МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ,
МОЛОДЕЖИ И СПОРТА УКРАИНЫ
ОДЕССКИЙ НАЦИОНАЛЬНЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИНСТИТУТ КОМПЬЮТЕРНЫХ СИСТЕМ

Конспект лекций

по дисциплине “Пакеты прикладных программ”

для студентов специальностей 7.091401 и 8.191401

“Компьютеризированные системы управления и автоматика”

всех форм обучения

2012

ББК ХХ.ХХ
УДК ХХХ.ХХ:ХХХ.ХХ:ХХХ.ХХ

Пакеты прикладных программ: конспект лекций для студентов специальностей 7.091401 и 8.191401 “Компьютеризированные системы управления и автоматика” всех форм обучения / А.А. Фомин. – Одесса: ОНПУ, 2012. – 90 с.

Авторы: **Фомин Александр Алексеевич**

Рецензент: **В.Д.Павленко**

Ответственный
за выпуск: **А.А.Фомин**

Утверждено
Ученым советом
Института компьютерных систем

Протокол № 1 от ХХХХХХХ

Утверждено
на заседании кафедры
“Компьютеризированных систем управления”

Протокол № 1 от ХХХХХХХ

Лекция № 1

Введение. Предмет и задачи дисциплины. Общие сведение о ЭВМ и ПО.

1.1. Цель и задачи курса, его место в подготовке инженеров в области автоматики и управления в технических системах.

Дисциплина «Пакеты прикладных программ» обеспечивает базовую подготовку инженеров в изучении теории и принципов работы прикладных программ, используемых при проектировании, моделировании систем управления и автоматики. Она подготовливает слушателей к освоению профилирующих дисциплин специальности, рассматривающих теорию управления, элементы и устройства автоматики, оптимальные и адаптивные системы.

В результате изучения дисциплины студенты должны:

- знать принципы построения прикладных информационных систем;
- уметь использовать современные программные средства для обработки разнородной информации;
- уметь автоматизировать процесс решения прикладных задач с помощью встроенных языков программирования;
- иметь представление о современном состоянии и тенденциях развития рынка прикладного программного обеспечения.

1.2. Классификация прикладных программ. Обзор основных этапов развития ППП, современное состояние.

Перед рассмотрением признаков классификации прикладных программ рассмотрим несколько основных понятий и определений, которые помогут построить четкую и стройную классификационную картину.

Программное обеспечение (ПО) – совокупность программ и данных, предназначенных для решения определенного круга задач и хранящиеся на носителях ЭВМ.

Программа – последовательность формализованных инструкций, представляющих алгоритм решения некоторой задачи и предназначенная для исполнения устройством управления вычислительной машины.

Прикладное программное обеспечение – программное обеспечение, ориентированное на конечного пользователя и предназначенное для решения пользовательских задач.

Прикладное ПО состоит из:

- отдельных прикладных программ и пакетов прикладных программ, предназначенных для решения различных задач пользователей;
- автоматизированных систем, созданных на основе этих пакетов.

Классификация программного обеспечения

При классификации программного обеспечения *по назначению* в качестве критерия используют уровень представления ИС, на который ориентирована та или иная программа. Соответственно выделяют следующие классы ПО:

Системное ПО – решает задачи общего управления и поддержания работоспособности системы в целом. К этому классу относят операционные системы, менеджеры загрузки, драйверы устройств, программные кодеки, утилиты и программные средства защиты информации;

Инструментальное ПО – включает средства разработки (трансляторы, отладчики, интегрированные среды, различные SDK и т.п.) и системы управления базами данных (СУБД);

Прикладное ПО – предназначено для решения прикладных задач конечными пользователями. Прикладное ПО является самым обширным классом программ, в рамках которого возможна дальнейшая классификация, например по предметным областям. В этом случае группировочным признаком является класс задач, решаемых программой.

В зависимости от *степени интеграции* многочисленные прикладные программные средства можно классифицировать следующим образом:

- отдельные прикладные программы;
- библиотеки прикладных программ;
- пакеты прикладных программ;
- интегрированные программные системы.

Отдельная прикладная программа пишется, как правило, на некотором высокоуровневом языке программирования (Pascal, Basic и т.п.) и предназначается для решения конкретной прикладной задачи.

Библиотека представляет собой набор отдельных программ, каждая из которых решает некоторую прикладную задачу или выполняет определенные вспомогательные функции (управление памятью, обмен с внешними устройствами и т.п.). Условно их можно разделить на библиотеки общего назначения и специализированные библиотеки.

Пакет прикладных программ (ППП) – это комплекс взаимосвязанных программ, ориентированный на решение определенного класса задач.

Интегрированная программная система – это комплекс программ, элементами которого являются различные пакеты и библиотеки программ.

Понятие пакета прикладных программ

Итак, ППП – это комплекс взаимосвязанных программ для решения определенного класса задач из конкретной предметной области. На текущем этапе развития информационных технологий именно ППП являются наиболее востребованным видом прикладного ПО. Это связано с упомянутыми ранее особенностями ППП. Рассмотрим их подробней:

- ориентация на решение класса задач. Одной из главных особенностей является ориентация ППП не на отдельную задачу, а на некоторый класс задач, в том числе и специфичных, из определенной предметной области.
- наличие языковых средств. Другой особенностью ППП является наличие в

его составе специализированных языковых средств, позволяющих расширить число задач, решаемых пакетом или адаптировать пакет под конкретные нужды. Поддерживаемые языки могут быть использованы для формализации исходной задачи, описания алгоритма решения и начальных данных, организации доступа к внешним источникам данных, разработки программных модулей, описания модели предметной области, управления процессом решения в диалоговом режиме и других целей.

- единообразие работы с компонентами пакета. Еще одна особенность ППП состоит в наличии специальных системных средств, обеспечивавших унифицированную работу с компонентами. К их числу относятся специализированные банки данных, средства информационного обеспечения, средства взаимодействия пакета с операционной системой, типовой пользовательский интерфейс и т.п.

Обзор основных этапов развития, современное состояние.

Современный пакет является сложной программной системой, включающей специализированные системные и языковые средства. В относительно короткой истории развития вычислительных ППП можно выделить 4 основных поколения (класса) пакетов. Каждый из этих классов характеризуется определенными особенностями входящих в состав ППП компонентов – входных языков, предметного и системного обеспечения.

Первое поколение. В качестве входных языков ППП первого поколения использовались универсальные языки программирования (Фортран, Алгол-60 и т.п.) или языки управления заданиями соответствующих операционных систем. Проблемная ориентация входных языков достигалась за счет соответствующей мнемоники в идентификаторах. Составление заданий на таком языке практически не отличалось от написания программ на алгоритмическом языке.

Второе поколение. Разработка ППП второго поколения осуществлялась уже с применением специализированных входных языков на базе универсальных языков программирования. Проблемная ориентация таких языков достигалась не только за счет использования определенной мнемоники, но также применением соответствующих языковых конструкций, которые упрощали формулировку задачи и делали ее более наглядной. Транслятор с такого языка представлял собой препроцессор (чаще всего макропроцессор) к транслятору соответствующего алгоритмического языка.

Третье поколение. Третий этап развития ППП характеризуется появлением самостоятельных входных языков, ориентированных на пользователей-непрограммистов. Особое внимание в таких ППП уделяется системным компонентам, обеспечивающим простоту и удобство. Это достигается главным образом за счет специализации входных языков и включения в состав пакета средств автоматизированного планирования вычислений.

Четвертое поколение. Четвертый этап характеризуется созданием ППП, эксплуатируемых в интерактивном режиме работы. Основным преимуществом диалогового взаимодействия с ЭВМ является возможность активной обратной

связи с пользователем в процессе постановки задачи, ее решения и анализа полученных результатов.

Большое внимание в настоящее время уделяется проблеме создания «интеллектуальных ППП». Такой пакет позволяет конечному пользователю лишь сформулировать свою задачу в содержательных терминах, не указывая алгоритма ее решения. Синтез решения и сборка целевой программы производятся автоматически. Такой способ решения иногда называют концептуальным программированием, характерными особенностями которого является программирование в терминах предметной области использование ЭВМ уже на этапе постановки задач, автоматический синтез программ решения задачи, накопление знаний о решаемых задачах в базе знаний.

1.3. Эксплуатация вычислительной техники. Перспективы развития прикладного программного обеспечения.

Перспективы развития прикладного программного обеспечения

На сегодняшний день в качестве основных факторов, влияющих на функциональность ППП и сложность их разработки, можно отметить следующие:

- рост производительности персональных компьютеров;
- расширение классов решаемых задач;
- увеличение общего числа пользователей;
- значительное количество ранее созданного (наследованного) ПО;
- развитие Интернет и корпоративных сетей.

Разработка приложений с учетом этих факторов привела к появлению прикладных пакетов и интегрированных сред, которые по своим характеристикам выходят за рамки ППП четвертого поколения. К отличительным чертам ПО нового поколения следующие:

- интеграция компонентов прикладного пакета не только с приложениями пакета, но и с окружением;
- широкое использование отраслевых стандартов;
- использование инфраструктуры Интернет;
- платформонезависимость.

Ввиду повсеместного проникновения Интернета, можно говорить о том, что прикладное программное обеспечение будет переходить в разряд сервиса, то есть пользователи будут работать с необходимым программным обеспечением через Сеть. В основе технологий, обеспечивающих подобные возможности, лежит ряд совместных наработок ведущих производителей ПО и организаций по стандартизации. К ним относятся сервисно-ориентированная архитектура корпоративных приложений (веб-сервисы) и стандартизованные форматы документов.

Перспективным направлением в развитии ППП является использование унифицированных форматов документов на основе открытых стандартов. Открытый стандарт – общедоступная спецификация свободная от лицензионных

ограничений при использовании. Использование открытых форматов в ППП позволяет гарантировать возможность доступа к данным из любого совместимого приложения без оглядки на лицензионные права и технические спецификации.

1.4. Обзор программ математических вычислений, их возможности и особенности.

В области инженерного проектирования выделяют три основных раздела:

CAD – Computer Aided Design;

CAM – Computer Aided Manufacturing;

CAE – Computer Aided Engeneering.

Сегодня серьезное конструирование уже не может обойтись без САПР, производства и расчетов. А математические пакеты являются составной частью мира CAE-систем, но эта часть никак не может считаться второстепенной, поскольку некоторые задачи вообще невозможно решить без помощи компьютера.

Современные математические пакеты можно использовать и как обычный калькулятор, и как средства для упрощения выражений при решении каких-либо задач, и как генератор графики или даже звука!

В настоящее время практически все современные CAE-программы имеют встроенные функции символьных вычислений. Однако наиболее известными и приспособленными для математических символьных вычислений считаются Maple, MathCad, Mathematica и MatLab.

Отметим, что спектр задач, решаемых подобными системами, очень широк:

- проведение математических исследований, требующих вычислений и аналитических выкладок;
- разработка и анализ алгоритмов;
- математическое моделирование и компьютерный эксперимент;
- анализ и обработка данных;
- визуализация, научная и инженерная графика;
- разработка графических и расчетных приложений.

При этом отметим, что поскольку CAE-системы содержат операторы для базовых вычислений, то почти все алгоритмы, отсутствующие в стандартных функциях, можно реализовать посредством написания собственной программы.

1.5. Краткий обзор возможностей MathLab.

Система MatLab относится к среднему уровню продуктов, предназначенных для символьной математики, но рассчитана на широкое применение в сфере CAE (то есть сильна и в других областях). MatLab – одна из старейших, тщательно проработанных и проверенных временем систем автоматизации математических расчетов, построенная на расширенном представлении и применении матричных операций. Это нашло отражение и в самом названии системы – MATrix LABoratory, то есть матричная лаборатория. Однако синтаксис языка программирования системы продуман настолько тщательно, что данная ориентация почти не ощущается теми пользователями, которых не интересуют непосредственно матричные вычисления.

Несмотря на то что изначально MatLab предназначалась исключительно для вычислений, в процессе эволюции к ней была подключена библиотека Simulink, позволяющая построить логическую схему сложной системы управления из одних только стандартных блоков, не написав при этом ни строчки кода. После конструирования такой схемы можно детально проанализировать ее работу.

В системе MatLab также существуют широкие возможности для программирования. Ее библиотека C Math (компилятор MatLab) является объектной и содержит свыше 300 процедур обработки данных на языке C. Внутри пакета можно использовать как процедуры самой MatLab, так и стандартные процедуры языка C, что делает этот инструмент мощнейшим подспорьем при разработке приложений (используя компилятор C Math, можно встраивать любые процедуры MatLab в готовые приложения).

Лекция № 2

Основы работы в MATLAB

2.1. Рабочая среда.

Рабочая среда MATLAB состоит из следующих основных окон:

- Command Window (интерфейс командной строки)
- Workspace (перечень всех созданных переменных)
- Command history (история команд)
- Editor (редактор m-файлов)
- Figures (вывод графики)
- Current Folder
- Variable Editor (редактор переменных)
- Help

2.2. Простейшие вычисления

Простейшие вычисления можно выполнять в окне Command Window в диалоговом текстовом режиме, например:

```
>> 1 + 2
ans =
     3
```

Результат вычислений помещается в специальную переменную ans, а командная строка переходит в режим ввода следующей команды.

Если требуется продолжить работу с предыдущим выражением, например, (1+2) / 4, то можно воспользоваться уже имеющимся результатом, хранящимся в переменной ans:

```
>> ans / 4
ans =
     0.7500
```

2.3. Форматы вывода результатов вычислений

При вычислениях в MATLAB используется режим двойной точности. При выводе результатов, по умолчанию выдаются числа с 4 цифрами после десятичной точки в действительной форме.

Для изменнения фората вывода, необходимо в командной строке перед выводимой величиной выполнить команду format name, где name - имя формата. Для числовых данных name может принимать следующие значения:

- short - короткое представление в фиксированном формате (5 знаков);
- shorte - короткое представление в экспоненциальной форме (5 знаков мантиссы и 3 знака порядка);
- long - длинное представление в фиксированном формате (15 знаков);

- longe - длинное представление в экспоненциальной форме (15 знаков мантиссы и 3 знака порядка).

Например, значения вектора x=[4/3 1.2345e-6] в различных форматах представления будут иметь следующий вид:

- format short 1.3333 0.0000
- format short e 1.3333E+000 1.2345E-006
- format long 1.333333333333338 0.000001234500000
- format long e 1.333333333333338E+000 1.234500000000000E-006
- format bank 1.33 0.00

Задание формата сказывается только на форме вывода чисел. Вычисления же всегда происходят в режиме двойной точности.

2.4. Алфавит языка

В MATLAB, как и в других системах, используются все буквы латинского алфавита от A до Z и арабские цифры от 0 до 9. Как и в C++, большие и малые буквы это разные переменные и константы.

2.5. Специальные символы

Кроме букв латинского алфавита используются все специальные символы клавиатуры компьютера.

2.6. Типы данных

Структура типов данных MATLAB представлена схемой по рис. 2.1.

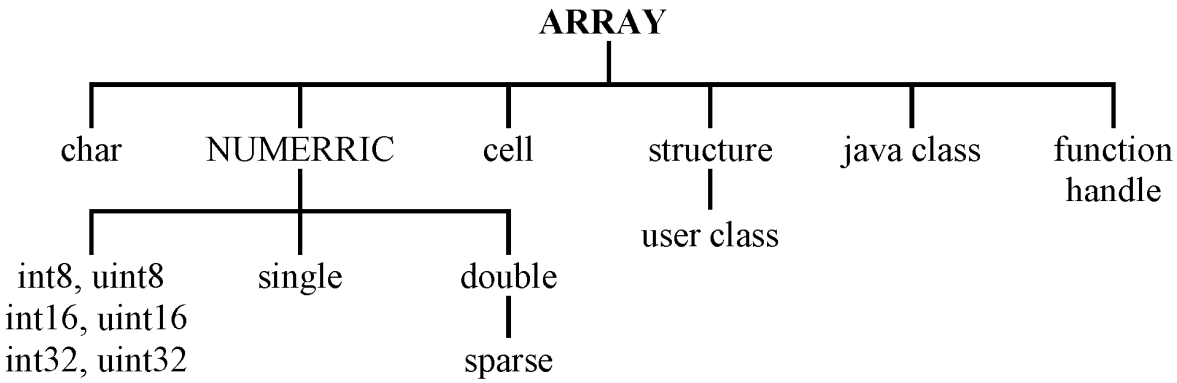


Рис. 2.1. Структура типов данных MATLAB

Типы данных *array* и *numerric* являются виртуальными ("кажущимися"), т. к. к ним нельзя отнести какие-либо переменные. Они служат для определения и комплектования некоторых типов данных. Таким образом, в MATLAB определены типы данных, представляющие собой многомерные массивы:

1. single - числовые массивы с числами одинарной точности;
2. double — числовые массивы с числами удвоенной точности;
3. char — строчные массивы с элементами - символами;
4. sparse - разреженные матрицы с элементами - числами двойной точности;

5. cell — массивы ячеек;

6. structure - массивы структур с полями;

7. function - handle - дескрипторы функций;

8. int8, ..., uint32 - массивы 8-, 16-, 32-разрядных целых чисел со знаками и без знаков.

2.7. Арифметические и логические операторы.

Таблица 2.1.

Список арифметических операторов.

Функция	Обозначение (синтаксис)
Сложение	+ (M1+M2)
Вычитание	- (M1-M2)
Матричное умножение	* (M1*M2)
Поэлементное умножение массивов	.* (M1.*M2)
Возведение матрицы в степень	^ (M ^ X)
Поэлементное возведение массива в степень	.^ (M .^ X)
Деление матриц слева направо	/ (Ml / M2)
Поэлементное деление массивов слева направо	./ (Ml ./ M2)
Деление матриц справа налево	\ (Ml \ M2)
Поэлементное деление массивов справа налево	.\ (Ml .\ M2)

Таблица 2.2.

Список операторов отношения.

Функция	Обозначение (синтаксис)
Равно	== (x = =y)
Не равно	~= (x~=y)
Меньше	< (x<y)
Больше	>(x>y)
Меньше или равно	<= (x<=y)
Больше или равно	>= (x>=y)

Данные операторы выполняют поэлементное сравнение векторов или матриц одинакового размера.

Таблица 2.3.

Список логических операторов.

Функция	Обозначение (синтаксис)
Логическое И	& ; and (and (a, b))
Логическое ИЛИ	| ; or (or (a, b))
Логическое НЕ	~ ; not (not (a, b))
Исключающее ИЛИ	xor (xor (a, b))
Верно, если все элементы вектора равны нулю	any (any (a))

Верно, если все элементы вектора не равны нулю	all (all (a))

Логические операторы служат для реализации поэлементных логических операций над элементами одинаковых по размеру массивов.

Полный список операторов можно получить, используя команду help ops.

2.8. Приоритеты операций

В математических выражениях операторы имеют определенный приоритет исполнения:

- приоритет логических операторов выше, чем арифметических,
- приоритет возведения в степень выше приоритетов умножения и деления,
- приоритет умножения и деления выше сложения и вычитания.

Для повышения приоритета операций нужно использовать круглые скобки. Степень вложения скобок не ограничивается.

2.9. Константы и системные переменные

Константа – это предварительно определенное числовое или символьное значение, представленное уникальным именем (идентификатором). Числа (например, 1, –2 и 1.23) являются безымянными числовыми константами.

Другие виды констант в MATLAB принято называть *системными переменными*, поскольку, с одной стороны, они задаются системой при ее загрузке, а с другой – могут переопределяться. Основные системные переменные, применяемые:

• i или j – мнимая единица (корень квадратный из –1);
• pi – число p = 3,1415926…;
• eps – погрешность операций над числами с плавающей точкой (2–52);
• realmin – наименьшее число с плавающей точкой (2–1022);
• realmax – наибольшее число с плавающей точкой (21023)
• inf – значение машинной бесконечности;
• ans – переменная, хранящая результат последней операции;
• NaN – указание на нечисловой характер данных (Not a Number).

Примеры применения системных переменных:

```
>> 2*pi
ans = 6.2832
>> eps
ans = 2.2204e-016
>> realmin
ans = 2.2251e-308
>> realmax
ans = 1.7977e+308
>> 1/0
Warning: Divide by zero.
ans = Inf
>> 0/0
Warning: Divide by zero.
ans = NaN
```

Как отмечалось, системные переменные могут переопределяться. Можно задать системной переменной eps иное значение, например eps=0.0001. Однако важно то, что их значения по умолчанию задаются сразу после загрузки системы.

Поэтому неопределенными, в отличие от обычных переменных, системные переменные не могут быть никогда.

