

**ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени АБУ РАЙХАНА БЕРУНИ**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ И ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ**

А. ГРИГОРЬЕВ

Программирование в MATLAB®

Под редакцией Р. ХАМДАМОВА

ТАШКЕНТ 2001

АННОТАЦИЯ	5
ЛЕКЦИЯ 1: «ПАКЕТ MATLAB®, ЕГО СТРУКТУРА, НАЗНАЧЕНИЕ. ОРГАНИЗАЦИЯ ДАННЫХ В MATLAB®»	6
1. Загрузка, настройка и работа в среде MATLAB®. Использование системы помощи (Help – системы). Завершение работы в MATLAB®	6
1.1 Запуск MATLAB®	6
1.2 Использование системы помощи (HELP – системы).	6
1.3 Работа в среде интерпретатора. Протокол сеанса работы.	7
1.4 Завершение работы в среде интерпретатора	9
2. Использование библиотек в MATLAB® (обзор)	10
2.1 Библиотека Wavelet	10
2.2 Библиотека Image Processing	11
2.3 Библиотека Signal Processing	12
2.4 Программа SIMULINK	12
2.5 Программа STATEFLOW	13
3. Представление и организация данных в MATLAB®. Типы данных. Создание матриц	14
3.1 Использование клавиатуры для ввода.	15
3.2 Загрузка из дисковых файлов.	16
3.3 Генерация числовых последовательностей средствами самого MATLAB®.	17
3.4 Удаление матриц	18
3.5 Типы данных.	19
3.5.1 Числовой тип данных	19
3.5.2 Строки и символы	19
4. Операции над матрицами. Специальные матрицы	20
4.1 Транспозиция матриц	21
4.2 Объединение массивов	21
4.3 Поворот матрицы	21
4.4 Поворот матрицы на 90°	22
Определения и термины	24
Новые команды	24
Контрольные вопросы	24
ЛЕКЦИЯ 2: «АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ. ОСНОВНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ»	25
1. Арифметические и логические операции в MATLAB®	25
1.1 Арифметические операции	25
1.1.1 Скалярные вычисления	25
1.1.2 Матричные вычисления	26
1.1.2.1 Сложение и вычитание матриц	26
1.1.2.2 Умножение матриц	27
1.1.3 Произведение массивов	28

1.1.4 Скалярное произведение двух векторов	29
1.1.5 Внешнее произведение двух векторов	29
1.2 Логические операции	29
1.2.1 Операции отношения	30
1.2.2 Логические операции	30
2. Основные математические функции в MATLAB®	31
2.1 Элементарные математические функции	32
2.1.1 Тригонометрические	32
2.1.2 Степенные (показательные)	32
2.1.3 Функции обработки чисел	33
2.1.4 Округление и остатки	33
2.2 Специализированные математические функции	33
2.2.1 Функции классической математики	33
2.2.2 Функции теории чисел	34
Контрольные вопросы	34
ЛЕКЦИЯ 3: «ТИПЫ ФАЙЛОВ В MATLAB®. ИНСТРУКЦИИ В MATLAB®»	35
1. Типы файлов в MATLAB®	35
1.1 Рабочие файлы (файлы – сценарии)	35
1.2 Файлы функций	36
2. Управляющие инструкции в MATLAB®	38
2.1 Оператор условного перехода (ветвления) IF	38
2.2 Оператор выбора SWITCH	39
2.3 Оператор WHILE	40
2.4 Оператор FOR	40
2.5 Команда управления BREAK	42
Определения и термины.	42
Новые команды	42
Контрольные вопросы	42
ЛЕКЦИЯ 4: «ГРАФИКА В MATLAB®»	43
1. Обзор графических возможностей MATLAB®	43
2. Простейшие команды двумерной графики	43
3. Декартовы (или X – Y) графики	44
3. Графики в полярных координатах	46
4. Гистограммы	47
5. Множественные графики	47
6. Рисование пар (точек) данных. Другие полезные возможности	48
7. Печать графиков	49

