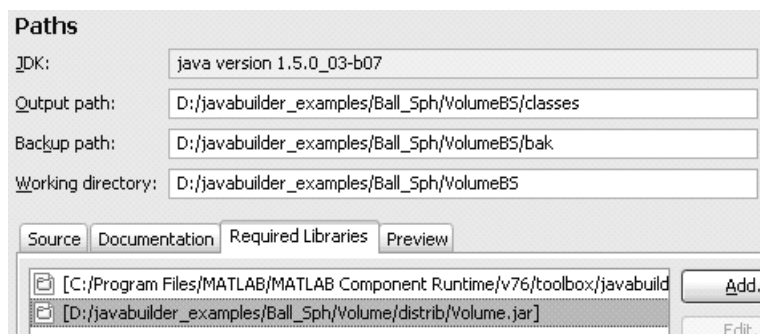


Рис. 3.7.4. Диалоговое окно задания путей

После прохождения всех этапов создания нового проекта создается большой набор файлов проекта. Создается окно **Frame1** проекта, исходный файл **Frame1.java** отображается в рабочей области **JBuilder**.

В этот исходный файл нужно записать две строки импорта классов. Раздел импорта находится в начале файла **Frame1.java**. После уже имеющихся строк импорта добавляем следующие:

```
import com.mathworks.toolbox.javabuilder.*;
import Volume.*;
```

Эта процедура завершает этап подключения классов **MWArray** и классов компонента. Для работы приложения нужно еще указать пути к библиотекам **MATLAB**:

```
C:\Program Files\MATLAB\MATLAB Component Runtime\v76\runtime\win32;
```

Однако, если библиотеки **MCR** установлены в **Windows XP**, то это делать не обязательно. Если мы запускаем проект в **JBuilder**, то переменные среды **Java** также можно не настраивать.

Создание окна приложения. Вкладка **Design** внизу рабочей области **JBuilder** открывает окно дизайнера, позволяющее задать элементы окна создаваемого приложения.

Параметр **layout** в свойствах окна **Frame1** управляет порядком расположения компонент на этом окне. Придадим ему значение **null**, тогда мы сможем самостоятельно устанавливать место и размеры компонент.

Определим на окне **Frame1** несколько информационных меток **jLabel** и три текстовых поля (рис. 3.7.5):

- **jTextField1** – для ввода размерности n . На вкладке **Events** установим событие для этого поля – **actionPerformed**, для того, чтобы после ввода числа и нажатия **ENTER** исполнить вычисление объема и площади;
- **jTextField2** – для вывода объема n -мерного шара;
- **jTextField3** – для вывода площади $(n-1)$ -мерной сферы.

Выделим более крупным и п/ж шрифтом информационный текст в метках **jLabel1** и **jLabel2**. Для того в свойствах метки выберем свойство **Font**. Тогда в пра-

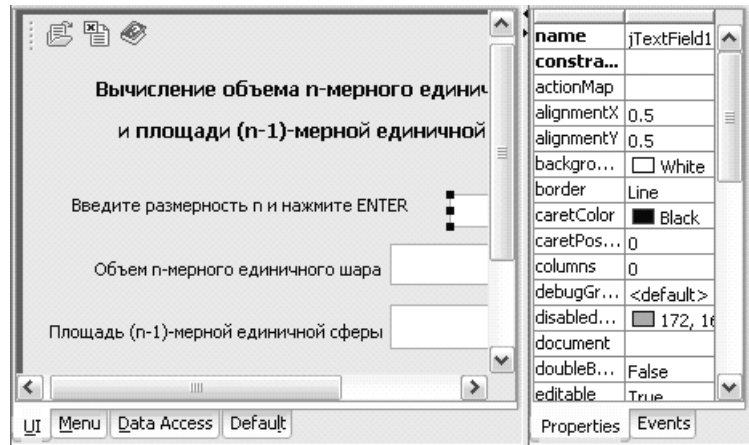


Рис. 3.7.5. Окно приложения

вой части свойства **Font** появляется кнопка, открывающая новое диалоговое окно выбора шрифта, рис. 3.7.6. Выберем шрифт: Tahoma, 12, Bold.

Создание программы приложения. Предполагается, что размерность n будет вводиться в текстовое поле `jTextField1`, а результаты будут выводиться в текстовые поля `jTextField2` и `jTextField3` после ввода n и нажатия клавиши **Enter**. Для того, чтобы реализовать эту программу, щелкнем 2 раза по полю `jTextField1`. В результате мы попадаем в раздел

```
public void jTextField1_actionPerformed(ActionEvent e) {
}
```

программы `Frame1.java`.

В это раздел внесем следующий код. Он содержит объявление данных, создание нового экземпляра `theVolume` класса `Volume`, вызов функций `Vol_n(1, m)` и `Sph_nmin1(1, m)` для вычисления объема шара и площади сферы соответственно. Результаты этих функций получаются в форме `Object[]`. Преобразуем этот тип в `MWNumericArray` и извлечем из него данные как тип `double` Java методом `getDouble(ind)`. В конце программы преобразуем числовые значения в `String` и отправим их в текстовые поля `jTextField2` и `jTextField2`.

```
public void jTextField1_actionPerformed(ActionEvent e)
{
    String an, bV, cS;                                // Значения текстовых строк
```

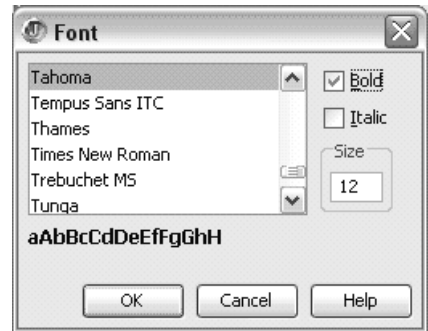


Рис. 3.7.6. Окно выбора шрифта

```

double n, V, S;                                // Входной параметр и результаты
an = jTextField1.getText();                    // Считывание с текстового поля
n = Double.parseDouble(an.trim()); // Преобразование в число

MWNumericArray m = null;    // Входное значение
Object[] result_V = null;   // Результат, объем шара
Object[] result_S = null;   // Результат, площадь сферы
MWNumericArray Vm = null;   // Результат в формате MWumericArray
MWNumericArray Sm = null;   // Результат в формате MWNumericArray

Volumeclass theVolumeclass = null; // Экземпляр класса Volumeclass
m = new MWNumericArray(n, MWClassID.DOUBLE);

try
{
    theVolumeclass = new Volumeclass();    // Экземпляр класса
    result_V = theVolumeclass.Vol_n(1, m); // Вычисление объема шара
    result_S = theVolumeclass.Sph_nmin1(1, m); // Вычисление площади

    Vm = new MWNumericArray(result_V[0], MWClassID.DOUBLE);
    Sm = new MWNumericArray(result_S[0], MWClassID.DOUBLE);
    V = Vm.getDouble(1);    // Выбор элемента 1 из результата
    S = Sm.getDouble(1);    // Выбор элемента 1 из результата

    bV = Double.toString(V); // Преобразование в строку
    cS = Double.toString(S); // Преобразование в строку
    jTextField2.setText(bV); // Запись в текстовое поле
    jTextField3.setText(cS); // Запись в текстовое поле
}
catch (Exception exception)
{
    exception.printStackTrace();
}
}
}

```

Нажимаем кнопку Run на инструментальной панели JBuilder и получаем приложение, показанное на рис. 3.7.7. Введем размерность $n = 21$ и нажмем ENTER. В результате будут вычислены объем единичного 21-мерного шара и площадь 20-мерной сферы, рис. 3.7.7. Классы этого приложения находятся в каталоге

D:\javabuilder_examples\Ball_Sph\VolumeBS\classes\volume_bs\,

а исходные файлы – в каталоге

D:\javabuilder_examples\Ball_Sph\VolumeBS\src\volume_bs\.

Создание пакета для распространения приложения

Для распространения приложения на другие машины необходимо создать jar-файл архива Java, содержащий основные файлы приложения. Для создания архива Java нужно воспользоваться пунктом меню **File** \Rightarrow **New**. В открывающемся

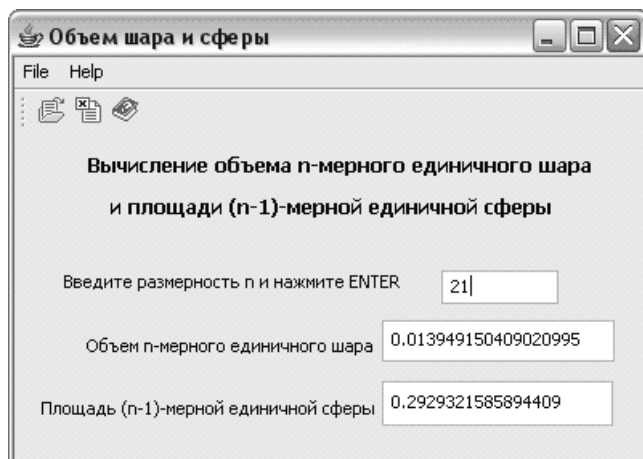


Рис. 3.7.7. Приложение для вычисления объема шара и площади сферы

диалоговом окне нужно выбрать раздел **Archive**, тип архива **Application** и нажать **OK** (рис. 3.7.8). В последующих диалоговых окнах Мастера построения имеется возможность задать параметры создания архива, мы выберем значения по умолчанию. В результате в проектной области окна JBuilder к проекту добавляется узел BallSphere, содержащий архив VolumeBS.jar. Для сохранения этого архивного файла нужно активизировать данный узел правой кнопкой мыши и выбрать **Make** (рис. 3.7.9). В результате файл VolumeBS.jar записывается в каталог проекта VolumeBS.

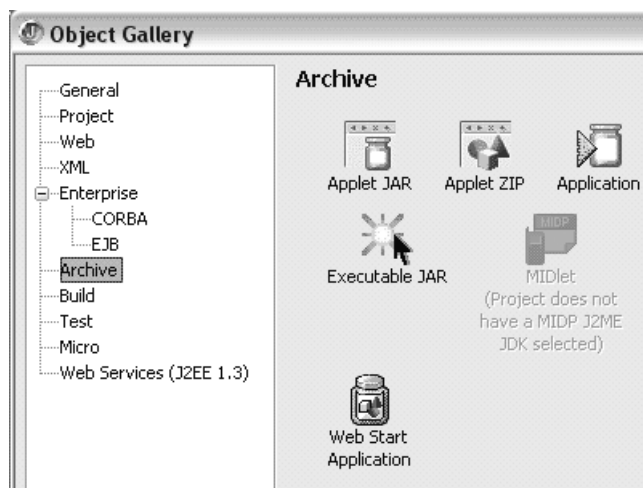


Рис. 3.7.8. Выбор типа архива

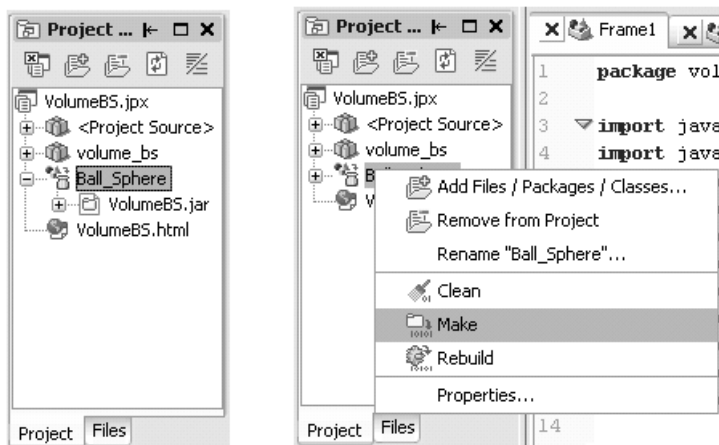


Рис. 3.7.9. Сохранение архива VolumeBS.jar

Для распространения приложения нужно еще включить файлы Volume.jar и Volume.ctf компонента Java Builder MATLAB и математические библиотеки MCR MATLAB.

Для работы приложения, кроме настройки среды Java, нужно установить среду исполнения MCR компонентов MATLAB, указать пути к файлам Volume.jar и Volume.ctf, к библиотекам MCR и к файлу javabuilder.jar из каталога <MCR>\v76\toolbox\javabuilder\jar.

3.7.2. Магический квадрат

В этом разделе рассмотрим создание приложения на JBuilder для вычисления магического квадрата размера n , вводимого в текстовое поле ввода. Магический квадрат выводится в виде таблицы. Напомним, что магический квадрат – это целочисленная матрица обладающая свойством: сумма элементов каждой строки, каждого столбца и главных диагоналей равны.

Компонент magic-square.jar, содержащий класс magic с методом makesqr для вычисления магического квадрата уже создан в параграфе 3.2. Поэтому мы не будем повторять здесь процедуру его построения. Отметим только, что каталог компонента D:\javabuilder_examples\magic_square\magic-square содержит два подкаталога **distrib** и **src**. Файл magic-square.jar находится в подкаталоге **distrib**. Подкаталог **src** содержит копии файлов magic-square.ctf и magic-square.jar, log-файлы регистрации процесса построения, а также файл созданный файл readme.txt, который содержит полезную информацию для распространения пакета. Кроме того, в подкаталоге **src\classes** содержатся созданные классы компонента, а в подкаталоге **src\magic-square** содержатся интерфейсные Java файлы.

Для создания приложения запустим сеанс JBuilder и создадим новый проект с именем Mag.jpx в каталоге D:\javabuilder_examples\ Mag.

Подключение классов MATLAB и созданного компонента *magicsquare*. Для создания и работы приложения нужно подключить к проекту файлы *javabuilder.jar* и *magicsquare.jar*. Первый файл *javabuilder.jar* находится в каталоге *C:\Program Files\MATLAB\MATLAB Component Runtime\v76\toolbox\javabuilder\jar* и содержит классы *MWArray* и подклассы, необходимые для компиляции и работы приложения. Второй файл *magicsquare.jar* находится в каталоге *D:\javabuilder_examples\magic_square\magicsquare\distrib* и содержит классы созданного компонента.

Для подключения этих файлов есть две возможности. Во-первых, можно использовать диалоговое окно выбора путей, которое открывается на очередном шаге по созданию нового проекта. В этом диалоговом окне выберем вкладку **RequiredLibraries**, нажать кнопку **Add**, откроется диалоговое окно задания путей. В этом окне выберем вкладку **Archives** и укажем местонахождение файла *javabuilder.jar*, рис. 3.7.3. Аналогично подключаем архив *magicsquare.jar*. В результате основное поле вкладки **RequiredLibraries** будет содержать полные пути подключенных файлов, рис. 3.7.10.

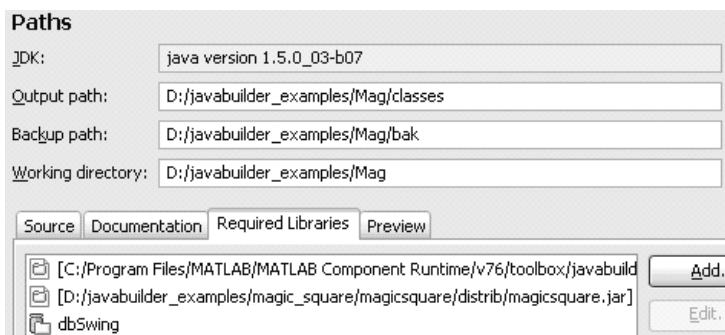


Рис. 3.7.10. Проект *Mag JBuilder*

Если это не сделано на начальном этапе создания нового проекта, то подключение можно провести используя свойства проекта, раздел меню **Project** ⇒ **ProjectProperties** ⇒ **RequiredLibraries** ⇒ **Archives**. В результате открывается то же окно задания путей и архивов (рис. 3.7.3 и 3.7.4).

После прохождения всех этапов организации нового проекта создается большой набор файлов проекта. Создается окно *Frame1* проекта, исходный файл *Frame1.java* отображается в рабочей области *JBuilder*.

В этот исходный файл нужно записать две строки импорта классов. Раздел импорта находится в начале файла *Frame1.java*. После уже имеющихся строк импорта добавляем следующие:

```
import com.mathworks.toolbox.javabuilder.*;
import magicsquare.*;
```

Эта процедура завершает этап подключения классов *MWArray* и классов компонента. Для работы приложения нужно еще указать пути к библиотекам *MATLAB*:

C:\Program Files\MATLAB\MATLAB Component Runtime\v76\runtime\win32;

Однако, если библиотеки MCR установлены в Windows XP, то это делать не обязательно. Если мы запускаем проект в JBuilder, то переменные среды Java также можно не настраивать.

Создание окна приложения. Вкладка **Design** внизу рабочей области JBuilder открывает окно дизайнера, позволяющее задать элементы окна создаваемого приложения.

Параметр **layont** в свойствах окна **Frame1** управляет порядком расположения компонент на этом окне. Придадим ему значение **null**, тогда мы сможем самостоятельно устанавливать место и размеры компонент.

Определим на окне **Frame1** метку **jLabel1** и текстовое поле **jTextField1** – для ввода размерности n магического квадрата, рис. 3.7.11. На вкладке Events установим событие для этого поля – **actionPerfomed**, для того, чтобы после ввода числа и нажатия ENTER исполнить вычисление магического квадрата.

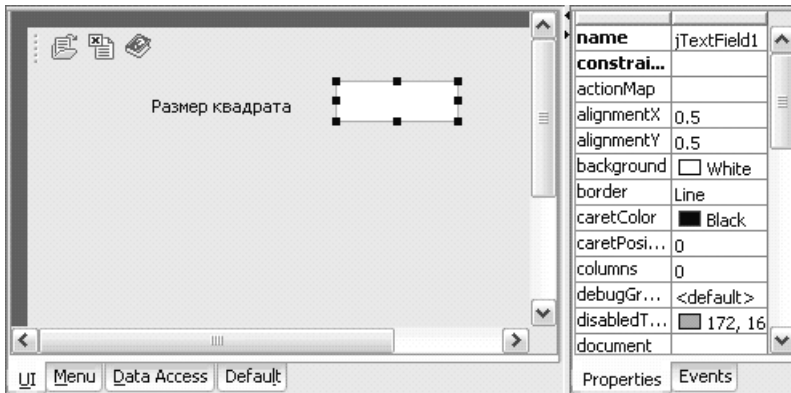


Рис. 3.7.11. Окно приложения

Создание программы приложения. Предполагается, что размерность n будет вводится в текстовое поле **jTextField1**, а результаты будут выводиться в таблицу **jTable1** после ввода n и нажатия клавиши **Enter**. Для того, чтобы реализовать эту программу, щелкнем 2 раза по полю **jTextField1**. В результате мы попадаем в раздел

```
public void jTextField1_actionPerformed(ActionEvent e)
{
}
```

программы **Frame1.java**.

В это раздел внесем следующий код. Он содержит объявление данных, создание нового экземпляра **theMagic** класса **magic**, вызов функции **makesqr** для вычисления магического квадрата по заданному размеру n матрицы. Результаты **result**

получаются в форме `Object[]`. Преобразуем первый элемент `result[0]` этого типа в `MWNumericArray` и извлечем из него данные как тип `double` Java методом `toArray()` с сохранением структуры массива. Мы хотим записать эти данные в таблицу `JTable`. Конструктор таблицы

```
JTable(dataTable, columnNames)
```

требует данных `dataTable` в формате `Object[][]` и заголовков столбцов `columnNames` в формате `String[]`. Поэтому из массива результатов типа `double` мы создаем массив `Object[][] dataTable`, содержащий целые элементы магического квадрата и создаем строку `String[] columnNames`. Аннотированный текст кода:

```
public void jTextField1_actionPerformed(ActionEvent e)
{
    String n; // Входное значение, строка
    double nd; // Входное значение, double
    int ni; // Входное значение, int

    Object[] result = null; // Результат, Object[]
    MWNumericArray res_mw = null; // Результат, MWNumericArray
    double[][] res_d = null; // Результат, double

    n = jTextField1.getText(); // Считывание с текстового поля
    nd = Double.parseDouble(n.trim()); // Преобразование в число double
    ni = Integer.parseInt(n.trim()); // Преобразование в число int

    magic theMagic = null; // Экземпляр класса magic

    try
    {
        theMagic = new magic(); // Экземпляр класса
        result = theMagic.makesqr(1, nd); // Вычисление маг. квадрата

        // Преобразование результата result[0] из Object[]
        // в массив MWNumericArray
        res_mw = new MWNumericArray(result[0], MWClassID.DOUBLE);
        res_d = (double[][])res_mw.toArray(); // Перевод в массив double
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
    }

    // Создание строки String[] заголовков столбцов
    String[] columnNames = new String[ni];
    for (int i = 0; i < ni; i++)
        columnNames[i] = " i ";
    //Создание массива данных типа Object[][]
    Object[][] dataTable = new Object[ni][ni];
    for (int i = 0; i < ni; i++)
        { for (int j = 0; j < ni; j++)
```

```
dataTable[i][j] = (int)res_d[i][j]; }
```

```
//Создание таблицы
```

```
JTable jTable1 = new JTable(dataTable, columnNames);
```

```
jTable1.setCellSelectionEnabled(true);
```

```
jTable1.setBounds(new Rectangle(28, 68, 337, 245));
```

```
contentPane.add(jTable1);
```

```
jTable1.setBorder(BorderFactory.createEtchedBorder());
```

```
jTable1.setGridColor(Color.lightGray);
```

```
}
```

Нажимаем кнопку **Run** на инструментальной панели JBuilder и получаем приложение, показанное на рис. 3.7.12, где вычислен магический квадрат для $n = 6$. Отметим, что таблица в окне появляется только после ввода значения n и нажатия Enter. Классы этого приложения находятся в каталоге

D:\javabuilder_examples\Mag\classes\mag\,

а исходные файлы – в каталоге D:\javabuilder_examples\Mag\src\mag\.



Рис. 3.7.12. Приложение для вычисления магического квадрата

MATLAB Builder для Excel

4.1. Введение	292
4.2. Общие вопросы создания компонент Excel Builder	305
4.3. Пример создания дополнения для спектрального анализа	308
4.4. Библиотека утилит Excel Builder	326
4.5. Справка по VBA	333

Как известно, Microsoft Excel является очень распространенной и очень удобной средой для матричных вычислений – листы Excel являются электронными таблицами, т.е. матрицами. С другой стороны, система MATLAB® является мощной математической матричной лабораторией. В системе MATLAB® имеется огромное количество различных математических функций для решения различных задач. Поэтому важно объединить удобства среды Excel с математической мощью MATLAB®. Такая возможность в MATLAB предусмотрена в пакете расширения MATLAB® Builder для Excel.

Пакет MATLAB® Builder для Excel (далее, Excel Builder) – это расширение к Компилятору MATLAB®, которое используется для создания надстроек (Add Ins) для Excel, которые позволяют выполнять функции MATLAB над данными рабочего листа Excel не обращаясь к MATLAB®. Excel Builder преобразовывает m-функции MATLAB в методы класса программной среды Visual Basic. Из этого класса, Excel Builder создает компонент, COM-объект, который доступен из Microsoft Excel. Созданный компонент можно использовать двумя способами: либо при помощи предлагаемой MATLAB надстройки «Мастер функций» для Excel, либо при помощи самостоятельно создаваемой надстройки (Add Ins) для Excel. Первый способ изложен в разделе 4.1, а второй – в разделе 4.3.

При распространении созданных при помощи Excel Builder компонентов в инсталляционный пакет нужно включать также среду исполнения компонентов MATLAB® (MCR), обеспечивающую выполнение математических функций MATLAB из Excel.

4.1. Введение

В данном параграфе будет рассмотрено краткое описание работы пакета MATLAB Builder для Excel. Будет представлена пошаговая процедура создания и упаковки компонента для Excel и процедура обращения к функциям компонента из Excel с использованием надстройки к Excel «Мастер функций» (mlfunction.xla), распространяемой с MATLAB.

Установка MATLAB Builder для Excel. Пакет MATLAB® Builder для Excel устанавливается обычным путем: при установке MATLAB нужно выбрать этот компонент вместе с MATLAB Compiler. Для работы Excel Builder необходимо также иметь установленный на системе внешний компилятор Microsoft Visual C/C++ (MSVC) версии 6.0, 7.1 или 8.0. Мы будем использовать компилятор Microsoft Visual C++ 2005 (8.0), входящий в Microsoft Visual Studio 2005.

Конфигурирование. Внешний компилятор Microsoft Visual C++ необходимо сконфигурировать для работы с Excel Builder. Для этого имеется утилита mbuild, которая вызывается из командной строки MATLAB,

```
mbuild -setup
```

При выполнении этой команды MATLAB определяет список всех имеющихся на системе компиляторов C/C++ и предлагает выбрать один из списка. Выбранный компилятор становится компилятором по умолчанию. Для замены компилятора нужно снова выполнить `mbuild -setup`.

Конфигурирование внешнего компилятора происходит автоматически при выполнении команды `mbuild -setup`. Для выбранного компилятора создается файл опций `compropts.bat`, который сохраняется в пользовательском (user profile) каталоге `C:\Documents and Settings\UserName\Application Data\MathWorks\MATLAB\R2007a`. Файл опций содержит параметры настройки и флаги, которые управляют работой внешнего C/C++ компилятора. Для создания файла опций система MATLAB имеет готовые сконфигурированные файлы опций, которые приведены ниже (они находятся в каталоге `<Matlab_root>\bin\win32\mbuildopts\`):

- `msvc60compp.bat` – для Microsoft Visual C/C++, Version 6.0;
- `msvc71compp.bat` – для Microsoft Visual C/C++, Version 7.1;
- `msvc80compp.bat` – для Microsoft Visual C/C++, Version 8.0.

Напомним, что при установке приложения на другую машину должна быть установлена среда выполнения MCR компонентов MATLAB и пути для библиотек, необходимых для работы созданного компонента. В частности, должен быть установлен путь `<mcr>\<ver>\runtime\win32`.

Графический интерфейс пользователя MATLAB Builder для Excel. Графический интерфейс разработки проектов Excel Builder предназначен для облегчения создания классов и компонент для Excel. Для его открытия достаточно выполнить следующую команду MATLAB:

`deploytool`

В результате открывается присоединяемое окно (рис. 4.1.1) справа от командного окна MATLAB и к строке меню MATLAB добавляется элемент меню **Project**. Это окно можно сделать и отдельным (рис 4.1.2).

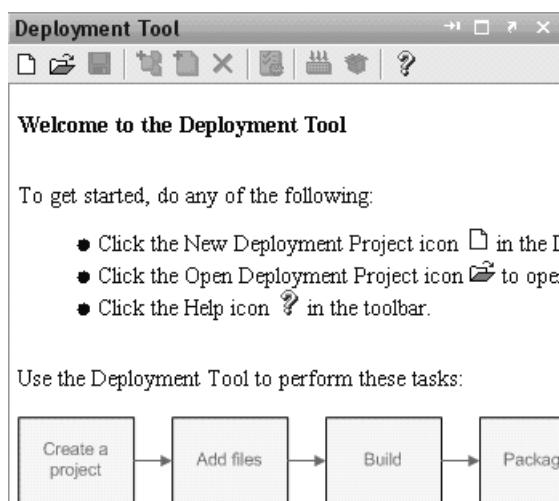


Рис. 4.1.1. Окно среды разработки Deployment Tool

4.1.1. Создание компонента для Excel

Для создания компонента нужно написать одну или несколько m-функций и создать проект в MATLAB Builder для Excel. Отметим, что имя m-функции не должно совпадать с именем встроенной m-функции MATLAB.

Рассмотрим процедуру создания компонента `matrix_xl` на простом примере создания библиотеки матричных функций MATLAB для их выполнения на листе Excel. Приведем пошаговую процедуру создания компонента.

Предполагается, что для Excel Builder установлен внешний компилятор Microsoft Visual C++ 2005 (8.0), входящий в Microsoft Visual Studio 2005.

1. Подготовка к созданию проекта. Выберем для проекта следующий каталог: `matlabroot\work\matrix_xl`. Затем составим m-функции, из которых будут создаваться класс и методы компонента. Создадим проект, содержащий несколько матричных операций: создание магического квадрата, умножение матриц, вычисление определителя, нахождение собственных чисел и матричной экспоненты. Большинство из этих функций являются встроенными функциями MATLAB. Поэтому для них нужно написать простые m-функции вызова. Например, для вычисления матричной экспоненты определяем следующую m-функцию:

```
function y = expxl(x)
y = expm(x);
```

Аналогично напишем и остальные функции. Назовем проект `matrix_xl`.

Устанавливаем в качестве текущего каталога MATLAB новый подкаталог проекта `matlabroot\work\matrix_xl`.

2. Создание нового проекта. Будем использовать графический интерфейс разработки. Он запускается из MATLAB следующей командой:

```
deploytool
```

Для создания нового проекта нужно сделать несколько простых действий:

- выбрать создание нового проекта, это можно сделать из меню **File** \Rightarrow **New Deployment Project**, или кнопкой **New Deployment Project** в инструментальной панели;
- в открывшейся навигационной области окна, выбрать **MATLAB Builder для Excel** и из списка компонентов, выбрать **Excel Add-in** как тип компонента, который предполагается создать, в нижней части окна напечатать имя проекта `matrix_xl.prj` (вместо `untitled1.prj`), проверить каталог, где создается проект и нажать **OK**. Проект содержит две папки: **matrix_xlclass** и **Other files**;
- добавить m-файлы выбранных функций в каталог `matrix_xlclass` проекта. Для этого нужно этот каталог активизировать и добавить в него файл либо используя меню **Project**, либо кнопку инструментальной панели, либо правую кнопку мыши;
- сохранить проект.

Опция **Setting** позволяет задать параметры настройки проекта. По умолчанию, имя проекта есть имя компонента. При создании нового проекта, диалоговое

окно Deployment Tool показывает папки, которые являются частью проекта. По умолчанию главная папка `matrix_xlclass` представляет класс, содержащий выбранные функции компонента. По умолчанию имя класса – то же самое, что и имя проекта. Имя класса можно изменить используя либо меню **Project** ⇒ **Rename Class**, либо используя правую кнопку мыши при активизированном основном каталоге (см. рис. 4.1.2). Сохраним проект.

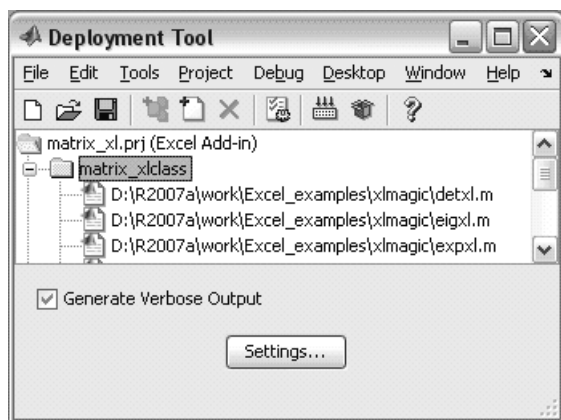


Рис. 4.1.2. Проект `matrix_xl`

3. Построение компонента. Для построения компонента, включающего выбранные функции, нужно исполнить команду **Build** из меню **Tools**, либо нажать кнопку построения (**Build**) на инструментальной панели Deployment Tool. Начинается процесс построения и создается log-файл регистрации процесса построения, в котором записываются все операции и ошибки при построении компонента. В случае успешного исхода в каталоге проекта создается подкаталог **matrix_xl**, который содержит два подкаталога **distrib** и **src**, в которые помещаются созданные файлы компонента.

В подкаталог **distrib** помещаются следующие созданные файлы, предназначенные для создания инсталляционного пакета:

- **matrix_xl_1_0.dll** – созданный компонент в виде динамически подключаемой библиотеки, содержащей откомпилированные функции проекта;
- **matrix_xl.bas** – код на VBA, необходимый для создания дополнения Excel (Add-in) из созданного компонента, содержит коды инициализации библиотек поддержки и коды вызова функций, входящих в компонент;
- **matrix_xl.ctf** – содержит зашифрованные m-функции, которые составляют компонент и все другие m-функции MATLAB, от которых зависят основные m-функции. Архив содержит все, основанное на MATLAB, содержание (m-файлы, MEX-файлы, и т.д.) связанное с компонентом.

В подкаталог **src** записываются копии файлов `matrix_xl.ctf`, `matrix_xl.bas` и `matrix_xl_1_0.dll`, log-файлы регистрации процесса построения, а также все

исходные файлы, соответствующие созданной библиотеке dll: def-файл lib-файл, exp-файл, заголовочные и файлы C/C++. Файл readme.txt содержит полезную информацию для распространения созданного компонента. Более подробно файлы каталога **src** обсуждаются в следующем параграфе.

4. Упаковка компонента. Для распространения созданного компонента удобно создать инсталляционный пакет. Это можно сделать командой **Package** из меню **Tools**, либо нажать кнопку упаковки (**Package**) на инструментальной панели Deployment Tool. В результате создается самораспаковывающийся архив **matrix_xl_pkg.exe**, содержащий файлы matrix_xl.ctf, matrix_xl.bas и matrix_xl_1_0.dll и файл _install.bat для запуска установки библиотек MCR MATLAB на машине, где устанавливается компонент и регистрации библиотеки matrix_xl_1_0.dll. Сам файл MCRInstaller.exe может быть также включен в инсталляционный пакет. Для этого нужно выбрать соответствующую опцию в **Setting**.

Созданная библиотека функций MATLAB для Excel может быть использована сразу на машине пользователя при условии, что установлено дополнение к Excel «Мастер функций». Его описание будет дано ниже. Если Мастер функций не предполагается использовать, то нужно для полученной библиотеки dll создать средствами VBA дополнение к Excel в виде файла типа *.xla. Это также будет рассмотрено ниже.

Полный пакет для распространения компонента содержит следующие файлы:

- componentname_projectversion.dll – скомпилированный компонент;
- componentname.ctf – архив CTF, технологический файл компоненты, он содержит зашифрованные m-функции и все другое содержание, связанное с компонентом;
- componentname.bas – код на VBA, необходимый для создания дополнения Excel (Add-in) из созданного компонента;
- componentname.xla – файл созданного дополнения к Excel, если такой файл создан;
- mlfunction.xla – файл дополнения к Excel, «Мастер функций»;
- MCRInstaller.exe – самораспаковывающаяся библиотека MCR (может быть включена в пакет);
- _install.bat – файл для регистрации компонента и вызова установки MCR.

Файлы дополнений componentname.xla и mlfunction.xla нужно перед упаковкой поместить в каталог projectdir\distrib (файл mlfunction.xla находится в каталоге C:\R2007a\toolbox\matlabxl\matlabxl). Далее, открыть проект в Deployment Tool и выполнить упаковку **Package** на панели инструментов. Файл _install.bat создается и упаковывается автоматически, автоматически также добавляется файл MCRRegCOMComponent.exe, необходимый для регистрации компонента по команде _install.bat.

Использование командной строки для построения компонента. Для построения компонента Excel Builder вместо Deployment Tool можно использовать команду mcs MATLAB. При этом каталоги project_directory\src и project_directory\distrib автоматически не создаются. Для создания этих каталогов и копирования ассоциированных файлов нужно использовать опцию -d команды mcs.

Общий синтаксис, для создания компоненты Excel Builder с msc следующий:

```
Msc -W 'excel: <component_name> [, <class_name> [, <major>. <minor>]]'
```

Здесь используется опция **W** для определения обертки **excel**. Нужно определить имя компоненты (<component_name>). Если не определяется название класса (<class_name>), то msc использует для этого имя компоненты как значение по умолчанию. Если не задается номер версии, то msc использует последнюю построенную версию или 1.0, если нет никакой предыдущей версии.

Пример. Использование команды msc, для создания COM-компонента, с именем mycomponent содержащего единственный класс, с именем myclass с методами foo и bar, версии 1.0. Опция **-T** указывает msc создать DLL.

```
msc -W 'excel:mycomponent, myclass, 1.0'-T link:lib foo.m bar.m
```

Для создания совместимой с Excel функции из каждого m-файла нужно определить опцию **-b** в командной строке следующим образом:

```
msc -W 'excel:mycomponent, myclass, 1.0'-b-T link:lib foo.m bar.m
```

В качестве альтернативы можно также использовать файл группы cexcel, чтобы упростить командную строку:

```
msc -B 'cexcel:mycomponent, myclass, 1.0' foo.m bar.m
```

4.1.2. Установка компонента на другие машины

При создании инсталляционного пакета, можно включить в него среду исполнения MCR компонентов MATLAB. Тогда при распаковке инсталляционного пакета одновременно устанавливается и MCR. Если MCR не включена в инсталляционный пакет, то она должна быть установлена отдельно до установки компонента. После установки MCR желательно сделать перезагрузку. Установка разработанного компонента на машину пользователя проводится в три этапа.

1. Установка файлов компонента. Задать каталог приложения. Скопировать в него инсталляционный файл `matrix_xl_pkg.exe`, содержащий пакет программ созданного компонента. Запустить самораспаковывающийся архив `matrix_xl_pkg.exe`. В результате будут получены файлы `matrix_xl_1_0.dll`, `matrix_xl.ctf`, `matrix_xl.bas`, `_install.bat` и, возможно, файлы `MCRInstaller.exe` и `mlfunction.xla`.

2. Регистрация библиотек. Зарегистрировать в командной строке DOS библиотеки `matrix_xl_1_0.dll` и `mwcomutil.dll`. Регистрация библиотеки `matrix_xl_1_0.dll` производится с помощью файла `_install.bat`. Регистрация библиотеки `mwcomutil.dll` производится при установке MCR.

Регистрацию можно провести также «вручную» с помощью команды `mwregsvr`, которая находится в каталоге `MATLAB Component Runtime\76\bin\win32`. Библиотека `matrix_xl_1_0.dll` находится в каталоге приложения, для ее регистрации нужно указывать полный путь. Библиотека `mwcomutil.dll` находится в каталоге `MATLAB Component Runtime\76\runtime\win32` и для ее регистрации указывать полный путь не нужно. Процедура регистрации:

- перейти в папку, находящуюся по адресу MATLAB Component Runtime\v76\bin\win32, где расположен файл mwregsvr;
- для регистрации mwcomutil.dll – ввести команду:

```
>mwregsvr mwcomutil.dll.
```
- для регистрации компонента matrix_xl_1_0.dll необходимо ввести команду из каталога MATLAB Component Runtime\v74\bin\win32, указывая полный путь к компоненту:

```
>mwregsvr <project_dir>\matrix_xl_1_0.dll
```

например,

```
>mwregsvr.exe  
C:\R2007a\work\Excel_examples\matrix_xl\matrix_xl\  
distrib\matrix_xl_1_0.dll
```

Отметим, что команда mwregsvr.exe не обрабатывает каталоги и файлы с именами на кириллице.

3. Подключение компонента к Excel. Когда имеется файл mlfunction.xla Мастера функций, то достаточно подключить только его. Для этого нужно выполнить следующее:

- запустить Excel;
- из главного меню Excel выбрать **Сервис** ⇒ **Настройки**;
- в открывшемся диалоговом окне выбрать **Обзор** и перейти в каталог приложения (куда был распакован архив matrix_xl_pkg.exe);
- выбрать mlfunction.xla и нажать **ОК** в обоих окнах.

В списке надстроек должно появиться название надстройки **MATLAB Function**. Само название компонента в списке надстроек не появляется. Работа с компонентом производится через надстройку MATLAB Function. Для использования установленного дополнения к Excel, нужно запустить Мастер функций: выбрать **Сервис** ⇒ **MATLAB Function**. В открывшемся диалоговом окне Мастера функций выбрать необходимые функции компонента и пользоваться ими в текущей сессии Excel.

4.1.3. Мастер функций

Мастер функций обеспечивает удобный интерфейс для управления функциями компонент, созданных при помощи MATLAB Builder для Excel. Мастер функций дает возможность выбрать необходимые функции компонент, задать параметры ввода и вывода, исполнить функцию и сделать ряд других операций над функциями. Мастер функций дает возможность передавать значения рабочего листа Microsoft Excel (Excel 2000 или позже) в функцию, созданного при помощи MATLAB компонента, и возвращать вывод в ячейку или диапазон ячеек на рабочем листе.

Отметим сразу, что Мастер функций в настоящее время не поддерживает следующие типы данных MATLAB: структуры, разреженные и комплексные массивы. Отметим также, что массивы ввода и вывода функции компонента являются обычными данными Excel и могут свободно копироваться в Excel, при этом связь

между этими массивами теряется. Массив вывода одной функции может служить вводом следующей функции, так, что можно составлять композиции функций.

Установка мастера функций. Графический интерфейс Мастера функций содержится в дополнении Excel (mlfunction.xla), которое находится в каталоге <matlab>\toolbox\matlabxl\matlabxl. Нужно установить это дополнение перед использованием мастера функций. Для этого нужно сделать следующее:

- выбрать **Сервис** ⇒ **Надстройки** из главного меню Excel;
- если мастер функций был предварительно установлен, то ссылка на мастер функций MATLAB появляется в списке. Выберите пункт списка и нажмите **ОК**. Если мастер функций не был предварительно установлен, выберите **Browse**, перейдите в каталог <matlab>\toolbox\matlabxl\matlabxl и выберите **mlfunction.xla**. Нажмите **ОК** на этом диалоговом окне и на предыдущем.

Мастер функций также упаковывается со всеми распространяемыми компонентами. Когда компонент устанавливается на другую машину, мастер функций mlfunction.xla размещается в каталоге верхнего уровня установленного компонента. Установка производится аналогично.

Запуск мастера функций. Чтобы запустить мастер функций, нужно выбрать MATLAB Function в меню **Сервис** Excel. Появляется начальное окно мастера функций, которое выглядит как на рис. 4.1.3.

Основное поле мастера функций отображает названия всех загруженных ранее функций. При первом запуске это поле – пустое. Флажок для каждой функции обозначает активное/неактивное состояние функции.

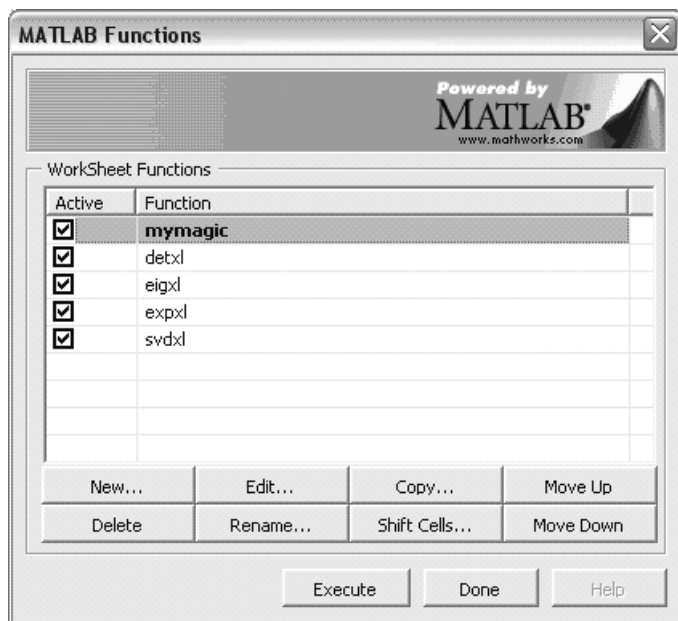


Рис. 4.1.3. Начальное окно мастера функций

Ниже списка функций – группа из десяти кнопок. Рассмотрим их немного подробнее.

Кнопка New. Она открывает список всех установленных компонент и их функций. Выбираем компонент и набор тех его функций, которые предполагается использовать. Для выбора функций используются кнопки **Add** и **Remove** (рис. 4.1.4).

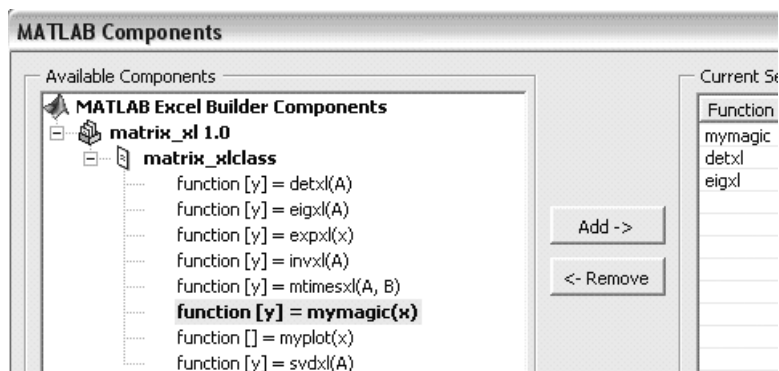


Рис. 4.1.4. Выбор компонента и функций

После выбора нажимаем **Ok** и открывается окно мастера функций с выбранными функциями (рис. 4.1.3). Функцию, которую предполагается использовать, нужно выделить мышкой, или стрелками «вверх» и «вниз» на клавиатуре. При этом флажки остальных функций отключать не требуется.

Кнопка Edit. Открывает диалоговое окно **Function Properties** (рис. 4.1.5) в котором можно установить параметры ввода, вывода и другие свойства функции.

В верхнем окне отображается синтаксис функции. Он необходим для того, чтобы пользователь мог видеть, какие аргументы требует функция и что она выводит. Далее идет строка, позволяющая изменить имя функции. Нижняя часть окна используется для установки параметров и других свойств функции при помощи целой группы диалоговых окон.

Для определения входных и выходных аргументов нужно выбрать вкладку **Inputs** или **Outputs**. Тогда в окне отображается список всех аргументов функции. Нужно выделить аргумент, нажать на кнопку **Properties** и установить или числовые значения (**Value**), или диапазоны (**Range**) (рис. 4.1.6).

При этом можно использовать опцию «Auto recalculate on change». Тогда при изменении аргументов на листе Excel, значения функции будут сразу пересчитываться.

Кнопка **Options** позволяет установить формат выбранного аргумента (рис. 4.1.7).

Если в списке аргументов имеются аргументы типа **varargin** или **varargout**, то при их выборе кнопка **Properties** окна **Function Properties** становится пассивной,

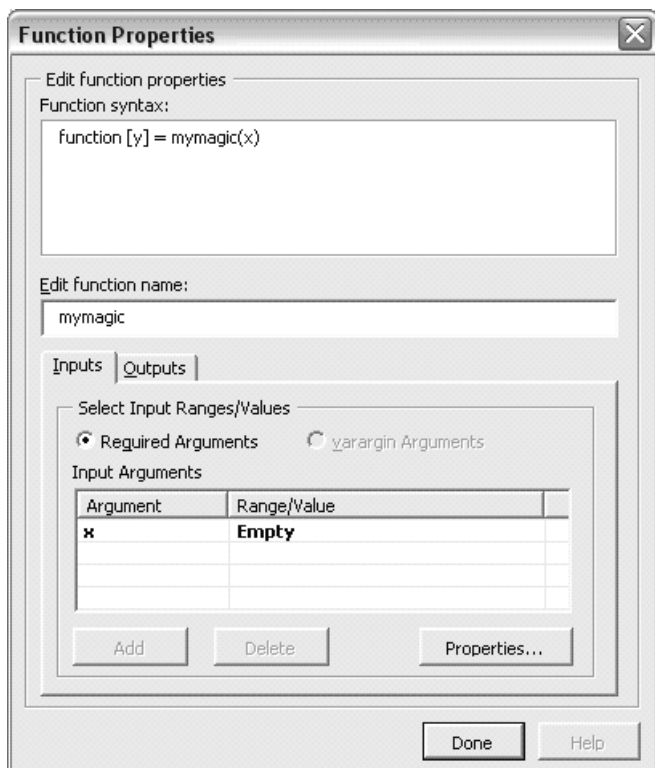


Рис. 4.1.5. Задание диапазонов аргументов и других свойств

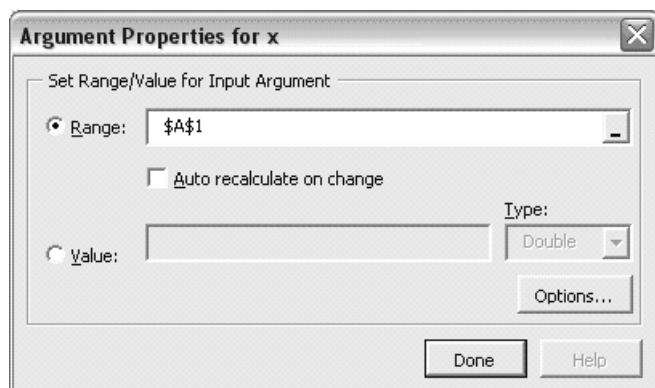


Рис. 4.1.6. Диалоговое окно свойств параметров ввода

но активизируются кнопки **Add** и **Delete**. Эти кнопки используются для добавления необходимого числа аргументов (напомним, что аргументы типа `varargin`/

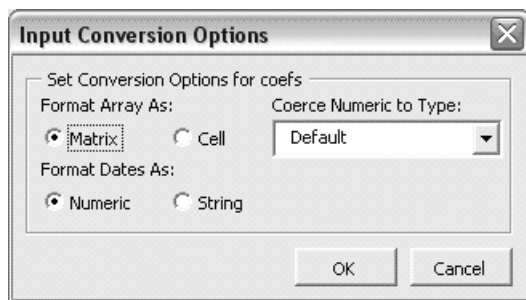


Рис. 4.1.7. Диалоговое окно определения формата параметров ввода/вывода

varargout позволяют ввести неопределенное заранее число аргументов). После добавления необходимого числа параметров varargin/varargout, их можно редактировать таким же образом, как и обязательные параметры.

Диалоговое окно параметров вывода несколько отличается (рис 4.1.8).