Символьная константа – это цепочка символов, заключенных в апострофы, например:

```
'Hello my friend!'
'2+3'
```

Если в апострофы помещено математическое выражение, то оно не вычисляется и рассматривается просто как цепочка символов. Так что '2+3' не будет возвращать число 5.

2.10. Переменные и присваивание им значений

Переменные – это имеющие имена объекты, способные хранить некоторые данные. В зависимости от этих данных переменные когут быть числовыми или символьными, векторными или матричными.

В MATLAB можно задавать переменным определенные значения. Для этого используется операция присваивания, вводимая знаком равенства:

```
Имя_переменной = Выражение
```

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Так, если это выражение – вектор или матрица, то переменная будет векторной или матричной. Переменная, имеющая единственное значение, рассматривается как матрица размера 1×1.

Имя переменной (ее идентификатор) может содержать сколько угодно символов, но запоминается и идентифицируется только 31 начальный символ. Имя любой переменной не должно совпадать с именами других переменных, функций и процедур системы, то есть оно должно быть уникальным. 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., поскольку в этом случае правильная интерпретация выражений становится невозможной.

Могут использоваться и символьные переменные, причем символьные значения заключаются в апострофы, например s='Demo'.

Для вывода списка объявленных в Workspace переменных используется команда who. Команда whos используется для вывода подробной информации о переменных Workspace.

Сохранение в файл и чтение из файла переменных в Workspace можно выполнив команды save filename и load filename. Расширение файла можно не указывать, в этом случае файл получит расширение mat. Переменные в mat-файл записываются в двоичном формате.

2.11. Уничтожение определений переменных

В памяти компьютера переменные занимают определенное место, называемое рабочей областью (workspace). Для очистки рабочей области используется функция clear в разных формах, например:

• clear – уничтожение определений всех переменных;

• clear x – уничтожение определения переменной x;

• clear a, b, c – уничтожение определений нескольких переменных.

Уничтоженная (стертая в рабочей области) переменная становится неопределенной. Использовать неопределенные переменные нельзя, и такие попытки будут сопровождаться выдачей сообщений об ошибке. Приведем примеры задания и уничтожения переменных:

```
>> x=2*pi
x = 6.2832
>> V=[1 2 3 4 5]
V = 1 2 3 4 5
>> clear V
>> V
??? Undefined function or variable 'V'.
>> clear
>> x
??? Undefined function or variable 'x'.
>> M
??? Undefined function or variable 'M'.
```

2.12. Элементарные функции

Набор элементарных функций представим их описанием. В тригонометрических функциях углы измеряются в радианах.

Таблица 2.4.

Список элементарных функций.

Функция	Обозначение (синтаксис)
|x| - модуль	abs(x)
e^x - экспонента	exp(x)
In x - натуральный логарифм	log(x)
$\log_2$ x - логарифм по основанию 2	log2(x)
lg x - десятичный логарифм	log10(x)
2^x - 2 в степени x	pow(x)
$x^{1/2}$ - квадратный корень	sqrt(x)
sin x - синус	sin(x)

cos x - косинус	cos(x)
tg x - тангенс	tan(x)
ctg x - котангенс	cot(x)
округление	round(x)

Следует помнить, что все элементарные функции должны записываться в программах малыми буквами.

С полным списком элементарных функций можно ознакомиться, выполнив команду help elfun, а со списком специальных функций – с помощью команды help specfun.

Лекция № 3

Работа с массивами в MATLAB

3.1. Основные соглашения.

Массив – упорядоченная, пронумерованная совокупность однородных данных. Массивы различаются по числу размерностей (измерений): одномерные, двумерные, многомерные. Размером массива называют число элементов, вдоль каждого из изменений.

Доступ к элементам осуществляется при помощи индекса. В MATLAB нумерация элементов массивов начинается с единицы.

Вектор может быть записан в столбик (вектор-столбец) или в строку (вектор-строка).

3.2. Создание векторов и матриц

Если надо задать вектор из трех элементов, то их значения надо перечислить в квадратных скобках, разделяя пробелами.

```
>> V = [1 2 3]
V =
     1    2    3
```

В данном случае задан вектор-строка. Если разделить элементы точкой с запятой, то получим вектор-столбец.

```
>> V = [1; 2; 3]
V =
     1
     2
     3
```

Задание матрицы требует указания нескольких строк. Для разграничения строк используется символ ; (точка с запятой).

```
>> T = [1 2 3; 4 5 6; 7 8 9]
T =
     1    2    3
     4    5    6
     7    8    9
```

Узнать размерность и размер массива можно при помощи функции size.

Для указания отдельного элемента вектора или матрицы используются выражения вида V(i) или T(i, j). Например:

```
>> T(3,2)
```

```
ans =
     8
```

Если элементу T(i, j) нужно присвоить новое значение x, то используют оператор присваивания:

```
>>T(3,2)=x;
```

Выражение T(i) с одним индексом дает доступ к элементам матрицы, развернутым в один столбец. Такая матрица образуется из исходной, если подряд выписать ее столбцы. Например:

```
>> T(3)
ans =
     7
>>T(8)
ans =
     6
```

Символ : (двоеточие) – формирование подвекторов и подматриц из векторов и матриц.

• A(:,j) – это j-й столбец из A;
• A(i,:) – это i-я строка из A;
• A(:,:) – эквивалент двумерного массива (для матриц это аналогично A);
• A(j:k) – это A(j), A(j+1),…,A(k);
• A(:,j:k) – это A(:,j),A(:,j+1),…,A(:,k);
• A(:,:,k) – это k-ая страница трехмерного массива A;
• A(:) – записывает все элементы массива À в виде столбца.

Имеются также ряд особых функций для задания векторов и матриц.

Для создания единичной матрицы, которая имеет единичные диагональные элементы и нулевые все остальные, служит функция eye:

• eye(n) – возвращает единичную матрицу размера nxn;
• eye(m,n) или eye([m n]) – возвращает матрицу размера mxn с единицами по диагонали и нулями в остальных ячейках;
• eye(size(A)) – возвращает единичную матрицу того же размера, что и A.

Пример использования функции eye:

```
>> S=eye(4,5)
S =
     1  0  0  0  0
     0  1  0  0  0
     0  0  1  0  0
     0  0  0  1  0
```

Единичная матрица не определена для многомерных массивов. Так, функция y = eye([2,3,4]) при попытке ее вычисления приведет к выводу сообщения об ошибке.

Для создания матриц, все элементы которых – единицы, используется функция ones:

• ones(n) – возвращает матрицу размера nxn, все элементы которой – единицы. Если n – не скаляр, то появится сообщение об ошибке;

• ones(m,n) или ones([m n]) – возвращают матрицу размера mxn, состоящую из единиц;

• ones(d1,d2,d3,…) или ones([d1 d2 d3…]) – возвращают массив из единиц с размерностью d1xd2xd3x… ;

• ones(size(A)) – возвращает массив единиц той же размерности и размера, что и A. Матрица с единичными элементами, в отличие от единичной матрицы, в MATLAB определена и для многомерных массивов.

```
>> S=ones(3,4)
s =
     1  1  1  1
     1  1  1  1
     1  1  1  1
```

Для создания матриц, все элементы которых – нули, используется функция zeros:

• zeros(n) – возвращает матрицу размера nxn, содержащую нули. Если n – не скаляр, то появится сообщение об ошибке;

• zeros(m,n) или zeros([m n]) – возвращают матрицу размера mxn, состоящую из нулей;

• zeros(d1,d2,d3,…) или zeros([d1 d2 d3…]) – возвращают массив из нулей размера d1xd2xd3x… ;

• zeros(size(A)) – возвращает массив нулей того же размера и размерности, что и A.

```
>> S=zeros(3,2)
S =
     0  0
     0  0
     0  0
```

Аналогично используются функции true(m,n) и false(m,n) для создания массивов размера mxn, содержащих соответственно логические 1 и 0.

Для создания линейного массива равноотстоящих точек используется функция linspace:

• linspace(a,b) – возвращает линейный массив из 100 точек, равномерно распределенных между a и b;

• linspace(a,b,n) – генерирует n точек, равномерно распределенных в нтервале от a до b.

```
>> M=linspace(4,20,14)
M =
```

```
Columns 1 through 7
4.0000 5.2308 6.4615 7.6923 8.9231 10.1538 11.3846
Columns 8 through 14
12.6154 13.8462 15.0769 16.3077 17.5385 18.7692 20.0000
```

Еще один вариант создания линейного массива равноотстоящих точек: a:b – создаст массив целых чисел начиная с a и заканчивая b. Для изменения шага значений массива можно использовать следующую апись: a:Δ:b, где Δ – шаг значений массива. Например:

```
>> x = 0 : 10;
>> y = -3 : 0.2 : 3
```

Для создания массива со случайными элементами используются следующие функции.

randperm(n) – возвращает *случайные перестановки* целых чисел 1:n в векторе_строке.

```
>> randperm(6)
ans =
     2 4 3 6 5 1
```

rand генерирует массивы случайных чисел, значения элементов которых *равномерно* распределены в промежутке (0,1):

• rand(n) – возвращает матрицу размера nxn. Если n – не скаляр, то появится сообщение об ошибке;

• rand(m,n) или rand([m n]) – возвращают матрицу размера mxn;

• rand(m,n,p,…) или rand([m n p…]) – возвращает многомерный массив;

• rand(size(A)) – возвращает массив того же размера и размерности, что и A, с элементами, распределенными по равномерному закону;

• rand (без аргументов) – возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет равномерный закон распределения;

```
>> Y=rand(4,3)
Y =
     0.9501 0.8913 0.8214
     0.2311 0.7621 0.4447
     0.6068 0.4565 0.6154
     0.4860 0.0185 0.7919
```

randn генерирует массив со случайными элементами, распределенными по нормальному закону с нулевым математическим ожиданием и среднеквадратическим отклонением, равным 1:

• randn(n) – возвращает матрицу размера n□n. Если n – не скаляр, то появится сообщение об ошибке;

• randn(m,n) или randn([m n]) – возвращают матрицу размера mxn;

• randn(m,n,p,…) или randn([m n p…]) – возвращает массив с элементами, значения которых распределены по нормальному закону;
• randn(size(A)) – возвращает массив того же размера, что и A, с элементами, распределенными по нормальному закону;
• randn (без аргументов) – возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет нормальное распределение;

```
>> Y=randn(4,3)
Y =
     -0.4326 -1.1465 0.3273
     -1.6656 1.1909 0.1746
     0.1253 1.1892 -0.1867
     0.2877 -0.0376 0.7258
```

3.3. Операции над векторами.

3.3.1. Конкатенация матриц

Конкатенацией называют объединение массивов или матриц. Это реализует следующая функция:

• C = cat(dim,A,B) – объединяет массивы A и B в соответствии со спецификацией размерности dim и возвращает объединенный массив; dim=1 – горизонтальная конкатенация, dim=2 – вертикальная, dim=3 – многомерный массив размерности 3 и т. д.;

• C = cat(dim,A1,A2,A3,A4,…) объединяет все входные массивы (A1, A2, A3, A4 и т. д.) в соответствии со спецификацией размерности dim и возвращает объединенный массив;

```
>> A =[2,4;3,5];B=[8,7;9,0];C=cat(1,A,B)
     C =
     2 4
     3 5
     8 7
     9 0
```

3.3.2. Удаление строк и столбцов

Для удаления отдельных столбцов и строк матриц используются пустые квадратные скобки – [ ]. Проделаем это с матрицей M.

```
     T(:,2)=[ ]
     T =
          1    3
          4    6
          7    9
```

3.3.4. Транспонирование матрицы

Транспонирование вектора или матрицы производится при помощи ‘

```
     T'
     ans =
```

```
     1     4     7
     3     6     9
```

3.3.4. Определитель матрицы

Для нахождения *определителя* (*детерминанта*) матриц в MATLAB имеется функция:

• det(X) – возвращает определитель квадратной матрицы X. Если X содержит только целые элементы, то результат – тоже целое число.

```
>> A=[2,3,6;1,8,4;3,6,7]
A =
     2 3 6
     1 8 4
     3 6 7
>> det(A)
ans = -29
```

3.3.5. Обращение матриц

Обратной называют матрицу, получаемую в результате деления единичной матрицы E на исходную матрицу X. Таким образом, X–1=E/X. Следующая функция обеспечивает реализацию данной операции:

• inv(X) возвращает матрицу, обратную квадратной матрице X.

Предупреждающее сообщение выдается, если X плохо масштабирована или близка к вырожденной.

```
>> inv(rand(4,4))
ans =
     2.2631 -2.3495 -0.4696 -0.6631
     -0.7620 1.2122 1.7041 -1.2146
     -2.0408 1.4228 1.5538 1.3730
     1.3075 -0.0183 -2.5483 0.6344
```

3.4. Статистические функции

Функция sum предназначена для суммирования элементов вектора.

Функция mean предназначена для вычисления среднего арифметического элементов вектора.

Для нахождения минимального и максимального элемента вектора используются функции min и max соответственно.

3.5. Решение систем линейных уравнений

Пусть задана система n линейных алгебраических уравнений с n неизвестными. Система уравнений в матричной форме представляется следующим образом:

$$AX = B \qquad (3.1.)$$

где A - квадратная матрица коэффициентов, размером n x n строк и столбцов;

X - вектор-столбец неизвестных;

B - вектор-столбец правых частей.
Решение в матричном виде представляется как
X = B/A

Пример.
Решить систему 4-х линейных уравнений:
1.116x1 +0.1397x2 +0.1254x3 +0.1490x4 =1.5471;
0.1582x1 +0.1768x2 +1.1675x3 +0.1871x4 =1.6471;
0.1968x1 +1.2168x2 +0.2071x3 +0.2271x4 =1.7471;
0.2368x1 +0.2568x2 +0.2471x3 +1.2671x4 =1.8471.

```
a = [1.1161 0.1397 0.1254 0.1490;
0.1582 0.1768 1.1675 0.1871 ;
0.1968 1.2168 0.2071 0.2271 ;
0.2368 0.2568 0.2471 1.2671];
b = [1.5471 ; 1.6471 ; 1.7471 ; 1.8471];
X = a\Ь
X =
     1.0406
     0.9351
     0.9870
     0.8813
```

Лекция № 4

Графика MATLAB

4.1. Графики функций в декартовой системе координат

Команда plot служит для построения графиков функций в декартовой системе координат. Эта команда имеет ряд параметров, рассматриваемых ниже.

• plot(X,Y) строит график функции $y(x)$, координаты точек (x,y) которой берутся из векторов одинакового размера Y и X. Если X или Y – матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы.

```
>> x=[0 1 2 3 4 5]; Y=[sin(x);cos(x)]; plot(x,Y)
```

• plot(Y) строит график $y(i)$, где значения y берутся из вектора Y, а i представляет собой индекс соответствующего элемента. Если Y содержит комплексные элементы, то выполняется команда plot(real(Y), imag(Y)).

Во всех других случаях мнимая часть данных игнорируется.

```
>> x=-2*pi:0.02*pi:2*pi; y=sin(x)+i*cos(3*x); plot(y)
```

• plot(X,Y,S) аналогична команде plot(X,Y), но тип линии графика можно задавать с помощью строковой константы S.

Значениями константы S приведены в Таблице 4.1.

Таблица 4.1.

Цвет линии		Тип точки		Тип линии	
y	желтый	.	точка	-	сплошная
m	фиолетовый	o	окружность	:	двойной пунктир
c	голубой	x	крест	-.	штрихпунктир
r	красный	+	плюс	--	штриховая
g	зеленый	*	звездочка		
b	синий	s	квадрат		
w	белый	d	ромб		
k	черный	v	треугольник (вниз)		
		^	треугольник (вверх)		
		<	треугольник (влево)		
		>	треугольник (вправо)		
		p	пятиугольник		
		h	шестиугольник		

Таким образом, с помощью строковой константы S можно изменять цвет линии, представлять узловые точки различными отметками и менять тип линии графика.

• plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,…) – эта команда строит на одном графике ряд линий, представленных данными вида (Xi,Yi,Si), где Xi и Yi – векторы

или матрицы, а S*i* – строки. С помощью такой конструкции возможно построение, например, графика функции линией, цвет которой отличается от цвета узловых точек.

При отсутствии указания на цвет линий и точек он выбирается автоматически из таблицы цветов (белый исключается). Если линий больше шести, то выбор цветов повторяется. Для монохромных систем линии выделяются стилем.

```
% Программа построения графиков трех функций
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'--ok')
```

4.2. Диаграммы и гистограммы.

Команды для построения столбцовых диаграмм:

• bar(x,Y) строит столбцовый график элементов вектора Y (или группы столбцов для матрицы Y) со спецификацией положения столбцов, заданной значениями элементов вектора x, которые должны идти в монотонно возрастающем порядке;

• bar(Y) строит график значений элементов матрицы Y так же, как указано выше, но фактически для построения графика используется вектор x=1:m;

• bar(x,Y,WIDTH) или BAR(Y,WIDTH) – команда аналогична ранее рассмотренным, но со спецификацией ширины столбцов (при WIDTH > 1 столбцы перекрываются). По умолчанию задано WIDTH = 0.8.

Возможно применение этих команд и в следующем виде: bar(…,'Спецификация') для задания спецификации графиков, например типа линий, цвета и т. д., по аналогии с командой plot. Спецификация ‘stacked’ задает рисование всех *n* столбцов друг на друге.

Пример построения столбцовой диаграммы матрицы размером 12x3 приводится ниже:

```
>> bar(rand(12,3),'stacked')
```

Помимо команды bar(…), существует аналогичная ей по синтаксису команда barh(…), которая строит столбцовые диаграммы с горизонтальным расположением столбцов.

Классическая *гистограмма* характеризует в виде столбцовой диаграммы числа попаданий значений элементов вектора Y в M интервалов.

Для получения данных для гистограммы служит функция hist:

• N=hist(Y) возвращает вектор чисел попаданий для 10 интервалов, выбираемых автоматически. Если Y – матрица, то выдается массив данных о числе попаданий для каждого из ее столбцов;

• N=hist(Y,M) аналогична вышерассмотренной, но используется M интервалов (M – скаляр);

• N=hist(Y,X) возвращает числа попаданий элементов вектора Y в интервалы, центры которых заданы элементами вектора X;

• [N,X]=HIST(…)возвращает числа попаданий в интервалы и данные о центрах интервалов.

```
>> x=-3:0.2:3; y=randn(1000,1);
>> hist(y,x); h=hist(y,x)
h =
Columns 1 through 12
0 0 3 7 8 9 11 23 33 43 57 55
Columns 13 through 24
70 62 83 87 93 68 70 65 41 35 27 21
Columns 25 through 31
12 5 6 3 2 1 0
```

Команды для созданиякруговых диаграмм:

• pie(X) строит круговую диаграмму по данным нормализованного вектора X/SUM(X). SUM(X) – сумма элементов вектора. Если SUM(X)<=1.0, то значения в X непосредственно определяют площадь секторов;

• pie(X,EXPLODE) строит круговую диаграмму, у которой отрыв секторов от центра задается вектором EXPLODE, который должен иметь тот же размер, что и вектор данных X.

Следующий пример строит цветную круговую диаграмму с пятью секторами, причем последний сектор отделен от остальных:

```
>> X=[1 2 3 4 5]; pie(X,[0 0 0 0 2])
```

4.3. Создание массивов данных для трехмерной графики

Поверхности как объекты трехмерной графики обычно описываются функцией двух переменных z(x,y). Специфика построения трехмерных графиков требует не просто задания ряда значений x и y, но и двумерных массивов – матриц. Для создания таких массивов служит функция meshgrid:

• [X,Y] = meshgrid(x,y) преобразует область, заданную векторами x и y, в массивы X и Y, которые могут быть использованы для вычисления функции двух переменных и построения трехмерных графиков. Строки выходного массива X являются копиями вектора x; а столбцы Y – копиями вектора y;

• [X,Y] = meshgrid(x) аналогична [X,Y] = meshgrid(x,x);

```
>> [X,Y] = meshgrid(1:4,13:17)
X =
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
Y =
13 13 13 13
14 14 14 14
15 15 15 15
16 16 16 16
```

```
17 17 17 17
```

4.4. Трёхмерная графика

Команда plot3(…) является аналогом команды plot(…), но относится к функции двух переменных z(x,y). Она строит аксонометрическое изображение трехмерных поверхностей и представлена следующими формами:

• plot3(x,y,z) строит массив точек, представленных векторами x, y и z, соединяя их отрезками прямых. Эта команда имеет ограниченное применение;

• plot3(X,Y,Z), где X, Y и Z – три матрицы одинакового размера, строит точки с координатами X(i,:), Y(i,:) и Z(i,:) и соединяет их отрезками прямых.