8. Преобразование координат	49
9. Трехмерная графика	49
Контрольные вопросы	51
ЛЕКЦИЯ 5: «РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ»	52
1. Постановка задачи решения систем линейных уравнений (СЛУ)	52
2. Обзор методов решения СЛУ	53
3. Функции MATLAB® для решения СЛУ	53
4. Решение квадратичных задач	
Контрольные вопросы	56
Список рекомендуемой литературы	56
ЛЕКЦИЯ 6: «ПЕРВИЧНАЯ ОБРАБОТКА РЕЗУЛЬТАТОВ НАБЛЮДЕНИЙ»	57
1. Введение в проблему статистической обработки	57
2. Метод структурной идентификации	57
3. Методы параметрической идентификации	59
3.1. Метод выбранных точек	59
3.2. Метод наименьших квадратов (МНК)	60
4. Основные функции MATLAB® для статистической обработки данных	61
5. Примеры решения практических задач статистического анализа	62
Контрольные вопросы	67
Список рекомендуемой литературы	67
ЛЕКЦИЯ 7: «РЕШЕНИЕ ЗАДАЧ ОПТИМИЗАЦИИ ФУНКЦИЙ ОДНОЙ ИЛИ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ»	68
1. Математическая постановка задачи оптимизации функций	68
2. Методы оптимизации функций (обзор)	68
3. Функции MATLAB® для решения задач оптимизации	69
3.1. Функции ядра MATLAB®	70
3.2. Обзор возможностей библиотеки (набора инструментов) Optimization	72
4. Примеры решения практических задач	73
Контрольные вопросы	77
Список рекомендуемой литературы	77

ЛЕКЦИЯ 8: «РЕШЕНИЕ ЗАДАЧ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ»	78
1. Обзор цифровой обработки сигналов (ЦОС)	78
1.1 Анализ сигналов	80
1.2 Фильтрация сигналов	80
2. Средства MATLAB® для решения задач ЦОС	81
3. Примеры решения некоторых стандартных задач ЦОС	82
Определения и термины	89
Контрольные вопросы	89
Список рекомендуемой литературы	89
ЛЕКЦИЯ 9: «РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ПОМОЩЬЮ MATLAB®»	90
1. Математическое описание дифференциальных уравнений, их классификация	90
2. Численные методы решения дифференциальных уравнений (обзор)	91
3. Решение дифференциальных уравнений средствами MATLAB®	93
3.1 Формулирование задачи решения ДУ	93
3.2 Функции для решения ДУ	96
3.3 Настройка параметра options функций решения ДУ	98
4. Примеры решения практических задач	99
Определения и термины	101
Контрольные вопросы	101
Список рекомендуемой литературы	102
ЛЕКЦИЯ 10: «РАБОТА С ПОЛИНОМАМИ. ИНТЕРПОЛИРОВАНИЕ ДАННЫХ»	103
1. Описание функций MATLAB® для работы с полиномами	103
2. Описание функций MATLAB® для интерполяции	105
Контрольные вопросы	108

Аннотация

Данный курс является прикладным и предназначен для ознакомления студентов магистратуры с возможностями пакета для решения научно – технических и вычислительных задач MATLAB®.

Курс рассчитан на 40 часов аудиторных занятий, из которых 20 часов составляют лекции и 20 часов – лабораторные работы.

Лекционный материал курса охватывает основные вопросы программирования в MATLAB® (представление и организация данных, арифметические и логические операции, основные встроенные функции, описание различных типов файлов, используемых в MATLAB®, алгоритмические конструкции языка программирования, преобразования матриц, а также графические возможности среды), а также вопросы прикладного применения пакета MATLAB® при решении практических (прикладных) задач.

Лабораторные работы предназначены для выработки и закрепления практических навыков в программировании и работе с пакетом MATLAB®. Задания к лабораторным работам следуют темам, рассматриваемым в лекциях, и позволяют студентам освоить использование возможностей пакета MATLAB®.

Данный курс подготовлен в рамках выполнения Согласованного Европейского Проекта (СЕП) T-JEP-10497-98 между Техническим Университетом Гамбург – Харбург (Германия), Университетом Твенте (Нидерланды), Ташкентским Государственным Техническим Университетом (Узбекистан) и Каражинским Инженерно – Экономическим Институтом (Узбекистан).