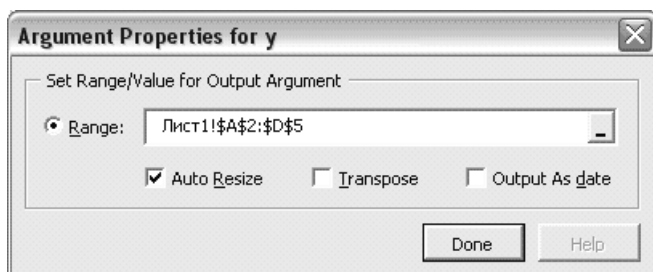


Рис. 4.1.8. Диалоговое окно свойств параметров вывода

Это диалоговое окно свойств параметра вывода позволяет выбрать:

- диапазон (Range);
- автоподбор (Auto resize) размера вывода, когда предполагаемый вывод от вызываемого метода есть диапазон ячеек в рабочем листе Excel, но размер массива вывода и его форма неизвестны во время вызова функции;
- транспонировать (Transpose output) массив параметров вывода;
- принудить (Output as date) значения вывода стать данными Excel.

Кнопка Rename. Вызывает диалоговое окно **Rename Function** для переименования функции.

Кнопка Copy. Вызывает диалоговое окно для создания копии текущей функции. По умолчанию, копии функции имеют те же значения параметров, что и исходная функция. Однако значения параметров копии могут быть изменены и тогда копии функции могут быть использованы для применения данной функции к нескольким разным массивам на рабочем листе Excel. Вкладка **Advanced** создает прямоугольный массив копий текущей функции на рабочем листе Excel. При-

ращения строк или столбцов указывают, с каким шагом будут располагаться диапазоны ввода и вывода копий по отношению к диапазонам ввода и вывода оригинальной функции на рабочем листе Excel.

Кнопки Delete, Move Up и Move Down. Они служат для удаления некоторых функций из списка в окне мастера функций, для перемещения выделенной функции вверх, или вниз.

Кнопка Shift Cells. Вызывает диалоговое окно **Move Function**, чтобы переместить выбранную функцию в новую позицию на рабочем листе Excel (т.е. на рабочем листе Excel перемещаются диапазоны ввода и вывода выбранной функции). Это важно потому, что копирование Excel данных диапазонов ввода и вывода выбранной функции не сохраняет зависимости между аргументами ввода и значениями вывода.

Кнопка Execute. Выполняет выбранную функцию.

Кнопка Done. Подтверждает сделанный выбор.

4.1.4. Работа с компонентами в Excel

Продemonстрируем работу дополнения к Excel на примере работы компонента `matrix_xl`.

Подключение компонента к Excel. Предположим, что библиотеки MCR установлены и компонент инсталлирован в некоторый каталог, например, в `c:\matrix` и зарегистрирован на системе. Предполагается, что в этом же каталоге находится файл `mlfunction.xla` мастера функций. Для подключения компонента, необходимо запустить Excel, выбрать **Сервис** \Rightarrow **Настройка** и в появившемся диалоговом окне, нажав **Обзор**, перейти в папку `c:\matrix`. Далее выбрать `mlfunction` и нажать **ОК**. Таким образом, мы подключаем компонент через Мастер функций.

Работа с дополнением к Excel. Далее выбираем **Сервис** \Rightarrow **MATLAB Function**. Появляется окно Мастера функций. Нажимаем кнопку **New** и выбираем компонент `matrix_xl_1.0.>matrix_xlclass` и несколько функций (рис 4.1.4). Из предложенного списка можно выбрать необходимые функции и нажать **Add.**, затем **ОК**, получаем список выбранных функций, рис. 4.1.3.

Выбираем функцию **mymagic** для вычисления магического квадрата. Напомним, что магический квадрат есть квадратная матрица, которая имеет одинаковые суммы элементов строк, столбцов и диагоналей. Далее нажимаем **Edit, Properties** и определяем положение ячейки Excel A1 в которой будет находиться порядок магического квадрата и положение левого верхнего угла A2 для выходного массива Excel магического квадрата. При этом в диалоговом окне выбора аргумента выбираем опцию «Auto recalculate on change», тогда при изменении порядка квадрата будет автоматически пересчитываться выводимый квадрат. Кроме того выбираем опцию автоматического выбора размеров диапазона вывода (тогда достаточно задать только левый верхний угол). После выбора входных и выходных массивов нажимаем кнопку **Done**. В Excel введем в ячейку A1 значение 4 и нажмем кнопку **Execute** мастера функций. Выводится ответ (рис.4.1.9), следующий магический квадрат:

	A	B	C	D	E
1	4				
2	16	2	3	13	
3	5	11	10	8	
4	9	7	6	12	
5	4	14	15	1	
6					

Рис. 4.1.9. Результат выполнения функции

Найдем определитель полученного магического квадрата. Для этого активизируем функцию **detxl** и в качестве матрицы ввода назначим полученный магический квадрат A2 : D5. Для вывода возьмем ячейку F1.

Найдем также собственные числа полученного магического квадрата. Для этого активизируем функцию **eigxl** и в качестве матрицы ввода назначим полученный магический квадрат. Для вывода возьмем столбец F2 : F5. Получаем следующий результат (рис. 4.1.10)

	A	B	C	D	E	F
1	4					0
2	16	2	3	13		34
3	5	11	10	8		8,944272
4	9	7	6	12		-8,94427
5	4	14	15	1		6,67E-17
6						

Рис. 4.1.10. Результат выполнения нескольких функции

Отметим, что вывод одной функции (**mymagic**) может быть входом других функций (**detxl**, **eigxl**). При изменении аргумента A1 первой функции (**mymagic**) меняется ее выходные данные A2 : D5 и, соответственно, меняются выходные данные F1 : F5 следующих функций (**detxl**, **eigxl**). Это означает, что на листе Excel можно составлять композиции функций. Заметим также, что выходные значения – это обычные числовые данные Excel, они не являются связанными с входными данными в обычном смысле Excel. Выходные значения допускают редактирование, копирование и все остальные операции Excel.

Таким образом, мы видим, что практически любая функция MATLAB, или написанный на m-языке алгоритм может быть легко скомпилирован и выполнен из Excel, не вызывая среду MATLAB. Столь же просто компилируются и функции MATLAB построения графиков. А интеграция созданных функций с возможностями среды Excel показывает потрясающую эффективность пакета Excel Builder.

4.2. Общие вопросы создания компонент Excel Builder

Каждый компонент Excel Builder создается как автономный COM-объект. Каждая функция MATLAB, включенная в данный компонент является методом созданного COM-класса. При работе компонента синтаксис VBA систематически преобразуется в синтаксис MATLAB для выполнения функций MATLAB.

Процесс создания компонента Excel Builder является полностью автоматическим с точки зрения пользователя. Достаточно определить список m-файлов для обработки и некоторую дополнительную информацию, такую как имя компонента, имена классов, и номера версии. Процесс построения компонента включает: генерацию кода, создание описания интерфейса, компиляцию C++, компоновку и связывание ресурсов, регистрацию компонента.

4.2.1. Процедура создания компонента

Рассмотрим подробнее эти этапы построения компонента.

Генерация кода. Первый шаг в процессе построения генерирует все *исходные коды* и другие файлы, необходимые для создания компонента (пусть он имеет имя *mycomponent*). Создается главный исходный файл *mycomponent_dll.cpp*, содержащий описание процедур инициализации MCR, WINAPI DllMain и регистрации компонента. Компилятор дополнительно производит IDL-файл *mycomponent_idl.idl* языка описания интерфейса (Interface Definition Language), содержащий спецификации для библиотеки компонента, интерфейса и класса с соответствующим GUID (GUID – глобально уникальный идентификатор, присваиваемый объекту регистрации в системном реестре Windows, это – 128-битовое целое число гарантированно уникальное).

Затем создаются C++ файлы описания класса и выполнения *myclass_com.hpp* и *myclass_com.cpp*, содержащие описание каждой экспортируемой функции. В дополнение к этим исходным файлам, компилятор генерирует файл экспорта DLL *mycomponent.def*, описание ресурсов *mycomponent.rc*, и технологический файл компонента *mycomponent.ctf*.

Создание описания интерфейса. Второй шаг процесса построения вызывает IDL-компилятор для IDL-файла *mycomponent_idl.idl*, созданного на первом шаге, создает заголовочный файл *mycomponent_idl.h* интерфейса, файл интерфейса GUID *mycomponent_idl_i.c*, и файл *mycomponent_idl.tlb* типа библиотеки компонента. Заголовочный файл содержит описание типов и объявление функций, основанное на определении интерфейса в IDL-файле. Интерфейс файла GUID содержит описание GUIDs из всех интерфейсов в IDL файле. Файл типа библиотеки компонента содержит бинарное представление всех типов и объектов, представленных в компоненте.

С++ компиляция. На третьем шаге компилируются все С/С++ исходные файлы, созданные первым и вторым этапах, в объектный код. При этом возникает один дополнительный файл (mclcomclass.h), содержащий ряд шаблонов С++ классов. Этот файл содержит реализации шаблонов всех необходимых СОМ классов, а также кодов регистрации и обработки ошибок.

Компоновка и связывание ресурсов. Четвертый шаг производит окончательную DLL для компонента. Этот шаг вызывает линковщик на объектные файлы, созданные на третьем этапе и необходимые библиотекам MATLAB для создания компонента DLL mycomponent_1_0.dll.

Регистрация компонента. Заключительный шаг – регистрация созданной DLL на системе, как описано ниже.

4.2.2. Регистрация компонента

Когда Excel Builder создает компонент, он автоматически генерирует бинарный файл (mycomponent_idl.tlb), называемый типом библиотеки. На заключительном этапе построения, этот файл связывается с законченным DLL как ресурс.

Получение информации о регистрации. Программируя с СОМ-компонентами иногда нужна дополнительная информация о компоненте. Можно использовать команду componentinfo, которая является функцией MATLAB, чтобы сделать запрос системной регистрации для деталей о любом установленном компоненте Excel Builder.

Пример. Запрос о регистрации компонента, названного mycomponent и версии 1.0. Этот компонент имеет четыре метода: mysum, randvectors, getdates, и myprimes, два свойства m и n, и одно событие myevent. Возвращенная структура содержит поля, соответствующие самой важной информации о регистрации и типе библиотеки для компонента (ниже приведена часть кода).

```
Info = componentinfo('mycomponent', 1, 0)

Info =
    Name: 'mycomponent'
    TypeLib: 'mycomponent 1.0 Type Library'
        LIBID: '{3A14AB34-44BE-11D5-B155-00D0B7BA7544}'
    MajorRev: 1
    MinorRev: 0
    FileName: 'D:\Work\ mycomponent\distrib\mycomponent_1_0.dll'
    Interfaces: [1x1 struct]
    CoClasses: [1x1 struct]

Info.Interfaces

ans =
    Name: 'Imyclass'
```



```
IID: '{3A14AB36-44BE-11D5-B155-00D0B7BA7544}'
```

```
Info.CoClasses
```

```
ans =
```

```
    Name: 'myclass'
```

```
    CLSID: '{3A14AB35-44BE-11D5-B155-00D0B7BA7544}'
```

```
    ProgID: 'mycomponent.myclass.1_0'
```

```
    VerIndProgID: 'mycomponent.myclass'
```

```
InprocServer32: 'D:\Work\mycomponent\distrib\mycomponent_1_0.dll'
```

```
    Methods: [1x4 struct]
```

```
    Properties: {'m', 'n'}
```

```
    Events: [1x1 struct]
```

Саморегистрирующиеся компоненты. Компоненты Excel Builder – все саморегистрирующиеся. Такой компонент содержит весь необходимый код, чтобы добавить или удалить полное описание о себе или о системной регистрации. Утилита `mwregsvr`, расположенная в MCR, регистрирует библиотеки `dll`. Например, чтобы *зарегистрировать* компонент по имени `mycomponent_1_0.dll`, достаточно запустить следующую команду в командной строке DOS.

```
mwregsvr mycomponent_1_0.dll
```

Команда

```
mwregsvr/u mycomponent_1_0.dll
```

удаляет регистрацию компонента.

Компонент Excel Builder, установленный на другую машину должен быть зарегистрирован утилитой `mwregsvr`. Если компонент перемещается в другой каталог на той же самой машине, то нужно повторить процесс регистрации. При удалении компонента, необходимо сначала удалить его регистрацию, чтобы не оставить ошибочной информацию о регистрации.

Замечание. Утилита `mwregsvr` работает аналогично утилите Windows `regsvr32.exe`. Последняя утилита `regsvr32.exe` также может использоваться для регистрации вашей библиотеки.

Глобально уникальный идентификатор (GUID). Информация сохраняется в реестре в виде ключей с одним или более соответствующими значениями. Сами ключи имеют значения, изначально, двух типов: читаемые строки и GUID-ы (128-битовое уникально целое число). Excel Builder автоматически генерирует GUID-ы для COM-классов, интерфейсов и типов библиотек, которые определены с компонентом, а также время построения и кодирует эти ключи в саморегистрационный код компонента.

Описание ключей имеется в документации MATLAB Builder for Excel.

4.2.3. Разработка новых версий

Компоненты MATLAB Builder for Excel поддерживают простой механизм управления версиями, созданный для облегчения создания и распространения обновленных версий одного и того же компонента. Номер версии компонента

является частью имени DLL, так же как часть зависимого от версии ID в системной регистрации.

Номер версии можно определить при создании компонента (значение по умолчанию = 1.0). В течение разработки определенной версии компонента, номер версии должен сохраняться. Когда это условие выполняется, для каждого последующего построения компонента компилятор MATLAB многократно использует библиотеку, класс и интерфейс GUIDs. Он избегает создания чрезмерного числа ключей регистрации для того же самого компонента в течение многократного построения.

Когда введен новый номер версии, компилятор MATLAB генерирует новый класс и интерфейс GUIDs так, чтобы система отличала их от предыдущих версий, даже если имя класса – то же самое.

Поэтому, когда Вы распространяете построенный компонент, то используйте новый номер версии при любых сделанных в компоненте изменениях. Это гарантирует простое управление этими двумя версиями после распространения нового компонента.

Если компилятор MATLAB находит существующий компонент с другой версией, он использует существующую библиотеку GUID и создает новый подключ для нового номера версии. Он генерирует новый GUIDs для нового класса и интерфейса. Если компилятор не находит существующий компонент с указанным именем, он генерирует новый GUID для библиотеки компонента, класса и интерфейса.

4.3. Пример создания дополнения для спектрального анализа

Рассмотрим (следуя документации Excel Builder) создание дополнения Excel для выполнения спектрального анализа без использования Мастера функций. Будет создана своя форма для данного дополнения и средствами VBA будет произведено подключение к Excel. Данное дополнение к Excel выполняет быстрое преобразование Фурье (FFT) на входном наборе данных рабочего листа Excel и возвращает результаты FFT: массив точек частот и частотный спектр входного сигнала. Эти результаты записываются в указываемые диапазоны рабочего листа. Можно также вывести график мощности спектра. Создание дополнения состоит из четырех основных этапов:

1. Построение автономного COM-компонента из кода MATLAB.
2. Составление необходимого кода VBA для интеграции с Excel.
3. Создание графического интерфейса пользователя компонента.
4. Создание дополнения Excel и упаковка всех необходимых компонентов для распространения приложения.

4.3.1. Построение компонента

Компонент будет иметь один класс с двумя методами, `computefft` и `plotfft`. Метод `computefft` вычисляет дискретное преобразование Фурье FFT, спектр данных и вычисляет вектор точек частоты, в соответствии с длиной введенных данных

и шагом выборки. Метод `plotfft` выполняет те же самые операции как `computefft`, но также и строит график входных данных и спектра в стандартном окне Figure MATLAB. Код MATLAB для этих двух методов находится в двух m-файлах, `computefft.m` и `plotfft.m` (которые можно найти в каталоге `matlabroot\toolbox\javabuilder\Examples\SpectraExample\SpectraDemoComp\`). Первый m-файл вычисляет быстрое преобразование Фурье используя функцию `y = fft(x)` MATLAB.

Напомним, что *дискретным преобразованием Фурье* сигнала $\{x_n\}$ конечной длины N называется сигнал $\{y_n\}$, полученный по формуле:

$$y_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi}{N} (k-1)(n-1)}, \quad k=1, 2, \dots, N.$$

Здесь предполагается, что сигнал $\{x_n\}$ получен оцифровкой функции $f(t)$ по формуле $x_n = f((n-1)\Delta t)$, $n=1, 2, \dots, N$. Параметр t меняется на промежутке $[0, N-1]$, Δt – шаг дискретизации. Величина $F = 1/\Delta t$ называется частотой дискретизации – число отсчетов в единицу времени. Тогда частоты сигнала могут принимать значения от 0 до F . Поэтому массив частот сигнала $\{x_n\}$ вычисляются по формуле $\omega_k = Fk/N$, $k=0, 1, \dots, N-1$. Еще одной характеристикой сигнала является спектр (или мощность спектра), который вычисляется по формуле

$$P_k = \frac{|y_k|}{\sqrt{N}}, \quad k=1, 2, \dots, N.$$

Приведем код функции `computefft.m`, которая имеет входные аргументы:

- `data` – входной сигнал $\{x_n\}$;
- `interval` – шаг дискретизации Δt

и выходные параметры:

- `fftdata` – выходной сигнал $\{y_n\}$;
- `freq` – массив частот;
- `powerspect` – мощность спектра.

```
function [fftdata, freq, powerspect] = computefft(data, interval)
    if (isempty(data))
        fftdata = [];
        freq = [];
        powerspect = [];
        return;
    end
    if (interval <= 0)
        error('Sampling interval must be greater then zero');
        return;
    end
    fftdata = fft(data);
    freq = (0:length(fftdata)-1)/(length(fftdata)*interval);
    powerspect = abs(fftdata)/(sqrt(length(fftdata)));
```

Функция `plotfft.m` обращается к функции `computefft` для спектрального разложения и выводит графики сигнала $\{x_n\}$ и частотного спектра P_k до частоты Найквиста $F/2$. Код функции `plotfft.m`:

```
function [fftdata, freq, powerspect] = plotfft(data, interval)
[fftdata, freq, powerspect] = computefft(data, interval);
len = length(fftdata);
if (len <= 0)
    return;
end
t = 0:interval:(len-1)*interval;
subplot(2,1,1), plot(t, data)
xlabel('Time'), grid on
title('Time domain signal')
subplot(2,1,2), plot(freq(1:len/2), powerspect(1:len/2))
xlabel('Frequency (Hz)'), grid on
title('Power spectral density')
```

Создание проекта Fourier. Выбранные m-функции поместим в каталог matlabroot\work\Excel_examples\SpectraExample. Напомним, что для Excel Builder должен быть установлен внешний компилятор Microsoft Visual C++. Мы будем использовать Microsoft Visual C++ 2005 (8.0), входящий в Microsoft Visual Studio 2005. Устанавливаем в качестве текущего каталога MATLAB новый подкаталог проекта matlabroot\work\Excel_examples\SpectraExample.

Будем использовать графический интерфейс разработки. Он запускается из MATLAB следующей командой:

```
deploytool
```

Создаем проект по имени Fourier и с именем класса Fourier. Добавляем к проекту m-файлы computefft.m и plotfft.m. Сохраняем проект. Строим компонент, нажимая кнопку Build на инструментальной панели Deployment Tool (рис. 4.3.1).

В процессе построения и создается log-файл регистрации процесса построения, в котором записываются все операции и ошибки при построении компонента.

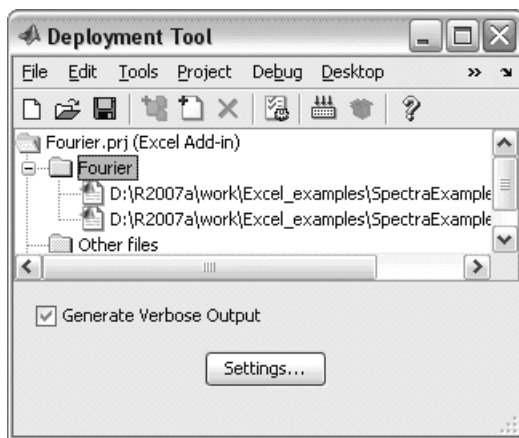


Рис. 4.3.1. Проект Fourier

В случае успешного исхода в каталоге проекта создается подкаталог **Fourier**, который содержит два подкаталога **distrib** и **src**, в которые записываются созданные файлы компонента.

В подкаталог **distrib** помещаются следующие созданные файлы: `Fourier.ctf`, `Fourier.bas` и `Fourier_1_0.dll`. Они предназначены для создания инсталляционного пакета для распространения. В подкаталог **src** записываются копии файлов `Fourier.ctf`, `Fourier.bas` и `Fourier_1_0.dll`, log-файлы регистрации процесса построения, файл `readme.txt`, который содержит полезную информацию для распространения пакета а также все исходные файлы, соответствующие созданной библиотеке `dll`: `def`-файл `lib`-файл, `exp`-файл, заголовочные и файлы `C/C++`.

4.3.2. Подключение компонента к Excel с использованием VBA

Для подключения построенного компонента к Excel нужно создать необходимый код VBA. Но сначала нужно зарегистрировать созданную библиотеку `Fourier_1_0.dll`. Для этого достаточно из строки DOS выполнить команду:

```
mwregsvr.exe C:\R2007a\work\Excel_examples\SpectraExample\Fourier\
distrib\Fourier_1_0.dll
```

Выбор библиотек, необходимых для разработки дополнения. Для этого достаточно сделать следующее:

- запустить Excel;
- открыть редактор Visual Basic, для этого нужно из главного меню Excel, выбрать **Сервис > Макросы > Редактор Visual Basic**;
- в редакторе Visual Basic Editor выбрать **Tools > References**, чтобы открыть диалоговое окно **References VBAProject**. В открывшемся списке выбрать (отметить «галочкой») библиотеки **Fourier 1.0 Type Library** и **MWComUtil 7.6 Type Library** (рис. 4.3.2).

В диалоговом окне указывается также и полный путь для выбранных библиотек. За этим нужно следить, поскольку возможно совпадение имен.

Создание кода VBA главного модуля приложения. Дополнение(add-in) требует, чтобы некоторый код инициализации и некоторые глобальные переменные поддерживали состояние приложения между функциональными вызовами. Для этого создается код модуля программы Visual Basic, управляющий этими задачами. Процедура создания модуля следующая (предполагается, что необходимые библиотеки уже подключены и мы находимся в редакторе Visual Basic):

- щелкнуть правой кнопкой мыши по пункту **VBAProject (Книга1)** в проектном окне **Project VBAProject** и выбрать **Insert ⇒ Module**, рис. 4.3.3. При этом появляется новый модуль в VBA Project;
- во вкладке модуля, установить свойство **Name** как **FourierMain**, рис. 4.3.4;
- написать код для модуля `FourierMain`.

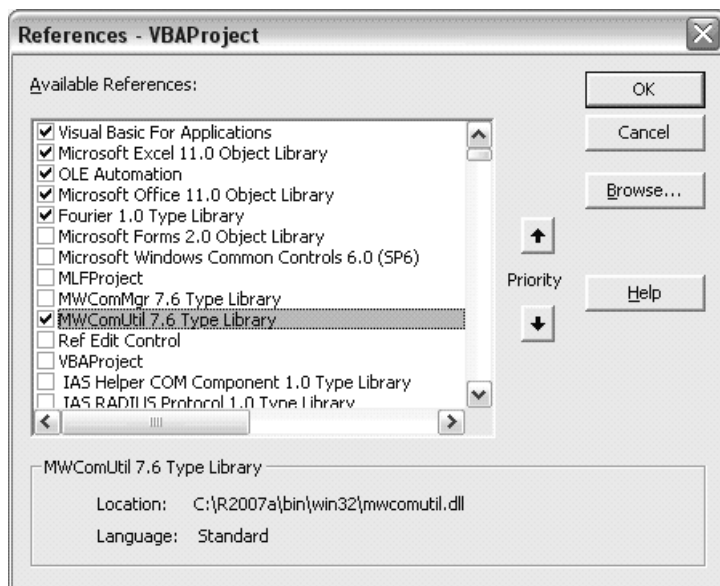


Рис. 4.3.2. Подключение библиотек для проекта VBA

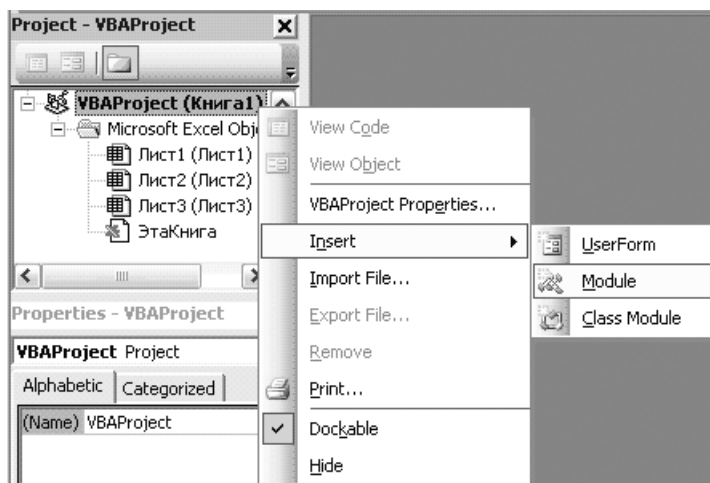


Рис. 4.3.3. Выбор модуля проекта VBA

Введем следующий код (взятый из документации MATLAB):

```
'Инициализация глобальных переменных
'
```

```
Public theFourier As Fourier.Fourier 'Глобальный экземпляр класса Fourier
Public theFFTData As MWComplex ' Глобальный экземпляр MWComplex для FFT
```

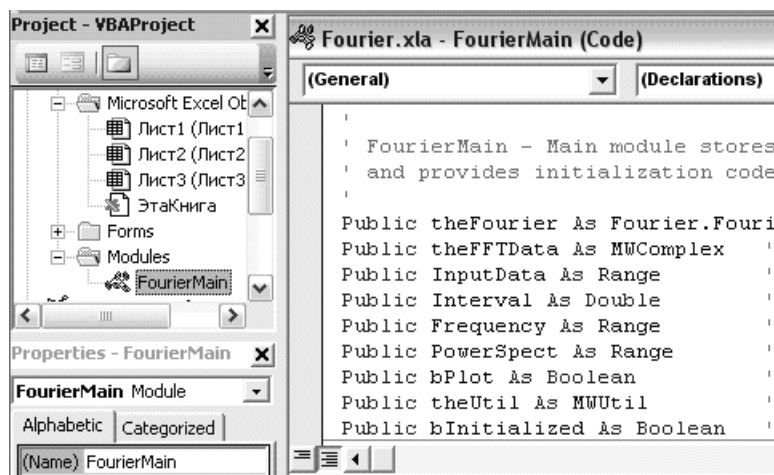


Рис. 4.3.4. Проект VBA

```
Public InputData As Range      ' Диапазон входных данных
Public Interval As Double     ' Шаг дискретизации
Public Frequency As Range     ' Диапазон выходных данных частоты
Public PowerSpect As Range    ' Диапазон выходных данных спектра
Public bPlot As Boolean        ' Флаг состояния графика
Public theUtil As MWUtil      ' Глобальный экземпляр MWUtil
Public bInitialized As Boolean ' Флаг инициализации модуля
```

```
'Загрузка формы спектрального анализа
```

```
Private Sub LoadFourier()
    Dim MainForm As frmFourier
    On Error GoTo Handle_Error
    Call InitApp
    Set MainForm = New frmFourier
    Call MainForm.Show
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
```

```
Private Sub InitApp()
'Инициализация классов и библиотек. Выполняется один раз
'для данной сессии Excel
    If bInitialized Then Exit Sub
    On Error GoTo Handle_Error
    If theUtil Is Nothing Then
        Set theUtil = New MWUtil
    Call theUtil.MWInitApplication(Application)
    End If
    If theFourier Is Nothing Then
        Set theFourier = New Fourier.Fourier
    End If
```

```

If theFFTDData Is Nothing Then
    Set theFFTDData = New MWComplex
End If
    bInitialized = True
Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub

```

Структура кода. Код состоит из трех частей. В первой части дается описание и инициализация необходимых приложению глобальных переменных. Вторая часть содержит код загрузки формы для спектрального анализа. Третья часть содержит код инициализации классов и библиотек. Она выполняется один раз для данной сессии Excel

4.3.3. Создание формы *Visual Basic*

Следующий шаг в процессе интеграции есть разработка пользовательского интерфейса дополнения, используя редактор Visual Basic. Для того чтобы создать новую пользовательскую форму и заполнить ее необходимыми компонентами, нужно сделать следующее:

- щелкнуть правой кнопкой мыши по пункту **VBAProject** в окне проекта VBA и выбрать **Insert** \Rightarrow **UserForm**, рис. 4.3.3. При этом появляется новая форма в окне проекта VBA;
- во вкладке формы, установить свойство **Name** как **frmFourier** и свойство **Caption** как **Spectral Analysis**;
- добавить ряд компонентов в форму, чтобы построить диалоговое окно для использования компонента.

Форма строится обычным образом, используя, хотя и небольшой, но достаточный набор элементов **Controls**, рис.4.3.5. Построим форму, содержащую два фрейма **Frame1** и **Frame2** с заголовками: Input Data, Output Data и две кнопки: **Ok** и **Cancel**, рис.4.3.5.

Во фрейм **Input Data** поместим следующие элементы, необходимые для задания входных параметров:

- **RefEdit** – окно ввода входного диапазона, имя **refedtInput**;
- **TextBox** – окно ввода шага дискретизации Δt , имя **edtSample**;
- **CheckBox** – для выбора графиков сигнала и спектра, имя **chkPlot**, заголовок: Plot time domain signal and power spectral density;
- соответствующие метки с именами **Label1**, **Label2** и заголовками Input Data и Sampling Interval.

Во фрейм **Output Data** поместим следующие элементы, необходимые для задания диапазонов вывода выходных параметров:

- **RefEdit** – окно выбора диапазона для массива частот, имя **refedtFreq**;
- **RefEdit** – окно выбора диапазона для массива вещественных частей данных, полученных после FFT, имя **refedtReal**;

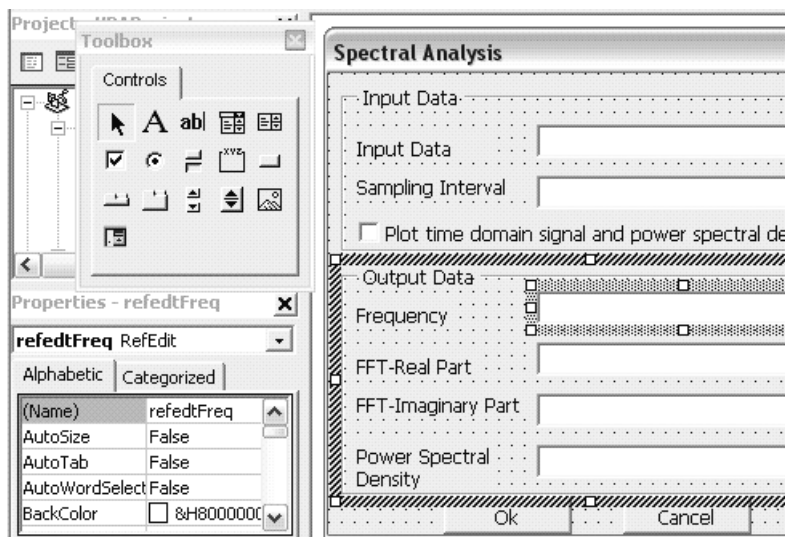


Рис. 4.3.5. Форма дополнения Spectral Analysis

- **RefEdit** – окно выбора диапазона для массива мнимых частей данных, полученных после FFT, имя refedtImag;
- **RefEdit** – окно выбора диапазона для массива спектра сигнала, имя refedtPowSpect;
- соответствующие метки с именами **Label3**, **Label4**, **Label5** и **Label4** и заголовками (Caption) Frequency, FFT – Real Part, FFT – Imaginary Part и Power Spectral Density.

Две кнопки:

- **CommandButton** – выполняет функцию и закрывает диалоговое окно, имя btnOK, со свойствами: Caption = OK, Default = True;
- **CommandButton** – закрывает диалоговое окно не выполняя функцию, имя btnCancel, со свойствами: Caption = Cancel, Default = True.

После того, как форма заполнена окнами данных, нужно щелкнуть правой кнопкой мыши по форме, выбрать **View Code** и ввести необходимый код обработки событий.

Следующий листинг показывает код, который приводит в исполнение программу создания графического интерфейса для примера Spectral Analysis.

```
' Обработчики событий frmFourier
Private Sub UserForm_Activate()
' Активация обработчика событий UserForm. Эту функцию вызывают до
' появления формы, она инициализирует все элементы с заданными
' значениями в глобальных переменных.
On Error GoTo Handle_Error
If theFourier Is Nothing Or theFFTData Is Nothing Then Exit Sub
'Инициализация элементов управления с текущим состоянием
```

```

    If Not InputData Is Nothing Then
        refedtInput.Text = InputData.Address
    End If
    edtSample.Text = Format(Interval)
    If Not Frequency Is Nothing Then
        refedtFreq.Text = Frequency.Address
    End If
    If Not IsEmpty (theFFTDData.Real) Then
    If IsObject(theFFTDData.Real) And TypeOf theFFTDData.Real Is Range Then
        refedtReal.Text = theFFTDData.Real.Address
    End If
    End If
    If Not IsEmpty (theFFTDData.Imag) Then
    If IsObject(theFFTDData.Imag) And TypeOf theFFTDData.Imag Is Range Then
        refedtImag.Text = theFFTDData.Imag.Address
    End If
    End If
    If Not PowerSpect Is Nothing Then
        refedtPowSpect.Text = PowerSpect.Address
    End If
    chkPlot.Value = bPlot
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub

' Обработка события кнопки Cancel. Выход без вычисления fft
' или обновления переменных.
Private Sub btnCancel_Click()
    Unload Me
End Sub

' Обработка события кнопки OK.
' Обновление состояния всех переменных из элементов управления
' и выполнение методов computefft или plotfft.
Private Sub btnOK_Click()
    Dim R As Range

    If theFourier Is Nothing Or theFFTDData Is Nothing Then GoTo Exit_Form
    On Error Resume Next
    ' Процесс ввода
    Set R = Range(refedtInput.Text)
    If Err <> 0 Then
        MsgBox ("Invalid range entered for Input Data")
        Exit Sub
    End If
    Set InputData = R
    Interval = CDBl(edtSample.Text)

```

```

    If Err <> 0 Or Interval <= 0 Then
        MsgBox ("Sampling interval must be greater than zero")
        Exit Sub
    End If
' Процесс вывода
Set R = Range(refedtFreq.Text)
    If Err = 0 Then
        Set Frequency = R
    End If
Set R = Range(refedtReal.Text)
    If Err = 0 Then
        theFFTData.Real = R
    End If
Set R = Range(refedtImag.Text)
    If Err = 0 Then
        theFFTData.Imag = R
    End If
Set R = Range(refedtPowSpect.Text)
    If Err = 0 Then
        Set PowerSpect = R
    End If
bPlot = chkPlot.Value
' Вычисление fft и построение графика спектра
If bPlot Then
    Call theFourier.plotfft(3, theFFTData, Frequency, PowerSpect,
                           InputData, Interval)
Else
    Call theFourier.computefft(3, theFFTData, Frequency, PowerSpect,
                              InputData, Interval)
End If
GoTo Exit_Form
Handle_Error:
    MsgBox (Err.Description)
Exit_Form:
    Unload Me
End Sub

```

4.3.4. Добавление пункта меню *Spectral Analysis* в Excel

Последний шаг в процессе интеграции – это добавление пункта меню в Excel так, чтобы можно было открыть этот инструмент из меню Excel **Сервис**, и сохранение дополнения. Для этого добавляются обработчики событий рабочей книги AddinInstall и AddinUninstall, которые устанавливают и деинсталлируют пункты меню. Пункт меню вызывает функцию LoadFourier в модуле FourierMain. Для того чтобы сделать пункт меню, нужно выполнить следующее:

- в окне проекта VBA открыть список **Microsoft Excel Objects**, щелкнуть правой кнопкой мыши по пункту **Эта Книга (This Workbook)** и выбрать **View Code**;
- поместить следующий код в открывшееся поле ввода кода:

```
Private Sub Workbook_AddinInstall()
' Вызывается, когда Addin установлено
    Call AddFourierMenuItem
End Sub

Private Sub Workbook_AddinUninstall()
' Вызывается, когда Addin не установлено
    Call RemoveFourierMenuItem
End Sub

Private Sub AddFourierMenuItem()
    Dim ToolsMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton

' Удаление, если уже существует
    Call RemoveFourierMenuItem
' Поиск меню Tools
    Set ToolsMenu = Application.CommandBars(1).FindControl(ID:=30007)
    If ToolsMenu Is Nothing Then Exit Sub
' Добавление пункта меню Spectral Analysis
    Set NewMenuItem = ToolsMenu.Controls.Add(Type:=msoControlButton)
    NewMenuItem.Caption = "Spectral Analysis..."
    NewMenuItem.OnAction = "LoadFourier"
End Sub

Private Sub RemoveFourierMenuItem()
    Dim CmdBar As CommandBar
    Dim Ctrl As CommandBarControl
    On Error Resume Next
' Поиск меню Tools и удаление пункт меню Spectral Analysis
    Set CmdBar = Application.CommandBars(1)
    Set Ctrl = CmdBar.FindControl(ID:=30007)
    Call Ctrl.Controls("Spectral Analysis...").Delete
End Sub
```

Сохранение дополнения. Теперь, когда создание кода VBA закончено, можно сохранить дополнение с именем Fourier.xla в каталог <project-directory>\distrib, который Deployment Tool создал, строя проект. Имя дополнения – Spectral Analysis. Процедура сохранения:

- из главного меню Excel, выбрать **Файл** ⇒ **Свойства**. В открывшемся диалоговом окне выбрать вкладку **Документ (Summary)**, ввести название как Spectral Analysis и нажать **ОК**;
- из главного меню Excel выбрать **Файл** ⇒ **Сохранить как**. В открывшемся диалоговом окне выберите **Настройка Microsoft Excel** (Microsoft Excel Add-In (*.xla)) как тип файла и каталог <project-directory>\distrib. Ввести имя файла Fourier.xla и сохранить дополнение.

4.3.5. Тестирование дополнения

Перед распространением дополнения его нужно протестировать. Спектральный анализ обычно используется, чтобы найти основные частоты, из которых составлен сигнал. Для примера создадим сигнал из двух частот, $x_n = \sin(n) + \sin(2n)$, $n = 1, \dots, 100$. Массив $\{n\}$ запишем в столбец Excel A2:A101, массив $\{x_n\}$ – в столбец B2:B101, выходные данные будем размещать в следующие столбцы рядом. Напомним, что для создания массива $\{x_n\}$ достаточно ввести в ячейку B2 формулу «= SIN (A2) + SIN (2 * A2) », скопировать ее в буфер (**Ctrl-Ins**), выделить массив B3:B101 и вставить в него формулу из буфера (**Shist-Ins**). Предположим, что это уже сделано.

Перед тестированием компонента его нужно подключить к Excel как Add-Ins. Для этого нужно из меню **Сервис** \Rightarrow **Надстройки** открыть диалоговое окно выбора надстроек, далее выбрать **Обзор**, найти каталог `matlabroot\work\Excel_examples\SpectraExample\Fourier\distrib`, где находится надстройка `Fourier.xls`, выбрать ее и нажать **ОК**. Если все сделано правильно, дополнение **Spectral Analysis** появится в списке доступных надстроек. Выберем в этом списке **Spectral Analysis** и **Add-Ins** и снова и нажимаем **ОК**. Если все было сделано правильно, это дополнение появляется в списке меню **Сервис**.

Для тестирования дополнения открываем его из меню **Сервис** \Rightarrow **Spectral Analysis**. Открывается диалоговое окно дополнения, т.е. созданная ранее форма этого приложения (рис. 4.3.6). В поле **Input Data** вводим массив B2:B101. Для этого достаточно поставить курсор в данное поле и мышкой выделить этот массив на листе Excel. В качестве шага дискретизации выбираем, например $\Delta t = 1$. Массивы

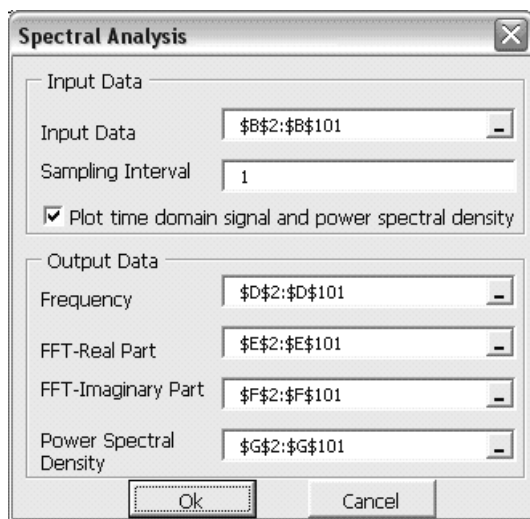


Рис. 4.3.6. Задание диапазонов входных и выходных параметров

для всех выходных данных выбираем аналогично (рис. 4.3.6). Выбираем также опцию вывода графиков сигнала и частотного спектра сигнала. Когда все выбрано, нажимаем кнопку **ОК**. В указанные диапазоны листа Excel записываются все полученные данные (рис. 4.3.7). Кроме этого в стандартном графическом окне MATLAB появляются два графика: исходного сигнала и спектра (рис. 4.3.8).

	A	B	C	D	E	F	G	H
1	n	sin(n)		Freq	Re FFT	Im FFT	Spectr	
2	1	1.750768		0	-0.3992	0.0000	0.0399	
3	2	0.152495		0.01	-0.4010	0.0290	0.0402	
4	3	-0.1383		0.02	-0.4066	0.0586	0.0411	
5	4	0.232556		0.03	-0.4162	0.0894	0.0426	
6	5	-1.50295		0.04	-0.4301	0.1223	0.0447	
7	6	-0.81599		0.05	-0.4491	0.1581	0.0476	
8	7	1.647594		0.06	-0.4741	0.1981	0.0514	
9	8	0.701455		0.07	-0.5065	0.2440	0.0562	
10	9	-0.33887		0.08	-0.5486	0.2984	0.0625	
11	10	0.368924		0.09	-0.6040	0.3651	0.0706	

Рис. 4.3.7. Исходные данные и результаты

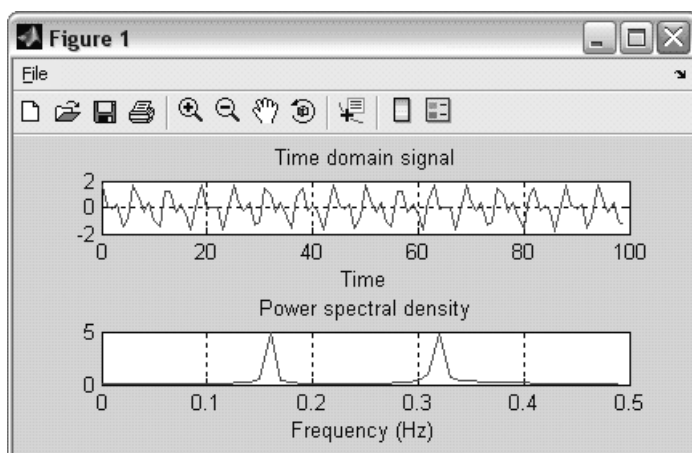


Рис. 4.3.8. Графики сигнала и его спектра

4.3.6. Упаковка и распространение дополнения

На заключительном этапе дополнение упаковывается для распространения на другую машину. Процедура упаковки достаточно простая и была уже рассмотрена ранее. Нужно поместить файл дополнения Fourier.xla в каталог **distrib**, от-

крыть снова проект в Deployment Tool и щелкнуть кнопкой **Package** на инструментальной панели. MATLAB Builder for Excel создает самораспаковывающийся архив Fourier_pkg.exe. Он содержит файлы _install.bat, MCRRegCOMComponent.exe, Fourier_1_0.dll, Fourier.ctf, Fourier.bas и Fourier.xla. При желании, используя опции **Settings**, можно в пакет включить среду исполнения MCR.

Для установки этого дополнения на другой компьютер нужно скопировать файл Fourier_pkg.exe в некоторый каталог и выполнить его. Напомним, что при установке приложения на другую машину необходимо установить путь для среды исполнения MCR,

```
PATH <mcr_root>\<ver>\runtime\win32;
```

4.3.7. Обсуждение программы VBA

В данном разделе, на разобранном выше примере, мы обсудим некоторые вопросы программирования.

Каждый компонент MATLAB Builder для Excel построен как COM-объект, который можно использовать в Microsoft Excel. Можно включить компоненты Excel Builder в проект VBA, создавая простой код модуля с функциями и/или подпрограммами, которые загружают необходимые компоненты, вызывают необходимые методы и обрабатывают ошибки.

Инициализация библиотек MATLAB Builder для Excel с Excel. Перед использованием любого компонента MATLAB Builder для Excel, инициализируются библиотеки поддержки. Это делается один раз для сессии Excel, чтобы использовать компоненты Excel Builder.

Для инициализации вызывается сервисная функция InitModule библиотеки MWInitApplication, которая является членом класса MWUtil. Этот класс есть часть библиотеки MWComUtil.

Один из способов добавления этого кода инициализации в модуль VBA состоит в том, чтобы создать подпрограмму, которая делает инициализацию один раз. Следующий образец программы Visual Basic инициализирует библиотеки с текущим экземпляром Excel. Глобальная переменная типа Object, называемая MCLUtil, содержит экземпляр класса MWUtil, и другая глобальная переменная типа Boolean, названная bModuleInitialized, хранит состояние процесса инициализации. Частная подпрограмма InitModule() создает экземпляр класса MWComUtil и вызывает метод MWInitApplication с параметром Application. После выполнения этой функции все последующие запросы проходят без переинициализации.

Следующий код инициализации по умолчанию записывается в коде VBA, созданном при построении компонента – это файл Fourier.bas из подкаталога \distrib. Каждая функция, которая использует компоненты Excel Builder, может включить вызов InitModule вначале, чтобы гарантировать, что инициализация всегда выполняется как необходимо.

```
Dim MCLUtil As Object  
Dim bModuleInitialized As Boolean
```

```

Dim Fourier As Object

Private Sub InitModule()
    If Not bModuleInitialized Then
        On Error GoTo Handle_Error
        If MCLUtil Is Nothing Then
            Set MCLUtil = CreateObject(«MWComUtil.MWUtil7.6»)
        End If
        Call MCLUtil.MWInitApplication(Application)
        bModuleInitialized = True
        Exit Sub
    Handle_Error:
        bModuleInitialized = False
    End If
End Sub

```

Сравните этот код с кодом инициализации в модуле FourierMain VBA при построении компонента Fourier.xla.

Создание экземпляра класса. Перед вызовом метода класса (компилированная функция MATLAB) нужно создать экземпляр класса, который содержит этот метод. Это можно сделать функцией **CreateObject** и оператором **New**.

Функция CreateObject. Этот метод использует для создания экземпляра класса функцию CreateObject Visual Basic API. Использование этого метода показано на следующем примере процедуры вызова функции computefft. Код взят из файла Fourier.bas созданного выше компонента (см. также предыдущий пример кода).

```

Function computefft(Optional data As Variant,
Optional interval As Variant) As Variant
    Dim fftdata, freq, powerspect As Variant

    On Error Goto Handle_Error
    Call InitModule
    If Fourier Is Nothing Then
        Set Fourier = CreateObject( "Fourier.Fourier.1_0")
    End If
    Call Fourier.computefft(1, fftdata, freq, powerspect, data, interval)
    computefft = fftdata
    Exit Function
Handle_Error:
    computefft = "Error in " & Err.Source & ": " & Err.Description
End Function

```

Оператор New. Этот метод использует оператор New из Visual Basic. Перед использованием этого метода нужно сослаться на библиотеку, содержащую класс в текущем проекте VBA. Для этого выбираем меню **Tools** в редакторе Visual Basic и затем выбираем **References**, чтобы показать список **Доступные Ссылки** (Available References). Из этого списка выбираем необходимый тип библиотеки. Если, например, выбрана библиотека **mycomponent 1.0**, то экземпляр aClass класса myclass создается строкой

```
Set aClass = New mycomponent.myclass
```


В рассматриваемом примере экземпляр класса задается следующей строкой в модуле `FourierMain`:

```
Set theFourier = New Fourier.Fourier
```

В предыдущем примере, экземпляр класса использует для выполнения метода локальные переменные процедуры. Это создает и уничтожает экземпляр класса при каждом вызове функции. Чтобы избежать этого, можно объявить один экземпляр класса,

```
Dim aClass As mycomponent.myclass
```

до текста функции, который многократно используется всеми вызовами функции. Пример этого способа см. в коде инициализации библиотек предыдущего примера, а также в первой части кода модуля `FourierMain` при построении компонента `Fourier`.

Вызов методов экземпляра класса. После того, как создан экземпляр класса, можно вызвать методы класса для получения доступа к компилируемым функциям MATLAB. MATLAB Builder для Excel применяет следующее стандартное преобразование исходного синтаксиса функции MATLAB.

Когда метод имеет выходные аргументы, первый аргумент всегда определяет количество выходных параметров, это аргумент `nargout`, который имеет тип `Long`. Этот входной параметр передает обычный параметр `nargout` MATLAB к компилируемой функции. Методы, которые не имеют аргументов вывода, не имеют аргумента `nargout`. После аргумента `nargout` следует список параметров выхода в том же самом порядке, в каком они стоят с левой стороны оригинальной функции MATLAB. Затем идут входные параметры в том же самом порядке, как они стоят с правой стороны оригинальной функции MATLAB. Все аргументы входа и выхода имеют тип `Variant`, тип данных Visual Basic.

Тип `Variant` может содержать любой из основных типов VBA, массивы любого типа, и объектные ссылки. Вообще, можно использовать любой тип Visual Basic в качестве аргумента в методе класса, за исключением Visual Basic UDT. Можно также передать диапазоны Excel Range непосредственно как аргументы выхода и входа.