Ниже дан пример программы построения трехмерной поверхности, описываемой функцией z(x,y)=x^2+y^2:

```
% Программа построения поверхности линиями
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z)
```

• plot3(X,Y,Z,S) обеспечивает построения, аналогичные рассмотренным ранее, но со спецификацией стиля линий и точек, соответствующей спецификации команды plot:

```
% Программа построения поверхности кружками
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,’o’)
```

• plot3(x1,y1,z1,s1,x2,y2,z2,s2,x3,y3,z3,s3,…) строит на одном рисунке графики нескольких функций z1(x1,y1), z2(x2,y2) и т. д. со спецификацией линий и маркеров каждой из них.

```
% Программа построения сетчатого графика функции
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,’-k’,Y,X,Z,'-k')
```

В данном случае строятся два графика одной и той же функции с взаимно перпендикулярными образующими линиями. Поэтому график имеет вид сетки без окраски ее ячеек (напоминает проволочный каркас фигуры).

Наиболее представительными и наглядными являются *сетчатые графики* поверхностей с заданной или функциональной окраской.

• mesh(X,Y,Z,C) выводит в графическое окно сетчатую поверхность Z(X,Y) с цветами узлов поверхности, заданными массивом C.

• mesh(X,Y,Z) – аналог предшествующей команды при C=Z. В данном случае используется функциональная окраска, при которой цвет задается высотой поверхности.

Возможны также формы команды mesh(x,y,Z), mesh(x,y,Z,C), mesh(Z) и mesh(Z,C). Функция mesh возвращает дескриптор для объекта класса surface. Ниже приводится пример программы с применением команды mesh:

```
% Программа построения графика поверхности с окраской
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
mesh(X,Y,Z)
```

MATLAB имеет несколько графических функций, возвращающих *матричный образ поверхностей*. Например, функция peaks(N) возвращает матричный образ поверхности с рядом пиков и впадин. Такие функции удобно использовать для проверки работы графических команд трехмерной графики.

```
>> z=peaks(25); mesh(z);
```

Особенно наглядное представление о поверхностях дают сетчатые графики, использующие *функциональную закраску* ячеек. Например, цвет окраски поверхности $z(x,y)$ может быть поставлен в соответствии с высотой z поверхности с выбором для малых высот темных тонов, а для больших – светлых.

Для построения таких поверхностей используются команды класса surf(…):

• surf(X,Y,Z,C) строит цветную параметрическую поверхность по данным матриц X, Y и Z с цветом, задаваемым массивом C;

• surf(X,Y,Z) аналогична предшествующей команде, где C=Z, так что цвет задается высотой той или иной ячейки поверхности;

• surf(x,y,Z) и surf(x,y,Z,C) с двумя векторными аргументами x и y – векторы x и y заменяют два первых матричных аргумента и должны иметь

длины length(x)=n и length(y)=m, где [m,n] = size(Z). В этом случае вершины областей поверхности представлены тройками координат (x(j), y(i), Z(i,j)). Заметим, что x соответствует столбцам Z, а y соответствует строкам;

• surf(Z) и surf(Z,C) используют x = 1:n и y = 1:m. В этом случае высота Z – однозначно определенная функция, заданная геометрически прямоугольной сеткой;

• h=surf(…) строит поверхность и возвращает дескриптор объекта класса surface.

```
% Программа построения графика параболоида с окраской
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
surf(X,Y,Z)
```

Графики поверхностей, в которых имитируется освещение от точечного источника света, расположенного в заданном месте координатной системыстроятся при помощи команды surfl:

• surfl(…) аналогична команде surf(…), но строит график поверхности с подсветкой от источника света;

• surfl(Z,S) или surfl(X,Y,Z,S) строит графики поверхности с подсветкой от источника света, положение которого в системе декартовых координат задается вектором S=[Sx,Sy,Sz];

• surfl(…,'light') позволяет при построении задать цвет подсветки с помощью объекта Light;

• surfl(…,'cdata') при построении имитирует эффект отражения;

• surfl(X,Y,Z,S,K) задает построение поверхности с параметрами, заданными вектором K=[ka,kd,ks,spread], где ka – коэффициент фоновой подсветки, kd – коэффициент диффузного отражения, ks – коэффициент зеркального отражения и spread – коэффициент глянцевитости;

• H=surfl(…)строит поверхность и возвращает дескрипторы поверхности и источников света.

```
% Программа построения поверхности с имитацией
% ее освещенности от точечного источника
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surfl(X,Y,Z);
```

Средства управления подсветкой и обзором фигур:

• diffuse – задание эффекта диффузионного рассеяния;

• lighting – управление подсветкой;

• material – имитация свойств рассеивания света различными материалами;

• specular – задание эффекта зеркального отражения.

Следующие три команды позволяют управлять углами просмотра, под которыми рассматривается видимая в графическом окне фигура:

• view – задание положения точки просмотра;

• viewmtx – задание и вычисление матрицы вращения;

• rotate3d – задание поворота трехмерной фигуры.

4.5. Цветные объемные круговые диаграммы

Иногда используются объемные круговые диаграммы. Для их построения служит команда pie3:

• pie3(…) аналогична команде pie(…), но дает построение объемных секторов.

В приведенном ниже примере строится объемная диаграмма с отделением двух секторов, показанная на рис. 6.52.

```
>> X=[1 2 3 4 5]; pie3(X,[0 0 1 0 1])
```

Лекция № 5

Редактирование графиков MATLAB

5.1. Текстовое оформление графиков

Установка титульной надписи. Для установки над графиком титульной надписи используется следующая команда:

• title('string') установка на двумерных и трехмерных графиках титульной надписи, заданной строковой константой 'string'.

Установка осевых надписей. Для установки надписей возле осей x, y и z используются следующие команды:

• xlabel('String')
• ylabel('String')
• zlabel('String')

Соответствующая надпись задается символьной константой или переменной 'String'.

```
% Программа построения графика поверхности
% с текстовым оформлением
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surfl(X,Y,Z); colorbar
colormap(gray); shading interp
xlabel('Axis X'); ylabel('Axis Y')
zlabel('Axis Z'); title('Surface graphic')
```

Ввод текста в любое место графика. Часто возникает необходимость добавления текста в определенное место графика, например для обозначения той или иной кривой графика. Для этого используется команда text:

• text(X,Y,'string') добавляет в двумерный график текст, заданный строковой константой 'string', так что начало текста расположено в точке с координатами (X,Y). Если X и Y заданы как одномерные массивы, то надпись помещается во все позиции [x(i),y(i)];

• text(X,Y,Z,'string') добавляет в трехмерный график текст, заданный строковой константой 'string', так что начало текста расположено в позиции, заданной координатами X, Y и Z.

```
>> x=-10:0.1:10; plot(x,sin(x).^3)
>> text(-4,0.7,'Graphic sin(x)^3')
```

Позиционирование текста с помощью мыши. Очень удобный способ ввода текста предоставляет команда gtext:

• gtext('string') задает выводимый на график текст в виде строковой константы 'string' и выводит на график перемещаемый мышью маркер в виде крестика. Установив маркер в нужное место, достаточно щелкнуть любой кнопкой мыши для вывода текста;

• gtext(C) позволяет аналогичным образом разместить многострочную надпись из массива строковых переменных C.

```
>> x=-15:0.1:15; plot(x, sin(x).^3)
>> gtext('Function sin(x)^3')
```

Установив перекрестие в нужное место графика, достаточно нажать любую клавишу или любую кнопку мыши, и на этом месте появится надпись.

Высокая точность позиционирования надписи и быстрота процесса делают данный способ нанесения надписей на графики одним из наиболее удобных.

5.2. Форматирование графиков

Вывод пояснений и легенды. Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется легендой. Для создания легенды используются различные варианты команды legend:

• legend(string1,string2,string3,…) добавляет к текущему графику легенду в виде строк, указанных в списке параметров;

• legend(H,string1,string2,string3,…) помещает легенду на график, содержащий объекты с дескрипторами H, используя заданные строки как метки для соответствующих дескрипторов;

• legend(AX,…) помещает легенду в осях (объект класса axes) с дескриптором AX;

• legend(M) размещает легенду, используя данные из строковой матрицы M;

• legend OFF устраняет ранее выведенную легенду;

• legend перерисовывает текущую легенду, если таковая имеется;

• legend(legendhandle) перерисовывает легенду, указанную дескриптором legendhandle;

• legend(…,Pos) помещает легенду в точно определенное место, специфицированное параметром Pos:

– Pos=0 – лучшее место, выбираемое автоматически;

– Pos=1 – верхний правый угол;

– Pos=2 – верхний левый угол;

– Pos=3 – нижний левый угол;

– Pos=4 – нижний правый угол;

– Pos=-1 – справа от графика.

Чтобы перенести легенду, установите на нее курсор, нажмите левую кнопку мыши и перетащите легенду в необходимую позицию.

Команда legend может использоваться с двумерной и трехмерной графикой и со специальной графикой – столбцовыми и круговыми диаграммами и т. д. Двойным щелчком можно вывести легенду на редактирование.

```
% Программа построения графика трех функций
% с выводом их обозначений – легендой
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'—ok')
legend('Function 1','Function 2','Function 3');
```

Незначительная модификация команды legend (применение дополнительного параметра -1) позволяет построить график трех функций с легендой вне поля графика. Это иллюстрирует следующая программа:

```
% Программа построения графика трех функций
% с выводом легенды вне поля графика
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'—ok')
legend('Function 1','Function 2','Function 3',-1);
```

В данном случае недостатком можно считать сокращение полезной площади самого графика.

Управление свойствами осей графиков

Обычно графики выводятся в режиме автоматического масштабирования. Следующие команды класса axis меняют эту ситуацию:

• axis([XMIN XMAX YMIN YMAX]) – установка диапазонов координат по осям x и y для текущего двумерного графика;

• axis([XMIN XMAX YMIN YMAX ZMIN ZMAX]) – установка диапазонов координат по осям x, y и z текущего трехмерного графика;

• axis auto – установка параметров осей по умолчанию;

• axis manual «замораживает» масштабирование в текущем состоянии, чтобы при использовании команды hold on следующие графики использовали те же параметры осей;

• axis tight устанавливает диапазоны координат по осям в соответствии с диапазонами изменения данных;

• axis ij задает «матричную» прямоугольную систему координат с началом координат в левом верхнем углу, ось i – вертикальная, размечаемая сверху вниз, ось j – горизонтальная и размечается слева направо;

• axis xy устанавливает декартову систему координат с горизонтальной осью x, размечаемой слева направо, и вертикальной осью y, размечаемой снизу вверх (начало координат размещается в нижнем левом углу);

• axis equal включает масштаб с одинаковым расстоянием между метками по осям x, y и z;

• axis image устанавливает масштаб, при котором пиксели изображения становятся квадратами;

• axis square устанавливает текущие оси в виде квадрата (или куба в трехмерном случае) с одинаковым расстоянием между метками и одинаковой длиной осей;

• axis normal восстанавливает масштаб, отменяя установки axis equal и axis square;

• axis vis3d «замораживает» пропорции осей для возможности поворота трехмерных объектов;

• axis off убирает с осей их обозначения и маркеры;

• axis on восстанавливает ранее введенные обозначения осей и маркеры;

• V=axis возвращает вектор-строку, содержащую коэффициенты масштабирования для текущего графика. Если текущий график двумерный, то вектор имеет 4 компонента, если трехмерный – 6 компонентов.

Следующий пример иллюстрирует применение команды axis при построении двумерного графика функции одной переменной:

```
>> x=-5:0.1:5; plot(x,sin(x)); axis([-10 10 -1.5 1.5])
```

Теперь масштабы осей заданы командой axis, а не диапазоном изменения значений x и y.

Включение и выключение сетки. Команды grid позволяют задавать построение сетки или отменять это построение:

• grid on добавляет сетку к текущему графику;

• grid off отключает сетку;

• grid последовательно производит включение и отключение сетки.

Команды grid устанавливают свойства объектов XGrid, Ygrid и Zgrid для текущих осей.

```
>> x=-5:0.1:5; plot(x,sin(x));
>> axis([-10 10 -1.5 1.5]); grid on
```

Наложение графиков друг на друга. Во многих случаях желательно построение многих наложенных друг на друга графиков в одном и том же окне. Для этого служит команда продолжения графических построений hold. Она используется в следующих формах:

• hold on обеспечивает продолжение вывода графиков в текущее окно, что позволяет добавлять последующие графики к уже существующим;

• hold off отменяет режим продолжения графических построений;

• hold работает как переключатель, последовательно включая режим продолжения графических построений и отменяя его.

Команда hold on устанавливает значение add для свойства NextPlot объектов figure и axes, a hold off устанавливает для этого свойства значение replace. Рекомендуется также ознакомиться с командами ishold, newplot, figure и axes.

Приведенный ниже пример показывает, как с помощью команды hold on на график синусоиды накладываются еще три графика параметрически заданных функций:

```
>> x=-5:0.1:5; plot(x,sin(x)); hold on
>> plot(sin(x),cos(x)); plot(2*sin(x),cos(x))
>> plot(4*sin(x),cos(x)); hold off
```

Разбиение графического окна. Для расположения в одном окне нескольких координатных осей с различными графиками без наложения их друг на друга используются команды subplot, применяемые перед построением графиков:

• subplot создает новые объекты класса axes (подокна);

• subplot(m,n,p) или subplot(mnp) разбивает графическое окно на mxn подокон, при этом m – число подокон по горизонтали, n – число подокон по вертикали, а p – номер подокна, в которое будет выводиться текущий график (подокна отсчитываются последовательно по строкам);

• subplot(H), где H – дескриптор для объекта axes, дает альтернативный способ задания подокна для текущего графика;

• subplot('position',[left bottom width height]) создает подокно с заданными нормализованными координатами (в пределах от 0.0 до 1.0);

• subplot(111) и clf reset удаляют все подокна и возвращают графическое окно в обычное состояние.

Следующая программа иллюстрирует применение команды subplot:

```
% Программа построения четырех графиков в разных подокнах
x=-5:0.1:5;
subplot(2,2,1),plot(x,sin(x))
subplot(2,2,2),plot(sin(5*x),cos(2*x+0.2))
subplot(2,2,3),contour(peaks)
subplot(2,2,4),surf(peaks)
```

Для всех графиков возможна индивидуальная установка дополнительных объектов, например титульных надписей, надписей по осям и т. д.

Изменение масштаба графика. Для изменения масштаба двумерных графиков используются команды класса zoom:

• zoom переключает состояние режима интерактивного изменения масштаба для текущего графика;

• zoom(FACTOR) устанавливает масштаб в соответствии с коэффициентом FACTOR;

• zoom on включает режим интерактивного изменения масштаба для текущего графика;

• zoom off выключает режим интерактивного изменения масштаба для текущего графика;

• zoom out обеспечивает полный просмотр, то есть устанавливает стандартный масштаб графика;

• zoom xon или zoom yon включает режим изменения масштаба только по оси x или по оси y;

• zoom reset запоминает текущий масштаб в качестве масштаба по умолчанию для данного графика;

• zoom(FIG,OPTION) применяется к графику, заданному дескриптором FIG, при этом OPTION может быть любым из перечисленных выше аргументов.

Команда zoom позволяет управлять масштабированием графика с помощью мыши. Для этого надо подвести курсор мыши к интересующей вас области рисунка. Если команда zoom включена (on), то нажатие левой кнопки увеличивает масштаб вдвое, а правой – уменьшает вдвое. При нажатой левой кнопке мыши можно выделить пунктирным черным прямоугольником нужный участок графика – при отпускании кнопки он появится в увеличенном виде и в том масштабе, который соответствует выделяющему прямоугольнику.

```
>> x=-5:0.01:5; plot(x,sin(x.^5)./(x.^5+eps)); zoom on
```

Команда zoom, таким образом, выполняет функцию «лупы», позволяющей наблюдать в увеличенном виде отдельные фрагменты сложных графиков.

5.3. Цветовая окраска графиков

Установка палитры цветов. Палитра цветов RGB задается матрицей MAP из трех столбцов, определяющих значения интенсивности красного (red), зеленого (green) и синего (blue) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например, [0 0 0] задает черный цвет, [1 1 1] – белый цвет, [0 0 1] – синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату RGB.

Для установки палитры цветов служит команда colormap, записываемая в следующих формах:

• colormap('default') устанавливает палитру по умолчанию, при которой распределение цветов соответствует радуге;

• colormap(MAP) устанавливает палитру RGB, заданную матрицей MAP;

• C=colormap – функция возвращает матрицу текущей палитры цветов C. m_файл с именем **colormap** устанавливает свойства цветов для текущего графика.

Команда help graph3d наряду с прочим выводит полный список характерных палитр, используемых графической системой MATLAB:

• hsv – цвета радуги;

• hot – чередование черного, красного, желтого и белого цветов;

• gray – линейная палитра в оттенках серого цвета;

• bone – серые цвета с оттенком синего;

• copper – линейная палитра с оттенками меди;

• pink – розовые цвета с оттенками пастели;

• white – палитра белого цвета;

• flag – чередование красного, белого, синего и черного цветов;

• lines – палитра с чередованием цветов линий;
• colorcube – расширенная палитра RGB;
• jet – разновидность палитры HSV;
• prism – призматическая палитра цветов;
• cool – оттенки голубого и фиолетового цветов;
• autumn – оттенки красного и желтого цветов;
• spring – оттенки желтого и фиолетового цветов;
• winter – оттенки синего и зеленого цветов;
• summer – оттенки зеленого и желтого цветов.

Все эти палитры могут служить параметрами команды colormap, например colormap(hsv) фактически устанавливает то же, что и команда colormap('default').

Окраска поверхностей. Для окраски поверхностей используется команда shading, которая управляет объектами surface (поверхность) и patch (заплата), создаваемыми командами и функциями surf, mesh, pcolor, fill и fill3. Команда shading (затенение) работает с параметрами и имеет следующий вид:

• shading flat задает окраску ячеек или граней в зависимости от текущих данных;
• shading interp задает окраску с билинейной интерполяцией цветов;
• shading faceted – равномерная раскраска ячеек поверхности (принята по умолчанию).

Эти команды устанавливают свойства EdgeColor и FaceColor для графических объектов surface и patch в зависимости от того, какая из команд – mesh (сетчатая поверхность) или surf (затененная поверхность) – используется.

Другие команды управления световыми эффектами. Отметим некоторые дополнительные команды, связанные с управлением цветовыми палитрами:

• colstyle – выделение цвета и стиля графика из заданного массива;
• rgbplot – изображение палитры;
• hsv2rgb – преобразование палитры HSV в палитру RGB;
• rgb2hsv – преобразование палитры RGB в палитру HSV;
• brighten – управление яркостью;
• contrast – управление контрастом;
• hidden – управление показом невидимых линий;
• whitebg – управление цветом фона.

Ознакомиться с информацией об этих командах можно с помощью команды help.

5.4. Анимационная графика

Движение точки на плоскостию Для отображения движения точки по траектории используется команда comet. Используются следующие формы представления этой команды:

• comet(Y) отображает движение «кометы» по траектории, заданной вектором Y;

• comet(X,Y) отображает движение «кометы» по траектории, заданной парой векторов Y и X;

• comet(X,Y,p) аналогична предшествующей команде, но позволяет задавать длину хвоста кометы (отрезка траектории, выделенного цветом) как p*length(Y), где length(Y) – размер вектора Y, а p<1. По умолчанию p = 0.11.

```
>> X=0:0.01:15; comet(X,sin(X),0.15)
```

Движение точки в пространстве. Команда comet3 позволяет наблюдать движение точки в трехмерном пространстве:

• comet3(Z) отображает движение точки с цветным «хвостом» по трехмерной кривой, определенной массивом Z;

• comet3(X,Y,Z) отображает движение точки «кометы» по кривой в пространстве, заданной точками [X(i),Y(i),Z(i)];

• comet3(X,Y,Z,p) аналогична предшествующей команде с заданием длины «хвоста кометы» как p*length(Z). По умолчанию параметр p равен 0.1.

```
>> W=0:pi/500:10*pi; comet3(cos(W),sin(W)+W/10,W)
```

7.1.3. Основные средства анимации

Для более сложных случаев анимации возможно применение техники мульти_

пликации. Она сводится к построению ряда кадров изображения, причем каждый

кадр появляется на некоторое время, затем стирается и заменяется на новый кадр,

несколько отличающийся от предшествующего. Если это отличие незначительно,

то создается иллюзия плавного перемещения объекта.

Отметим кратко основные команды, реализующие анимацию в системе MATLAB:

• capture – захват видеоизображения;

• getframe – создание кадра для анимации;

• moviein – выполнение анимации;

• rotate – вращение фигуры;

• frame2im – преобразование кадра в графический образ;

• im2frame – преобразование графического образа в кадр.