**Лекция 1: «Пакет MATLAB®, его структура, назначение.
Организация данных в MATLAB®»**

План лекции:

1. Загрузка, настройка и работа в среде MATLAB®. Использование системы помощи (Help – системы). Завершение работы в MATLAB®;
 2. Использование библиотек в MATLAB® (обзор);
 3. Представление и организация данных в MATLAB®. Типы данных. Создание матриц;
 4. Операции над матрицами. Специальные матрицы.
1. Загрузка, настройка и работа в среде MATLAB®. Использование системы помощи (Help – системы). Завершение работы в MATLAB®

MATLAB® – это профессиональный коммерческий пакет программ, разработанный компанией MathWorks и предназначенный для решения математических и технических задач любой сложности.

MATLAB® – это интерпретатор, имеющий собственный язык программирования, который (язык) позволяет пользователям создавать собственные приложения, предназначенные для решения узкоспециализированных задач.

Существуют версии MATLAB® для любых практически применимых платформ – Windows 95/98, UNIX, VAX (для Linux существует собственный пакет, аналогичный по своим возможностям MATLAB®).

1.1 Запуск MATLAB®

Для того, чтобы запустить MATLAB®, необходимо напечатать команду

`matlab`

в командной строке (в ОС UNIX) или воспользоваться значком MATLAB, который устанавливается в меню ПУСК -> ПРОГРАММЫ -> MATLAB при установке MATLAB® (в ОС Windows 95/98).

После этого на экране появится сообщение о загруженной версии и приглашение интерпретатора.

1.2 Использование системы помощи (HELP – системы).

В MATLAB® существует встроенная система помощи, которая становится доступной после загрузки среды.

Для того, чтобы запустить систему помощи, нужно напечатать команду

`help`

после приглашения интерпретатора. По этой команде на экране будет распечатано оглавление встроенного файла помощи в формате «MATLAB\<раздел>». Выбрав необходимый раздел, введите команду

`help <раздел>`

После этого на экране появится список функций, переменных и операторов, доступных в данном разделе. Для того, чтобы получить помощь уже по какой-либо конкретной функции, переменной или оператору, нужно ввести команду

`help <имя функции, переменной, оператора>`

На экране будет отображено содержимое HELP – файла для указанного параметра.

(Естественно, что если имя функции, переменной или оператора известно заранее, то предварительные команды поиска помощи по разделам можно пропустить и сразу задать поиск помощи по известному имени).

В том случае, если неизвестно имя функции, переменной или оператора, но известно какое-либо ключевое слово, характеризующее это имя (например, раздел математики или имя автора функции), то можно воспользоваться следующей командой

`lookfor <ключевое слово>`

Эта команда ищет указанное <ключевое слово> во всех разделах HELP-системы MATLAB®. По мере обнаружения вхождения заданного ключевого слова в тот или иной раздел HELP – файла имя этого раздела (или функции) вызывается на экран. После этого можно просмотреть содержимое помощи по каждому из разделов, найденных командой lookfor.

1.3 Работа в среде интерпретатора. Протокол сеанса работы.

Отличительной чертой MATLAB® является то, что в своей работе он использует только один способ организации данных – матрицы.

Матрица – набор числовых значений, которые организованы в определенное сочетание строк и столбцов

Число строк и столбцов может быть любым. Например, 3 строки, и 4 столбца определяют матрицу размером 3 x 4, которая имеет всего 12 элементов. Скаляр представлен матрицей размером 1 x 1.

Вектор из n измерений (или элементов) может быть представлен матрицей n x 1, которая называется вектор-столбец, или вектор может быть представлен матрицей 1 x n, которая в данном случае называется вектором-строкой из n элементов.

Именем матрицы может быть любая последовательность символов (включая буквы и цифры) длиной до 19 элементов, но всегда начинающаяся с буквы. Таким образом 'x1' может быть именем переменной, но '1x' - не может. 'Supercalafragilesticexpalladotious' может

быть именем переменной, однако только первые 19 символов будут сохранены! Следует помнить, что MATLAB® чувствителен к регистру. Например, для него переменные 'MID', 'Mid' и 'mid' различны.