Следующий пример функции иллюстрирует процесс передачи параметров входа от VBA к методам класса компонента Excel Builder и получения трех выходов. Функция вызывает метод класса, который соответствует `m-`функции MATLAB вида:

```
[fftdata, freq, powerspect] = computefft(data, interval)
```

Пример кода:

```
Function computefft(Optional data As Variant,  
                   Optional interval As Variant) As Variant  
    Dim fftdata, freq, powerspect As Variant  
    Call InitModule  
    If Fourier Is Nothing Then  
        Set Fourier = CreateObject( "Fourier.Fourier.1_0")
```

```
End If  
Call Fourier.computefft(3, fftdata, freq, powerspect, data, interval)  
Exit Function  
End Function
```

Обработка ошибок при вызове метода. Ошибки, которые происходят при создании экземпляра класса или в течение метода класса, создают исключение в текущей процедуре. Visual Basic обеспечивает обработку исключений через директиву On Error Goto <label>, в котором указывается, что выполнение программы возвращается к <label>, когда происходит ошибка (<label> должен быть расположен в той же самой процедуре что и On Error Goto <label>). Все ошибки обрабатываются этим способом, включая ошибки в пределах исходного кода MATLAB. Исключение создает объект Visual Basic ErrObject в переменной по имени Err. См. примеры обработки ошибок в приведенных выше кодах.

4.3.8. Использование флагов

Каждый компонент MATLAB Builder для Excel имеет простое свойство чтения/записи по имени MWFlags типа MWFlags. Оно состоит из двух наборов констант: флажки, форматирующее массивы и флажки преобразования данных. Флажки, форматирующее массивы затрагивают преобразование множеств, тогда как флажки преобразования данных имеют дело с преобразованиями типа индивидуальных элементов множества.

Флаги, форматирующие массив. Они делают преобразование массивов входных, или выходных переменных. Для их использования нужно выбрать библиотеку MWComUtil в текущем проекте VBA, выбирая **Tools** \Rightarrow **References** и выбирая **MWComUtil 7.6 Type Library** из списка.

Флаг InputArrayFormat. Значение по умолчанию для флага InputArrayFormat есть mwArrayFormatMatrix. Это значение по умолчанию используется потому, что данные массива, происходят из диапазонов Excel, которые находятся всегда в форме массива Variant и функции MATLAB наиболее часто имеют дело с матричными аргументами.

Если нужно получить, например, массив ячеек, то необходимо установить флаг InputArrayFormat в положение mwArrayFormatCell. Для этого достаточно добавить следующую строку после создания класса и перед вызовом метода:

```
aClass.MWFlags.ArrayFormatFlags.InputArrayFormat =  
mwArrayFormatCell
```

Установка этого флага представляет весь входной массив компилируемой функции MATLAB как массив ячеек.

Флаг OutputArrayFormat. Применяется совершенно аналогично для управления форматом выходных аргументов.

Флаг AutoResizeOutput. Используется для объектов Excel Range, передаваемых непосредственно как параметры выхода. Когда этот флаг установлен, выходной диапазон автоматически изменяет размеры, чтобы соответствовать окончательному массиву. Если этот флаг не установлен, диапазон для вывода должен

иметь размеры не меньше, чем размеры выходного массива, иначе данные будут обрезаны.

Флаг TransposeOutput. Транспонирует весь выходной массив. Этот флаг полезен, когда функции MATLAB имеют выходные одномерные массивы. По умолчанию, MATLAB понимает одномерные множества как 1-на-n матрицы (векторы строки), которые становятся в строку рабочего листа Excel. Однако в Excel предпочтительнее одномерные данные записывать в виде столбца.

Следующий пример автоматически изменяет размеры и транспонирует диапазон выхода:

```
Sub foo(Rout As Range, Rin As Range )
    Dim aClass As mycomponent.myclass
    Set aClass = New mycomponent.myclass
    aClass.MWFlags.ArrayFormatFlags.AutoResizeOutput = True
    aClass.MWFlags.ArrayFormatFlags.TransposeOutput = True
    Call aClass.foo(1,Rout,Rin)
Exit Sub
End Sub
```

Флаги преобразования данных. Они имеют дело с преобразованиями индивидуальных элементов массива. По умолчанию, компоненты Excel Builder считывают установки флагов преобразования данных в свойстве класса MWFlags. Это верно для всех типов Visual Basic, за исключением типов Excel Builder: MWStruct, MWField, MWComplex, MWSparse и MWArg. Каждый из этих типов выставляет свое собственное свойство MWFlags и игнорирует свойства класса, метод которого вызывается. Класс MWArg дается специально для случая, когда отдельный аргумент нуждается в другом описании, отличном от свойств класса по умолчанию.

Два флага преобразования данных, CoerceNumericToType и InputDateFormat, управляют преобразованием чисел и дат от VBA в MATLAB. Если исходная функция MATLAB ожидает double для аргументов, то нужно присвоить double этим аргументам, но это может быть иногда невозможно. Тогда устанавливают флаг CoerceNumericToType, заставляя преобразовать весь числовой вход в double. Для этого можно поместить следующую строку после создания класса, например, aClass, и перед вызовом метода:

```
aClass.MWFlags.DataConversionFlags.CoerceNumericToType =
mwTypeDouble
```

Флаг InputDateFormat управляет преобразованием даты VBA. Например, в строку:

```
aClass.MWFlags.DataConversionFlags.InputDateFormat =
mwDateFormatString
```

Следующий пример использует объект MWArg для изменения флага преобразования для отдельного аргумента в вызове метода. В этом случае первый аргумент выхода (y1) приводится к типу Date, а второй аргумент выхода (y2) использует текущие флаги преобразования по умолчанию из aClass.

```
Sub foo(y1 As Variant, y2 As Variant)
    Dim aClass As mycomponent.myclass
```

```
Dim ytemp As MWArg
Dim today As Date
today = Now
Set aClass = New mycomponent.myclass
Set y1 = New MWArg
y1.MWFlags.DataConversionFlags.OutputAsDate = True
Call aClass.foo(2, ytemp, y2, today)
y1 = ytemp.Value
Exit Sub
End Sub
```

Замечание 1. Более подробную информацию о применении флагов и дополнительные примеры см. в документации класса MWFlags.

Замечание 2. MATLAB Builder для Excel создает единственную MCR, когда в приложении возникает первый СОМ-класс. Эта среда MCR многократно используется и общедоступна для всех последующих экземпляров класса в пределах компонента, и приводит к более эффективному использованию памяти и не требует нового запуска MCR в каждом последующем экземпляре класса. Все экземпляры класса имеют общий доступ к одному рабочему пространству MATLAB и общий доступ к глобальным переменным в m-файлах при построении компоненты. Это делает свойства СОМ-класса статическими свойствами вместо свойств типа экземпляра.

4.4. Библиотека утилит Excel Builder

В данном параграфе дается описание функций и сервисных классов пакета расширения MATLAB Builder для Excel.

4.4.1. Функции MATLAB Builder для Excel

Пакет расширения MATLAB Builder для Excel имеет две функции:

- **componentinfo** – запрос о системной регистрации компонента, созданного MATLAB Builder для Excel;
- **deploytool** – вызов графического интерфейса разработки MATLAB Builder для Excel и MATLAB Compiler.

Функция `deploytool` достаточно подробно была рассмотрена выше. Вторая функция может применяться следующим образом:

- `Info = componentinfo` – информация о всех установленных компонентах;
- `Info = componentinfo('mycomponent')` – информация о компоненте `mycomponent`;
- `Info = componentinfo('mycomponent', 1, 0)` – информация о компоненте `mycomponent` версии 1.0.

Возвращаемое значение `Info` – это массив структур, представляющий всю информацию о регистрации и типе, необходимую для загрузки и использования компоненты. Информация о компоненте содержится в следующих полях:

- Name – имя компонента;
- TypeLib – тип библиотеки компонента;
- LIBID – тип библиотеки компонента GUID;
- MajorRev – номер старшей версии;
- MinorRev – номер низшей версии;
- FileName – имя файла библиотеки и путь. Так как все компоненты Excel Builder имеют тип библиотеки DLL, то это имя файла совпадает с именем DLL;
- Interfaces – массив структур, определяющий весь интерфейс определений в типе библиотеки. Каждая структура содержит два поля:
 - Name – имя интерфейса;
 - IID – интерфейс GUID;
- CoClasses – массив структур, определяющий все COM классы в компоненте. Каждая структура содержит поля:
 - Name – имя класса;
 - CLSID – GUID класса;
 - ProgID – зависимая от версии, программа ID;
 - VerIndProgID – независимая от версии, программа ID;
 - InprocServer32 – полное имя и путь для компоненты DLL;
 - Methods – структура, содержащая функциональные прототипы всех методов класса, определенных для этого интерфейса. Эта структура содержит четыре поля:
 - IDL – массив прототипов функций IDL;
 - M – массив прототипов функций MATLAB;
 - C – массив прототипов функций C-языка;
 - VB – массив прототипов функций VBA;
 - Properties – массив ячеек, содержащий имена всех свойств класса;
 - Events – структура, содержащая прототипы функций всех событий, определенных для этого класса. Эта структура содержит четыре поля:
 - IDL – массив прототипов функций IDL;
 - M – массив прототипов функций MATLAB;
 - C – массив прототипов функций C-языка;
 - VB – массив прототипов функций VBA;

Пример вызова функции `componentinfo` приведен в разделе 4.2.2.

4.4.2. Библиотека утилит Excel Builder

В этом разделе рассматривается библиотека `MWComUtil`, которая предоставлена MATLAB Builder для Excel. Данная библиотека является свободно распространяемой и включает несколько функций, используемых в обработке массивов, а также определения типов, используемые в преобразовании данных. Эта библиотека содержится в файле `mwcomutil.dll`. Она должна быть зарегистрирована один раз машине, где используются компоненты Builder Excel. Для регистрации библиотеки `MWComUtil` в командной строке DOS нужно исполнить команду:

```
mwregsvr mwcomutil.dll
```

Библиотека MWComUtil включает семь классов и три перечисления. Перед использованием библиотеки нужно сделать явную ссылку на MWComUtil в интегрированной среде разработки Visual Basic (Excel). Для этого из главного меню редактора VB нужно выбрать **Tools** ⇒ **References**. Появляется диалоговое окно **References** с прокручиваемым списком доступных типов библиотек. В этом списке, выберите **MWComUtil 7.6 Type Library** и нажмите **ОК** (рис. 4.3.2.).

Классы библиотеки утилит. Библиотеки утилит MATLAB Builder для Excel представлена следующими классами:

- класс MWUtil – содержит ряд статических сервисных методов, используемых в обработке массива и инициализации приложений;
- класс MWFlags – содержит ряд флагов форматирующих массив и флагов преобразования данных;
- класс MWStruct – передает или получает тип Struct в или от скомпилированного метода;
- класс MWField – содержит ссылку поля в объекте MWStruct;
- класс MWComplex – передает или принимает комплексный числовой массив в или от скомпилированного метода;
- класс MWSparse – передает или принимает двумерный разреженный числовой массив в или от скомпилированного метода;
- класс MWArg – передает параметр в скомпилированный метод, когда флаги преобразования данных изменены для этого одного параметра.

Рассмотрим немного подробнее эти классы. Дальнейшую информацию можно найти в документации MATLAB Builder для Excel.

Класс MWUtil

Содержит ряд статических сервисных методов, используемых в обработке массива и инициализации приложений. Методы MWUtil:

- MWInitApplication(pApp As Object) – инициализирует библиотеку с текущим экземпляром Excel. Эту функцию нужно вызвать один раз для каждого сеанса Excel, который использует компоненты Excel Builder;
- MWPack(pVarArg, [Var0], [Var1], ... ,[Var31]) – упаковывает список переменной длины аргументов Variant в единственный массив Variant. Эта функция обычно используется для того, чтобы создать ячейку varargin из списка отдельных вводов;
- MWUnpack(VarArg, [nStartAt As Long], [bAutoResize As Boolean = False], [pVar0], [pVar1], ..., [pVar31]) – распаковывает массив Variant в индивидуальные параметры Variant. Эта функция обратная к MWPack и обычно используется для обработки ячейки varargout в индивидуальные параметры Variant;
- MWDate2VariantDate(pVar) – преобразование выходных дат типа MATLAB к датам Variant.

Класс MWFlags

Содержит ряд флагов форматирующих массив и флагов преобразования данных. Все компоненты Excel Builder содержат ссылки на объект MWFlags, который может изменить правила преобразования данных на уровне объекта. Этот класс со-

держит следующие свойства: `ArrayFormatFlags As MWArrayFormatFlags`, `DataConversionFlags As MWDataConversionFlags` и метод `Clone(ppFlags As MWFlags)`.

Свойство `ArrayFormatFlags As MWArrayFormatFlags`. Свойство `ArrayFormatFlags` управляет форматированием массива (матрица или массив ячеек) с применением этих правил к вложенным массивам. Класс `MWArrayFormatFlags` содержит следующие свойства:

- `InputArrayFormat As mwArrayFormat` – свойство типа `mwArrayFormat` управляет форматированием массивов, которые передаются как входные параметры для методов класса `Builder Excel`. Значение по умолчанию есть `mwArrayFormatMatrix`;
- `InputArrayIndFlag As Long` – управляет уровнем применения правила, установленного свойством `InputArrayFormat` для вложенных массивов, значение по умолчанию есть 0;
- `OutputArrayFormat As mwArrayFormat` – свойство типа `mwArrayFormat` управляет форматированием выходных массивов методов класса `Excel Builder`. Значение по умолчанию `mwArrayFormatAsIs`;
- `OutputArrayIndFlag As Long` – аналогично свойству `InputArrayIndFlag`, управляет уровнем применения правила, установленного свойством `OutputArrayFormat` для массивов вывода;
- `AutoSizeOutput As Boolean` – относится только к диапазонам `Excel`. Когда для вывода метода указан диапазон ячеек в рабочем листе `Excel`, а размер массива вывода и его форма неизвестны во время вызова, то значение `True` этого флага обязывает изменять размеры каждого диапазона `Excel`, чтобы соответствовать размеру вывода. Изменение размеров применяется относительно верхнего левого угла каждого диапазона. Значение по умолчанию для этого флага `False`;
- `TransposeOutput As Boolean` – установка `True` этого флажка транспонирует параметры вывода. Этот флажок полезен, поскольку функция `MATLAB` возвращает выводы как векторы строки, а в `Excel` удобнее столбцы. Значение по умолчанию для этого флажка `False`.

Свойство `DataConversionFlags As MWDataConversionFlags`. Свойство `DataConversionFlags` управляет обработкой входных переменных, если необходимо приведение типа. Класс `MWDataConversionFlags` содержит следующие свойства:

- `CoerceNumericToType As mwDataType` – преобразовывает все числовые входные параметры в один тип `MATLAB`. Этот флаг полезен, когда переменные в программе `Visual Basic` имеют различные типы, например, `Long`, `Integer`, и т.д., а все переменные, которые передают к скомпилированному коду `MATLAB` должны быть `double`. Значение по умолчанию для этого свойства `mwTypeDefault`;
- `InputDateFormat As mwDateFormat` – преобразует даты, передаваемые как входные к методам классов `Excel Builder`. Значение по умолчанию есть `mwDateFormatNumeric`;
- `OutputAsDate As Boolean` – свойство обрабатывает параметр вывода в виде даты. По умолчанию, числовые даты, которые являются выходными от

скомпилированных функций MATLAB, передаются как Double, для возможности использования смещения. Установка True этого флага преобразовывает все значения вывода в тип Double;

- **DateBias As Long** – регулирует согласование методов представления дат в числах в MATLAB и в Excel. Напомним, что в MATLAB числу 1 соответствует дата 01-Jan-0000, а в Excel числу 1 соответствует дата 01.01.1900. Разница между ними составляет смещение 693961. Поэтому при передаче данных из Excel в MATLAB нужно прибавлять 693961 – это есть значение по умолчанию данного свойства.

Метод Clone(ppFlags As MWFlags). Создает копию объекта MWFlags. Распределяет новый объект MWFlags и создает настоящую копию содержания объекта. Параметр ppFlags имеет тип MWFlags и ссылается на неинициализированный объект MWFlags, который получает копию

Class MWStruct

Передаёт или получает тип Struct в, или, от скомпилированного метода. Этот класс содержит следующие свойства и методы:

- метод Initialize([varDims], [varFieldNames]);
- свойство Item([i0], [i1], ..., [i31]) As MWField;
- свойство NumberOfFields As Long;
- свойство NumberOfDims As Long;
- свойство Dims As Variant;
- свойство FieldNames As Variant;
- метод Clone(ppStruct As MWStruct).

Более подробную информацию об этом классе можно найти в документации MATLAB Builder для Excel.

Класс MWField

Содержит единственную ссылку поля в объекте MWStruct. Этот класс содержит следующие свойства и методы:

- свойство Name As String;
- свойство Value As Variant;
- свойство MWFlags As MWFlags;
- метод Clone(ppField As MWField).

Более подробную информацию об этом классе можно найти в документации MATLAB Builder для Excel.

Класс MWComplex

Передаёт или принимает комплексный числовой массив в, или, от скомпилированного метода класса. Этот класс содержит следующие свойства и методы:

- **Real As Variant** – хранит вещественную часть комплексного массива (чтение/запись). Свойство Real есть свойство по умолчанию класса MWComplex. Значение этого свойства может быть любым типом приведенным к Variant. Допустимые числовые типы Visual Basic для комплексных массивов включают Byte, Integer, Long, Single, Double, Currency и Variant/vbDecimal;

- **Imag As Variant** – хранит мнимую часть комплексного массива (чтение/запись). Свойство **Imag** является дополнительным и может быть **Empty** для чисто вещественного массива;
- **MWFlags As MWFlags** – хранит ссылку на объект **MWFlags**. Это свойство устанавливает или получает флаги форматирующие массив и флаги преобразования данных для отдельного комплексного массива. Каждый объект **MWComplex** имеет собственное свойство **MWFlags**. Это свойство отменяет значение любого набора флажков на объекте, методы которого вызываются;
- **Clone(ppComplex As MWComplex)** – создает настоящую копию объекта **MWComplex**. Эта функция вызывается, когда требуется отдельный объект вместо общедоступной копии существующей ссылки объекта.

Пример. Следующая программа Visual Basic создает комплексный массив со следующими значениями (синтаксис MATLAB) $x = [1+i, 1+2i; 2+i, 2+2i]$:

```
Sub foo()
    Dim x As MWComplex
    Dim rval(1 To 2, 1 To 2) As Double
    Dim ival(1 To 2, 1 To 2) As Double
    For I = 1 To 2
        For J = 1 To 2
            rval(I,J) = I
            ival(I,J) = J
        Next
    Next
    Set x = new MWComplex
    x.Real = rval
    x.Imag = ival
    Exit Sub
End Sub
```

Class MWSparse

Передаёт или принимает двумерный разреженный числовой массив в или от скомпилированного метода. Этот класс имеет следующие свойства и методы:

- свойство **NumRows As Long**;
- свойство **NumColumns As Long**;
- свойство **RowIndex As Variant**;
- свойство **ColumnIndex As Variant**;
- свойство **Array As Variant**;
- свойство **MWFlags As MWFlags**;
- метод **Clone(ppSparse As MWSparse)**.

Более подробную информацию об этом классе можно найти в документации MATLAB Builder для Excel.

Класс MWArg

Передаёт аргумент в скомпилированный метод, когда флаги преобразования данных применяются только для этого одного параметра. Класс имеет следующие свойства и методы:

- Value As Variant – хранит фактическое значение аргумента для передачи. Любой тип, который можно передать к скомпилированному методу, допустим для этого свойства;
- MWFlags As MWFlags – хранит ссылку на объект MWFlags. Это свойство устанавливает или получает флаги, форматирующие массив и флаги преобразования данных для отдельного параметра. Каждый объект MWArg имеет свое собственное свойство MWFlags. Это свойство отменяет значение любого набора флагов на объекте, методы которого вызываются;
- Clone(ppArg As MWArg) – метод создает копию объекта MWArg. Параметр ppArg имеет тип MWArg и ссылается на неинициализированный объект MWArg, который получает копию. Метод Clone распределяет новый объект MWArg и создает реальную копию содержания объекта.

Перечисления

Библиотека утилит MATLAB Builder для Excel имеет три перечисления (наборы констант): Enum mwArrayFormat, Enum mwDataType, Enum mwDateFormat.

Enum mwArrayFormat. Перечисление mwArrayFormat – это ряд констант, которые обозначают правила, форматирования массива. Таблица 4.4.1 перечисляет члены этого перечисления.

Таблица 4.4.1. Значения mwArrayFormat

Константа	Значение	Описание
mwArrayFormatAsIs	0	Массив не переформатируется
mwArrayFormatMatrix	1	Массив форматируется как матрица
mwArrayFormatCell	2	Массив форматируется как массив ячеек

Enum mwDataType. Перечисление mwDataType – это ряд констант, которые обозначают числовой тип MATLAB. Таблица 4.4.2 перечисляет члены этого перечисления.

Таблица 4.4.2. Значения mwDataType

Константа	Значение	Тип MATLAB
mwTypeDefault	0	N/A
mwTypeLogical	3	logical
mwTypeChar	4	char
mwTypeDouble	6	double
mwTypeSingle	7	single
mwTypeInt8	8	int8
mwTypeUInt8	9	uint8
mwTypeInt16	10	int16
mwTypeUInt16	11	uint16
mwTypeInt32	12	int32
mwTypeUInt32	13	uint32

Enum mwDateFormat. Перечисление mwDateFormat – это ряд констант, которые обозначают правила форматирования для дат. Таблица 4.4.3 перечисляет члены этого перечисления.

Таблица 4.4.3. Значения mwDateFormat

Константа	Значение	Описание
mwDateFormatNumeric	0	Формат дат в виде числовых значений
mwDateFormatString	1	Формат дат в виде строк

4.5. Справка по VBA

Хотя язык программирования Visual Basic можно считать общеизвестным, дадим в этом разделе небольшую справку по основам VBA. Для подробностей можно обратиться к какому-нибудь руководству, например, [Ga].

Общие правила написания кода Visual Basic. Программа на VB состоит из нескольких модулей. Каждый модуль также состоит из нескольких частей. В начале идет раздел описаний модуля, где описываются общие объекты и переменные, а затем – коды процедур. Это хорошо видно на примере кода модуля FourierMain для рассмотренного во втором параграфе проекта VBAProject (Fourier.xla).

Текст, следующий в программе за символом (') одинарной кавычки представляет собой комментарий и до конца строки игнорируется компилятором.

Строка кода не заканчивается каким-либо символом (типа точки с запятой), достаточно перейти на следующую строку для ввода следующей команды. Если строка является слишком длинной, то ее можно продолжить на следующей строке, соблюдая правило переноса строк. Сочетание символов <пробел>+<знак подчеркивания> в конце строки означает *перенос строки* кода на следующую строку. Например,

```
Function computefft(Optional data As Variant, _
                    Optional interval As Variant) As Variant
```

Типы данных. Тип данных определяет значения, которые может принимать указанная переменная. В VBA имеются следующие *типы данных*: Boolean (логический), Byte (байт), Integer (целое), Long (длинное целое), Decimal (целое 96-битное, 28 десятичных разрядов), Single (с плавающей запятой, обычной точности), Double (с плавающей запятой, двойной точности), Date (дата и время), String (строка), Currency (денежный), Object (объект) или Variant (тип, используемый по умолчанию). Кроме того, для Excel имеется тип данных Range (диапазон), а для Excel Builder имеются и другие типы данных, например, классов MWComplex, MWSparse.

Тип данных Variant. Если при описании константы, переменной, или аргумента не указан тип данных, им автоматически присваивается тип данных *Variant*. Переменные, описанные с типом данных Variant, могут содержать строку, дату,

время, логические (Boolean) или числовые значения и могут автоматически преобразовываться к другому типу. Числовые значения Variant занимают 16 байт памяти и доступ к ним осуществляется медленнее, чем к переменным, которые описаны явным образом. Строковое значение Variant занимает 22 байта памяти. Переменные, описанные неявно, получают тип данных Variant. Данные типа Variant могут иметь особое значение Null, которое означает, что данные отсутствуют или неизвестны.

Описание переменных. Перед использованием переменной должен быть указан ее статус и тип. Статус переменной определяется инструкциями: Dim, Public, Private и Static. Эти инструкциями определяют область определения или видимости переменной.

Инструкция **Public** используется для описания общих переменных.

```
Public strName As String
```

Общие переменные могут использоваться в любой процедуре проекта. Если общая переменная описана в стандартном модуле или в модуле класса, она также может использоваться в любом проекте, в котором имеется ссылка на проект, где описана эта переменная.

Инструкция **Private** используется для описания личных переменных уровня модуля.

```
Private MyName As String
```

Личные переменные доступны только для процедур одного и того же модуля.

Инструкция **Dim**. На уровне модуля инструкция Dim эквивалентна инструкции Private. Для создания локальной переменной на уровне процедуры применяется инструкция описания Dim внутри процедуры. Тогда эта переменная может использоваться только в данной процедуре.

Чтобы создать переменную на уровне модуля, инструкция описания Dim располагается в начале модуля, в разделе описаний. Тогда переменная доступна для всех процедур данного модуля, но не может использоваться процедурами из других модулей проекта.

Переменные, описанные с помощью инструкции **Static** вместо инструкции Dim, сохраняют свои значения при выполнении программы.

Если тип данных не задан, по умолчанию переменная принимает тип Variant. Имеется также возможность создать определяемый пользователем тип данных с помощью инструкции Type.

Допускается описание нескольких переменных в одной строке. В этом случае, чтобы задать тип данных, надо указать определенный тип для каждой переменной, например,

```
Dim X As Integer, Y As Long, Z As Double
```

В следующей строке X и Y описываются как Variant; и только Z описывается как Double,

```
Dim X, Y, Z As Double
```

Инструкция **Option Explicit**. В языке Visual Basic можно неявно описать переменную, просто используя ее в инструкции присвоения. Все неявно описанные переменные имеют тип Variant. Если неявные описания нежелательны, инструкция Option Explicit должна предшествовать в модуле всем процедурам. Эта инструкция налагает требование явного описания всех переменных этого модуля. Если модуль содержит инструкцию Option Explicit, при попытке использования неопisanного или неверно введенного имени переменной возникает ошибка во время компиляции.

Замечание. Явное описание динамических массивов и массивов с фиксированной размерностью обязательно.

Описание констант. В отличие от переменных константы не могут изменять свои значения. Для описания *константы* и определения ее значения используется инструкция Const, перед которой может стоять модификатор доступа Public или Private. После описания константу нельзя модифицировать и нельзя присваивать ей новое значение. Константа описывается в процедуре или в начале модуля, в разделе описаний. При описании общих констант уровня модуля инструкции Const должно предшествовать ключевое слово Public. Для явного описания личных констант перед инструкцией Const надо поставить ключевое слово Private.

Константы могут быть того же типа, что и переменные. Поскольку значение константы уже известно, можно задать тип данных в инструкции Const. В следующем примере константа Public constAge описывается как Integer и ей присваивается значение 34:

```
Public Const constAge As Integer = 34
```

Допускается также описание нескольких констант в одной строке. В этом случае, чтобы задать тип данных, надо указать определенный тип для каждой константы, например,

```
Const constAge As Integer = 34, constWage As Currency = 35000
```

Использование массивов. Как и другие переменные, *массивы* описываются с помощью инструкций Dim, Static, Private или Public. Если тип данных при описании массива не задается, подразумевается, что элементы массива имеют тип Variant. Формально, массив отличается от обычной переменной только тем, что после имени массива нужно в скобках указать количество его элементов, либо размеры. Допускается до 60 измерений. Приведем несколько примеров,

```
Dim A(9) As Integer 'Задание вектора из десяти целых чисел
```

```
Dim B(2,3) As Single 'Задание матрицы 3-на-4 действительных чисел
```

По умолчанию, индексация в массивах начинается с нуля. Однако это можно изменить директивой Option Base, например, после команды

```
Option Base 1
```

индексация начинается с единицы. Допускается также явное задание нижней границы индексов массива с помощью предложения To. Например, в следующем примере,

```
Dim C(1 To 4,3 To 10) As Single
```

задается матрица 4-на-8 действительных чисел, в которой первый индекс меняется от 1 до 4, а второй – меняется от 3 до 10.

Инициализация массива. Определить значения элементов массива можно обычными операторами присвоения:

```
B(i, j) = k
```

Оператор `Array` объединяет в массив все элементы, перечисленные как аргументы в скобках, через запятую. Например,

```
Dim varData As Variant  
varData = Array(1, 2, 3, 4)
```

Динамические массивы. Иногда требуется изменить размер массива. В этом случае его объявляют как *динамический*, без указания его размеров,

```
Dim D() As Double
```

Если в процессе работы программы становится известен размер массива, то командой `ReDim` устанавливаются границы его индексов,

```
Dim D() As Double  
ReDim D(2 To 5)
```

Допускается повторное применение `ReDim`.

Создание объектных переменных. Объектная переменная может рассматриваться как объект, ссылку на который она содержит. С ее помощью возможно задание или возвращение свойств объекта или использование любых его методов. Для создания объектной переменной необходимо: описать объектную переменную и присвоить эту объектную переменную объекту.

Описание объектной переменной. Для описания *объектной переменной* применяется инструкция `Dim` или одна из других инструкций описания (`Public`, `Private` или `Static`). Переменная, которая ссылается на объект, должна иметь тип `Variant`, `Object`, или тип определенного объекта. Например, возможны следующие описания:

```
Dim MCLUtil As Object  
Dim bModuleInitialized As Boolean  
Dim fourierclass As Object
```

Если объектная переменная используется без предварительного описания, она по умолчанию приобретает тип данных `Variant`.

Имеется возможность описать объектную переменную с типом данных `Object` в том случае, если определенный объектный тип не известен до выполнения процедуры. Тип данных `Object` позволяет создать универсальную ссылку на любой объект.

Если определенный объектный тип известен, следует описать объектную переменную с этим объектным типом. Например, если используемое приложение содержит объектный тип `Sample`, возможно описание переменной для этого объекта с помощью одной из следующих инструкций:

```
Dim MyObject As Object ' Описывает объект как универсальный  
Dim MyObject As Sample ' Описывает объект только с типом Sample
```

Описание определенных объектных типов обеспечивает автоматическую проверку типа данных, более быстрое выполнение и улучшает читабельность текста программы.

Присвоение объекта объектной переменной. Для присвоения объекта объектной переменной применяется инструкция `Set`. Имеется возможность присвоить объектное выражение или `Nothing`. Например допустимы следующие присвоения объектной переменной:

```
Set MyObject = YourObject ' Присваивает ссылку на объект.  
Set MyObject = Nothing    ' Удаляет ссылку на объект.
```

Можно комбинировать описание объектной переменной с присваиванием ей объекта с помощью оператора `New` или функции `CreateObject` в инструкции `Set`. Например:

```
Set MyObject = New Object ' Создать и присвоить
```

Задание для объектной переменной значения `Nothing` прекращает сопоставление этой переменной с каким-либо определенным объектом. Это предотвращает случайное изменение объекта при изменении переменной. Объектная переменная всегда имеет значение `Nothing` после закрытия объекта, с которым она сопоставляется, поэтому легко проверить, указывает ли объектная переменная на реальный объект. Например, в процедуре инициализации:

```
Private Sub InitModule()  
    If Not bModuleInitialized Then  
        If MCLUtil Is Nothing Then  
            Set MCLUtil = CreateObject("MWComUtil.MWUtil")  
        End If  
        Call MCLUtil.MWInitApplication(Application)  
        bModuleInitialized = True  
    Exit Sub  
End If  
End Sub
```

Описание объектной переменной для программирования объектов. Когда приложение используется для управления объектами из другого приложения, необходимо создать ссылку на библиотеку типов второго приложения. Когда ссылка определена, имеется возможность описать объектные переменные с наиболее подходящим для них типом. Например, если при работе в Microsoft Access определяется ссылка на библиотеку типов Microsoft Excel, то внутри Microsoft Access можно описать переменную типа `Worksheet`, чтобы она представляла объект `Worksheet Microsoft Excel`.

Если для управления объектами Microsoft Access используется другое приложение, то, как правило, объектные переменные описываются с наиболее подходящим для них типом. Возможно также использование ключевого слова `New` для автоматического создания нового экземпляра объекта. Однако необходимо указать, что это объект Microsoft Access. Например, если описывается объектная переменная, представляющая форму Microsoft Access внутри Microsoft Visual Basic, необходимо различать объект `Form Microsoft Access` и объект `Form Visual Basic`.

Имя библиотеки типов включается в описание переменной, как показано в следующем примере:

```
Dim frmOrders As New Access.Form
```

Программирование объектов. Программирование объектов (ранее программирование OLE) является свойством модели COM (Component Object Model), стандартной технологии, которая используется приложениями, чтобы предоставить свои объекты в распоряжение средств разработки, макроязыков и других приложений, поддерживающих программирование объектов. Например, приложение для работы с электронными таблицами может предоставлять для использования лист, диаграмму, ячейку или диапазон ячеек в качестве различных типов объектов.

Если приложение поддерживает программирование объектов, предоставляемые им объекты доступны из языка VB. Visual Basic позволяет проводить обработку этих объектов с помощью методов этих объектов или с помощью чтения или установки свойств этих объектов. Например, если был создан программируемый объект по имени MyObj, для управления этим объектом можно использовать следующую программу:

```
MyObj.Insert "Всем привет." ' Размещает текст.
```

```
MyObj.Bold = True ' Форматирует текст.
```

```
MyObj.SaveAs "C:\WORDPROC\DOCS\TESTOBJ.DOC" ' Сохраняет объект.
```

Следующие функции позволяют получить доступ к программируемому объекту:

- CreateObject – создает новый объект указанного типа;
- GetObject – загружает объект из файла.

Процедуры VBA. Возможны *процедуры* двух типов: Sub и Function. Все процедуры по умолчанию являются общими, за исключением процедур обработки событий. Когда Visual Basic создает процедуру обработки события, перед ее описанием автоматически вставляется ключевое слово Private. Все другие процедуры, не являющиеся общими, должны быть описаны явным образом с ключевым словом Private.

Процедура Sub. Процедура Sub представляет собой последовательность инструкций языка Visual Basic, ограниченных инструкциями Sub и End Sub, которая выполняет действия, но не возвращает значение. Процедура Sub может получать аргументы, как например константы, переменные, или выражения, передаваемые ей вызывающей процедурой. Если процедура Sub не имеет аргументов, инструкция Sub должна содержать пустые скобки. Например,

```
Private Sub InitModule()
```

```
    If Not bModuleInitialized Then
```

```
        On Error GoTo Handle_Error
```

```
        If MCLUtil Is Nothing Then
```

```
            Set MCLUtil = CreateObject("MWComUtil.MWUtil")
```

```
        End If
```

```
        Call MCLUtil.MWInitApplication(Application)
```

```
        bModuleInitialized = True
```

```
    Exit Sub
```



```
Handle_Error:
    bModuleInitialized = False
End If
End Sub
```

Процедура Function. Она представляет собой последовательность инструкций языка Visual Basic, ограниченных инструкциями Function и End Function. Процедура Function подобна процедуре Sub, однако в отличие от последней она возвращает значения. Процедура Function может получать аргументы, как например константы, переменные, или выражения, передаваемые ей вызывающей процедурой. Если процедура Function не имеет аргументов, ее инструкция Function должна содержать пустые скобки. Возврат значения осуществляется путем его присвоения имени функции в одной или нескольких инструкциях процедуры. Например,

```
Function computefft(Optional data As Variant, _
                    Optional interval As Variant) As Variant
Dim fftData, freq, powerSpect As Variant

Call InitModule
If fourierclass Is Nothing Then
    Set fourierclass = CreateObject( "fourier.fourierclass.1_0")
End If
    Call fourierclass.computefft(1, fftData, freq, powerSpect, data,
interval)
    computefft = fftData
    Exit Function
End Function
```

Массивы параметров. Для передачи аргументов в процедуру может использоваться массив параметров. При описании процедуры не требуется указывать число элементов такого массива. Для обозначения массива параметров используется ключевое слово **ParamArray**. Такой массив описывается как массив типа Variant и всегда представляет последние элементы из списка аргументов в описании процедуры.

Создание компонентов для .NET при помощи .NET Builder

5.1. Среда разработки Microsoft .NET	342
5.2. Основы языка C#	349
5.3. Введение в .NET Builder	378
5.4. Создание консольный приложений	388
5.5. Некоторые вопросы программирования с компонентами .NET Builder ...	403
5.6. Среда разработки Visual Studio 2005	416
5.7. Программирование на Visual Studio 2005 с использованием математических процедур MATLAB	421

Пакет MATLAB® Builder для .NET (называемый также .NET Builder) есть расширение пакета MATLAB® Compiler. Он используется для преобразования функций MATLAB в один или более классов .NET, которые составляют компонент .NET, или пакет. Каждая функция MATLAB преобразуется в метод некоторого класса и может быть вызвана из приложения .NET. Приложения, использующие методы, созданные при помощи .NET Builder, при своей работе не требуют установленной системы MATLAB. Однако должна быть установлена MCR – среда исполнения для компонентов MATLAB® (MATLAB Component Runtime).

В данной главе мы рассмотрим создание математических компонентов для .NET при помощи пакета MATLAB® Builder для .NET и приложений, использующих эти компоненты. Сначала мы рассмотрим основы среды разработки Microsoft .NET и элементы языка C#, специально созданного для программирования в среде .Net. Затем изучим возможности .NET Builder, а в конце рассмотрим создание Windows-приложений на Visual Studio 2005, в которых используются математические функции, созданные из m-функций MATLAB® используя .NET Builder.

5.1. Среда разработки Microsoft .NET

В данном параграфе рассмотрим среду разработки Microsoft .NET. Среда разработки .NET (читается Dot Net) является открытой языковой средой. Это означает, что наряду с языками программирования, включенными в среду фирмой Microsoft – Visual C++ .Net, Visual C# .Net, J# .Net, Visual Basic .Net, в нее могут добавляться любые языки программирования, при соблюдении определенных условий. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки Visual Studio .Net, должны использовать единый каркас – .Net Framework. Благодаря этому достигаются многие свойства: легкость использования компонентов, разработанных на различных языках; возможность разработки нескольких частей одного приложения на разных языках; возможность бесшовной отладки такого приложения; возможность писать класс на одном языке, а его потомков – на других языках. Единый каркас приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства.

5.1.1. Основные элементы платформы Microsoft .NET

Платформа – в контексте информационных технологий – это среда, обеспечивающая выполнение программного кода. Платформа Microsoft .NET обеспечивает среду выполнения программ и определяет особенности разработки и выполнения программного кода на ПК. Она имеет средства организации взаимодействия с операционной системой и прикладными программами, методы доступа к базам данных, средства поддержки сетевых приложений, языки программирования и множество базовых классов.

Платформа Microsoft .NET включает в себя следующие основные компоненты: общезыковую среду выполнения CLR (Common Language Runtime), стандартную систему типов CTS (Common Type System) и общую спецификацию языков программирования CLS (Common Language Specification). Поскольку .NET Framework является общей средой выполнения многих программ и является надстройкой над операционной системой, то она устанавливается в каталог Windows, например: C:\WINDOWS\Microsoft.NET\Framework\v3.0\ . Рассмотрим эти основные компоненты немного позднее, а теперь введем ряд новых понятий, свойственных платформе .NET.

Новые понятия

Тип в .NET это – общий термин, относящийся к классам, структурам, интерфейсам, перечислениям и т.п.

Управляемый код (managed code) – код, предназначенный для выполнения в среде .NET Framework. Код C#, Visual Basic, и JScript является управляемым по умолчанию. Одной из особенностей управляемого кода является наличие механизмов, которые позволяют работать с управляемыми данными. Вообще, все, что предназначено для выполнения в среде .NET Framework, имеет прилагательное «управляемый».

Управляемые данные – объекты, которые в ходе выполнения модуля кода размещаются в управляемой памяти (в управляемой куче) и уничтожаются сборщиком мусора. Данные C#, Visual Basic и JScript .NET являются управляемыми по умолчанию.

Родные ресурсы (native resources). Собственные ресурсы, которые существуют вне управления общезыковой среды выполнения CLR, например, встроенные числовые типы данных C#.

Управляемый исполняемый модуль. Независимо от компилятора (и входного языка) результатом трансляции .NET-приложения является *управляемый исполняемый модуль*. Это переносимый исполняемый (PE – Portable Executable) файл Windows. Элементы управляемого модуля:

- заголовок PE – показывает тип файла (например, DLL), содержит временную метку (время сборки файла), содержит сведения о выполняемом коде;
- заголовок CLR – содержит информацию для среды выполнения модуля (версию требуемой среды исполнения, характеристики метаданных, ресурсов и т.д.). Собственно, эта информация делает модуль управляемым;
- метаданные – таблицы метаданных: типы, определенные в исходном коде и типы, на которые имеются в коде ссылки;
- IL – собственно код, который создается компилятором при компиляции исходного кода. На основе IL в среде выполнения впоследствии формируется множество команд процессора.

Управляемый модуль содержит управляемый код. Управляемые модули объединяются в, так называемые, сборки. Сборка является логической группировкой одного или нескольких управляемых модулей или файлов ресурсов. Управляемые модули в составе сборок исполняются в среде выполнения.

Промежуточный язык MSIL (Microsoft Intermediate Language) платформы Microsoft .NET. Исходные тексты программ для .NET-приложений пишутся на языках программирования, соответствующих общезыковым правилам CLS. Программы на этих языках могут транслироваться в промежуточный код на MSIL или, короче, на IL. Благодаря соответствию общезыковым правилам CLS, в результате трансляции программного кода, написанного на разных языках, получается совместимый IL-код. Языки, для которых реализован перевод на IL – это: Visual Basic, Visual C++, Visual C# и еще много других языков.

Сборка (Assembly). Результат компиляции исходного текста программы представляется транслятором в виде сборки – файла на промежуточном языке IL. Сборка может быть двух видов:

- исполняемое приложение, файл с расширением .exe;
- библиотека, файл с расширением .dll, предназначен для использования составе какого-либо приложения.

При этом ничего общего с обычными (старого образца) исполняемыми приложениями и библиотечными модулями сборка не имеет. Содержимое бинарных файлов .NET – это платформенно независимый код на промежуточном языке Microsoft IL, набор инструкций на IL. Компилятор .NET генерирует код IL вне зависимости от того, на каком языке был написан исходный код (C++, C#, VB и т.п.). Сборка компилируется в соответствующий платформе исполняемый файл только тогда, когда к ней происходит обращение. Компиляцию выполняет JIT-компилятор (Just-In-Time compiler), который так называется потому, что вызывается по мере необходимости. JIT-компилятор входит в среду выполнения .NET. Откомпилированные платформенно-зависимые инструкции помещаются в кэш-память.

Сборка, хотя и представляет код на промежуточном языке IL, она представлена в бинарном виде. Для просмотра содержимого любой сборки имеется утилита ildasm.exe, которая находится в каталоге C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin.

Метаданные. Кроме кода на промежуточном языке, бинарный файл .NET (сборка) содержит метаданные, которые подробно описывают все типы, используемые в этом бинарном модуле: описание классов, интерфейсов, методов, свойств и событий класса. Метаданные содержат также описание и самой сборки: информацию о текущей версии, ограничения по безопасности, поддержка естественных языков, а также список всех других сборок, которые требуются для выполнения данной сборки. Эта часть метаданных называется манифестом (manifest).

Декларация сборки (Manifest) – составная часть сборки. Это набор таблиц метаданных, который:

- идентифицирует сборку в виде текстового имени, ее версию и цифровую сигнатуру (если сборка распределяется среди приложений);
- определяет входящие в состав файлы (по имени и хэшу);
- указывает типы и ресурсы, существующие в сборке, включая описание тех, которые экспортируются из сборки;

- перечисляет зависимости от других сборок;
- указывает набор прав, необходимых сборке для корректной работы.

Эта информация используется в период выполнения для поддержки корректной работы приложения.

Таким образом, на основе входного кода транслятор строит модуль на IL, манифест и формирует сборку. В дальнейшем сборка либо может быть выполнена после JIT-компиляции, либо может быть использована в составе других программ.

Сборка может состоять как из одного бинарного файла (single file assembly), так и из нескольких (multifile assembly). В последнем случае каждый бинарный файл сборки называется модулем, манифест тогда содержится в одном из модулей.

Сборка – это основной строительный блок приложения в .NET Framework. Аналог байт-кода Java.

Пространство имен (Namespace). Среда .NET Framework располагает большим набором функций. Каждая из них является членом какого-либо класса. Классы группируются по пространствам имен. Это означает, что в общем случае имя класса может иметь сложную структуру – состоять из последовательности имен, разделенных между собой точками. Последнее имя в этой последовательности и является именем класса. Классы, имена которых различаются лишь последними членами (собственно именами классов) последовательностей, считаются принадлежащими одному пространству имен. Средством «навигации» по пространствам имен, а точнее, средством, которое позволяет сокращать имена классов, является оператор

```
using <Имя.Пространства.Имен>;
```

Наиболее часто используемое пространство имен – это System. Соответствующая сборка, которая содержит классы, сгруппированные в пространстве имен System, располагается в файле mscorlib.dll.

Использование пространства имен существенно упрощает кодирование. Если не использовать пространство имен, то при обращении к функции некоторого класса некоторого пространства имен, пришлось бы указывать полное имя функции. Например, корректное обращение к функции WriteLine(...) – члену класса Console пространства имен System, выглядело бы следующим образом:

```
System.Console.WriteLine("текст для вывода!");
```

В процессе построения сборки транслятор должен знать расположение сборок с заявленными для использования пространствами имен. Расположение некоторых сборок (например, mscorlib.dll) известно изначально. Расположение всех остальных требуемых сборок указывается явно. Непосредственно в Visual Studio, при работе над проектом нужно открыть окно Solution Explorer, выбрать пункт References, далее Add Reference... – там надо задать или выбрать соответствующий .DLL- или .EXE-файл.

Пространства имен C# аналогичны пакетам Java.

Сборка мусора – механизм, позволяющий среде исполнения определить, когда объект становится недоступен в управляемой памяти программы. При сборке

мусора управляемая память освобождается. Для разработчика приложения наличие механизма сборки мусора означает, что он больше не должен заботиться об освобождении памяти. Однако это может потребовать изменения в стиле программирования, например, особое внимание следует уделять процедуре освобождения системных ресурсов. Необходимо реализовать методы, освобождающие системные ресурсы, находящиеся под управлением приложения.

Хэш-код – целочисленное значение, идентифицирующее конкретный экземпляр объекта данного типа

Теперь рассмотрим основные компоненты среды выполнения .NET Framework.

5.1.2. Среда выполнения .NET Framework

Как известно, программа, созданная на каком-либо языке, требует для своего выполнения определенный набор служб – *среду выполнения*. Например, программа, созданная на Java, требует для своей работы большой набор файлов, входящих в состав виртуальной машины Java. Среду выполнения программы на C++, использующей математические библиотеки MATLAB, составляют библиотеки математических процедур (набор файлов dll). Своя среда выполнения требуется и приложениям .NET. Она является единой для всех приложений, написанных на любых языках .NET. Среда выполнения .NET состоит из двух основных компонентов. Это общезыковая среда выполнения .NET (Common Language Runtime, CLR) и библиотека базовых классов.

Общезыковая среда выполнения CLR (Common Language Runtime). Виртуальная машина. Обеспечивает выполнение сборки. Это основной компонент .NET Framework, ядро среды выполнения, которое реализовано в виде библиотеки mscorlib.dll. При обращении к приложению .NET, библиотека mscorlib.dll автоматически загружается в память и выполняет работу по выполнению сборки данного приложения. Ядро среды выполнения .NET исполняет множество задач, из которых главными являются следующие: управление процессом загрузки в память сборки данного приложения и анализ метаданных сборки; компиляция промежуточного кода IL в платформенно-зависимые инструкции; выполнение приложения.

Библиотека базовых классов FCL (.NET Framework Class Library). Это второй главный компонент среды выполнения. Это библиотека классов, интерфейсов и системы типов, которые включаются в состав платформы Microsoft .NET. Библиотека обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений. Библиотека может использоваться всеми .NET-приложениями, независимо от используемого при разработке языка программирования. Библиотека состоит из множества сборок, главной из которых является mscorlib.dll. В библиотеке базовых классов содержится огромное количество типов для решения распространенных задач при создании приложения. Для языка программирования C# используется библиотека базовых типов среды .NET. Для организации типов (классов, структур, интерфейсов, встроенных типов данных и т.п.) в этой библиотеке используется концеп-

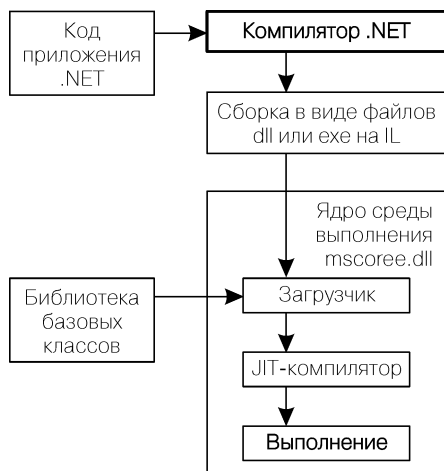


Рис. 5.1.1. Схема выполнения .NET-приложения в среде CLR

ция пространства имен. Вне зависимости от языка программирования, доступ к определенным классам обеспечивается за счет их группировки в рамках общих пространств имен.

5.1.3. Стандартная система типов

Стандартная Система Типов CTS (Common Type System). Это формальная спецификация, которая определяет, как какой-либо тип (класс, структура, интерфейс, тип данных) должен быть определен для его правильного восприятия средой выполнения .NET. Поддерживается всеми языками платформы. Стандартная система типов CTS является важной частью среды выполнения, определяет структуру синтаксических конструкций, способы объявления, использования и применения общих типов среды выполнения. В CTS сосредоточена основная информация о системе общих предопределенных типов, об их использовании и управлении (правилах преобразования значений). CTS играет важную роль в деле интеграции разноязыких управляемых приложений. Для организации системы типов в единую группу служит пространство имен.