Применение некоторых из этих команд мы рассмотрим далее на конкретных примерах.

Лекция № 6

Программирование в MATLAB

6.1. Основные средства программирования

Программы на языке MATLAB хранятся в m-файлах текстового формата, содержащих определенную последовательность команд. Язык программирования системы MATLAB имеет следующие средства:

- данные различного типа;
- константы и переменные;
- операторы, включая операторы математических выражений;
- встроенные команды и функции;
- функции, определенные пользователем;
- управляющие структуры;
- системные операторы и функции;
- средства расширения языка.

Коды программ в системе MATLAB пишутся на языке высокого уровня. Язык программирования MATLAB является типичным *интерпретатором*. Это означает, что MATLAB не создает исполняемых конечных программ в виде машинных кодов; программы существуют лишь в виде m_файлов.

Высокая скорость выполнения программ обеспечена наличием заведомо откомпилированного ядра, хранящего в себе критичные к скорости выполнения инструкции.

Для выполнения программ необходима среда MATLAB, занимающая на диске много места. Однако для программ на языке MATLAB созданы компиляторы, транслирующие программы MATLAB в коды языков программирования, вчастности, C/C++, JAVA, HTML. Это решает задачу создания исполняемых программ, первоначально разрабатываемых в среде MATLAB.

6.2. Виды программирования

Язык программирования системы MATLAB вобрал в себя все средства, необходимые для реализации различных видов программирования:

- процедурного;
- операторного;
- функционального;
- логического;
- структурного (модульного);
- объектно_ориентированного;
- визуально_ориентированного.

В основе *процедурного*, *операторного* и *функционального* типов программирования лежат процедуры, операторы и функции, используемые как основные объекты языка. Эти типы объектов присутствуют в MATLAB.

Логическое программирование реализуется в MATLAB с помощью логических операторов и функций. Они позволяют реализовать основные идеи логического программирования.

MATLAB представляет собой яркий пример плодотворности *структурного* программирования. Подавляющее большинство функций и команд языка представляют собой вполне законченные модули, обмен данными между которыми происходит через их входные параметры, хотя возможен обмен данными и через глобальные переменные. Программные модули оформлены в виде все тех же текстовых m_файлов, которые хранятся на диске и подключаются к программам по мере необходимости.

6.3. Основные понятия программирования

Отличительной особенностью языка программирования MATLAB является отсутствие необходимости в объявлении типов переменных. Их тип автоматически устанавливается в соответстствии с типом данных, используемых при присваивании переменным тех или иных значений.

Другая особенность – отсутствие указания на то, откуда берется та или иная функция или оператор, заданные пользователем. Все они сохраняются на жестком диске и вызываются точно так же, как встроенные в ядро функции и операторы. Подавляющее большинство команд и функций системы MATLAB поставляется в виде таких модулей. Их можно читать (с помощью как редактора MATLAB, так и любого текстового редактора), разбирать по смыслу, модифицировать и исполнять. Таким образом реализуется идея *открытого программирования*.

Объектно-ориентированное программирование также широко представлено в системе MATLAB. Оно особенно актуально при программировании задач графики. Что касается *визуально-ориентированного* программирования, то в MATLAB оно представлено в основном проектированием графического интерфейса пользователя GUI и моделированием заданных блоками устройств и систем в пакете расширения Simulink.

6.3. Двойственность операторов, команд и функций

Для языка системы MATLAB различие между командами (выполняемыми при вводе с клавиатуры) и программными операторами (выполняемыми из программы) является условным. И команды, и программные операторы могут выполняться как из программы, так и в режиме прямых вычислений.

Функция преобразует одни данные в другие. Для многих функций характерен возврат значений в ответ на обращение к ним с указанием списка входных параметров – аргументов.

Функции, возвращающие единственное значение (или один массив), записываются малыми (строчными) буквами в виде:

```
f_name(Список_параметров)
```

Для функций, возвращающих ряд значений или массивов (например, X, Y, Z,…), запись имеет следующий вид:

```
[X,Y,Z,…]=f_name(Список_параметров)
```

Важное значение имеет *двойственность* операторов и функций. Многие операторы имеют свои аналоги в виде функций. Так, например, оператор «+» имеет аналог в виде функции sum. Команды, записанные в виде:

```
Command argument
```

нередко имеют форму записи и в виде функции:

```
Command('argument')
```

Например:

```
>> type('sin')
sin is a built-in function.
>> type sin
sin is a built-in function.
```

Указанная двойственность лежит в основе выбора между процедурным и функциональным типами программирования, каждый из которых имеет своих поклонников и противников и может (в той или иной мере) подходить для решения различных классов задач.

6.4. Некоторые ограничения

Поскольку язык программирования системы MATLAB ориентирован на структурное программирование, в нем программных операторов безусловного перехода GO TO. Имеются лишь управляющие структуры следующих типов:

условных выражений if…else…

elseif…end,

циклы for…end и while…end.

С позиций теории структурного программирования этих средств достаточно для решения любых задач. В MATLAB имеются также операторы-переключатели типа case, облегчающие создание программных конструкций с множественным ветвлением.

Однако в MATLAB исключены те средства, возможности которых можно реализовать уже имеющимися средствами. Зато резко увеличен набор средств для программирования решения математических задач, прежде всего сводящихся к матричным вычислениям и реализации современных численных методов.

6.5. Исполнение программных объектов

m-файлы сценариев (script-файлы) и функций должны иметь уникальные имена. Длина имен не ограничивается, но только первый 31 символ учитывается при идентификации имени. При исполнении программного объекта его имя сравнивается со списком имен, хранящихся в рабочей области и в директориях m_файлов. Если имя оказывается неуникальным, соответствующий программный объект не исполняется, и выводится сообщение об ошибке. Если же имя уникально, то программный объект исполняется в интерпретирующем режиме.

6.6. М-файлы сценариев и функций

Структура и свойства файлов-сценариев

Для создания m_файлов может использоваться как встроенный редактор, так и любой текстовый редактор, поддерживающий форматы ASCII и Unicode. Подготовленный и записанный на диск m-файл становится частью системы, и его можно вызывать как из командной строки, так и из другого m_файла. Есть два типа m-файлов: файлы-сценарии и файлы-функции.

Файл-сценарий, именуемый также Script-файлом, является просто записью серии команд без входных и выходных параметров. Он имеет такую структуру:

```
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
```

Важны следующие свойства файлов-сценариев:

• они не имеют входных и выходных аргументов;

• работают с данными из рабочей области;

• в процессе выполнения не компилируются;

• представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется в сессии.

Основным комментарием (в фирменной документации он назван H1) является первая строка текстовых комментариев, а дополнительным – последующие строки. Основной комментарий выводится при выполнении команд lookfor и help имя_каталога. Полный комментарий выводится при выполнении команды help Имя_файла.

```
%Plot with color red
%Строит график синусоиды линией красного цвета
%с выведенной масштабной сеткой в интервале [xmin,xmax]
x=xmin:0.1:xmax;
plot(x,sin(x),'r')
grid on
```

Первые три строки здесь – это комментарий, остальные – тело файла. Знак % в комментариях должен начинаться с первой позиции строки. В противном случае команда help name не будет воспринимать комментарий и возвратит сообщение вида No help comments found in name.m.

Переменные, используемые в файлах-сценариях, являются глобальными, то есть они действуют одинаково в командах сессии и внутри программного блока, которым является файл-сценарий. Поэтому заданные в сессии значения переменных используются и в теле файла.

Имена файлов-сценариев нельзя использовать в качестве параметров функций, поскольку файлы-сценарии не возвращают значений.

Структура М-файла-функции

М-файл-функция является типичным полноценным объектом языка программирования системы MATLAB. Одновременно он является полноценным модулем с точки зрения структурного программирования, поскольку содержит входные и выходные параметры и использует аппарат локальных переменных. Структура такого модуля с одним выходным параметром выглядит следующим образом:

```
function var=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
var=выражение
```

М-файл-функция имеет следующие свойства:

• он начинается с объявления function, после которого указываются имя переменной var – выходного параметра, имя самой функции и список ее входных параметров;

• функция возвращает свое значение и может использоваться в виде name(Список_параметров) в математических выражениях;

• все переменные, имеющиеся в теле файла_функции, являются локальными, то есть действуют только в пределах тела функции;

• файл_функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;

• правила вывода комментариев те же, что у файлов_сценариев;

• файл_функция служит средством расширения системы MATLAB;

• при обнаружении файла_функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы MATLAB.

Последняя конструкция var=выражение вводится, если требуется, чтобы функция возвращала результат вычислений.

Приведенная форма файла-функции характерна для функции с одним выходным параметром. Если выходных параметров больше, то они указываются в квадратных скобках после слова function. При этом структура модуля имеет следующий вид:

```
function [var1,var2,…]=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
```

```
var1=выражение
var2=выражение
...
```

Такая функция во многом напоминает процедуру. Ее нельзя слепо использовать непосредственно в математических выражениях, поскольку она возвращает не единственный результат, а множество результатов – по числу выходных параметров. Если функция используется как имеющая единственный выходной параметр, но имеет ряд выходных параметров, то для возврата значения будет использоваться первый из них. Это зачастую ведет к ошибкам в математических вычислениях. Поэтому, как отмечалось, данная функция используется как отдельный элемент программ вида:

```
[var1,va2,...]=f_name(Список_параметров)
```

После его применения переменные выхода var1, var2, ... становятся определенными, и их можно использовать в последующих математических выражениях и иных сегментах программы. Если функция используется в виде f_name (Список_параметров), то возвращается значение только первого выходного параметра – переменной var1.

Статус переменных в функциях

Переменные, указанные в списке параметров функции, являются *локальными* и служат для переноса значений, которые подставляются на их место при вызовах функций.

Возврат из функции производится после обработки всего тела функции, тоесть при достижении конца файла функции. При использовании в теле функции условных операторов, циклов или переключателей иногда возникает необходимость осуществить возврат функции раньше, чем будет достигнут конец файла.

Для этого служит команда return. В любом случае результатом, возвращаемым функцией, являются значения выходных параметров, присвоенные им на момент возврата.

```
function z=fun(x,y)
z=x^2+y^2;
```

Команда глобализации переменных global

Переменные в файлах-сценариях являются *глобальными*, а в файлах_функциях – *локальными*. Нередко применение глобальных переменных в программных модулях может приводить к побочным эффектам. Применение локальных переменных устраняет эту возможность и отвечает требованиям структурного программирования.

Однако передача данных из модуля в модуль в этом случае происходит только через входные и выходные параметры, что требует тщательного планирования такой передачи.

При создании файлов-функций порой желательно применение глобальных переменных. Ответственность за это должен брать на себя программист, создающий программные модули.

Команда global var1 var2… позволяет объявить переменные модуля-функции глобальными. Таким образом, внутри функции могут использоваться и такие переменные, если это нужно по условиям решения вашей задачи. Чтобы несколько программных модулей могли совместно использовать глобальную переменную, ее идентификатор должен быть объявлен в составе global *во всех модулях*.

Использование подфункций

В функции системы MATLAB можно включать *подфункции*. Они объявляются и записываются в теле основных функций и имеют идентичную им конструкцию. Не следует путать эти функции с внутренними функциями, встроенными в ядро системы MATLAB. Ниже представлен пример функции с подфункцией:

```
function [mean,stdev] = statv(x)
%STATV Interesting statistics.
%Пример функции с встроенной подфункцией
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);
%———————————-
function m = avg(x,n)
%Mean subfunction
m = sum(x)/n;
```

В этом примере среднее значение элементов вектора x вычисляется с помощью подфункции avg(x,n), тело которой записано в теле основной функции statv.

Пример использования функции statv представлен ниже:

```
>> V=[1 2 3 4 5]
V =
     1 2 3 4 5
>> [a,m]=statv(V)
a =
     3
m =
     1.4142
>> statv(V)
ans =
     3
>> help statv
     STATV Interesting statistics.
```

Подфункции определены и действуют локально, то есть только в пределах m-файла, определяющего основную функцию. Команда help name выводит

комментарий, относящийся только к основной функции, тогда как команда type name выводит весь листинг m-файла. Так что заданные в некотором m-файле подфункции нельзя использовать ни в командном режиме работы, ни в других m-файлах.

При обращении к функции интерпретатор системы MATLAB прежде всего просматривает m-файл на предмет выявления подфункций. Если они обнаружены, то задаются как локальные функции. Благодаря локальному действию подфункций их имена могут совпадать с именами основных функций системы. Если в функции и подфункциях должны использоваться общие переменные, их надо объявить глобальными как в функции, так и в ее подфункциях.

Частные каталоги

Для записи m-файлов используются каталоги, называемые *родительскими каталогами*. Они содержат группы файлов определенного функционального назначения, например по статистическим расчетам, матричным операциям, вычислению определенных классов функций и т. д.

Есть возможность в родительских каталогах создавать *частные каталоги* с именем **PRIVATE**. Расположенные в них m_файлы доступны только файлам родительского каталога. Файлы частных каталогов просматриваются интерпретатором системы MATLAB в первую очередь. Применение частных каталогов позволяет изменять исходные файлы, сохраняя оригиналы в родительском каталоге в неизменном виде.

Если вы решили отказаться от применения измененного файла, достаточно стереть его в частном каталоге. Такая возможность связана с тем, что интерпретатор при поиске m-файла прежде всего просматривает частный каталог и интерпретирует найденный в нем файл. И только если файл не найден, ищется файл в родительском каталоге.

6.7. Обработка ошибок и комментарии

Вывод сообщений об ошибках

При появлении ошибки вычисления могут завершиться досрочно с выводом сообщения об ошибке. Однако, не все ошибки вызывают остановку вычислений.

Некоторые сопровождаются только выдачей предупреждающей надписи. Такие ситуации должны учитываться программистом, отмечаться как ошибочные и по возможности устраняться. Для вывода сообщения об ошибке служит команда

```
error('Сообщение об ошибке')
```

при выполнении которой вычисления прерываются и выдается сообщение об ошибке, заданное в апострофах. Например:

sd(*x*)=sin(*x*)/*x*, в котором задано сообщение об ошибке на русском языке:

```
function f=sd(x)
if x==0 error('Ошибка – деление на 0'), end
f=sin(x)/x
```

Для выявления ситуации об ошибке использован оператор условного перехода if, который будет описан детально несколько позднее. Результат выполнения данной функции приводится ниже:

```
>> sd(1)
f =
0.8415
ans =
0.8415
>> sd(0)
??? Error using ==> sd
Ошибка – деление на 0
```

Если остановка программы при появлении ошибки нежелательна, то может использоваться команда вывода предупреждающего сообщения:

```
warning('Предупреждающее сообщение')
```

Эта команда выводит стоящее в апострофах сообщение, но не препятствует дальнейшей работе программы. Признаком того, что является ошибкой, а что – предупреждением, являются символы ??? и слово Warning в соответствующих сообщениях.

Функция nargchk часто используется внутри m_файлов для проверки соответствия количества входных параметров (аргументов):

• msg = nargchk(low,high,number) возвращает сообщение об ошибке, если число number меньше, чем low, или больше, чем high, в противном случае возвращается пустая строка.

```
>> msg = nargchk(4,9,5)
msg = [ ]
>> msg = nargchk(4,9,2)
msg = Not enough input arguments.
```

• msg = nargoutchk(low,high,number) возвращает сообщение об ошибке, если число number выходных параметров (выходных аргументов в терминологии MATLAB) меньше, чем low, или больше, чем high, в противном случае возвращается пустая строка.

Комментарии

Как отмечалось, команда help name, где name – имя m-файла, обеспечивает чтение первой строки с текстовым комментарием и тех строк с комментариями, которые следуют непосредственно за первой строкой. Комментарий, расположенный за пределами этой области, не выводится. Это позволяет создавать невыводимый программный комментарий, например:

```
Z=X+Y %Массив Z является суммой массивов X и Y
```

Пустая строка прерывает вывод комментария при исполнении команды help name. Команда type name выводит текст программы со всеми комментариями, в том числе и следующими после пустых строк. Надо быть осторожными с вводом русскоязычных комментариев – нередко это является причиной неустойчивой работы программ.

Команда help catalog, где catalog – имя каталога с m-файлами, позволяет вывести комментарий, общий для всего каталога. Такой комментарий содержится в файле **contents.m**, который пользователь может создать самостоятельно с помощью редактора m-файлов. Если такого файла нет, то будет выведен список первых строк комментариев для всех m-файлов каталога.

6.8. Особенности работы с m-файлами

Выполнение m-файлов-функций

М-файлы-функции могут использоваться как в командном режиме, так и вызываться из других М-файлов. При этом необходимо указывать все входные и выходные параметры. Исключением является случай, когда выходной параметр единственный – в этом варианте функция возвращает единственный результат и может использоваться в математических выражениях. При использовании глобальных переменных они должны быть объявлены во всех m-файлах, используемых в решении заданной задачи, и во всех входящих в них встроенных подфункциях.

Имена функций должны быть уникальными. Это связано с тем, что при обнаружении каждого нового имени MATLAB проверяет, относится ли это имя к переменной, подфункции в данном m-файле, частной функции в каталогах **PRIVATE** или функции в одном из каталогов пути доступа. Если последняя

встречается, то будет исполнена именно эта функция. В новой версии MATLAB возможно переопределение функции, но это не рекомендуется делать подавляющему большинству пользователей системы.

Если аргумент функции используется только для вычислений и его значения не меняются, то аргумент передается ссылкой, что уменьшает затраты памяти.

В других случаях аргумент передается значением. Для каждой функции выделяется своя (рабочая) область памяти, не входящая в область, предоставляемую системе MATLAB. Глобальные переменные принадлежат ряду областей памяти.

При их изменении меняется содержимое всех этих областей.

При решении задач с большим объемом данных может ощущаться нехватка оперативной памяти. Признаком этого становится появление сообщения об ошибке

«Out of memory». В этом случае может быть полезным применение следующих мер:

- стирание ставших ненужными данных, прежде всего больших массивов;
- увеличение размеров файла подкачки Windows;
- уменьшение размера используемых данных;
- снятие ограничений на размеры используемой памяти;
- увеличение объема физической памяти компьютера.

Чем больше емкость ОЗУ компьютера, на котором используется система

MATLAB, тем меньше вероятность возникновения указанной ошибки. Опыт показывает, что даже при решении задач умеренной сложности емкость ОЗУ не должна быть менее 16–32 Мб.

Лекция № 7

Управляющие структуры

7.1. Управляющие структуры

Помимо программ с *линейной структурой* инструкции которых исполняются строго по порядку, существует множество программ, структура которых *нелинейна*. При этом ветви программ могут выполняться в зависимости от определенных условий, иногда с конечным числом повторений – циклов, иногда в виде циклов, завершаемых при выполнении заданного условия. Практически любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры. Они имеются в любом языке программирования и, в частности, в MATLAB.

Диалоговый ввод

Пример диалоговой программы для многократного вычисления длины окружности по вводимому пользователем значению радиуса r:

```
% Вычисление длины окружности с диалоговым вводом радиуса
r=0;
while r>=0,
r=input('Введите радиус окружности r=');
if r>=0 disp(' Длина окружности l='); disp(2*pi*r), end
end
```

На примере этой программы можно увидеть работу управляющих структур типа if…end и while…end, а также функций диалогового ввода input('String') и вывода disp.

Диалог реализован с помощью команды input. Функция input может использоваться и для ввода произвольных строковых выражений.

```
input('Комментарий','s')
```

При выполнении этой функции она останавливает вычисления и ожидает ввода строкового комментария. После ввода возвращается набранная строка.

С помощью команды disp при выводит надпись «Длина окружности l=» и вычисленное значение длины окружности. Она представляет собой одну из наиболее простых управляющих структур типа if…end.

Управляющая структура while…end необходима для циклического повторения вычислений. Пока r>=0, цикл повторяется. Но стоит задать r<0, вычисление длины окружности перестает выполняться, а цикл завершается.

Если данная программа записана в виде m_файла circ.m, то работа с ней будет выглядеть следующим образом:

```
>> circ
```

```
Введите радиус окружности r=1
Длина окружности l=
6.2832
Введите радиус окружности r=2
Длина окружности l=
12.5664
Введите радиус окружности r=-1
>>
```

Условный оператор if…elseif…else…end

Условный оператор if в общем виде записывается следующим образом:

```
if Условие
Инструкции_1
elseif Условие
Инструкции_2
else
Инструкции_3
end
```

Эта конструкция допускает несколько частных вариантов:

```
if Условие Инструкции end
```

пока Условие возвращает логическое значение 1 (то есть «истина»), выполняются Инструкции, составляющие тело структуры if…end. При этом оператор end указывает на конец перечня инструкций. Инструкции в списке разделяются запятой или точкой с запятой. Если Условие не выполняется (дает логическое значение 0, то есть «ложь»), то Инструкции также не выполняются.

```
if Условие Инструкции_1 else Инструкции_2 end
```

выполняет Инструкции_1, если выполняется Условие, или Инструкции_2 в противном случае.

