Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Радиотехнический факультет (РТФ)

Кафедра средств радиосвязи (СРС)

Кологривов В.А.

Функциональная среда программирования системы MatLab. Справочное пособие

Учебно-методическое пособие
к практическим занятиям и самостоятельной работе
по дисциплине "Информатика"
для студентов радиотехнических специальностей

2012

Кологривов В.А.

Функциональная среда программирования системы **MatLab**. Справочное пособие. Учебно-методическое пособие к практическим занятиям и самостоятельной работе по дисциплине “Информатика”. – Томск: ТУСУР. Образовательный портал, 2012. - 75 с.

Учебное методическое пособие посвящено краткому описанию входного языка и функциональной среды системы для инженерного и научного проектирования **MatLab**.

В пособии по системе **MatLab** описаны: режимы работы, типы данных, стандартные операции, операторы языка программирования, правила написания функций пользователя, доступные функции - математические, обработки строк, векторно-матричные, работы с полиномами, дифференцирования, интегрирования, аппроксимации, интерполяции, решения нелинейных уравнений, преобразования Фурье, построения графиков, работы с файлами и так далее.

Пособие предназначено для практических занятий и самостоятельной работы студентов очной формы обучения высшего специального образования, по направлениям: «Радиотехника», «Телекоммуникации» и др.

Аннотация

В пособии дано краткое руководство по функциональной среде системы для инженерных и научных расчетов **MatLab 4.0/5.x.**

В систематизированном виде представлены актуальные сведения о составе и структуре системы **MatLab,** входном языке, режимах и элементах программирования, содержании функциональной среды.

Должное внимание уделено функциям работы с массивами данных, векторно-матричным операциям и функциям, обработке строковых данных, графическим возможностям системы, работе с файлами данных.

Справочное пособие предназначено для практического изучения функциональной среде системы **MatLab,** ориентированной на инженерные и научные исследования и широко используемой в учебном процессе крупнейших университетов.

Пособие предназначено для студентов радиотехнических специальностей.

Содержание

Предисловие

Научным сотрудникам, инженерам, исследователям, аспирантам и студентам часто приходится выполнять численные эксперименты, производить обработку результатов экспериментальных исследований, просчитывать характеристики инженерных решений, выполнять расчеты, связанные с дипломными и курсовыми работами. В каждой конкретной области, как правило, существуют специализированные пакеты и системы, позволяющие выполнять численное моделирование процессов и устройств.

На практике в тоже время встречаются задачи, не укладывающиеся в традиционные рамки, либо не требующие привлечения сложных универсальных моделей, либо наоборот связанные с разработкой новых более эффективных моделей. В этом случае более эффективным оказывается использование общих математических пакетов и систем, позволяющих численно либо аналитически решать традиционные математические задачи. Тогда, владея предметной областью и имея достаточную математическую подготовку, можно интерпретировать и решать прикладные задачи на языке численных математических методов.

Наибольшее распространение в настоящее время получили такие общематематические пакеты (системы) как **MathCad** и **MatLab**, позволяющие выполнять как численные так и аналитические операции.

Пакет **MathCad** не требует написания, как такового традиционного текста программы, все вычисления производятся по мере написания выражений и формул в окне редактирования и сохраняются в файлах специального формата, что не позволяет просмотреть их обычными текстовыми редакторами. Выбор метода решения также в некоторой степени ограничен, так как производится большей частью автоматически. Для работы с пакетом **MathCad** от пользователя требуются минимальные знания, как программирования, так и численных методов.

Система **MatLab** напротив, имеет простой, но достаточно гибкий входной язык программирования, позволяющий писать программы в традиционном виде, которые хранятся в обычных текстовых файлах. Обширные библиотеки стандартных подпрограмм (функций), по существу превращают программу в краткую запись структуры алгоритма. Для эффективной работы с системой **MatLab,** уместны знания как технологии программирования, так и численных методов, так как пользователь может активно влиять на выбор метода решения.

Применительно к **MatLab**, говорят о системе для инженерных и научных расчетов, так как в ее состав кроме ядра входят различные пакеты расширения системы ориентированные на различные приложения.

В связи со сказанным система **MatLab** имеет более широкое применение и пригодна, как для освоения технологии программирования и изучения численных методов, так и для реализации прикладных задач самого широкого

плана. Широкий набор универсальных и весьма эффективных базовых функций, а также наличие специализированных библиотек (пакетов расширения) системы **MatLab** ставит его в разряд наиболее перспективных как для учебных, так и исследовательских целей.

Простой и емкий входной язык, богатая функциональная среда, современная объектная технология программирования и гибкий графический интерфейс **MatLab for Windows** обеспечивает возможность быстрой реализации прикладных программ, не уступающих по своим возможностям, программам реализованным на алгоритмических языках высокого уровня **Си** и **Фортран.**

Учитывая перспективность использования системы **MatLab** в учебном процессе, в 1999 году был составлен электронный вариант данного пособия, который постоянно использовался в качестве дополнительного пособия при изучении дисциплин “Информатика”, “Вычислительная практика”, выполнении лабораторных работ по дисциплинам “Прикладные математические методы в радиотехнике” и “Основы компьютерного проектирования и моделирования устройств РЭС”. Данное издание пособия практически полностью соответствует электронной версии, уточнению подверглась лишь рубрикация разделов и расширен список литературы, посвященный описанию системы **MatLab,** пакетов расширения системы и их применению.

На радиотехническом факультете Томского государственного университета систем управления и радиоэлектроники система **MatLab** находит все более широкое применение. Достаточно отметить, что система **MatLab** и отдельные пакеты расширения системы как средства расчета и моделирования используются в дисциплинах “Информатика”, “Вычислительная практика”, “Теория электрических цепей”, “Теория радиотехнических сигналов”, “Прикладные математические методы в радиотехнике”, “Основы компьютерного проектирования и моделирования устройств РЭС”, “Цифровая обработка сигналов”, “Телекоммуникационные системы передачи информации” и других. Это обстоятельство лишний раз свидетельствует об эффективности данной системы и необходимости разработки соответствующего методического обеспечения.

Введение. Функциональная среда программирования системы MatLab

Цель справочного пособия: Способствовать использованию системы для инженерных и научных расчетов **MatLab** при изучении различного рода дисциплин, в частности при выполнении практических заданий, лабораторных практикумов, курсовых и дипломных работ и проектов.

Задачи пособия: Ознакомиться на практике с использованием функциональной среды системы для инженерных и научных расчетов **MatLab,** составом и структурой системы **MatLab,** входным языком и элементами программирования. Освоить программную реализацию используемых в различных дисциплинах методов анализа расчета и моделирования с использованием функциональной среды системы **MatLab.**

Система **MatLab (Matrix Laboratory)** является интерактивной системой для выполнения инженерных и научных расчетов, ориентированной на работу с массивами данных. Система **MatLab** разработана фирмой **MathWork Inc.** (США, г. Нейтик, штат Массачусетс).

Система содержит встроенную матричную и комплексную арифметику, поддерживает выполнение операций с векторами, матрицами и массивами данных, работу с алгебраическими полиномами, решение нелинейных уравнений и задач оптимизации, интегрирование в квадратурах, решение дифференциальных и разностных уравнений, построение различных видов графиков, трехмерных поверхностей и линий уровней, а также работу с файлами данных.

Систему отличает простой язык программирования и удобная операционная среда из широкого набора стандартных математических функций, позволяющая формулировать проблемы и получать решения, в удобной форме не прибегая к детализации. Простота входного языка и его полнота с точки зрения структурного программирования позволяет реализовать любой алгоритм доступный языкам высокого уровня. Кроме того, допускается подключение процедур и функций написанных на языках **Си** и **Фортран.**

Язык **Matlab** обеспечивает возможность работы с файлами с помощью **Си** подобных функций. Кроме того, в системе реализованы стандартные диалоговые панели для чтения и записи файлов.

MatLab является открытой системой - практически все процедуры и функции доступны не только для использования, но и для коррекции и модификации. Система устроена так что, программируя, пользователь расширяет ее возможности новыми программами, процедурами и функциями доступными наравне со стандартными функциями.

Доступ к регистрам общего пользования системы **MatLab**, позволяет при наличии плат сопряжения компьютера с внешним оборудованием реализовывать системы контроля и управления реального времени.

Современные версии системы включают широкий набор специализированных пакетов расширения - цифровой обработки сигналов, анализа и синтеза линейных систем автоматического управления, интерактивного моделирования динамических систем, ядро символьной математики из известного пакета (системы) **Maple V** и другие.

Возможность аналитических вычислений в системе **MatLab** обеспечивается пакетом расширения **Symbolic Math Toolbox.** Кроме того, известны специализированные пакеты расширения системы для аналитического решения, как обыкновенных дифференциальных уравнений, так и уравнений в частных производных.

Полезным свойством системы **MatLab** является возможность создания текстовых документов редактором **Word** с возможностью проведения в нем вычислений на **MatLab** и фиксированием в **Word-документе** числовых и графических результатов. Создание таких **Word-документов** или **M-книг** обеспечивается пакетом расширения **NoteBook.**

С реализацией системы **MatLab** под **Windows** появилась возможность проектирования графического программ с помощью элементов дескрипторной графики.

К недостатку системы **MatLab** можно отнести отсутствие режима компиляции, что препятствует использованию разработанных программ вне среды системы. В тоже время существуют программы конверторы для перевода программ с языка **MatLab** на язык **Си**, что позволяет частично решить указанную проблему.

Данное справочное пособие посвящено в основном описанию ядра системы **MatLab** версий **4.0/5.x.** Для начального ознакомления с системой этих сведений вполне достаточно. Подробное обозрение функциональной среды ядра достаточно для реализации большинства прикладных задач на основе численных методов, реализованных набором универсальных функций. В пособии также дан подробный обзор функций графического представления результатов численного моделирования и функций файлового ввода - вывода. Кроме того, рассмотрены общие вопросы работы в системе **MatLab,** входной язык программирования и ряд вспомогательных функций организации диалоговых программ в функциональной среде системы **MatLab.**

1 Режимы работы системы MatLab

Система **MatLab** позволяет работать в двух режимах:

диалоговом - при этом, команды и операторы вводятся в командной строке и обрабатываются построчно, с сохранением в памяти предшествующих результатов диалога;

программном - при этом, программа предварительно заносится в файл текстовым редактором и выполняется при вызове в командном окне по имени.

Эти два режима практически не различаются, однако программный режим представляется более удобным, так как не требует повторного ввода серии команд для повторных вычислений. Диалоговый режим целесообразнее использовать для предварительных вычислений и проверки значений переменных, после окончания работы программы с целью отладки.

С помощью клавиш **("вверх", "вниз"**) можно перебирать стек ранее использованных команд для модификации и или повторного выполнения.

1.1 Режимы программирования. Типы файлов. Программа MatLab

Остановимся подробнее на режиме программирования.

Отдельным текстовым редактором, либо встроенным в систему **MatLab,** открывается файл, куда записывается программа. Все файлы в системе **MatLab** имеют расширение **.m.**

Запуск программы осуществляется из командного окна по имени после знака приглашения **(>>)** с последующим нажатием клавиши **Enter.**

В системе **MatLab** для программирования используют и различают два вида файлов:

script файлы - используются для хранения текстов управляющих программ.

function файлы - используются для хранения подпрограмм, оформленных в виде универсальных функций. Имена **function** файлов должны совпадать с именами функций.

Программы, т.е. **script** файлы выполняются в режиме интерпретации, а **function** файлы, если не включен режим **echo**, предварительно компилируются.

Стандартные функции системы **MatLab** делятся на два класса: встроенные в ядро системы и внешние в виде **m-** файлов.

Все имена программ, функций и переменных должны быть уникальны, т.е. не совпадать с именами стандартных функций и переменных. В системе **MatLab** зарезервированы имена следующих переменных:

i, j - мнимая единица, для обозначения мнимой части комплексных чисел и выражений;

pi - число равное **3.141592653598793**;

inf - машинная бесконечность;

NaN - неопределенность типа **0/0** или **inf/inf**;

ans - внутренняя переменная для обозначения непоименованных результатов.

eps – внутренняя вещественная переменная, используемая для оценки сходимости итерационных методов – это наименьшее число, для которого на данной ЭВМ выполняется соотношение **1.0 + eps > 1.0.**

Программа на входном языке **MatLab** представляет собой последовательность команд операторов, представляющих собой либо вызов стандартной функции или переменной по имени, либо определенные операции над переменными.

Команды и операторы отделяются друг от друга знаками **(;)** и **(,)**. Отсутствие знака **(;)** после оператора или вызова функции приводит к выводу значения оператора или значений функции, поэтому при записи нескольких операторов либо команд в одной строке используется альтернативный разделитель **(,)**.

2 Типы данных

Язык **MatLab** практически не требует предварительного описания типов переменных, они определяются автоматически по контексту. Переменные комплексного типа формируются по следующему правилу:

<имя>=<выражение действительной части> (+/-) (i/j)*<выражение мнимой части>, где **i, j** - зарезервированы под обозначение мнимой единицы. В числовом виде допустима запись комплексного числа в виде **z=2.7 + 3.1i**.

Иногда все же требуется уточнить, имеем ли мы дело, например, с вектором столбцом либо строкой. Для этих целей используются стандартные функции заполнения массивов заданной конфигурации нулями либо единицами.

Система **MatLab** ориентирована на работу с массивами, поэтому даже скалярные переменные интерпретируются как частный случай массива. При таком подходе большинство стандартных функций работают как со скалярными переменными, так и с массивами, если это не противоречит здравому смыслу. Иногда при таких операциях производятся действия не соответствующие общепринятому математическому смыслу. Такие ситуации, только обогащают возможности языка, так как существуют иные возможности реализовать действия над массивами в полном соответствии с математическим языком.

Разновидностью массивов состоящих из набора символов являются строковые переменные и константы. Строковая константа представляет собой набор символов заключенный в апострофы. Из строк одинаковой длины можно формировать векторы.

В младших версиях системы **MatLab** допустимы только двумерные массивы - матрицы, но, начиная с версии **5.0,** появились многомерные массивы и более сложные типы данных вплоть до объектов.

3 Стандартные операции

Арифметические операции: сложения **(+)**, вычитания **(-)**, умножения **(*),** деления **(/)**, возведения в степень **(^).**

Логические операции: не **(~)**, и **(&)**, или **(|)**, исключительное или реализована стандартной функцией **(xor).**

Операции отношения: равно **(==)**, не равно **(~=)**, больше **(>),** меньше **(<),** больше или равно **(>=)**, меньше или равно **(<=).**

При работе с массивами знак точки (.) перед знаком операции означает покомпонентные действия, в отличие от операций принятых в математике.

4 Команды общего назначения

Обычно знакомство с системой **MatLab** начинают с команд **ver**, **info**, **help**, **demo**.

Заметим, что система **MatLab** по умолчанию требует подачи команд строчными буквами, так как интерпретатор системы различает прописные и строчные буквы - режим **casesen on**. Для переключения режима различения используется команда **casesen off**. Команда **casesen** используется для перехода от одного режима к другому.

По командам **ver** или **ver <имя ППП>** выдаются номера текущих версий системы **MatLab** либо запрошенного прикладного программного пакета **(ППП).**

По команде **info** выводится информация о системе, фирме производителе, составе **ППП** и так далее.

Команды **help, help <раздел/функция/оператор>, help <спец. символ>** выдают соответственно список разделов, содержание разделов или описание функции или синтаксис оператора и список спецсимволов и логических функций.

Команда **demo** выводит на экран меню демонстрационных программ.

Команда **matlabrc** запускает одноименный файл с исходными установками системы **matlab** и путей доступа. Обычно файл **matlabrc.m** резервируется для системного менеджера. В более поздних версиях системы **MatLab** файл **matlabrc.m** не используется.

Для получения информации о путях доступа и их модификации используется команда **path** и ее разновидности.

Формат вывода значений переменных устанавливается параметрами команды **format.**

Для выдачи результатов, кроме отсутствия знака **(;)** после выражения используются разновидности оператора **disp(<выражение/имя переменной>)** или **disp('текст/пусто').**

Останов для просмотра результатов на указанное время в секундах или до нажатия клавиши **enter** производится командами **pause(n)** или **pause**.

Ввод данных с клавиатуры в диалоге осуществляется командами **input**:

x=input('приглашение/подсказка') - ввод скалярной переменной. Переменная типа массив вводится в квадратных скобках **[ ]** в виде списка компонент. Строковая переменная вводится в апострофах;

x=input('приглашение/подсказка','s') - позволяет ввести строку, не используя апострофы.

Комментарии в программах оформляются с использованием знака **(%)**. Знак **(%)** действует до конца строки.

Знаком переноса длинных выражений служит троеточие **(...)**.

Команды **echo off/on** запрещают или устанавливают режим выдачи команд программы на экран во время выполнения.

Команда **diary <имя файла>** позволяет сохранить сценарий диалога программы и пользователя на диске в файле по указанному имени.

Освобождение рабочей области и памяти от использованных переменных и функций используются разновидности команды **clear**.

Для завершения работы системы используются команды **quit** и **exit**.

Для завершения циклических операторов и выхода из программы без завершения работы системы используется

5 Основные операторы языка программирования

Оператор присваивания.

Синтаксис:

переменная=выражение;.

Под выражением следует понимать совокупность констант, переменных и функций, объединенных знаками операций.

Примеры оператора присваивания:

```
a=5; b=7.889;
c=1+a-b*sin(pi/4);
d=1+i*2; e=3+4i;
f=a+i*b;
```

Условный оператор (условное выражение).

Синтаксис:

```
if <логическое выражение>
  <операторы>
end;
if <логическое выражение>
```

<операторы 1>
else
<операторы 2>
end;
if <логическое выражение 1>
<операторы 1>
elseif <логическое выражение 2>
<операторы 2>
else
<операторы 3>
end;.

Первый вариант условного оператора работает, если логическое выражение истинное, т.е. не равно нулю.

Во втором варианте, если логическое выражение истинно, то выполняется первая группа операторов, а вторая группа пропускается. Ложное значение логического выражения приводит к пропуску первой группы операторов и выполнению второй группы.

В третьем варианте, если первое логическое выражение истинно, то первая группа операторов выполняется и оставшаяся часть оператора пропускается. Если первое логическое выражение ложно, то первая группа операторов пропускается и проверяется истинность второго логического выражения. Если второе логическое выражение истинно, то выполняется вторая группа операторов и оставшаяся часть оператора пропускается. Если все предшествующие логические выражения оказались ложными, то выполняется третья группа операторов стоящая за словом **else**.