Классы CTS. Класс – это набор свойств, методов и событий, объединенных в единое целое. В CTS предусмотрены абстрактные члены классов. Множественное наследование в CTS запрещено. Закрытые классы не могут быть базовыми для других классов. В CTS может быть любое количество интерфейсов. Абстрактные классы создаются как базовые для других классов. Для каждого класса должен быть определен атрибут области видимости.

Структуры CTS. В CTS предусмотрены структуры – это упрощенные разновидности классов. Все CTS-совместимые структуры произведены от единого базового класса System.Value.Type. Этот базовый класс определяет структуру как

тип данных для работы только со значениями, но не со ссылками. Структуры могут иметь любое количество конструкторов с параметрами, с помощью которых можно определить значение любого поля структуры в момент создания объекта. В структуре может быть любое количество интерфейсов. Структуры являются закрытыми.

Интерфейсы CTS. Это наборы абстрактных методов, свойств и определений событий. При создании интерфейса на .NET-совместимом языке программирования можно произвести интерфейс сразу от нескольких базовых интерфейсов.

Члены типов CTS. Член (member) – это либо метод, либо свойство, либо поле, либо событие. В классах и структурах может быть любое количество членов. Для каждого члена существует определенный набор характеристик, типа области видимости, абстрактности и статичности.

Перечисления в CTS. Это удобная программная конструкция, которая позволяет ассоциировать с именами определенные целые числа (имя-значение). В CTS все перечисления являются производными от единственного базового класса System.Enum.

Делегаты в CTS. Это эквивалент указателя на функцию в С. Делегаты используются, когда необходимо чтобы одна сущность передала вызов другой сущности. Делегаты используются в технологии обработки событий .NET.

Встроенные типы данных CTS. В .NET предусмотрен богатый набор встроенных типов данных. Этот набор единый для всех языком программирования .NET. При программировании на выбранном .NET-совместимом языке программирования можно использовать имена простых типов этого языка программирования, поскольку существует соответствие между именами простых типов в конкретном языке и именами типов, объявленных в библиотеке базовых классов, см. напр. табл. 5.1.1.

Таблица 5.1.1. Встроенные типы данных CTS

Встроенный тип данных .NET	Название в C#	Название в Visual Basic .NET	Диапазон
System.SByte	sbyte	нет	8-разрядное со знаком, -128 : 127
System.Byte	byte	Byte	8-разрядное без знака, 0 : 255
System.Int16	short	Short	-32768 : 32767
System.UInt16	ushort	нет	0 : 65535
System.Int32	int	Integer	-2147483648 : 2147483647
System.UInt32	uint	нет	0 : 4294967295
System.Int64	long	Long	9223372036854775808 : 9223372036854775807
System.UInt64	ulong	нет	0 : 18446744073709551615
System.Char	char	Char	16 разрядов. Символ UNICODE
System.Single	float	Single	32 разряда. Стандарт IEEE
System.Double	double	Double	64 разряда. Стандарт IEEE
System.Decimal	decimal	Decimal	128-разрядное значение повышенной точности с плавающей запятой

Таблица 5.1.1. Встроенные типы данных CTS (окончание)

Встроенный тип данных .NET	Название в C#	Название в Visual Basic .NET	Диапазон
System.Boolean	bool	Boolean	true или false
System.Object	object	Object	
System.String	string	String	

5.1.4. Общая спецификация языков программирования

Общая спецификация языков программирования CLS (Common Language Specification) – это набор правил, которыми необходимо руководствоваться для создания приложений и библиотек для среды .NET Framework. CLS – это основа межязыкового взаимодействия в рамках платформы Microsoft .NET. Если следовать правилам CLS, то программные модули, написанные на разных языках программирования, будут нормально взаимодействовать между собой. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки программирования, включенные в состав Visual Studio соответствуют CLS (Visual C#, Visual Basic, Visual C++, Visual J#) и могут интегрироваться друг с другом. Правила CLS относятся только к тем частям программы, которые предназначены для взаимодействия за пределами сборки, в которой они определены.

Для среды выполнения CLR все сборки одинаковы, независимо от того, на каких языках программирования они были написаны. Главное – чтобы они соответствовали CLS.

5.2. Основы языка C#

В данном параграфе рассмотрим очень кратко особенности языка программирования C#. Более полную информацию можно найти в учебниках, например, [Ка], [Ла], [Ма], [Тро], [Фа]. Язык C# создавался параллельно с каркасом .Net Framework и в полной мере учитывает все его возможности – как FCL, так и CLR. Язык C# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами. Язык C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от C++ к C#. Сохранив основные черты C/C++, язык C# стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства C++, как указатели, адресация, разыменование, адресная арифметика. Благодаря каркасу .Net Framework, ставшему надстройкой над операционной системой,

программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных.

5.2.1. Элементы синтаксиса языка C#

В этом разделе рассмотрим основные правила написания кода программы на языке C#.

Алфавит и слова C#

Алфавит языка программирования C# составляют символы таблицы кодов ASCII. Алфавит C# служит для построения слов, которые разбиваются на пять типов:

- идентификаторы;
- ключевые слова;
- знаки (символы) операций;
- литералы;
- разделители.

Правила образования идентификаторов. Идентификаторы – это имена переменных, типов данных, функций. Первым символом идентификатора C# может быть только буква. Следующими символами идентификатора могут быть буквы, цифры и знак подчеркивания. Использование идентификаторов, которые начинаются с символа подчеркивания, нежелательно.

Язык C# различает строчные и прописные буквы. Свои имена можно записывать как угодно, но нужно учитывать следующие правила: имена классов начинаются с прописной буквы; если имя составлено из несколько слов, то каждое слово начинается с прописной буквы.

Имена переменных, классов, методов и других объектов могут быть простыми (идентификаторы) и составными. Составное имя – это несколько идентификаторов, разделенных точками, без пробелов, например, имя пространства имен System.Collections.Generic.

Ключевые слова. Часть идентификаторов C# входит в фиксированный словарь ключевых слов. Их нельзя использовать для образования классов, функций, переменных. Примеры ключевых слов: abstract, as, base, bool, break, byte, case, catch, char, checked, class, const.

Литералы. В C# существует четыре типа литералов: целочисленный литерал; вещественный литерал; символьный литерал; строковый литерал.

Целочисленный литерал служит для записи целочисленных значений и является соответствующей последовательностью цифр (возможно, со знаком «минус»). Целочисленный литерал, начинающийся со знака 0, воспринимается как восьмеричное целое. В этом случае цифры 8 и 9 не должны встречаться среди составляющих литерал символов. Целочисленный литерал, начинающийся с 0x или 0X, воспринимается как шестнадцатеричное целое.

Вещественный литерал служит для отображения вещественных значений. Он фиксирует запись соответствующего значения в обычной десятичной или научной нотациях. В научной нотации мантисса отделяется от порядка литерой E (или e). Непосредственно за литералом может располагаться один из двух специальных суффиксов: F (или f – float) и L (или l – long).

Символьный литерал представляет собой последовательность одной или нескольких литер, заключенных в одинарные кавычки. Символьный литерал служит для представления литер в одном из форматов представления: обычном, восьмеричном и шестнадцатеричном. Значением символьного литерала является соответствующее значение ASCII кода.

Строковые литералы являются последовательностью литер в одном из возможных форматов представления, заключенных в двойные кавычки.

Структура программы C#

Программа представляет собой текстовый файл, состоящий из операторов языка и комментариев.

Инструкции (операторы) программы. Каждая инструкция обязательно должна заканчиваться точкой с запятой (;). Инструкции C# рассматриваются в соответствии с порядком их записи в тексте программы. Компилятор начинает рассматривать код программы с первой строки и заканчивает концом файла. Перенос части оператора на другую строку возможен в любом месте, где может быть пробел (если это не строковое выражение). Несколько подряд идущих пробелов считаются за один пробел.

Разделители. В языке C# пробелы, знаки табуляции и переход на новую строку рассматриваются как разделители. В инструкциях языка C# лишние разделители игнорируются. Исключение состоит в том, что пробелы в пределах строкового выражения не игнорируются. Если вы напишете:

```
Console.WriteLine("Здравствуй Мир !");
```

каждый пробел между словами «Здравствуй», «Мир» и знаком «!» будет обрабатываться как отдельный символ строки.

Комментарии. Они в тексте программы вводятся обычным образом. За двумя наклонными чертами подряд //, без пробела между ними, начинается комментарий, который продолжается до конца строки. За наклонной чертой и звездочкой /* начинается комментарий, который может занимать несколько строк, до звездочки и наклонной черты */.

Частью комментария являются информационные XML-теги. Если в нужном месте (перед объявлением класса, метода) набрать подряд три символа косой черты, то это распознается как тэг документирования, так что останется только дополнить его соответствующей информацией. Этот тэг распознается специальным инструментарием, строящим XML-отчет проекта. Идея самодокументируемых программных проектов, у которых документация является неотъемлемой частью, является важной составляющей стиля программирования на C#. Заметим, что кроме тега <summary> возможны и другие тэги, включаемые в отчеты. Некоторые тэги добавляются автоматически. Покажем это на следующем примере кода, ко-

торый создает Мастер создания Windows-приложения на Visual C#. Описанию метода Main (как точки входа приложения) предшествует заданный в строчном комментарии XML-тег <summary>. Информация, которая идет после <summary> автоматически включается в XML-отчет проекта.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Атрибуты. В C# атрибуты – это аннотации, которые могут быть применены к типу (классу, интерфейсу, структуре), члену, сборке или модулю. В вышеприведенном примере кода атрибут [STAThread] предшествует описанию процедуры Main. В данном случае атрибут [STAThread] (Single Thread Apartment) задает однопоточную модель выполнения. Заметим, что атрибуты необязательны и если Вы нечетко представляете, каков смысл однопоточной модели, то атрибут можно удалить.

Так же, как и тэги документирования, атрибуты распознаются специальным инструментарием и становятся частью метаданных. Атрибуты могут быть как стандартными, так и заданными пользователем. Стандартные атрибуты используются CLR и влияют на то, как она будет выполнять проект. Приведем несколько встроенных атрибутов: CLSCompliant – определяет совместимость всех типов сборки с CLS; Serializable – помечает класс или структуру как сериализуемые (доступные для сохранения на диске и восстановления с него).

Переменные и константы C#

Переменные могут иметь значения, которыми они проинициализированы, эти значения могут быть изменены программно. Чтобы создать переменную, нужно задать тип переменной и имя. Можно инициализировать переменную во время ее объявления или присвоить ей новое значение во время выполнения программы. C# требует, чтобы перед использованием переменная должна быть инициализирована.

Константы. Это переменные, значения которых не могут меняться. Константы объявляются с дополнительным спецификатором `const`. Они требуют непосредственной инициализации. Существует три разновидности констант: литералы, символьные константы и перечисления.

Примером литеральной константы может служить значение 100. Значение этой константы 100 не меняется – это всегда 100.

Символьные константы устанавливают имя для некоторого постоянного значения. Например, символьная константа `pi` типа `double`:

```
const double pi = 3.1415926535897932384626433832795;
```

Константа обязательно должна быть инициализирована и ее значение не может быть изменено во время выполнения программы.

Перечисления – это особый тип значений, который состоит из набора именованных констант. Перечисление объявляется с помощью ключевого слова `enum`, идентифицируется по имени и представляет собой непустой список неизменяемых именованных значений целого типа. Первое значение в перечислении по умолчанию инициализируется нулем. Каждое последующее значение отличается от предыдущего по крайней мере на единицу, если объявление значения не содержит явного дополнительного присвоения нового значения. Примеры объявления перечислений:

```
enum En1 { One, Two, Three };  
enum Colors { Red = 1, Green = 2, Blue = 4, Yellow = 8 };
```

Первое перечисление создает целочисленные типы со значениями: `One=0`, `Two=1`, `Three=2`. Такое присвоение производится по умолчанию. Во второй строке явно заданы значения элементов перечисления.

Обращение к элементу перечисления осуществляется посредством выражения, состоящего из имени класса перечисления, операции доступа к элементу перечисления «`.`», имени элемента перечисления. В следующем примере переменная `xVal` инициализируется значением перечисления:

```
int xVal = Colors.Red;
```

Строковые константы. Для объявления в программе константной строки необходимо заключить содержимое строки в двойные кавычки ("My string"). Это можно делать практически в любом месте программы: в передаче параметров функции, в инициализации переменных.

Объявление переменных. Область видимости и время жизни

Объявление – это предложение языка C#, которое используется для объявления переменных и констант, классов и структур, перечислений и элементов перечислений, конструкторов классов и методов. Объявление используется непосредственно в теле класса для объявления членов класса (в этом случае объявлению может предшествовать спецификатор доступа) или для объявления переменных в конструкторах и методах класса. Переменные можно объявлять в любом месте блока. Место объявления переменной в буквальном смысле соответствует месту

ее создания. Обращение к переменной или константе до места ее объявления лишено смысла.

Предложение объявления предполагает (возможное) наличие различных спецификаторов, указание имени типа, имени объекта и выражения инициализации.

Имена в программе повсюду, и главная проблема заключается в том, чтобы не допустить неоднозначности при обращении из разных мест программы к классам, членам классов, перечислениям, переменным и константам. Избежать конфликта имен в C# позволяет такая синтаксическая конструкция, как блок операторов. Блок – это множество предложений (возможно пустое), заключенное в фигурные скобки. Различается несколько категорий блоков:

- тело класса (структуры), это место объявления членов класса;
- тело метода, это место расположения операторов метода;
- блок в теле метода.

Новый блок – это новая область видимости. Объекты, объявляемые во внутренних блоках, не видны во внешних блоках. Объекты, объявленные в методе и во внешних блоках, видны и во внутренних блоках. Одноименные объекты во вложенных областях конфликтуют.

Объекты, объявляемые в блоках одного уровня вложенности в методе, не видны друг для друга. Конфликта имен не происходит.

Имена данных, членов класса, не конфликтуют с одноименными переменными, объявляемыми в теле методов, поскольку в теле метода обращение к членам класса обеспечивается выражениями с операцией доступа, и никакого конфликта в принципе быть не может.

Объявления классов вводятся ключевым словом `class`. Структура объявляется ключевым словом `struct`.

Приведем примеры объявления и инициализации переменных. Во-первых отметим эквивалентные формы записи операторов определения переменных элементарных значащих типов:

```
int a;           // объявление целой переменной в C#
System.Int32 a;  // объявление целой переменной как типа .NET
```

Следует учитывать одно важное обстоятельство. CLR не допускает использования в выражениях неинициализированных локальных переменных. В C# к таковым относятся переменные, объявленные в теле метода. Так что при разработке алгоритма следует обращать на это особое внимание.

5.2.2. Система типов

Тип в .NET это – общий термин, относящийся к классам, структурам, интерфейсам, перечислениям и т.п. В C# нужно объявлять тип каждого объекта, например, целые числа, числа с плавающей точкой, строки, форма, кнопки, и т. д. Тип объекта указывает компилятору размер объекта и его свойства (например, форма может быть видима и невидима, и т.д.).

Типы данных принято разделять на встроенные и типы, определенные программистом. Встроенные типы изначально принадлежат языку программиро-

вания и составляют его основу. Типы данных разделяются также на статические и динамические. Для данных статического типа память отводится в момент объявления, требуемый размер данных (памяти) известен при их объявлении. Для данных динамического типа размер данных в момент объявления обычно неизвестен и память им выделяется динамически по запросу в процессе выполнения программы. C# также подразделяет типы на две другие категории: значимые и ссылочные.

Значимые и ссылочные типы

Основное различие между ними – это способ, которым их значения сохраняются в памяти. Значимые типы сохраняют свое фактическое значение в стеке. Ссылочные типы хранят в стеке лишь адрес объекта, а сам объект сохраняется в куче. Куча – это основная память программ, доступ к которой осуществляется намного медленнее чем к стеку.

Особенности стека и кучи. *Стек* – это структура данных, которая сохраняет элементы по принципу «первым пришел, последним ушел». Стек относится к области памяти, поддерживаемой процессором, в которой сохраняются локальные переменные. Доступ к стеку во много раз быстрее, чем к общей области памяти, поэтому использование стека для хранения данных ускоряет работу программы. В C# значимые типы (например, целые числа) располагаются в стеке: для их значений зарезервирована область в стеке, и доступ к ней осуществляется по названию переменной. Ссылочные типы (например, объекты) располагаются в куче. *Куча* – это оперативная память компьютера. Доступ к ней осуществляется медленнее, чем к стеку. Когда объект располагается в куче, то переменная хранит лишь адрес объекта. Этот адрес хранится в стеке. По адресу программа имеет доступ к самому объекту, все данные которого сохраняются в общем куске памяти (куче). Сборщик мусора уничтожает объекты, располагающиеся в стеке, каждый раз, когда соответствующая переменная выходит за область видимости. Таким образом, если вы объявляете локальную переменную в пределах функции, то объект будет помечен как объект для сборки мусора. И он будет удален из памяти после завершения работы функции. Объекты в куче тоже очищаются сборщиком мусора, после того как конечная ссылка на них будет разрушена.

В языке C# жестко определено, какие типы относятся к ссылочным, а какие – к значимым.

Значимые типы. К *значимым типам* относятся: логический, арифметический, структуры, перечисление. Структуры C# представляют частный случай класса. Определив свой класс как структуру, программист получает возможность отнести класс к значимым типам, что иногда бывает крайне полезно.

Ссылочные типы. Массивы, строки и классы относятся к *ссылочным типам*. В C# массивы рассматриваются как динамические, их размер может определяться на этапе вычислений, а не в момент трансляции. Строки в C# также рассматриваются как динамические переменные, длина которых может изменяться. Поэтому строки и массивы относятся к ссылочным типам, требующим распределения памяти в куче.

Отличие значащих и ссылочных типов. Для значащих типов память выделяется в программном стеке и имя переменной прямо связывается с этой памятью. Если инициализируется другая переменная того же типа, то ей выделяется другой участок памяти. При выполнении операции присваивания копируется значение. Например, в следующем фрагменте кода переменным *x* и *y* отведены разные участки памяти, они могут принимать независимые значения

```
int x = 100; // создание и инициализация переменной x
int y = x;   // создание и инициализация переменной y
x = 200;     // другое значение переменной x не вызывает изменения y
```

Для ссылочных типов значением служит ссылка на адрес участка кучи, в которой расположена переменная. Если создается другая переменная того же ссылочного типа, она может получить ссылку на тот же участок кучи, что и первая. Например, при выполнении операции присваивания копируется ссылка. Поэтому изменение одной ссылочной переменной приводит к изменению другой. Таким образом, переменные одного ссылочного типа могут быть зависимыми.

При создании экземпляров классов (ссылочный тип), поля которых относятся к значащим типам, эти поля по умолчанию инициализируются нулевыми значениями. При объявлении переменной значащего типа, она не инициализируется.

Системные встроенные типы

Для C# система встроенных типов аналогична встроенным типам других языков. Имеются арифметический, логический (булев) и символьный типы. Арифметический тип разбивается на подтипы. Допускается организация данных в виде массивов и записей (структур). Внутри арифметического типа допускаются преобразования, есть функции, преобразующие строку в число и обратно. Поскольку язык C# является непосредственным потомком языка C++, то и системы типов этих двух языков близки и совпадают вплоть до названия типов и областей их определения. Однако есть и некоторые отличия, на которые мы в дальнейшем укажем. Простые встроенные типы перечислены в следующей таблице 5.2.1.

Таблица 5.2.1. Встроенные типы данных CTS

Встроенный тип данных .NET	Название в C#	Диапазон	Описание
System.SByte	sbyte	-128 : 127	8-разрядное целое со знаком
System.Byte	byte	0 : 255	8-разрядное целое без знака
System.Int16	short	-32768 : 32767	16-разрядное целое со знаком
System.UInt16	ushort	0 : 65535	16-разрядное целое без знака
System.Int32	int	-2147483648 : 2147483647	32-разрядное целое со знаком

Таблица 5.2.1. Встроенные типы данных CTS (окончание)

Встроенный тип данных .NET	Название в C#	Диапазон	Описание
System.UInt32	uint	0: 4294967295	32-разрядное целое без знака
System.Int64	long	-9223372036854775808 : 9223372036854775807	64-разрядное целое со знаком
System.UInt64	ulong	0: 18446744073709551615	64-разрядное целое без знака
System.Char	char	U+0000 - U+ffff, Целые, от 0 до 65535	16 разрядов. Символ Unicode
System.Single	float	$1.5 \cdot 10^{(-45)}$: $3.4 \cdot 10^{(38)}$	32-разрядное вещественное, 7 значащих цифр
System.Double	double	$5.0 \cdot 10^{(-324)}$: $1.7 \cdot 10^{(308)}$	64-разрядное вещественное, 15-16 значащих цифр
System.Decimal	decimal	$1.0 \cdot 10^{(-28)}$: $7.9 \cdot 10^{(28)}$	128-разрядное значение повышенной точности с плавающей запятой, 28–29 значащих цифр
System.Boolean	bool	true или false	Логический тип
System.Object	object	Все что угодно	Базовый класс для всех типов
System.String	string	Без ограничений	Набор символов Unicode

Встроенные числовые типы данных C# являются производными от типа (структуры) System.Value.Type.

Упаковка и распаковка (boxing и unboxing). Как известно, в Java каждому простому типу соответствует класс, который обертывает значение примитивного типа в объект. Этот объект содержит единственное поле, тип которого является типом соответствующего примитива. Например, класс Double обертывает значение примитивного типа double в объект. Аналогичная конструкция предусмотрена в языке C#. Значащий тип можно преобразовать в ссылочный и наоборот. Эти процедуры называются соответственно упаковкой и распаковкой. При упаковке создается класс-оболочка (wrapper), содержащая значение переменной. В управляемой куче создается новый объект и в него копируются внутренние данные старого объекта из стека. Распаковка должна производиться в соответствующий значащий тип (перед распаковкой производится проверка типов). Упаковка и распаковка показана на следующем примере:

```
int x = 1000;           // создание и инициализация переменной x
object cl_x = x;        // упаковка ее в класс-оболочку cl_x
int y = (int) cl_x;     // распаковка класса-оболочки cl_x в переменную y
```

Приведение типов

В операторах присваивания часто возникает необходимость переменной одного типа присвоить значение выражения другого типа. Тогда нужно учитывать совме-

стимость типов. Будем считать один тип больше другого, если диапазон первого типа шире, чем диапазон второго типа. Два типа называются совместимыми, если выполняется одно из следующих свойств:

- оба типа целые;
- оба типа вещественные;
- один тип – вещественный, а второй – целый, причем вещественный тип выше целого;
- один тип – строка, а второй – символ, причем строковый тип выше символьного.

Приведение типов производится неявно, когда при вычислении выражения встречаются операнды разных, но совместимых типов. При этом происходит повышение меньшего типа операнда и результат будет иметь высший тип операндов. Например, если операнды арифметической операции имеют разные типы (`int` и `double`), то происходит повышение меньшего типа операнда и результат будет иметь высший тип операндов (`double`).

Если такое действие не устраивает, можно выполнить явное приведение типа. Для этого перед выражением или операндом в круглых скобках указывается тип, к которому он приводится. Например, если `b1` и `b2` имеют тип `byte`, а желателен результат типа `short`, то можно использовать код:

```
short k = (short) (b1 + b2) ;
```

Расширяющее преобразование – когда значение одного типа преобразуется к значению другого типа, которое имеет такой же или больший размер. Например, значение, представленное в виде 32-разрядного целого числа со знаком, может быть преобразовано в 64-разрядное целое число со знаком. Расширяющее преобразование считается безопасным, поскольку исходная информация при таком преобразовании не искажается. Однако некоторые расширяющие преобразования типа могут привести к потере точности. Следующая таблица описывает варианты преобразований, которые иногда приводят к потере информации. Это может произойти в следующих случаях:

- `Int32` в `Single`;
- `UInt32` в `Single`;
- `Int64` в `Single`, `Double`;
- `UInt64` в `Single`, `Double`;
- `Decimal` в `Single`, `Double`.

Сужающее преобразование – значение одного типа преобразуется к значению другого типа, которое имеет меньший размер (из 64-разрядного в 32-разрядное). Сужающие преобразования могут приводить к потере информации. Если тип, к которому осуществляется преобразование, не может правильно передать значение источника, то результат преобразования оказывается равен константе `PositiveInfinity` или `NegativeInfinity`.

При этом значение `PositiveInfinity` интерпретируется как результат деления положительного числа на нуль, а значение `NegativeInfinity` – как результат деления отрицательного числа на нуль.

Между арифметическими типами можно использовать простой, скобочный способ приведения к нужному типу. Но таким способом нельзя привести, например, переменную типа `string` к типу `int`. Здесь необходим вызов метода `ToInt32` класса `Convert`. Класс `Convert`, определенный в пространстве имен `System`, играет важную роль, обеспечивая необходимые преобразования между различными типами. Методы класса `Convert` поддерживают общий способ выполнения преобразований между типами. Класс `Convert` содержит 15 статических методов вида `To<Type>(ToBoolean(),...ToUInt64())`, где `Type` может принимать значения от `Boolean` до `UInt64` для всех встроенных типов, перечисленных в табл. 5.2.1. Единственным исключением является тип `object`, метода `ToObject` нет по понятным причинам, поскольку для всех типов существует неявное преобразование к типу `object`. Среди других методов можно отметить общий статический метод `ChangeType`, позволяющий преобразование объекта к некоторому заданному типу.

Логический тип

Отметим, что в отличие от C++, в языке C# логическим типам `bool` нельзя присваивать числовые значения, вроде 0 или 1, а можно присваивать только значения логических констант «true» или «false».

Строковые и символьные типы

Это типы `char` и `string`. Для текстовых данных C# имеется только два типа `char` и `string`. Причем тип `char` является значащим, тип `string` – ссылочным.

Константа типа `char` (символьный литерал) задается символом в апострофах:

```
char c = 'A';
```

Символьную константу можно также ввести и в виде целочисленного значения в диапазоне от 0 до 65535. Тогда перед числом нужно в скобках указать символ приведения типа (`char`), например, `(char)44` – это символ ' ', ' '.

Тип `char` представлен структурой, в которой имеется метод `GetNumericValue(char c)` для возвращения числового кода указанного символа и ряд специальных методов для работы с символами.

Строковые выражения `string` ограничиваются символами «"», например,

```
Console.WriteLine("Здравствуй, " + name + "!");
```

Если нужно включить в строку сам символ ", то перед ним ставится символ «\». Все строки в C# происходят от единственного базового класса `System.String`. Этот класс имеет множество специальных методов для работы со строками. Приведем несколько примеров методов класса `String`:

- `Length` – возвращает длину указанной строки;
- `Concat()` – объединяет две строки в одну;
- `Copy()` – создает копию существующей строки;
- `Format()` – форматирует строку с использованием других примитивов и подстановочных выражений;
- `Insert()` – вставка строки в существующую.

Управляющие последовательности. Эта категория литералов используется для создания дополнительных эффектов и простого форматирования выводимой

информации. Следующие последовательности символов являются управляющими последовательностями:

- `\b` – возврат на одну позицию;
- `\f` – переход на новую страницу;
- `\n` – переход на новую строку
- `\r` – возврат каретки;
- `\t` – горизонтальная табуляция;
- `\v` – вертикальная табуляция;
- `\0` – пустой символ (NULL);
- `'` – одинарная кавычка, апостроф
- `"` – двойная кавычка;
- `\\` – обратная косая черта;
- `\a` – системное оповещение (звонок).

Помимо управляющих последовательностей в C# предусмотрен также специальный префикс `@` для дословного (verbatim string) ввода строки вне зависимости от наличия в ней управляющих последовательностей. Символ `@` располагается непосредственно перед строкой, заключенной в двойные кавычки. Представление двойных кавычек в дословной строке обеспечивается их дублированием. Например, следующие строки

```
... "c:\My Documents\sample.txt" ...  
... @ "c:\My Documents\sample.txt" ...
```

имеют одно и то же значение

```
c:\My Documents\sample.txt
```

Обычные строки типа `String` являются неизменяемыми объектами. Методы класса `String` не изменяют строку, а возвращают измененную ее копию. В C# имеется возможность работать со строками напрямую используя класс `StringBuilder` (пространство имен `System.Text`). Он представляет собой модификацию класса `String` для работы со строками большого размера. Объекты `StringBuilder` легко преобразуются в обычные строки методом `ToString`.

Перечисления

Тип `enum`. *Перечисление* объявляется с помощью ключевого слова `enum`, идентифицируется по имени и представляет собой непустой список неизменяемых именованных значений целого типа. Первое значение в перечислении по умолчанию инициализируется нулем. Каждое последующее значение отличается от предыдущего по крайней мере на единицу, если объявление значения не содержит явного дополнительного присвоения нового значения. Примеры объявления перечислений:

```
enum En1 { One, Two, Three };  
enum Colors { Red = 1, Green = 2, Blue = 4, Yellow = 8 };
```

Первое перечисление создает целочисленные типы со значениями: `One=0`, `Two=1`, `Three=2`. Такое присвоение производится по умолчанию. Во второй строке явно заданы значения элементов перечисления.

Обращение к элементу перечисления осуществляется посредством сложного выражения, состоящего из имени класса перечисления, операции доступа к эле-

менту перечисления «.», имени элемента перечисления. В следующем примере переменная `xVal` инициализируется значением перечисления:

```
int xVal = Colors.Red;
```

Каждое перечисление имеет свой базовый тип, которым может быть любой встроенный целочисленный тип C# (`int`, `long`, `short` и т. д.), за исключением `char`. Перечисление задается следующим образом:

```
[атрибуты] [модификаторы] enum идентификатор[: базовый тип] { список перечислений};
```

Базовый тип – это тип значений для перечисления. Если не учитывать этот описатель при создании перечисления, то будут использоваться значения по умолчанию `int`. Но можно использовать любой из целочисленных типов (например, `ushort`, `long`), за исключением `char`. Например, следующий фрагмент кода объявляет перечисление целых чисел без знака (`uint`):

```
enum Sizes: uint { Small = 1, Middle = 2, Large = 3 };
```

Перечисление является классом, который имеет методы сравнения значений перечисления, методы преобразования значений перечисления в строковое представление, методы перевода строкового представления значения в перечисление, а также средства для создания объектов – представителей класса перечисления.

Организация системы типов

Способом организации определенной системы типов (классов, интерфейсов, делегатов, структур, перечислений) в единую группу является пространство имен. Вне зависимости от языка программирования, доступ к определенным классам обеспечивается за счет их группировки в рамках общих пространств имен. Приведем некоторые пространства имен библиотеки базовых классов .NET:

- `System` – множество низкоуровневых классов для работы с простыми типами, выполнения сборки мусора и т.п. Содержит класс `Object` и другие классы, которые обеспечивают самые важные функции C#. Каждое нормально работающее приложение использует это пространство имен;
- `System.Data` – классы для обращения к базам данных;
- `System.Collections`, `System.Collections.Generic` – классы для работы с контейнерными объектами, такими как `ArrayList`, `Queue`, `SortedList`;
- `System.Diagnostics` – классы для трассировки и отладки программного кода;
- `System.Drawing` – классы графической поддержки, такие, как: `Bitmap`, `Brush`, `Color`, `Graphics`, `Image`, `pen`, `Size`, `Region` и др.;
- `System.IO` – типы, отвечающие за операции ввода/вывода в файл, буфер и т.п.;
- `System.Reflection` – классы для обнаружения, создания и вызова во время выполнения пользовательских типов, содержит класс `Assembly` – для загрузки и изучения сборки и выполнения с ней различных операций и другие классы для получения информации: `EventInfo`, `FieldInfo`, `MemberInfo`, `MethodInfo`, `Module` и другие;
- `System.Reflection.Emit` – классы для создания динамических сборок и типов. Содержит класс `AssemblyBuilder` – для создания сборки в процессе работы

программы и другие классы: `TypeBuilder`, `EnumBuilder`, `MethodBuilder`, `FieldBuilder`, `ModuleBuilder` и другие;

- `System.Runtime.InteropServices`, `System.Runtime.Remoting` – поддержка взаимодействия с «обычным кодом» – DLL, COM-серверы, удаленный доступ;
- `System.ComponentModel` – классы компонент, как визуальных, так и невидимых. Основоположителем всех компонент является класс `Component`;
- `System.Windows.Forms` – для создания обычных приложений .NET с графическим интерфейсом, содержит класс `Control`, содержит классы для форм и видимых компонент, таких как `Application`, `Button`, `CheckBox`, `Form`, `FileDialog`, `Menu`, `Timer`, `ScrollBar` и др.

В программе использование пространства имен обеспечивается несколькими строками в начале программы, например:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using MathWorks.MATLAB.NET.Arrays;
```

5.2.3. Массивы

Массивом называется упорядоченная совокупность элементов одного типа. Каждый элемент массива имеет индексы, определяющие положение элементов. Число индексов характеризует размерность массива. Каждый индекс изменяется в некотором диапазоне. В языке C#, как и во многих других языках, индексы задаются целочисленным типом. При объявлении массива границы задаются выражениями. Если все границы заданы константными выражениями, то число элементов массива известно в момент его объявления и ему может быть выделена память еще на этапе трансляции. Такие массивы называются статическими. Если же выражения, задающие границы, зависят от переменных, то такие массивы называются динамическими, поскольку память им может быть отведена только динамически в процессе выполнения программы, когда становятся известными значения соответствующих переменных. Массиву, как правило, выделяется непрерывная область памяти.

В .NET все массивы являются наследниками одного общего (базового) класса `System.Array`. Класс `System.Array` имеет специальный набор методов для создания, управления, поиска и сортировки элементов массива.

В языке C# имеются одномерные и многомерные (классические прямоугольные) массивы. Кроме того, как и в C++, определены и массивы массивов. Это позволяет определять зубчатые (ступенчатые, *jagged*) массивы, в которых размеры массива по некоторым размерностям (например, длины строк), могут иметь разную длину. Напомним, что индексация в C# начинается с нуля.

Описание массива и его инициализация. Массивы относятся к ссылочным типам, поэтому должны инициализироваться при помощи оператора **new**. Операция

`new` используется для создания массива и инициализации его элементов предопределенными значениями. В результате выполнения этого оператора все элементы массива будут установлены нулевыми (пустая строка для строковых массивов). Для описания массива нужно указать квадратные скобки за именем типа данных массива. Например,

```
int[] arrInt;           // объявление массива
arrInt = new int[10];   // инициализация массива нулевыми значениями
for (int i=0; i<10; i++)
    arrInt[i] = i*i;     // задание элементов массива
```

Можно объявлять массив и одновременно инициализировать его (нулевыми значениями):

```
double[] arrDoub = new double[10];
string[] myStringArray = new string[6];
```

Можно также объявлять массив и одновременно задать его значения непосредственно списком, в этом случае необязательно задавать размерности. Если объявление совмещается с инициализацией, операция `new` может быть опущена,

```
int[] arrInt = {1, 2, 3, 4, 5};
float[] arrFlo = new float[5] {1.0, 2.0, 3.0, 4.0, 5.0};
string[] weekDays = {"Sun", "Sat", "Mon", "Tue", "Wed", "Thu", "Fri"};
```

Многомерные массивы. Они определяются как массивы массивов и как обычные многомерные. При объявлении многомерного массива указывается пара квадратных скобок, в которых указываются запятые, разделяющие незаполненные поля размеров, либо – нужное количество квадратных скобок. Следующие два примера показывают разницу.

```
int[][] arrInt1;           // объявление двумерного массива массивов
arrInt1 = new int[10][10]; // инициализация массива нулевыми значениями
for (int i=0; i<10; i++){
    for (int j=0; j<10; j++)
        arrInt1[i][j] = i*j; // задание элементов массива
```

Объявление двумерного прямоугольного массива может быть сделано так:

```
int[, ] arrInt2;           // объявление массива 2-на-2
arrInt2 = new int[10,10]; // инициализация массива нулевыми значениями
for (int i=0; i<10; i++){
    for (int j=0; j<10; j++)
        arrInt2[i,j] = i*j; // задание элементов массива
```

Можно также объявлять массив и одновременно задать его значения непосредственно списком, в этом случае необязательно задавать размерности:

```
int[, ] arrInt3 = {{1,2}, {3,4}, {5,6}, {7,8}};
```

Отметим, что первый способ, с несколькими парами скобок, является более гибким. Он соответствует идее «массива массивов». Например, в следующей строке объявлен одномерный массив из 2-х элементов, каждый элемент которого является одномерным массивом из 3-х элементов, каждый элемент которого также является одномерным массивом из 4-х элементов `int`:


```
int[][][] arrInt3;  
arrInt3 = new int[2][3][4];
```

В следующей строке объявлен одномерный массив из 2-х элементов (страниц), каждый элемент является двумерным массивом 3-на-4 int:

```
int[][ , ] arrInt4;  
arrInt4 = new int[2][3,4];
```

В следующей строке объявлен двумерный массив из 2-на3, каждый элемент которого является одномерным массивом из 4-х элементов int:

```
int[ , ][] arrInt5;  
arrInt5 = new int[2,3][4];
```

При использовании такой конструкции, как массив массивов, для инициализации необходимо задавать обязательно размер первой составляющей массива. Все остальные размеры могут оставаться пустыми. Например, следующие способы объявления являются корректными:

```
int [][] arrInt6 = new int[15][];  
int dim1 = 100;  
int [][][] arrInt7 = new int[dim1][][];  
int [,][]arrInt8 = new int[dim1,7][];
```

Такая форма определения массива предполагает многоступенчатую инициализацию, при которой производится последовательная инициализация составляющих массива.

В силу того, что массив является объектом ссылочного типа, составляющие одного массива могут быть использованы для инициализации другого массива. Например:

```
int [] arrInt = {0,1,2,3,4,5,6,7,8,9};  
arrInt2[1][0] = arrInt;
```

Рассмотрим объявление и инициализацию зубчатых массивов. Следующий пример задает одномерный массив, состоящий из трех элементов, каждый из которых является одномерным массивом разной длины.

```
int[][] myJaggedArray = new int[3][];  
myJaggedArray[0] = new int[5];  
myJaggedArray[1] = new int[4];  
myJaggedArray[2] = new int[2];
```

Ниже показан пример использования заполняющей инициализации, при которой одновременно с определением (созданием) массивов производится присвоение элементам массивов конкретных значений:

```
myJaggedArray[0] = new int[] {1,3,5,7,9};  
myJaggedArray[1] = new int[] {0,2,4,6};  
myJaggedArray[2] = new int[] {11,22};
```

Вышеупомянутый массив может быть объявлен и проинициализирован и таким образом:

```
int[][] myJaggedArray = new int [][]  
{
```

```
new int[] {1,3,5,7,9},  
new int[] {0,2,4,6},  
new int[] {11,22}  
};
```

Доступ к элементам массива обеспечивается посредством выражений индексации:

```
myJaggedArray[0][1] = 33; // Задание нового значения элементу массива  
myJaggedArray[2][1] = 44;
```

Массивы как параметры. В качестве параметра методу всегда можно передать ссылку на массив. Массив должен быть полностью построен, либо должен быть описан как одномерный массив массивов. Тип и количество составляющих данный массив других компонентов для механизма передачи параметров значения не имеют. Важно, что в стеке будет выделено определенное (соответствующее значению первой размерности) количество проинициализированных ссылок на составляющие данный одномерный массив элементы. Передаваемый в качестве параметра массив может быть предварительно проинициализирован.

Неопределенное (переменное) количество однотипных параметров или список параметров переменной длины передается в функцию в виде ссылки на одномерный массив. Эта ссылка в списке параметров функции должна быть последним элементом списка параметров. Ссылке должен предшествовать спецификатор `params`.

В выражении вызова метода со списком параметров, члены списка могут присутствовать либо в виде списка однотипных значений (этот список преобразуется в массив значений), либо в виде ссылки на одномерный массив значений определенного типа.

Динамические массивы. Во всех вышеприведенных примерах объявлялись статические массивы, поскольку нижняя граница индекса равна нулю по определению, а верхняя всегда задавалась в этих примерах константой. В C# все массивы, независимо от того, каким выражением описывается граница, рассматриваются как динамические и память для них распределяется в куче. В действительности реальные потребности в размере массива, скорее всего, выясняются в процессе работы в диалоге с пользователем. Чисто синтаксически нет существенной разницы в объявлении статических и динамических массивов. Выражение, задающее границу изменения индексов, в динамическом случае содержит переменные. Единственное требование – значения переменных должны быть определены в момент объявления. Это вполне естественное ограничение.

5.2.4. Операции и выражения

Выражение строится из операндов (констант, переменных, функций), объединенных знаками операций и скобками. Рассмотрим *операции* языка C#.

Операция new. Используется для создания объектов и передачи управления конструкторам, например:

```
Class1 myVal = new Class1(); // Объект ссылочного типа. Создается в куче.
```

Операция `new` также используется для обращения к конструкторам объектов типа значений:

```
int myInt = new int(); // Объект типа int размещается в стеке!
```

При определении объекта `myInt` ему было присвоено начальное значение 0, которое является значением по умолчанию для типа `int`. Следующий оператор имеет тот же самый эффект:

```
int myInt = 0;
```

Арифметические операции. В языке `C#` имеются обычные для всех языков арифметические операции: `+`, `-`, `*`, `/`, `%`. Все они перегружены. Операции `+` и `-` могут быть унарными и бинарными. Операция деления `/` над целыми типами осуществляет деление нацело, для типов с плавающей и фиксированной точкой – обычное деление. Операция `%` определена над всеми арифметическими типами и возвращает остаток от деления нацело. Тип результата зависит от типов операндов.

Операции отношения. Операции отношения в объяснениях не нужны. Всего операций шесть: `==`, `!=`, `<`, `>`, `<=`, `>=`. Для тех, кто не привык работать с языком `C++`, стоит обратить внимание на запись операций «равно» и «не равно».

Логические операции. Это операции `&`, `|`, `^`, `!`, `?:`, `&&`, `||`. Правила работы с логическими выражениями в языках `C#` и `C++` имеют принципиальные различия. В языке `C++` практически для всех типов существует неявное преобразование в логический тип: ненулевые значения трактуются как истина, нулевое – как ложь. В языке `C#` неявных преобразований к логическому типу нет даже для целых арифметических типов.

Отметим также операцию `+` соединения строк и индексацию: `[]`.

Контроль за переполнением, `checked` и `unchecked`. Арифметические типы имеют ограниченные размеры. Поэтому любая арифметическая операция может привести к переполнению. По умолчанию в `C#` переполнение, возникающее при выполнении операций, никак не контролируется. Возможный неверный результат вычисления остается всего лишь результатом выполнения операции, и никого не касается, как эта операция выполнялась.

Механизм контроля за переполнением, возникающим при выполнении арифметических операций, обеспечивается ключевыми словами `checked` (включить контроль за переполнением) и `unchecked` (отключить контроль за переполнением), которые используются в составе выражений. Конструкции управления контролем за переполнением имеют две формы:

- операторную, которая обеспечивает контроль над выполнением одного выражения:

```
short x = 32767, y = 32767, z = 0;
z = checked(x + unchecked(x+y));
return z;
```

- блочную, которая обеспечивает контроль над выполнением операций в блоке операторов:

```
short x = 32767, y = 32767, z = 0, w = 0;
unchecked
```

```
{ w = x+y; }  
checked  
{ z = x+w; }
```

Особенности выполнения арифметических операций. Они связаны с ограниченностью диапазонов чисел и с ограниченной точностью переменных типа `float` (7 значащих цифр) и `double` (16 значащих цифр). Если результат целой операции выходит за диапазон своего типа `int` или `long`, то автоматически происходит приведение по модулю, равному длине этого диапазона, и вычисления продолжаютсЯ с неправильным результатом.

Ограниченная точность значений типа `System.Single` проявляется при ее приведении к типу `System.Double`. Например, число типа `float`, равное значению 1.0, может быть не равно 1.0 как число типа `double`.

Если переменной `x` типа `float`, или `double` присвоить значение, которое выходит за пределы диапазона значений, то в случае слишком малого положительного числа результатом является положительный нуль (+0), в случае слишком малого отрицательного числа результатом является отрицательный нуль (−0), а в случае слишком большого положительного или отрицательного числа значением является положительная или отрицательная бесконечность (+Infinity, −Infinity). Выполнение операции деления над значениями типами с плавающей точкой (0.0/0.0) дает неопределенность NaN (Not a Number).

5.2.5. Управление последовательностью выполнения операторов

Рассмотрим основные операторы, которые позволяют управлять ходом выполнения программы. Поскольку они такие же, что и в других языках программирования, то описание будет кратким.

Оператор *if...else* условного перехода

Условный переход можно реализовать в программе с помощью ключевых слов языка: **if**, **else** или **switch**.

Действие оператора `if...else` определяется условием, которое анализируется инструкцией `if`.

```
if ( expression ) statement1  
[else statement2 ]
```

Если значение выражения `expression` истинно, то выполняется блок инструкций `statement1`. Если же выражение ложно, произойдет выполнение блока инструкций `statement2`. Необходимо заметить, что вторая часть оператора (`else statement`) может не указываться. Если инструкций в блоках `statement1` или `statement2` больше одной, то блок обязательно нужно брать в фигурные скобки.

Для обработки сложных условий возможно вложение условных операторов в блоки инструкций других условных операторов. Оператор `if` в инструкции сравнения может применять несколько инструкций, объединенных арифметическими опе-

раторами. В качестве последних используются операторы ($\&\&$ – И), ($\mid \mid$ – ИЛИ) и ($!$ – НЕ).

Рассмотрим пример использования составных инструкций в блоке if.

```
using System;
namespace C_Sharpprogramming
{
class Conditions
{
    static void Main(string[] args)
    {
        int n1 = 5;
        int n2 = 0;
        if ((n1 == 5) && (n2 == 5))
            Console.WriteLine("Инструкция И верна");
        else
            Console.WriteLine("Инструкция И не верна");
            if((n1 == 5) || (n2 == 5))
                Console.WriteLine("Инструкция ИЛИ верна");
            else
                Console.WriteLine("Инструкция ИЛИ не верна");
        }
    }
}
```

В данном примере каждая инструкция if проверяет сразу два условия. Первое if условие использует оператор ($\&\&$) для проверки условия. Необходимо, чтобы сразу два условия в блоке if были истинны. Только тогда выражение будет считаться верным. При этом если первое выражение однозначно определяет результат операции, то второе уже не проверяется. Так, если бы условие $n1 == 5$ было ложным, то условие $n2 == 5$ уже не проверялось бы, поскольку его результат перестал бы играть роль.

Второе выражение использует оператор ($\mid \mid$) для проверки сложного условия: `if((n1 == 5) || (n2 == 5))`

В данном случае для верности всего выражения достаточно истинности лишь одного из условий. И тогда действует правило проверки условий до выяснения однозначности результата. То есть, если условие $n1 == 5$ верно, то $n2 == 5$ уже проверяться не будет.

Оператор switch

Достаточно часто встречаются ситуации, когда вложенные условные операторы выполняют множество проверок на совпадение значения переменной, но среди этих условных операторов только один является истинным. Тогда лучше воспользоваться оператором **switch**. Выражение условия помещено в круглые скобки после оператора switch. Внутри тела оператора switch есть секция выбора case и секция действия по умолчанию default. Секция выбора нужна для определения действия, которое будет выполняться при совпадении соответствующего константного выражения заданному выражению в switch. В этой секции обязательно нужно указать одно или несколько действий. Секция default может в операторе

`switch` не указываться. Она выполняется в том случае, если не совпала ни одна константная инструкция из секции выбора.

Оператор **case** требует обязательного указания значения для сравнения (литеральная или символическая константа, или перечисление), а также блока инструкций и оператора прерывания действия. Если результат условия совпадет с константным значением оператора `case`, то будет выполняться соответствующий ему блок инструкций. Как правило, в качестве оператор перехода используют оператор **break**, который прерывает выполнение оператора `switch`. Пример:

```
switch ( myValue )
{
    case 10: Console.WriteLine("myValue равно 10" ) ;
            break;
    case 20: Console.WriteLine("myValue равно 20");
            break;
}
```

В C# каждая непустая секция инструкций оператора `case` должна содержать в себе оператор `break`.

Оператор цикла *while*

Эта циклическая инструкция работает по принципу: «Пока выполняется условие – происходит работа». Как и в других инструкциях, выражение – это условие, которое оценивается как булево значение. Если результатом проверки условия является истина, то выполняется блок инструкций, в противном случае в результате выполнения программы, **while** игнорируется. В следующем примере программа проверяет условие $i < 10$, выводит сообщение на экран и наращивает i .

Пример.

```
public static int Main()
{
    int i = 0;
    while(i < 10)
    {
        Console.WriteLine("i: {0}", i) ;
        i++;
    }
    return 0;
}
```

Цикл **while** проверяет значение i перед выполнением блока `statement`. Это гарантирует, что цикл не будет выполняться, если проверяемое условие ложно. Таким образом, если первоначально i примет значение 10 и более, цикл не выполнится ни разу.

Оператор цикла *do... while*

Бывают случаи, когда цикл `while` не совсем удовлетворяет нашим требованиям. Например, нужно проверять условие не в начале, а в конце цикла. В таком случае лучше использовать цикл **do...while**. Подобно `while`, выражение – это условие, которое оценивается как булево значение. Разница с циклом `while` состоит в том, что

цикл `do..while` выполняется всегда минимум один раз, до того как произойдет проверка условия выражения. Пример:

```
public static int Main()
{
    int i = 0;
    do { Console.WriteLine ("i:{0}", i); }
    while(i < 10) ;
    return 0;
}
```

На этом примере видно, что если даже первоначально `i` примет значение 10 и более, цикл выполнится. Затем произойдет проверка условия `while (i < 10)`, результатом которой станет ложь (`false`), и повтора выполнения цикла не произойдет.

Оператор цикла *for*

Цикл `for` имеет синтаксис, показанный на следующем примере:

```
public static int Main ()
{
    for(int i = 0; i < 10; i++)
    {
        Console.WriteLine("i: {0}", i);
    }
    return 0;
}
```

Наращивание переменной внутри цикла происходит на такое число единиц, на которое вы сами зададите. Операция `i++` означает «нарастить переменную на 1». Если вы хотите использовать другой шаг изменения `i`, то смело можете написать так `i += 2`. В этом случае переменная `i` будет изменяться на 2 единицы, и на экране вы увидите:

```
0 2 4 6 8
```

Операторы *break* и *continue*

Бывают ситуации, когда необходимо прекратить выполнение цикла досрочно (до того как перестанет выполняться условие) или при каком-то условии не выполнять описанные в теле цикла инструкции, не прерывая при этом цикла. Для таких случаев очень удобно использовать инструкции **`break`** и **`continue`**. Для того чтобы прервать выполнение цикла, используется инструкция `break`. Оператор `continue` в отличие от `break` не прерывает хода выполнения цикла. Он лишь приостанавливает текущую итерацию и переходит сразу к проверке условия выполнения цикла.