Условия записываются в виде:

```
Выражение_1 Оператор_отношения Выражение_2
```

причем в качестве Операторов_отношения используются следующие операторы: ==, <, >, <=, >= или ~=.

Циклы типа for…end

Циклы типа for…end обычно используются для организации вычислений с заданным числом повторяющихся циклов:

```
for var=Выражение, Инструкция, …, Инструкция end
```

Следующие примеры поясняют применение цикла для получения квадратов значений переменной цикла:

```
>> for i=1:5 i^2, end;
ans = 1
ans = 4
ans = 9
ans = 16
ans = 25
```

Возможны вложенные циклы, например:

```
for i=1:3
    for j=1:3
    A(i,j)=i+j;
    end
end
```

В результате выполнения этого цикла формируется матрица A:

```
>> for2
>> A
A =
2 3 4
3 4 5
4 5 6
```

MATLAB допускает использование в качестве переменной цикла массива A размера *m*x*n*. При этом цикл выполняется столько раз, сколько столбцов в массиве A, и на каждом шаге переменная var представляет собой вектор, соответствующий текущему столбцу массива A:

```
>> A=[1 2 3;4 5 6]
A =
1 2 3
4 5 6
>> for var=A; var, end
var =
1
4
var =
2
5
var =
3
6
```

Циклы типа while…end

Цикл типа while выполняется до тех пор, пока выполняется Условие:

```
while Условие
Инструкции
End
```

Досрочное завершение циклов реализуется с помощью операторов break или continue.

Цикл while часто используется для подготовки итерационных программ, завершающих свою работу по какому либо условию. Ниже представлен фрагмент программы, в которой формируется фрактальное изображение, напоминающее лист папоротника.

```
i=10
while i<1
i=i/2
end
```

Конструкция переключателя switch…case…end

Для осуществления множественного выбора (или ветвления) используется конструкция с переключателем типа switch:

```
switch switch_Выражение
case case_Выражение
    Список_инструкций
case {case_Выражение1, Case_выражение2, case_Выражение3,…}
    Список_инструкций
...
otherwise,
    Список_инструкций
end
```

Если выражение после заголовка switch имеет значение одного из выражений case_Выражение…, то выполняется блок операторов case, в противном случае – список инструкций после оператора otherwise. При выполнении блока case исполняются те списки инструкций, для которых case_Выражение совпадает с switch_Выражением. Case_Выражение может быть числом, константой, переменной, вектором ячеек или даже строчной переменной. В последнем случае оператор case истинен, если функция strcmp (значение, выражение) возвращает логическое значение «истинно».

```
switch var
case {1,2,3}
disp('Первый квартал')
case {4,5,6}
disp('Второй квартал')
case {7,8,9}
disp('Третий квартал')
case {10,11,12}
disp('Четвертый квартал')
```

```
otherwise
disp('Ошибка в задании')
end
```

Эта программа в ответ на значения переменной var – номера месяца – вычисляет, к какому кварталу относится заданный месяц, и выводит соответствующее сообщение:

```
>> var=2;
>> sw1
Первый квартал
>> var=4;sw1
Второй квартал
>> var=7;sw1
Третий квартал
>> var=12;sw1
Четвертый квартал
>> var=-1;sw1
Ошибка в задании
```

Конструкция try…catch…end

Конструкция блока вывода ошибок try…catch…end:

```
try,
Список инструкций
...,
Список инструкций
catch,
Список инструкций
...,
Список инструкций
End
```

Эта конструкция выполняет все списки инструкций. Если в каком то списке до оператора catch появляется ошибка, то выводится сообщение об ошибке, но системная переменная последней ошибки lasterr не меняется. В сообщениях после catch сообщения об ошибке не выводятся.

В следующем примере задано появление ошибки (переменная aaa не определена), после чего выполняется блок try…catch…end:

```
aaa
??? Undefined function or variable 'aaa'.
try
2+3;
3+4;
2/0;
catch
4+5;
end;
Warning: Divide by zero.
```

```
>> lasterr
ans =
Undefined function or variable 'aaa'.
```

Обратите внимание, что в конце блока команда lasterr выводит ее начальное значение. А в другом примере ошибка задана в блоке try…catch…end уже после оператора catch:

```
>> try
2+3;
3+4;
4+5;
catch
5/0;
end;
>> lasterr
ans =
Undefined function or variable 'aaa'.
```

Как нетрудно заметить, на этот раз ошибка в вычислении 5/0 не выводится, а значение lasterr осталось тем, что было изначально.

Операторы break, continue и return

В управляющих структурах, в частности в циклах for и while, часто используются операторы, влияющие на их выполнение.

Так, оператор break может использоваться для досрочного прерывания выполнения цикла. Как только он встречается в программе, цикл прерывается:

```
for i=1:10 i, if i==5 break, end, end;
i = 1
i = 2
i = 3
i = 4
i = 5
```

Оператор continue передает управление в следующую итерацию цикла, пропуская операторы, которые записаны за ним, причем во вложенном цикле он передает управление на следующую итерацию основного цикла. Приведенный ниже пример обеспечивает подсчет числа строк в файле magic.m:

```
fid = fopen('magic.m','r'); count = 0;
while ~feof(fid)
line = fgetl(fid);
if isempty(line) | strncmp(line,'%',1)
continue
end
count = count + 1;
end
disp(sprintf('%d lines',count));
```

```
25 lines
```

Оператор return обеспечивает нормальный возврат в вызывающую функцию или в режим работы с клавиатурой. Пример применения оператора return:

```
function d = det(A)
%DET det(A) is the determinant of A.
if isempty(A)
d = 1;
return
else
…
end
```

В данном примере, если матрица A пустая, будет выведено значение 1, после чего управление будет передано в блок else…end.

Лекция № 8

Управляющие структуры

8.1. Работа с файлами

Открытие и закрытие файлов

Перед использованием любого файла он должен быть *открыт*, а по окончании использования – *закрыт*. Много файлов может быть открыто и доступно для чтения одновременно.

Команда *open имя*, где имя должно содержать массив символов или символьную переменную, открывает файлы в зависимости от анализа параметра имя и расширения в имени имя:

• переменная – открывает массив, названный по имени, в редакторе массивов (Array Editor);

• .mat – открывает файл, сохраняет переменные в структуре в рабочей области;

• .fig – открывает его в редакторе дескрипторной графики Property Editor;

• .m – открывает m_файл в редакторе_отладчике;

• .mdl – открывает модель в Simulink;

• .p – открывает, если он есть, m_файл с тем же именем;

• .html – открывает HTML_документ в браузере помощи.

Если файлы с расширением существуют в пути MATLAB, то открывается тот файл, который возвращается командой which имя, если нет – то файл из файловой системы. Если файл не имеет расширения имени, то он открывается той программой, формат файлов которой был бы обнаружен функцией which('имя файла'). По умолчанию для всех файлов с окончаниями, отличными от вышеперечисленных, вызывается openother. Open вызывает функции openxxx, где xxx – расширение файла. Исключение – переменные рабочей области, для которых вызывается openvar, и рисунки, для работы с которыми вызывается openim. Создавая m_файлы с именем openxxx, пользователи могут изменять обработку файлов и добавлять новые расширения в список. Закрывать файлы, открытые при помощи open, нужно из редакторов, вызываемых openxxx.

• [FILENAME, PATHNAME] = uigetfile(FILTERSPEC, Title). Открывает диалог с именем Title и фильтром FILTERSPEC (например, массивом ячеек, содержащим расширения файлов) и возвращает файл, выбранный пользователем, и путь к нему. Возвращает FILENAME=0, если файл не существует или если пользователь нажал на Cancel.

• [FILENAME, PATHNAME] = uigetfile (FILTERSPEC, Title, X, Y) размещает окно диалога в точке *X*, *Y* (координаты в пикселях). Пример:

```
[filename, pathname] = uigetfile('*.m;*.fig;*.mat;*.mdl', 'All MATLAB Files (*.m, *.fig, *.mat, *.mdl)';
```

– 'r' – открытие файла для чтения (по умолчанию);

– 'r+' – открытие файла для чтения и записи;

– 'w' – удаление содержимого существующего файла или создание нового и открытие его для записи;

– 'a' – создание и открытие нового файла или открытие существующего для записи с добавлением в конец файла.

Добавление к этой строке 'b' (подразумевается по умолчанию) предписывает системе открыть файл в двоичном режиме.

Добавление же вместо b к этой строке 't', например 'rt', в операционных системах, которые имеют различие между текстовыми и двоичными файлами, предписывает системе открыть файл в текстовом режиме. Например, во всех версиях MATLAB для Windows/MS_DOS и VMS нельзя открыть текстовый файл без параметра 'rt'. При вводе файлов с использованием fopen в текстовом режиме удаляются все символы «возврат каретки» перед символом новой строки.

• [fid,message] = fopen(filename,permission,format) открывает файл, как описано выше, возвращая идентификатор файла и сообщение.

Кроме того, значение параметра format позволяет точно определить числовой формат. Возможны 8 форматов, описание которых можно найти в справочной системе. В частности, строка format может иметь значения 'native' (формат компьютера, на котором установлена система), 'vax', 'cray' (компьютеры VAX и Cray) и т. д.

Определенные вызовы функций fread или fwrite могут отменить числовой формат, заданный при вызове функции fopen.

• fids = fopen('all') возвращает вектор_строку, содержащую идентификаторы всех открытых файлов, не включая стандартные потоки 0, 1 и 2.

Число элементов вектора равно числу открытых пользователем файлов.

• [filename,permission,format] = fopen(fid) возвращает полное имя файла, строку permission и строку format. При использовании недопустимых значений fid возвращаются пустые строки для всех выходных аргументов.

Команда fclose закрывает файл. Она имеет следующие варианты.

• status = fclose(fid) закрывает файл, если он открыт. Возвращает статус файла status, равный 0, если закрытие завершилось успешно, и –1 в противном случае. Аргумент fid – это идентификатор, связанный с открытым файлом (см. функцию fopen для более подробного описания).

• status = fclose('all') закрывает все открытые файлы. Возвращает 0 в случае успешного завершения и –1 в противном случае.

```
>> fid=fopen('c:\ex','a+')
fid = 4
>> fclose(4)
ans = 0
```

• [A,count] = fread(fid,size,precision) считывает двоичные данные из заданного файла и помещает их в матрицу A. Выходной аргумент count содержит число удачно считанных элементов. Значение идентификатора fid – это целое число, возвращенное функцией fopen; size – аргумент, определяющий количество

считываемых данных. Если аргумент size не определен, функция fread считывает данные до конца файла.

Используются следующие параметры size:

– n – чтение n элементов в вектор_столбец;

– inf – чтение элементов до конца файла и помещение их в вектор_столбец, содержащий такое же количество элементов, что и в файле;

– [m,n] – считывает столько элементов, сколько нужно для заполнения матрицы $m \times n$.

Заполнение происходит по столбцам. Если элементов в файле мало, то матрица дополняется нулями. Если считывание достигает конца файла, не заполнив матрицу необходимого размера, то матрица дополняется нулями.

Если происходит ошибка, чтение останавливается на последнем считанном значении. Параметр precision – строка, определяющая числовую точность считывания значений, она контролирует число считанных бит для каждого значения и интерпретирует эти биты как целое число, число с плавающей запятой или как символ.

• [A,count] = fread(fid,size,precision,skip) включает произвольный аргумент skip, определяющий число байтов, которые необходимо пропустить после каждого считывания. Это может быть полезно при извлечении данных в несмежных областях из записей фиксированной длины.

Если precision имеет битовый формат, такой как 'bitN' или 'ubitN', значение skip определяется в битах. Обширный список возможных значений параметра precision можно найти в справочной системе MATLAB.

• count=fwrite(fid,A,precision) записывает элементы матрицы A в файл, представляя их с заданной точностью. Данные записываются в файл по столбцам, выходной аргумент count содержит число удачно записанных элементов. Значение идентификатора fid – это целое число, полученное при использовании функции fopen. Добавляет символы «возврат каретки» перед началом новой строки.

• count=fwrite(fid,A,precision,skip) делает то же, но включает произвольный аргумент skip, определяющий число байтов, которые надо пропустить перед каждой записью. Это полезно при вставке данных в несмежные области в записях фиксированной длины. Если precision имеет битовый формат, такой как 'bitN' или 'ubitN', значение skip определяется в битах.

```
>> fid = fopen('c:\prim','a+')
fid = 3
>> A=[30 39 48; 38 47 7; 46 6 8]
>> count = fwrite(3,A)
count = 49
>> status = fclose(3)
status = 0
>> fid = fopen('c:\prim','r')
fid = 3
>> [B,count] = fread(3,[7,7])
B =
30 39 48
38 47 7
```

```
46 6 8
count = 49
```

Операции над форматированными файлами

Файлы, содержащие форматированные данные, называют *форматированными файлами*. Ниже представлены функции, которые служат для работы с такими файлами.

• line = fgetl(fid) возвращает строку из файла с идентификатором fid с удалением символа конца строки. Если функция fgetl обнаруживает конец файла, то она возвращает значение –1 (см. функцию fopen с более подробным описанием fid).

• line = fgets(fid) возвращает строку из файла с идентификатором fid, не удаляя символ конца строки. Если функция fgets обнаруживает конец файла, то она возвращает значение –1.

• line = fgets(fid,nchar) возвращает не больше чем nchar первых символов строки. После признака конца строки или конца файла никакие дополнительные символы не считываются (см. примеры к функции fscanf).

• count = fprintf(fid,format,A,…) форматирует данные, содержащиеся в действительной части матрицы A, под контролем строки format и записывает их в файл с идентификатором fid. Функция fprintf возвращает число записанных байтов. Значение идентификатора fid – целое число, возвращаемое функцией fopen.

Если опустить идентификатор fid в списке аргументов функции fprintf, то вывод будет осуществляться на экран, так же как при использовании стандартного вывода (fid=1).

• fprintf(format,A,…) – запись осуществляется на стандартное устройство – экран (но не в файл). Строка format определяет систему счисления, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные буквы алфавита наряду со спецификаторами, знаками выравнивания и т. д.

Функция fprintf ведет себя как аналогичная функция fprintf() языка ANSI C с некоторыми исключениями и расширениями. Ниже описаны специальные символы, встречающиеся в строке format.

\n - Новая строка

\t - Горизонтальная табуляция

\b - Возврат на один символ

\r - Возврат каретки

\f - Новая страница

\\ - Обратный слэш

\" - или " Одиночная кавычка

%% - Процент

Для вывода числовых или символьных данных в строке формата необходимо использовать *спецификаторы*, перечисленные ниже.

%c - Одиночный символ

%d - Десятичная система обозначений (со знаком)

%e - Экспоненциальное представление чисел с использованием символа «e» в нижнем регистре, например 3.1415e + 00

%E - Экспоненциальное представление чисел с использованием символа «E» в верхнем регистре, например 3.1415E + 00

%f - Система обозначений с фиксированной точкой

%g - Наиболее компактный вариант из %e и %f. Незначащие нули не выводятся

%G - То же самое, что и %g, но используется верхний регистр для символа «E»

%o - Восьмеричная система обозначений (без знака)

%s - Строка символов

%u - Десятичная система обозначений (без знака)

%x - Шестнадцатеричная система обозначений с использованием символов нижнего регистра («a»…«f»)

%X - Шестнадцатеричная система обозначений с использованием верхнего регистра символов («A»…«F»)

Между знаком процента и буквой в спецификатор могут быть вставлены дополнительные символы. Их значение ниже.

Знак «минус» (-) Выравнивание преобразованных аргументов %–5.2d по левому краю

Знак «плюс» (+) Всегда печатать знак числа (+ или -) %+5.2d

Нуль (0) Заполнение нулями вместо пробелов %05.2d

Цифры Определяет минимальное число знаков, %6f которые будут напечатаны

Цифры (после точки) Число после точки определяет количество %6.2f символов, печатаемых справа от десятичной точки

• A = fscanf(fid,format) читает все данные из файла с идентификатором fid, преобразует их согласно значению параметра format и возвращает в виде матрицы A. Значение идентификатора fid – целое число, возвращаемое функцией fopen. Параметр format представляет собой строку, определяющую формат данных, которые необходимо прочитать.

• [A,count] = fscanf(fid,format,size) считывает количество данных, определенное параметром size, преобразует их в соответствии с параметром format и возвращает вместе с количеством успешно считанных элементов count. Параметр size – это произвольный аргумент, определяющий количество считываемых данных. Допустимы следующие значения:

• n – чтение n элементов в вектор_столбец;

• inf – чтение элементов до конца файла и помещение их в вектор_столбец, содержащий такое же количество элементов, что и в файле;

• [m,n] – считывает столько элементов, сколько требуется для заполнения матрицы размера $m \times n$. Заполнение происходит по столбцам. Величина n (но не m!) может принимать значение Inf.

Строка format состоит из обычных символов и (или) спецификаторов. Спецификаторы указывают тип считываемых данных и включают символ %, опцию ширины поля и символы формата. Возможные символы формата перечислены ниже

%c Последовательность символов; параметр ширины поля определяет количество считываемых символов

%d Десятичное число

%e, %f, %g Число с плавающей точкой

%i Целое число со знаком

%o Восьмеричное число со знаком

%s Последовательность непробельных символов

%u Десятичное целое число со знаком

%x Шестнадцатеричное целое число со знаком

[...] Последовательность символов

Между символом % и символом формата допустимо вставлять следующие символы:

• звездочка (*) означает, что соответствующее значение не нужно сохранять в выходной матрице;

• строка цифр задает максимальную ширину поля;

• буква обозначает размер полученного объекта: h для короткого целого числа (например, %hd), l для длинного целого числа (например, %ld) или для числа с двойной точностью с плавающей запятой (например, %lg).

Примеры:

```
>> x = 0:pi/10:pi;y=[x;sin(x)];
>> fid = fopen('c:\sin.txt','w');
>> fprintf(fid,'%5.3f %10.6f\n',y);fclose(fid);
0.000 0.000000
0.314 0.309017
0.628 0.587785
0.942 0.809017
1.257 0.951057
1.571 1.000000
1.885 0.951057
2.199 0.809017
2.513 0.587785
2.827 0.309017
3.142 0.000000
>> fid = fopen('c:\sin.txt','r');
>> q=fscanf(fid,'%g',[2,10]);
>> q'
ans =
0 0
0.3140 0.3090
0.6280 0.5878
0.9420 0.8090
1.2570 0.9511
1.5710 1.0000
1.8850 0.9511
```

```
2.1990 0.8090
2.5130 0.5878
2.8270 0.3090
>> fgetl(fid)
ans = 3.142 0.000000
>> fgets(fid)
ans = -1
>> fclose(fid)
ans = 0
```

Позиционирование файла

При считывании и записи файлов они условно представляются в виде линейно расположенных данных, наподобие записи на непрерывной магнитной ленте.

Место, с которого идет считывание в данный момент (или позиция, начиная с которой идет запись), определяется специальным *указателем*. *Файлы последовательного доступа* просматриваются строго от начала до конца, а в *файлах произвольного доступа* указатель может быть размещен в любом месте, начиная с которого ведется запись или считывание данных файла.

Таким образом, указатель обеспечивает позиционирование файлов. Имеется ряд функций позиционирования:

• eofstat = feof(fid) проверяет, достигнут ли конец файла с идентификатором fid. Возвращает 1, если указатель установлен на конец файла, и 0 в противном случае;

• message = ferror(fid) возвращает сведения об ошибке в виде строки message. Аргумент fid – идентификатор открытого файла (см. функцию fopen с ее подробным описанием);

• message = ferror(fid,'clear') очищает индикатор ошибки для заданного файла;

• [message,errnum] = ferror(…) возвращает номер ошибки errnum последней операции ввода_вывода для заданного файла.

Если последняя операция ввода_вывода, выполненная для определенного значением fid файла, была успешной, значение message – это пустая строка, а errnum принимает значение 0.

Значение errnum, отличное от нуля, говорит о том, что при последней операции ввода_вывода произошла ошибка. Параметр message содержит строку, содержащую информацию о характере возникшей ошибки.

Пример:

```
>> fid=fopen('c:\example1','a+')
fid = 3
>> t = fread(3,[4,5])
t =
Empty matrix: 4-by-0
>> ferror(3)
ans =
Is the file open for reading? . . .
```

• frewind(fid) устанавливает указатель позиции в начало файла с идентификатором fid;

• status = fseek(fid,offset,origin) – устанавливает указатель в файле с идентификатором fid в заданную позицию – на байт, указанный параметром offset относительно origin.