Заметим, что альтернатив **elseif** может быть несколько, что позволяет реализовывать много альтернативные конструкции. Напомним, что логическое выражение в данном случае представляет собой совокупность констант, переменных, других выражений и функций, объединенных знаками операций отношения и логических операций.

Оператор цикла - конечный.

Синтаксис:

for v=<выражение-массив>
<операторы>
end;.

Переменная цикла **v** является массивом заданным явно либо в виде арифметической прогрессии. Конструкция **v=m:n** задает компоненты массива от **m** до **n** с шагом **1**. Конструкция **v=m:h:n** задает компоненты массива от m до **n** с шагом **h**. Для принудительного прекращения оператора цикла можно воспользоваться оператором **break**, например **if <логическое выражение> break end;.**

Оператор цикла - условный.

Синтаксис:

while <логическое выражение>

<операторы>

end;.

Цикл выполняется до тех пор, пока логическое выражение истинно. Напомним, что элементы логического выражения должны изменятся в цикле, иначе он никогда не завершится. Для принудительного прекращения оператора цикла также используют оператор **break**, в виде **if <логическое выражение> break end;.**

6 Функции пользователя

Наряду с использованием стандартных функций в системе **MatLab** предусмотрена возможность создания функций пользователя, чем и обеспечивается важное свойство расширяемости системы. Функции хранятся в отдельных **m**- файлах с именами совпадающими с именами функций.

Оформление функции.

Синтаксис:

function [список вых. параметров]=<имя>(список вх. параметров);

% строки комментариев

<пустая строка>

<тело функции>;.

Списки формальных параметров могут включать разнородные по типу переменные. Наличие параметров обеспечивает важное свойство универсальности функций. Списки могут отсутствовать, если функция не имеет входных параметров и или выходных. Разрешается не ставить квадратных скобок, если список выходных параметров содержит один элемент.

За заголовком обычно идут комментарии, раскрывающие назначение функции и расшифровку параметров. После комментариев (интерфейсной части) рекомендуют оставлять, пустую строку, означающую границу выдачи информации по команде **help**. Далее следуют операторы тела функции.

Заметим, что внутренние переменные функции являются локальными, т.е. существуют на время действия функции и удаляются после ее завершении. Имена локальных и внешних переменных не пересекаются.

Вызов функции осуществляется следующим образом:

[список выходных параметров]=<имя>(список входных параметров);.

Здесь используются списки фактических параметров. Должно соблюдаться соответствие формальных и фактических параметров по числу, типу и порядку следования.

Вложенный вызов функций допустим, что позволяет реализовать новые функции с использованием ранее разработанных или стандартных.

7 Дополнительные функции организации программ

Рассмотрим некоторые важные стандартные функции, существенно расширяющие возможности организации программ.

Функция контекстной подстановки eval.

Синтаксис:

eval('<выражение>');

x=eval('<выражение>');.

Первый вызов обеспечивает перевод строковой переменной типа **t=<'выражение'>** в команду, оператор, выражение или его часть, и подстановку в виде текста программы с последующим выполнением. Второй вызов обеспечивает перевод текста во фрагмент программы и присваивание результатов вычислений переменной **x.**

Функция динамической передачи параметров внешним функциям feval.

Синтаксис:

feval('<имя функции>',x1,...,xn);

[y1,...,yk]=feval('<имя функции>',x1,...,xn);.

Первое обращение передает входные параметры вызываемой функции. Второе обращение не только передает входные параметры, но и возвращает выходные параметры. Функция используется при передаче внешних функций в виде параметров других функций.

Функция организации меню выбора.

Синтаксис:

k=menu('<заголовок>','выбор 1','выбор 2',...,'выбор n');.

Функция выводит графическое кнопочное меню выбора обрабатывающего реакцию курсора мыши. При установке курсора на соответствующую кнопку выбора и нажатии клавиши мыши выходная переменная **k** принимает значение номера нажатой клавиши. Далее в программе с помощью условного оператора организуется разветвление алгоритма. С помощью этой функции можно организовать многоуровневые меню, что существенно облегчает реализацию сервисной части программ.

Функции вычисления времени и дат.

Функция определения текущей даты и времени clock.

Синтаксис:

dt=clock;.

Функция возвращает вектор-строку **dt** из шести элементов, определяющих текущую дату и время в десятичном формате **[гол месяц день час минуты секунды]**. Первые пять элементов – целые числа, а шестой элемент – вещественное число, мантисса которого зависит от установленного формата вывода командой **format.** Обращение **fix(clock)** округляет секунды до целого значения.

Функция определения текущей даты date.

Синтаксис:

s=date;.

Функция возвращает символьную строку **s** текущей даты в формате **‘день-месяц-год’.**

Функция определения временного интервала etime.

Синтаксис:

e=etime(t2,t1);.

Функция возвращает длительность интервала времени в секундах, задаваемого векторами **t2** и **t1** из шести элементов**,** по формату возвращаемому функцией **clock.** Пример использования:

t1=clock;

«последовательность операторов»

etime(clock,t1);.

Фрагмент определяет интервал времени между моментами запуска функции **clock.** На границе месяца или года интервал времени может быть определен некорректно.

Функции запуска и остановки таймера tic, toc.

Пример использования:

tic;

«последовательность операторов»

toc;.

Последовательный запуск функций **tic, toc** возвращает показание таймера в секундах.

Функция подсчета процессорного времени cputime.

Пример использования:

t0=cputime;

«последовательность операторов»

t=cputime-t0;.

Фрагмент определяет процессорное время между моментами запуска функции **cputime.** При превышении значения процессорного времени внутреннего представления отсчет времени начинается заново.

Подсчет количества выполненных операций с плавающей точкой - функция flops.

f=flops;

flops(0);.

Первое обращение возвращает общее число выполненных операций с плавающей точкой. Второе обращение обнуляет счетчик числа операций с плавающей точкой.

8 Математические функции

В системе **MatLab** имеется широкий набор элементарных математических функций. Общая форма вызова стандартных математических функций имеет вид:

<результат>=<имя функции>(<список аргументов по значению или наименованию>);.

Как правило, аргументы могут быть любого типа, в том числе комплексными и массивами, если это не противоречит здравому смыслу. В случае массивов скалярные функции работают покомпонентно. Кратко перечислим некоторые классы функций.

Тригонометрические и обратные им функции.

Аргумент тригонометрических функций задается в единицах радиан. Результат обратных тригонометрических функций выдается в единицах радиан.

sin(z) - синус аргумента **z;**

asin(z) - значение арксинуса в диапазоне от **-pi/2** до **pi/2**;

cos(z) - косинус;

acos(z) - значение арккосинуса в диапазоне от **0** до **pi**;

tan(z) - тангенс;

atan(z) - значение арктангенса в диапазоне от **-pi/2** до **pi/2**;

atan2(x,y)- значение арктангенса в диапазоне от **-pi** до **pi**. В качестве аргументов задаются проекции точки на ортогональные оси прямоугольной системы координат, либо действительная и мнимая части комплексного аргумента.

Экспоненциальная, логарифмические и другие функции.

exp(z) - экспонента числа **z;**

log(z) - натуральный логарифм;

log2(z) - логарифм по основанию **2**;

pow2(n) - возведение **2** в степень **n**;

log10(z)- десятичный логарифм;

sqrt(z) - квадратный корень из числа **z;**

abs(z) - модуль числа **z.**

Дополнительные функции комплексного аргумента.

real(z) - выделение действительной части **z;**

imag(z) - выделение мнимой части **z;**

angle(z)- аргумент комплексного числа в диапазоне от **-pi** до **pi**;

conj(z) - функция комплексного сопряжения.

Функции преобразования типа.

fix(z) - округление до ближайшего целого в сторону нуля;

floor(z)- округление до ближайшего целого в строну **-inf**;

ceil(z) - округление до ближайшего целого в строну **+inf**;

round(z) - обычное округление до ближайшего целого;

rem(x,y) - выделение остатка от деления **x** на **y**;

sign(z) - вычисление сигнум - функции числа **z** (равная **0,** при z=0; **-1,** при z<0;**1,** при z>0).

Применение функций **fix, floor, ceil, round** к символьным и строковым переменным преобразует их в **ASCII** коды.

Кроме перечисленных функций, в системе **MatLab** определены прямые и обратные гиперболические функции, специальные функции и целый ряд других математических функций.

Функции проверки.

Проверку элементов массива на логическое условие позволяют осуществить функции **all, any**

y=all(x); **y=all(<условие>);**

y=any(x); **y=any(<условие>);.**

Функция **all** проверяет на соответствие условию все элементы массива, а функция **any** проверяет на соответствие условию хотя бы одного элемента массива. Таким образом, функция **all** как бы соответствует логической операции **И**, а функция **any** соответствует операции **ИЛИ**. Результат функций **all** и **any** является либо **1 – истина**, либо **0 – ложь.**

Если аргументом функций является просто имя одномерного массива - вектора, то это соответствует условию проверки элементов массива на равенство нулю. Если аргументом является логическое условие, включающее имя массива, то это условие распространяется на каждый элемент массива. Если аргументом функций является двумерный массив - матрица, либо он входит в условие проверки, то функции **all** и **any** проверяют каждый столбец и результатом является логический вектор - строка из нулей и единиц.

Поиск индексов и значений элементов массива, удовлетворяющих заданному условию, позволяет осуществить функция **find**

k=find(x); **k=find(<условие>);**

[i,j]=find(x); **[i,j]=find(<условие>);**

[i,j,s]=find(x); **[i,j,s]=find(<условие>);.**

Варианта обращений первой строки выдают вектор столбец индексов элементов одномерного массива, равных нулю либо удовлетворяющих заданному логическому условию. Варианты обращений второй строки, в случае двумерного входного массива выдает векторы столбцы индексов массива, равных нулю либо удовлетворяющих заданному условию. Варианты обращений третьей строки дополнительно возвращают вектор столбец значений элементов равных нулю, если условие не задано и логический вектор из единиц, если в качестве входного параметра задано условие.

Проверку существования переменной файла или функции можно осуществить функцией **exist**

k=exist('имя');.

Выходная переменная k может принимать следующие значения:

5 – для встроенной функции **MatLab;**
4 – для откомпилированной функции **Simulink**;
3 – для **MEX**-файла;
2 – для **M**-файла;
1 – для переменной;
0 – для объекта неопознанного класса.

Кроме перечисленных функций, в системе **MatLab** существует множество других функций проверки, например, проверки: типа операционной системы, наличия **IEEE** арифметики, принадлежности к глобальным переменным, выявления элементов массива типа **Inf** и **NaN,** выявления конечных элементов, принадлежности символа латинскому алфавиту, принадлежности символа строковой переменной, выявление пустого и разреженного массивов, проверки графического режима наложения и так далее.

9 Обработка строк

9.1 Основные функции

Строка представляет собой набор символов заключенный в одиночные апострофы и является разновидностью одномерного массива или вектора. В основе представления символов в строках лежит таблица **ASCII** кодов. **ASCII** таблица кодов ставит в однозначное соответствие каждому символу числовой код из диапазона от **0** до **255.**

Первые 127 чисел соответствуют **ASCII** кодам букв латинского алфавита, арабским цифрам и спецзнакам. Они соответствуют, так называемой, основной таблице. Вторая часть кодов образует дополнительную таблицу, которая может использоваться для представления символов других языков, например, кириллицы – русского.

Для определения длины строки символов **s** используется функция **length**
n=length(s);.

Длина вектора **s** соответствует числу символов строки, включая пробелы. Апостроф внутри строки вводится символом двойных кавычек.

Для преобразования массива положительных целых чисел **x** в строку **s** используются функции **char, setstr, string**

s=char(x);
s=char(x1,x2,...);
s=setstr(x);
s=string(x);.

Преобразование производится в соответствии с таблицей **ASCII** кодов. Возвращаемый параметр **s** представляет собой символьный массив той же структуры, что и массив **x**. Функция **char** может иметь несколько входных

параметров в виде массивов, при этом формируется вектор столбец строк из преобразованных массивов.

Для преобразования символов строки **s** в числовой вектор **x** используются функции **double, round, floor, ceil, fix**

x=double(s);
x=round(s);
x=floor(s);
x=ceil(s);
x=fix(s);.

Преобразование производится в соответствии с таблицей **ASCII** кодов. Функции округления **round, floor, ceil** и усечения дробной части числа **fix** в данном случае выступают как функции преобразования типа.

Функция **isstr** проверяет, является ли переменная **s** строкой

l=isstr(s);.

Функция возвращает **единицу**, если **s** строка и **ноль**, в противном случае.

Сформировать строку из заданного числа **n** пробелов позволяет функция **blanks**

s=blanks(n);.

Удалить из строки последние пробелы и вернуть новую строку позволяет функция **deblank**

s1=deblank(s);.

Проверить, является ли отдельно взятый символ и скалярная переменная символом латинского алфавита или пробелом, позволяют соответственно функции **isletter** и **isspace**

l=isletter(c);
l=isspace(c);.

В случае истины функции возвращают **единицу**, иначе **ноль**. Если входной параметр **c** является массивом, то проверка производится покомпонентно и результатом является логический массив из **0** и **1**.

9.2 Операции над строками

Горизонтальное или вертикальное объединение строк реализуют соответствующие функции **strcat** и **strvcat**

s=strcat(s1,s2,...,sn);
s=strvcat(s1,s2,...,sn);.

В результате горизонтального объединения получается длинная строка, включающая все строки. Если входные параметры одинаковой размерности массивы строк, то результатом будет горизонтальное объединение этих массивов. В случае вертикального объединения строк возвращается вектор строк. Если входные параметры одинаковой размерности массивы строк, то результатом будет вертикальное объединение этих массивов.

Горизонтальное или вертикальное объединение строк можно реализовать через операции формирования массива

s=[s1 s2 ... sn];

s=[s1;s2;... sn];.

Сравнение двух строк реализует функция **strcmp**

l=strcmp(s1,s2);

l=strcmp(s1,s2,n);.

Функция возвращает **единицу**, если элементы строк совпадают и их длины одинаковы и **ноль** в противном случае. Второе обращение позволяет указать число элементов **n**, подлежащих сравнению. Если входными параметрами являются массивы строк одинаковой размерности, то результатом будет логический массив соответствующей размерности, так как массивы сравниваются поэлементно.

Поиск подстроки **s2** в строке **s1** позволяет осуществить функция **findstr**

l=findstr(s1,s2);.

Функция возвращает начальный индекс или вектор индексов, начиная с которых подстрока **s2** входит в строку **s1**, иначе возвращается пустой массив.

Заменить все вхождения подстроки **s2** на подстроку **s3** в строке **s1** позволяет функция **strrep**

s=strrep(s1,s2,s3);.

Возвращаемая строка **s** отражает все замены, произведенные в строке **s1.** Если входные параметры массивы строк, то операция замены производится покомпонентно и результат **s** также массив строк.

Выровнять массив символов позволяет функция **strjust**

s2=strjust(s1);

s2=strjust(s1,'right');

s2=strjust(s1,'left');

s2=strjust(s1,'center');.

Функция возвращает выровненный справа, слева или по центру массив символов.

Найти все совпадения строки **s2** и подстроки **s1** позволяет функция **strmatch**

l=strmatch(s1,s2);

l=strmatch(s1,s2,'exast');.

Функция просматривает массив символов или многомерный массив символов **s2** по строкам, находит строки символов начинающиеся с **s1** и возвращает соответствующие индексы строк, как элементов массива. При второй форме обращения возвращаются индексы строк, как элементов массива, точно совпадающих с **s1**.

Возвратить часть строки **s1** ограниченную разделителем **delimiter** позволяет функция **strtok**

s=strtok(s1,delimiter);

s=strtok(s1);

[token,rem]=strtok(...);.

В качестве разделителя **delimiter** можно указать любой символ. Результатом является начальная часть исходной строки до указанного символа разделителя. Если символ разделителя не указан, то в качестве такового используется первый из встретившихся естественных разделителей: **символ табуляции** – **ASCII** код 9; **символ возврата каретки** – **ASCII** код 13; **пробел** – **ASCII** код 32. Третья форма обращения позволяет вернуть не только часть строки до разделителя **token**, но и остаток исходной строки **rem**.

Перевод символов строки в нижний регистр реализует функция **lover**

s=lower(s1);.

Возвращается строка, в которой все символы верхнего регистра, заменены символами нижнего регистра.

Перевод символов строки в верхний регистр реализует функция **upper**

s=upper(s1);.

Возвращается строка, в которой все символы нижнего регистра, заменены символами верхнего регистра.

9.3 Преобразование символов и строк

Преобразование целого числа или массива целых чисел в символьное представление реализует функция **int2str**

s=int2str(x);.

В данном случае значение числа как целого переводится в изображение его в виде набора цифр.

Преобразование вещественного числа или массива чисел в символьное представление реализует функция **num2str**

s=num2str(x);

s=num2str(x,precision);

s=num2str(x,format);.

Преобразование чисел осуществляется по умолчанию с округлением до четырех разрядов дробной части. Параметр **precision** позволяет указать число разрядов используемых для представления преобразованных в изображение чисел, при этом, если нужно используется экспоненциальное представление. Вместо параметра **precision** можно использовать параметр **format**, описанный в разделе посвященном работе с файлами.

Преобразование строки в арифметическое выражение и его вычисление реализует функция **str2num**

n=str2num(s);.

Допускается представление в строке знаков чисел, известных операций и функций, а также изображений комплексных чисел. Используя пробелы, между выражениями можно получить вектор строку численных результатов. Действия,

выполняемые функцией, напоминают работу калькулятора использующего стек памяти. Данная функция особо примечательна тем, что позволяет осуществить переход от символьного изображения математических выражений к вычислению результата.

Преобразование матрицы в строку реализует функция **mat2str**

s=mat2str(x);

s=mat2str(x,n);.

Матрица преобразуется в строку, с указанием разделителей строк, таким образом, чтобы применение функции подстановки **eval** давало исходную матрицу. Компоненты матрицы могут быть как числами, так выражениями с числами допустимыми во входном языке. По умолчанию точность преобразования составляет **15** значащих цифр. С помощью дополнительного параметра **n** можно конкретизировать точность преобразования и представления чисел.