```
for (int j = 0; j < 100; j++)
{
    if (j%2 == 0)
        continue;
    Console.WriteLine("{0}", j);
}
```

Такой цикл позволит вывести на экран все нечетные числа. Работает он следующим образом: перебирает все числа от 0 до 100. Если очередное число четное – все дальнейшие операции в цикле прекращаются, наращивается число *j*, и цикл начинается сначала.

5.2.6. Класс и структура

Объявления классов вводятся ключевым словом `class`, а структура – ключевым словом `struct`. Классы и структуры являются программно определяемыми типами, которые позволяют создавать новые типы, специально приспособленные для решения конкретных задач. В рамках объявления класса и структуры описывается множество переменных различных типов, правила порождения объектов – представителей структур и классов, их основные свойства и методы.

Классы

Для объявления *класса* используется ключевое слово `class`, за которым следует имя класса и далее, в фигурных скобках `{}` – тело класса. Более точно, объявление класса состоит из следующих элементов:

- указание атрибутов (необязательный элемент объявления);
- указание модификаторов, в том числе модификаторов прав доступа (необязательный элемент объявления);
- указание спецификатора разделения объявления класса `partial` (необязательный элемент объявления);
- ключевое слово `class`;
- имя класса;
- имена предков класса и интерфейсов (необязательный элемент объявления);
- тело класса.

Атрибуты. Это средство добавления декларативной (вспомогательной) информации к элементам программного кода. Назначение атрибутов: организация взаимодействия между программными модулями, дополнительная информация об условиях выполнения кода, управление сериализацией (правила сохранения информации), отладка и многое другое.

Модификаторы прав доступа. Представлены следующими значениями:

- `public` – обозначение для общедоступных членов класса. К ним можно обратиться из любого метода любого класса программы;
- `protected` – обозначение для членов класса, доступных в рамках объявляемого класса и из методов производных классов;
- `internal` – обозначение для членов класса, которые доступны из методов классов, объявляемых в рамках сборки, где содержится объявление данного класса;
- `protected internal` – обозначение для членов класса, доступных в рамках объявляемого класса, из методов производных классов, а также доступных

из методов классов, которые объявлены в рамках сборки, содержащей объявление данного класса;

- **private** – обозначение для членов класса, доступных в рамках объявляемого класса.

При объявлении класса допускается лишь один явный модификатор – **public**. Отсутствие модификаторов доступа в объявлениях членов класса эквивалентно явному указанию модификаторов **private**.

Спецификатор **partial.** Позволяет разбивать код объявления класса на несколько частей, каждая из которых размещается в собственном файле. Если объявление класса занимает большое количество строк, его размещение по нескольким файлам может существенно облегчить работу над программным кодом, его документирование и модификацию. Транслятор способен восстановить полное объявление класса. Спецификатор **partial** может быть использован при объявлении классов, структур и интерфейсов.

Сочетание ключевого слова **class** (**struct**, **interface**) и имени объявляемого класса (структуры или интерфейса) задает имя типа.

Тело класса в объявлении ограничивается парой фигурных скобок { }, между которыми располагаются объявления данных – членов и методов класса.

Метод **Main.** У каждого приложения на C# должен быть метод **Main**, определенный в одном из его классов. Кроме того, этот метод должен быть определен как **static**. Для компилятора C# не важно, в каком из классов определен метод **Main**, а класс, выбранный для этого, не влияет на порядок компиляции. Компилятор C# самостоятельно просматривает файлы исходного кода и отыскивает метод **Main**. Этот метод является точкой входа во все приложения на C#. Можно поместить метод **Main** в любой класс, но для его размещения рекомендуется создавать специальный класс (см. пример 2 ниже).

Членами класса могут быть методы, поля, свойства и события.

Методы и конструкторы. Это функции, которые исполняются в данном классе. Метод – это именованная часть программы, к которой можно обращаться из других частей программы столько раз, сколько потребуется. Хотя метод возвращает единственное значение, в C# возможно возвращение нескольких параметров. В этом случае их нужно включить в число входных аргументов и пометить ключевым словом **ref**. Тогда передаваемые аргументы указывают на ту же область памяти, что и переменные вызывающего кода. Таким образом, если вызванный метод изменяет их и возвращает управление, переменные вызывающего кода также подвергнутся изменениям. Следует учитывать, что перед вызовом метода вы должны инициализировать передаваемые аргументы с ключевым словом **ref**. Альтернативный способ передачи аргументов, изменения значений которых должны быть видимы вызывающему коду – с помощью ключевого слова **out**. В этом случае инициализация переменных не обязательна.

Можно задавать переменное число параметров метода через ключевое слово **params** и указание массива в списке аргументов метода:

```
public void DrawLine(params Point[] p)
```

Конструктором называется специальный метод класса, который выполняет создание объекта класса. Например, в следующем коде создается экземпляр obj класса Odeclass вызовом конструктора Odeclass():

```
Odeclass obj = new Odeclass();
```

Конструкторы объявляются с модификатором **public**, так как они обычно вызываются вне данного класса. Обычно имя конструктора совпадает с именем класса. Если в классе не объявлен конструктор, то автоматически такой класс снабжается конструктором по умолчанию.

Поля. Они предназначены для хранения связанных с экземпляром класса данных. Поле – это переменная или константа, содержащая некоторое значение. Это характеристики конкретного экземпляра класса. При создании каждый экземпляр класса получает свой набор характеристик и имеет общие для класса методы, свойства и события. Для определения поля как константы, перед ним указывается ключевое слово **const**, например,

```
public const double pi = 3.1415;
```

Константа определяется до компиляции. Если возникает потребность в константе, которая возникает в период выполнения программы, то она определяется как неизменяемое поле (read-only field) с помощью ключевого слова **readonly**. Значение такого поля можно установить лишь в одном месте – в конструкторе. Если нужна константа статическая, но инициализируемая в период выполнения, то нужно определить поле с модификаторами **static** и **readonly**, а затем создать особый, статический тип конструктора. Статические конструкторы (static constructor) используются для инициализации статических, неизменяемых и других полей.

Свойства класса. Во многом подобны полям. Они могут стоять в правой части оператора присваивания, либо являться членом выражения в его правой части. Объявление свойства отличается от объявления поля. Синтаксически свойство подобно методу: имеет фигурные скобки, в которых записываются два метода **get()** и **set()**. Метод **get()** возвращает текущее значение свойства, а метод **set()** – устанавливает значение свойства. Текущее значение хранится в зарезервированной переменной **value**. Если свойство не имеет одного из этих методов, то оно является свойством только для чтения или только для записи. Пример свойства Salary с методами чтения и записи:

```
public float Salary
{
    get{return Salary;}
    set(Salary = value;)
}
```

События. Позволяют классам автоматически реагировать на действия пользователя и на изменения в состоянии программы. Событие вызывает исполнение некоторого фрагмента кода. События – неотъемлемая часть программирования для Microsoft Windows. Например, события возникают при движении мыши, щелчке или изменении размеров окна.

Абстрактные классы. Методы класса могут быть объявлены как абстрактные. Это означает, что в этом классе нет реализации этих методов. Абстрактные мето-

ды пишутся с модификатором `abstract`. Класс, в котором есть хотя бы один абстрактный метод, называется абстрактным (в таком классе могут быть и обычные методы). Нельзя создавать экземпляры абстрактного класса – такой класс может использоваться только в качестве базового класса для других классов. Для потомка такого класса есть две возможности: или он реализует все абстрактные методы базового класса (и в этом случае для такого класса-потомка мы сможем создавать его экземпляры), или реализует не все абстрактные методы базового класса (в этом случае он является тоже абстрактным классом, и единственная возможность его использования – это производить от него классы-потомки).

Приведем примеры кодов, в которых определяются классы.

Пример 1. Приведем листинг кода `Program.cs` класса `Program`, который создается в Visual Studio 2005 и исполняет все, что реализовано в форме приложения `Form1`. Это специальный класс для метода `Main`, кроме этого метода он фактически ничего не содержит.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
namespace OdeWinAppl
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Пример 2. Листинг кода класса `Form1`, который создается пользователем в Visual Studio 2005 и исполняет все, что реализовано в форме приложения `Form1`. Файл `Form1.cs` находится в каталоге `Gl_5_C#_Examples\Diff_Ury\OdeWinAppl` на прилагаемом CD. В данном случае по нажатию кнопки `button1` (событие `button1_Click`) на форме `Form1` считываются все необходимые данные и решается задача Коши для дифференциального уравнения первого порядка, а при нажатии кнопки `button2` (событие `button2_Click`) строится график решения.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```

using MathWorks.MATLAB.NET.Arrays;
using Ode;

namespace OdeWinAppl
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // --- Дифференциальное уравнение первого порядка -----
        // Объявления входных переменных
        double t0, t1, y0;           // Границы времени и начальное y0
        double[] tspan = new double[2]; // Интервал изменения t
        string func, st0, st1, sy0;    // Строковые входные переменные
        // Объявления выходных переменных
        MWArray[] mw_ArrayOut = null; // Выходной массив параметров
        MWNumericArray mw_T1 = null;  // Выходной параметр время
        MWNumericArray mw_Y1 = null;  // Выходной параметр, решение

        private void button1_Click(object sender, EventArgs e)
        {
            // Текст программы решения дифференциального уравнения
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // Текст программы построения графика y=y(t)
        }
    }
}

```

Структуры

Для объявления структуры используется ключевое слово `struct`, за которым следует имя структуры и далее, в фигурных скобках `{}` – тело структуры. С точки зрения синтаксиса, между объявлениями классов и структур существуют незначительные различия. В структуре не допускается объявлений членов класса со спецификаторами доступа `protected` и `protected internal`, кроме того, при объявлении структуры не допускается объявление конструктора без параметров. Основное их различие состоит в том, что класс и структура принадлежат к двум различным категориям типов: класс – это ссылочный тип, а структура – значащий тип.

Интерфейсы

Интерфейс представляет собой полностью абстрактный класс без полей, все методы которого описаны, но не реализованы. Интерфейс объявляется ключевым словом `interface`, а функции интерфейса, несмотря на свою «абстрактность», объявляются без ключевого слова `abstract`. Основное отличие интерфейса от абстрактного класса заключается в том, что производный класс может наследовать одновременно несколько интерфейсов.

Когда создается интерфейс и в определении класса задается его использование, то говорят, что класс реализует интерфейс. Класс, исполняющий интерфейс, называется интерфейсным.

Интерфейсы – это набор характеристик поведения, а класс определяется как реализующий их. Интерфейсы могут содержать методы, свойства и события, но ни одна из этих сущностей не реализуется в самом интерфейсе. Поскольку интерфейс определяет связь между фрагментами кода, любой класс, реализующий интерфейс, должен определять каждый элемент этого интерфейса, иначе код не будет компилироваться.

5.2.7. Отражение

Одним из главных достоинств .NET Framework и общезыковой исполняющей среды CLR является обилие доступной информации о типах. Система *отражения* (reflection) позволяет запрашивать информацию о типах и управлять ею. Отражение позволяет запрашивать и генерировать код от целых сборок и модулей до отдельных выражений.

Код в CLR упаковывается в сборку. Метаданные сборки используются CLR для исполнения кода и содержат информацию о типах классов, структур, делегатов и интерфейсов, содержащихся в сборке. Информация о типе включает описание его методов, свойств, событий, делегатов и перечислений. Кроме того, метаданные кода включают его размер и локальные переменные.

Обычно сборку отождествляют с файлом, но фактически это логический контейнер для следующих данных, необходимых CLR для исполнения кода:

- метаданные сборки, это манифест;
- метаданные типа, это пространство имен, имя типа (класса), члены типа – методы, свойства и конструкторы;
- код на промежуточном языке IL;
- ресурсы – это объекты, которые используются кодом, такие как строки, изображения и другие файлы.

Помещать сборку в один файл не всегда удобно. Иногда желательно разделить сборку на несколько файлов, например, хранить некоторые ресурсы вне файла сборки и загружать их по мере необходимости. Модули – это контейнеры типов, расположенные внутри сборки. Модуль может быть и в простой и в многофайловой сборке.

Для получения свойств сборки нужно создать экземпляр класса *Assembly* и запросить свойства сборки. Приведем для примера несколько свойств и методов класса *Assembly* (полная информация имеется в документации).

Свойства класса *Assembly*:

- *EntryPoint* – получает метод, представляющий код, с которого начинается исполнение сборки;
- *FullName* – получает полное имя сборки;
- *Location* – получает путь к сборке.

Методы класса *Assembly*:

- *CreateInstance* – создает экземпляр определенного типа, входящего в сборку;

- `GetCustomAttributes` – возвращает массив атрибутов сборки;
- `GetExportedTypes` – возвращает набор открытых типов, доступных извне сборки;
- `GetFile` – возвращает объект `FileStream` для файла, входящего в ресурсы сборки;
- `GetFiles` – возвращает массив объектов `FileStream`, представляющих все файлы, входящие в ресурсы сборки;
- `GetLoadedModules` – возвращает массив загруженных в настоящий момент модулей сборки;
- `GetModule` – возвращает заданный модуль сборки.
- `GetTypes` – возвращает массив всех типов, определенных в модулях сборки.

Класс `Assembly` также поддерживает загрузку сборки только для получения информации о ней. При загрузке сборки можно прочитать ее содержимое и загрузить его в `AppDomain`.

Информация об атрибутах сборки добавляется в файл `AssemblyInfo.cs`. Получить атрибуты сборки можно вызовом метода `GetCustomAttributes`.

Система отражения, на основе полученной информации о сборке, позволяет динамически создавать объекты даже из сборок, ссылок на которые не было до сих пор.

Система отражения позволяет динамически генерировать код на основе информации о типах, собранных посредством отражения. Подробнее об этом см. [НУР] и [Тро].

5.3. Введение в .NET Builder

В данном параграфе рассмотрим основы пакета расширения MATLAB Builder для .NET (называемого также .NET Builder), который позволяет из набора м-функций MATLAB создавать компоненты для .NET, которые могут быть затем использованы для разработки приложений на CLS-совместимых языках программирования.

Требования для MATLAB Builder для .NET. Системные требования и ограничения на использование MATLAB Builder для .NET включают все требования для Компилятора MATLAB, кроме того, должен быть установлен .NET Framework 1.1 или 2.0, а также среда программирования Visual Studio 2003 или Visual Studio 2005. Для работы MATLAB Builder для .NET требуется Компилятор MATLAB. Отметим, что MATLAB Builder для .NET доступен только на Windows.

Перед использованием .NET Builder необходима установка внешнего C/C++ компилятора для MATLAB командой `mbuild -setup`. Желательно установить компилятор MSVC. Это тот компилятор, который MathWorks использует наиболее часто в тестировании.

Для работы программы, использующей методы компонент, созданных на .NET Builder, на конечной машине пользователя требуется установить MCR – среду выполнения компонентов MATLAB.

.NET компоненты. .NET Builder преобразовывает функции MATLAB в .NET методы, которые инкапсулируют код MATLAB. Каждый *компонент* .NET Builder содержит один или более классов и каждый класс обеспечивает интерфейс для m-функций, которые добавляются к классу во время построения. Компонент создается в виде двух файлов, один из которых (dll) представляет сборку компонента, а другой – технологический файл (ctf) содержит все необходимое из MATLAB для работы этого компонента. Кроме того, для работы компонента должны быть установлена среда исполнения MCR MATLAB. По умолчанию создается частная сборка, но .NET Builder поддерживает также создание сборки свободного доступа.

.NET Builder обеспечивает устойчивое преобразование данных, индексацию и возможности форматирования массива для сохранения гибкости MATLAB при вызове методов из управляющего кода. Для поддержки типов данных MATLAB, .NET Builder имеет классы MWArray преобразования данных, которые определены в сборке MWArray пакета .NET Builder. Для преобразовать родных массивов в массивы MATLAB и наоборот, нужно сослаться на эту сборку в управляемом приложении (managed application). Отметим, что .NET Builder не поддерживает некоторые типы данных MATLAB, например, объектные типы данных MATLAB (Time Series Objects) и многие беззнаковые числовые типы.

.NET Builder обеспечивает также обычную обработку ошибок, как при стандартном управлении исключениями. .NET Builder может также использоваться для создания COM-компонент.

Среда разработки Deployment Tool. Для создания .NET компонента нужно написать одну или несколько m-функций MATLAB и затем, используя среду разработки Deployment Tool, создать проект в MATLAB Builder для .NET. При создании компонента .NET Builder преобразовывает функции MATLAB, которые определены в проекте компонента в методы, принадлежащие управляемому классу, который также создается в процессе построения. Имеется возможность использования и командной строки. Описание среды разработки Deployment Tool достаточно подробно изложено в предыдущих главах.

При создании компонента нужно задать имя класса и имя компонента. Это имя компонента также определяет имя сборки, которая осуществляет компонент. Имя класса обозначает название того класса, который инкапсулирует функции MATLAB.

Чтобы обратиться к методам созданного компонента, нужно создать экземпляр класса, созданного .NET Builder и затем вызвать методы, которые инкапсулируют функции MATLAB.

Использование классами среды выполнения MCR компонента MATLAB. При работе приложения создается единственный экземпляр среды выполнения MCR для каждого компонента .NET Builder. Этот экземпляр MCR многократно используется и доступен для всех классов в пределах компонента, что приводит к более эффективному использованию памяти и уменьшению затрат на запуск MCR для каждой последующей реализации класса. Все экземпляры класса совместно используют единственное рабочее пространство MATLAB и совместно ис-

пользуют глобальные переменные в m-файлах, используемых для построения компонента.

Отметим, что на конечных машинах инсталлятор MCR помещает сборку MWArray в каталог `MCR_directory\toolbox\dotnetbuilder\bin\version_number`.

5.3.1. Библиотека классов .NET MWArray

Библиотека классов .NET MWArray (сборка MWArray.dll) имеет два пространства имен:

- `MathWorks.MATLAB.NET.Arrays` – классы, которые обеспечивают доступ к массивам MATLAB из любого .NET CLS совместимого языка;
- `MathWorks.MATLAB.NET.Utility` – сервисные классы, оказывают общую поддержку классов MWArray в среде исполнения компонента MATLAB MCR.

Полную информацию об иерархии класса MWArray см. в документации MATLAB Builder for .NET в разделе MWArray Class Library Reference (см. также файл `C:\R2007a\help\toolbox\dotnetbuilder\MWArrayAPI\MWArrayAPI.chm`). При выборе данного элемента документации MATLAB, открывается окно справки, вид которого показан на рис. 5.3.1. При выборе файла MWArrayAPI.chm открывается обычное Windows-окно справки.

Рассмотрим эти пространства имен немного подробнее.

Пространство имен `MathWorks.MATLAB.NET.Arrays`. Содержит классы для поддержки преобразования данных между управляемыми типами и типами

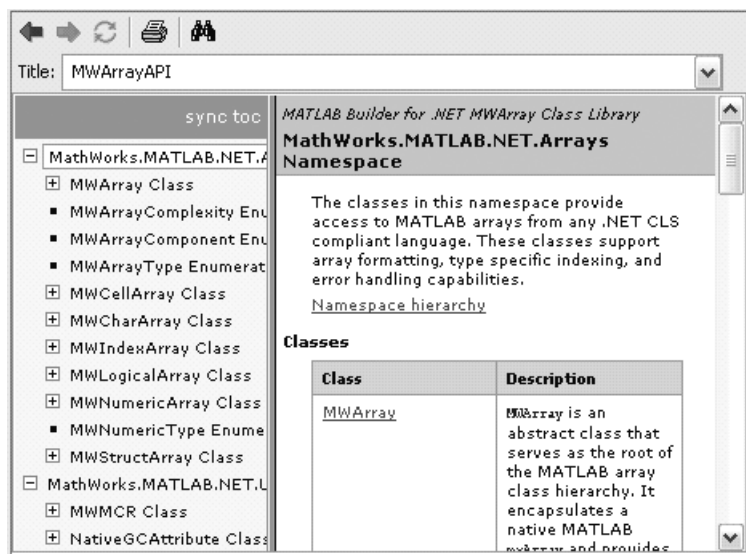


Рис. 5.3.1. Справочная система для классов MWArray

MATLAB. Каждый класс имеет конструкторы, деструкторы и набор свойств и методов для того, чтобы обращаться к состоянию основного массива MATLAB. Классы представляют стандартные типы массивов MATLAB:

- **MWArray** – это абстрактный класс, корень иерархии классов массивов MATLAB. Он инкапсулирует родной тип MATLAB `mxArray` и обеспечивает возможность обращения, форматирования и управления массивом;
- **MWNumericArray** – управляемое представление для массивов MATLAB числовых типов. Его эквивалент MATLAB – это заданный по умолчанию тип массива `double`, используемый большинством математических функций MATLAB;
- **MWLogicalArray** – управляемое представление для массивов MATLAB логического типа. Как и его эквивалент MATLAB, **MWLogicalArray** содержит только единицы и нули (`true/false`);
- **MWCharArray** – управляемое представление для массивов MATLAB символьного типа. Как и его эквивалент MATLAB, **MWCharArray** поддерживает создание строк и манипуляции со строками;
- **MWCellArray** – управляемое представление для массивов ячеек MATLAB. Каждый элемент в массиве ячеек – это контейнер, который может содержать **MWArray** или один из его производных типов, включая другой **MWCellArray**;
- **MWStructArray** – управляемое представление для массивов структур MATLAB. Как и его эквивалент MATLAB, он состоит из имен полей и значений полей;
- **MWIndexArray** – это абстрактный класс, который служит корнем для классов индексации **MWArray**. Эти классы, представьте типы массивов, которые могут использоваться как входные параметры для оператора индексации массива `[]`.

Данное пространство имен содержит также перечисления числовыми типами MATLAB: **MWArrayComplexity**, **MWArrayComponent**, **MWArrayType** и **MWNumericType**.

Пространство имен MathWorks.MATLAB.NET.Utility. Сервисные классы в этом пространстве имен оказывают общую поддержку классов **MWArray** в среде исполнения компонента MATLAB MCR. Содержит два класса:

- **MWMCR** – класс, который обертывает неуправляемый `mcrInstance` и обеспечивает управляемый интерфейс для создания экземпляра, инициализации и завершения MCR. Содержит два статических метода: **InitializeApplication** – для инициализации первого экземпляра MCR с определенным набором пользователя опций запуска и **TerminateApplication** – для закрытия экземпляра класса MCR. Имеет еще несколько методов: **Dispose** – для удаления объекта **MWMCR** и экземпляра `mcr` и методы **Equals**, **GetHashCode**, **GetType**, **ToString**, унаследованные от **Object**;
- **NativeGCAttribute Class** – определяет атрибут сборки, который явно вызывает сборщика мусора CLR, когда распределение родной динамической памяти для экземпляров классов массивов MATLAB достигло порога определенного, пользователем.

5.3.2. Правила преобразования данных

Как известно, простота и гибкость языка программирования MATLAB основана на очень удобных типах данных. К сожалению, эти типы данных доступны только в среде MATLAB. Для того, чтобы использовать обычные типы данных в методе, который создан из m-функции MATLAB, нужно передать данные в форме массива MATLAB. Для этих целей .NET Builder предлагает классы, в которых реализованы аналоги типов данных MATLAB, и которые могут быть использованы в .NET-совместимых языках программирования. Кроме того, .NET Builder имеет методы преобразования обычных типов данных в аналоги типов MATLAB. Для того, чтобы понять C#-коды следующего параграфа, рассмотрим здесь кратко основные правила преобразования данных. Более подробная информация имеется в справочной системе классов MWArray, см также раздел 5.5.5.

Соответствие управляемых типов и массивов MATLAB. Таблица 5.3.1 перечисляет правила преобразования, используемые при преобразовании исходных .NET типов в массивы MATLAB. Правила преобразования, перечисленные в этих таблицах применимы к скалярам, векторам, матрицам, и многомерным массивам исходных перечисленных типов.

Таблица 5.3.1. Соответствие управляемых типов и массивов MATLAB

Родной тип .NET	Массив MATLAB	Комментарии
System.Double	double	
System.Single	single	Доступны только, когда параметр конструктора makeDouble установлен как false. Значение по умолчанию есть true, что создает тип MATLAB double
System.Int64	int64	
System.Int32	int32	
System.Int16	int16	
System.Byte	int8	
System.String	char	
System.Boolean	logical	

Таблица 5.3.2 перечисляет соответствие типов данных, используемое при преобразовании массивов MATLAB в типы .NET. Правила преобразования применимы к скалярам, векторам, матрицам, и многомерным массивам перечисленных типов MATLAB.

Таблица 5.3.2. Правила преобразования массивов MATLAB в управляемые типы

Тип MATLAB	Тип .NET (Примитив)	Тип .NET (Класс)	Комментарии
cell	Net	MWCellArray	Массивы ячеек и структур
structure	Net	MWStructArray	MATLAB не имеют соответствующих типов .NET
char	System.String	MWCharArray	Значение по умолчанию – тип double
double	System.Double	MWNumericArray	Не поддерживается
single	System.Single	MWNumericArray	
uint64	System.Int64	MWNumericArray	

Таблица 5.3.2. Правила преобразования массивов MATLAB в управляемые типы (окончание)

Тип MATLAB	Тип .NET (Примитив)	Тип .NET (Класс)	Комментарии
uint32	System.Int32	MWNumericArray	Не поддерживается
uint16	System.Int16	MWNumericArray	Не поддерживается
uint8	System.Byte	MWNumericArray	
logical	System.Boolean	MWLogicalArray	
Function handle	Нет	Нет	
Object	Нет	Нет	

Приведем пример кода, который показывает, как преобразовать значение (5.0) типа `double` в `MWNumericArray`, а затем использовать его в методе, основанном на функции MATLAB:

```
MWNumericArray arraySize = 5.0;
magicSquare = magic.MakeSqr(arraySize);
```

В этом примере, метод MATLAB есть `magic.MakeSqr(arraySize)`.

Преобразование символов и строк. Родная строка .NET преобразуется в 1-на-N массив символов MATLAB, с числом N, равным длине .NET строки. Массив .NET строк (`string[]`) преобразуется в массив символов M-на-N, с M, равным числу элементов в массиве строк (`[]`) и N, равным максимальной длине строк в массиве. Многомерные массивы `String` преобразуются так же. Вообще, N-мерный массив строк преобразуется в (N+1)-мерный массив символов MATLAB с соответствующим дополнением нулями, когда у строк массива – различные длины.

Неподдерживаемые типы массивов MATLAB. MATLAB .NET Builder не поддерживает следующие типы массива MATLAB: `int8`, `uint16`, `uint32`, `uint64`, потому что они не CLS-совместимы.

Существуют некоторые типы данных, обычно используемые в MATLAB, которые не доступны как родные типы.NET. Например, это массивы ячеек, структур и массивы комплексных чисел. Эти типы массивов представлены как экземпляры класса `MWCellArray`, `MWStructArray`, и `MWNumericArray`, соответственно.

Автоматическое приведение типов к типам MATLAB. Обычно при использовании в программе C# в качестве входного параметра родного .NET примитива или массива, .NET Builder без участия пользователя преобразовывает его в экземпляр соответствующего класса `MWArray` для его передачи методу. .NET Builder может преобразовать большинство CLS-допустимых строк, числовых типов или многомерных массивов этих типов к соответствующим типам `MWArray`. Например, в следующей инструкции .NET:

```
result = theFourier.plotfft(3, data, interval);
```

третий параметр, а именно, `interval`, имеет родной тип `System.Double` .NET. Он приводится автоматически к типу массива 1-на-1 `double` MATLAB `MWNumericArray`. Отметим, что это преобразование делается без участия пользователя в приложениях C#, но может потребовать явного оператора приведения типов на других языках, например, в Visual Basic.

Классы преобразования данных. Для поддержки преобразования данных между управляемыми типами и типами MATLAB, .NET Builder обеспечивает ряд классов преобразования данных, производных от абстрактного класса, `MWArray`. Классы преобразования данных построены как иерархия классов, которая представляет главные типы массивов MATLAB. Корень иерархии – это абстрактный класс `MWArray`. У класса `MWArray` есть следующие подклассы, представляющие главные типы MATLAB: `MWNumericArray`, `MWLogicalArray`, `MWCharArray`, `MWCellArray`, и `MWStructArray`. Классы `MWArray` и `MWIndexArray` являются абстрактными. Другие классы представляют стандартные типы массивов MATLAB: ячеек, символов, логических, числовых и структур.

Классы преобразования данных `MWArray` позволяют передавать большинство родных типов .NET в качестве параметров в методы .NET Builder непосредственно, не используя явное преобразование данных. Есть неявный оператор приведения для большинства родных числовых и строковых типов, которые преобразуют родной тип в соответствующий массив MATLAB.

В случае явного преобразования числовых типов обычно используется конструктор `MWNumericArray`. При этом, по умолчанию любой числовой тип C# преобразуется в массив MATLAB `double`. Например, в следующем примере

```
int data = 24;
MWNumericArray mw_data = new MWNumericArray(data);
```

родное целое число (`int data`) преобразуется в тип `MWNumericArray`, содержащий массив 1-на-1 MATLAB `double`, который является значением типа MATLAB по умолчанию. Для того, чтобы сохранить целочисленный тип (а не преобразовать в заданный по умолчанию тип `double`), можно использовать дополнительные аргументы конструктора класса `MWNumericArray`.

Если нужно создать числовой массив MATLAB определенного типа, то дополнительный параметр `makeDouble` конструктора класса `MWNumericArray` устанавливается как `False`. Тогда родной тип определяет тип создаваемого массива MATLAB в соответствии с таблицей 5.3.1. Например, в следующем коде массив `mw_data` создается как 16-разрядный целочисленный 1-на-1 массив MATLAB:

```
short data = 24;
MWNumericArray mw_data = new MWnumericArray(data, False);
```

Для некоторых типов данных (массивы ячеек, структур и комплексных чисел) обычно используемых в MATLAB, которые не доступны как родные типы .NET, необходимо создавать экземпляры `MWCellArray`, `MWStructArray`, или `MWNumericArray`.

Возвращаемые данные от MATLAB в управляемый код. Все данные, возвращенные от метода MATLAB .NET Builder, представлены в виде экземпляров соответствующих подклассов `MWArray`. Например, массив ячеек MATLAB возвращается как объект `MWCellArray`.

Об индексации массива MATLAB. .NET Builder обеспечивает индексы для поддержки части индексации массивов MATLAB. Если нужно получить индексированный массив, то возвращенный массив MATLAB должен преобразовываться

к родному массиву используя метод `ToArray()`, поскольку этот метод сохраняет индексацию массива.

5.3.3. Интерфейсы, создаваемые .NET Builder

MATLAB поддерживает разные сигнатуры для вызовов функций. Когда .NET Builder обрабатывает m-код, он создает несколько перегруженных методов, которые осуществляют функции MATLAB. Каждый из этих перегруженных методов соответствует вызову общей функции MATLAB с определенным числом входных параметров. В дополнение к этим методам .NET Builder создает другой метод, который определяет возвращаемые значения функции MATLAB как входные параметры. Этот метод моделирует внешний интерфейс `feval` в MATLAB.

Для каждой функции MATLAB, которая определяется как часть .NET компонента, .NET Builder производит следующие интерфейсы, основанные на сигнатуре функции MATLAB:

- интерфейс единственного вывода, который предполагает, что требуется только единственный вывод и возвращает результат в единственном объекте `MWArray`, а не массиве из `MWArray`.
- стандартный интерфейс, который определяет входы типа `MWArray` и возвращаемые значения как массив из `MWArray`.
- интерфейс `feval`, который включает оба параметра ввода и вывода в список аргументов ввода вместо того, чтобы вернуть выходы как возвращаемые значения. Параметры вывода определены вначале, далее следуют входные параметры.

Интерфейс единственного вывода. Обычно *интерфейс единственного вывода* используется для функций MATLAB, которые возвращают единственный параметр. Его можно также использовать, когда необходимо использовать вывод функции как ввод к другой функции. Выводом является один объект `MWArray`.

Для функций MATLAB, .NET Builder производит интерфейсный класс, который реализует перегруженные методы для осуществления различных форм вызова общей функции MATLAB. Например, для общей функции `foo` MATLAB,

```
function Out = foo(In1, In2, ..., InN, varargin)
```

ниже показано несколько форм интерфейса единственного вывода, которые производит .NET Builder для `foo`.

Интерфейс, когда нет входных аргументов:

```
public MWArray foo();
```

Интерфейс, если имеются входные аргументы

```
public MWArray foo(MWArray In1, MWArray In2, ..., MWArray inN);
```

Интерфейс, если возможны дополнительные входные аргументы

```
public MWArray foo(MWArray In1, MWArray In2, ..., MWArray inN,
    params MWArray[] varargin);
```

В этом примере аргументы ввода In1, In2, и inN имеют типа объектов MWArray.

Точно так же, в случае дополнительных параметров, аргументы params имеют тип MWArray (параметр varargin подобен аргументу varargin в функции MATLAB – это позволяет пользователю передавать переменное число параметров).

Когда вызывается метод класса в .NET приложении, сначала определяются все обязательные вводы, а затем дополнительные параметры. Функции, имеющие единственный целочисленный ввод, требуют явного приведения типа в MWNumericArray, чтобы отличить сигнатуру метода от стандартной сигнатуры, когда нет никаких входных параметров, а единственное целое число показывает количество выводов.

Стандартный интерфейс. Обычно *стандартный интерфейс* используется для функций MATLAB, которые возвращают несколько значений вывода. При стандартным интерфейсом все параметры вывода возвращаются как массив элементов MWArray. Укажем несколько форм стандартного интерфейса для общей функции MATLAB,

```
function [Out1,Out2,..., vararginout] = foo(In1,In2,...,InN,  
varargin)
```

которые производит .NET Builder.

Без входных аргументов,

```
public MWArray[] foo(int numArgsOut);
```

С одним входным аргументом,

```
public MWArray [] foo(int numArgsOut, MWArray In1);
```

Более одного входного аргумента,

```
public MWArray[] foo(int numArgsOut, MWArray In1, MWArray In2,  
... , MWArray InN);
```

С дополнительными входными аргументами, представленными в параметре varargin,

```
public MWArray[] foo(int numArgsOut, MWArray in1, MWArray in2, ...,  
MWArray InN, params MWArray[] varargin);
```

Параметры в этих примерах стандартных сигнатур имеют следующий смысл:

- numArgsOut – целое число, указывающее число выводов. Параметр numArgsOut всегда должен быть первым в списке параметров;
- In1, In2, ..., InN – обязательные входные параметры. Все аргументы, которые следуют за numArgsOut в списке параметров, являются вводами к вызываемому методу. Сначала определяются все обязательные вводы. Каждый обязательный ввод должен иметь тип MWArray или один из его производных типов;
- varargin – дополнительные вводы. Можно определить также дополнительные вводы, если m-код использует ввод varargin: перечислить дополнительные вводы, или поместить их в параметр MWArray [], размещая этот массив последним в список параметров;

Пример. Если функция MATLAB, которую инкапсулирует компонент `MyComponentName` имела, например, сигнатуру

```
function [y1,y2,y3] = myfunc(x1, x2, x3, x4)
```

то вывод соответствующего метода будет массивом `MWArray[]` из трех элементов (массив ячеек). Каждый элемент сам является массивом некоторого размера и типа. Для того, чтобы выбрать массив, содержащийся, например, в первом выводе `y1` метода, нужно сначала выбрать этот элемент, а затем – преобразовать его в массив соответствующего типа. Это можно сделать, например, следующим образом:

```
MWArray[] mw_ArrayOut = null; // Объявление выходного массива
MWNumericArray mw_y1 = null; // Объявление массива первого аргумента
mw_ArrayOut = classInstance.myfunc (3, x1, x2, x3, x4); // Вызов метода
mw_y1 = (MWNumericArray)mw_ArrayOut[0]; // Выбор первого элемента из
// массива MWArray[] и его преобразование в числовой массив
```

Теперь `mw_y1` является числовым (`double`) массивом MATLAB и его можно преобразовать в обычный массив C# (см. раздел 5.5.5), либо использовать как аргумент в следующем методе .NET Builder. Обратите внимание, что массивы MATLAB `mw_ArrayOut` и `mw_y1` должны быть инициализированы.

Замечание. Если используется функция, которая имеет единственный вывод, то, вообще говоря, можно использовать как стандартную сигнатуру метода, так и сигнатуру с единственным выводом. В случае стандартной сигнатуры результат выводится в виде массива 1-на-1 `MWArray[]` и обращение к результату производится так же, как указано выше: нужно сначала выбрать первый (и единственный) элемент массива `MWArray[]`. Если используется сигнатура с единственным выводом, то результат выводится не как массив из одного элемента, а просто как массив `MWArray` того вида, который предусмотрен *m*-функцией. В этом случае к результату можно обратиться сразу, как указано в следующем примере.

```
MWArray mw_mw_y = null; // Объявление выходного массива
mw_y = classInstance.myfunc (x1, x2); // Вызов метода
```

Отметим, что использование стандартной сигнатуры предпочтительнее, поскольку при компиляции приложения в Visual C# с использованием методов с сигнатурой единственного вывода возникают ошибки.

Интерфейс *feval*. В дополнение к методам в интерфейсе единственного вывода и стандартном API, в большинстве случаев .NET Builder производит дополнительный перегруженный метод, *интерфейс feval*, который все переменные вывода ставит в число аргументов функции с атрибутом `ref`. Данный атрибут указывает, что эти массивы передаются как ссылочные и, следовательно, могут быть изменены во время работы метода. Если исходный *m*-код не содержит никаких параметров вывода, то .NET Builder не будет генерировать интерфейс *feval*. Для функции общего вида:

```
function [Out1, Out2, ..., varargout] = foo(In1, In2, ..., InN,
varargin)
```

.NET Builder генерирует следующий интерфейс *feval*:

```
public void foo(int numArgsOut, ref MWArray [] ArgsOut, MWArray[] ArgsIn)
```

где параметры следующие:

- `numArgsOut` – целое число, указывающее число выводов. При этом массив `varargout` считается как один параметр;
- `ref MWArray [] ArgsOut` – аргументы вывода. Массив `ArgsOut` – это все выводы исходного `m`-кода, в том же самом порядке, как они стоят в исходном `m`-коде. Атрибут `ref` указывается для всех параметров вывода, он показывает, что эти массивы передаются как ссылочные;
- `MWArray[] ArgsIn` – входные аргументы. Типы `MWArray` или поддерживаемые `.NET` примитивные типы.

5.3.4. Задание сборки компонента и пространства имен

Для использования сборки компонента произведенного с использованием `MATLAB Builder` для `.NET` из приложения, нужно сделать ссылки на пространства имен сборки `MWArray.dll` преобразования данных `MATLAB` и сервисных классов:

```
using MathWorks.MATLAB.NET.Utility;  
using MathWorks.MATLAB.NET.Arrays;
```

Необходимо также сделать ссылку на пространство имен сборки `.NET Builder`, созданной для этого компонента, например:

```
using MyComponentName;
```

Предположим, что созданный компонент назван `MyComponentName` и нужно использовать его в программе по имени `MyApp.cs`. Тогда в начале текста кода `MyApp.cs` нужно использовать следующие операторы:

```
using System;  
using MathWorks.MATLAB.NET.Utility;  
using MathWorks.MATLAB.NET.Arrays;  
using MyComponentName;
```

5.4. Создание консольных приложений

В этом параграфе рассмотрим на двух простых примерах работу `.NET Builder` по созданию компонент и консольных приложений, использующих эти компоненты. Примеры являются учебными для `.NET Builder` и находятся в каталоге `matlabroot\toolbox\dotnetbuilder\Examples\VS8(VS7)`.

Чтобы создать компонент, нужно написать `m`-функции и затем создать проект в `MATLAB Builder` для `.NET`, который инкапсулирует эти функции. Для этой цели удобно использовать среду разработки `Deployment Tool`. Запуск среды разработки производится командой `deploytool` в командной строке `MATLAB`. В результате открывается диалоговое окно `Deployment Tool`. Далее создается новый проект, указываются имя проекта, класса, каталога проекта. К проекту добавляются `m`-файлы, которые предполагается инкапсулировать. При необходимости устанавливаются дополнительные свойства построения и упаковки. Перед построе-

нием компонента проект сохраняется. При построении компонента создается C# код для интерфейсного класса в подкаталоге **src** проектного каталога. Создается сборка компонента, содержащая интерфейсные классы и stf-файл в подкаталоге **distrib** проектного каталога. Файлы в каталоге distrib представляют .NET компонент. Напомним, что stf-файл – есть технологический файл компоненты, который обязан поддерживать все необходимое из MATLAB, что требуется инкапсулированной функции MATLAB. В области окна Output отображаются сообщения о процесс построения и о любых возникающих проблемах.

При создании компонента нужно задать имя класса и имя компонента. Это имя компонента также определяет имя сборки, которая осуществляет компонент. Имя класса обозначает название того класса, который инкапсулирует функции MATLAB. Общие правила .NET рекомендуют использовать для первого символа в идентификаторе и первого символа каждого последующего составного слова прописные буквы. Например, MakeSquare.

Среда разработки предполагает возможность создания инсталляционного пакета. При этом, можно включить в пакет и MCR MATLAB в случае необходимости.

Для использования компонента, созданного на MATLAB Builder для .NET нужно его установить на машине разработки приложения и установить среду исполнения MCR. При создании приложения нужно добавить ссылку на .NET компонент в проекте Visual Studio или из командной строки CLS-совместимого компилятора. Необходимо также добавить ссылку на библиотеку классов MWArray.

Создание экземпляров класса и вызов методов класса .NET Builder производится точно так же, как и в случае любого обычного класса .NET. Для преобразования данных между родными типами .NET и типами массивов MATLAB нужно использовать классы преобразования данных MWArray.

Рассмотрим два достаточно простых примера создания компонент и консольных приложений, которые используют эти компоненты. Первый пример – вычисление магического квадрата. Второй пример посвящен матричной математике.

5.4.1. Пример магического квадрата

В этом разделе рассмотрим общие этапы создания компонент и приложений, использующих функции MATLAB на примере приложения, вычисляющего магический квадрат. Пример также показывает, как использовать этот компонент в типовом консольном приложении.

Создание .NET компонента

Перед использованием .NET Builder нужно установить внешний C/C++ компилятор для MATLAB командой `mbuild -setup`. Для рассматриваемых примеров установим компилятор Microsoft Visual C++ 2005. Создание компонента состоит из нескольких простых этапов.

1. Создание m-функций. На первом шаге нужно подготовить те m-функции, которые предполагается скомпилировать в .NET-методы. Поскольку мы рассмат-

риваем учебный пример MATLAB, то все m-функции уже имеются в каталогах примеров MATLAB. Отметим, из Microsoft Visual Studio .NET можно загрузить проекты для всех примеров, открывая файл DotNetExamples.sln из каталога matlabroot\toolbox\dotnetbuilder\Examples\VS8\.

Скопируем файлы для этого примера из соответствующего каталога примеров .NET Builder в рабочий каталог, в качестве которого возьмем D:\Work. MATLAB имеет два каталога (VS7 и VS8) примеров для двух версий Microsoft Visual Studio .NET: VS7 – для версии Microsoft Visual Studio .NET 2003 и VS8 – для Microsoft Visual Studio 2005. Для данного примера скопируем файлы из каталога matlabroot\toolbox\dotnetbuilder\Examples\VS8\MagicSquareExample в каталог D:\Work.

Для создания компонента используем m-функцию makesquare (она находится в рабочем каталоге MagicSquareExample\MagicDemoComp\makesquare.m). Код для функции makesquare:

```
function y = makesquare(x)
y = magic(x);
```

Напомним, что встроенные функции MATLAB не компилируются. Поэтому и создается такая m-функция, вызывающая встроенную функцию MATLAB magic.m.

2. Создание проекта. В MATLAB установим текущим рабочим каталогом новый подкаталог MagicSquareExample в рабочем каталоге D:\Work. Вызовем среду разработки, исполнив команду `deploytool` в командной строке MATLAB.

Создадим новый проект, нажимая кнопку **New Project** в инструментальной панели. Открывается диалоговое окно **New Deployment Project**. Выбираем создание .NET компонента. Определяем имя компонента MagicDemoComp. Рабочий каталог – D:\Work\MagicSquareExample\. После установки этих данных в диалоговом окне **Deployment Tool** появляются папки проекта. По умолчанию папка класса имеет имя MagicDemoCompclass. Используя правую кнопку мыши, изменим это имя класса на MagicSquare. Выберем также опцию **Generate verbose output** – для подробного вывода сообщений о процессе компиляции.

Добавим к проекту файл makesquare.m (из подкаталога MagicDemoComp) используя кнопку **Add** инструментальной панели и сохраним проект (рис. 5.4.1).

3. Построение .NET компонента.

Для этого достаточно нажать кнопку **Build** на инструментальной панели **Deployment Tool**. Начинается процесс построения и в области окна **Deployment Tool Output** (внизу рабочей области MATLAB) появляется log-файл регистрации процесса построения. В случае успешного завершения

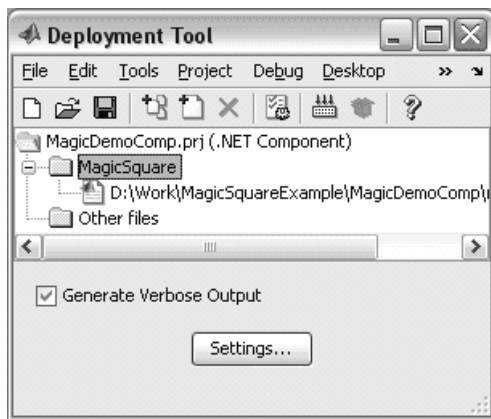


Рис. 5.4.1. Проект MagicDemoComp

построения в каталоге проекта, D:\Work\MagicSquareExample\, создается подкаталог проекта MagicDemoComp, его имя соответствует названию проекта. В процессе построения в каталоге MagicDemoComp создаются два новых подкаталога, **src** и **distrib**, и ряд файлов, которые помещаются в эти подкаталоги.

Каталог **distrib** содержит следующие файлы, созданные Компилятором из m-файла magicssquare.m:

- **MagicDemoComp.dll** – файл управляемой сборки (рис. 5.4.2), библиотека, содержащая метод, соответствующий функции magicssquare.m;
- **MagicDemoComp.ctf** – архив ctf. Этот файл содержит сжатый и зашифрованный архив m-файлов, которые составляют компонент, а также содержит другие файлы, от которых зависят основные m-файлы и еще ряд файлов, необходимые во время выполнения.

Каталог **src** содержит следующие файлы, созданные Компилятором из m-файла magicssquare.m:

- **MagicDemoComp_mcc_component_data.cs** – C# файл данных компонента, содержащий данные, необходимые MCR для инициализации и использования компонента. Эти данные включают ключи шифрования, информацию о путях и другую информацию о создаваемом компоненте для MCR;
- **MagicSquare.cs** – C# файл, содержащий интерфейсный класс MagicSquare и реализующий различные интерфейсы, соответствующие m-функции makesquare;
- **build.log** – файл отчета о процессе построения;
- **mccExcludedFiles.log** – файл, который содержит список различных функций toolbox-ов, которые не включены в файл CTF. Некоторые из этих функций могут быть из toolbox-ов, которые не используются в приложении. Причина этого в том, что эти toolbox-ы перегружают некоторые методы, которые вызывает код;
- **MagicDemoComp.xml** – XML файл документации .NET компонента;
- **readme.txt** – содержит необходимую информацию для распространения компонента;
- **MagicDemoComp.dll** и **MagicDemoComp.ctf** – копии файлов.

Замечание 1. При разработке проекта можно определить компиляцию частной или общедоступной сборки (меню **Project** ⇒ **Settings**). Частная сборка копируется при создании приложения в подкаталог приложения и принадлежит исключительно приложению. Общедоступная сборка обычно постоянно находится в каталоге C:\WINDOWS\assembly\GAC, называемом Глобальным Кэшем Сборок (Global Assembly Cache), и может быть непосредственно использована многими приложениями. По умолчанию создается частная сборка.

Замечание 2. Можно создать инсталляционный пакет. Для этого имеется кнопка **Package** на инструментальной панели. Создается файл _install.bat – для запуска инсталляции и самораспаковывающийся файл MagicDemoComp_pkg.exe, который содержит файлы MagicDemoComp.dll и MagicDemoComp.ctf. Для того, чтобы включить в инсталляционный пакет архив MCR, нужно выбрать соответствующую опцию (Install MCR) в установках **Settings**. Для добавления в пакет других

файлов, например MagicDemoComp.xml, их нужно добавить при помощи **Project** ⇒ **Settings/Packing**).

Замечание 3. Для установки компонента на другой машине нужно запустить инсталлятор компонента. При этом выполняются следующие операции. Если среда исполнения MCR MATLAB еще не установлена и если она была включена в инсталляционный пакет, то инсталлятор компонента устанавливает MCR, затем устанавливает файлы пакета в каталог, из которого запущен инсталлятор и копирует сборку MWArray в глобальный кэш сборки (C:\WINDOWS\assembly\GAC_32\), как часть установки MCR.