Аргументы:

• fid – идентификатор файла, возвращенный функцией fopen;

• offset – значение, которое интерпретируется следующим образом:

• offset>0 – изменяет позицию указателя на offset байт в направлении к концу файла;

• offset=0 – не меняет позицию указателя;

• offset<0 – изменяет позицию указателя на offset байт в направлении к началу файла;

• origin – аргумент, принимающий следующие значения:

– 'bof' или –1 – начало файла;

– 'cof' или 0 – текущая позиция указателя в файле;

– 'eof' или 1 – конец файла;

• status – выходной аргумент. Принимает значение 0, если операция fseek прошла успешно, и –1 в противном случае. Если произошла ошибка, используйте функцию ferror для получения более подробной информации.

• position=ftell(fid) – возвращает позицию указателя для файла с идентификатором fid, полученным с помощью функции fopen. Выходной аргумент position – неотрицательное целое число, определяющее позицию указателя в байтах относительно начала файла. Если запрос был неудачным, position принимает значение –1. Используйте функцию ferror для отображения характера ошибки. Примеры:

```
>> fid=fopen('c:\example','a+')
fid = 3
>> count = fwrite(3,magic(6))
count = 36
>> ftell(3)
ans = 36
>> frewind(3);ftell(3)
ans = 0
>> fseek(3,12,0);ftell(3)
ans = 12
>> feof(3)
ans = 0
>> fclose(3)
ans = 0
```

• s=sprintf(format,A,…) форматирует данные в матрице A в формате, заданном параметром format, и создает из них строковую переменную s.

• [s,errrmsg] = sprintf(format,A,…) аналогична ранее описанной функции, но дополнительно возвращает строку ошибки errmsg, если ошибка имела место, или пустую строку в противном случае. Строка format определяет систему счисления,

выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные символы наряду со спецификаторами, знаками выравнивания и т. д. Функция fprintf ведет себя, как и аналогичная функция fprintf() языка ANSI C с некоторыми исключениями и расширениями. Примеры:

```
>> sprintf('%0.5g',(1+sqrt(7))/4)
ans = 0.91144
>> sprintf('%s','привет')
ans =
привет
```

Функция sscanf аналогична функции fscanf, за исключением того, что она считывает данные из символьной переменной системы MATLAB, а не из файла.

• A = sscanf(s,format) считывает данные из символьной переменной s, преобразует их согласно значению format и создает на основе этих данных матрицу A. Параметр format определяет формат данных, которые нужно считать.

• A = sscanf(s,format,size) считывает количество данных, определенное параметром size, и преобразует их согласно строке format. Параметр size представляет собой аргумент, определяющий количество данных для чтения. Допустимы следующие значения:

– n – чтение n элементов в вектор_столбец;

– inf – чтение элементов до конца символьной переменной и помещение их в вектор_столбец, содержащий такое же количество элементов, как и в строковой переменной;

– [m,n] – считывает столько элементов, сколько требуется для заполнения матрицы размера $m \times n$. Заполнение происходит по столбцам. Величина n (но не m!) может принимать значение Inf.

• [A,count,errmsg,nextindex] = sscanf(…)считывает данные из символьной переменной s, преобразует их согласно значению format и возвращает в матрицу A. Параметр count – выходной аргумент, который возвращает число успешно считанных элементов; errmsg – выходной аргумент, который возвращает строку ошибки, если ошибка произошла, и пустую строку в противном случае; nextindex – выходной аргумент, который содержит число, на единицу большее, чем количество символов в s.

Строка format состоит из обычных символов и спецификаторов. Спецификаторы указывают тип данных и включают в себя символ %, опцию ширины поля и символы формата. Пояснения можно найти в описании функции fscanf.

Пример:

```
>> s = '4.83 3.16 22 45';
>> [A,n,err,next] = sscanf(s,'%f')
A =
4.8300
3.1600
```

```
22.0000
45.0000
n = 4
err = ''
next = 16
```

Специализированные файлы

Приведенные ниже функции относятся к некоторым *специализированным файлам*.

• M = dlmread(filename,delimiter) считывает данные из файла filename с ASCII_разделителем, используя разделитель delimiter, в массив M. Используйте '\t', чтобы определить в качестве разделителя символ табуляции.

• M = dlmread(filename,delimiter,r,c) считывает данные из файла filename с ASCII_разделителем, используя разделитель delimiter, в массив M, начиная со смещения r (по строкам) и c (по столбцам). Параметры r и c отсчитываются, начиная с нуля, так что r=0, c=0 соответствует первому значению в файле.

• M = dlmread(filename,delimiter,r,c,range) импортирует индексированный или именованный диапазон данных с разделителями в формате ASCII. Для использования диапазона ячеек нужно определить параметр range в следующем виде:

range = [ВерхняяСтрока, ЛевыйСтолбец, НижняяСтрока, ПравыйСтолбец]

Аргументы функции dlmread следующие: delimiter – символ, отделяющий отдельные матричные элементы в электронной таблице формата ASCII, символ запятой (,) – разделитель по умолчанию, r,c – ячейка электронной таблицы, из которой берутся матричные элементы, соответствующие элементам в верхнем левом углу таблицы, range – вектор, определяющий диапазон ячеек электронной таблицы.

Команда dlmwrite преобразует матрицу MATLAB в файл с ASCII_разделителями, читаемый программами электронных таблиц:

• dlmwrite(filename,A,delimiter) записывает матрицу A в верхнюю левую ячейку электронной таблицы filename, используя разделитель delimiter для отделения элементов матрицы. Используйте '\t' для создания файла с элементами, разделенными табуляцией. Все элементы со значением 0 опускаются. Например, массив [1 0 2] появится в файле в виде '1,,2' (если разделителем является запятая);

• dlmwrite(filename,A,delimiter,r,c) записывает матрицу A в файл filename, начиная с ячейки, определенной r и c, используя разделитель delimiter.

• M = wk1read(filename) считывает электронную таблицу Lotus123 (WK1) в матрицу M.

• M = wk1read(filename,r,c) считывает данные, начиная с ячейки, определенной значениями (r,c). Параметры r и c отсчитываются от нуля, так что r=0, c=0 определяют первую ячейку в файле.

• M = wk1read(filename,r,c,range) считывает диапазон значений, определенный параметром range, где range может быть представлен в одной из следующих форм:

• вектор с четырьмя элементами, определяющий диапазон ячеек в формате [верхняя_строка, левый_столбец, нижняя_строка, правый_столбец];

• диапазон ячеек, определенный строкой, например 'A1...C5';

• имя диапазона, определенное в виде строки, например 'Sales'.

wk1write(filename,M) записывает значения матрицы M в файл filename электронной таблицы Lotus123 WK1.

wk1write(filename,M,r,c) записывает данные, начиная с ячейки, определенной значениями (r,c). Параметры r и c отсчитываются от нуля, так что r=0,

c=0 определяют первую ячейку в электронной таблице.

Необходимо отметить, что большинство рассмотренных выше функций редко применяются пользователями. Но они довольно широко используются в системных целях и представляют большой интерес для специалистов.

11.6.8. Пустые матрицы в структурах if
и while

В управляющих структурах if и while в качестве условия могут применяться пус_
тые матрицы. Такое условие возвращает логическое значение false (0). Пример
фрагмента программы с пустой матрицей:

```
A=ones(10,0,4);
if A S1 else S0 end
```

11.6.9. Создание паузы в вычислениях

Для остановки программы используется оператор pause. Он используется в сле_
дующих формах:

- pause останавливает вычисления до нажатия любой клавиши;
- pause(N) останавливает вычисления на N секунд;
- pause on включает режим отработки пауз;
- pause off выключает режим отработки пауз.

Следующий пример поясняет применение команды pause:

```
for i=1:20;
x = rand(1,40);
y = rand(1,40);
z = sin(x.*y);
tri = delaunay(x,y);
trisurf(tri,x,y,z)
pause(1);
end
```

Команда pause(1) здесь обеспечивает показ 20 рисунков – построений трех_
мерных поверхностей из треугольных окрашенных областей со случайными пара_
метрами.

11.7. Основы объектноО ориентированного программирования

11.7.1. Основные понятия

Мы уже много раз упоминали различные *объекты* языка программирования сис_
темы MATLAB. Это является одним из признаков *объектно_ориентированного программирования* (ООП), причем чисто внешним. В основе объектно_ориенти_
рованного программирования лежат три основных положения.

- *Инкапсуляция* – объединение данных и программ и передача данных через входные и выходные параметры функций. В результате появляется новый элемент программирования – *объект*.__
- *Наследование* – возможность создания родительских объектов и новых

дочерних объектов, наследующих свойства родительских объектов. Воз_
можно также *множественное* наследование, при котором класс наследует
свойства *нескольких* родительских объектов. На наследовании основаны
система задания типов данных, дескрипторная графика и многие другие
приемы программирования. Примеры наследования мы уже неоднократно
отмечали.

• *Полиформизм* – присвоение некоторому действию одного имени, которое
в дальнейшем используется по всей цепочке создаваемых объектов сверху
донизу, причем каждый объект выполняет это действие присущим ему спо_
собом.

В дополнение к этим положениям объектно_ориентированное программирова_
ние в MATLAB допускает *агрегирование* объектов, то есть объединение частей
объектов или ряда объектов в одно целое.

11.7.2. Классы объектов

Объект можно определить как некоторую структуру, принадлежащую к опре_
деленному *классу*. В MATLAB определены следующие семь основных классов
объектов:

• double – числовые массивы с элементами_числами двойной точности;
• sparse – двумерные числовые или комплексные разреженные матрицы;
• char – массивы символов;
• struct – массивы структур (записей);
• cell – массивы ячеек;
• javaarray – массивы Ява;
• function_handle – дескрипторы функций.

С объектами этих классов мы многократно встречались, особо не оговаривая их
принадлежность к объектно_ориентированному программированию. Для MATLAB
вообще характерно, что никакие классы объектов (в том числе заново создавае_
мые) не требуют объявления. Например, создавая переменную name=’Иван’,
мы автоматически получаем объект в виде переменной name класса char. Таким
образом, для переменных принадлежность к тому или иному классу определяется
их значением. Является ли переменная объектом, можно определить при помощи
функции isobject(имя переменной). Аналогичная функция isjava опреде_
ляет, является ли переменная объектом языка Java.

Для создания новых классов объектов служат *конструкторы классов*. По суще_
ству, это m_файлы, имена которых совпадают с именами классов @Имя_класса,
но без символа @. Этим символом помечаются поддиректории системы MATLAB,
в которых имеются конструкторы классов. Множество таких директорий с при_
мерами конструкторов классов вы найдете в поддиректориях MATLAB\
TOOLBOX.

В качестве примера рассмотрим поддиректорию @SYM в директории
TOOLBOX\SYMBOLIC. В этой поддиректории можно найти конструкторы для
более чем сотни объектов пакета символьной математики. К примеру, конструк_
тор функции, вычисляющей арктангенс, выглядит следующим образом:

```
>> help @sym/atan.m
ATAN Symbolic inverse tangent.
>> type @sym/atan.m
function Y = atan(X)
%ATAN Symbolic inverse tangent.
% Copyright (c) 1993-98 by The MathWorks, Inc.
% $Revision: 1.10 $ $Date: 1997/11/29 01:05:16 $
Y = maple('map','atan',X);
```

В данном случае для конструирования нужного объекта используется функ_
ция maple, дающая вход в ядро системы символьной математики Maple, которое
поставляется в составе системы MATLAB по лицензии фирмы MapleSoft, Inc.

Этот пример, кстати, наглядно показывает, что пользователь системы MATLAB
может существенно расширить число объектов класса sym, поскольку ядро систе_
мы Maple содержит намного больше определений, чем пакет символьной матема_
тики системы MATLAB. Для создания новых классов объектов служит функция
class, описанная ниже.

Пакеты прикладных программ системы MATLAB позволяют разработчикам
с большим успехом использовать возможности объектно_ориентированного про_
граммирования путем создания новых классов и объектов. M_файлы системы

представляют собой массу наглядных примеров объектно_ориентированного
программирования на языке MATLAB. Это дает основание ограничиться спра_
вочным описанием основных средств такого программирования с приведением
минимума простых примеров.

11.7.3. Создание класса или объекта

Для создания класса объектов или объектов, а также для и их идентификации слу_
жит функция class. Формы ее применения представлены ниже:

• class(OBJ) возвращает класс указанного объекта OBJ. Типы стандартных классов double, sparse, char, cell, struct, function_handle были перечислены выше. int8 – 8_разрядный массив целых чисел со знаком; uint8 – 8_разряд_ный массив целых чисел без знака;int16 – 16_разрядный массив целых чи_сел со знаком; uint16 – 16_разрядный массив целых чисел без знака; int32 – 32_разрядный массив целых чисел со знаком; uint32 – 32_разрядный массив целых чисел без знака; int64 – 64_разрядный массив целых чисел со знаком; uint64 – 64_разрядный массив целых чисел без знака; <class_name> – класс, определенный пользователем; <java_class> – имя класса Ява;

• OBJ=class(S,'class_name',PARENT1,PARENT2,…) создает объект класса 'class_name' на базе структуры S и родительских объектов PARENT1, PARENT2, ... При этом создаваемый объект наследует структуру и
поля родительских объектов. Объекту OBJ в данном случае присуще *мно_жественное наследование*;__

• OBJ=class(struct[ ],'class_name',PARENT1,PARENT2,…) не мо_жет иметь никаких полей, кроме унаследованных от родительских объектов.

Обратите внимание на то, что эта функция обычно используется в составе m_файлов конструкторов классов объектов.

11.7.4. Проверка принадлежности объекта к заданному классу

Для контроля принадлежности заданного объекта к некоторому классу служит
функция isa:

• isa(OBJ,'Имя_класса') возвращает логическую 1, если OBJ принадле_жит классу с указанным именем. Дополнительно к вышеописанным выделя_
ет классы numeric и single. Но не обнаруживает класс logical. Нужно исполь_
зовать функцию islogical, чтобы проверить принадлежность к этому классу. Примеры применения этой функции:

```
>> X=[1 2 3];
>> isa(X,'char')
```

ans = 0

>> isa(X,'double')

ans = 1

11.7.5. Другие функции объектно>

ориентированного программирования

Для получения списка методов данного класса объектов сейчас чаще используют_

ся функции methodsview и methods. Отличиями от what имя класса является

то, что эти функции возвращают информацию также и о классах Java, но инфор_

мация выводится в отдельном окне, не сообщается информация о папках, все ме_

тоды из всех папок собраны вместе, и повторяющиеся имена методов удалены:

• methodsview имя класса или methods имя класса –full в отдельном окне возвращают полное описание методов класса, включая информацию о наследовании, а для классов Java – и о подписях и атрибутах;

• M=methods ('имя класса',' –full ')возвращает ту же информацию в массиве ячеек M;

• M=methods(' имя класса ') возвращает массив ячеек с перечислением методов, относящихся к заданному классу объектов;

• methods имя класса возвращает перечень методов в отдельном окне. На_ пример:

>> methods char

Methods for class char:

delete diff int

Следующие две функции могут использоваться только внутри конструкторов

классов:

inferiorto('CLASS1','CLASS2',…) и superiorto('CLASS1','CLASS2',…)

Они определяют низший и высший приоритеты классов по отношению к клас_

су конструктора.

Для дескрипторов перегружаемых функций существует функция functions.

F=functions(дескриптор функции), возвращающая массив структур

F.METHODS, вложенный в массив F, при этом именем поля в массиве

F.METHODS является имя класса, а значением поля – название метода, который

вызывается тогда, когда входной аргумент принадлежит этому классу.

Дополнительно functions возвращает следующие структуры:

• F.function – строка, используемая для создания дескриптора функции (су_ ществует также отдельная функция func2str для получения этой инфор_ мации и обратная ей функция str2func, превращающая строку в дескрип_ тор функции);

• F.type содержит simple (простая), overloaded (перегружаемая) или

subfunction (подфункция), то есть указывает тип функции;

• F.default указывает путь к тому файлу, который первый в алгоритме поиска

MATLAB и не определен никаким классом;

• функция which имя метода находит загруженный Java_класс и все классы

MATLAB, которым принадлежит данный метод;

• функция which –all имя метода находит все классы, которым принадле_

жит данный метод.

Любой оператор в системе MATLAB можно *переопределить* (то есть сделать

его функцию перегружаемой) путем задания m_файла с новым именем в соответ_

ствующем каталоге классов. В частности, в уроке 8 отмечалось, что все арифмети_

ческие операторы имеют представления в виде соответствующих функций.

11.8. HandleO и inlineOфункции

11.8.1. Задание handle>функции

MATLAB позволяет создавать особые объекты, называемые handle_функциями

(анонимными, или дескрипторными, функциями). В этих функциях могут ис_

пользоваться все объекты структурного программирования. Для конструирова_

ния handle_функции используется описатель в виде единичного символа @. Со_

здадим, к примеру, handle_функцию с именем fhsin, вычисляющую значение

синуса:

```
>> fhsin=@sin
fhsin =
@sin
```

В том, что эта функция вовсе не обычная функция sin(*x*), убеждает следующий

пример на неудачную попытку вычисления значения sin(1):

```
>> fhsin(1)
ans =
@sin
```

Нетрудно заметить, что вычисления не произошло и просто выдано определе_

ние handle_функции.

Итак, handle_функция характеризуется своим именем – в нашем случае fhsin.

Однако она не имеет списка параметров в круглых скобках (в общем случае

handle_функция может иметь несколько аргументов). На имена handle_функций

накладываются те же ограничения, что и на имена функций в виде m_файлов.

С помощью команд save и load можно записывать handle_функции на диск и счи_

тывать их в рабочую область.

11.8.2. Вычисление и применение handle>функций

Для вычисления handle_функций служит функция

feval(fhandle, a1, a2, a3,…)

Здесь – имя handle_функции без описателя @, a1, a2, a3, … – список аргументов

handle_функции. Handle_функции иногда называют *анонимными функциями*.

Теперь мы можем вычислить нашу handle_функцию:

```
>> feval(fhsin,1)
ans = 0.8415
```

Можно также построить график handle_функции, например в нашем случае используя команду:

```
>> plot(feval(fhsin,0:.01:2*pi))
```

Handle_функция часто используется в качестве функции пользователя – см. примеры в уроке 1, помещенные в разделе функций пользователя.

11.8.3. Inline>функции

Как уже отмечалось в уроке 1, в MATLAB имеется еще один важный класс функций,

способных задавать функции пользователя. Это так называемые inline_функции:

```
g = inline(expr) g = inline(expr,arg1,arg2,...)
g = inline(expr,n)
```

Здесь expr – выражение, a1, a2, … – аргументы, n – число параметров вида p1,

p2, ….

В функциях пользователя особенно удобно задавать expr в виде строкового выражения в апострофах. Например:

```
>> sc2=inline('sin(x).^2+cos(y)^.2')
sc2 =
Inline function:
sc2(x,y) = sin(x).^2+cos(y).^2
```

В апострофах определения inline могут быть записаны любые допустимые ма_

тематические выражения, что открывает простор для задания функций пользова_

теля на основе этих выражений.

11.8.4. Преобразования handle> и inline>функций

На основе inline_функций можно создавать handle_функции. Например:

>> fh=@sc2;

Для преобразования handle_функции в функцию, заданную именем, использу_

ется функция преобразования func2str. Пример:

>> func2str(fhsin)

ans =

sin

Для обратного преобразования (функции, заданной именем, в handle_функ_

цию) служит функция преобразования str2func, например:

>> str2func('cos')

ans =

@cos

Для дополнительного знакомства с особенностями применения handle_функций стоит просмотреть материалы и примеры по ним в справке MATLAB.

Лекция № 9

Управляющие структуры

9.1. Работа с файлами изображений

Информация о графическом файл

Функция info = imfinfo(filename, fmt) возвращает структуру info, в полях которой содержится информация об изображении в графическом файле с именем filename и форматом fmt. Файл с именем filename должен находиться в текущей директории или директории с явно указанным путем, в противном случае будет произведен поиск файла с именем filename и расширением fmt. Отсутствие файла приведет к сообщениям об ошибках при попытке использования функций для работы с файлами.

Возможные значения для аргумента fmt:

Формат	Тип файла
'bmp'	Windows Bitmap (BMP)
'hdf'	Hierarchical Data Format (HDF)
'jpg' или 'jpeg'	Joint Photographic Experts Group (JPEG)
'pcx'	Windows Paintbrush (PCX)
'tif' или 'tiff'	Tagged Image File Format (TIFF)
'xwd'	X Windows Dump (XWD)

Если filename – TIFF или HDF-файл, содержащий более одного изображения, то info представляет собой массив структур с отдельным элементом для каждого изображения в файле. Например, info(3) будет в таком случае содержать информацию о третьем изображении в файле. Множество полей в info зависит от конкретного файла и его формата.