Объединение строк в символьную матрицу реализует функция **str2mat**

s=str2mat(s1,s2,...);.

Построение символьной матрицы производится объединением строк в вектор столбец строк, короткие строки доращиваются пробелами.

10 Функции преобразования систем счисления

Преобразование положительного десятичного числа в двоичную строку реализует функция **dec2bin**

b=dec2bin(d);

b=dec2bin(d,n);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. С помощью дополнительного параметра **n** можно указать число бит для представления изображения двоичного числа. Если числа указанных бит недостаточно, то вывод осуществляется необходимым числом бит.

Преобразование строки двоичного представления числа в десятичное число реализует функция **bin2dec**

d=bin2dec(b);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. Если **b** является символьным массивом, то каждая строка символьного массива интерпретируется как представление двоичного числа и функция **bin2dec** возвращает вектор столбец десятичных чисел. Если **b** является матрицей строк, то элементы строки матрицы объединяются в одну строку и интерпретируются представлением двоичного числа, результат возвращается в виде вектора столбца десятичных чисел.

Преобразование положительного десятичного числа в строку изображения шестнадцатеричного числа реализует функция **dec2hex**

h=dec2hex(d);

h=dec2hex(d,n);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. С помощью дополнительного параметра **n** можно указать число разрядов для представления изображения шестнадцатеричного числа. Если числа указанных разрядов недостаточно, то вывод осуществляется необходимым числом разрядов.

Преобразование строки шестнадцатеричного представления числа в десятичное число реализует функция **hex2dec**

d=hex2dec(h);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. Если **h** является символьным массивом, то каждая строка символьного массива интерпретируется как представление шестнадцатеричного числа и функция **hex2dec** возвращает вектор столбец десятичных чисел. Если **h** является матрицей строк, то элементы строки матрицы объединяются в одну строку и интерпретируются представлением шестнадцатеричного числа, результат возвращается в виде вектора столбца десятичных чисел.

Преобразование положительного десятичного числа **d** в **B**-строку реализует функция **dec2base**

Bs=dec2base(d,b);

Bs=dec2base(d,b,n);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. Параметр **b** указывает базовую систему счисления целым числом равным от **2** до **36**. С помощью дополнительного параметра **n** можно указать число разрядов для представления изображения **B**-строки. Если числа указанных разрядов недостаточно, то вывод осуществляется необходимым числом разрядов.

Преобразование **B**-строки в десятичное число реализует функция **base2dec**

d=base2dec(s,b);.

Диапазон представления чисел от **0** до $\mathbf{2^{52}}$. . Параметр **b** указывает базовую систему счисления целым числом равным от **2** до **36**. Если **s** является символьным массивом, то каждая строка символьного массива интерпретируется как представление B-строки и функция **base2dec** возвращает вектор столбец десятичных чисел. Если **s** является матрицей строк, то элементы строки матрицы объединяются в одну строку и интерпретируются представлением **B**-строки, результат возвращается в виде вектора столбца десятичных чисел.

Преобразовать шестнадцатеричное число, заданное **IEEE** символьной строкой, в **IEEE** вещественное десятичное число в плавающем формате с удвоенной точностью позволяет функция **hex2num**

d=hex2num(s);.

Если **s** является символьным массивом, то каждая строка символьного массива интерпретируется как представление шестнадцатеричного числа и функция **hex2num** возвращает вектор столбец вещественных десятичных чисел.

Если **s** является матрицей строк, то элементы строки матрицы объединяются в одну строку и интерпретируются представлением шестнадцатеричного числа, результат возвращается в виде вектора столбца вещественных десятичных чисел.

11 Операции с векторами и матрицами

11.1 Операции формирования массивов

Система **MatLab**, как уже отмечалось, специально приспособлена для обработки массивов данных векторов, матриц и строковых переменных.

В качестве формирователя массивов используются квадратные скобки. Для явного указания пустого массива достаточно записать выражение вида **<имя>=[];.**

Вектор строку можно задать списком компонент (элементов) заключенных в квадратные скобки

<имя>=[список_компонент];
<имя>=[v1 v2 ... vn];
<имя>=[v1,v2,...,vn];.

Допускается задавать компоненты вектора в виде констант, переменных и выражений. Список компонент перечисляется через пробелы либо запятую.

Существует краткая форма задания регулярного массива в виде записи напоминающей описание арифметической прогрессии позволяющая опускать квадратные скобки

<имя>=n:h:k;
<имя>=n:k;
<имя>=[n:h:k];
<имя>=[n:k];,

где **n** - начальное значение; **k** - конечное значение; **h** - шаг. Если шаг не указан, то полагается **h=1**. Следует отметить, что таким образом формируются векторы строки.

Для формирования векторов столбцов и матриц необходимо задавать массивы построчно и в конце списка строки ставить знак **(;)**

```
v=[v1;v2;...vn];
m=[m11 m12 ... m1n; m21 m22 ... m2n; ... mn1 mn2 ... mnn];
m=[m11 m12 ... m1n;
  m21 m22 ... m2n;
  ... ... ... ...
  mn1 mn2 ... mnn];.
```

Доступ к элементам массивов осуществляется по индексам

```
a=v(k);
b=v(k,l);
```

v(k)=a;
m(k,l)=b;.

Для формирования (заполнения) массивов можно использовать циклы по индексным переменным

```
for k=1:2;
 for l=1:3;
  m(k,l)=k+l;
 end;
end;
m.
```

Вектор строки и столбцы можно формировать, не используя индексы, путем наращивания или конкатенации элементов массива

v=[]; for k=1:3; v=[v k]; end; v ;
v=[]; for k=1:3; v=[v;k]; end; v .

11.2 Функции формирования массивов

Для явного задания обнуленных массивов с указанием размерностей используется функция **zeros** с указанием числа строк и столбцов

v=zeros(1,n);
v=zeros(n,1);
m=zeros(m,n);
m=zeros(n,n);
m=zeros(n);.

Здесь первое обращение к функции задает обнуленную вектор строку из **n** компонент. Второе обращение задает соответственно вектор столбец. Третье обращение задает нулевую прямоугольную матрицу размером **m*n**. Два последних обращения задают квадратную нулевую матрицу размером **n*n**.

Матрицы, заполненные единицами, формируются функцией **ones(m,n)**.

Для задания нулевых матриц с единицами на главной диагонали используется функция **eye(m,n)**.

Функция **rand(m,n)** создает матрицу случайных чисел размером **m*n** с равномерным законом распределения в диапазоне от **0** до **1**.

Функция **randn(m,n)** создает матрицу случайных чисел с нормальным законом распределения, нулевым математическим ожиданием и среднеквадратическим отклонением равным единице.

11.3 Функции преобразования матриц

Функция **fliplr(a)** осуществляет поворот матрицы относительно вертикальной оси.

Функция **flipud(a)** осуществляет поворот матрицы относительно горизонтальной оси.

Поворот матрицы на **90** градусов против часовой стрелки осуществляется функцией **rot90(a)**. Можно указать число поворотов **n,** используя обращение **rot90(a,n)**, где **n** может принимать как положительные, так и отрицательные целые значения.

Функция **rechape(a,m2,n2)** преобразует матрицу **a** размером **m1*n1** в новую матрицу размером **m2*n2** путем выборки элементов и заполнения новой матрицы по столбцам при условии, что **m1*n1=m2*n2**.

Функция **tril(a)** образует из матрицы **a** нижнюю треугольную матрицу, путем обнуления элементов выше главной диагонали.

Функция **triu(a)** образует из матрицы **a** верхнюю треугольную матрицу, путем обнуления элементов ниже главной диагонали.

Функция **diag(x)** формирует или извлекает диагональ матрицы:

если **x** вектор, то **diag(x)** создает диагональную квадратную матрицу с вектором **x** на диагонали. Указав, положительный или отрицательный номер диагонали **diag(x,n),** получим прямоугольную матрицу, с над - или под - диагональю, отсчитанной от главной диагонали и состоящей из элементов вектора **x**;

наоборот, если **x** матрица, то функция **diag(x)** возвращает вектор столбец из главной диагонали матрицы. Указав при обращении **diag(x,n)** соответствующий номер над - или под – диагонали, сформируем из нее вектор столбец**.**

11.4 Функции определения размеров массивов

Функция **size(m)** выдает вектор из двух компонент означающих число строк и столбцов массивов

[n,p]=size(m);.

Функция **length(v)** от векторного аргумента выдает его длину, то есть число компонент, а при матричном аргументе выдает размер по максимальному измерению

n=length(v);

n=length(m);.

11.5 Операции преобразования векторов и матриц

Операция двоеточия **(:)**, используемая вместо индексов массива, означает неявный цикл и позволяет реализовать массу полезных операций преобразования векторов и матриц.

Запись оператора в виде **y=x(2:n)** позволяет сформировать вектор **y** из компонент вектора **x** со второй по **n**-ную. Вместо верхней границы измерения в старших версиях допускается использовать зарезервированное слово **end**.

Перенос столбца или строки матрицы, соответственно в вектор столбец или строку можно реализовать следующими фрагментами

v=m(:,2);

v=m(2,:);.

Аналогичным способом легко осуществить перезапись фрагмента массива в новый массив

m2=m1(2:k,3:l);

и наоборот вписать массив фрагментом большего массива

m1(2:k,3:l)=m2;.

Можно реализовать перенос фрагмента из одного массива в другой со сдвигом

m2(2:7,4:9)=m1(1:6,7:12);.

Развернуть матрицу по столбцам в вектор столбец можно с помощью простой записи

v=m(:);.

Используя операцию конкатенации, можно из массивов фрагментов имеющих одинаковое количество строк или столбцов производить горизонтальное или вертикальное объединение фрагментов в новый массив

a=[a1,a2,...,an];

b=[b1;b2;...;bm];.

Удаление из массива строки, столбца, диапазона строк или столбцов реализуются следующими фрагментами

m(2,:)=[]; m(:,3)=[];

m(2:5,:)=[]; m(:,3:7)=[].

11.6 Векторные и матричные операции

Векторные и матричные операции на **MatLab** можно разделить на две группы: стандартные с точки зрения математического аппарата и нестандартные, которым нет соответствия в традиционной математике.

К стандартным операциям относятся следующие векторно-матричные операции.

Умножение вектора или матрицы на скаляр **c**

v2=c*v1;

m2=c*m1;.

Алгебраическое сложение **(+/-)** векторов и матриц одинакового размера

v3=v1+v2; v4=v2-v1;

m4=m1+m2-m3;.

Умножение векторов и матриц при условии сопряжения числа столбцов предыдущего сомножителя с числом строк последующего, то есть (**m*n)*(n*k)=(m*k)**

x=a*y; p=r*g;

a=a*b*c;.

Операция транспонирования массивов отображается апострофом **(')** после имени массива

m2=m1';.

Следует отметить, что в случае комплексных массивов производится не только транспонирование, но и комплексное сопряжение, то есть, выполняется операция эрмитова сопряжения, поэтому для простого сопряжения комплексных матриц перед апострофом ставиться знак точки **(.')**

m2=m1.';.

Определитель невырожденной квадратной матрицы вычисляется обращением к функции **det(m)**

d=det(m);.

Операция обращения невырожденной квадратной матрицы осуществляется стандартной функцией **inv**

b=inv(a);.

В частности решение линейных алгебраических систем уравнений **a*x=y**, если **x,y** - столбцы и **x*a=y**, если **x,y** - строки, могут быть записаны в виде

x=inv(a)*y;, если **x,y** - векторы столбцы;

x=y*inv(a);, если **x,y** - векторы строки.

Решение линейной алгебраической системы уравнений можно реализовать, не прибегая к обращению матрицы, а, используя операции деления справа **(/)** и деления слева **(\)**

x=a\y;, если **x,y** - векторы столбцы;

x=y/a;, если **x,y** - векторы строки.

Аналогичным образом можно решать и матричные уравнения вида a*x=y, где **x,y** - матрицы

x=a\y;.

К квадратным невырожденным матрицам применима операция возведения в целую степень **(^)**

b=a^n;, где **n** - положительное или отрицательное целое число.

При этом результат равен **n**-кратному умножению исходной матрицы **a** самой на себя, если **n** положительное и **n**-кратному умножению обратной матрицы от **a**, если **n** - отрицательное. Отметим, что получение обратной матрицы в этом случае эквивалентно возведению ее в степень **-1**.

11.7 Нестандартные матричные операции

К нестандартным матричным операциям можно отнести следующие операции.

Применение элементарных математических функций к векторам и матрицам приводит к их покомпонентному преобразованию в соответствии с заданной функцией и результатом является вектор или матрица того же размера с преобразованными элементами.

b=sin(a);

c=exp(a);.

Применение элементарных математических функций к векторам и матрицам позволяет реализовать некоторые нестандартные экзотические операции по преобразованию массивов данных.

В системе **MatLab** допустима операция алгебраического сложения **(+/-)** векторов и матриц со скаляром **c**

v2=v1+c;

m2=m1-c;.

При этом скаляр алгебраически суммируется со всеми элементами вектора или матрицы.

Возможны также покомпонентные операции умножения **(.*)**, деления **(./)** или **(.\)** и возведения в степень **(.^)** векторов и матриц одинакового размера

c=a.*b;

c=a./b;

c=a.\b;

c=a.^b;.

Результатом покомпонентного умножения массивов является массив из произведения соответствующих элементов сомножителей. Правое покомпонентное деление дает массив отношения элементов слева направо. Левое покомпонентное деление дает массив отношения элементов справа налево. Покомпонентная операция возведения в степень соответствует возведению элемента левого массива в степень определяемую значением элемента правого массива.

11.8 Дополнительные векторные и матричные функции

Векторное произведение двух векторов в трехмерном пространстве реализует функция **cross**

c=cross(a,b);.

Результирующий вектор удовлетворяет следующему соотношению

$$c = a * b = (a_y \cdot b_z - a_z \cdot b_y) \cdot i + (a_z \cdot b_x - a_x \cdot b_z) \cdot j + (a_x \cdot b_y - a_y \cdot b_x) \cdot k .$$

Тензорное произведение векторов и матриц реализует функция **kron**

z=kron(x,y);.

Тензорное или кронекерово произведение двух массивов размерности **(k*l)** и **(m*n)** имеет размерность **(km*ln)** и соответствует выражению

$$z = x \otimes y = \left[x_{ij} * y \right] = \begin{bmatrix} x_{11} * y & x_{12} * y & \dots & x_{1k} * y \\ x_{21} * y & x_{22} * y & \dots & x_{2k} * y \\ \dots & \dots & \dots & \dots \\ x_{l1} * y & x_{l2} * y & \dots & x_{lk} * y \end{bmatrix}.$$

Нормы векторов и матриц можно вычислить, используя функцию **norm**

k=norm(v,p);

k=norm(m,p);, где параметр **p** определяет вид нормы (если параметр **p** не указан, то вычисляется **2**-норма).

Число обусловленности матрицы по отношению к операции обращения возвращает функция **cond(m)**

k=cond(m);.

r=rank(m) вычисляет ранг матрицы **m.**

След матрицы равный сумме диагональных элементов вычисляет функция **trace**

t**=trace(m);.**

q=null(m) вычисляет ортонормальный базис нуль-пространства матрицы **m**.

q=orth(m) выдает ортонормальный базис матрицы **m**.

l=rref(m) приводит матрицу **m** к треугольному виду методом Гаусса с частичным выбором главного значения.

r=chol(m) находит разложение Холецкого, для симметричных действительных и комплексных эрмитовых матриц из условия **r*r=m**.

Функция **lu(m)** выполняет **LU**- факторизацию матрицы **m** на произведение нижней треугольной матрицы **l** с единичной диагональю и верхней треугольной матрицы **u** возможно с перестановками отображаемыми матрицей **p**

[l,u]=lu(m);

[l,u,p]=lu(m);.

Функция **qr(m)** выполняет **QR**- факторизацию матрицы **m** на произведение унитарной матрицы **q** и верхней треугольной матрицы **r** возможно с перестановками отображаемыми матрицей **p**

[q,r]=qr(m);

[q,r,p]=qr(m);.

Сингулярное разложение матрицы реализуется стандартной функцией **svd(m)**

s=svd(m);

[u,s,v]=svd(m);

[u,s,v]=svd(m,0);.

Сингулярным разложением действительной прямоугольной матрицы **m** размера **m*n** при **m>=n** понимается представление вида **m=u*s*v',** где **u'*u=v*v'=1** и **s=diag(s1,...,sn).**

Псевдообращение матрицы по Муру-Пенроузу реализуется функцией **pinv**

p=pinv(m);

p=pinv(m,tol);.

Псевдообратная матрица **p** имеет размер матрицы **a'** и удовлетворяет условиям **a*p*a=a; p*a*p=p**. Вычисление псевдообратной матрицы, основано на использовании функции **svd(m)** и приравнивании нулю всех сингулярных чисел меньше величины **tol**.

Вычисление собственных значений и векторов осуществляет функция **eig**

l=eig(m);

[h,l]=eig(m);.

Первое обращение вычисляет собственные значения матрицы **m,** в виде вектора столбца **l.** Второе обращение кроме диагональной матрицы собственных значений **l** возвращает матрицу правых собственных векторов **h** в виде вектор столбцов, удовлетворяющих условию **m*h=l*h.** Норма каждого вектора равна единице. Левые собственные вектора можно найти, используя обращение

[h,l]=eig(m');.

Обобщенная проблема собственных значений, соответствующая нетривиальному решению системы уравнений **m*h=l*n*h,** представимых в виде **inv(n)*m*h=l*h,** где **n –** невырожденная матрица решается следующим обращением к функции

[h,l]=eig(m,n);.

При этом обобщенные собственные вектора и значения удовлетворяют условию **a*h=l*n*h.**

Преобразование матрицы к форме Шура в виде верхней треугольной матрицы **t** с собственными значениями на диагонали с помощью унитарного преобразования **u,** удовлетворяющего условию **m=u*t*u'** осуществляется функцией **schur**

t=schur(m);

[u,t]=schur(m);.

Комплексные собственные значения выдаются в виде диагональных блоков 2*2.