Отметим также, что при установке инсталлятор MCR помещает сборку MWArray в каталог installation_directory\toolbox\dotnetbuilder\bin\version_number.

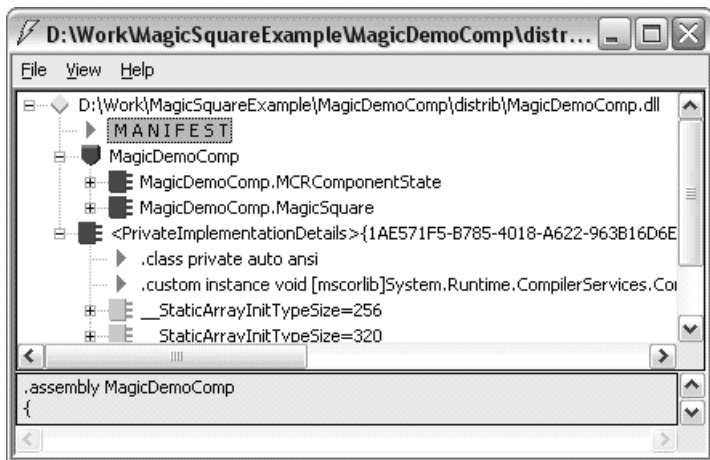


Рис. 5.4.2. Структура сборки MagicDemoComp.dll

Использование компонента в приложении

Для использования в приложении созданного .NET компонента необходимо написать соответствующий C#-код. Для нашего учебного примера исходный C# текст MagicDemoApp.cs приложения находится в каталоге MagicSquareExample\Magic DemoCSharpApp\. Отметим, что можно обратиться к компоненту из любого CLS-допустимого .NET языка, например, Visual Basic.

Приведем листинг кода приложения MagicDemoApp.cs.

```
// MagicDemoApp.cs
```

```
using System;
using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays;
using MagicDemoComp;
```

```

namespace MathWorks.Demo.MagicSquareApp
{
    /// <summary>
    /// MagicSquareApp демонстрационный класс вычисляет
    /// магический квадрат используя указанный размер.
    /// </summary>
    /// <remarks>
    /// args[0] - положительное целое число, размер массива.
    /// </remarks>
    class MagicDemoApp
    {
        #region MAIN
        /// <summary>
        /// Главная точка входа приложения
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            MWNumericArray arraySize= null;
            MWNumericArray magicSquare= null;

            try
            {
                // Проверка аргументов командной строки
                arraySize= (0 != args.Length) ? System.Double.Parse(args[0]) : 4;

                // Создание объекта магического квадрата
                MagicSquare magic= new MagicSquare();

                // Вычисление магического квадрата и печать матрицы
                magicSquare = (MWNumericArray)magic.makesquare((MWArray)arraySize);
                Console.WriteLine("Magic square of order {0}\n\n{1}", arraySize,
                magicSquare);

                // Преобразование массива магического квадрата в двумерный родной
                // массив double
                array double[,] nativeArray =
                    (double[,])magicSquare.ToArray(MWArrayComponent.Real);

                Console.WriteLine("\nMagic square as native array:\n");

                // Вывод на экран элементов массива:
                for (int i= 0; i < (int)arraySize; i++)
                    for (int j= 0; j < (int)arraySize; j++)
                        Console.WriteLine("Element({0},{1})= {2}", i, j, nativeArray[i,j]);

                Console.ReadLine(); //Ожидание пользователя для выхода из приложения
            }
        }
    }
}

```

```

catch(Exception exception)
{
    Console.WriteLine("Error: {0}", exception);
}
}

```

Построим приложение, используя Visual Studio 2005. Для этого откроем проектный файл примера магического квадрата (MagicDemoCSharpApp.csproj) в Visual Studio 2005. Нужно добавить в проекте ссылку на компонент MWArray: matlabroot\toolbox\dotnetbuilder\bin\architecture\version_number. Для этого в Visual Studio необходимо из меню **Project** ⇒ **Add Reference** открыть диалоговое окно **Add Reference** и в нем указать на файл MWArray.dll из каталога C:\R2007a\toolbox\dotnetbuilder\bin\win32\v2.0\ (рис. 5.4.3).

Также необходимо добавить в проекте ссылку на компонент магического квадрата (MagicDemoComp.dll), который находится в подкаталоге distrib. Для этого в диалоговом окне **Add Reference** открыть вкладку **Browse** (рис. 5.4.3) и найти файл MagicDemoComp.dll в каталоге .\MagicSquareExample\MagicDemoComp\distrib\.

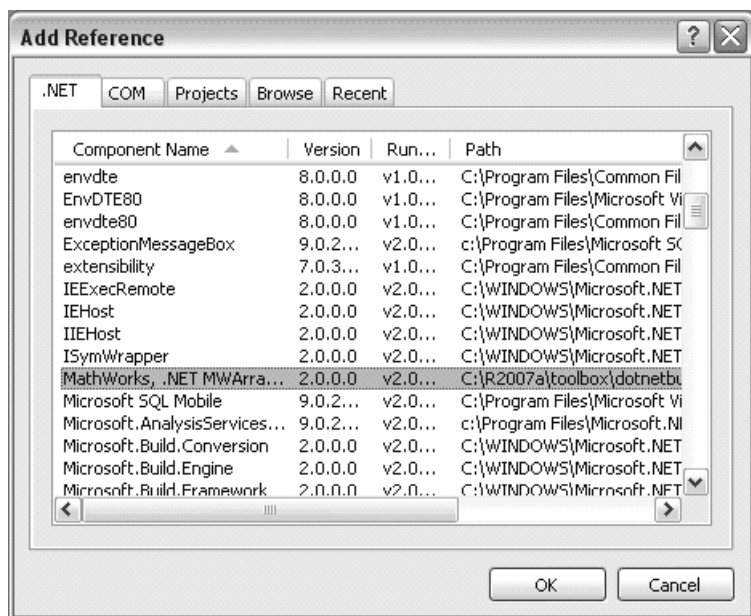


Рис. 5.4.3. Добавление ссылок на компоненты в Visual Studio 2005

Теперь построим и выполним приложение в Visual Studio 2005. Для этой цели служит кнопка **Start Debugging** инструментальной панели Visual Studio, а также меню **Build**. При использовании **Start Debugging** создается сборка GetMagicApp.exe, которая сохраняется в подкаталоге .\MagicDemoCSharpApp\bin\Debug\, а при использовании меню **Build** сборка GetMagicApp.exe сохраняется в подкаталоге

.\MagicDemoCSharpApp\bin\Release\). Строение сборки GetMagicApp.exe показано на рис. 5.4.4. После запуска этого приложения и задания, например, числа 4 в командной строке, результат будет такой, как на рис. 5.4.5. Данное консольное приложение ожидает закрытия пользователем (нажатия клавиши **Enter**).

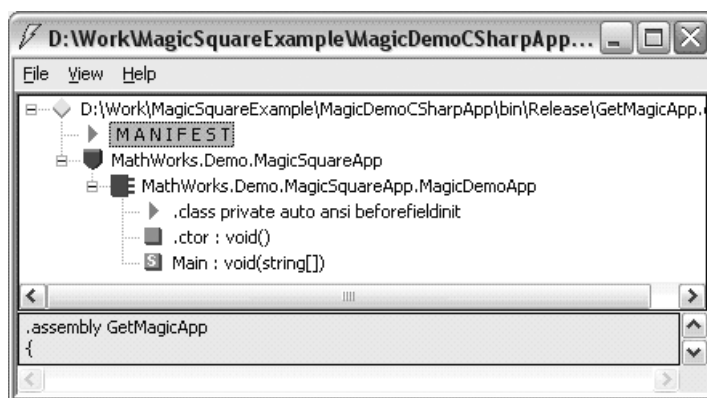


Рис. 5.4.4. Структура сборки GetMagicApp.exe

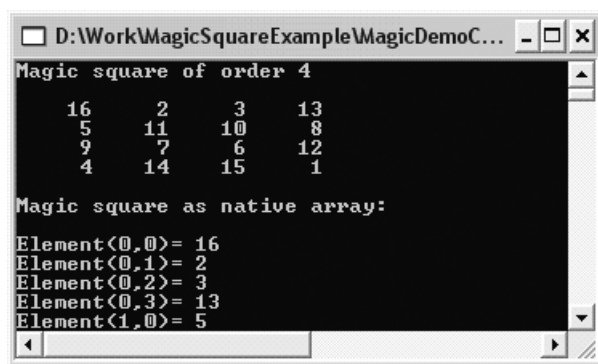


Рис. 5.4.5. Результат работы приложения GetMagicApp.exe

Обратите внимание, что созданная ранее частная сборка MagicDemoComp.dll и технологический файл MagicDemoComp.ctf компонента скопированы в подкаталог приложения MagicSquareExample\bin\Release(Debug).

При первом запуске приложения создается каталог MagicDemo_MCR в подкаталоге MagicSquareExample\bin\Release(Debug), куда распаковывается архив MagicDemoComp.ctf. Этот подкаталог MagicDemo_MCR содержит зашифрованные версии m-файлов, которые необходимы приложению для выполнения метода magic.makesquare.

Замечание. Разобранный пример является учебным и для него уже имеется файл проекта (и для следующих примеров этого параграфа). Однако можно создать проект и самостоятельно в Visual Studio 2005. Для этого можно использовать мастер создания проектов для создания консольного приложения C#. При этом будут созданы все необходимые файлы. В основной файл C#, в котором находится метод Main, нужно внести основные блоки файла приложения MagicDemoApp.cs.

5.4.2. Пример матричной математики

В этом примере построим компонент .NET, который инкапсулирует функции MATLAB, выполняющие матричные разложения Холецкого, LU и QR. Программа вызывает эти методы для выполнения разложений простой тридиагональной матрицы (матрица конечных разностей) следующей формы:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Размер матрицы задается в командной строке, программа создает матрицу A и выполняет эти три разложения на множители. Оригинальная матрица A и результаты печатаются на стандартный вывод (на экран). Можно выполнить вычисления, считая эту матрицу разреженной, для этого вторым параметром в командной строке нужно указать строку «sparse».

Функции MATLAB, которые будут инкапсулироваться:

```
function [L] = Cholesky(A)
L = chol(A);

function [L,U] = LUDecomp(A)
[L,U] = lu(A);

function [Q,R] = QRDecomp(A)
[Q,R] = qr(A);
```

Напомним, что встроенные функции MATLAB не компилируются. Поэтому и создаются такие m-функции, где каждая вызывает только одну встроенную функцию MATLAB.

Создание .NET компонента

Предполагается, что для Компилятора MATLAB установлен внешний компилятор Microsoft Visual C++ 2005. Для создания компонента нужно выполнить несколько простых этапов.

1. Создание m-функций. На первом шаге нужно подготовить m-функции, которые предполагается скомпилировать в .NET-методы. Поскольку мы рассматриваем учебный пример MATLAB, то все m-функции уже имеются в каталогах при-

меров MATLAB. Скопируем следующий подкаталог MATLAB `matlabroot\toolbox\dotnetbuilder\Examples\VS8\MatrixMathExample` в рабочий каталог `D:\Work`. Коды m-функций находятся в рабочем каталоге в `MatrixMathExample\MatrixMathDemoComp\`.

2. Создание проекта. В MATLAB установим текущим рабочим каталогом новый подкаталог `MatrixMathExample` в рабочем каталоге `D:\Work`. Вызовем среду разработки, исполнив команду `deploytool` в командной строке MATLAB.

Создадим новый проект со следующими установками: имя компонента – `MatrixMathDemoComp`; имя класса – `Factor`; каталог проекта – подкаталог `\MatrixMathExample\MatrixMathDemoComp` в рабочем каталоге; подробный вывод.

Добавим файлы `cholesky.m`, `ludcomp.m` и `qrdecomp.m` к проекту. Сохраним проект.

3. Построение .NET компонента. Для этого достаточно нажать кнопку **Build** на инструментальной панели **Deployment Tool**. Начинается процесс построения и в области окна **Deployment Tool Output** (внизу рабочей области MATLAB) появляется log-файл регистрации процесса построения. В случае успешного завершения построения в каталоге проекта, `D:\Work\MatrixMathExample`, создается подкаталог проекта `MatrixMathDemoComp`, его имя соответствует названию проекта. Созданные файлы компонента помещаются в два вновь создаваемых подкаталога, **src** и **distrib**, в каталоге `MatrixMathDemoComp`. Каталог **distrib** содержит два файла: `MatrixMathDemoComp.dll` – файл библиотеки, содержащей методы, соответствующие функциям `cholesky.m`, `ludcomp.m` и `qrdecomp.m` и файл `MatrixMathDemoComp.ctf` – архив CTF.

Использование компонента в приложении

Для использования в приложении созданного .NET компонента необходимо написать соответствующий C#-код. Для нашего учебного примера исходный C# текст `MatrixMathDemoApp.cs` примера приложения находится в подкаталоге `MatrixMathExample\MatrixMathDemoCSharpApp\`. Приведем листинг кода приложения `MatrixMathDemoApp.cs` (опуская некоторые информационные XML-теги).

```
// MatrixMathDemoApp.cs

using System;
using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays;

using MatrixMathDemoComp;
    // Отключение автоматического управления родной памятью
    [assembly: NativeGC(false)]
namespace MathWorks.Demo.MatrixMathApp
{
    /// <remarks>
    /// Аргументы командной строки:
```

```
/// <newpara></newpara>
/// args[0] - Порядок N матрицы
/// <newpara></newpara>
/// args[1] - (дополнительный) sparse; Для разреженных матриц
/// </remarks>
class MatrixMathDemoApp
{
    #region MAIN

    [STAThread]
    static void Main(string[] args)
    {
        bool makeSparse= true;
        int matrixOrder= 4;
        MWNumericArray matrix= null; // Матрица для разложения
        MWArray argOut= null; // Сохраняет один результат разложения
        MWArray[] argsOut= null; // Для нескольких результатов разл.

        try
        {
            //Если не задан параметр, то по умолчанию, N=4
            if (0 != args.Length)
            {
                // Преобразование порядка матрицы
                matrixOrder= System.Int32.Parse(args[0]);
                if (0 >= matrixOrder)
                {
                    throw new ArgumentOutOfRangeException("matrixOrder",
                        matrixOrder, "Must enter a positive integer
                        for the matrix order(N)");
                }
            }
            makeSparse= ((1 < args.Length) && (args[1].Equals("sparse")));
        }
        // Создание тестовой матрицы. Если второй аргумент есть "sparse",
        // создается разреженная матрица.
        matrix= (makeSparse) ? MWNumericArray.MakeSparse(matrixOrder,
            matrixOrder, MWArrayComplexity.Real, (matrixOrder+(2*(matrixOrder-1))))
            : new MWNumericArray(MWArrayComplexity.Real,
                MWNumericType.Double, matrixOrder, matrixOrder);

        // Инициализация тестовой матрицы
        for (int rowIdx= 1; rowIdx <= matrixOrder; rowIdx++)
            for (int colIdx= 1; colIdx <= matrixOrder; colIdx++)
                if (rowIdx == colIdx)
                    matrix[rowIdx, colIdx]= 2.0;
                else if ((colIdx == rowIdx+1) || (colIdx == rowIdx-1))
                    matrix[rowIdx, colIdx]= -1.0;

        // Создание нового объекта factor
```

```

Factor factor = new Factor();

    // Печать тестовой матрицы
    Console.WriteLine("Test Matrix:\n{0}\n", matrix);

    // Вычисление и печать разложения Холецкого
    argOut = factor.cholesky((MWArray)matrix);

    Console.WriteLine("Cholesky Factorization:\n{0}\n", argOut);

    // Вычисление и печать LU разложения, другой синтаксис вывода
    argsOut = factor.ludecomp(2, matrix);

    Console.WriteLine("LU Factorization:\nL Matrix:\n{0}\nU
    Matrix:\n{1}\n", argsOut[0], argsOut[1]);

    MWNumericArray.DisposeArray(argsOut);

    // Вычисление и печать QR разложения
    argOut = factor.qrdecomp(2, matrix);

    Console.WriteLine("QR Factorization:\nQ Matrix:\n{0}\nR
    Matrix:\n{1}\n", argsOut[0], argsOut[1]);

    Console.ReadLine();
}

catch(Exception exception)
{
    Console.WriteLine("Error: {0}", exception);
}

finally
{
    // Освобождение родных ресурсов
    if (null != (object)matrix) matrix.Dispose();
    if (null != (object)argOut) argOut.Dispose();
    MWNumericArray.DisposeArray(argsOut);
}

}

#endregion
}
}

```

Отметим, что оператор

```
Factor factor = new Factor();
```

создает экземпляр класса `Factor`. Следующие операторы вызывают методы, которые инкапсулируют функции MATLAB:

```
argOut = factor.cholesky((MWArray)matrix);  
argsOut = factor.ludcomp(2, matrix);  
argsOut = factor.qrdecomp(2, matrix);
```

Программа MatrixMathDemo принимает один или два параметра из командной строки. Первый параметр преобразуется в целочисленный порядок тестовой матрицы A. Если задается второй параметр в виде строки `sparse`, то создается разреженная матрица, содержащая тестовый массив A. Затем вычисляются разложения Холецкого, LU и QR и отображаются результаты. Программа имеет три части.

Первая часть генерирует матрицу A, создает новый объект `factor`, и вызывает методы `cholesky`, `ludcomp` и `qrdecomp`. Эта часть выполняется в блоке `try`. Если происходит исключение во время выполнения, то выполняется соответствующий блок `catch`.

Вторая часть – блок `catch`. Код печатает сообщение на стандартный вывод, что-бы сообщить пользователю об ошибке, если она произошла.

Третья часть есть блок `finally`, чтобы вручную очистить родные ресурсы перед выходом.

Построим приложение MatrixMathDemoApp, используя Visual Studio 2005. Для этого откроем проектный файл этого примера MatrixMathDemoCSharpApp.csproj в Visual Studio 2005. Нужно добавить в проекте ссылку на компонент MWArray: `matlabroot\toolbox\dotnetbuilder\bin\win32\v2.0\`. Для этого в Visual Studio необходимо из меню **Project** ⇒ **Add Reference** открыть диалоговое окно **Add Reference** и в нем указать на файл MWArray.dll из указанного каталога (рис. 5.4.3).

Также необходимо добавить в проекте ссылку на компонент MatrixMathDemoComp.dll, который находится в подкаталоге `distrib` каталога `D:\Work\MatrixMathExample\MatrixMathDemoComp\`. Для этого в диалоговом окне **Add Reference** открыть вкладку **Browse** (рис. 5.4.3) и найти файл MatrixMathDemoComp.dll.

Теперь построим и выполним приложение в Visual Studio 2005. Для этой цели служит кнопка **Start Debugging** инструментальной панели Visual Studio, а также меню **Build**. При использовании **Start Debugging** создается сборка MatrixMathDemoApp.exe, которая сохраняется в подкаталоге `\MatrixMathExample\MatrixMathDemoCSharpApp\bin\Debug`, а при использовании меню **Build** сборка PlotDemoApp.exe сохраняется в подкаталоге `\MatrixMathExample\MatrixMathDemoCSharpApp\bin\Release\`.

Для запуска этого приложения нужно, находясь в каталоге приложения, выполнить следующую команду в строке DOS:

```
MatrixMathDemoApp N
```

Результат выполнения при N=5 выглядит следующим образом, рис. 5.4.6. При нажатии клавиши Enter для завершения программы, мы возвращаемся в командную строку DOS, консольное окно при этом не закрывается.

```

C:\WINDOWS\system32\cmd.exe - MatrixMathDemoApp 5
D:\Work\MatrixMathExample\MatrixMathDemoCSharpApp\bin\
5
Test Matrix:
 2   -1   0   0   0
-1   2  -1   0   0
 0  -1   2  -1   0
 0   0  -1   2  -1
 0   0   0  -1   2

Cholesky Factorization:
1.4142  -0.7071   0   0   0
 0   1.2247  -0.8165   0   0
 0   0   1.1547  -0.8660   0
 0   0   0   1.1180  -0.8944
 0   0   0   0   1.0954

LU Factorization:
L Matrix:
1.0000   0   0   0   0
-0.5000  1.0000  0   0   0

```

Рис. 5.4.6. Результат работы приложения MatrixMathDemoApp.exe

5.4.3. Использование командной строки для создания компоненты .NET

Вместо того, чтобы использовать среду разработки Deployment Tool для создания .NET компоненты можно использовать команду msc. В этом разделе рассмотрим синтаксис команды msc и опции, которые требуются для создания .NET компонент.

Следующая строка определяет полный синтаксис команды msc со всеми необходимыми и дополнительными параметрами, используемыми для создания .NET компонент. Скобки указывают дополнительные (необязательные) части синтаксиса. Далее идет объяснение каждой части этого синтаксиса.

```

msc -W 'dotnet:component_name,class_name, 0.0|1.1|2.0,
Private|Encryption_Key_Path' file1
[file2...fileN][class{class_name:file1 [,file2,...,fileN]}], ... ,
[-d output_dir_path] -T link:lib

```

Опция -W. Указывает компилятору создать обертку функции. Эта опция принимает строковый параметр, который определяет следующие характеристики компонента.

- dotnet: – ключевое слово (сопровождается двоеточием), которое сообщает компилятору тип создаваемого компонента;
- component_name – определяет название компонента и его пространства имен, которое является разделенным точками именем, таким как например companyname.groupname.component;

- `class_name` – определяет имя класса .NET, который будет создан;
- `0.0|1.1|2.0` – определяет версию .NET Framework, которую предполагается использовать, чтобы компилировать компонент. Можно определить одно из трех значений: 0.0 – использование последней версии; 1.1 – версия 1.1 Framework; 2.0 – версия 2.0 Framework;
- `Private|Encryption_Key_Path` – для создания общедоступной сборки нужно определить полный путь к файлу ключа шифрования используемого для сборки.

Строка `file1 [file2...fileN]`. Определяет m-файл или m-файлы, которые должны инкапсулироваться как методы в создаваемом классе, (`class_name`).

Строка `class{class_name:file1 [,file2,...,fileN]},...` Необязательная. Определяет дополнительные классы, которые включаются в компонент. Чтобы использовать эту опцию, определяется имя класса, сопровождаемое двоеточием и затем названиями файлов, которые включаются в класс. Можно включить это многократно, чтобы определить несколько классов.

Строка `[-d output_dir_path]`. Необязательная. Указывает .NET Builder создавать каталог и копировать в него выходные файлы. Отметим, что при использовании `mcs` вместо среды разработки, каталоги `project_directory\src` и `project_directory\distrib` автоматически не создаются.

Опция `-T`. Определяет тип вывода. Чтобы создать .NET компонент, определите ключевое слово `link:lib`, которое связывает объекты в общедоступную библиотеку (DLL).

Использование файлов групп. Для того, чтобы упростить командную строку для создания .NET компоненты, можно использовать файл группы .NET Builder (пакетный файл), имеющий имя `dotnet`. Используя этот файл, необходимо указывать все четыре части текстовой строки параметра `-W`, однако не нужно определять опцию `-T`.

Следующий пример создает .NET компонент, названный `mycomponent`, содержащий единственный класс .NET, названный `myclass` с методами `foo` и `bar`. Когда используется опция `-B`, слово `dotnet` определяет название уже имеющегося файла группы .NET Builder.

```
mcs -B 'dotnet:mycomponent,myclass,2.0,encryption_keyfile_path'
foo.m bar.m
```

В этом примере .NET Builder использует версию 2.0 .NET Framework для компиляции компонента в общедоступную сборку, используя файл ключей, определенный в `encryption_keyfile_path`.

Следующий пример создает .NET компонент из двух m-файлов `foo.m` и `bar.m`.

```
mcs -B 'dotnet:mycompany.mygroup.mycomponent,myclass,0.0,Private'
foo.m bar.m
```

Пример создает .NET компонент, названный `mycomponent`, у которого есть следующее пространство имен: `mycompany.mygroup`. Компонент содержит единственный .NET класс, `myclass`, который содержит методы `foo` и `bar`.

Для использования класса `myclass` нужно поместить следующий оператор в C#-код:

```
using mycompany.mygroup;
```

Следующий пример создает .NET компонент, который включает более одного класса. Этот пример использует дополнительный параметр `class{...}` к команде `mcc`.

```
mcc -B 'dotnet:mycompany.mycomponent,myclass,2.0,Private' foo.m  
bar.m class{myclass2:foo2.m,bar2.m}
```

Этот пример создает .NET компонент `mycomponent` с двумя классами:

- `myclass` – имеет методы `foo` and `bar`;
- `myclass2` – имеет методы `foo2` and `bar2`.

5.5. Некоторые вопросы программирования с компонентами .NET Builder

В этом разделе мы обсудим некоторые вопросы программирования, характерные при использовании компонентов .NET Builder: обязательные элементы программы, использующей компонент .NET Builder; преобразование родных типов данных к типам данных MATLAB и наоборот; определение типов и использование `MWArray` для обработки параметров; обработка ошибок и управление родными ресурсами.

5.5.1. Обязательные элементы программы

Для использования .NET компонента, созданного и упакованного средствами .NET Builder, нужно установить его машине разработки приложения. Компонент должен содержать `ctf`-файл и `dll`-файлы (`componentname.ctf` и `componentname.dll`). Кроме того, должна быть установлена среда исполнения MATLAB MCR (файл `MCRInstaller.exe`, он находится в каталоге `matlab\toolbox\compiler\deploy\win32\`).

Программа, использующая методы, созданные при помощи .NET Builder, должна обязательно содержать следующие три элемента.

1. Ссылки на компоненты. Для использования сборки компонента (например, `MyComponentName`), созданного с использованием .NET Builder, в программе нужно сделать ссылку на саму используемую сборку и на компонент `MWArray.dll`, содержащий классы `MWArray`. Он находится в каталоге `<MATLAB>\toolbox\dotnetbuilder\bin\win32\v2.0\`, либо в аналогичном каталоге `C:\Program Files\MATLAB\MATLAB Component Runtime\v76\` среды исполнения MCR. В код программы нужно включить строки, указывающие на соответствующие пространства имен, например:

```
using MathWorks.MATLAB.NET.Utility;  
using MathWorks.MATLAB.NET.Arrays;  
using MyComponentName;
```

2. Создание экземпляра класса. Как и для любого класса .NET, перед использованием класса, созданного .NET Builder, нужно сначала создать экземпляр этого класса, например,

```
MyComponentClass classInstance = new MyComponentClass();
```

Один и тот же экземпляр класса можно использовать для нескольких методов в одном блоке программы.

3. Вызов функции компонента .NET Builder. Если функция MATLAB, которую инкапсулирует компонент MyComponentName имела, например, сигнатуру

```
function [y1,y2,y3] = myfunc(x1, x2, x3, x4)
```

то вывод соответствующего метода класса MyComponentClass будет массивом MWArray[] из трех элементов (массив ячеек). Каждый элемент сам является массивом некоторого размера и типа. Для того, чтобы выбрать массив, содержащийся, например, в первом выводе y1 метода, нужно сначала выбрать этот элемент, а затем – преобразовать его в массив соответствующего типа. Это делается следующим образом:

```
MWArray[] mw_ArrayOut = null; // Объявление выходного массива
MWNumericArray mw_y1 = null; // Объявление массива первого аргумента y1
mw_ArrayOut = classInstance.myfunc (3, x1, x2, x3, x4); // Вызов метода
mw_y1 = (MWNumericArray)mw_ArrayOut[0]; // Выбор первого элемента из
// массива MWArray[] и его преобразование в числовой массив
```

Теперь mw_y1 является числовым (double) массивом MATLAB и его можно преобразовать в обычный массив C# (см. раздел 5.6.5), либо использовать как аргумент в следующем методе .NET Builder. Обратите внимание, что массивы MATLAB mw_ArrayOut и mw_y1 должны быть инициализированы.

Замечание. Если используется функция, которая имеет единственный вывод, то, вообще говоря, можно использовать как стандартную сигнатуру метода, так и сигнатуру с единственным выводом. В случае стандартной сигнатуры результат выводится в виде массива 1-на-1 MWArray[] и обращение к результату производится так же, как указано выше, нужно сначала выбрать первый (и единственный) элемент массива MWArray[]. Если используется сигнатура с единственным выводом, то результат выводится не как массив из одного элемента, а просто как массив MWArray того вида, который предусмотрен m-функцией. В этом случае к результату можно обратиться сразу, как указано в следующем примере.

```
// Вычисление магического квадрата и печать матрицы
magicSquare = (MWNumericArray)magic.makesquare((MWArray)arraySize);
Console.WriteLine("Magic square of order {0}\n\n{1}", arraySize,
    magicSquare);
```

Отметим, что использование стандартной сигнатуры предпочтительнее, поскольку при компиляции Windows-приложения в Visual C# с использованием методов с сигнатурой единственного вывода часто возникают ошибки.

5.5.2. Передача входных параметров

Сигнатура вызова для методов, основанных на функциях MATLAB, использует один из классов преобразования данных MATLAB для передачи аргументов и возвращения вывода. Следующий пример явно создает числовую константу, используя конструктор класса `MWNumericArray` с аргументом `System.Int32`. После этого константа может использоваться как параметр в одном из методов, созданных .NET Builder.

```
int data = 24;
MWNumericArray array = new MWNumericArray(data);
Console.WriteLine("Array is of type " + array.NumericType);
```

При выполнении этого примера получаем следующее:

```
Array is of type double
```

В этом примере, родное целое число (`int data`) преобразуется в тип `MWNumericArray`, содержащий массив 1-на-1 MATLAB `double`, который является значением типа MATLAB по умолчанию. Для того, чтобы сохранить целочисленный тип (а не преобразовать в заданный по умолчанию тип `double`), можно использовать дополнительные аргументы конструктора класса `MWNumericArray`. Отметим, что .NET Builder не поддерживает некоторые типы массивов MATLAB, потому что они не CLS-совместимы.

Если нужно создать числовой массив MATLAB определенного типа, то дополнительный параметр `makeDouble` конструктора класса `MWNumericArray` устанавливается как `False`. Тогда родной тип определяет тип создаваемого массива MATLAB в соответствии с таблицей 5.3.1. Например, в следующем коде массив создается как 16-разрядный целочисленный 1-на-1 массив MATLAB:

```
short data = 24;
MWNumericArray array = new MWnumericArray(data, False);
Console.WriteLine("Array is of type " + array.NumericType);
```

При выполнении этого примера получаем следующее:

```
Array is of type int16
```

Определение дополнительных параметров. В MATLAB используются переменные `varargin` и `varargout` для ввода и вывода неопределенного числа параметров. Рассмотрим следующую `m`-функцию:

```
function y = mysum(varargin)
y = sum([varargin{:}]);
```

Эта функция возвращает сумму вводов. Вводы предоставлены как аргумент `varargin`, что означает, что вызывающая программа может определить любое число вводов функции. Результат возвращается как скалярный массив `double`. Для функции `mysum` .NET Builder создает следующие интерфейсы:

```
// Интерфейс с единственным выводом
public MWArray mysum()
```

```
public MWArray mysum(params MWArray[] varargin)

// Стандартный интерфейс
public MWArray[] mysum(int numArgsOut)
public MWArray[] mysum(int numArgsOut, params MWArray[] varargin)

// Интерфейс feval
public void mysum(int numArgsOut, ref MWArray ArgsOut,
                 params MWArray[] varargin)
```

Аргументы `varargin` можно передать или как `MWArray[]`, или как список явных входных параметров. В C #, модификатор `params` для параметра метода определяет, что метод принимает любое число параметров определенного типа. Использование `params` позволяет коду добавлять любое число дополнительных вводов к инкапсулированной m-функции. Приведем пример использования метода `mysum` в приложении .NET для вычисления суммы $2+4$ и $2+4+6+8$ двух и четырех чисел:

```
[STAThread]
static void Main(string[] args)
{
    MWArray sum= null;
    MySumClass mySumClass = null;
    try
    {
        mySumClass = new MySumClass();
        sum = mySumClass.mysum((double)2, 4);
        Console.WriteLine("Sum= {0}", sum);
        sum = mySumClass.mysum((double)2, 4, 6, 8);
        Console.WriteLine("Sum= {0}", sum);
    }
}
```

Число входных параметров может меняться. Отметим, что для этой специфической сигнатуры нужно явно привести первый параметр к `MWArray` или другому типу, кроме целого числа. Иначе первый параметр может быть воспринят как число (целое) параметров вывода.

Примеры передачи входных параметров

Следующие примеры демонстрируют созданный метод для следующей m-функции `myprimes`:

```
function p = myprimes(n)
```

Следующий пример кода создает `data` как `MWNumericArray`, для передачи в качестве входного параметра методу `myprimes`:

```
MWNumericArray data = 5;
MyPrimesClass myClass = new MyPrimesClass();
MWArray primes = myClass.myprimes(data);
```

Передача родного типа .NET. Этот пример передает родной тип `double` к функции.

```
MyPrimesClass myClass = new MyPrimesClass();
MWSArray primes = myClass.myPrimes((double)13);
```

Входной параметр преобразуется в 1-на-1 массив MATLAB `double`, как требуется `m`-функцией. Это – заданное по умолчанию правило преобразования для родного типа `double`.

Использование интерфейса `feval`. Этот интерфейс передает оба параметра ввода и вывода справа от вызова функции. Параметру вывода `primes`, должен предшествовать атрибут `ref`.

```
MyPrimesClass myClass = new MyPrimesClass();
MWNumericArray maxPrimes = 13;
MWSArray primes = null;
myClass.myprimes(1, ref primes, maxPrimes);
```

Передача массива вводов

Следующий пример осуществляет более общий метод, который берет массив числовых примитивов .NET и преобразовывает каждый в `MWNumericArray`, который затем передается функции `mySum`.

```
public double getsum(int[] argsIn)
{
    MWSArray sum = null;
    MWSArray[] argsInArray;
    MySumClass mySumClass = null;
    try
    {
        argsInArray = new MWSArray[argsIn.Length];
        for (int idx = 0; idx < argsIn.Length; idx++)
        {
            argsInArray[idx] = new MWNumericArray((double)argsIn[idx]);
        }
        mySumClass = new MySumClass();
        sum = mySumClass.mysum(argsInArray);
        return (double)sum;
    }
}
```

Передача переменного числа выводов. Когда представлен параметр `varargout`, то аргументы обрабатываются таким же образом, что и параметры `varargin`. Рассмотрим следующую `m`-функцию:

```
function varargout = randvectors()
for i=1:nargout
    varargout{i} = rand(1, i);
end
```

Эта функция возвращает список случайных векторов `double` таким образом, что длина i -го вектора равна i . .NET Builder производит интерфейсы .NET к этой функции следующим образом:

```
public void randvectors()
public MWArray[] randvectors(int numArgsOut)
public void randvectors(int numArgsOut, ref MWArray[] varargout)
```

Обработка глобальных переменных MATLAB

Программируя с компонентами .NET Builder, нужно иметь ввиду, что экземпляр класса MCR создается один для каждого экземпляра нового класса. Если сборка содержит n различных классов, будет создано максимум n экземпляров MCR, причем каждый соответствует одному или более экземплярам одного из классов. Поэтому, избегайте использовать глобальные переменные, которые пересекаются с разными классами или экземплярами других классов.

Обработка возвращаемых значений

Предыдущие примеры показывают рекомендации для использования, когда известен тип и размерность параметра вывода. Иногда, в программировании MATLAB, эта информация неизвестна, или может измениться. В этом случае, код, который вызывает метод, возможно, должен сделать запрос о типе и размерности параметров вывода. Есть два способа сделать такой запрос:

- использовать отражение .NET, чтобы сделать запрос о типе любого объекта;
- использовать любой из нескольких методов, предоставленных классом MWArray, чтобы сделать запрос информации об основном массиве MATLAB.

Использование отражения .NET. Отражение (reflection) позволяет запрашивать информацию о типах и управлять ею. На основе полученной информации отражение позволяет динамически создать экземпляр типа, связать тип с существующим объектом, или получить тип из существующего объекта. Тогда можно вызвать методы типа или обратиться к его полям и свойствам.

Следующий пример кода вызывает метод `myprimes`, и затем определяет тип, используя отражение. Пример предполагает, что вывод возвращен как числовой векторный массив, но точный числовой тип неизвестен.

```
public void GetPrimes(int n)
{
    MWArray primes = null;
    MyPrimesClass myPrimesClass = null;
    try
    {
        myPrimesClass = new MyPrimesClass();
        primes = myPrimesClass.myprimes((double)n);
        Array primesArray =
        ((MWNumericArray)primes).ToVector(MWArrayComponent.Real);
        if (primesArray is double[])
        {
            double[] doubleArray = (double[])primesArray;
            /* Некоторые действия с doubleArray . . . */
        }
        else if (primesArray is float[])
```

```

    {
        float[] floatArray = (float[])primesArray;
        /* Некоторые действия с floatArray . . . */
    }
    else if (primesArray is int[])
    {
        int[] intArray = (int[])primesArray;
        /* Некоторые действия с intArray . . . */
    }
    else
    {
        throw new ApplicationException("
            Bad type returned from myprimes");
    }
}

```

Пример использует метод `toVector` для возвращения массива примитивов .NET (`primesArray`), который представляет основной массив MATLAB. Отметим, что `toVector` – это метод класса `MWNumericArray`. Он возвращает копию массива компонента в столбцовом порядке. Тип элементов массива определен типом данных числового массива.

Использование запросов *MWArray*

Следующий пример использует метод `NumericType` класса `MWNumericArray`, наряду с перечислением `MWNumericType`, чтобы определить тип основного массива MATLAB.

```

public void GetPrimes(int n)
{
    MWArray primes = null;
    MyPrimesClass myPrimesClass = null;
    try
    {
        myPrimesClass = new MyPrimesClass();
        primes = myPrimesClass.myprimes((double)n);
        if ((!primes.IsNumericArray) || (2 !=
primes.NumberofDimensions))
        {
            throw new ApplicationException("Bad type returned
                by mwprimes");
        }
        MWNumericArray _primes = (MWNumericArray)primes;
        MWNumericType numericType = _primes.NumericType;
        Array primesArray = _primes.ToVector(MWArrayComponent.Real);
        switch (numericType)
        {
            case MWNumericType.Double:
            {
                double[] doubleArray = (double[])primesArray;
                /* Некоторые действия с doubleArray . . . */
            }
        }
    }
    catch { }
}

```

```

        break;
    }
    case MWNumericType.Single:
    {
        float[] floatArray= (float[])primesArray;
        /* Некоторые действия с floatArray . . . */
        break;
    }
    case MWNumericType.Int32:
    {
        int[] intArray= (int[])primesArray;
        /* Некоторые действия с intArray . . . */
        break;
    }
    default:
    {
        throw new ApplicationException («Bad type returned
            by myprimes»);
    }
}
}

```

Код в примере также проверяет размерность, вызывая `NumberOfDimensions`. Этот запрос вызывает исключение, если массив не является числовым и надлежащей размерности.

5.5.3. Обработка ошибок

Приложение, которое вызывает метод, созданный .NET Builder, может обработать ошибки следующим образом: либо захватить и обработать исключение локально, либо разрешить вызывающему методу захватить ошибки. Приведем примеры для каждого способа обработки ошибок. В примере `GetPrimes` сам метод обрабатывает исключение.

```

public double[] GetPrimes(int n)
{
    MWArray primes = null;
    MyPrimesClass myPrimesClass = null;
    try
    {
        myPrimesClass = new MyPrimesClass();
        primes= myPrimesClass.myprimes((double)n);
    }
    return (double[]) (MWNumericArray)primes).ToVector(MWArrayComponent.Real);
    catch (Exception ex)
    {
        Console.WriteLine("Exception: {0}", ex);
        return new double[0];
    }
}

```

В следующем примере метод, который вызывает `myprimes`, не захватывает исключение. Вместо этого его метод вызова обрабатывает исключение.

```
public double[] GetPrimes(int n)
{
    MWArray primes = null;
    MyPrimesClass myPrimesClass = null;
    try
    {
        myPrimesClass = new MyPrimesClass();
        primes = myPrimesClass.myprimes((double)n);
    }
    return (double[]) (MWNumericArray)primes).ToVector(MWArrayComponent.Real);
}
```

5.5.4. Управление родными ресурсами

Классы `MWArray` преобразования данных используют родные ресурсы. Каждый класс в иерархии класса `MWArray` есть управляемый интерфейсный класс, который инкапсулирует MATLAB `mxArray` и который распределен в родной куче памяти.

Реализация класса `MWArray` создает очень маленькую управляемую обертку, которая обычно инкапсулирует достаточно большой массив `mxArray`, который распределен в родной динамической памяти. Во время выполнения приложения, когда создаются экземпляры типов `MWArray`, распределение родной памяти растет значительно, в то время как распределение управляемой памяти, для интерфейсных классов остается относительно маленьким. В результате сборщик мусора CLR вызывается очень редко и родная динамическая память становится быстро исчерпанной.

Для решения этих проблем, классы `MWArray` преобразования данных следят за приблизительным размером распределения родной памяти, используемых инкапсулированными объектами `mxArray`. Когда достигается указанный порог распределения памяти, явно вызывается сборщик мусора CLR, чтобы освободить неиспользуемые экземпляры класса `MWArray`. Когда CLR вызывает `finalizer` для этих экземпляров класса, это освобождает родную память, распределенную для инкапсулированного `mxArray`. В результате код не должен вызывать деструктор явно.

Отметим, что заданная по умолчанию схема управления памятью всех компонентов .NET Builder допускает автоматическое управление памятью с пороговым размером блока 10 Мбайт. Можно изменить это значение по умолчанию.

Для явного разрешения или отключения возможности управления памятью `MWArray` и определения порога распределения родной памяти используется атрибут `NativeGC` для сборки.

Следующий фрагмент C# сегмент явно допускает управлению памятью и устанавливает порог распределения памяти 100Mb.

```
[assembly: NativeGC(true, GCBlockSize=100)]
```

Напомним, что по умолчанию управление родной памятью допускается.

Следующие два фрагмента кода из примера магического квадрата показывают, как приложение, использующее компоненты .NET Builder, решает проблемы ресурсов памяти, с допущенным и, соответственно, заблокированным управлением памятью.

Управление ресурсами с заблокированным управлением памятью
[assembly: NativeGC(false)]

```
. . .
int arraySize= System.Int32.Parse(args[0]);
    MagicSquare magic= new MagicSquare();

// Возвращение магического квадрата указанного размера
magicSquare= magic.makesquare( (MArray)arraySize);
}
    finally
{
// Явное освобождение памяти для магического квадрата
if (null != (Object)magicSquare) magicSquare.Dispose();
}
```

Управление ресурсами с допущенным управлением памятью.

[assembly: NativeGC(true)] //Это по умолчанию

```
. . .
MagicSquare magic = new MagicSquare();
// Возвращение магического квадрата указанного размера
magicSquare= magic.makesquare( (MArray)arraySize);
```

Рекомендуется использовать автоматизированную сборку мусора, предоставленную классами MArray.

Если Вы не хотите использовать автоматическую сборку мусора, предоставленную .NET Builder, можно использовать любую из следующих альтернатив:

- использовать сборку мусора, предоставленную CLR;
- освободить родные ресурсы завершением;
- использовать Dispose для явного освобождения ресурсов.

Объекты MArray используют пространство для родных ресурсов. Хотя эти ресурсы могут быть весьма большими, они не видимы для CLR и не будут освобождены классом finalizer, пока CLR не решит, что уместно вызвать сборщик мусора. Чтобы избежать исчерпания неуправляемой динамической памяти кучи, объекты MArray должны быть явно освобождены как можно скорее приложением, которое их создает (если автоматическая сборка мусора не была включена).

Использование Dispose для явного освобождения ресурсов. Следующий пример распределяет родной массив на 8 Мбайт. Для CLR размер обернутого объекта – только несколько байтов (размер экземпляра интерфейсного класса MWNumericArray) и таким образом – незначительного размера для вызова сборщика мусора. Рекомендуется явное освобождение MArray, в том случае, когда

не используется автоматическая сборка мусора, предоставленная классами преобразования данных.

Обычно метод `Dispose` вызывают из раздела `finally` в блоке `try-finally`, что демонстрирует следующий пример:

```
try
{
    /* Распределение большого массива */
    MWNumericArray array = new MWNumericArray(1000,1000);
    . . . // использование массива
}
finally
{
    /* Явное освобождение управляемого массива и его родных ресурсов */
    if (null != array)
    {
        array.Dispose();
    }
}
```

Оператор `array.Dispose()` освобождает память, занятую и управляемой оберткой и родным массивом MATLAB. Класс `MWArray` обеспечивает два метода освобождения: `Dispose` и `DisposeArray`. Метод `DisposeArray` является более общим в том, что он избавляется и от единственного `MWArray`, и от массива массивов типа `MWArray`.

5.5.5. Преобразования между типами C# и `MWNumericArray`

В этом разделе приведем примеры кодов для явного преобразования различных типов массивов C# в массивы `MWNumericArray` и обратно, см. также раздел 5.3.2 и [LePh2]. При таких преобразованиях необходимо использовать следующие пространства имен:

```
using System;
using System.Collections.Generic;
using MathWorks.MATLAB.NET.Arrays;
```

Преобразование скаляров

Рассмотрим преобразования вещественных и комплексных скаляров.

1. Преобразование вещественного скаляра `db_a` типа `double` в скаляр `mw_a` типа `MWNumericArray`

```
double db_a = 1.1 ;
MWNumericArray mw_a = new MWNumericArray(db_a) ;
```
2. Преобразование вещественного скаляра `mw_a` типа `MWNumericArray` в обычный скаляр `db_a2` типа `double`

```
double db_a2 = mw_a.ToScalarDouble() ;
```

3. Преобразование комплексного числа `double` в комплексный `MWNumericArray`.

```
double dbReal_a = 1.1 ;
double dbImag_a = 2.2 ;
MWNumericArray mwComplex_a = new MWNumericArray(dbReal_a,
dbImag_a) ;
```

4. Преобразование комплексного числа `MWNumericArray` в обычные скаляры `double`. Если число не содержит мнимую часть, то возможна ошибка, поэтому здесь нужно использовать блок `try/catch`.

```
double[] dbReal_a2 = new double[1] ;
double[] dbImag_a2 = new double[1] ;
dbReal_a2 = (double[])
mwComplex_a.ToVector(MWArrayComponent.Real) ;
    try
    {
dbImag_a2 = (double[])
mwComplex_a.ToVector(MWArrayComponent.Imaginary) ;
    }
    catch
    {
        // ничего, dbImag_a2[] присвоено нулевое значение
    }
mw_a.Dispose() ;
mwComplex_a.Dispose() ;
```

Преобразование векторов

Рассмотрим преобразования вещественных и комплексных векторов.

1. Преобразование вещественного вектора `db_a` типа `double` в вектор `mw_a` типа `MWNumericArray`

```
double[] db_a = { 1.1, 2.2, 3.3 } ;
MWNumericArray mw_a = new MWNumericArray(db_a) ;
```

2. Преобразование вещественного вектора `mw_a` типа `MWNumericArray` в вещественный вектор `db_a` типа `double`.

```
double[] db_a2 =
    (double[]) mw_a.ToVector(MWArrayComponent.Real) ;
```

3. Преобразование комплексного вектора `double` в комплексный вектор `MWNumericArray`

```
double[] dbReal_a = { 1.1, 2.2, 3.3 } ;
double[] dbImag_a = { 11.1, 22.2, 33.3 } ;
MWNumericArray mwComplex_a = new MWNumericArray(dbReal_a,
dbImag_a) ;
```

4. Преобразование комплексного вектора `mwComplex_a` типа `MWNumericArray` в комплексный вектор типа `double`. Если вектор не содержит мнимых частей, то возможна ошибка, поэтому здесь нужно использовать блок `try/catch`.

```

int vectorSize =
mwComplex_a.ToArray(MWArrayComponent.Real).GetUpperBound(0) -
mwComplex_a.ToArray(MWArrayComponent.Real).GetLowerBound(0) + 1 ;

double[] dbReal_a2 = new double[vectorSize] ;
double[] dbImag_a2 = new double[vectorSize] ;

dbReal_a2 = (double[]) mwComplex_a.ToVector(MWArrayComponent.Real) ;

try
{
dbImag_a2 = (double[])
mwComplex_a.ToVector(MWArrayComponent.Imaginary) ;
}
catch
{
// Ничего, dbImag_a2[] присвоены нулевые значения
}
mw_a.Dispose();
mwComplex_a.Dispose();

```

Преобразование матриц

Рассмотрим преобразования вещественных и комплексных матриц.

1. Преобразование вещественной матрицы db_A типа double в матрицу mw_A типа MWNumericArray.

```

double[,] db_A = {{1.1, 2.2, 3.3}, {4.4, 5.5, 6.6}, {7.7, 8.8,
9.9}} ;
MWNumericArray mw_A = new MWNumericArray(db_A) ;

```

2. Преобразование вещественной матрицы mw_A типа MWNumericArray в матрицу db_A типа double.

```

double[,] db_A2 =
(double[,]) mw_A.ToArray(MWArrayComponent.Real) ;

```

3. Преобразование комплексной матрицы типа double в комплексную матрицу MWNumericArray.

```

double[,] dbReal_A = {{1.1, 2.2, 3.3}, {4.4, 5.5, 6.6}, {7.7,
8.8, 9.9}};
double[,] dbImag_A = {{ 11, 12, 13}, {14, 15, 16}, {17, 18,
19}} ;
MWNumericArray mwComplex_A = new MWNumericArray(dbReal_A,
dbImag_A) ;

```

4. Преобразование комплексной матрицы типа MWNumericArray в комплексную матрицу типа double. Если матрица не содержит мнимых частей, то возможна ошибка, поэтому здесь нужно использовать блок try/catch.

```

int row = mwComplex_A.ToArray(MWArrayComponent.Real).GetUpperBound(0) -
mwComplex_A.ToArray(MWArrayComponent.Real).GetLowerBound(0) + 1;

int col = mwComplex_A.ToArray(MWArrayComponent.Real).GetUpperBound(1) -

```

```

mwComplex_A.ToArray(MWArrayComponent.Real).GetLowerBound(1) + 1;

double[,] dbReal_A2 = new double[row,col] ;
double[,] dbImag_A2 = new double[row,col] ;

dbReal_A2 = (double[,]) mwComplex_A.ToArray(MWArrayComponent.Real) ;
try
{
dbImag_A2 = (double[,]) mwComplex_A.ToArray(MWArrayComponent.Imaginary) ;
}
catch
{
// Ничего, dbImag_A2 присвоены нулевые значения
}
mw_A.Dispose();
mwComplex_A.Dispose();

```

5.6. Среда разработки Visual Studio 2005

В этом параграфе рассмотрим очень кратко среду проектирования Visual Studio 2005, которая позволяет создавать Windows-приложения на языках программирования Visual Basic, Visual C#, Visual J# и Visual C++. Она построена на тех же принципах, что и аналогичные среды разработки других языков. Поэтому переход на Visual Studio 2005 не представляет трудностей. Начинаящим программистам полезно обратиться к какому-нибудь руководству, например [Фа], [Тро]. В следующем параграфе будут представлены примеры создания Windows-приложений на Visual Studio 2005, в работе которых используются математические процедуры MATLAB.