Однако первые девять полей всегда одинаковы.

Поле	Значение
Filename	Имя файла (если файл находится не в текущей директории, строка содержит полный путь к файлу
FileModDate	Дата последнего изменения файла
FileSize	Целое число, указывающее размер файла в байтах
Format	Формат файла, заданный параметром fmt (для JPEG и TIFFфайлов возвращается значение, состоящее из трех символов)
FormatVersion	Версия формата
Width	Ширину изображения в пикселях
Height	Высота изображения в пикселях
BitDepth	Число битов на пиксель
ColorType	Тип изображения: 'truecolor' для RGB-изображения, 'grayscale' для полутонового изображения или 'indexed' для изображения с индексированными цветами

Функция info = imfinfo(filename) возвращает информацию о формате файла по его содержанию.

```
>> imfinfo('moon.png')
ans =
Filename: 'C:\Program Files\MATLAB704\toolbox\images\imdemos\saturn.png'
FileModDate: '13-Oct-2002 09:47:58'
FileSize: 1166148
Format: 'png'
FormatVersion: []
Width: 1200
Height: 1500
BitDepth: 24
ColorType: 'truecolor'
FormatSignature: [137 80 78 71 13 10 26 10]
Colormap: []
Histogram: []
InterlaceType: 'none'
Transparency: 'none'
SimpleTransparencyData: []
BackgroundColor: []
RenderingIntent: []
Chromaticities: []
Gamma: []
XResolution: []
YResolution: []
ResolutionUnit: []
XOffset: []
YOffset: []
OffsetUnit: []
SignificantBits: []
ImageModTime: '24 Jun 2002 19:02:45 +0000'
Title: []
Author: []
Description: 'Voyager 2 image, 1981-08-24,
NASA catalog #PIA01364'
Copyright: []
CreationTime: []
Software: []
Disclaimer: []
Warning: []
Source: []
Comment: []
OtherText: []
```

Чтение изображения из файла

Функция A = imread(filename, fmt) читает из файла с именем filename полутоновое или полноцветное изображение и создает A. Если исходное изображение полутоновое, то A – двумерный массив, если исходное изображение полноцветное, то A – трехмерный массив размера m×n×3.

Другие формы этой функции:

• [A, map] = imread(filename,fmt) читает из файла с именем filename палитровое изображение в массив A с цветовой картой map;

• [...] = imread(filename) пытается определить информацию о формате файла по его содержанию. Параметры filename и fmt были подробно рассмотрены в описании функции imfinfo;

• [...] = imread(...,idx) читает одно изображение из TIFF-файла. idx – целое число – номер изображения по порядку.

Запись изображения в файл

Для *записи массива с изображением* в файл служит функция imwrite. Она имеет следующие формы.

• imwrite(A,filename,fmt) записывает изображение в файл с именем filename в формате fmt из массива A. A может быть матрицей размера $M \times N$ для полутонового изображения, и массивом размера $M \times N \times 3$ для полноцветного изображения. Если A относится к классу uint8 или uint16, то функция imwrite записывает фактические значения из массива в файл.

Если A относится к классу double, то функция imwrite перемасштабирует значения в исходном массиве перед записью по формуле uint8(round(255*A)).

При этом числа с плавающей запятой в диапазоне [0,1] преобразуются в 8_битовые целые числа в диапазоне [0,255].

• imwrite(X,map,filename,fmt) записывает палитровое изображение в файл с именем filename в формате fmt из массива X и соответствующей цветовой карты map. Если X относится к классу uint8 или uint16, то функция imwrite записывает фактические значения из массива в файл. Если X относится к классу double, то функция imwrite смещает значения в исходном массиве перед записью по формуле uint8(X–1). Массив map должен быть цветовой картой MATLAB класса double, функция imwrite перемасштабирует исходные значения массива map по формуле uint8(round(255*map)).

Большинство графических файлов не поддерживают цветовых карт с количеством ячеек больше, чем 256.

• imwrite(...,filename) аналогична описанным выше функциям, но формат файла определяется по расширению filename.

Параметры, используемые при записи графических файлов функцией imwrite.

Параметр	Значение
Параметры для JPEG_файлов	
'Quality'	Число между 0 и 100; чем больше число, тем выше качество файла (меньше искажений при сжатии файла) и тем больше его размер

```
I=imread('moon.tif');
imwrite(I,'moon.jpg', 'Quality',20)
imfinfo('kids.jpg')
ans =
Filename: 'moon.jpg'
FileModDate: '31–Jul–2001 19:02:52'

FileSize: 3251
Format: 'jpg'
FormatVersion: ''
Width: 318
Height: 400
BitDepth: 8
ColorType: 'grayscale'
FormatSignature: ''
```

Здесь уместно сказать, что новый файл помещается в директорию WORK. Исходный цветной файл превратился в полутоновый типа grayscale.

В целом MATLAB содержит минимальный набор средств для работы с файлами изображений. Используя матричные операции системы MATLAB, можно организовать обработку файлов изображений. Профессионально такая обработка организована в ряде пакетов расширения MATLAB: Image Processing Toolbox, Video and Image Processing Blockset, Wavelet Toolbox и др.

9.2. Работа со звуковыми данными

Функции для работы со звуками

Изначально система имела единственную звуковую команду.

• sound(Y,FS) воспроизводит сигнал из вектора Y с частотой дискретизации FS с помощью колонок, подключенных к звуковой карте компьютера. Компоненты Y могут принимать значения в следующих пределах: –1.0<=y<=1.0. Для воспроизведения стереозвука на допускающих это компьютерных платформах Y должен быть матрицей размера *Nx*2.

• sound(Y) функционирует аналогично, принимая частоту дискретизации по умолчанию равной 8192 Гц.

• sound(Y,FS,BITS) функционирует аналогично с заданием разрядности звуковой карты BITS=8 или BITS=16.

Функции звука в MATLAB 6.1/6.5

Дополнительные команды воспроизведения звука:

• soundsc(Y,…) масштабирует и воспроизводит сигнал из массива Y. По синтаксису команда аналогична sound(Y,…);

• soundsc(Y,…,SLIM) аналогична предшествующей команде, но позволяет задать параметр SLIM = [SLOW SHIGH], определяющий тот диапазон значений Y,

который будет соответствовать полному динамическому диапазону звука. По умолчанию SLIM = [MIN(Y) MAX(Y)];

• beep on или off, соответственно, разрешают или запрещают гудок;

• s=beep возвращает состояние on|off;

• beep при s=on издает гудок.

Кроме того, введены команды для считывания и записи файлов звукового формата **.WAV**, стандартного для операционных систем класса Windows:

• wavwrite(Y,WAVEFILE) записывает файл типа WAVE под именем WAVEFILE. Данные по каждому каналу в случае стерео записываются в разных столбцах массива. Величины должны быть в диапазоне [–1;1];

• wavwrite(Y,FS,WAVEFILE) делает то же с заданием частоты дискретизации FS (в герцах);

• wavwrite(Y,FS,NBITS,WAVEFILE) делает то же с заданием числа бит на отсчет NBITS, причем NBITS<=16;

• Y=wavread(FILE) считывает файл типа WAVE с именем FILE и возвращает данные в массиве Y;

• [Y,FS,BITS]=wavread(FILE) считывает файл типа WAVE с именем FILE и возвращает массив данных Y, частоту дискретизации FS (в герцах) и разрядность BITS кодирования звука (в битах);

• […]=wavread(FILE,N) возвращает только первые *N* отсчетов из каждого канала файла;

• […]=wavread(FILE,[N1 N2]) возвращает только отсчеты с номерами от N1 до N2 из каждого канала;

• SIZ=wavread(FILE,'size') возвращает объем аудиоданных в виде вектора SIZ=[samples channels] (samples – число отсчетов, channels – число каналов);

• auwrite записывает файл в соответствии со звуковым форматом фирм Sun и Next; auread воспроизводит файлы в MATLAB 6 на Sun и в MATLAB 5 на Next.

8.3. Общение MATLAB с операционной системой

Работа с папками

Общение системы MATLAB с операционной системой MS_DOS многим покажет_
ся рудиментарной возможностью. Так, во время написания данной книги подоб_
ное общение вообще не потребовалось. Однако это очень важно для систем, рабо_
тающих в реальном масштабе времени, причем наличие наряду с командами,
вводимыми знаком «!», возможности явного задания ОС (dos, unix, vms) позволя_
ет программировать для ОС на управляющем компьютере, отличающемся от ОС
пользователя MATLAB.

Так что, как говорится, из песни слов не выкинешь – MATLAB позволяет из командой строки пользоваться основными услугами старушки MS_DOS и Windows. Есть возможность общения и с другими операционными системами и
даже с глобальной сетью Интернет, в том числе и с помощью собственного HTML_браузера MATLAB (браузера помощи).

Для перехода в новую папку служит команда cd:

• cd wd – переход в указанную папку wd;

• cd (или произвольное имя переменной ad ad=cd) – возврат строки с пол_ ным именем текущей папки;

• cd .. – переход к папке, родительской по отношению к текущей.

Примеры (предполагается, что MATLAB установлен на диске **E**):

```
>> cd
C:\MATLAB701\work
>> cd C:\MATLAB701\tool
??? Error using ==> cd
Cannot CD to C:\MATLAB701\tool (Name is nonexistent or not a

directory).
>> cd C:\MATLAB701\toolbox
>> cd
C:\MATLAB701\toolbox
```

Для указания пути к текущей папке может использоваться функция pwd:

```
>> pwd
ans =
C:\MATLAB701\toolbox
```

Для получения информации о содержимом текущей папки используется команда dir. Она выводит довольно длинный список имен папок.

11.11.2. Выполнение команд !, dos, unix и vms

Из командной строки MATLAB возможно выполнение команд наиболее распрос_
траненных операционных систем:

• ! команда – выполнение заданной команды из набора операционной сис_
темы, в среде которой установлена система MATLAB;

• dos команда – выполнение заданной команды из набора команд MS_DOS или установленной ОС семейства Windows, в последнем случае команда выполняется в фоновом режиме.

Выведем блокнот Windows для редактирования m_файла.

```
dos 'notepad myfile.m'
```

или

```
[s w]=dos('notepad myfile.m')
```

s=0, когда команда выполнена успешно, в противном случае s=1, w содер_
жит сообщение DOS.

• unix команда – выполнение заданной команды из операционной системы UNIX или UNIX_подобных систем (версии Linux);

• vms команда – выполнение заданной команды из операционной системы VMS (Open VMS).

11.11.3. Общение с Интернетом из командной строки

В последние годы резко возросло значение новых информационных технологий и
Интернета [67, 68]. В частности, из Интернета можно получить файлы обновле_
ния текущей версии MATLAB, файлы различных примеров ее применения, мож_
но получить документацию и справочные данные по функциям системы и даже
квалифицированные ответы на свои вопросы к разработчикам системы.

Для общения с Интернетом служит команда web:

• web спецификация дает связь с Web_сервером.

Полезно отметить, что команда web с параметром –browser (например, web http://www.mathworks.com –browser) вызывает вместо браузера помощи MATLAB браузер HTML, установленный в ваших настройках операционной сис_
темы Windows как браузер по умолчанию.

Примеры применения команды web:

• web http://www.mathworks.com загружает Web_страницу **MathWorks Web** в браузер помощи (команда support сразу открывает страницу тех_
нической поддержки MATLAB);

• web mailto:email_address использует программу для отправки элект_
ронной почты, установленную по умолчанию в настройках операционной системы;

• все формы команды web могут использоваться в функциях. Например, функция s = web('www.mathworks.com', '-browser') запускает бра_ узер Интернет операционной системы и выдает **s=0**, если браузер запущен, даже если браузер Интернет открывает страницу в автономном режиме (off_line) или не может ее найти, **s=1**, если браузер Интернет не был обнару_ жен, **s=2**, если браузер был обнаружен, но не был запущен.

Такой выход в Интернет иначе, чем экзотикой, назвать трудно, благо в Windows 98/Me/2000/NT4/XP есть куда более простые способы выхода в Ин_ тернет [67, 68] – например, браузер Internet Explorer и почтовый клиент Outlook Express. Отнесем эту возможность к числу приятных мелочей, которых в MATLAB очень много.

11.11.4. Некоторые другие команды

Есть еще несколько команд для общения с операционными системами:

• delete name – стирание файла с заданным именем name (имя записывает_ ся по правилам операционной системы);

• getenv('name') – возвращение значения переменной 'name' среды окружения. Пример:

```
>> getenv('temp')
ans = C:\TEMP
```

Команда tempdir дает информацию о папке для хранения временных файлов:

```
>> tempdir
ans = C:\TEMP\
```

Еще одна команда – computer – используется в двух формах:

```
>> computer
ans = PCWIN
```

и

```
>> [C S]=computer
C = PCWIN
S = 2.1475e+009__
```

Во втором случае, помимо сообщения о типе компьютера, выводится макси_ мально возможное число элементов в массивах. Оно зависит от объема памяти и свойственных операционной системе ограничений.

Для установки типа терминала может использоваться еще одна команда – terminal. Возможные типы терминалов можно найти в справке по этой команде, выводимой командой help terminal. На этом рассмотрение команд прямого об_

щения с операционными системами можно считать законченным.

М-файлы
Операторы цикла
Операторы ветвления. Логические выражения
Исключительные ситуации
Понятия файлов-сценариев и файлов-функций

Лекция № 10

Среда GUID

10.1. Принципы создания приложений с GUI

Последние версии системы MATLAB приобрели специальные инструментальные средства для визуально ориентированного программирования и проектирования приложений с GUI. Состав этих средств следующий:

• GUIDE – конструктор графического интерфейса;

• Property Inspector – инспектор свойств;

• Object Browser – браузер объекта;

• M-file Editor – редактор М-файлов;

• Component Callbacks – средство создания функций обработки событий для компонентов;

• Figure Callbacks – средства создания функций для обработки событий окон;

• Align Objects – средства выравнивания положения объектов;

• Grid and Rules – средства управления выводом сетки и линейками просмотра;

• Menu Editor – редактор меню;

• The Order Editor – средство изменения порядка активизации компонентов при нажатии клавиши **Tab**.

Эти средства дают достаточный набор возможностей для визуально-ориентированного программирования и проектирования GUI. Такой подход характерен для ряда современных визуально-ориентированных языков программирования, таких как Visual BASIC, Visual C и др., и существенно облегчает создание приложений с GUI.

Главным средством в визуально ориентированном проектировании GUI является приложение GUIDE (GUI Designer). При работе с инструментом GUIDE можно создавать окна GUI путем выбора мышью нужных элементов управления и перемещения их в окно GUI. Таким образом, могут создаваться различные элементы интерфейса, например кнопки, раскрывающиеся списки, линейки прокрутки и т. д.

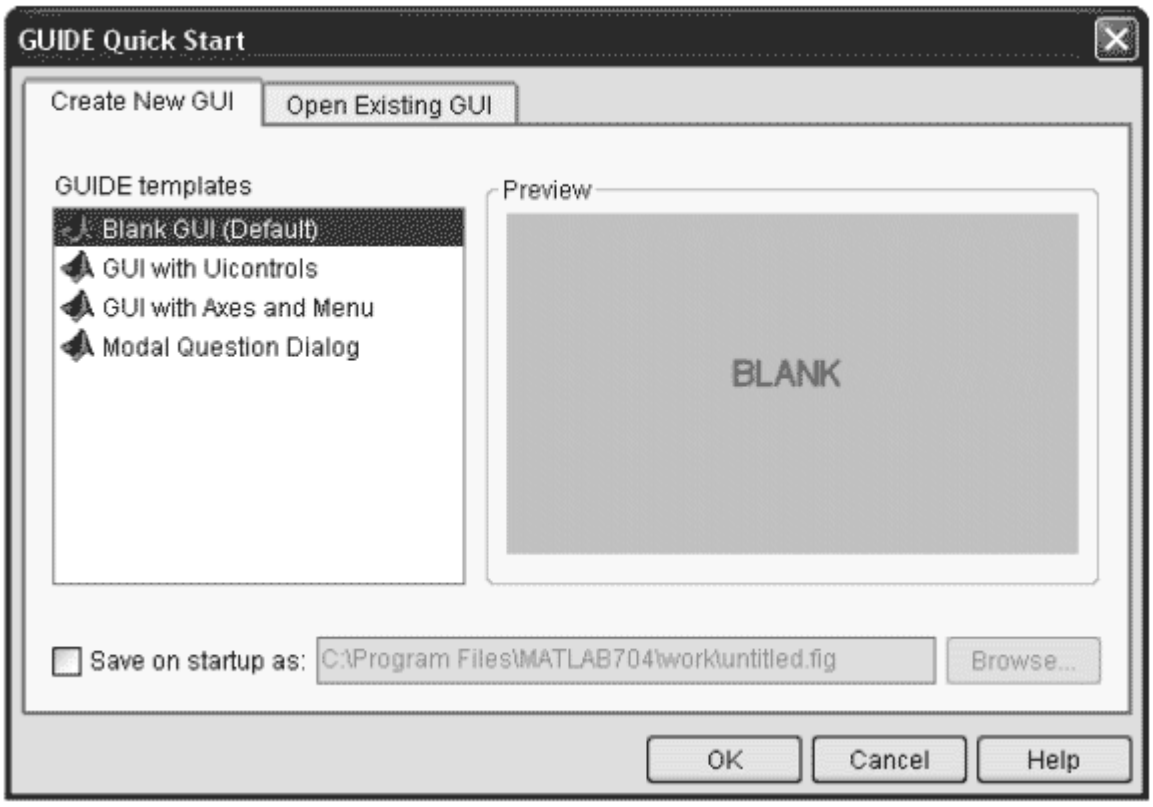
Приложение с GUI может состоять из одного окна (основного) или нескольких окон и осуществлять вывод графической и текстовой информации как в основное окно приложения, так и в отдельные окна. MATLAB имеет ряд функций создания стандартных диалоговых окон для открытия и сохранения файлов, печати, выбора шрифта для текстовых объектов, создания окон для ввода данных и др. Эти средства (объекты) можно использовать в приложениях пользователя.

10.2. Открытие окна инструмента GUIDE

Для открытия окна инструмента GUIDE используется команда

```
>> guide
```

В Отрівшемся окне представлены две вкладки:
• **Create New GUI** – создание нового приложения с GUI;
• **Open Existing GUI** – открытие уже существующего приложения с GUI.