Функция **qz** приводит к обобщенной форме Шура пару матриц **m,n**

[mm,nn,q,z,v]=qz(m,n);

При этом **mm** и **nn** являются комплексными верхними треугольными матрицами; **q, z** -–матрицы приведения к верхней треугольной форме; **v** –

матрица обобщенных собственных векторов. Кроме того, выполняются условия **q*m*z=mm; q*n*z=nn; m*v*diag(nn)=n*v*diag(mm).**

В качестве примера использования **qz** алгоритма, рассмотрим описание технической системы, с одним входом и одним выходом, уравнениями состояния

$$Q * \dot{x} + R * x = b * u,$$
$$y = c^t * x + d * u$$

объединяющими систему обыкновенных дифференциальных уравнений относительно переменной состояния **x** и систему алгебраических уравнений выхода. При этом для определения полюсов и нулей передаточной функции системы последовательно решаются две системы уравнений:

для вычисления полюсов

$$R * h = -\lambda * Q * h;$$

для вычисления нулей

$$\begin{bmatrix} -R & b \\ c^t & d \end{bmatrix} * h = \lambda * \begin{bmatrix} Q & 0 \\ 0 & 0 \end{bmatrix} * h,$$

где λ - собственное значение; h - собственный вектор. Как видим, задача требует решения обобщенной проблемы собственных значений. Используя **qz** алгоритм, к указанным парам матриц находим полюса и нули передаточной функции системы, как отношения диагональных элементов соответствующей пары матриц приведенных к верхнему треугольному виду.

Вычисление собственных значений матричного полинома степени **p** вида $\mathbf{(m_0+l*m_1+...+l^p*m_p)*h=0}$ производит функция **polyeig**

[h,l]=polyeig(m0,m1,...,mp);,

здесь **m0,m1,...,mp –** матрицы порядка **n; h –** матрица собственных векторов размера **n*(n*p); l –** матрица собственных значений размером **n*p.**

Функция **polyvalm** вычисляет значение матричного полинома вида $\mathbf{y(x)=p_n*x^n+p_{n-1}*x^{n-1}+...+p_2*x^2+p_1*x+p_0}$ по заданной матрице **x** и вектору коэффициентов $\mathbf{p=[p_n,p_{n-1},...,p_0]}$

y=polyvalm(p,x);.

11.9 Функции от матриц

Матричные функции вычисляются в соответствии с традиционным математическим аппаратом. В системе **MatLab** определены следующие аналитические функции от невырожденных квадратных матриц.

Экспоненту от матричного аргумента можно вычислить, используя стандартные функции

b=expm(a);

c=expm1(a);
d=expm2(a);
e=expm3(a);.

Функция **expm** является встроенной функцией **MatLab**, **expm1** является **m** файлом и вычисляет матричную экспоненту путем разложения Паде матрицы **a**, **expm2** вычисляет матричную экспоненту, используя разложение Тейлора, а **expm3** вычисляет матричную экспоненту путем спектрального разложения.

Натуральный логарифм от матричного аргумента вычисляется функцией **logm**

b=logm(a);.

Функция **sqrtm** вычисляет квадратный корень от матричного аргумента **b=sqrtm(a);,** причем **b*b=a.**

Для вычисления произвольной функции от матричного аргумента **a** можно воспользоваться стандартной функцией **funm**

b=funm(a,'<имя функции>');
[b,esterr]=funm(a,'<имя функции>');.

Первое обращение выдает результат в виде матрицы **b** и предупреждение при появлении заметной погрешности в виде относительной невязки. Второе обращение выдает как результат, так и значение относительной невязки.

12 Функции обработки данных

Система **MatLab** содержит набор универсальных функций обработки данных представленных в векторной и матричной форме.

Функции **sum** и **cumsum** возвращают сумму и вектор строку промежуточных сумм элементов вектора либо матрицы. Матрицы обрабатываются по столбцам

s=sum(v); s=sum(m);
sp=cumsum(v); sp=cumsum(m);.

Функции **prod** и **cumprod** возвращают произведение и вектор строку промежуточных произведений элементов вектора либо матрицы. Матрицы обрабатываются по столбцам

s=prod(v); s=cumprod(m);
sp=cumprod(v); sp=cumprod(m);.

Функции **max** и **min** позволяют найти соответственно, максимальные и минимальные элементы векторов и матриц, а также номера первых позиций расположения этих элементов

z=max(v); z=min(v);
[z,p]=max(v); [z,p]=min(v);
z=max(m); z=min(m);
[z,p]=max(m); [z,p]=min(m);.

Матрицы обрабатываются по столбцам и выдаются соответственно векторы строки максимальных либо минимальных элементов и их номеров позиций первого вхождения в случае нескольких одинаковых элементов. В случае комплексных массивов сравнение происходит по модулю.

Сортировка элементов массива по возрастанию осуществляется функцией **sort**

y=sort(v); y=sort(m);

[y,i]=sort(v); [y,i]=sort(m);.

В случае матрицы производится упорядочение элементов каждого столбца. Вторая форма обращения дополнительно возвращает массив индексов **i** позволяющих восстановить структуру исходного массива с помощью следующих конструкций

```
v=(i(:))=y(:);
[l,c]=size(m);
for k=1:c;
 m=(i(:,k),k)=y(:,k);
end;.
```

Определение среднего значения вектора или вектора строки средних значений столбцов матрицы определяет функция **median**

mdv=median(v);

mdm=median(m);.

Определение среднеквадратического отклонения от среднего значения вектора или столбцов матрицы определяет функция **std**

sv=std(v);

sm=std(m);.

13 Операции над полиномами

Вычисление значения полинома $\mathbf{p(x)=p_1*x^n+p_2*x^{n-1}+...+p_n*x+p_{n+1}}$ при заданных коэффициентах и значении аргумента **x** производится функцией **polyval**

y=polyval(p,x);

y=polyval(p,xm);.

Первое обращение производит вычисление значения полинома при известном векторе коэффициентов $\mathbf{p=[p_1,p_2,...,p_n,p_{n+1}]}$ в точке **x.** Во втором обращении используется одномерный или двумерный массив аргументов **xm** и вычисление производится для каждого элемента массива и результат **y** также выдается в виде массива той же размерности, что и **xm**.

Вычисление матричного полинома $\mathbf{p(x)=p_1*x^n+p_2*x^{n-1}+...+p_n*x+p_{n+1}}$, где **x** – матрица, при известном векторе коэффициентов $\mathbf{p=[p_1,p_2,...,p_n,p_{n+1}]}$ производится функцией **polyvalm**

y=polyvalm(p,xt);.

Вычисление произведения полиномов **p1** степени **m** и **p2** степени **n,** в соответствии с выражением

$$p3(k) = \sum_{j=\max(1,k+1-n)}^{\min(k,m)} p1(j) \cdot p2(k+1-j),$$

реализует функция **conv**

p3=conv(p1,p2);,

здесь **p1=[p1$_1$,p1$_2$,...,p1$_m$,p1$_{m+1}$]; p2=[p2$_1$,p2$_2$,...,p2$_n$,p2$_{n+1}$]; p3 - в**ектор коэффициентов результирующего полинома имеющего размерность **m*n**.

Деление полиномов **p1** на **p2**, заданных векторами коэффициентов производится функцией **deconv**

[p3,r]=deconv(p1,p2);.

Результат возвращается векторами коэффициентов полиномов целой части от деления **p3** и остатка **r**.

Вычисление производных степенного полинома выполняет функция **polyder**

dp=polyder(p);

dp3=polyder(p1,p2);

[dp1,dp2]=polyder(p1,p2);.

Полиномы задаются векторами коэффициентов, начиная с коэффициента при старшей степени аргумента. Первое обращение вычисляет коэффициенты полинома соответствующие производной **dp(x)/dx.** Второе обращение вычисляет коэффициенты производной от произведения полиномов **d(p1(x)*p2(x))/dx.** Третье обращение вычисляет производную от отношения полиномов **d(p1(x)/p2(x))/dx** в виде коэффициентов полиномов числителя и знаменателя **dp1,dp2.**

Вычисление корней полинома заданного вектором строкой коэффициентов **p=[p$_1$,p$_2$,...,p$_n$,p$_{n+1}$]** реализует функция **roots**

r=roots(p);.

Корни возвращаются в виде вектора столбца **r**.

Вычисление вектора строки коэффициентов полинома **p** по заданному вектору столбцу корней полинома **r** реализует функция **poly**

p=poly(r);.

Если вместо вектора столбца корней полинома задана матрица **m**, то функция **poly** возвращает вектор строку коэффициентов характеристического полинома матрицы **p(s)=det(s*I-m)**, где **I**- единичная матрица

p=poly(m);.

Разложение дробно–рациональных функций на простые дроби и вычисление вычетов, полюсов и коэффициентов полинома целой части от деления числителя на знаменатель производят функции **residue** и **resi2**

[r,p,k]=residue(p1,p2);

rj=resi2(p1,p2,pole,m,j);

[p1,p2]=residue(r,p,q);.

Первое обращение - по заданным коэффициентам числителя **p1** и знаменателя **p2** функция **residue,** в случае простых корней:

$$\frac{p1(s)}{p2(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + ... + \frac{r_n}{s-p_n} + q(s);$$

вычисляет вектор столбец **r** вычетов, вектор столбец **p** полюсов и вектор строку **q** коэффициентов целой части дробно-рациональной функции. Количество полюсов определяется формулой

n=length(p1)-1=length(r)=length(p).

Вектор коэффициентов целой части будет пустым, если **length(p1)<length(p2),**

в противном случае

length(k)=length(p1)-length(p2)+1.

В случае кратных корней, **j**-тый полюс кратности **m** дает слагаемые

$$\frac{r_j}{s-p_j} + \frac{r_{j+1}}{(s-p_j)^2} + ... + \frac{r_{j+m-1}}{(s-p_j)^m};$$

Применение функции **resi2** с указанием полюса **pole**, кратности **m** и порядкового номера полюса **j** выдает для него вектор вычетов **rj**.

Третье обращение реализует функцию свертки разложения в дробно-рациональную функцию отношения полиномов **p1(s)/p2(s).**

Заметим, что разложение дробно-рациональной функции на простые дроби плохо обусловлено и может давать заметную погрешность в случае кратности корней знаменателя. В этом случае предпочтительно использовать описание в пространстве состояния или представление дробно-рациональной функции в виде полюсов и нулей.

14 Функции дифференцирования и интегрирования

Функция вычисления конечных разностей и численного дифференцирования **diff** выдает вектор разностей соседних элементов вектора или элементов столбцов матрицы и выдает вектор на единицу меньшего размера или матрицу с числом строк на единицу меньше исходной

dv=diff(v); y=diff(x);

dm=diff(m);.

Обращение вида

y=diff(x,n); вычисляет конечные разности порядка **n,** удовлетворяющие рекуррентному соотношению **diff(x,n)=diff(x,n-1).** Аппроксимацией производной **n**-го порядка является отношение **diff(y,n)./diff(x,n).**

Вычисление конечных разностей и приближенного значения градиента позволяет осуществить функция **gradient**

[px,py]=gradient(f);

[px,py]=gradient(f,dx,dy);.

Первое обращение вычисляет конечные разности функции **f(x,y)** заданной на двумерной сетке и представляющий двумерный массив чисел – матрицу. Второе обращение возвращает численное значение производных в виде массивов **px=df/dx** и **py=df/dy,** где **dx,dy** могут быть скалярами равными постоянным шагам разбиения сетки по осям **x** и **y**, либо векторами координат узлов сетки при переменных шагах.

Пятиточечная аппроксимация Лапласиана осуществляется функцией **del2**

v=del2(u);.

Функция возвращает массив того же размера, каждый элемент которого равен разности среднего значения соседних элементов и элемента рассматриваемого узла. Внутри области узел имеет четыре соседних узла, а на границе и в углах только два или три узла. Если матрицу **u** рассматривать как функцию **u(x,y)**, вычисленную в точках квадратной сетки, то **4*del2(u)** представляет конечно-разностную аппроксимацию дифференциального оператора Лапласа от функции **u**

$$\Delta^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Интегрирование таблично заданной функции методом трапеций осуществляет стандартная функция **trapz**

i=trapz(x,y);

i=trapz(y);.

Первое обращение вычисляет интеграл от таблично заданной функции **y(x)** методом трапеций. Аргументы **x** и **y** обычно одномерные массивы – вектора. Если **y** матрица, имеющая столбцы длиной **x,** то интеграл вычисляется для каждого столбца матрицы как отдельно заданной функции. Второе обращение подразумевает шаг интегрирования равный единице. Для шага не равного единице достаточно полученное значение интеграла умножить на величину шага **h.**

Вычисление интегралов методом квадратур производят функции **quad** и **quad8**

[i,cnt]=quad(<'имя функции'>,a,b,tol,trase);

[i,cnt]=quad8((<'имя функции'>,a,b,tol,trase);.

Функция **quad** использует квадратурные формулы Ньютона–Котеса 2-го порядка, а функция **quad8** – формулы 8-го прядка. Параметры - **входные:** первый - имя встроенной или внешней функции; **a, b** – пределы интегрирования; **tol** – относительная погрешность (по умолчанию **1e-3**); **trase** – не равный нулю приводит к выводу графика показывающего ход вычисления интеграла; **выходные: i** – значение интеграла; **cnt** – возвращает количество

вычислений подынтегральной функции. Последние два входных параметра можно опускать без пропуска в списке параметров**!**

15 Интегрирование обыкновенных дифференциальных уравнений (ОДУ)

Численное решение задачи Коши для систем ОДУ реализуются функциями **ode23** и **ode45**

[t,x]=ode23(<'имя функции'>,t0,tf,x0,tol,trase);

[t,x]=ode45(<'имя функции'>,t0,tf,x0,tol,trase);.

Система, в общем случае нелинейных дифференциальных уравнений, предполагается представленной в нормальной форме Коши системой ОДУ первого порядка $\frac{dx}{dt} = f(x,t)$, где **x** – вектор состояний; **t** – независимая переменная, например время; **f** – в общем случае нелинейная вектор – функция. Указанные функции реализуют соответственно формулы Рунге – Кутты интегрирования систем ОДУ 2,3- го и 4,5- го порядка.

Параметры:

входные: первый - имя встроенной или внешней функции; **t0** – начальное значение аргумента (независимой переменной); **tf** – конечное значение аргумента; **x0** – вектор начальных условий; **tol** – задаваемая точность (по умолчанию **1e-6**); **trace** – флаг не равный нулю приводит к графическому выводу хода интегрирования;

выходные: t – вектор аргумента искомой функции; **x** – двумерный массив, где каждый столбец соответствует одной из системы определяемых функций.

16 Аппроксимация и интерполяция данных

Аппроксимация данных степенным полиномом по методу наименьших квадратов реализует функция **polyfit**

p=polyfit(x,y,n);.

Входными параметрами функции являются массив аргументов **x**, массив значений **y** и степень аппроксимирующего полинома **n**. Результат выдается в виде вектора коэффициентов аппроксимирующего полинома **p** размера **n+1.** Если длина массивов равна **n+1**, то получаем интерполирующий полином.

При решении задачи аппроксимации решается переопределенная система линейных алгебраических уравнений вида **xx*p=y**, где **xx** – матрица Вандермонда с элементами $xx_{ij} = x_i^{n-j}$. В соответствии с методом наименьших квадратов решение переопределенной системы ищется в виде **p=(xx'*xx)^-1*xx'*y.**

Аппроксимацию периодической функции на основе быстрого преобразования Фурье реализует функция **interpft**

yp=interpft(y,n);.

Здесь **y** – входной массив отсчетов периодической функции; **n** – число точек аппроксимации; **yp** – массив отсчетов аппроксимирующей функции. Если **m** число отсчетов входного массива, а **dx** интервал дискретизации входной функции, то интервал дискретизации для yp определяется по формуле **dyp=dx*m/n**, причем **n** всегда превышает **m.**

Кубическую интерполяцию таблично заданной функции одной переменной реализует функция **icubic**

yi=icubic(y,xi);

yi=icubic(x,y,xi);.

Здесь **y** - массив значений функции; **x** – массив значений аргумента; **xi** – массив аргументов с более мелким шагом заданный внутри интервала интерполяции по оси **x**. Функция **icubic** использует для интерполяции кубические полиномы. Более мелкую сетку допустимо использовать при условии, что аргумент **x** изменяется монотонно и сетка равномерна. Если **y** – двумерный массив, то интерполяция производится для каждого столбца и выходной массив **yi** также двумерный.

Интерполяцию таблично заданной функции одной переменной кубическим сплайном реализует функция **spline**

yi=spline(x,y,xi);

pp=spline(x,y);

v=ppval(pp,xx);

[breaks,coefs,l,k]=unmkpp(pp);

pp=mkpp(breaks,coefs);.

Функции **ppval, unmkpp, mkpp –** образуют небольшой пакет системы **MatLab** для работы с кусочно-гладкими полиномами.

Первое обращение интерполирует кубическими сплайнами функцию, заданную массивом аргументов **x** и значений **y**, используя более мелкую сетку массива **xi** внутри интервала интерполяции. Второе обращение позволяет вернуть **pp**-форму интерполирующего сплайна. Третье обращение вычисляет значения кусочно-гладкого полинома **pp** для аргументов заданных массивом **xx.** Четвертое обращение возвращает характеристики кусочно-гладкого полинома **pp**:

breaks – вектор разбиения аргумента;

coefs – коэффициенты кубических сплайнов;

l=length(breaks)-1;

k=length(coefs)/l.

Универсальную одномерную табличную интерполяцию реализует функция **interp1**

yi=interp1(x,y,xi);

yi=interp1(x,y,xi,<'метод'>);.

Функция **interp** выдает массив значений интерполирующей функции **yi** по заданным массивам аргументов **x** и значений **y**, используя дополнительный массив аргументов **xi** с более мелкой сеткой внутри интервала интерполяции **x.** Если **y** двумерный массив, то интерполяция производится для каждого столбца и выходной массив **yi** также двумерный. Четвертый параметр позволяет задать метод интерполяции:

'linear' – линейная;

'cubic' – кубическая;

'spline' – кубические сплайны.

По умолчанию реализована **линейная** интерполяция. Принято, что аргумент **x** меняется монотонно, а для кубической интерполяции предполагается, что сетка по **x** равномерна.

Универсальная двумерная табличная интерполяция реализуется функцией **interp2**

zi=interp2(x,y,xi,yi);

zi=interp2(x,y,xi,yi, ,<'метод'>);.