Ниже даны примеры матриц, которые могли быть определены в MATLAB®. Обратите внимание, что набор числовых значений или элементов матрицы ограничен скобками [].

$c = 5.66$ или $c = [5.66]$ c - скаляр или матрица размером 1×1

$x = [3.5, 33.22, 24.5]$ x – вектор-строка или
матрица размером 1×3

$x1 = [2$
 5
 3
 $-1]$ $x1$ – вектор-столбец или
матрица размером 4×1

$A = [1 2 4$ матрица размером 4×3
 $2 -2 2$
 $0 3 5$
 $5 4 9]$

Индивидуальный элемент матрицы может быть определен с помощью обозначений $A(i,j)$ или A_i, j (в общем случае), или $A(4,1) = 5$ (для конкретного элемента).

В среде MATLAB® существует ряд команд, с помощью которых пользователь может просмотреть информацию о матрицах (точнее, переменных), созданных в процессе работы. К таким командам относятся следующие:

who - эта команда создает список всех переменных в рабочем пространстве (в памяти) MATLAB® и выводит этот список на экран;

whos - эта команда выводит на экран список используемых переменных и отображает размер этих переменных;

В MATLAB®, как и в других интерпретаторах, есть возможность вызова на экран определенного количества последних введенных команд. Для этого можно использовать клавишу $[\uparrow]$. Для исправления ошибок при введении команд можно использовать клавиши **[DELETE]** или **[BACKSPACE]** (однако, одна из этих двух клавиш может быть недействительна на отдельных системах).

Также при работе в среде MATLAB® у пользователя есть возможность загрузки в рабочее пространство (память) данных (переменных), созданных ранее и сохраненных в специальных файлах на диске, а также записи (сохранения) данных (переменных), созданных в текущем сеансе работы, в специальных дисковых файлах. (Подробнее команды загрузки данных рассмотрены во второй лекции данного курса. Команды сохранения данных рассмотрены в следующем пункте данной лекции).

Также иногда бывает необходимо вести протокол сеанса, т.е. регистрировать все введенные, выполненные команды, все сообщения и результаты. Для этого можно использовать команду

diary [<имя файла-протокола>]

По этой команде в текущем рабочем каталоге будет создан файл – протокол с указанным именем (по умолчанию – *diary*), в который в текстовом формате будут записаны все сообщения, отправляемые интерпретатором на стандартный вывод (чаще всего – монитор).

Режим протокола может быть отключен командой

diary off

Команда

diary on

включает этот режим. При этом для ведения протокола будет использоваться тот файл, который был задан в последнем вызове команды *diary* с параметром.

1.4 Завершение работы в среде интерпретатора

Для того, чтобы завершить работу с MATLAB®, можно воспользоваться любой из двух аналогичных по функциям команд:

quit

или

exit

Обе эти команды завершают сеанс работы с MATLAB® и передают управление в операционную систему.

Иногда бывает необходимо перед выходом из среды сохранить результаты текущего сеанса работы. Для этого можно использовать команду *save* (с параметрами или без них).

Если выполнить команду *save* в MATLAB® прежде, чем выйти, все матрицы, которые были определены и/или созданы, будут сохранены в файле с названием *matlab.mat*, который хранится в вашем рабочем каталоге. Если нужно сохранить определенные матрицы в течение любого сеанса работы, это можно выполнить с помощью команды *save* с именем матрицы. Формат этой команды дан ниже:

save <имя_файла> x y z

Эта команда сохраняет матрицы *x*, *y* и *z* в файл с названием <имя_файла>.mat. Затем сохраненные данные можно использовать в дальнейших расчетах.

С помощью команды save данные можно сохранять в двух форматах:

в формате ASCII – для этого при вызове команды save после имен сохраняемых матриц нужно указать параметр –ascii;

в двоичном формате – этот формат используется по умолчанию.

Данные можно сохранять в формате ASCII в том случае, если они будут затем распечатаны на принтере или на экране.