При запуске Visual Studio 2005 открывается окно среды разработки в котором можно выполнить большинство функций разработки: редактирование кода, визуальное проектирование, навигацию, просмотр, компиляцию, отладку, и другие операции. Это окно – рабочее пространство Visual Studio 2005 и оно состоит из нескольких областей, предназначенных для выполнения функций разработки приложения (рис. 5.6.1).

Возможно, что при начальной загрузке среды Visual Studio 2005 не будет необходимых окон главного окна приложения, например, окна Toolbox визуальных компонент и окна Properties, в котором устанавливаются значения свойств выбранных ко мponentов. Чтобы они отображались, необходимо их открыть в меню **View** ⇒ **Toolbox** и **View** ⇒ **Properties Window**. Меню **View** дает существенные возможности для настройки рабочего пространства Visual Studio 2005. Полезно попробовать открыть все пункты меню **View**. Главное окно, с открытыми окнами **Toolbox**, **Properties**, **Class View** и **Solution Explorer** показано на рис. 5.6.1.

Дадим краткое описание описание этих областей среды Visual Studio 2005, считая, что создается приложение на Visual C#.

Рабочая область. Это центральная и самая большая часть окна Visual Studio 2005. Содержит ряд закладок. Закладка Form1.cs[Design] используется для проектирования формы приложения (рис. 5.6.1). Закладка Form1.Designer.cs содер-

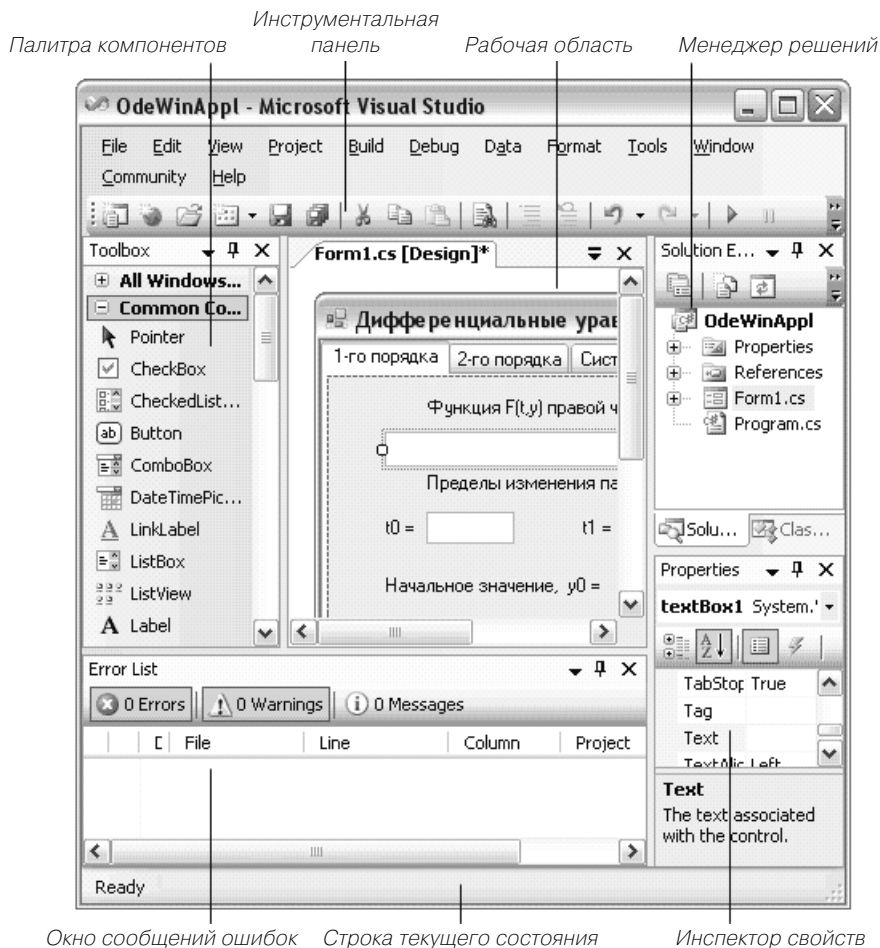


Рис. 5.6.1. Visual Studio 2005

жит файл класса формы Form1. Закладка Form1.cs содержит файл, в котором разработчик описывает действия элементов формы – это основной рабочий файл приложения. Обратите внимание на то, что в верхней части закладок этих файлов открываются списки, которые содержат ссылки на все элементы создаваемого приложения (рис. 5.6.2). Закладка Program.cs содержит файл класса Program, содержащий метод Main. В рабочей области, в отдельной закладке, может быть открыт любой файл (меню **File**). Все открытые исходные файлы отображаются сразу в редакторе кода Visual Studio 2005. Редактор кода представляет файл в удобном наглядном виде, используя выделение цветом ключевых слова и синтаксических конструкций.

Палитра компонентов Toolbox. Компоненты Visual C# – это классы C#, которые размещены в палитре компонентов и могут быть перенесены в разрабатываемое

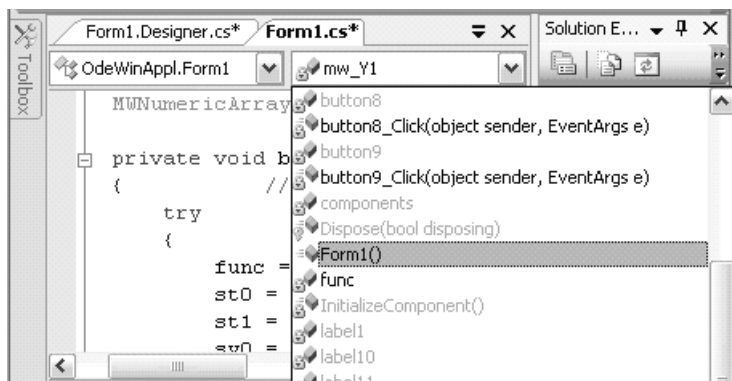


Рис. 5.6.2. Файл Form1.cs и список элементов программы

мую форму. Компоненты Visual C# разделяются на визуальные и не визуальные. *Визуальные компоненты* во время выполнения разработанного приложения отображаются на форме точно так же, как и во время проектирования. Эта категория компонент, исходя из названия, необходима для представления данных на экране компьютера. Примерами таких компонент являются текстовые поля, метки, кнопки, списки и т.д. *Невизуальные компоненты* во время исполнения программы не отображаются, а в процессе проектирования отображаются на форме в виде соответствующих иконок. Такие компоненты обладают определенной функциональностью (например, вызывают стандартные диалоги открытия или закрытия файлов, обрабатывают системные события, обеспечивают доступ к базам данным и т.д.)

Компоненты разбиты на несколько смысловых групп. Выбранный компонент помещается на форму обычным образом, при помощи левой кнопки мыши. Вверху окна палитры компонентов имеется небольшая кнопка «Auto Hide». При помощи нее можно менять представление окна – оно может быть открытым постоянно, либо сворачиваться в вертикальную метку **Toolbox**, которая открывается в обычное полное окно при наведении стрелки мыши. Это удобно, когда все визуальные компоненты на форме уже определены. Отметим, что такое удобство имеется и для всех других окон среды разработки Visual Studio 2005.

Менеджер решений Solution Explorer. В среде разработки Visual Studio 2005 проекты логически организованы в решения (Solution). Каждое решение состоит из одного или нескольких проектов. В свою очередь, каждый проект состоит из множества исходных файлов, ссылок на внешние сборки и прочих ресурсов. Любой из этих ресурсов можно открыть при помощи менеджера решений. Данное окно дает информацию о структуре проекта, его свойствах, ссылках и файлах проекта. Позволяет быстро находить и открывать необходимые файлы проекта и управлять компонентами, включенными в проект. Например, для того чтобы добавить в проект новую форму, достаточно выбрать в контекстном меню, открываемом по щелчку правой кнопки мыши, пункт **Add/Add Windows Form**. При выборе одного из пунктов окна **Solution Explorer**, в окне **Properties** отображаются

свойства выбранного элемента. Поэкспериментируйте с Solution Explorer. Это позволит вам более уверенно чувствовать себя в среде Visual Studio .NET.

Окно Class View. Отображает объектно-ориентированную иерархию приложения. Это окно позволяет перемещаться по всем элементам программного проекта, включая отдельные процедуры. При этом с помощью Class View можно добавлять новые методы, классы, данные. Все эти возможности будут доступны вам благодаря контекстному меню Class View. При этом, каждый элемент дерева проекта имеет свой тип контекстного меню. Также рекомендуется попробовать все возможности Class View на каком-нибудь пробном проекте. Если окно Class View отсутствует на экране, нужно выбрать пункт меню **View/Class View**.

Инспектор свойств Properties Explorer. Содержание окна свойств Properties Explorer всегда соответствует выбранному (активному) элементу проекта. Например, на рис. 5.6.1 активным является текстовое поле textBox1 и именно его свойства отображаются в окне. Внизу окна автоматически появляется поясняющий текст для выбранного свойства (рис. 5.6.1). Данное окно позволяет устанавливать свойства форм и их компонентов. Properties Explorer содержит список всех свойств выбранного в текущий момент компонента и их значений (слева). При этом значения свойств могут быть представлены в любой форме, например, как текстовое поле, как выпадающий список допустимых значений, как окно выбора цвета и т. д. Если вы измените значение свойства по умолчанию, то оно будет выделено жирным цветом. В этом случае контроль за вносимыми в проект изменениями становится более наглядным. Кроме того, Properties Explorer позволяет сортировать свойства либо по алфавиту, либо по принадлежности к определенной группе.

Второй важной задачей, которую выполняет Properties Explorer, является управление событиями. Для того чтобы переключиться на закладку событий, нажмите кнопку с изображением молнии вверху окна (рис. 5.6.1). Окно событий позволит вам настраивать реакцию вашей формы или компонента на различные действия со стороны пользователя или операционной системы, например, создать обработчик событий от мыши или клавиатуры. В левой части окна содержится список всех доступных событий, а в правой – имен методов, обрабатывающих события. По умолчанию список методов пуст. Вы можете добавить новый обработчик, вписав имя метода в соответствующую ячейку, либо создать обработчик с именем по умолчанию, щелкнув два раза по ячейке левой кнопкой мыши. Многие свойства компонента, отображаемые в колонке инспектора свойств, имеют значения, устанавливаемые по умолчанию.

Для того чтобы добавить *обработчик событий*, необходимо выбрать на форме компонент, которому необходим обработчик событий, затем открыть страницу событий инспектора свойств и дважды щелкнуть левой клавишей мыши на колонке значений рядом с событием, чтобы заставить Visual C# сгенерировать прототип обработчика событий и показать его в редакторе кода (Form1.cs). При этом автоматически генерируется текст пустой функции, и редактор открывается в том месте, где следует вводить код. Курсор позиционируется внутри операторных скобок { }. Далее нужно ввести код, который должен выполняться при наступле-

нии события. Обработчик событий может иметь параметры, которые указываются после имени функции в круглых скобках.

Окно сообщений об ошибках Error List. При отладке программы в нем отражаются ошибки программы. При двойном щелчке на строку вывода, мы сразу попадаем в строку программы, где находится данная ошибка.

5.6.1. Создание нового проекта

Создание нового проекта начинается с выбора пункта **New Project** в меню **File**. В результате открывается диалоговое окно (рис. 5.6.3) в котором можно выбрать тип проекта.

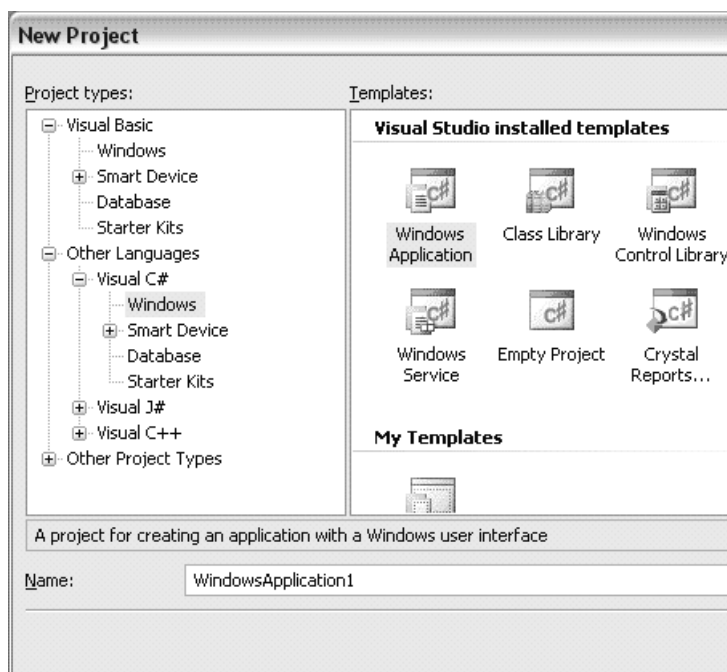


Рис. 5.6.3. Диалоговое окно выбора шаблона проекта

Выбираем тип проекта Visual C# Windows. Тогда в правой части окна появятся несколько шаблонов проектов, на основе которых можно начать построение своего проекта. Выбираем шаблон Windows-приложения. Внизу окна предлагается определить имя проекта. Выбираем имя по умолчанию WindowsApplication1. Это все шаги по созданию нового проекта. Нажимаем **ОК** и появляется основное рабочее окно, в котором расположена пустая форма. Кроме того, в окне **Solution Explorer** отображается структура проекта (пока виртуального). Можно открыть два файла: Form1.Designer.cs и Program.cs. Файл Form1.cs становится доступным

только после того, как по форме Form1 щелкнуть два раза мышкой. Теперь все готово для создания проекта. Из палитры компонентов выбираем нужные, определяем их свойства и события, пишем необходимые процедуры в файле Form1.cs.

Отладка проекта может быть произведена как кнопкой интрузивного меню, так и из меню **Debug**. Построение проекта производится из меню **Build**.

Сохранение проекта производится из меню **File** выбором пункта **Save All**. В результате открывается диалоговое окно (рис. 5.5.4) в котором можно выбрать имя проекта, его размещение и имя решения, для которого создается новая директория. Сохраним проект в каталог D:\Temp, выбрав имя решения как Win-Appl (рис. 5.6.4).

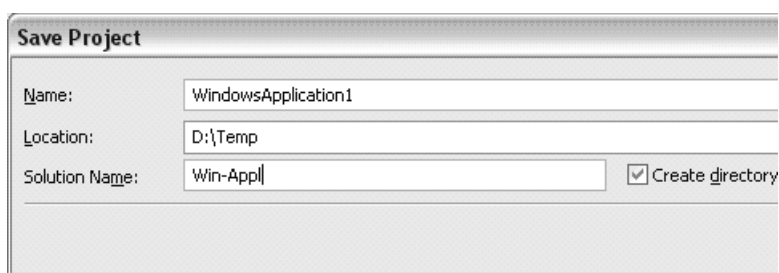


Рис. 5.6.4. Диалоговое окно сохранения проекта

В результате в каталоге D:\Temp\Win-Appl сохраняется весь набор файлов проекта. Созданное в результате отладки приложение (exe-файл) сохраняется в каталоге D:\Temp\Win-Appl\WindowsApplication1\bin\Debug\, а приложение, которое создано в результате построения из меню **Build**, сохраняется в каталоге D:\Temp\Win-Appl\WindowsApplication1\bin\Release\.

5.7. Программирование на Visual Studio 2005 с использованием математических процедур MATLAB

В этом разделе рассмотрим несколько примеров создания Windows-приложений в среде разработки Visual Studio 2005 на языке C# и с использованием математических функций MATLAB. Сначала создается компонент .NET Builder, который инкапсулирует m-функции MATLAB, а затем разрабатывается Windows-приложение, использующее в своей работе созданные компоненты. Для работы такого приложения необходима среда исполнения CLR (.NET Framework 2.0) и среда исполнения MCR MATLAB (которая может быть установлена на системе независимо от MATLAB). Примеры являются учебными, поэтому они имеют только самые необходимые элементы. Читатель может самостоятельно добавить, например, ин-

дикаторы выполнения ProgressBar, или обработчики ошибок. Исходные тексты примеров находятся на прилагаемом компакт-диске в каталоге Gl_5_C#_Examples.

5.7.1. Вычисление интегралов

Создадим приложение для вычисления однократного и двойного интегралов.

Создание .NET компонента

Сначала подготовим необходимые m-функции. Для вычисления однократного интеграла используем встроенную функцию MATLAB `quad(@fun,a,b)`, которая реализует адаптивный метод Симпсона вычисления интеграла функции $f(x)$ на промежутке $[a,b]$. Квадратура `quad(@fun,a,b)` в качестве первого аргумента принимает имя функции, которая подлежит интегрированию. Создадим эту функцию при помощи процедуры MATLAB `inline`, которая принимает строку задания функции и строит соответствующую функцию. Отметим, что строка `strfunc`, определяющая функцию должна быть записана по правилам синтаксиса MATLAB, а именно, нужно учитывать необходимость ставить точку перед арифметическими операциями MATLAB. Например, `strfunc = sin(x.^2 + eps)./(x.^2 + eps)`. Листинг функции `int_1`:

```
function y = int_1(strfunc, a, b)
F = inline(strfunc) ;
y = quad(F, a, b) ;
```

Для вычисления двойного интеграла функции $f(x,y)$ используем встроенную функцию MATLAB `dblquad(@fun,a,b,c,d)` и функцию `inline`. Отметим также, что строка `strfunc`, определяющая функцию должна быть записана по правилам синтаксиса MATLAB. Листинг функции `int_2`:

```
function int2func = int_2(strfunc, x1, x2, y1, y2)
F = inline(strfunc) ;
int2func = dblquad(F, x1, x2, y1, y2) ;
```

Полезно также посмотреть графики тех функций, которые мы собираемся интегрировать. Поэтому включим в пакет m-функций еще две функции, которые строят графики.

График функции одного переменного, строим по строке `strfunc` функции и пределам изменения $[a, b]$ аргумента при помощи встроенной функции MATLAB `fplot`:

```
function y = myfplot(strfunc, a, b)
F = inline(strfunc) ;
fplot(F, [a, b])
```

График функции двух переменных строим по строке `strfunc` функции и пределам изменения $[a, b]$ и $[c, d]$ аргументов и помощи встроенной функции MATLAB `ezsurf`:

```
function y = myezsurf(strfunc, a, b, c, d)
F = inline(strfunc) ;
ezsurf(F, [a, b], [c, d])
```

Для создания .NET компонента будем использовать среду разработки Deployment Tool, ее описание достаточно подробно изложено в предыдущих главах. Делаем текущим рабочим каталогом MATLAB каталог проекта D:\Work\Integration и вызываем среду разработки командой MATLAB:

```
deploytool
```

Выбираем создание .NET компонента по имени **Integr**, класс назовем **Integrclass**, он включает все указанные выше функции. Определим каталог проекта D:\Work\Integration. Компонент представляет собой частную сборку **Integr.dll** и технологический файл **Integr.ctf**, которые находятся в каталоге D:\Work\Integration\distrib.

Создание приложения

В среде разработки Visual Studio 2005 на языке C# создадим проект по имени **Integration** (решение Integr_Appl) и поместим его в каталог D:\Work\Integration\. Создание приложения для вычисления интегралов и построения графиков проведем в несколько этапов.

1. Проектирование формы приложения. Форму приложения **Form1** назовем «Интеграл» – это имя нужно внести в свойство **Text** инспектора свойств формы. На форму **Form1** приложения поместим компонент вкладок **TabControl** из контейнера компонентов **Common Control**. Первая вкладка – для однократного интегрирования, поэтому назовем ее «Интеграл» (свойство **Text** инспектора свойств), вторая – для двойного интеграла.

На первую вкладку поместим несколько меток Label, текстовых полей TextBox и две кнопки (рис. 5.7.1).

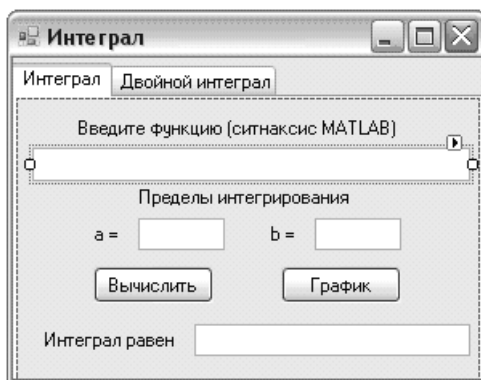


Рис. 5.7.1. Конструирование формы приложения

В первое текстовое поле вводится строка, определяющая функцию в соответствии с синтаксисом MATLAB. В два следующих текстовые окна вводятся пределы интегрирования. Последнее текстовое поле – для вывода результата. Две кнопки button1 и button4 – для вычисления интеграла и отображения графика.

Аналогично оформляется и вторая вкладка для вычисления двойного интеграла. После проектирования формы, файл Form1.cs имеет следующий вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Integration
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

2. Подключение внешних классов. Предполагается, что приложение будет использовать методы класса Integrclass, созданного .NET Builder компонента. Для того, чтобы это было возможным, мы должны подключить внешние классы компонента Integr и классы среды исполнения MATLAB MCR. Для этого нужно добавить в проекте ссылку на компонент MWArray.dll, содержащий классы MWArray. Он находится в каталоге C:\R2007a\toolbox\dotnetbuilder\bin\win32\v2.0\, или в соответствующем каталоге MCR. Данный компонент зарегистрирован в системе, поэтому достаточно сделать следующее: в Visual Studio 2005 из меню **Project** \Rightarrow **Add Reference** открыть диалоговое окно **Add Reference** и в нем найти и отметить строку с именем компонента (рис. 5.7.2).

Также необходимо добавить в проекте ссылку на компонент Integr.dll, который находится в подкаталоге distrib каталога D:\Work\Integration\Integr\.. Для этого в диалоговом окне **Add Reference** нужно открыть вкладку **Browse** (рис. 5.7.2) и найти и отметить файл Integr.dll в каталоге D:\Work\Integration\Integr\distrib\.

Если программа разрабатывается без MATLAB, то нужно указывать компонент MWArray.dll среды исполнения MCR MATLAB в том каталоге, где установлен MCR:

```
.\MATLAB Component Runtime\v76\toolbox\dotnetbuilder\bin\win32\v2.0\
```

Отметим, что при установке MCR данный компонент также регистрируется в системе, поэтому его подключение вполне аналогично.

После указания ссылок добавим пространства имен указанных компонентов в код программы:

```
using MathWorks.MATLAB.NET.Utility;
using MathWorks.MATLAB.NET.Arrays;
using Integr;
```

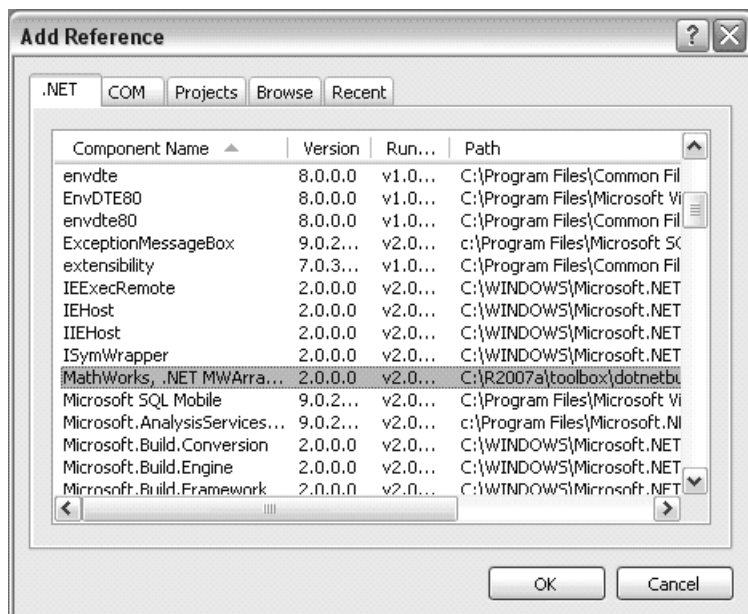


Рис. 5.7.2. Добавление ссылок на компоненты в Visual Studio 2005

3. Описание событий. Предполагается, что по первой кнопке «Вычислить» происходит вычисление интеграла и отображение результата в нижнем текстовом поле. По кнопке «График» – построение графика подынтегральной функции.

По нажатию кнопки «Вычислить» программа должна выполнить следующие действия:

- считать строку функции (func) из текстового поля textBox1 и пределы интегрирования (at, bt) из текстовых полей textBox2 и textBox3;
- преобразовать полученные текстовые данные (типа string) в те типы, которые может принять метод интегрирования: MWCharArray mw_strfunc и double a, b;
- создать экземпляр (obj) класса Integrclass и вызвать функцию int_1 для вычисления интеграла;
- из массива MWArray[] выбрать элемент (mw_y), представляющий значение интеграла типа MWNumericArray;
- преобразовать полученное значение типа MWNumericArray сначала в числовой тип (double y), а затем – в строку (string I), с тем, чтобы ее можно было отобразить в окне ответа textBox3.

В соответствии с этой программой и будем описывать событие button1_Click. Двойной клик по кнопке «Вычислить» на форме создает в коде Form1.cs пустую заготовку для описания события:

```
ivate void button1_Click(object sender, EventArgs e)
{

}
```

в которую и пишется программа. Приведем соответствующий фрагмент листинга программы Form1.cs для вычисления определенного интеграла. Отметим, что строка функции и пределы интегрирования объявлены вне блока события button1_Click. Это сделано для того, чтобы эти переменные были бы видны из блока построения графика.

Напомним, что стандартный вывод функции компонента .NET Builder является массивом MWArray[]. Каждый элемент сам является массивом некоторого размера и типа. Для того, чтобы выбрать массив, содержащийся, например, в первом выводе метода, нужно сначала выбрать этот элемент, а затем – преобразовать его в соответствующий массив. В нашем случае это делается следующим образом:

```
MWArray[] mw_ArrayOut = null; // Выходной массив метода int_1
MWNumericArray mw_y = null; // Массив первого параметра вывода
Integrclass obj = new Integrclass(); // Экземпляр класса компонента
mw_ArrayOut = obj.int_1(1, mw_strfunc, a, b); // Обращение к методу int_1
// Выбор первого элемента из массива MWArray[]
// и его преобразование в числовой тип MWNumericArray
mw_y = (MWNumericArray)mw_ArrayOut[0];
```

Фрагмент листинга:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    string func, at, bt, ct, dt, I; // Объявления переменных
    double a, b, c, d;
// ----- Вычисление однократного интеграла -----
    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            func = textBox1.Text; // Считывание с текстового поля
            at = textBox2.Text;
            bt = textBox3.Text;
            a = Convert.ToDouble(at); // Преобразование в число
            b = Convert.ToDouble(bt);
            MWArray[] mw_ArrayOut = null; // Выходной массив метода int_1
            MWNumericArray mw_y = null;
// Преобразование строки функции в тип MWCharArray
            MWCharArray mw_strfunc = new MWCharArray(func);
// Экземпляр obj класса Integrclass компонента
            Integrclass obj = new Integrclass();
```

```
// Обращение к методу int_1 для вычисления интеграла
    mw_ArrayOut = obj.int_1(1, mw_strfunc, a, b);
// Выбор первого элемента из массива MWArray[]
    mw_y = (MWNumericArray)mw_ArrayOut[0];
    double y = (double)mw_y; // Преобразование в число C#
    I = Convert.ToString(y); // Преобразование в строку
    textBox4.Text = @I;      // Вывод результата
}
catch { }
```

Для обработки события `button3_Click` второй кнопки создается аналогичная программа. Эта кнопка может работать независимо от первой для того, чтобы перед интегрированием можно было бы посмотреть вид функции. Поэтому в программу снова включен блок считывания и преобразования данных текстовых строк. Листинг программы:

```
//----- График функции одной переменной -----
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        func = textBox1.Text; // Считывание с текстового поля
        at = textBox2.Text;
        bt = textBox3.Text;
        a = Convert.ToDouble(at); // Преобразование в число
        b = Convert.ToDouble(bt);
        // Преобразование строки функции в тип MWCharArray
        MWCharArray mw_strfun = new MWCharArray(func);
        // Экземпляр plot1 класса Integrclass компонента
        Integrclass plot1 = new Integrclass();
        // Обращение к методу myfplot для построения графика
        plot1.myfplot(0, mw_strfun, a, b);
    }
    catch
    {
    }
}
```

В случае двойного интеграла все совершенно аналогично. Событие для `button2_Click` заключается в считывании строки функции двух переменных, считывании пределов интегрирования, создание экземпляра класса `Integrclass` и вызов функции `Int_2` для вычисления двойного интеграла. Событие для `button4_Click` заключается в считывании строки функции двух переменных, считывании пределов интегрирования, создание экземпляра класса `Integrclass` и вызов функции `myezsurf` для построения графика.

4. Построение и выполнение приложения. Теперь построим и выполним приложение. В Visual Studio 2005 для этой цели служит кнопка **Start Debugging** инструментальной панели Visual Studio, а также меню **Build**. При использовании **Start**

Debuging создается сборка **Integration.exe**, которая сохраняется в подкаталоге `bin\Debug\` каталога `D:\Work\Integration\Integr_Appl\Integration\`, а при использовании меню **Build** сборка `Integration.exe` сохраняется в подкаталоге `.\bin\Release\`.

При построении компонента `Integr`, .NET Builder было указано, что создается частная сборка `Integr.dll`. Поэтому при построении приложения, файлы компонента `Integr.dll` и `Integr.ctf` автоматически помещаются в каталог, где находится приложение `Integration.exe`. Следовательно, для работы приложения не нужно указывать пути к `Integr.dll` и `Integr.ctf`.

Работа приложения `Integration.exe` для вычисления однократного интеграла показана на рис. 5.7.3 и 5.7.4. Найдем интеграл функции $y = \sin(\pi/x)$ в пределах от 0 до 0,6 и построим график этой функции. Строка вводится в синтаксисе MATLAB (никаких кавычек не нужно, а нужно ставить точку перед математическими операциями). Имеется одна особенность, в однократном интеграле переменная x долж-

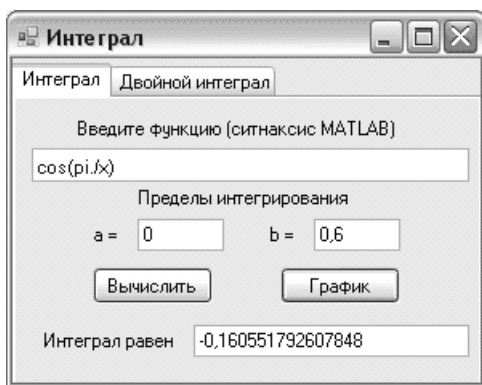


Рис. 5.7.3. Вычисление интеграла

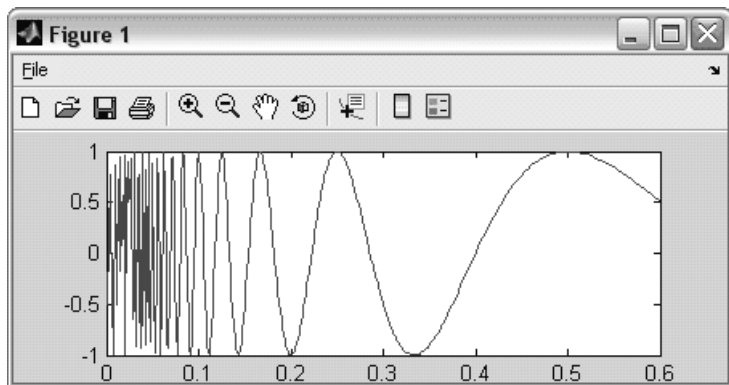


Рис. 5.7.4. График подынтегральной функции

на быть в выражении функции обязательно, хотя бы формально, в виде $0 \cdot x$, а в двойном интеграле должны присутствовать обе переменные интегрирования x и y , хотя бы формально, в виде $0 \cdot x$, или $0 \cdot y$. Переменные интегрирования можно обозначать любыми буквами. Обратите внимание, что число π можно ввести как символ π – так же, как и в MATLAB.

График функции выводится в стандартном графическом окне MATLAB (рис. 5.7.4 и 5.7.6). Заметим, что все кнопки инструментальной панели этого окна, включая вращение и лупу – действуют.

Работа приложения Integration.exe для вычисления двойного интеграла показана на рис. 5.7.5 и 5.7.6. Вычислим двойной интеграл функции

$$f(x, y) = \frac{\sin(\rho + \text{eps})}{\rho + \text{eps}}, \quad \rho = \sqrt{x^2 + y^2}$$

по прямоугольнику: $x \in [-8, 2]$, $y \in [-8, 2]$ и построим график. Обратите внимание, что можно использовать встроенную константу eps MATLAB.

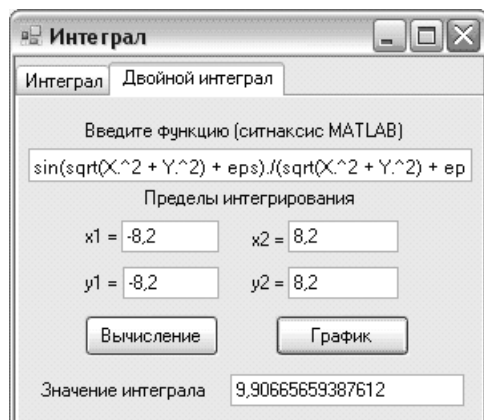


Рис. 5.7.5. Вычисление двойного интеграла

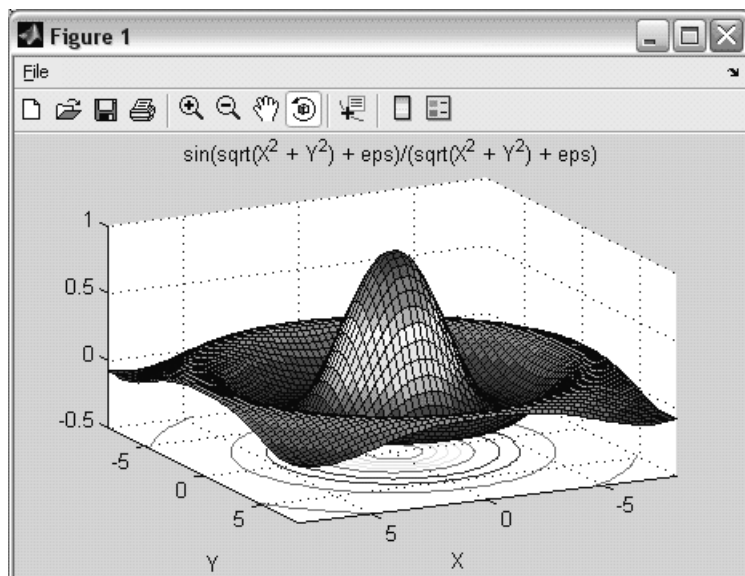


Рис. 5.7.6. График подынтегральной функции

5.7.2. Решение обыкновенных дифференциальных уравнений

Создадим программу для решения задачи Коши обыкновенного дифференциального уравнения первого порядка вида $y' = F(t, y)$, дифференциального уравнения второго порядка вида $y'' = F(t, y, y')$ и системы из трех дифференциальных уравнений первого порядка.

В системе MATLAB имеется достаточно много решателей дифференциальных уравнений [ККШ]. В нашем примере будем использовать решатель ode45, использующий явный метод Рунге-Кутты 4-го и 5-го порядков. Для нежестких систем эта функция дает хорошие результаты.

Создание .NETкомпонента ODE

Сначала создадим пакет необходимых m-функций для их использования в приложении. Включим в этот пакет следующие m-функции:

- ode45_1.m – решение одного дифференциального уравнения 1-го порядка;
- ode45_2.m – решение одного дифференциального уравнения 2-го порядка;
- ode45_3.m – решение системы трех дифференциальных уравнений 1-го порядка;
- odefunc.m – функция правой части дифференциального уравнения 1-го порядка;
- odefunc2.m – функция правой части дифференциального уравнения 2-го порядка;
- odefunc3.m – вектор-функция правой части системы дифференциальных уравнений 1-го порядка;
- myplot.m – функция для создания графика решения $y=y(t)$;
- myplot2.m – функция для создания графиков двух функций $y(t)$ и $y'(t)$ в одном окне;
- myplot2ph.m – кривая $y=y(t)$ и $y'=y'(t)$ в фазовом пространстве;
- myplot3.m – функция для изображения кривой решения $x=x(t)$, $y=y(t)$, $z=z(t)$.

Решатели дифференциальных уравнений. Функция ode45_1.m имеет три аргумента:

- strfunc – строка, определяющая функцию $F(t, y)$ правой части дифференциального уравнения $y' = F(t, y)$;
- tspan – вектор $[t_0, t_1]$ начального и конечного моментов времени;
- y0 – начальное значение решения дифференциального уравнения.

Выходные переменные – это массив отсчетов времени t и массив значений искомой функции $y(t)$. Строка strfunc – это обычная строка, записанная в синтаксисе MATLAB, с использованием (символьных) переменных t и y .

```
function [t, y] = ode45_1(strfunc, tspan, y0)
[t, y] = ode45(@odefunc, tspan, y0, [], strfunc) ;
```

Для функции ode45_2.m строка strfunc – это обычная строка, представляющая функцию переменных t , y и $y1$. Последняя переменная обозначает первую произ-

водную, $F(t, y, y') = F(t, y, y1)$. Аргумент $y0$ – это вектор начальных значений $y0 = [y(t_0), y'(t_0)]$. Выходные переменные – это массив отсчетов времени t и соответствующие массивы значений искомой функции и ее производной $y(t)$ и $y'(t)$.

```
function [t, y1, y2] = ode45_2(strfunc, tspan, y0)
    [t,y] = ode45(@odefunc2, tspan, y0, [], strfunc) ;
    y1 = y(:,1);
    y2 = y(:,2);
```

Для функции `ode45_3.m` строки `func1`, `func2`, `func3` – это функции правых частей системы дифференциальных уравнений, записанные с использованием переменных t , x , y и z . Аргумент $y0$ – это вектор начальных значений $y0 = [x(t_0), y(t_0), z(t_0)]$. Выходные переменные – это массив отсчетов времени t и соответствующие массивы значений искомым функций $x(t)$, $y(t)$ и $z(t)$.

```
function [t, y1, y2, y3] = ode45_3(func1, func2, func3, tspan, y0)
% Создание массива func из трех строк
    func = char(func1, func2, func3);
    [t,y] = ode45(@odefunc3, tspan, y0, [], func) ;
    y1 = y(:,1);
    y2 = y(:,2);
    y3 = y(:,3);
```

Правые части дифференциальных уравнений. Функции, представляющие правые части дифференциальных уравнений созданы с использованием функций `inline` и `feval`. Функция `inline` используется для создания функции из символьного выражения `strfunc`. Функция `feval` используется для вычисления значения созданной функции при заданных конкретных значениях параметров. Таким образом, функции `odefunc`, `odefunc2` и `odefunc3` вычисляют значение символьного выражения, заданного аргументом `strfunc` при заданных значениях переменных t и y . Для того, чтобы функция `inline` определяла бы функцию всех переменных, в выражение `strfunc` дописывается нулевое слагаемое, формально содержащее все переменные. Листинги функций:

```
function dydt = odefunc(t, y, strfunc)
% Если нет переменных t, y, то их формально добавляем,
% чтобы функция считалась зависящей от t, y
    strfunction = strcat(strfunc, '+ 0*t + 0*y') ;
    F = inline(strfunction) ;
    dydt = feval(F, t, y) ;
```

```
function dy = odefunc2(t, y, strfunc)
% Вектор-функция правой части дифференциального уравнения второго порядка
% определяется строкой strfunc, в которой задается функция
% переменных t, y, y1, F(t, y)
% переменная y – двумерная, состоит из двух компонент
% y = (y, y1) = (y(1), y(2))
% Переменную y обозначаем y(1) и пусть y(2) = y'(1)
% Тогда получается система
% y'(1)=y(2)
```

```
% y'(2) = F(t, y(1), y(2))
% Создание функции f1(x)=x для задания соотношения y'(1)=y(2)
f1 = inline('x') ;
dy(1,:) = feval(f1, y(2)) ;
% Если нет переменных t, y, y1, то правильнее их добавить формально:
strfunction = strcat(strfunc, '+ 0*t + 0*y1 + 0*y') ;
f2 = inline(strfunction) ;
dy(2,:) = feval(f2, t, y(1), y(2)) ;

function dy = odefunc3(t, yv, func)
% func: F1(t, x, y, z), F2(t, x, y, z), F3(t, x, y, z),
% Если нет переменных t, x, y, z, то правильнее их добавить формально:
strfunction1 = strcat(func(1,:), '+ 0*t + 0*x + 0*y + 0*z') ;
strfunction2 = strcat(func(2,:), '+ 0*t + 0*x + 0*y + 0*z') ;
strfunction3 = strcat(func(3,:), '+ 0*t + 0*x + 0*y + 0*z') ;
F1 = inline(strfunction1) ;
F2 = inline(strfunction2) ;
F3 = inline(strfunction3) ;
dy(1,:) = feval(F1, t, yv(1), yv(2), yv(3)) ;
dy(2,:) = feval(F2, t, yv(1), yv(2), yv(3)) ;
dy(3,:) = feval(F3, t, yv(1), yv(2), yv(3)) ;
```

Построение графиков. Соответствующие функции для построения графиков в комментариях не нуждаются.

```
function myplot(t, y)
plot(t,y)
xlabel('Время t')
ylabel('Решение')
title('График функции (решения)')

function myplot2(t, y, y1)
plot(t,y)
hold on
plot(t,y1,'r-')
xlabel('Время t')
ylabel('Решение y, y\prime')
title('Графики функций y=y(t) и y\prime=y\prime(t)')

function myplotph2(x, y)
L=length(x);
plot(x,y)
hold on
plot(x(1),y(1),'r.')
plot(x(L),y(L),'k.')
xlabel('Решение y')
ylabel('Производная y\prime')
title('График фазовой кривой y=y(t), y\prime=y\prime(t)')

function myplot3(x, y, z)
L=length(x);
plot3(x,y,z)
```

```
hold on
plot3(x(1),y(1),z(1),'r.')
plot3(x(L),y(L),z(L),'k.')
xlabel('Ось x')
ylabel('Ось y')
zlabel('Ось z')
title('Траектория решения системы дифференциальных уравнений,
x=x(t), y=y(t), z=z(t)')
```

Для создания .NET компонента будем использовать среду разработки Deployment Tool, ее описание достаточно подробно изложено в предыдущих главах. Делаем текущим рабочим каталогом MATLAB каталог проекта D:\Work\Diff_Ury и вызываем среду разработки командой MATLAB:

```
deploytool
```

Выбираем создание .NET компонента по имени **Ode**, класс назовем **Odeclass**, он включает все указанные выше функции. Определим каталог проекта D:\Work\Diff_Ury. Компонент представляет собой частную сборку **Ode.dll** и технологический файл **Ode.ctf**, которые находятся в каталоге D:\Work\Diff_Ury\Ode\distrib.

Создание Windows-приложения

В среде разработки Visual Studio 2005 на языке C# создадим проект по имени **OdeWinAppl** (решение **OdeWinAppl**) и поместим его в каталог D:\Work\Diff_Ury\. Создание приложения для вычисления интегралов и построения графиков проведем в несколько этапов.

1. Проектирование формы приложения. Форму приложения **Form1** назовем «Дифференциальные уравнения» – это имя нужно внести в свойство **Text** инспектора свойств формы. На форму **Form1** приложения поместим компонент вкладки **TabControl** из контейнера компонентов **Common Control**. Первая вкладка – для дифференциального уравнения 1-го порядка, поэтому назовем ее «1-го порядка» (свойство **Text** инспектора свойств), вторая – для уравнения 2-го порядка и третья – для системы из трех уравнений первого порядка. Соответственно определим имена вкладок (рис. 2.7.7).

На первую вкладку поместим несколько пояснительных меток **Label**, текстовых полей **TextBox** и две кнопки (рис. 5.7.7). В первое текстовое поле вводится строка, определяющая функцию в соответствии с синтаксисом MATLAB для символьных переменных. В два следующих текстовые окна вводятся пределы изменения параметра t , далее – текстовое поле для начального значения y_0 . Две кнопки **button1** и **button4** – для решения дифференциального уравнения и отображения графика решения.

Соответственно спроектируем другие вкладки (рис. 5.7.9 и 2.7.12).

2. Подключение внешних классов. Предполагается, что приложение будет использовать методы класса **Odeclass**, созданного .NET Builder компонента. Для того, чтобы это было возможным, мы должны подключить внешние классы компонента **Ode** и классы среды исполнения MATLAB MCR. Для этого нужно добавить в проекте ссылку на компонент **MWArray.dll**, содержащий классы **MWArray**.

Данный компонент зарегистрирован в системе, поэтому достаточно сделать следующее: в Visual Studio 2005 из меню **Project** \Rightarrow **Add Reference** открыть диалоговое окно **Add Reference** и в нем найти и отметить строку с именем компонента (рис. 5.7.2).

Также необходимо добавить в проекте ссылку на компонент Ode.dll, который находится в подкаталоге distrib каталога D:\Work\Diff_Ury\Ode\.. Для этого в диалоговом окне **Add Reference** нужно открыть вкладку **Browse** (рис. 5.7.2) и найти и отметить файл Ode.dll.

После указания ссылок добавим пространства имен указанных компонентов в код программы Form1.cs:

```
using MathWorks.MATLAB.NET.Utility;  
using MathWorks.MATLAB.NET.Arrays;  
using Ode;
```

3. Описание событий. Мы рассмотрим уравнение, разрешенное относительно производной, вида $y'=F(t,y)$. На первой вкладке формы заданы текстовые окна для задания функции $F(t,y)$ в синтаксисе MATLAB, для задания начального t_0 и конечного t_1 моментов времени и для задания начального значения $y(t_0)$.

Предполагается, что по нажатию первой кнопки «Решение» происходит решение дифференциального уравнения. По кнопке «График» – построение графика решения.

По нажатию кнопки «Решение» программа должна выполнить следующие действия:

- считать строку функции (func) из текстового поля textBox1, пределы изменения параметра (st0, st1) и начальное значение sy0 из соответствующих текстовых полей;
- преобразовать полученные текстовые данные (типа string) в те типы, которые может принять метод интегрирования: MWCharArray mw_strfunc и double t0, t1, y0;
- создать экземпляр (obj) класса Odeclass и вызвать функцию ode45_1 для решения дифференциального уравнения;
- из массива MWArray[] выбрать элементы (mw_T1 и mw_Y1), представляющие массивы отсчетов времени и решения в типе MWNumericArray;

В соответствии с этой программой и будем описывать событие button1_Click. Двойной клик по кнопке «Вычислить» на форме создает в коде Form1.cs пустую заготовку для описания события:

```
private void button1_Click(object sender, EventArgs e)  
{  
  
}
```

в которую и пишется программа. Приведем соответствующий фрагмент листинга программы Form1.cs для решения дифференциального уравнения. Отметим, что строка функции, пределы интегрирования, начальное значение и выходные мас-

сивы объявлены вне блока события button1_Click. Это сделано для того, чтобы эти переменные были бы видны из блока построения графика решения.