Функция **interp2** интерполирует данные, определяющие некоторую поверхность на двумерной сетке **(x,y)** двумерным массивом **zi**. Возможно использование более мелкой сетки **(xi,yi)** в пределах интервалов интерполяции по осям **x,y**. Дополнительный параметр во втором обращении позволяет конкретизировать метод интерполяции: **'linear' –** линейная; **'cubic' –** кубическая. По умолчанию реализована **линейная** интерполяция. Принято, что аргументы **x** и **y** меняются монотонно, а для кубической интерполяции предполагается, что сетка **(x,y)** равномерна.

Двумерная интерполяция на неравномерной сетке реализуется функцией **griddata**

zi=griddata(x,y,z,xi,yi);

[xi,yi,zi]=griddata(x,y,z,xi,yi);.

Функция **griddata** возвращает двумерный массив, определенный на более мелкой сетке **(xi,yi),** как результат интерполяции исходной функции **z(x,y)** на неравномерной сетке. При второй форме обращения возвращаются также упорядоченные массивы **xi** и **yi.**

17 Вычисление минимумов и нулей функций

Минимизация функции одной переменной реализуется стандартными функциями **fmin, foptions**

xmin=fmin(<'имя_функции'>,x1,x2);

xmin=fmin(<'имя_функции'>,x1,x2,options);

[xmin,options]=fmin(<'имя_функции'>,x1,x2,optoins,p1,...,p10);

options=foptions;.

Первое обращение позволяет вернуть значение локального минимума **xmin** функции заданной первым параметром в интервале **x1<=x<=x2.**

Второе обращение использует вектор управляющих параметров **options,** включающий **18 компонентов** и предназначенный для настройки алгоритмов оптимизации используемых, как в системе **MatLab**, так и в дополнительном пакете программ **Optimization Toolbox.** Функция **fmin** использует только три параметра **options(1), options(2), options(14).** Все параметры имеют значения по умолчанию. С целью изменения отдельных управляющих параметров, разрешается передавать только часть из них, но первые параметры должны быть перечислены без пропуска.

Третье обращение позволяет передать дополнительно до **10 параметров** оптимизируемой функции **y=fun(x,p1,...,p10);,** а также возвращает значения управляющего вектора **options.**

Четвертое обращение возвращает вектор строку исходных значений управляющих параметров используемых функциями минимизации.

Назначение и исходные значения по умолчанию компонент управляющего вектора **options:**

options(1) – вывод промежуточных результатов (**0** – не выводятся; **1** – выводятся);

options(2) – итерационная погрешность (разность соседних итераций) для аргумента, по умолчанию **1e-4;**

options(3) - итерационная погрешность (разность соседних итераций) для функции, по умолчанию **1e-4;**

options(4) – значение критерия соблюдения ограничений, по умолчанию **1e-6;**

options(5) – алгоритм стратегии – **0**;

options(6) – алгоритм оптимизации – **0;**

options(7) – алгоритм линейного поиска – **0;**

options(8) – значение целевой функции – **0;**

options(9) – для контроля градиента установить **1**, по умолчанию **0**;

options(10) – количество выполненных итераций, начальное значение **0;**

options(11) – количество вычисленных градиентов, начальное значение **0;**

options(12) - количество вычисленных ограничений, начальное значение **0;**

options(13) – количество ограничений в виде неравенств, начальное значение **0;**

options(14) – максимальное количество итераций, начальное значение **0,** по умолчанию для **fmin - 500**, для **fmins - 200*n**, где **n** – число переменных;

options(15) – параметр, используемый при наличии целевой функции;

options(16) – минимальное приращение переменных при вычислении градиента, по умолчанию **1e-8**;

options(17) - максимальное приращение переменных при вычислении градиента, по умолчанию **0.1;**

options(18) – размер шага минимизации, по умолчанию **h<=1**.

Минимизацию функции нескольких переменных реализует стандартная функция **fmins**

xmin=fmins(<'имя_функции'>,x0);

xmin=fmins(<'имя_функции'>,x0,options);

xmin=fmins(<'имя_функции'>,x0,options,[],p1,...,p10);.

Первое обращение возвращает вектор **xmin**, соответствующий локальному минимуму функции в окрестности **x0**.

Второе обращение использует вектор управляющих параметров **options.** Функция **fmins** использует только первые **4** параметра - **options(1), options(2), options(3), options(14).**

Третье обращение позволяет передать дополнительно до **10 параметров** оптимизируемой функции **y=fun(x0,p1,...,p10);,** а также возвращает значения управляющего вектора **options.** Четвертый входной параметр используется для совместимости с функцией **fminu** из пакета **Optimization Toolbox.**

Нахождение нулей функции одной переменной реализуется стандартной функцией **fzero**

z=fzero((<'имя_функции'>,x0);

z=fzero((<'имя_функции'>,x0,tol);

z=fzero((<'имя_функции'>,x0,tol,trace);.

Первое обращение возвращает значение нуля функции в окрестности точки **x0.** Второе обращение позволяет задать погрешность возвращаемого результата, по умолчанию **tol=eps**. Наличие четвертого входного параметра позволяет выдать на экран промежуточные результаты поиска нуля. Функция **fzero** использует методы деления отрезка пополам, секущей и обратной квадратичной интерполяции.

18 Формирование узлов одномерной, двумерной и трехмерной сеток

При построении двумерных и трехмерных изображений используются функции построения линейных и логарифмических сеток в прямоугольной системе координат, которые могут быть использованы, например, при моделировании задач теории поля.

Формирование линейного массива равноотстоящих узлов реализует функция **linspace**

x=linspace(x1,x2);

x=linspace(x1,x2,n);.

Первое обращение формирует линейный массив размера **(1*100)** в границах **(x1,x2).** Второе обращение формирует линейный массив размера **(1*n)** в границах **(x1,x2).**

Формирование узлов логарифмической сетки реализует функция **logspace**

x=logspase(d1,d2);

x=logspase(d1,d2,n);.

Первое обращение формирует вектор строку, содержащую **50** равноотстоящих в логарифмическом масштабе точек в диапазоне от $\mathbf{10^{d1}}$ до $\mathbf{10^{d2}}$.

Второе обращение формирует вектор строку, содержащую **n** равноотстоящих в логарифмическом масштабе точек в диапазоне от $\mathbf{10^{d1}}$ до $\mathbf{10^{d2}}$.

Упорядоченные массивы двумерных и трехмерных сеток реализует функция **meshgrid**

[x,y]=meshgrid(x,y);

[x,y]=meshgrid(x);

[x,y,z]=meshgrid(x,y,z);.

Первое обращение формирует упорядоченные массивы, определяющие координаты узлов прямоугольника задаваемого векторами **x** и **y.** Этот прямоугольник можно интерпретировать как область определения трехмерной поверхности либо распределения поля. Второе обращение есть сокращенная форма записи **[x,y]=meshgrid(x,x);.** Третье обращение формирует упорядоченные массивы **x,y,z** определяющие координаты узлов параллелепипеда, задаваемого исходными векторами **x,y,z.** Этот параллелепипед может интерпретироваться как область определения трехмерных параметрических поверхностей.

19 Преобразование Фурье, свертка и фильтрация

Прямое одномерное дискретное и обратное преобразование Фурье реализуют функции **fft** и **ifft**

y=fft(x); y=fft(x,n);

x=ifft(y); x=ifft(y,n);.

Прямое и обратное дискретные преобразования Фурье для одномерных массивов длиной N описываются выражениями

$$y(k) = \sum_{j=1}^{N} x(j) * e^{-j2\pi(l-1)(k-1)/N}$$

$$x(l) = \frac{1}{N} \sum_{k=1}^{N} y(k) * e^{j2\pi(l-1)(k-1)/N}.$$

Первое обращение реализует дискретное преобразование Фурье массива **x,** используя **FFT-алгоритм быстрого Фурье преобразования**. Если массив **x** двумерный, то преобразование выполняется для каждого столбца отдельно и

результатом также является двумерный массив **y**. Второе обращение реализует n-точечное дискретное преобразование Фурье, **быстрое**, если $\mathbf{n=2^k}$, где **к** – целое, и **обычное дискретное,** в противном случае. Если **n>length(x)**, то массив доращивается нулями, если **n>length(x)**, то лишние элементы отбрасываются.

Следующие два обращения реализуют обратное дискретное преобразование Фурье, как быстрое, так и обычное в зависимости от длины последовательности заданной массивом.

Временная и частотная оси прямого и обратного преобразований Фурье связаны следующими соотношениями: если **T**– период или время наблюдения, а **dt**– шаг по времени, то дискрет по частоте равен **df=1/T**, а частотный диапазон равен Δ**F=1/dt.**

Перегруппировку выходных массивов прямого преобразования Фурье осуществляет функция **fftshift**

y=fftshift(x);.

Эта функция перегруппировывает выходные массивы прямого преобразования Фурье, размещая нулевую частоту в центре спектра. Обратное преобразование переформированного массива даст неверный результат.

Двумерное дискретное прямое и обратное преобразования Фурье осуществляются функциями **fft2** и **ifft2.**

Операция свертки одномерных массивов осуществляется с помощью ранее рассмотренных функций **conv** и **deconv**

z=conv(x,y);

[q,r]=deconv(z,x);.

Для массивов **x** длиной **m** и **y** длиной **n** свертка **z** является массивом длиной **m+n-1**, элементы которой определяются выражением

$$z(k) = \sum_{j=\max(1,k+1-n)}^{\min(k,m)} x(j)\cdot y(k+1-j).$$

Свертку можно интерпретировать как свертку дискретного представления входного сигнала с дискретным представлением импульсной характеристики системы с целью получения дискретного представления выходного сигнала.

В соответствии со спектральной теорией свертке сигналов представленных массивами во временной области соответствует произведение их спектров в частотной области.

Функция **deconv** реализует операцию обратную свертке и если интерпретировать массив **x** как дискретно заданный входной сигнал, а **y** как дискретную форму импульсной характеристики системы, то второе обращение к результату полученному из первого обращения, равносильно определению импульсной характеристики системы при **q=y**, **r=0**.

Операции свертки двумерных массивов реализуется функцией **conv2.**

Дискретная одномерная фильтрация реализуется функцией **filter**

y=filter(b,a,x);

[y,zf]=filter(b,a,x,zi);.

Функция **filter** реализует преобразование эквивалентное преобразованию входного сигнала **x** дискретным фильтром, представленным передаточной характеристикой

$$(E^k+a_2*E^{k+1}+...+a_n*E^{n+k})*y=(b_1*E^k+...+b_m*E^{m+k})*x,$$

где **E** – оператор сдвига; **n** – порядок числителя; **m** – порядок знаменателя; **x** – входное воздействие. Сигнал на выходе дискретного фильтра **y** в этом случае находится из рекуррентного соотношения

$$y_k=b_1*x_k+...+b_m*x_{k+m}-a_2*y_{k+1}-...-a_n*y_{k+n},$$

где $b=[b_1,b_2,...,b_m]$; $a=[1,a_2,...,a_n]$; $x=[x_1,x_2,...,x_m]$.

Второе обращение позволяет учесть запаздывания входного **zi** и выходного **zf** сигналов.

Дискретная двумерная фильтрация реализуется функцией **filter2.**

20 Функции построения графиков

20.1 Двумерные графики

Построение графиков функции одной переменной реализуется функцией **fplot**

fplot(<'имя_функции'>,[<интервал>],n,<приращение_угла>, <дробление_шага>);

[x,y]=fplot(<'имя_функции'>,[<интервал>],n,<приращение_угла>, <дробление_шага>);.

Первое обращение от второго отличается тем, что последнее позволяет возвратить массивы аргумента и значений функции для последующего использования. Первые два входных параметра обязательны, а следующие можно опускать полностью или частично, но без пропуска.

Первый параметр задает имя встроенной или внешней функции. Второй параметр определяет интервал по аргументу **[xmin,xmax].** Третий параметр задает число частей разбиения интервала, по умолчанию **n=25**. Четвертый параметр задает условие разбиения интервала такое, чтобы изменение наклона графика на соседних участках не отличалось более чем на заданную величину. По умолчанию **<приращение_угла>=10^0.** Пятый параметр задает условие разбиения интервала такое, чтобы изменение наклона графика на соседних участках не превышало заданного приращения угла, но не требовало больше заданного дробления шага. По умолчанию **<дробление_шага>=20.**

Двумерный график в линейном масштабе реализуется функцией **plot**

plot(y);

plot(x,y,s);

plot(x1,y1,s1,...xn,yn,sn);.

Первое обращение соответствует построению графика значений функции заданной массивом **y,** в зависимости от номера элемента. Если массив **y** комплексный, то строится график **plot(real(y), imag(y)).** Если **y** двумерный действительный массив, то графики строятся для каждого из столбцов, а, если элементы комплексные, то мнимые части игнорируются.

Второе обращение соответствует случаю, когда заданы массив аргументов **x** и массив значений функции **y**. Массивы **x** и или **y** могут быть двумерными. Если **y** двумерный, а **x** одномерный, то каждому столбцу матрицы **y** соответствует отдельный график от **x**. Если **y** одномерный, а **x** двумерный, то каждому столбцу матрицы **x** соответствует отдельный график от **y.** Если оба массива **x** и **y** двумерные, то столбцы матрицы **y** отображаются относительно столбцов матрицы **x**. Все графики изображаются на одном поле. Третий входной параметр является необязательным и позволяет указать способ отображения линий, способ отображения точек, цвет линий и точек в виде строковой переменной **s** содержащей до трех символов, причем указывается либо тип линии, либо тип точки. Соответственно графики изображаются линиями либо дискретно символами. Типы линий: **-** непрерывная; **--** штриховая; **:** двойной пунктир; **-.** штрих-пунктирная. Типы точек: **.** точка; **+** плюс; ***** звездочка; **o** кружок; **x** крестик. Набор цветов **y** – желтый, **m** – фиолетовый, **c** – голубой, **r** – красный, **g** – зеленый, **b** – синий, **w** – белый, **k** - черный. Если цвет линий не указан, то он выбирается по умолчанию из набора циклически. Если третий параметр отсутствует, то тип и цвет линий выбирается автоматически.

Третья форма обращения позволяет вывести несколько графиков на одном поле с указанием способов отображения. Для повышения разнообразия изобразительных средств, во входной список можно дважды включить один и тот же график, отобразив его один раз линией, а другой раз дискретно символом, возможно разных цветов.

Двумерный график в логарифмическом масштабе реализуется функцией **loglog**

loglog(y);
loglog(x,y,s);
loglog(x1,y1,s1,...xn,yn,sn);.

Функция **loglog** в отличие от функции **plot** использует логарифмический масштаб по обеим осям. Узлы логарифмической сетки по оси **x** формируются функцией **logspace**.

Графики в полулогарифмических масштабах по осям **x** и **y** реализуются функциями **semilogx, semilogy**

semilogx(y);
semilogx(x,y,s);
semilogx(x1,y1,s1,...xn,yn,sn);
semilogy(y);
semilogy(x,y,s);

semilogy(x1,y1,s1,...xn,yn,sn);.

Функция **semilogx** использует логарифмический масштаб по оси **x**, а функция **semilogy** - по оси **y**.

График в полярной системе координат реализуется функцией **polar**

polar(phi,rho);

polar(phi,rho,s);.

Первый входной параметр представляет массив углов выраженных в радианах, а второй – массив радиусов. Для комплексных функций это соответственно аргументы и модули значений. Третий входной параметр является необязательным и позволяет указать способ отображения, по аналогии с функцией **plot**. Отметим, что данная функция позволяет отобразить лишь один график.

20.2 Трехмерные графики

Рассмотрим наиболее простые функции построения трехмерных графиков, линий уровня и сечений. В трехмерной графике значение элементов двумерного числового массива **z** интерпретируются как **z**-координаты над плоскостью **(x,y)**. Возможно несколько способов визуализации трехмерного изображения: соединение точек в сечении (функция **plot3**); построение сетчатых поверхностей (функции **mesh** и **surf**). Поверхность, построенная функцией **mesh** сетчатая, ячейки которой имеют цвет фона, а цвет их границ определяются свойством **EdgeColor** графического объекта **surface**. Поверхность, построенная с помощью функции **surf** сетчатая, ячейки и границы которой имеют цвета, определяемые свойством **FaceColor** графического объекта **surface**. Для полного понимания реализации трехмерной графики требуется знание объектно-ориентированного программирования.

Построение линий и точек в трехмерном пространстве функция **plot3**

plot3(x,y,z);

plot3(x,y,z,s);

plot3(x1,y1,z1,s1,...,xn,yn,zn,sn);.

Функция **plot3** является трехмерным аналогом функции **plot**. По первому обращению функция строит точки по координатам указанным одномерными массивами **x,y,z** и соединяет их прямыми линиями. Вместо одномерных координатных массивов можно использовать двумерные **X,Y,Z**, в этом случае функция строит точки с координатами по столбцам двумерных массивов и соединяет их прямыми. Двумерные массивы могут быть, например, сформированы и специальным образом упорядочены из одномерных массивов координат функцией **meshgrid**. Второе обращение позволяет указать для графика способ отображения в виде типа линии или типа точки и цвета, задаваемые строковой переменной **s**, в полном соответствии с функцией **plot.**

Третья форма обращения позволяет объединить на одном поле несколько графиков с указанием для каждого из них способа отображения.

Трехмерная сетчатая поверхность реализуется функциями **mesh, meshc**, **meshz**

mesh(X,Y,Z,C);	**meshc(X,Y,Z,C);**	**meshz(X,Y,Z,C);**
mesh(x,y,Z,C);	**meshc(x,y,Z,C);**	**meshz(x,y,Z,C);**
mesh(Z,C);	**meshc(Z,C);**	**meshz(Z,C);**
mesh(X,Y,Z);	**meshc(X,Y,Z);**	**meshz(X,Y,Z);**
mesh(x,y,Z);	**meshc(x,y,Z);**	**meshz(x,y,Z);**
mesh(Z);	**meshc(Z);**	**meshz(Z);.**

Функция **mesh** просто строит трехмерную поверхность, функция **meshc,** в дополнение к трехмерным поверхностям строит проекции линий постоянного уровня, функция **meshz** в дополнение к трехмерным поверхностям строит плоскость отсчета на нулевом уровне, закрывая при этом поверхность лежащую ниже. В качестве входных параметров малыми буквами **x,y,z** указаны одномерные массивы координат, а большими буквами **X,Y,Z** указаны двумерные массивы координат. Двумерные массивы могут быть, например, сформированы и специальным образом упорядочены из одномерных массивов координат функцией **meshgrid**. Цвета узлов поверхности задаются массивом **C,** с особенностями задания цветов и переопределения свойств **EdgeColor** объекта **surface** можно ознакомится по имеющимся публикациям. Если массив **C** не указывается, то полагается **C=Z**. Цвет в этом случае пропорционален высоте поверхности. Если входные массивы **X,Y,Z** двумерные, то строится сетчатая поверхность для значений массива **Z**, определяемых на множестве значений массивов **X** и **Y**. Если входные массивы **x, y** одномерные, а **Z**-двумерный, то узлы сетчатой поверхности определяются тройками чисел **(x(j),y(i),Z(i,j)).** Если входным параметром указан только двумерный массив **Z**, то в качестве сетки используются порядковые номера строк и столбцов массива **Z**.