На вкладке создания нового приложения **Create New GUI** имеется список из четырех щаблонов:

• **Blank GUI** – пустое окно приложения,

• **GUI with Uicontrols** – заготовка с кнопками, переключателями и областями ввода, иллюстрирующая создание приложения с GUI вычислительного характера;

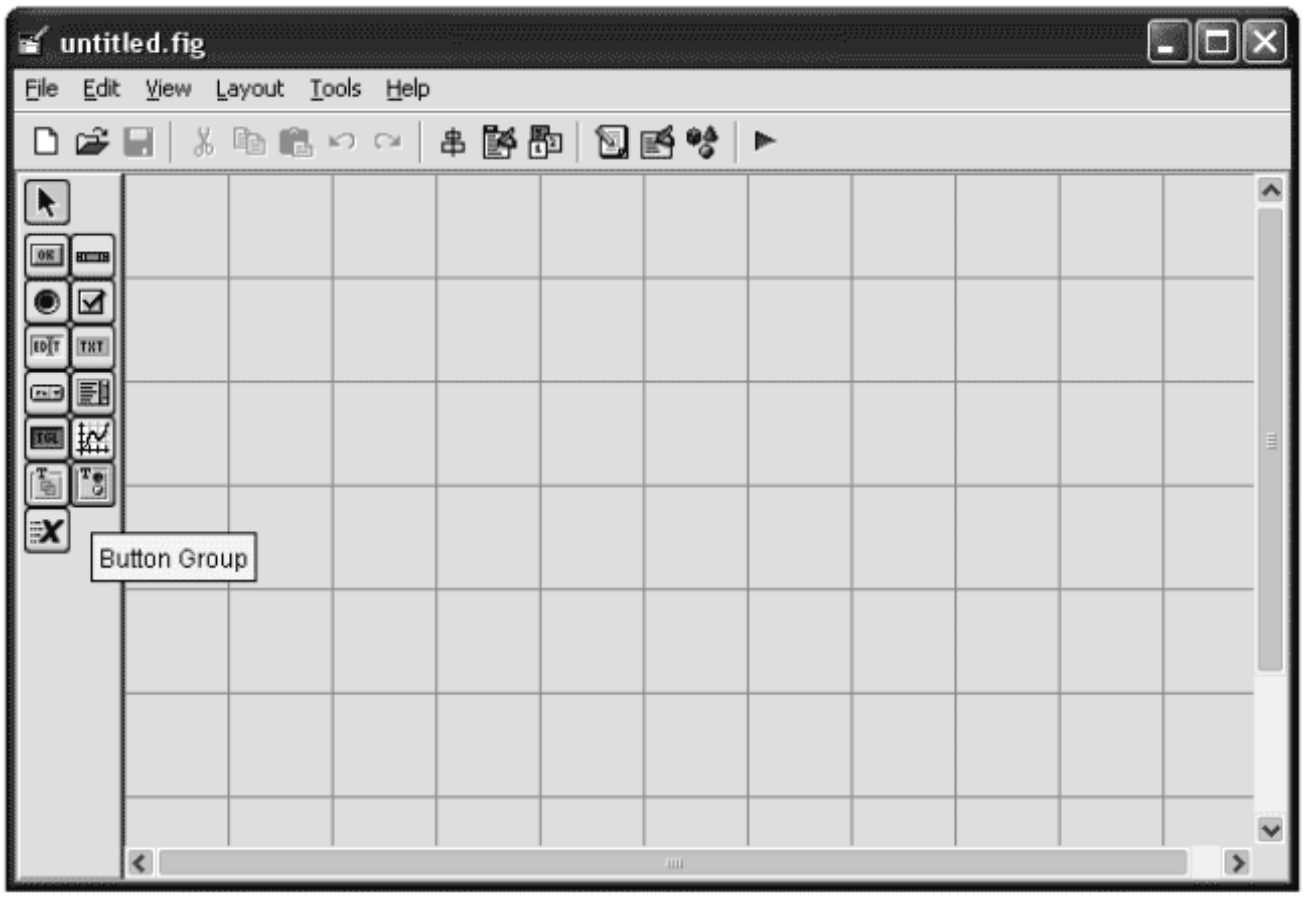
• **GUI with Axes and Menu** – заготовка с осями, меню, кнопкой и раскрывающимся списком, иллюстрирующая создание приложения с GUI графического характера;

• **Modal Question Dialog** – заготовка для модального окна, иллюстрирующего простейший диалог.

12.1.3. Окно создания нового приложения с GUI

Основное окно среды GUIDE, содержит поле окна приложения, вертикальную панель инструментов для добавления элементов интерфейса, горизонтальную панель инструментов и обычное меню.

Горизонтальная панель инструментов позволяет работать с окном приложения с GUI, не обращаясь к его меню.



На этой панели расположены следующие кнопки:

10) **Menu Editor** – вывод окна редактора меню;

11) **Tab Order Editor** – вывод окна редактора порядка обвода элементов клавишей **Tab**;

12) **M-file Editor** – вывод окна редактора М-файлов;

13) **Property Inspector** – вывод окна инспектора свойств;

14) **Object Browser** – вывод окна браузера объектов;

15) **Run** – запуск созданного приложения с GUI.

Вертикальная панель с объектами GUI и номерами кнопок содержит следующие кнопки:

1) **Select** – селекция (выбор) объектов создаваемого окна GUI;

2) **Push Button** – обычная кнопка;

3) **Radio Button** – радиокнопка (беспроводная);

4) **Edit Text** – кнопка ввода и редактирования текста;

5) **Pop_up Menu** – открывающийся список;

6) **Toggle Button**– кнопка_переключатель;

7) **Panel** – панель;

8) **ActiveX Component** – компонент ActiveX;

9) **Slider** – линейка прокрутки (слайдер);

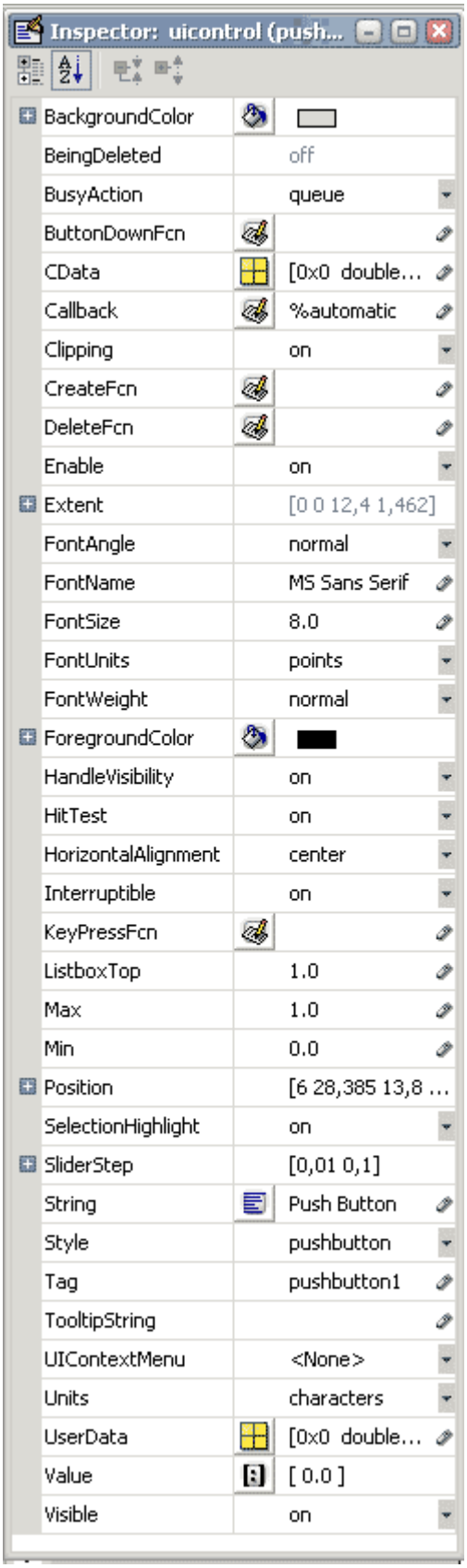
10) **Check Box** – область задания опции (флаг);

11) **Static Text** – область ввода текста;

12) **Listbox** – список;

13) **Axes** – оси графика;

14) **Button Group** – кнопка для группы объектов.



10.3. Свойства объектов GUI

Каждый объект (компонент) GUI имеет ряд свойств. Их полный набор задается таблицей из 41 свойства.

BackgroundColor - Цвет фона (серый по умолчанию, выбор Вектор [RGB] из окна выбора цветов) [0.753, 0.753, 0.753]

BeingDeleted - Признак возможности удаления on – включено, off – выключено

BusyAction - Реакция на возникновение нового прерывания при обработке текущего события в очередь queone – поставить, cancel – проигнорировать

ButtonDownFc - Указатель на функцию обработки нажатия правой клавиши мыши

Cdata - Массив для хранения изображения, Тип truecolor нанесенного на поверхность объекта

Callback - Указатель на функцию обработки события

Children - Массив указателей на потомков

Clipping - Признак отсечения при выходе изображения on – включено, off – за границы объекта выключено

CreateFn - Указатель на функцию обработки события «Создание объекта»

DeleteFn - Указатель на функцию обработки события «Удаление объекта»

Enable - Признак доступа к объекту on – доступ есть, off – доступа нет

Extend - Габаритный прямоугольник

FontAngle - Признак наклона букв normal, italic, oblicue

FontName - Наименование шрифта

FontSize - Размер шрифта Список размеров шрифтов Windows

FontUnits - Единицы измерения шрифтов inches, centimeterts, normalized, pixels, points (пункт = 1/72 дюйма)

FontWeight - Толщина контура букв normal, light, demi, bold

ForegroundColor - Цвет рисования Вектор [RGB] [0.0, 0.0, 0.0] – черный

HandleVisibility - Признак видимости указателя обработчика on – доступен, off – событий не доступен

HitTest - Признак разрешения поиска объекта on – разрешен, off – по значению свойства не разрешен

HorizontalAlligment - Выравнивание надписей в поле объекта left, right, center по горизонтали

Interruptible - Признак разрешения на прерывание on – разрешено, обработчика событий off – не разрешено

KeyPressFcn - Указатель на функцию обработки события «Нажата кнопка» в момент, когда компонент находится в фокусе

ListBoxTop - Индекс строки раскрывающегося списка, 0 которая отображается в верхнем окне

Max - Максимальное значение свойства Value 1

Min - Минимальное значение свойства Value 0

Perent - Указатель на родительский объект

Position - Позиция объекта в виде вектора, задающего координаты нижнего левого угла объекта, его ширину и высоту в заданных единицах

Selected - Признак выбора объекта on – выбран, off – не выбран

SelectionHighlight - Признак повышенной яркости для выделения on – включено, off – элемента в объекте выключено

SliderStep Вектор малых и больших перемещений ползунка слайдера

String - Символьная строка, задающая надпись или значение, приписанное объекту

Style - Символьная строка с типом объекта

Tag Тег – символьная строка с именем объекта

TooltipString - Текст всплывающей подсказки

Type - Класс объекта

UIContextNenu - Всплывающее меню объекта

Units - Единицы измерения линейных величин inches, centimeterts,

(зависят от типа объекта) normalized, pixels, points (пункт =1/72 дюйма)

UserData - Массив данных, ассоциированный [ ] – пустой массив с объектом

Value - Значение, характерное для данного объекта

Visible - Признак видимости объекта on – виден, off – не виден

Этот набор свойств присущ не обязательно каждому объекту. Некоторые из свойств данного объекта могут быть для него не имеющими смысла.

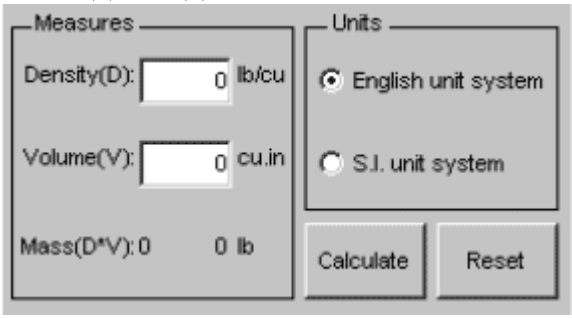
оздание нового окна с GUI сводится к переносу объектов из панели объектов в окно GUI будущего окна с приложением пользователя и указанием их свойств.

Уместно отметить, что каждое окно приложения с GUI описывается двумя файлами с расширениями .fig и .m. Первый файл описывает фигуру – окно со всеми ее деталями, а второй – программу обработки событий. Последний файл создается и редактируется в редакторе М-файлов системы MATLAB.

12.2. Работа с заготовками примеров

Пример вычисления массы вещества.

Рассмотрим пример, представляющий окно приложения с GUI, позволяющего вводить плотность и объем вещества, а также систему единиц для измерения массы, вычисляемой как произведение плотности вещества на его объем. Пример предусматривает также применение двух кнопок для вычисления массы Calculate и сброса исходных данных.



Открыв сгенерированный системой М-файл, можно увидеть программу в окне редактора М_файлов:

```
function varargout = demo1(varargin)
     gui_Singleton = 1;
     gui_State = struct('gui_Name', mfilename, ...
     'gui_Singleton', gui_Singleton, ...
     'gui_OpeningFcn', @demo1_OpeningFcn, ...
     'gui_OutputFcn', @demo1_OutputFcn, ...
     'gui_LayoutFcn', [] , ...
     'gui_Callback', []);

     if nargin && ischar(varargin{1})
          gui_State.gui_Callback = str2func(varargin{1});
     end
     if nargout
          [varargout{1:nargout}]= gui_mainfcn(gui_State, ...
          varargin{:});
     else
          gui_mainfcn(gui_State, varargin{:});
     end

function demo1_OpeningFcn(hObject, eventdata, handles, varargin)
     handles.output = hObject;
     guidata(hObject, handles);
     initialize_gui(hObject, handles, false);

function varargout = demo1_OutputFcn(hObject, eventdata,
handles)
     varargout{1} = handles.output;

function density_CreateFcn(hObject, eventdata, handles)
     usewhitebg = 1;
```

```
        if usewhitebg
             set(hObject,'BackgroundColor','white');
        else
             set(hObject,'BackgroundColor',get(0, ...
             'defaultUicontrolBackgroundColor'));
        end

    function density_Callback(hObject, eventdata, handles)
        density = str2double(get(hObject, 'String'));
        if isnan(density)
             set(hObject, 'String', 0);
             errordlg('Input must be a number','Error');
        end
        handles.metricdata.density = density;
        guidata(hObject,handles)

    function volume_CreateFcn(hObject, eventdata, handles)
        usewhitebg = 1;
        if usewhitebg
             set(hObject,'BackgroundColor','white');
        else
             set(hObject,'BackgroundColor',get(0, ...
             'defaultUicontrolBackgroundColor'));
        end

    function volume_Callback(hObject, eventdata, handles)
        volume = str2double(get(hObject, 'String'));
        if isnan(volume)
             set(hObject, 'String', 0);
             errordlg('Input must be a number','Error');
        end
        handles.metricdata.volume = volume;
        guidata(hObject,handles)

    function calculate_Callback(hObject, eventdata, handles)
        mass = handles.metricdata.density * ...
        handles.metricdata.volume;
        set(handles.mass, 'String', mass);

    function reset_Callback(hObject, eventdata, handles)
        initialize_gui(gcbf, handles, true);

    function    unitgroup_SelectionChangeFcn(hObject,    eventdata,
handles)
        if (hObject == handles.english)
             set(handles.text4, 'String', 'lb/cu.in');
             set(handles.text5, 'String', 'cu.in');
             set(handles.text6, 'String', 'lb');
        else
             set(handles.text4, 'String', 'kg/cu.m');
             set(handles.text5, 'String', 'cu.m');
             set(handles.text6, 'String', 'kg');
        end
```

```
function initialize_gui(fig_handle, handles, isreset)
     if isfield(handles, 'metricdata') && ~isreset
          return;
     end
     handles.metricdata.density = 0;
     handles.metricdata.volume = 0;
     set(handles.density, 'String', handles.metricdata.density);
     set(handles.volume, 'String', handles.metricdata.volume);
     set(handles.mass, 'String', 0);
     set(handles.unitgroup, 'SelectedObject', handles.english);
     set(handles.text4, 'String', 'lb/cu.in');
     set(handles.text5, 'String', 'cu.in');
     set(handles.text6, 'String', 'lb');
     guidata(handles.figure1, handles);
```

Созданную программу можно запустить как указанием ее имени в командной строке MATLAB:

>> demo1

так и командой **Run** из окна редактора М-файлов (см. рис. 12.16). При запуске появляется окно GUI данной программы.

12.3. Программирование событий

По умолчанию, приложение содержится в двух файлах: с расширением fig (графическое окно с размещенными на нем элементами управления) и с расширением m (файл-функция hello с подфункциями, которые обрабатывают различные события, возникающие в ходе взаимодействия приложения с пользователем).

Для того, чтобы создать функцию, обрабатывающую определенное событие элемента управления в контекстном меню элемента управления нужно выбрать в пункте View Callbacks подпункт с названием события, например, Callback. При этом происходит переход в редактор М-файлов к подфункции обработки события Callback, заголовок которой и комментарии генерируется автоматически:

После заголовка требуется разместить те операторы, которые будут выполняться при срабатывании данного события.

Смысл входных аргументов функции Callback следующий.

· Аргумент hObject содержит указатель на кнопку Hello, т.е. объект Uicontrol с тегом btnHello (он нам сейчас не понадобится).

· Аргумент eventdata зарезервирован для использования в следующих версиях MatLab.

· Аргумент handles является структурой с указателями на все объекты приложения. Поля структуры handles являются тегами этих объектов. Так handles.btnHello содержит указатель на кнопку Hello, handles.figure1 - указатель на

окно приложения, handles.txtWin - указатель на область вывода текста (как раз этот указатель нам сейчас и пригодится).

Свойства графических объектов задаются при помощи функции set.

```
set(handles.edit1, 'String', volume);
```

Получить значение любого свойства графических объектов можно при помощи функции get.

```
volume = get(handles.edit1,'String');
```

Лекция № 14

Символические вычисления

14.1. Символические переменные и функции

В предыдущих лекциях рассматривались численные процедуры MATLAB, выполняемые с аргументами, заданными в виде чисел или числовых массивов. В состав MATLAB входит ToolBox Symbolic Math, который добавил к системе возможность символьных вычислений. Помимо типовых аналитических вычислений (таких как упрощение математических выражений, подстановка, нахождение пределов, дифференцирование, интегрирование, интегральные преобразования, получение решений уравнений и систем уравнений, включая дифференциальные и т.д.) пакет Symbolic позволяет реализовывать арифметические операции с любой точностью. Функции пакета Symbolic Math реализуют интерфейс между средой MATLAB и ядром системы символьной математики Maple, причем работа в MATLAB не требует установки Maple. Последняя версия системы символьной математики Maple 7 в своем ядре и в расширениях имеет около 3000 функций. Система MATLAB с пакетом Symbolic, включающем в себя около сотни символьных команд и функций, намного уступает Maple по количеству таких команд и функций. Однако, в данный пакет включены лишь наиболее важные и широко распространенные функции. Расширение ToolBox позволяет пользователям, имеющим опыт работы в Maple, использовать ресурсы ядра Maple практически в полном объеме.

Для получения полного списка функций ToolBox Symbolic Math нужно ввести команду:

```
>> help symbolic
```

Наиболее общие из них будут рассмотрены ниже.
Символический объект создается при помощи функции syms, например:

```
>> syms x
```

Создает символическую переменную x. Символические переменные можно также создавать так:

```
>> x=sym('x');
```

Конструирование символических функций производится при помощи обычных арифметических операций над уже существующими символьными переменными и функциями:

```
>> f = sin(5*x)
```

Символическую функцию можно задать без предварительного объявления переменных при помощи функции syms:

```
>> z = sym('c^2/(d+1)')
```

Команда pretty(z) выводит символьную функцию в более привычном виде:

```
>> pretty(z)
```

14.2. Графическое представление функций

Визуализация символической функции одной переменной осуществляется при помощи команды ezplot:

```
>> f=sym('x^2*sin(x)');
>> ezplot(f)
```

Вторым аргументов функции может быть задан вектор с границами отрезка, на котором требуется построить график.

```
>> ezplot(f, [-3 2])
```

ToolBox Symbolic Math содержит также функции работы с трехмерными графиками ezmesh, ezplot3, ezsurf.

14.3. Преобразование в численные значения

Вычисления в символьном виде позволяют получать значение символьных переменных с любой точностью. Фукция vpa предназначена для вычисления символических выражений:

```
>> c=sym('sqrt(2)');
>> cn=vpa(c,70);
```

По умолчанию Выводится 32 цифры. Второй необязательный параметр задает количество значящих цифр.

Для переводя символьной переменной в числовую используется функция double:

```
>>cnum=double(cn)
```

14.4. Дифференцирование

Пример 1. Производная простой функции.

```
>> syms x
>> f = sin(5*x)
>> diff(f)
>> ans =
5*cos(5*x)
```

Пример 2. Производная сложной функции.

```
>> g = exp(x)*cos(x)
>> diff(g)
>> ans =
exp(x)*cos(x) - exp(x)*sin(x)
```

Пример 3. Вторая производная функции.

```
diff(g,2)
ans =
-2*exp(x)*sin(x)
```

Тот же результат дает последовательное применение операции дифференцирования:

```
>> diff(diff(g))
>> ans =
-2*exp(x)*sin(x)
```

Пример 4. Производная костанты.

```
>> c = sym('5');
>> diff(c)
>> ans =
0
```

14.5. Пределы

Пример 1. Вычисление предела в 0.

```
>> syms h x
>> limit((cos(x+h) - cos(x))/h, h, 0)
>> ans =
-sin(x)
```

Пример 2. Вычисление предела в бесконечности.

```
>> limit((1 + x/n)^n, n, inf)
>> ans =
```

```
exp(x)
```

Пример 3. Вычисление одностороннего предела функции x/|x|, в точке 0 слева и справа.

```
>> syms x;
>> limit(x/abs(x), x, 0, 'left')
ans =
-1

>> limit(x/abs(x), x, 0, 'right')
ans =
1
```

14.6. Интегрирование

Если f – символьная переменная, то функция int(f) возвращает символьную переменную, соответствующую неопределенному интегралу функции f. Подобно дифференцированию int(f,v) использует переменную v как переменную интегрирования если задана функция нескольких переменных.

Пример 1. Определенный интеграл

```
>> syms y;

>> f = y^(-1);

>> int(f)
>> ans =

log(y)
```

Пример 2. Открытый интеграл

```
>> syms a x
>> f = (a^2 + (x+1)/x^2);
>> g = int(f, x, -inf, inf)
>> g =
Inf
```