```
// -- Дифференциальное уравнение первого порядка -----
// Объявления входных переменных
double t0, t1, y0;           // Границы времени и начальное y0
double[] tspan = new double[2]; // Интервал [t0, t1]
string func, st0, st1, sy0;   // Их строковые представления
// Объявления выходных переменных
MWArray[] mw_ArrayOut = null; // Выходной массив параметр
MWNumericArray mw_T1 = null;  // Выходной параметр время
MWNumericArray mw_Y1 = null;  // Выходной параметр, решение

private void button1_Click(object sender, EventArgs e)
{
    // Решение дифференциального уравнения
    try
    {
        func = textBox1.Text; // Считывание с текстового поля
        st0 = textBox2.Text;
        st1 = textBox3.Text;
        sy0 = textBox4.Text;

        t0 = Convert.ToDouble(st0); // Преобразование в число
        t1 = Convert.ToDouble(st1);
        y0 = Convert.ToDouble(sy0);
        // Входные параметры
        // Преобразование строки функции в тип MWCharArray
        MWCharArray mw_strfunc = new MWCharArray(func);

        tspan[0] = t0;
        tspan[1] = t1;
        MWNumericArray mw_tspan = new MWNumericArray(tspan);

        // Экземпляр obj класса Odeclass компонента
        Odeclass obj = new Odeclass();

        // Обращение к методу ode45_1 для решения ОДУ
        mw_ArrayOut = obj.ode45_1(2, mw_strfunc, mw_tspan, y0);

        // Выбор первого элемента из массива MWArray[]
        mw_T1 = (MWNumericArray)mw_ArrayOut[0];
        // Выбор второго элемента из массива MWArray[]
        mw_Y1 = (MWNumericArray)mw_ArrayOut[1];
    }
    catch { }
}
```

Приведем код программы, описывающей событие button1_Click – нажатия кнопки «График»:

```
// Построение графика y=y(t)
private void button11_Click(object sender, EventArgs e)
{
    try
    {
        Odeclass obj_plot = new Odeclass();
        obj_plot.myplot(mw_T1, mw_Y1);
    }
    catch { }
}
```

Для других вкладок события описываются совершенно аналогично.

4. Построение и выполнение приложения. Теперь построим и выполним приложение. В Visual Studio 2005 для этой цели служит кнопка **Start Debugging** инструментальной панели Visual Studio, а также меню **Build**. При использовании **Start Debugging** создается сборка OdeWinAppl.exe, которая сохраняется в подкаталоге bin\Debug\ каталога .\Diff_Ury\OdeWinAppl\OdeWinAppl\, а при использовании меню **Build** сборка OdeWinAppl.exe сохраняется в подкаталоге .\bin\Release\.

При построении компонента Ode, .NET Builder было указано, что создается частная сборка Ode.dll. Поэтому при построении приложения, файлы компонента Ode.dll и Ode.ctf автоматически помещаются в каталог, где находится приложение OdeWinAppl.exe. Следовательно, для работы приложения не нужно указывать пути к Ode.dll и Integr.ctf.

Работу программы рассмотрим на примере решения дифференциального уравнения вида:

$$y' = e^{-t/5}(\cos t - \sin t), \quad t_0 = 0, \quad t_1 = 20, \quad y(0) = 1.$$

Основное диалоговое окно и графики решений приведены на рис. 5.7.7 и 5.7.8. В текстовые поля вводим необходимые данные (для ввода функции нужно пользоваться синтаксисом MATLAB). Решение дифференциального уравнения

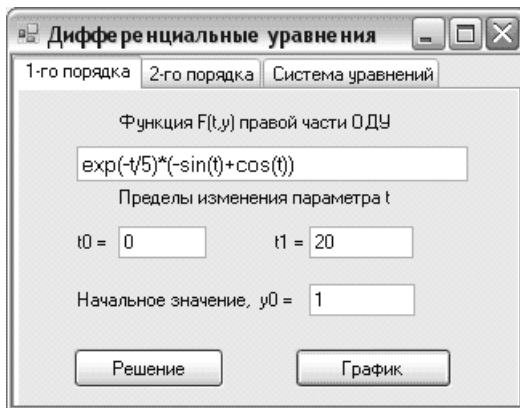


Рис. 5.7.7. Дифференциальное уравнение первого порядка

начинается при нажатии кнопки **Решение**. После окончания решения при нажатии кнопки **График** строится его график в стандартном графическом окне MATLAB, рис. 5.7.8.



Рис. 5.7.8. Решение дифференциального уравнения первого порядка

Решение дифференциального уравнения второго порядка. Рассматриваемое дифференциальное уравнение имеет вид $y'' = F(t, y, y')$. Предполагается, что аргумент t меняется на промежутке $[t_0, t_1]$. Решается обычная задача Коши нахождения решения $y=y(t)$, удовлетворяющего начальным условиям: $y(t_0)=y_0$ и $y'(t_0)=y_{10}$.

Вторая вкладка формы Form1 содержит элементы, необходимые для задания правой части дифференциального уравнения $y'' = F(t, y, y')$, задания промежутка $[t_0, t_1]$ изменения независимой переменной и начальных значений (рис. 5.7.9). Назначение четырех кнопок понятно из их названия. Отметим, что при задании строки функции следует использовать синтаксис MATLAB для символьных переменных и производную обозначать символом $y1$. В текстовые поля сразу введем тестовые данные для решения уравнения Ван-Дер-Поля. Описание событий для кнопок совершенно аналогично рассмотренному выше случаю уравнения первого порядка (полный листинг программы Form1.cs имеется на прилагаемом CD).

Рассмотрим для примера классический пример MATLAB решения уравнения Ван-Дер-Поля:

$$y'' = \mu(1 - y^2)y' - y$$

в случае значения параметра $\mu = 1$. Используя диалоговое окно (рис. 5.7.9) вводим все необходимые данные, учитывая, что в выражении функции вместо производной нужно использовать переменную $y1$. Графики решений приведены на рис. 5.7.10 и 5.7.11. Кривая в фазовом пространстве – это кривая, заданная параметрически $y = y(t)$, $y' = y'(t)$ в пространстве переменных (y, y') . Начальная точка такой кривой обозначена красной точкой, а конечная – черной.

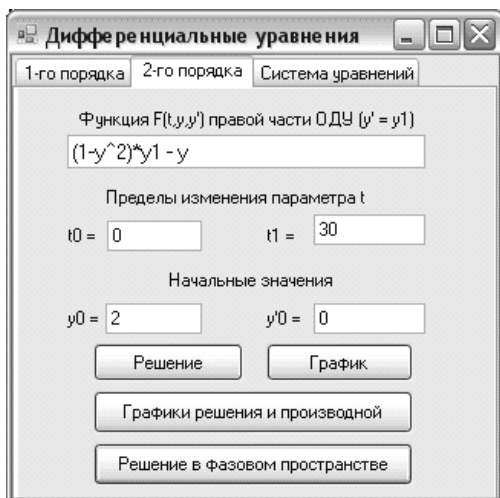


Рис. 5.7.9. Дифференциальное уравнение второго порядка

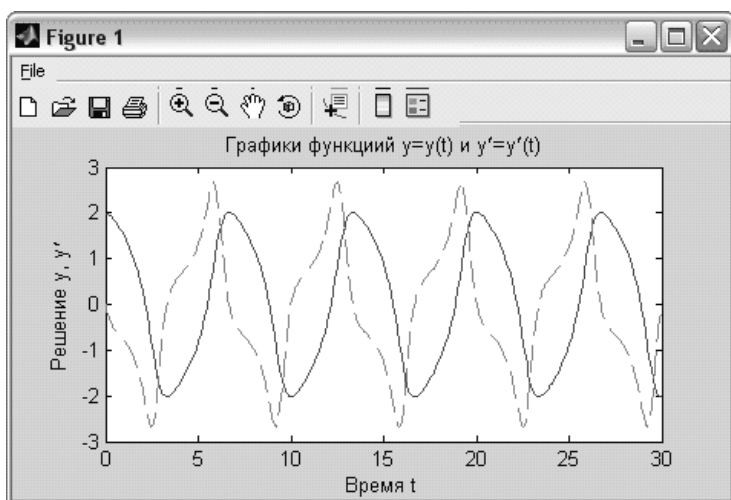


Рис. 5.7.10. Графики решения и производной решения

Решение системы дифференциальных уравнений. Рассмотрим решение задачи Коши следующей системы трех дифференциальных уравнений:

$$\begin{cases} x' = F_1(t, x, y, z) \\ y' = F_2(t, x, y, z), & t \in [t_0, t_1], \quad x(t_0) = x_0, \quad y(t_0) = y_0, \quad z(t_0) = z_0 \\ z' = F_3(t, x, y, z) \end{cases}$$

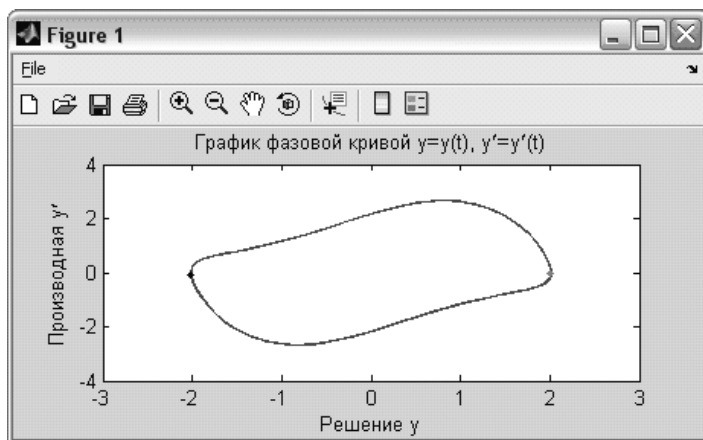


Рис. 5.7.11. Решение в фазовом пространстве

Третья вкладка формы Form1 содержит элементы, необходимые для задания функций правых частей системы дифференциальных уравнений, задания промежутка $[t_0, t_1]$ изменения независимой переменной и начальных значений (рис. 5.7.12). Назначение пяти кнопок понятно из их названия. При задании строки функции следует использовать синтаксис MATLAB для символьных переменных. В текстовые поля сразу введем тестовые данные для решения системы уравнений Лоренца. Описание событий для кнопок совершенно аналогично рассмотренному выше случаю уравнения первого порядка (полный листинг программы имеется на прилагаемом CD).

Работу созданного приложения рассмотрим на примере решения системы уравнений Лоренца:

$$\begin{cases} x' = s(y - x) \\ y' = -y + (r - z)x \\ z' = -bz + xy \end{cases}$$

в случае значения параметров $s = 10$, $r = 24$ и $b = 2.5$. Основное диалоговое окно и графики решений приведены на следующих рис. 5.7.12–5.7.14. Отметим, что изображение на рис. 5.7.13 можно вращать и увеличивать при помощи мыши и соответствующих кнопок инструментальной панели. Начальная точка траектории решения на графике рис. 5.7.13 отмечается красной точкой, а конечная – черной.

Отметим, что программа Form1.cs приложения OdeWinAppl.exe предусматривает хранение всех массивов решений под разными именами. Поэтому при переходе с одной вкладки на другую, массивы сохраняются и готовы для повторного вызова функции построения графика.

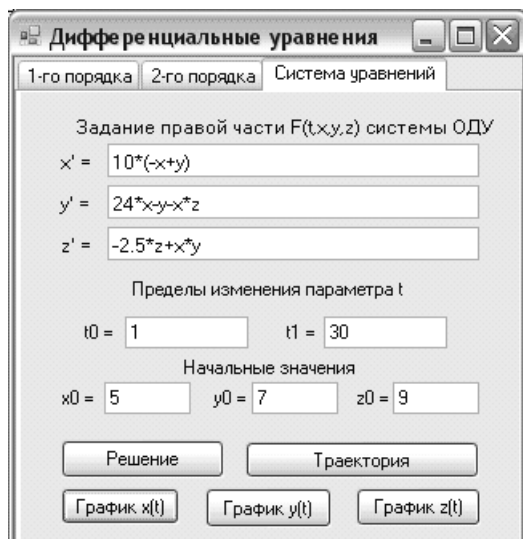


Рис. 5.7.12. Система дифференциальных уравнений первого порядка

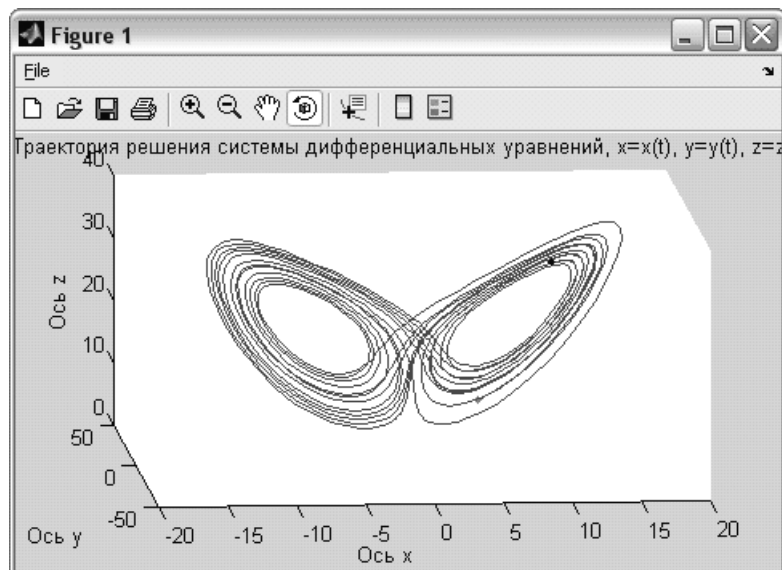


Рис. 5.7.13. Решение системы уравнений Лоренца

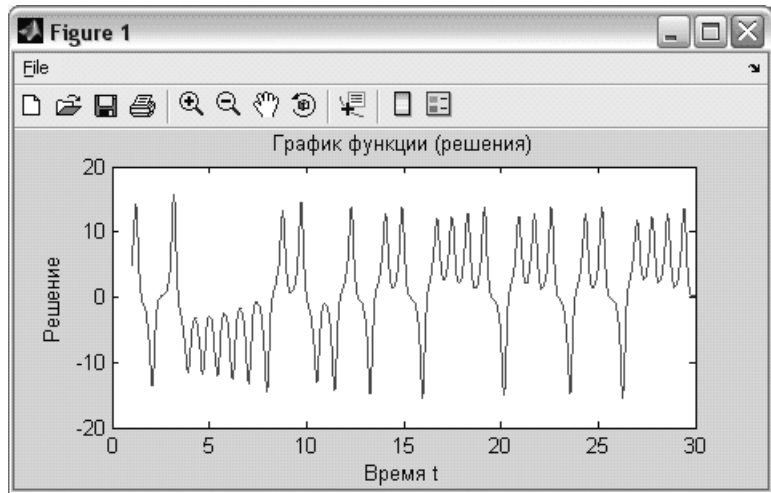


Рис. 5.7.14. График решения $x = x(t)$ системы Лоренца

5.7.3. Открытие, обработка и сохранение файлов

В этом разделе рассмотрим создание программы, в которой можно открыть текстовый форматированный файл, состоящий из нескольких столбцов, обработать его и записать результаты в виде форматированного файла.

Создание .NETкомпонента

Сначала подготовим необходимые m-функции. Для открытия и сохранения файла создадим несколько m-функций, с использованием m-функций MATLAB: fopen, fscanf и fprintf.

```
function y=mwOpeninf(name,n) %Открытие файла name из n столбцов до конца
fid = fopen(name,'rt')      %Открытие файла *.txt с данными
y=fscanf(fid,'%g',[n inf]); %Считывание данных из файла *.txt
fclose(fid);
```

```
function y=mwOpen_nm(name,n, m); %Открытие файла из n столбцов и m строк
fid=fopen(name,'rt')          %Открытие файла *.txt с данными
y=fscanf(fid,'%g',[n m]);     %Считывание данных из файла *.txt
fclose(fid);
```

```
function mwSave(name,format,x)
% Запись в текстовый форматированный файл со строкой формата format
% Например, format = '%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t\n'
fid = fopen(name,'wt')      %Открытие файла *.txt с данными
fprintf(fid,format,x);
fclose(fid);
```

Загружаемый файл представляет собой несколько столбцов чисел, разделенных табулятором. Это могут быть записи кардиосигнала по 8-ми стандартным каналам. Поэтому в дальнейшем весь такой файл мы будем называть многомерным сигналом, а каждый столбец будем называть отдельным каналом. В качестве примера рассмотрим следующие операции над многомерным сигналом:

- построение графиков всех каналов сигнала;
- выбор фрагмента сигнала и построение графиков фрагмента;
- выбор отдельного канала, построение его графика;
- вейвлет-разложение выбранного канала и построение графиков коэффициентов разложения;
- пороговая обработка вейвлет-коэффициентов выбранного канала с целью удаления шума;
- построение графиков исходного канала и после удаления шума;
- запись обработанного канала в исходный многомерный фрагмент и сохранение результатов в виде многомерного форматированного текстового файла.

В соответствии с этим планом обработки сигнала определим необходимые m-функции.

Функции построения графиков вполне аналогичны тем, что были использованы в предыдущих примерах. Отметим только следующую m-функцию для построения графиков всех каналов в одном окне, когда число графиков задается в качестве аргумента функции:

```
function mySubplot(X,m) % Графики m строк матрицы
for i=1:m
    LX =length(X(i,:))
    MaxX = max(X(i,:));
    minX = min(X(i,:));
    subplot (m,1,i); plot(X(i,:)); axis([1 LX minX MaxX]);
end
```

Выбор фрагмента сигнала X, начиная от отсчета с номером n до отсчета с номером m, осуществляется следующей функцией:

```
function Y = Fragment(X,n,m) % Выбор фрагмента многомерного сигнала
Y=X(:,n:m);
```

Вейвлет-разложение и восстановление (до второго уровня разложения) осуществляются функциями пакета Wavelet Toolbox [См]:

```
function [cA,cD]=mydwt(X,LoD,HiD) % Вейвлет-разложение
[cA,cD] = dwt(X,LoD,HiD);
```

```
function Z=myidwt(cA,cD,LoR,HiR,L) % Нужно указывать длину массива L
Z=idwt(cA,cD,LoR,HiR,L);
```

Здесь LoD,HiD – массивы фильтров разложения выбранного вейвлета и LoR,HiR – массивы фильтров восстановления вейвлета.

Обработка (удаление шума) канала производится по детализирующим вейвлет-коэффициентам и заключается в удалении d% наименьших по абсолютной

величине коэффициентов. Для этого рассматривается абсолютная величина $\text{abs}(X)$ элементов сигнала, производится упорядочивание элементов по возрастанию и заносятся $d\%$ первых элементов в упорядоченном списке:

```
function Y = den(X,d) % Удаление d% наименьших элементов
LX=length(X);
B=floor(LX*(d/100));
[SI,SI]=sort(abs(X),'ascend');
for i=1:B;
    X(SI(i))=0;
end
Y=X;
```

После такой операции удаления шумовых компонент сигнал восстанавливается функцией `myidwt` по новым коэффициентам вейвлет-разложения. Запись обработанного сигнала делается при помощи функции

```
function Z=Vstavka(X,Y,k)
% Замена k-ой строки из массива X на строку Y
X(k,:)=Y;
Z=X;
```

Для создания .NET компонента будем использовать среду разработки `Deployment Tool`, ее описание достаточно подробно изложено в предыдущих главах. Делаем текущим рабочим каталогом MATLAB каталог проекта `D:\Work\Files` и вызываем среду разработки командой MATLAB:

```
deploytool
```

Выбираем создание .NET компонента по имени **fopensave**, класс назовем **fopensaveclass**, он включает все указанные выше функции. Определим каталог проекта `D:\Work\Files`. Компонент представляет собой частную сборку **fopensave.dll** и технологический файл **fopensave.ctf**, которые находятся в каталоге `D:\Work\Files\distrib`.

Создание приложения

В среде разработки Visual Studio 2005 на языке C# создадим проект по имени **Open_Save** (решение `Open_Save`) и поместим его в каталог `D:\Work\Files`. Создание приложения для вычисления интегралов и построения графиков проведем в несколько этапов.

1. Проектирование формы приложения. Форму приложения **Form1** назовем «Вейвлет-разложение» – это нужно внести в свойство **Text** инспектора свойств формы. На форму **Form1** приложения поместим несколько меток **Label**, текстовых полей **TextBox** и кнопок (рис. 5.7.15). В текстовых полях можно указать параметры загрузки сигнала, начало и конец фрагмента, номер канала, количество (в процентах) числа удаляемых коэффициентов и формат записи в файл результата. На рис. 5.7.15 указаны значения по умолчанию: число каналов равно 8 (для кардиосигнала), сигнал загружается до конца (`inf`), размеры фрагмента совпадают с размерами всего сигнала, канал № 1, удаляемых коэффициентов – 0%, строка формата сохранения – для 8 столбцов чисел с фиксированной запятой, разделен-

3. Описание программы. Рассмотрим основные элементы программы Form1.cs. Полные тексты листингов приведены на прилагаемом компакт-диске.

Раздел 1. Загрузка данных.

```
// ---- Загрузка файла и считывание данных -----
private void MenuItemOpen_Click(object sender, EventArgs e)
{
    try
    {
        Ns = textBox1.Text; // Считывание числа каналов
        N = Convert.ToInt32(Ns); // Преобразование в число
        // Загрузка файла
        openFileDialog1.ShowDialog();
        string fileName = openFileDialog1.FileName; // Получаем имя файла
        // Перевод строки в тип MWCharArray
        MWCharArray fil = new MWCharArray(fileName);
        // Экземпляр класса fopensave компонента
        fopensaveclass open = new fopensaveclass();
        // Обращение к методу mwOpeninf
        mw_ArrayOut = open.mwOpeninf(1, fil, N);
        X = (MWNumericArray)mw_ArrayOut[0];
        // Определение длины сигнала методом myLength
        MWArray[] mw_LenOut = null; // Выходной массив как MWArray[]
        mw_LenOut = open.myLength(1, X);
        L = (MWNumericArray)mw_LenOut[0];
        double Len = (double)L; // Преобразование в число
        Len_sig = Convert.ToString(Len); // Преобразование в строку
        textBox2.Text = @Len_sig; // Запись длины сигнала в текстовое поле
        textBox5.Text = @Len_sig;

        // Загрузка вейвлета Хаара - по умолчанию
        double[] double_LoD = { 0.70710678118655, 0.70710678118655 };
        double[] double_HiD = {-0.70710678118655, 0.70710678118655 };
        double[] double_LoR = { 0.70710678118655, 0.70710678118655 };
        double[] double_HiR = {0.70710678118655, -0.70710678118655 };
        // Преобразование векторов в тип MWNumericArray
        LoD = new MWNumericArray(double_LoD);
        HiD = new MWNumericArray(double_HiD);
        LoR = new MWNumericArray(double_LoR);
        HiR = new MWNumericArray(double_HiR);
    }
    catch { }
}
```

Раздел 2. Загрузка вейвлета из файла. Каждый файл вейвлета представляет собой пять столбцов чисел, разделенных табуляцией. Первые 4 столбца – это фильтры разложения и восстановления в следующем порядке:

- LoD – низкочастотный фильтр разложения;
- HiD – высокочастотный фильтр разложения;

- LoR – низкочастотный фильтр восстановления;
- HiR – высокочастотный фильтр восстановления.

Файлы вейвлетов находятся в подкаталоге WF каталога проекта и в рабочем каталоге D:\Work\Files. Их имена – принятые в MATLAB сокращенные названия вейвлетов, например db6.txt – файл фильтров вейвлета Добеши db6.

```
// ----- Загрузка вейвлета -----
private void WaveMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.ShowDialog();
        string WavName = openFileDialog1.FileName; // Получаем имя файла
        MWCharArray Wav = new MWCharArray(WavName); // Перевод строки
                                                    имени файла в тип MWCharArray
// Экземпляр класса fopensave компонента
        fopensaveclass open = new fopensaveclass();
        MWArray[] mw_WF = null; // Выходной массив как MWArray[]
// Обращение к методу mwOpeninf
        mw_WF = open.mwOpeninf(1, Wav, 5);
        MWNumericArray Filt = null;
        Filt = (MWNumericArray)mw_WF[0];
        MWArray[] LoDarr = null; // Фильтр низкоч. разл.
        MWArray[] HiDarr = null; // Фильтр высокоч. разл.
        MWArray[] LoRarr = null; // Фильтр низкоч. восстан.
        MWArray[] HiRarr = null; // Фильтр высокоч. восстан.
        LoDarr = open.Canal(1, Filt, 1);
        HiDarr = open.Canal(1, Filt, 2);
        LoRarr = open.Canal(1, Filt, 3);
        HiRarr = open.Canal(1, Filt, 4);
        LoD = (MWNumericArray)LoDarr[0];
        HiD = (MWNumericArray)HiDarr[0];
        LoR = (MWNumericArray)LoRarr[0];
        HiR = (MWNumericArray)HiRarr[0];
    }
    catch { }
}
```

Раздел 3. Выбор и просмотр фрагмента сигнала. В текстовых полях textBox4 и textBox5 указываются начало и конец фрагмента и затем делается выборка методом Fragment. Находится длина фрагмента и строятся графики всех N каналов

```
// ---- Выбор и просмотр фрагмента сигнала -----
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        K0s = textBox4.Text; // Начало фрагмента
        K1s = textBox5.Text; // Конец фрагмента
        K0 = Convert.ToInt32(K0s);
```

```

        K1 = Convert.ToInt32(K1s);
// Экземпляр класса fopensave компонента
        fopensaveclass Fragn = new fopensaveclass();
// Обращение к методу Fragment для выбора фрагмента сигнала
        MWArray[] mw_Fr = null; // Выходной массив как MWArray[]
        MWArray[] mw_LenFr = null; // Выходной массив как MWArray[]
        mw_Fr = Fragn.Fragment(1, X, K0, K1);
        Fr = (MWNumericArray)mw_Fr[0];
        Fr_dn = (MWNumericArray)mw_Fr[0]; // Фрагмент для очистки
        mw_LenFr = Fragn.myLength(1, Fr); // Определение длины фрагмента
        LFr = (MWNumericArray)mw_LenFr[0];
        Fragn.mySubplot(0, Fr, N);
    }
    catch {}
}

```

Раздел 4. Вейвлет-разложение. Зададим разложение k -го канала X_k до второго уровня. Обозначим $cA1$ и $cA2$ – коэффициенты аппроксимации первого и второго уровня разложения, $cD1$ и $cD2$ – коэффициенты детализации первого и второго уровня разложения [См]. Считается, что шумовые компоненты сигнала сосредоточены в малых вейвлет-коэффициентах детализации $cD1$ и $cD2$. Удаление этих малых коэффициентов с последующим восстановлением и приводит к удалению шума в сигнале.

```

// ---- Вейвлет-разложение фрагмента канала -----
private void button3_Click(object sender, EventArgs e)
{
    try
    {
// Экземпляр класса fopensave компонента
        fopensaveclass Wav = new fopensaveclass();
// Обращение к методу mydwt для построения вейвлет-разложения
        MWArray[] mw_WavDec1 = null; // Выходной массив как MWArray[]
        mw_WavDec1 = Wav.mydwt(2, Xk, LoD, HiD);
        cA1 = (MWNumericArray)mw_WavDec1[0];
        cD1 = (MWNumericArray)mw_WavDec1[1];
// Второй уровень разложения
        MWArray[] mw_WavDec2 = null; // Выходной массив как MWArray[]
        mw_WavDec2 = Wav.mydwt(2, cA1, LoD, HiD);
        cA2 = (MWNumericArray)mw_WavDec2[0];
        cD2 = (MWNumericArray)mw_WavDec2[1];
        Wav.plot_dec(0, Xk, cA2, cD1, cD2); // Графики коэффициентов
    }
    catch{}
}

```

Раздел 5. Сохранение данных. Зададим сохранение фрагмента в файл из N столбцов чисел, разделенных табуляцией. Формат сохранения берется из текстового поля textBox8. По умолчанию в этом текстовом окне содержится следующая строка формата `%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t\n`, указывающая на

сохранение в 8 столбцах чисел с фиксированной запятой, разделенных табуляцией. Можно указывать и другие методы записи в файл

```
// ----- Сохранение фрагмента -----

private void FragmentSave_Click(object sender, EventArgs e)
{
    try
    {
        // Считывание строки формата
        format = textBox8.Text;
        saveFileDialog1.ShowDialog();
        string fileName = saveFileDialog1.FileName; // Задаем имя файла
        // Экземпляр Save класса fopensave компонента
        fopensaveclass Save = new fopensaveclass();
        // Перевод строк в тип MWCharArray
        MWCharArray fil = new MWCharArray(fileName);
        MWCharArray form = new MWCharArray(format);
        // Обращение к методу mwSave
        Save.mwSave(0, fil, form, Fr);
    }
    catch {}
}
```

Замечание 1. Полный текст программы Form1.cs приведен на прилагаемом компакт-диске. Указанные фрагменты листинга показывают, что достаточно сложные элементы программы выглядят очень просто с использованием функций .NET компонента fopensave.

Тестирование приложения. В заключение проверим работу созданного приложения. Загрузим тестовый файл Test_Sig.txt с записью кардиосигнала, он находится в рабочем каталоге D:\Work\Files. Выберем сигнал вейвлета Добеши db6. Файлы вейвлетов находятся в подкаталоге WF каталога проекта D:\Work\Files\Open_Save\Open_Save и в рабочем каталоге D:\Work\Files. Их имена – принятые в MATLAB сокращенные названия вейвлетов, например db6.txt – файл фильтров вейвлета Добеши db6. Все эти файлы и исходные тексты программы находятся на прилагаемом компакт-диске в каталоге Gl_5_C#_Examples\Files.

После просмотра графиков кардиосигнала выберем 5-ый канал кардиосигнала и его интересный фрагмент – начиная с отсчета 250 и до 500, рис. 5.7.17 и 5.7.18. Проведем его вейвлет-разложение, рис. 5.7.19. Для очистки от шума удалим 98% коэффициентов cD1 детализации первого уровня разложения и 97% коэффициентов cD2 детализации второго уровня разложения. По оставшимся коэффициентам восстановим сигнал и сравним его с исходным. Результаты вполне удовлетворительны, рис. 5.7.20. В заключение вставим обработанный канал 5 в полный фрагмент сигнала (это происходит автоматически по кнопке «Удаление шума») и сохраним его в файле Save_1.txt при помощи опции меню **Файл/Сохранить обработанный фрагмент**.

Замечание 2. Во всех созданных приложениях используется частные сборки компонентов .NET Builder. Это значит, что dll- и ctf-файлы при запуске приложения автоматически помещаются в каталог, где находится исполняемый exe-файл

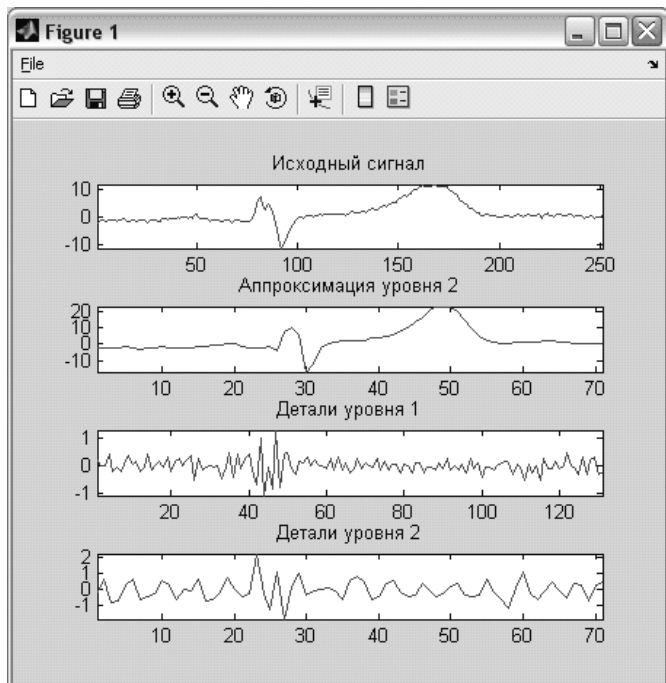


Рис. 5.7.19. Коэффициенты вейвлет-разложения фрагмента

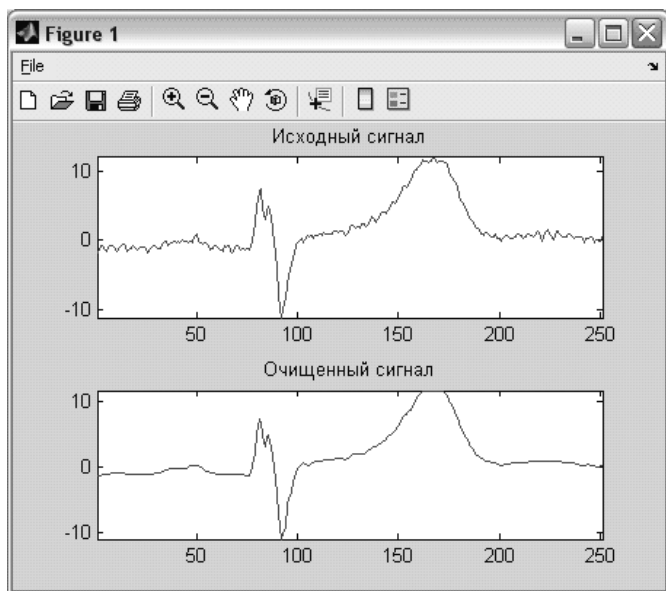


Рис. 5.7.20. Исходный и очищенный от шума канал

Предметный указатель

А

Атрибуты в C#, 354

Б

Байт-код Java, 174
Библиотека базовых классов
FCL, 348
Библиотека классов .NET
MWArray, 381
Блок try-catch MATLAB, 81
Браузер справки MATLAB, 24

В

Вложенные классы Java, 199
Внешние интерфейсы
MATLAB, 138
Встроенные типы C#, 358

Д

Декларация сборки, 346
Динамические массивы,
C#, 367
Динамические массивы
VBA, 338

З

Значимые типы C#, 357

И

Иерархия классов, 193
Импорт классов и пакетов
Java, 200
Индексация в MATLAB, 33
Инкапсуляция в Java, 192
Интерфейс .NET Builder
стандартный, 387
feval, 388
единственного вывода, 386
Интерфейс Java, 201
Информационные
XML-теги, 353
Исходные коды компонента
Excel, 307

К

Класс Java, 192
Класс mwArray, 128, 156
Класс mwExceptions, 137
Класс mwString, 136
Класс C#, 373
Команда mcs, 101
Комментарий Java, 177
Компонент .NET Builder, 380
Компоненты
визуальные Visual C#, 419
невизуальные
Visual C#, 419
Константы Java, 177
Конструктор класса
Java, 196

М

М-файл
сценарий (скрипт), 43
функция, 43
Магический квадрат, 31
Макросы mcs, 106
Массив MATLAB
символов, 49
структура, 62
ячеек, 55
Массивы Java, 189
Массивы в C#, 364
Мастер функций, 300
Математическая библиотека
C++ MATLAB, 153
Метаданные, 346
Метод main Java, 198
Многомерные массивы
Java, 191
Модификатор final Java, 195

Н

Норма матрицы, 39

О

Обработчик событий
Visual C#, 421

Общезыковая среда
выполнения
CLR, 345, 348
Оператор MATLAB
for...end, 79
if...else...end, 77
swith...case...end, 78
while...end, 79
двоеточия, 35
объединения, 33
Оператор new Java, 196
Операторы Java, 186
Операторы MATLAB
арифметические, 73
логические, 74
отношения, 74
Операции в C#, 367
Описание класса Java, 193
Описание констант VBA, 337
Описание массивов VBA, 337
Описание объектной
переменной VBA, 338
Описание переменных
VBA, 336
Опции mcs, 102
Отражение в C#, 378

П

Пакеты Java, 199
Перегрузка метода, 258
Переменные MATLAB
varargin, 70
varargout, 70
глобальные, 72
локальные, 72
Переменные экземпляра
класса Java, 197
Перенос строки кода VBA, 335
Перечисления C#, 362
Подфункции MATLAB, 68
Преобразование MATLAB
систем счисления, 54
чисел в символы, 52
Преобразование простых
типов Java, 181
Преобразование Фурье
дискретное, 244, 311

Приведение типов Java, 180
 Приведение типов в C#, 360
 Прimitives типы класса
 mwArray, 127
 Промежуточный язык IL, 346
 Пространство имен, 347
 Процедура C++ML
 cat(), 161
 horzcat(), 160
 vertcat(), 160
 Процедуры VBA, 340
 Псевдокод MATLAB, 69
 Псевдокомментарий
 %#external, 108
 %#function, 109

P

Рабочая область MATLAB, 23
 Рабочий стол MATLAB, 21
 Реализация интерфейса
 Java, 201
 Регистрация компонента
 Excel, 299, 309
 Редактор массива
 MATLAB, 23
 Редактор/отладчик
 MATLAB, 26
 Родные ресурсы C#, 345

C

Сборка, 346
 Сборка мусора C#, 347
 Сигнатура метода
 Java, 195, 258
 Сингулярные числа, 38
 Системные константы
 MATLAB, 27
 Среда выполнения .NET
 Framework, 348
 Среда выполнения MCR
 MATLAB, 94
 Ссылочные типы C#, 357
 Стандартная Система Типов
 CTS, 349
 Статические методы класса
 Java, 197
 Статические переменные
 класса Java, 197

Строковые и символьные
 типы C#, 361

T

Текущий каталог
 MATLAB, 24
 Технологический файл
 компоненты CTF, 96
 Тип данных MATLAB
 double, 30
 single, 30
 sym, 41
 дескриптор функции, 30
 логический массив, 29
 целые числа int*, 29
 Тип данных mwArray, 156
 Тип данных mxArray, 138
 Тип данных Variant, 335
 Типы Java
 вещественные, 183
 логические, 179
 простые, 178
 ссылочные, 178
 строки, 180
 целые, 180
 Типы данных VBA, 335

Y

Упаковка и распаковка
 в C#, 359
 Управляемые данные, 345
 Управляемый исполняемый
 модуль, 345
 Управляемый код, 345
 Утилита mbuild, 110

Ф

Файлы групп Компилятора
 MATLAB, 106
 Файлы обертки Компилятора
 MATLAB, 107
 Формат числа MATLAB, 32
 Функции MATLAB
 комплексных чисел, 33
 логические, 75
 массивов символов, 49

матриц, 37
 округления чисел, 32
 создания одномерных
 массивов, 34
 создания ячеек, 56
 справочной системы, 26
 управления памятью, 83, 84
 Функция
 mclInitializeApplication, 116
 mclTerminateApplication, 117

Функция MATLAB

cat, 62
 celldisp, 57
 cellplot, 57
 clear, 69
 eig, 38
 eval, 80
 fclose, 45
 feval, 80
 fieldsnames, 64
 fopen, 44
 fprintf, 45
 fscanff, 45
 horzcat, 35
 load, 48
 nargin, 70
 nargsout, 70
 save, 47
 struct, 64
 svd, 38
 vertcat, 35
 Функция интерфейса
 mlf, 118
 mlx, 118

X

Хэш-код, 348

Ч

Частный каталог
 MATLAB, 69

Э

Экземпляр класса Java, 192
 Элементарные матрицы, 36

Содержание компакт-диска

Прилагаемый к книге компакт-диск содержит исходные тексты примеров программ, рассматриваемых в книге.

Глава 2. Примеры к главе 2 приведены в двух каталогах.

Каталог **G1_2_Compiler_Examples** содержит несколько примеров, которые показывают создание приложений и библиотек при помощи Компилятора MATLAB версии 4.6 (R2007a).

1. Подкаталог **magic** содержит пример создания автономного приложения для вычисления магического квадрата.
2. Подкаталог **Par_2_4_1_C-Lib** содержит пример создания C-библиотеки совместного использования из трех m-файлов **addmatrix.m**, **multiplymatrix.m** и **eigmatrix.m** создание консольного приложения **matrixdriver.exe** на C, которое использует функции созданной библиотеки **libmatrix.dll**.
3. Подкаталог **Par_2_4_1_CPP_Lib** содержит пример создания CPP-библиотеки совместного использования из трех m-файлов **addmatrix.m**, **multiplymatrix.m** и **eigmatrix.m** создание консольного приложения **matrixdriver.exe** на CPP, которое использует функции созданной библиотеки **libmatrixp.dll**.
4. Подкаталог **Par_2_4_2_Mrank_Appl_1** содержит пример создания автономного приложения для вычисления рангов магических квадратов, из двух m-файлов, один из которых является «главным» и при своей работе обращается ко второму файлу.
5. Подкаталог **Par_2_4_2_Mrank_Appl_2** содержит пример создания автономного приложения для вычисления рангов магических квадратов, из двух m-файлов и файла на C.

Каталог **G1_2_Borland_Builder_Examples** содержит несколько примеров создания Windows-приложений на Borland C++ Builder, которые используют математические библиотеки MATLAB.

1. Подкаталог **Project_1** содержит пример создания проекта (**Project_1.exe**), в котором предусмотрено чтение числовых данных из файла в формате ASCII, обработка данных функциями математической библиотеки C++ MATLAB, вывод результатов в графическое окно и запись результатов в файл в формате ASCII.
2. Подкаталог **Project_2** содержит пример создания проекта (**Project_2.exe**) на Borland C++ Builder, в котором предусмотрено чтение числовых данных из текстового файла в формате ASCII, обработка сигнала с использованием функций математической библиотеки C++ MATLAB и построение графиков полученных числовых массивов. Основная цель примера – показать, что графики данных из массива **mwArray** строятся так же просто, как и графики на основе обычных массивов C++.

3. Подкаталог `Borland_Builder_Appl` содержит полный набор файлов математической библиотеки C++ MATLAB и файлы среды исполнения Borland C++Builder, необходимые для работы приложений `Project_1.exe` и `Project_2.exe`.
4. Подкаталог `BorlBui_Files` содержит необходимые файлы среды исполнения Borland C++Builder, необходимые для работы приложений.
5. Подкаталог `Factor2` содержит проекты создания двух библиотек на C и C++ при помощи Компилятора MATLAB версии 4.6. Каждая библиотека содержит одну функцию `fact2` вычисления $n!$, т.е. $n! = 1*3*5*...*n$, если n – нечетное и $n! = 2*4*6*...*n$, если n – четное. Подкаталог `Project_Fact` содержит проект на Borland C++Builder создания приложения, использующего функцию C-библиотеки `Factlib2C.dll`.

Глава 3. Каталог `Gl_3_javabuilder_examples` содержит примеры создания компонентов для Java и приложений Java, которые используют созданные Компилятором MATLAB компоненты.

1. Подкаталог `magic_square` содержит пример создания компонента для Java, который инкапсулирует функцию MATLAB для вычисления магического квадрата. Подкаталог `MagicDemoJavaApp` содержит пример консольного приложения, `getmagic`, созданного на Java, которое использует метод `makesqr` созданного класса `magic` и в качестве результата отображает массив магического квадрата на экране.
2. Подкаталог `SpectraExample` содержит пример создания компонента для Java, который инкапсулирует функции MATLAB быстрого преобразования Фурье исходного сигнала и изображения графиков самого сигнала и частотного спектра сигнала в графическом окне MATLAB. Подкаталог `SpectraDemoJavaApp` содержит пример Java-приложения, которое использует функции созданного компонента.
3. Подкаталог `MatrixMathExample` содержит пример создания компонента Java для выполнения некоторых математических операций над матрицами. Подкаталог `MatrixMathDemoJavaApp` содержит пример консольного Java-приложения, которое использует функции созданного компонента.
4. Подкаталог `Ball_Sph` содержит пример создания компонента Java и Windows-приложения на Borland JBuilder, которое использует функции компонента для вычисления объема n -мерного шара единичного радиуса и площади $(n-1)$ -мерной единичной сферы.
5. Подкаталог `Mag` содержит пример создания компонента Java и Windows-приложения на Borland JBuilder, которое вычисляет магический квадрат заданного размера и отображает его в диалоговом окне приложения в виде таблицы.

Глава 4. Каталог `Gl_4_Excel_examples` содержит примеры создания компонентов для Excel и дополнений к Excel, которые используют функции созданных компонентов.

1. Подкаталог `matrix_xl` содержит пример создания компонента `matrix_xl`, который имеет набор матричных функций MATLAB для их выполнения на листе Excel.

2. Подкаталог SpectraExample содержит пример создания дополнения Excel для выполнения спектрального анализа с созданием формы для данного дополнения и средствами VBA

Глава 5. Каталог Gl_5_C#_Examples содержит примеры создания компонентов .NET Builder и приложений на C#, которые используют созданные Компилятором MATLAB компоненты..

1. Подкаталог MagicSquareExample содержит пример создания .NET компонента MagicDemoComp, вычисляющего магический квадрат. Приводится также пример создания на C# консольного приложения MagicSquareApp.cs, которое отображает на экране массив магического квадрата.
2. Подкаталог MatrixMathExample содержит пример создания .NET компонента MatrixMathDemoComp для выполнения некоторых математических операций над матрицами. Приводится также пример создания на C# консольного приложения MatrixMathDemoApp.exe, которое обращается к функциям компонента и отображает на экране результаты матричных операций.
3. Подкаталог Integration содержит пример создания .NET компонента Integr.dll для вычисления интегралов функций одной и двух переменных. Приводится также пример создания Windows-приложения на Visual Studio 2005 (C#), которое позволяет в текстовой строке ввести функцию, пределы интегрирования и вычислить ее интеграл.
4. Подкаталог Diff_Ury содержит пример создания .NET компонента Ode.dll для решения дифференциальных уравнений, разрешенных относительно старших производных. Приводится также пример создания Windows-приложения на Visual Studio 2005 (C#), которое позволяет в текстовой строке ввести функцию правой части дифференциального уравнения, пределы изменения аргумента t и начальные данные, найти решение и построить графики.
5. Подкаталог Files содержит пример создания .NET компонента fopensave.dll для чтения форматированных массивов данных, их математической обработке (вейвлет-анализ и удаление шума) и сохранения обработанных данных на диск и построения графиков. Приводится также пример создания Windows-приложения на Visual Studio 2005 (C#), которое использует функции компонента для чтения данных из файлов, их вейвлет-разложения, удаления шума, построения всех необходимых графиков и сохранения обработанных данных в файл.

Каталог MCR. Содержит файл для установки среды MCR исполнения компонентов MATLAB версии 7.6. Эта среда исполнения необходима для работы компонент и приложений всех примеров. Каталог содержит еще несколько полезных утилит. Подкаталог Math_Lib_C++ содержит математическую библиотеку (mglinstaller.exe) для MATLAB 6.5, которая необходима для работы приложений из второй главы.

Литература

- [Ан] *Ануфриев И.Е.* Самоучитель MATLAB 5.3/6.x. – СПб.: БХВ-Петербург, 2002. – 736 с.
- [Ба] *Баженова И.Ю.* Jbuilder. Программирование на Java. – М.: КУДИЦ-ОБРАЗ, 2000. – 448 с.
- [Га] *Гарнаев А.Ю.* Самоучитель VBA – СПб.: БХВ-Петербург, 2004. – 560 с.
- [ГП] *Говорухин В., Цибулин В.* Компьютер в математическом исследовании: Учеб. курс. – СПб.: Питер, 2001. – 624 с.
- [ДХ] *Довбуш Г.Ф., Хомоненко А.Д.* Visual C++ на примерах. – СПб.: БХВ-Петербург, 2007. – 528 с.
- [Д] *Дьяконов В.П.* MATLAB 6: Учеб. курс. – СПб.: Питер, 2001. – 582 с.
- [Ко] *Кондрашов В.Е., Королев С.Б.* MATLAB как система программирования научно-технических расчетов. – М.: Мир, 2002. – 350 с.
- [ККШ] *Кетков Ю.Л., Кетков А.Ю., Шульц М.М.* MATLAB 7. Программирование, численные методы. – СПб.: БХВ-Петербург, 2005. – 752 с.
- [Ка] *Кариев Ч.А.* Разработка Windows-приложений на основе Visual C# (+ CD-ROM). – Интернет-университет информационных технологий, Бином, 2007. – 768 с. <http://www.all-ebooks.com/main/books/programming/c/page/5/>
- [Кр] *Кривилев А.В.* Основы компьютерной математики с использованием системы MATLAB. – М.: Лекс-Книга, 2005. – 496 с.
- [Ла] *Лабар В.В.* Си Шарп: Создание приложений для Windows. – Минск: Харвест, 2003. – 384 с.
- [Ма] *Мартынов Н.Н.* Введение в MATLAB 6. – М.: Кудиц-Образ, 2002. – 352 с.
- [Ма] *Марченко А.Л.* Основы программирования на C# 2.0. – Интернет-университет информационных технологий, Бином, 2007. – 552 с. (<http://www.all-ebooks.com/main/books/programming/c/page/6/>).
- [НУР] *Нортроп Т., Уилдермьюс Ш., Райан Б.* Основы разработки приложений на платформе .NET Framework: Учеб. курс Microsoft / Пер. с англ. – СПб.: Питер, 2007. – 846 с.
- [Пон] *Пономарев В.А.* Самоучитель Jbuilder 6/7. – СПб.: БХВ-Петербург, 2003. – 304 с.
- [ППС] *Подкур М.Л., Подкур П.Н., Смоленцев Н.К.* Программирование в среде Borland C++ Builder с математическими библиотеками MATLAB C/C++. – М.: ДМК Пресс, 2006. – 496 с.
- [Пот] *Потемкин В.Г.* Вычисления в среде MATLAB. – М.: ДИАЛОГ-МИФИ, 2004. – 720 с.
- [См] *Смоленцев Н.К.* Основы теории вейвлетов. Вейвлеты в MATLAB. – М.: ДМК Пресс, 2005. – 304 с.
- [Тро] *Троелсен Э.* C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2004. – 796 с.
- [Фа] *Фарафонов В.В.* Программирование на языке C#: Учеб. курс. – СПб.: Питер, 2007. – 240 с.
- [Хо] *Холзнер С.* Visual C++: Учеб. курс. – СПб.: Питер, 2005. – 570 с.
- [ЧЖИ] *Чен К., Джиллин П., Ирвинг Ф.* MATLAB в математических исследованиях. – М.: Мир, 2001. – 346 с.
- [LePh1] MATLAB® C/C++ Book for MATLAB Compiler 4.5. – LePhan Publishing, <http://www.lephanpublishing.com/MATLABBookCplusplus.html>
- [LePh2] MATLAB® C# Book. – LePhan Publishing, <http://www.lephanpublishing.com/MatlabCsharp.html>
- [W] Материалы по использованию системы MATLAB на сайте <http://www.mathworks.com/access/helpdesk/help/helpdesk.html>.

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **order@abook.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: **www.aliants-kniga.ru**.

Оптовые закупки: тел. **(095) 258-91-94, 258-91-95**; электронный адрес **books@aliants-kniga.ru**.

Смоленцев Н. К.

Создание Windows-приложений с использованием математических процедур MATLAB

Главный редактор *Мовчан Д. А.*
dm@dmkpress.ru

Корректор ***

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Подписано в печать ***2008. Формат 70×100 ¹/₁₆.

Гарнитура «Петербург». Печать офсетная.

Усл. печ. л. ***. Тираж *000 экз.

№

Издательство ДМК Пресс. 123007, Москва, 1-й Силикатный пр-д, д. 14

Web-сайт издательства: www.dmk-press.ru

Internet-магазин: www.abook.ru

Электронный адрес издательства: books@dmk-press.ru