Затененная трехмерная сетчатая поверхность реализуется функциями **surf** и **surfc**

surf(X,Y,Z,C);	**surfc(X,Y,Z,C);**
surf(x,y,Z,C);	**surfc(x,y,Z,C);**
surf (Z,C);	**surfc(Z,C);**
surf (X,Y,Z);	**surfc(X,Y,Z);**
surf (x,y,Z);	**surfc(x,y,Z);**
surf (Z);	**surfc(Z);.**

Функция **surf** просто строит затененную трехмерную поверхность, функция **surfc,** в дополнение к трехмерным поверхностям строит проекции линий постоянного уровня. В качестве входных параметров малыми буквами **x,y,z** указаны одномерные массивы координат, а большими буквами **X,Y,Z** указаны двумерные массивы координат. Двумерные массивы могут быть, например, сформированы и специальным образом упорядочены из одномерных

массивов координат функцией **meshgrid**. Цвета ячеек поверхности задаются массивом **C,** с особенностями задания цветов и переопределения свойств **FaceColor** объекта **surface** можно ознакомится по имеющимся публикациям. Если массив **C** не указывается, то полагается **C=Z**. Цвет в этом случае пропорционален высоте поверхности. Если входные массивы **X, Y, Z** двумерные, то строится сетчатая поверхность для значений массива **Z**, определяемых на множестве значений массивов **X** и **Y**. Если входные массивы **x, y** одномерные, а **Z**-двумерный, то узлы сетчатой поверхности определяются тройками чисел **(x(j), y(i), Z(i,j))**. Если входным параметром указан только двумерный массив **Z**, то в качестве сетки используются порядковые номера строк и столбцов массива **Z**.

Формирование массива описания линий уровня для трехмерных поверхностей реализуется функцией **contourc**

C=contourc(Z); **C=contourc(x,y,Z);**
C=contourc(Z,n); **C=contourc(x,y,Z,n);**
C=contourc(Z,v); **C=contourc(x,y,Z,v);.**

Здесь заглавные буквы **C, Z** – соответствуют двумерным массивам, а **x, y, v** означают одномерные массивы. Первые два варианта обращения формируют массив линий уровня **C** для функции **contour**, без учета и с учетом диапазонов изменения координат **x, y**. Обращения во второй строке указывают необходимое число линий уровня. В третьей строке обращения позволяют выделить линии уровня заданные в массиве **v** значениями.

Изображение линий уровня для трехмерной поверхности реализуется функцией **contour**

contour(Z); **contour(x,y,Z);**
contour(Z,n); **contour(x,y,Z,n);**
contour(Z,v); **contour(x,y,Z,v);**
contour(...,'тип_линии');
C=contour(...);
[C,h]=contour(...);.

Обращения первой строки отображают двумерные линии уровня для массива **Z**, определяющего поверхность в трехмерном пространстве, без учета и с учетом диапазонов изменения координат **x, y**. Обращения второй строки позволяют отобразить **n** линий уровня (по умолчанию **n=10**). Обращения третьей строки отображают линии уровня для значений указанных в векторе **v.** Четвертая строка иллюстрирует обращение, которое позволяет указать тип и цвет линий уровня аналогично функции **plot**. Обращение пятой строки возвращает массив описания линий уровня **C**, аналогично функции **contourc** для последующего использования функцией **clabel**. Обращение шестой строки позволяет дополнительно вернуть вектор дескрипторов **h** графических объектов **line** для каждой линии уровня.

При использовании функции **contour** предполагается, что элементы массивов **x** и **y** монотонно возрастают.

Изображение трехмерных линий уровня реализуется функцией **contour3**

contour3(Z); **contour3(x,y,Z);**
contour3(Z,n); **contour3(x,y,Z,n);**
C=contour3(...);
[C,h]=contour3(...);.

Обращения первой строки отображают трехмерные линии уровня для массива **Z**, без учета и с учетом диапазонов изменения координат **x, y**. Обращения второй строки отображают n трехмерных линий уровня (по умолчанию **n=10**). Обращение третьей строки возвращает массив описания линий уровня **C**, аналогично функции **contourc** для последующего использования функцией **clabel**. Обращение четвертой строки позволяет дополнительно вернуть вектор дескрипторов **h** графических объектов **line** для каждой линии уровня.

При использовании функции **contour3** предполагается, что элементы массивов **x** и **y** монотонно возрастают.

Сечения функций от трех переменных реализуются функцией **slice**

slice(x,y,z,V,xi,yi,zi,n);
slice(X,Y,Z,,V,xi,yi,zi,n);
slice(V,xi,yi,zi,n);
h=slice(...);.

Первое обращение позволяет построить плоские сечения функции **V(x,y,z)** вдоль осей **x,y,z,** позиции сечений определяются векторами **xi, yi, zi.** Размер двумерного массива равен **(m*n*p,m*n*p)**, где **m=length(y); n=length(x); p=length(z).** Во втором обращении вместо одномерных массивов **x,y,z** используются двумерные массивы **X,Y,Z** сформированные функцией **meshgrid.** Третья форма обращения строит сечения в области **(x,y,z)** задаваемой номерами компонент одномерных массивов **xi,yi,zi** по каждому измерению **x=1:n; y=1:m; z=1:p**. Четвертое обращение возвращает вектор столбец дескрипторов **h** графических объектов **surface,** являющихся сечениями трехмерной функции **V(x,y,z).**

20.3 Вспомогательные графические функции

Вызов очередного графического окна для построения графика реализует функция **figure**

figure(n);.

Позволяет задать новое графическое окно и графический объект **figure**, указав его порядковый номер. В противном случае графики последовательно отображаются в одном графическом окне.

Возвратить дескриптор текущего графического объекта **figure** позволяет функция **gcf**

h=gcf;.

Очистить текущее графическое окно позволяет вызов функции **clf;**.

Закрыть графическое окно с указанием дескриптора позволяет обращение к функции **close(h);.**

Масштабирование осей и вывод на экран реализует функция **axis**

axis(xmin,xmax,ymin,ymax);
axis(xmin,xmax,ymin,ymax,zmin,zmax);
axis('auto');
axis(axis);
v=axis;
axis(v);
axis('ij');
axis('xy');
axis('square');
axis('equal');
axis('off');
axis('on');
[s1,s2,s3]=axis('state');
axis(s1,s2,s3);.

Функция **axis** используется в случае, если пользователя не устраивает режим **авто масштабирования** действующий по умолчанию.

Функция **axis** обеспечивает преемственность предшествующих версий системы **MatLab,** ориентированных на символьный режим **(DOS)**, с версиями ориентированными на графический интерфейс **(Windows).**

Первые две формы обращения позволяют установить масштаб по осям **x,y,z** по осям активного графического окна. Следующее обращение позволяет вернуться к режиму **авто масштабирования** действующему по умолчанию. Четвертая форма фиксирует текущие значения масштабов для последующих графиков, аналог режима наложения графиков **hold**. Пятая форма позволяет вернуть вектор строку **v** действующих масштабов по осям. Шестая форма позволяет установить масштабы по осям заданные вектор строкой **v**. Седьмая форма перемещает начало отсчета осей в левый верхний угол, так называемая матричная система координат. Восьмая форма возвращает начало отсчета осей в левый нижний угол (стандартная форма). Девятая форма устанавливает одинаковый диапазон изменения переменных по осям, нивелирует разную разрешающую способность экрана по вертикали и горизонтали. Десятая форма устанавливает масштаб, обеспечивающий одинаковые расстояния между метками по осям. Одиннадцатая форма устанавливает масштаб, обеспечивающий квадратные размеры пикселей. Двенадцатая форма восстанавливает полноразмерный масштаб, отменяя масштабы установленные

обращениями **axis('square')** и **axis('equal').** Тринадцатая и четырнадцатая формы соответственно снимают и восстанавливают обозначения и маркеры осей. Пятнадцатая и шестнадцатая формы возвращают и устанавливают строку вектора состояния объекта **axes,** компоненты которого могут иметь следующие значения: **s1='auto'|'manual'; s2='on'|'off'; s3='xy'|'ij'.** По умолчанию вектор состояния имеет значения **['auto' 'on' 'xy']**.

Управление режимом сохранения текущего графического окна осуществляется функцией **hold on/off**. Ключ **on** обеспечивает режим сохранения текущего графика и свойств объекта **axes** – режим наложения последующих графиков в окно с сохранением прежних масштабов по осям. Ключ **off** отключает режим сохранения. Вызов **hold** без указания ключа обеспечивает переключение режимов. Более детально, ключ **on** присваивает свойству **NextPlot** для активных объектов **figure** и **axes** значение **add – наложение**, а ключ **off** присваивает свойству **NextPlot** для активных объектов **figure** и **axes** значение **replase – замена**.

Управление масштабом графика позволяет реализовать функция **zoom**

zoom on;
zoom off;
zoom;
zoom out;.

Ключ **on** включает режим масштабирования активного графика. В этом режиме при установке курсора мыши в окне и нажатии левой клавиши масштаб окна увеличивается в два раза, а при нажатии правой клавиши масштаб в два раза уменьшается. Удерживая левую клавишу и перемещая курсор по диагонали, можно выделить, интересующую область графика. Ключ **off** выключает режим масштабирования. Вызов **zoom,** без указания ключа, приводит к переключению режима. Ключ **out** возвращает график в исходное состояние.

Возвратить дескриптор текущего графического объекта **axes** позволяет функция **gca**

h=gca;.

Разбиение графического окна на части реализует функция **subplot**

subplot(m,n,p);
subplot(mnp);
subplot(h);.

Функция вызывается перед построениями графиков для указания их размещения в различных частях графического окна.

Первые две формы обращения полностью перекрываются и предназначены для разбивки графического окна на подокна с созданием нескольких объектов **axes**. Первый параметр **m** задает число разбиений по вертикали, второй **n** – число разбиений по горизонтали, третий **p** – указывает номер текущего подокна. Окна нумеруются слева направо и сверху вниз.

Вместо **p** можно использовать массив следующих подряд окон типа **p=[1,2,3**]. Это часто необходимо при оформлении графических результатов, когда с помощью функции **subplot** поле разбивается на большее число подокон, чем необходимо. Затем часть из них объединяется для вывода графиков, а часть окон используется для вывода поясняющего текста функцией **text.** Предварительно с помощью функции **axis('off'),** с этих окон удаляются маркеры осей. Окна имеют относительные размеры **(0-1,0-1)**, поэтому используются относительные координаты для указания местоположения текста.

Третья форма обращения позволяет осуществить выбор графического подокна для размещения графика, используя дескриптор **h** объекта **ases** текущего объекта **figure**.

Обращения типа:

clf;
subplot(1,1,1);
subplot(111);,

каждое из них позволяет удалить все подокна и возвратить графическое окно в исходное состояние по умолчанию.

Функция **grid on/off** позволяет включать и выключать режим нанесения координатной сетки на график. Вызов **grid** без указания ключа переключает режим нанесения координатной сетки.

Заголовки для двух и трехмерных графиков позволяет нанести функция **title**

title(<'текст'>);.

Функция размещает указанный текст над графиком.

Подписи координатных осей двух и трехмерных графиков позволяют осуществить функции

xlabel(<'текст'>);
ylabel(<'текст'>);
zlabel(<'текст'>);.

Функции размещают текст вдоль осей либо в случае трехмерного графика под графиком. Повторный вызов функций с другим текстом приведет к замене старого текста.

Маркировку линий уровня, представленных в виде массива описания **c** функциями **contour** и **contour3** позволяет осуществить функция **clabel**

clabel(c);
clabel(c,v);
clabel(c,'manual');.

Первое обращение размещает метки линий уровня в случайно выбранных позициях. Второе обращение маркирует линии уровня заданные вектором **v.** Третье обращение позволяет осуществить маркировку линий уровня указанных нажатием левой клавиши мыши. Нажатие клавиши **Enter** либо **правой клавиши мыши** завершает маркировку. Если мышь недоступна, то для

перехода между линиями уровня используется клавиша **Space-пробел,** и клавиши **«стрелки»** для выбора позиции.

Размещение текстовой информации на текущем графике реализует функция **text**

text(x,y,<'текст'>);

text(x,y,z,<'текст'>);.

Функция **text** размещает текст на двух и трехмерном графике с позиций **(x,y)** или **(x,y,z).** Если координаты **x,y,z** заданы массивами векторами, то указанный текст размещается по позициям соответствующих компонент векторов.

Позиционирование текста на текущем графике с помощью мыши реализует функция **gtext**

gtext('текст');.

Текст размещается, начиная с позиции курсора мыши при нажатии ее клавиши. В версиях системы **MatLab,** не отображающих русский текст, следует в программу поместить строки

set(ht,'units',units);

set(ht,'FontName',<'имя_шрифта'>);.

Размещение пояснений к текущему графику реализует функции **legend**

legend(<'текст1'>,<'текст2'>,...);

legend(<'тип_линии1'>,<'текст1'>,(<'тип_линии2'>,<'текст2'>,...);

legend(h,...);

legend(m);

legend(h,m);

legend off;

legend(...,n);.

Функция **legend** реализована в старших версиях системы **MatLab**. Первое обращение размещает пояснения, в поле графика либо рядом, в виде указанных текстовых строк. Второе обращение позволяет пояснить используемые на графике типы линий. Третье обращение вставляет пояснения к графику с дескриптором **h**. Четвертое обращение свидетельствует, что пояснения могут быть объединены в массив строк одинаковой длины. Использование ключа **off** удаляет пояснения с текущего графика. Параметр **n** в последнем обращении позволяет указать число позиций для размещения пояснений. Если пояснения не помещаются на графике, то график перестраивается, и пояснения располагаются рядом в пределах графического окна. Если **n=1**, то пояснения размещаются вне области графика. Если **n=-1**, то пояснения размещаются в области графика при наличии места.

Во избежание пересечения пояснений с линиями графика или координатной сетки, необходимо текущему графическому объекту **axes** присвоить дескриптор пояснения **legend**

hl=legend(...);

axes(hl);

print-dbitmap;.

Пояснения, как объект в целом, можно позиционировать с помощью мыши. Для перемещения необходимо, находясь в пределах объекта, нажать левую клавишу, и не отпуская ее, переместить объект в нужную позицию.

20.4 Элементы дескрипторной графики

Графические возможности системы **MatLab** реализованы на основе технологии объектного программирования в виде системы функций дескрипторной графики **Handle Graphics,** позволяющей эффективно работать с графическими объектами (графическим окном, координатными осями, линией, поверхностью и другими). К дескрипторной графике относятся одиннадцать исходных объектов:

root- корневой объект, соответствующий экрану и создаваемый системой **MatLab** в начале сеанса;

figure- графическая панель на экране;

axes- объект на поле графика, ассоциируемый с графическими осями;

uicontrol- пользовательский интерфейс для реализации панелей управления (кнопки, списки, слайдеры, окна редактирования и так далее);

uimenu- пользовательское меню;

image- поле (область) на графической панели, ассоциируемое с изображением, задаваемым двумерным массивом;

line- объект на поле графика, ассоциируемый с линией;

patch- объект на поле графика, соответствующий закрашенному многоугольнику;

surface- объект на поле графика, соответствующий поверхности;

text- объект на поле графика, соответствующий строке символов;

light- объект на поле графика, соответствующий подсветке элементов графика.

Все перечисленные имена являются **конструкторами** соответствующих объектов. Каждый объект при его создании получает некоторое значение указателя на него – **дескриптор.** С каждым объектом связан набор его свойств – **property.** Для получения дескриптора на объект достаточно обратиться к соответствующему конструктору в виде

hObj=Name_Obj('Свойство_1',Значение_1, …,'Свойство_n',Значение_n);,

где **hObj**- указатель на текущий объект; **Name_Obj**- имя конструктора.

Сохраняя дескриптор в значении некоторой переменной **hObj**, можно воспользоваться им для изменения свойств объекта. Для получения указателя на текущие активные объекты **figure** и **axes** используются функции **gcf (get current figure), gca (get current axes)**

hFig=gcf;

hAx=gca;.

Для задания свойств объекта используется функция **set,** например,

set(hFig,'Color','red'); или set(gcf,Çolor','red');.

Свойства текущего объекта, в том числе, установленные по умолчанию, можно получить, используя функцию **get**

get(Name_Obj); или **get(Name_Obj('Style','Name_Style'));.**

Отметим, что списки свойств конкретных объектов достаточно большие, поэтому здесь не приводятся. Второе обращение к функции **get** использует уточняющее имя **"стиля"** для сложного объекта типа **uicontrol,** имеющего несколько стилей: **Style: [{pushbutton} | togglebutton | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu].** Здесь приведены различные значения свойства **Style** для объекта **uicontol.** Стиль **pushbutton** объекта типа **uicontrol** установлен по умолчанию.

Возможные значения свойств объектов можно получить, используя функцию **set**

set(Name_Obj); или **set(Name_Obj('Style','Name_Style'));.**

Описание свойств объектов можно найти в соответствующих руководствах по системе **MatLab,** смотри список литературы.

Для удаления текущих графических объектов типа **figure** и **axes** используются функции **clf (clear current figure)** и **cla (clear current axes).**

20.5 Специальная графика

Раздел специальной графики включает функции построения столбцовых диаграмм, гистограмм, функций отображения векторов и комплексных элементов, дискретную последовательность данных, а также движущихся траекторий в плоскости и пространстве. Раздел специальной графики расширяется и совершенствуется от версии к версии.

Функция построения столбцовых диаграмм **bar**

bar(y); **bar(y,<'тип_линии'>);**

bar(x,y); **bar(x,y,<'тип_линии'>);**

[xb,yb]=bar(...);.

Обращения, указанные в первой строке, позволяют вывести график функции в виде вертикальных прямоугольников шириной равной шагу по аргументу и высотой равной значению функции в начале каждого интервала. По аналогии с функцией **plot,** можно указать используемый тип линии. В данной форме обращения, в качестве массива аргументов используются порядковые номера компонент массива **y**. Во второй строке обращений, ось аргументов маркируется в соответствии с массивом **x**. Если **X** и **Y** двумерные массивы одинаковой размерности, то диаграммы строятся по столбцам массивов. Обращение в третьей строке не строит диаграммы, а выводит массивы **xb, yb**, по которым функция **plot(xb,yb)** строит соответствующую диаграмму.

Отображение графика интервала погрешностей реализует функция **errorbar**

errorbar(x,y,l,u,<'тип_линии'>); **errorbar(x,y,e);**
errorbar(y,l,u,<'тип_линии'>); **errorbar(y,l,e);.**

Первое обращение строит график функции **y** от **x** с указанием интервала погрешности, который определяется массивами **u** и **l** – соответственно погрешность с **плюсом** и **минусом**. Задание типа линии осуществляется аналогично функции **plot.** Если массивы **X,Y,U,L** двумерные, то графики строятся по соответствующим столбцам. Второе обращение первой строки строит график симметричных погрешностей $\pm e$ в каждой точке. Обращения второй строки отличаются от таковых в первой строке тем, что ось аргументов задается порядковым номером элемента **y**.

Построение гистограммы реализуется функцией **hist**

hist(y);
hist(y,n);
hist(y,x);
[y,x]=hist(y,...);.

Функция **hist** подсчитывает и отображает в виде столбцовой диаграммы количество элементов массива **y**, попавших в заданный интервал разбиения диапазона значений. Если число интервалов разбиения, как в первом обращении, не задано, то по умолчанию **n=10**. Во втором типе обращения указывается число интервалов разбиения **n** всего диапазона. В третьем обращении элементы вектора аргументов **x**, автоматически задают интервалы разбиения диапазона. Четвертое обращение позволяет вернуть массивы **y** и **x** для построения гистограммы функцией **bar**.

Графики в форме дискретно-заданных последовательностей строится функцией **stem**

stem(y); **stem(y,<'тип_линии'>);**
stem(x,y); **stem(x,y,<'тип_линии'>);.**

График функции выводится в виде вертикальных линий пропорциональных значению функции и заканчивается кружочком. Данная форма вывода графиков удобна для отображения дискретно заданных функций и спектральной картины радиотехнических сигналов. Если массив аргументов **x** не задан, то его роль исполняют порядковые номера компонент вектора **y**. По аналогии с функцией **plot** можно задать тип линий отображения.

Графики в форме ступенчато-заданных функций реализуются стандартной функцией **stairs**

stairs(y);
stairs(x,y);
[xb,yb]=stairs(...);.

Данная форма графиков удобна для отображения процессов в дискретно-непрерывных системах. Массивы **x** и **y** должны быть упорядочены в порядке

возрастания аргумента. Если массив аргументов **x** не задан, то график отображается относительно порядковых номеров компонент массива **y.** Последняя форма обращения не строит графика, а выдает массивы для отображения графика функцией **plot(xb,yb).**

Функция **rose** построения гистограммы в полярной системе координат типа роза ветров

rose(phi);
rose(phi,n);
rose(phi,x);
[phi,r]=rose(phi,...);.

Функция подсчитывает и отображает на графике число элементов массива **phi,** попавших в заданный интервал разбиения диапазона значений. По умолчанию диапазон разбивается на **n=20** интервалов. Число интервалов разбиения может быть указано при обращении. Третья форма обращения строит отображение с учетом массива аргументов **x**. Четвертая форма обращения возвращает массивы для построения гистограммы в полярных координатах функцией **polar(phi,r).**

Графический вывод комплексных элементов массивов в виде векторов-стрелок реализуют функции **compass** и **feather**

compass(z); **feather(z);**
compass(x,y); **feather(x,y);**
compass(z,<'тип_линии'>); **feather(z,<'тип_линии'>);**
compass(x,y,<'тип_линии'>); **feather(x,y,<'тип_линии'>);**
[hc,hb]=compass(...);.

Функция **compass** выводит комплексные компоненты в виде векторов-стрелок исходящих из одной точки (начала системы координат). Функция **feather** выводит комплексные компоненты в виде векторов-стрелок исходящих из равноотстоящих точек оси аргументов.

Первая форма обращений соответствует заданию комплексного массива **z.** Вторая форма обращений равносильно заданию вектора **z=x+i*y.** При обращениях, по аналогии с функцией **plot,** можно указать тип используемых линий. Последняя форма обращения позволяет лишь вернуть массивы для последующего отображения графика функцией **plot(hc,hb).**

Поле градиентов функции в виде стрелок соответствующего размера и направления позволяет отобразить функция **quiver**

quiver(X,Y,DX,DY); **quiver(x,y,DX,DY);**
quiver(DX,DY);
quiver(x,y,DX,DY,s); **quiver(DX,DY,s);**
quiver(...,<'тип_линии'>);.

Здесь **X,Y,DX,DY** двумерные массивы сформированные, например, функцией **griddata.** Массивы **X** и **Y** парными элементами**,** задают координаты стрелок, а массивы **DX** и **DY** задают размер и направление стрелок.

Одномерные массивы **x** и **y** задают координаты сетки двумерной области, в узлах которой располагаются стрелки. Поле градиентов в этом случае задается четверками чисел **(x(j),y(i),DX(i,j),DY(i,j)).** Размеры массивов **x** и **y** связаны с размерами массивов **DX** и **DY** следующим образом: **n=length(x); m=length(y); [m,n]=size(DX)=size(DY);.** Если массивы, определяющие координаты стрелок не заданы, то координаты определяются номерами строк и столбцов массивов **DX** и **DY**. Скаляр **s** используется как масштабный коэффициент размера стрелок. По аналогии с функцией **plot** можно задать тип и цвет отображающих линий.

Отображение плоской кривой в виде, траектории движения светящейся точки типа кометы реализует функция **comet**

comet(y);
comet(x,y);
comet(x,y,p);
comet;.

Одномерные массивы **x** и **y** задают координаты движения светящейся точки. Если массив аргументов **x** не задан, то в качестве аргументов используются номера компонент массива **y**. Параметр **p** управляет длиной светящегося хвоста кометы **p*length(y),** по умолчанию **p=0.1**. Имя **comet** без параметров запускает демонстрационный пример.

Отображение пространственной кривой в виде, траектории движения светящейся точки типа кометы реализует функция **comet3**

comet3(z);
comet3(x,y,z);
comet3(x,y,z,p);
comet3;.

Одномерные массивы **x,y,z** задают координаты движения светящейся точки. Если массивы **x** и **y** не заданы, то в качестве аргументов используются номера компонент одномерного массива **z**. Параметр **p** управляет длиной светящегося хвоста кометы **p*length(z),** по умолчанию **p=0.1**. Имя **comet3** без параметров запускает демонстрационный пример.

Закрашенный многоугольник в плоскости реализует функция **fill**

fill(x,y,<'цвет'>);
fill(x,y,c);
fill(X,Y,C);
fill(X1,Y1,C1,...,Xn,Yn,Cn);
h=fill(...);.

Одномерные массивы **x** и **y** задают координаты вершин многоугольника. Поскольку многоугольник предполагается замкнутым, то начальная и конечная вершины соединяются линией. Цвет задается либо одним из символов **'r', 'g', 'b', 'c', 'm', 'y', 'w', 'k'** либо **RGB** вектором строкой **[r g b]**, каждая компонента которого задается **0** или **1**. Если используется вектор цветов **c** той же длины, что

и **x,y,** то многоугольник закрашивается цветом задаваемым вектором **c**. При этом вектор **c** масштабируется функцией **caxis,** и его элементы используются как индексы текущей палитры для задания цветов в вершинах многоугольника, а цвет внутри определяется билинейной интерполяцией цветов в узлах. Если вектора **X** и **Y** двумерные, то многоугольники строятся для каждой пары столбцов. Если вектор **C** есть вектор строка, длиной равной числу столбцов массивов **X** и **Y**, то каждый многоугольник закрашивается собственным цветом, режим **shading flat**. Если **C** двумерный массив, совпадающий по размерам с массивами **X,Y**, то цвет выбирается методом интерполяции, режим **shading interpolated**. Функция **fill** позволяет закрасить сразу конечное число многоугольников, а также позволяет вернуть вектор столбец дескрипторов объектов **patch** – закрашенный многоугольник. Использование функции **fill** задает свойству **FaceColor** объекта **patch** одно из значений **‘flat’, ‘interp’** или **[r g b].**

Закрашенный многоугольник в пространстве реализует функция **fill3**

fill3(x,y,z,<’цвет’>);
fill3(x,y,z,c);
fill3(X,Y,Z,C);
fill3(X1,Y1,Z1,C1,...,Xn,Yn,Zn,Cn);
h=fill3(...);.

Одномерные массивы **x,y,z** задают координаты вершин многоугольника. Поскольку многоугольник предполагается замкнутым, то начальная и конечная вершины соединяются линией. Цвет задается либо одним из символов **‘r’, ‘g’, ‘b’, ‘c’, ‘m’, ‘y’, ‘w’, ‘k’** либо **RGB** вектором строкой **[r g b]**, каждая компонента которого задается **0** или **1**. Если используется вектор цветов **c** той же длины, что и **x,y,z,** то многоугольник закрашивается цветом задаваемым вектором **c**. При этом вектор **c** масштабируется функцией **caxis,** и его элементы используются как индексы текущей палитры для задания цветов в вершинах многоугольника, а цвет внутри определяется билинейной интерполяцией цветов в узлах. Если вектора **X**,**Y,Z** двумерные, то многоугольники строятся для каждой тройки столбцов. Если вектор **C** есть вектор строка, длиной равной числу столбцов массивов **X**,**Y**,**Z,** то каждый многоугольник закрашивается собственным цветом, режим **shading flat**. Если **C** двумерный массив, совпадающий по размерам с массивами **X,Y,Z,** то цвет выбирается методом интерполяции, режим **shading interpolated**. Функция **fill** позволяет закрасить сразу конечное число многоугольников, а также позволяет вернуть вектор столбец дескрипторов объектов **patch** – закрашенный многоугольник. Использование функции **fill** задает свойству **FaceColor** объекта **patch** одно из значений **‘flat’, ‘interp’** или **[r g b].**

21 Функции работы с файлами

21.1 Основные функции

В системе **MatLab** имеются **C**-подобные функции для работы с файлами. По способу доступа к записям файла различают последовательные файлы и файлы прямого доступа. Последовательные файлы подразделяются по особенностям чтения и записи на файлы бинарные или двоичные и форматированные. При работе с файлами прямого доступа используется специальный указатель, устанавливаемый в начало файла либо в заданную позицию.

Для открытия файла используется функция **fopen**

fid=fopen('filename',permission);

[fid,message]=fopen('filename',permission,format);

fids=fopen('all');

[filename,permission,format]=fopen(fid);.

Входные параметры:

'filename' – имя либо полное имя файла, включая путь;

permission – режим работы с файлом.

Опция **permission** может принимать одно из следующих значений:

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

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

'w' – открытие нового файла либо удаление содержания существующего файла для последующей записи;

'w+' - открытие нового файла либо удаление содержания существующего файла для последующей записи и чтения;

-+\'W' – запись без автоматического заполнения, используется с **ЗУ** на магнитной ленте;

'a' – открытие нового либо существующего файла для записи или добавления;

'a+' - открытие нового либо существующего файла для записи или добавления и чтения;

'A' - добавление без автоматического заполнения, используется с **ЗУ** на магнитной ленте.

В операционных системах различающих текстовые и двоичные файлы к опции **permission** добавляется **'t'** или **'b'**, например, **'rt','rb'**. Режим **'b'** установлен по умолчанию.

Выходной параметр **fid** – идентификатор файла, используемый в функциях работы с файлами и принимающий следующие значения:

0 – при чтении с клавиатуры, установлена опция **'r'**;

1 – вывод на дисплей, установлена опция **'a'**;

2 – вывод сообщения об ошибке, установлена опция **'a'**;

-1 – вывод сообщения **message** о типе ошибки при неудаче открытия файла;

Значения, начиная с **3** и так далее, присваиваются идентификаторам в порядке открытия внешних файлов. Одновременно открытыми и доступными для чтения могут быть несколько файлов.

Вторая форма обращения позволяет дополнительно указать числовой формат данных входным параметром **format** и выходным параметром **message** вернуть сообщение об успешности открытия файла.

Опция **format** может принимать одно из следующих значений:

‘**native’** или **‘n’** – локальный машинный формат, установлен по умолчанию;

'ieee-le' или **'l'** - IEEE – плавающий формат, с укороченным числом байт под порядок;

'ieee-be' или **'b'** - IEEE плавающий формат, с увеличенным числом байт под порядок.

Определенные вызовы функций чтения **fread** и записи **fwrite** могут отменить числовой формат установленный функцией **fopen**.

Третья форма обращения возвращает вектор строку **fids**, содержащую идентификаторы всех открытых файлов, не включая **0, 1, 2**. Число элементов вектора равно числу открытых файлов.

Четвертая форма обращения возвращает полное **имя файла**, строку **permission** и строку **format**. При использовании недопустимых значений **fid** функция **fopen** возвращает пустые строки для всех выходных аргументов.

По окончании работы с файлом он закрывается функцией **fclose**

status=fclose(fid);

status=fclose(‘all’);.

Первое обращение закрывает конкретный файл, а второе обращение закрывает все открытые файлы. Возвращаемый параметр **status** принимает значение **0**, если закрытие завершилось успешно и **1**, в противном случае.

Открытие файлов для чтения и записи удобно реализовать вызовом стандартных меню выбора файлов **uigetfile, uiputfile**

[filename,pathname]=uigetfile('filter','title',x,y);

[filename,pathname]=uiputfile('initFile','title',x,y);.

Обе функции разворачивают стандартные панели выбора имени файла для чтения или записи. Выбор имени файла осуществляется либо с помощью мыши, из существующих файлов, либо набором с клавиатуры.

Выходные параметры **filename, pathname** позволяют сформировать полное имя файла для последующего открытия функцией **fopen.** Входные параметры **‘filter’, ‘initFile’** позволяют задать шаблон имен файлов высвечиваемых в меню выбора. Параметр **‘title’** задает заголовок меню выбора файлов. Параметры **x, y** определяют координаты вывода меню на экране, некоторые операционные системы не воспринимают эти параметры.

21.2 Операции с двоичными файлами

Операции с двоичными файлами представлены следующими стандартными функциями.

Чтение файла данных осуществляется функцией **fread**

[A,count]=fread(fid,size,precision);

[A,count]=fread(fid,size,precision,skip);.

Функция считывает двоичные данные из файла с идентификатором **fid** в матрицу **A**. Выходной параметр **count** возвращает число считанных элементов.

Параметр **size** задает количество считываемых данных. Если параметр **size** не указан, то считывание производится до конца файла. Параметр **size** можно задавать в следующем виде:

n – чтение **n** элементов в вектор столбец **A;**

inf – чтение элементов до конца файла и помещение их в вектор столбец **A** соответствующей длины;

[m,n] – считывает число элементов необходимых для заполнения матрицы **A** размером **m*n**.

Заполнение матрицы **A** производится по столбцам, недостающие элементы заполняются нулями. В случае ошибки, чтение останавливается на последнем считанном элементе.

Параметр **precision** задает точность считываемых значений в виде строки задающей форму интерпретации считываемых данных. Параметр **precision** может принимать следующие значения:

‘char’ -.символьный - **8** бит;

‘schar’ – знаковый символьный - **8** бит;

‘short’ – короткий целый – **16** бит;

'int' – целый – **16 или 32** бита;

'long' – длинный целый – **32** бита;

'float' – с плавающей точкой – **32** бита;

'double' – двойной, с плавающей точкой – **64** бита;

'uchar' – беззнаковый символьный – **8** бит;

'ushort' – беззнаковый целый – **16** бит;

'uint' – беззнаковый целый – **16 или 32** бита;

'ulong' – беззнаковый длинный целый – **32** бита;

'char' – символьный - **8** бит;

'float32' – с плавающей точкой – **32** бита;

'float64' – двойной, с плавающей точкой – **64** бита;

'int8' – короткий целый – **8** бит;

'int16' – целый – **16** бит;

'int32' – длинный целый – **32** бита;

'intN' – знаковый целый – **N** бит;

'uintN' – беззнаковый целый **N** бит;

'bitN' – знаковый битовый **N** бит;

'ubitN' –беззнаковый битовый **N** бит.

Параметр **skip** определяет число байт, которое необходимо пропустить после каждого считывания, используется при извлечении данный из несмежных областей в записях фиксированной длины, значение указывается в битах.

Запись файла данных осуществляется функцией **fwrite**

count=fwrite(fid,A,precision);

count=fwrite(fid,A,precision,skip);.

Функция записывает элементы матрицы **A** в файл с идентификатором **fid**, представляя их с заданной точностью **precision**. Данные записываются в файл по столбцам. Выходной параметр **count** возвращает число записанных элементов.

Параметр **skip** определяет число байт, которое необходимо пропустить перед каждой записью, используется при вставке данных в несмежные области в записях фиксированной длины, значение указывается в битах.

21.3 Операции с форматированными файлами

Операции с форматированными файлами представлены следующими стандартными функциями.

Считывание текущей строки из файла реализуют функции **fgetl, fgets**

line=fgetl(fid);

line=fgets(fid);

line=fgets(fid,nchar);.

Функции **fgetl** и **fgets** возвращают текущую строку **line** из файла с идентификатором **fid**, причем функция **fgetl** с удалением символа конца строки, а функция **fgets** – без удаления признака конца строки. Если функции **fgetl** и **fgets** обнаруживают признак конца файла, то они возвращают значение **1.** Параметр **nchar** указывает число возвращаемых символов текущей строки. При этом никакие дополнительные символы не считываются после признака конца строки или файла.

Запись форматированных данных в файл осуществляется функцией **fprintf**

count=fprintf(fid,format,A,...);

count=fprintf(format,A,...);.

Функция форматирует данные действительной части матрицы **A**, в соответствии с параметром **format** и записывает их в файл с идентификатором **fid**. Выходной параметр **count** возвращает число записанных байтов. Если входной параметр **fid** отсутствует, то вывод осуществляется на экран. Функция **fprintf** может быть использована для оформления выдачи на экран монитора в виде таблиц содержащих символьную и числовую информацию.

Параметр **format**, заданный в виде строки определяет систему обозначений, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Эта строка может содержать обычные буквы алфавита наряду со спецификаторами, знаками выравнивания и так далее.

Функция **fprintf**, за небольшими исключениями и расширениями аналогична соответствующей функции языка **ANSI C**.

Следующие таблицы описывают неалфавитные символы, встречающиеся в строке **format** и их значение**:**

ESC-последовательности:

\n – новая строка;

\t – горизонтальная табуляция;

\b – возврат на один символ;

\r – перевод каретки;

\f – новая страница;

\\ - обратный слеш;

\" или " – единственная кавычка;

%% - процент;

Спецификаторы:

%c – символ;

%d – десятичное целое;

%e – плавающий формат вещественного с показателем **e**;

%E – плавающий формат вещественного с показателем **E**;

%f – фиксированный формат вещественного;

%g – компактный формат вещественного с показателем **e,** (незначащие нули отсутствуют);

%G – компактный формат вещественного с показателем **E,** (незначащие нули отсутствуют);

%o – восьмеричное без знака;

%s – строка символов;

%u – десятичное без знака;

%x – шестнадцатеричное с использованием символов **a-f**;

%X – шестнадцатеричное с использованием символов **A-F;**

Дополнительные знаки:

- - выравнивание по левому краю;

+ - указывать знаки (**+/-**);

0 – использование нулей вместо пробелов;

n – цифра, определяющая минимальный размер поля вывода;

n.m – цифры, определяющие общий размер поля вывода и размер поля мантиссы.

Дополнительные знаки вставляются в спецификаторы между знаком **%** и другим идентификатором.

Чтение и форматирование данных из файла осуществляется функцией **fscanf**

A=fscanf(fid,format);

[A,count]=fscanf(fid,format,size);.

Функция читает данные из файла с идентификатором **fid**, форматирует в соответствии с параметром **format** и записывает их в матрицу **A.**

Вторая форма обращения позволяет дополнительно вернуть число считанных элементов **count** и указать количество элементов, подлежащих считыванию **size**. Параметр **size** может принимать следующие значения:

n – чтение **n** элементов в вектор столбец **A**;

inf – чтение элементов до конца файла и помещение их в вектор столбец **A** соответствующей длины;

[m,n] – считывает число элементов необходимых для заполнения матрицы **A** размером **m*n**.

Заполнение матрицы **A** производится по столбцам, **n** может принимать значение **Inf**, но не **m**.

Параметр **format** в виде строки включает в себя обычные символы и или спецификаторы, указывающие тип данных в виде символа **%**, опцию ширины поля и символы формата.

Допустимые символы формата при чтении:

%c – символ;

%d – десятичное целое;

%e, %f, %g – плавающий формат вещественного;

%i – целое число со знаком;

%o – восьмеричное без знака;

%s – строка символов;

%u – десятичное со знаком;

%x – шестнадцатеричное целое со знаком.

Между символом % и символом формата могут добавляться следующие символы:

***** - означает пропуск соответствующего значения в матрице;

n – число, указывающее максимальную ширину поля;

c – символ типа **h - short** и **l - long**, указывающие размер считанного данного.

21.4 Файлы прямого доступа

Работа с файлами прямого доступа обеспечивается функциями позиционирования специального указателя.

Функция **feof** проверяет, установлен ли указатель на признак конца файла

eofstat=feof(fid);.

Значение выходного параметра **eofstat=1** свидетельствует, что указатель соответствует признаку конца файла с идентификатором **fid**, в противном случае **eofstat=0**.

Функция **ferror** возвращает сведения об ошибке **message**

message=ferror(fid);.

Обращение типа

message=ferror(fid,'clear');

очищает индикатор ошибки файла с идентификатором **fid.**

Обращение типа

[message,errnum]=ferror(...);

возвращает дополнительно номер статуса ошибки **errnum** последней операции ввода-вывода данного файла. Если последняя операция ввода-вывода для данного файла была успешной, то значение **message=''**, а значение **errnum=0**. Значение **errnum** отличное от нуля, свидетельствует о произошедшей ошибке ввода-вывода, а параметр **message** содержит строку с информацией о характере возникшей ошибки.

Функция **fseek** устанавливает указатель в заданную параметром **offset** позицию, относительно **origin,** файла с идентификатором **fid**

status=fseek(fid,offset,origin);.

Входные параметры **offset** и **origin** могут иметь следующие значения:

offset>0 – изменяет позицию указателя на заданное число байт в направлении конца файла;

offset=0 – не изменяет позицию указателя;

offset<0 - изменяет позицию указателя на заданное число байт в направлении начала файла;

origin='bof' или –1 – обозначает начало файла;

origin='cof' или 0 – обозначает текущую позицию указателя;

origin='eof' или 1 – обозначает конец файла.

Выходной параметр **status** принимает значение **0**, если операция **fseek** завершилась успешно, и значение **1** – в противном случае. В случае ошибки используется функция **ferror** для получения более подробной информации.

Функция **ftell** устанавливает текущую позицию указателя для файла с идентификатором **fid**

position=ftell(fid);.

Выходной параметр **position** определяет позицию указателя в байтах относительно начала файла. В случае ошибки, **position** принимает значение равное **–1**. Для получения более подробной информации об ошибке используется функция **ferror**.

Установить указатель в начало файла с идентификатором **fid** позволяет функция **frewind**

frewind(fid);.

Система **MatLab** содержит также набор стандартных функций для работы со **специальными файлами данных.**

22 Форматирование данных

Операции, реализуемые функциями sprintf, sscanf по преобразованию строк и форматированному выводу.

Функция **sprintf** форматирует данные из матрицы **A**, в соответствии с параметром **format** и возвращает их в виде строковой переменной **s**

s=sprintf(format,A,...);

[s,errmsg]=sprintf(format,A,...);.

Вторая форма обращения дополнительно возвращает строку сообщения об ошибке **errmsg**. Параметр **format,** заданный в виде строки управляющих символов и спецификаторов, определяет систему обозначений, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Функция **sprintf** аналогична функции **fprintf** и отличается тем, что записывает форматированные данные матрицы **A** не в файл, а в строку **s**.

Функция **sscanf** считывает данные из строковой переменной **s**, форматирует их в соответствии с параметром **format** и создает из них матрицу **A**

A=sscanf(s,format);

A=sscanf(s,format,size);

[A,count,errmsg,nextindex]=sscanf(...);.

Вторая форма обращения содержит дополнительный входной параметр **size,** определяющий количество считываемых элементов строки **s**. Параметр **size** может принимать следующие значения:

n – чтение **n** элементов в вектор столбец **A**;

inf – чтение элементов до конца строки **s** и помещение их в вектор столбец **A** соответствующей длины;

[m,n] – считывает число элементов необходимых для заполнения матрицы **A** размером **m*n**.

Заполнение матрицы **A** производится по столбцам, **n** может принимать значение **Inf**, но не **m**.

Третья форма обращения позволяет с помощью выходных параметров вернуть количество успешно считанных элементов **count**, строку сообщений об ошибке **errmsg** и число **nextindex**, на единицу большее числа символов строки **s**.

Параметр **format** в виде строки включает в себя обычные символы и или спецификаторы, указывающие тип данных в виде символа **%**, опцию ширины поля и символы формата.

Заключение

Предлагаемое вниманию читателей справочное пособие по функциональной среде системы **MatLab** отражает основные сведения, необходимые для систематического использования при решении прикладных задач различного содержания. В пособии систематизированы основные сведения по программированию в системе **MatLab** и использованию богатой функциональной среды системы для анализа, расчета и визуализации результатов моделирования различных физических процессов на основе аналитических и численных методов, реализованных в системе.

Основное внимание уделено векторно-матричным операциям и функциям системы, обработке входных данных, заданных массивами, функциям графического отображения результатов моделирования, файловому вводу-выводу данных.

Справочное пособие может быть использовано при выполнении практических заданий, лабораторных работ, курсовых работ и проектов по изучаемым дисциплинам.

Список литературы

1. Дьяконов В.П. Справочник по применению системы **PC MatLab**. - М.: Физматлит, 1993.- 112 с.
2. Потемкин В.Г. Система **MatLab**. Справочное пособие. - М.: ДИАЛОГ-МИФИ, 1997.- 350 с.
3. Потемкин В.Г. Система **MatLab 5** для студентов. Справочное пособие. - М.: ДИАЛОГ-МИФИ, 1998.- 314 с.
4. Гультяев А.К. **MatLab 5.2.** Имитационное моделирование в среде **Windows**: Практическое пособие. - СПб.: Корона Принт, 1999.- 288 с.
5. Дьяконов В.П., Абраменкова И.В. **MatLab 5.0/5.3**. Система символьной математики. - М.: Нолидж, 1999.- 640 с.
6. Медведев В.С., Потемкин В.Г. **Control System Toolbox. MatLab 5** для студентов. / Под общ. ред. В.Г. Потемкина (Серия: Пакеты прикладных программ; Кн. 1). - М.: ДИАЛОГ-МИФИ, 1999.- 287 с.
7. Потемкин В.Г., Рудаков П.И. Система **MatLab 5** для студентов. - М.: ДИАЛОГ-МИФИ, 1999.- 448 с.
8. Потемкин В.Г. Система инженерных и научных расчетов **MatLab 5.x**: В 2-х т., том 1. - М.: ДИАЛОГ-МИФИ, 1999.- 336 с., том 2. - М.: ДИАЛОГ-МИФИ, 1999.- 304 с.
9. Гультяев А.К. Визуальное моделирование в среде **MatLab**: учебный курс. – СПб.: Питер, 2000. -432 с.
10. Мартынов Н.Н., Иванов А.П. **MatLab 5.x**. Вычисления, визуализация и программирование. - М.: КУДИЦ-ОБРАЗ, 2000.- 336 с.
11. Лазарев Ю.С. **MatLab 5.x**. - К.: BVH, 2000.- 384 с.
12. Потемкин В.Г. Введение в **MatLab**. – М.: ДИАЛОГ-МИФИ, 2000.– 247 с.
13. Потемкин В.Г. Инструментальные средства **MatLab 5.x**. – М.: ДИАЛОГ-МИФИ, 2000.- 336 с.
14. Рудаков П.И., Сафонов И.В. Обработка сигналов и изображений. **MatLab 5.x**. / Под общ. ред. В.Г. Потемкина (Серия: Пакеты прикладных программ; Кн. 2). - М.: ДИАЛОГ-МИФИ, 2000.- 416 с.
15. Гультяев А.К. **MatLab 5.3.** Имитационное моделирование в среде **Windows:** Практическое пособие. – СПб.: КОРОНА принт, 2001.- 400 с.
16. Говорухин В.Н., Цибулин В.Г. Компьютер в математическом исследовании. Учебный курс. – СПб.: Питер, 2001.- 624 с.
17. Дьяконов В.П. **MatLab**: учебный курс. – СПб.: Питер, 2001.- 560 с.
18. Дьяконов В.П., Абраменкова И.В., Круглов В.В. **MatLab 5.3.1** с пакетами расширений. - М.: Нолидж, 2001.- 880 с.
19. Дьяконов В.П. Компьютерная математика. Теория и практика. – М.: Нолидж, 1999, Нолидж, 2001.- 1296 с.

20. Дьяконов В.П., Круглов В.В. Математические пакеты расширения **MatLab.** Специальный справочник. – СПб.: Питер, 2001.- 480 с.

21. Дьяконов В.П. **MatLab 6**: учебный курс. – СПб.: Питер, 2001.- 592 с.

22. Мэтьюз Д.Г., Финк К.Д. Численные методы. Использование **MatLab.** - М.: Вильямс, 2001.- 720 с.

23. Чен К., Джиблин П., Ирвинг А. **MatLab** в математических исследованиях. - М.: Мир,2001.- 346 с.

24. Герман-Галкин С.Г. Компьютерное моделирование полупроводниковых систем в **MatLab 6.0:** Учебное пособие. – СПб.: КОРОНА принт, 2001.- 320 с.

25. Лавров К.Н., Цыплякова Т.П. Финансовая аналитика. **MatLab 6.** / Под общ. ред. В.Г. Потемкина (Серия: Пакеты прикладных программ; Кн. 3). – М.: ДИАЛОГ-МИФИ, 2001.- 363 с.

26. Герман-Галкин С.Г. Силовая электроника. Лабораторные работы на ПК. - СПб.: Учитель и ученик, КОРОНА Принт, 2002.- 304 с.

27. Дьяконов В.П. **Simulink 4.**Специальный справочник. – СПб.: Питер, 2002.- 528 с.

28. Дьяконов В.П., Круглов В.В. **MatLab.** Анализ, идентификация и моделирование систем. Специальный справочник. - СПб.: Питер, 2002.- 448 с.

29. Дьяконов В.П., Абраменкова И.В. **MatLab.** Обработка сигналов и изображений. Специальный справочник. - СПб.: Питер, 2002.- 608 с.

30. Мартынов Н.Н. Введение в **MatLab 6.** – М.: Кудиц-образ, 2002.- 352 с.

31. Ануфриев И.Е. Самоучитель **MatLab 5.3.6.x.** - СПб.: БХВ-Петербург, 2002.- 736 с.

32. Герман-Галкин С.Г. Линейные электрические цепи. Лабораторные работы. - СПб.: Учитель и ученик, КОРОНА Принт, 2002.- 192 с.

33. Кондрашов В.Е., Королев С.Б. **MatLab,** как система программирования научно-технических расчетов. – М.: Мир, 2002.- 350 с.

34. Медведев В.С., Потемкин В.Г. Нейронные сети. **MatLab 6.** / Под общ. ред. В.Г. Потемкина (Серия: Пакеты прикладных программ; Кн. 4). – М.: ДИАЛОГ-МИФИ, 2002.- 496 с.

35. Дьяконов В.П. **MatLab 6/6.1/6.5 + Simulink 4/5.** Основы применения. Полное руководство пользователя. – М.: СОЛОН-Пресс, 2002.- 768 с.

36. Дьяконов В.П. **MatLab 6/6.1/6.5 + Simulink 4/5** в математике и моделировании. Полное руководство пользователя. – М.: СОЛОН-Пресс, 2002.- 576 с.

37. Лавров Ю.Ф., Циплякова Т.П. Финансовая аналитика. **MatLab 6.** - М.: ДИАЛОГ-МИФИ, 2002.- 368 с.

38. Дэбни Дж., Харман Т. **Simulink 4**. Секреты мастерства. / Дж. Дэбни, Т. Харман. – М.: БИНОМ. Лаборатория знаний, 2003.- 403 с.

39. Леоненков А.В. Нечеткое моделирование в среде **MatLab** и **FuzzyTech.** – СПб.: БХВ-Петербург, 2003.- 736 с.

40. Поршнев С.В. Компьютерное моделирование физических процессов в пакете **MatLab**. Учебное пособие. – М.: Горячая линия – Телеком, 2003.- 592 с.

41. Потемкин В.Г. **MatLab 6:** среда проектирования инженерных приложений. – М.: ДИАЛОГ-МИФИ, 2003.- 448 с.

42 Черных И.В. **Simulink:** среда создания инженерных приложений. / Под общ. ред. В.Г. Потемкина – М.: ДИАЛОГ-МИФИ, 2003.- 496 с.

43. Кетков Ю.Л., Кетков А.Ю., Шульц М.М. **MatLab 6.x**: программирование численных методов. – СПб.: БХВ-Петербург, 2004.- 672 с.

44. Поршнев С.В. Вычислительная математика. Курс лекций. – СПб.: БХВ-Петербург, 2004.- 320 с.

45. Дьяконов В.П. **VisSim + Mathcad + MatLab.** Визуальное математическое моделирование. (Серия «Полное руководство пользователя»). – М.: СОЛОН – Пресс, 2004.- 384 с.

46. Новгородцев А.Б. Расчет электрических цепей в **MatLab:** Учебный курс. – СПб.: Питер, 2004.- 250 с.

47. Андриевский Б.Р. Избранные главы теории автоматического управления с примерами на языке **MatLab.** – СПб.: Наука, 1999.- 467 с.

48. Андриевский Б.Р., Фрадков А. Элементы математического моделирования в программных средах **MatLab 5 и SciLab**. - М.: Наука, 2001.- 286 с.

49. Филлипс Ч., Харбор Р. Системы управления с обратной связью. – М.: Лаборатория Базовых Знаний, 2001.- 616 с.

50. Дорф Р., Бишоп Р. Современные системы управления. – М.: Лаборатория Базовых Знаний, 2002.- 832 с.

51. Сергиенко А.Б. Цифровая обработка сигналов. – СПб.: Питер, 2003.- 608 с.

52. Основы цифровой обработки сигналов: Курс лекций. / Авторы: А.И. Солонина, Д.А. Улахович, С.М. Арбузов, Е.Б. Соловьева, И.И. Гук. – СПб.: БХВ-Петербург, 2003.- 608 с.

53. Алексеев Е.Р., Чеснокова О.В. **MatLab 7** / Алексеев Е.Р., Чеснокова О.В. – М.: НТ Пресс, 2006.- 464 с.

54. Половко А.М., Бутусов П.Н. MatLab для студента. – СПб.: БХВ-Петербург, 2005.- 320 с.

55. Плохотников К. Э. Вычислительные методы. Теория и практика в среде **MatLab**: курс лекций. Учебное пособие для вузов.– М.: Горячая линия–Телеком, 2009. – 496 с.

56. Дьяконов В.П. **Simulink 5/6/7.** Самоучитель.- М.: ДМК Пресс, 2008.- 784 с.

57. Шампайн Л.Ф. Решение обыкновенных дифференциальных уравнений с использованием **MatLab**. Учебное пособие из раздела Математика и статистика. **MatLab, MathCad, Maple.** – М.: Лань, 2009.- 304 с.

58. Дьяконов В.П. **MatLab и Simulink** для радиоинженеров. – М.: ДКМ Пресс, 2011.- 976 с.

59. Hunt, Brian R. **MatLab**: официальный учебный курс Кембриджского университета. Пер. с англ. – М.: Триумф, 2008.- 352 с.

60. Кривилёв А.В. Основы компьютерной математики с использованием системы **MatLab**. – М.: Лекс-Книга, 2005.- 496 с.