ПРИМЕЧАНИЕ - При использовании MATLAB® на рабочих станциях, файлы будут сохранены в каталоге, из которого загружен MATLAB® (из которого задана команда matlab). При использовании рабочей станции создайте для MATLAB® подкаталог с названием 'MATLAB' (или подобным). После этого сохраняйте все файлы и проводите все сеансы работы с MATLAB® именно в этом подкаталоге.

2. Использование библиотек в MATLAB® (обзор)

Приступая к описанию библиотек пакета MATLAB®, их возможностей, следует сказать о структуре пакета MATLAB®.

MATLAB® состоит как бы из двух больших частей: ядра и «подключаемых» библиотек (или “toolboxes” = «комплектов инструментов»). Ядро MATLAB® предоставляет большинство функций и операций общего назначения, а библиотеки содержат узкоспециализированные функции, которые позволяют пользователям выполнять вычисления и обработку данных в строго определенной области.

Существует большое количество библиотек, созданных как компанией MathWorks, так и самими пользователями MATLAB®. Функции в этих библиотеках решают задачи в области математической логики, теории управления, нейронных сетей, обработки сигналов, других современных практических областей.

Рассмотрим некоторые из стандартных библиотек.

2.1 Библиотека Wavelet

Комплект инструментов (библиотека) Wavelet - коллекция функций, сформированная на базе MATLAB®. Эта коллекция (набор) обеспечивает инструментальные средства для анализа и синтеза сигналов и изображений, использующих элементарные волны и пакеты элементарной волны в пределах структуры MATLAB®.

Комплект инструментов обеспечивает две категории инструментальных средств:

- функции командной строки
- графические интерактивные инструментальные средства

Первая категория инструментальных средств составлена из функций, которые могут быть вызваны непосредственно из командной строки или из других приложений. Большинство этих функций - M - файлы, которые

реализуют специализированный анализ элементарной волны или алгоритмы синтеза. Можно изучить код этих функций с помощью команды

`type <имя_функции>`

Можно просмотреть заголовок функции - вспомогательную часть - используя инструкцию

`help <имя_функции>`

Полный список функций библиотеки Wavelet становится доступным с помощью команды

`help wavelet`

Можно изменять работу любой функции в библиотеке (наборе инструментов) путем копирования, переименования M – файлов и дальнейшего изменения созданной копии. Можно также расширять библиотеку (комплект инструментов) путем добавления собственных M - файлов.

Вторая категория инstrumentальных средств - коллекция графических инструментальных средств интерфейса, которая предоставляет доступ к обширным функциональным возможностям. Обратиться к этим инструментальным средствам можно, печатая команду

`wavemenu`

в командной строке.

2.2 Библиотека Image Processing

Комплект инструментов Image Processing - набор функций, которые расширяют возможности MATLAB®. Комплект инструментов (библиотека) поддерживает широкий диапазон операций по обработке изображений, включая:

- Геометрические операции;
- Линейная фильтрация и разработка фильтров;
- Преобразования;
- Анализ изображений;
- Операции с двоичными изображениями;

Версия II этой библиотеки предлагает много преимуществ по сравнению с версией I, включая поддержку для 8-разрядных данных изображения и многомерных массивов. Многие из функций версии I были переписаны с целью повышения быстродействия и использования меньшего объема памяти. Кроме того, имеется множество новых функций, которые расширяют возможности библиотеки.

Для того, чтобы получить список всех функций в библиотеке Image Processing, напечатайте такую команду:

`helpwin images/Contents` (или `help images/Contents`)

2.3 Библиотека Signal Processing

Комплект инструментов (библиотека) Signal Processing – набор инструментальных средств, сформированных в MATLAB®. Комплект инструментов поддерживает широкий диапазон операций по обработке сигналов – от генерации волн до разработки и реализации фильтров, параметрического моделирования и спектрального анализа. Комплект инструментов предоставляет две категории инструментальных средств:

- Функции обработки сигналов;
- Графические, интерактивные инструментальные средства;

Первая категория инструментальных средств составлена из функций, которые могут быть вызваны из командной строки или из других приложений.

Вторая категория – это набор интерактивных инструментальных средств, которые позволяют обращаться ко многим функциям через графический интерфейс пользователя (Графический Интерфейс Пользователя = GUI). Основанные на GUI (Графическом Интерфейсе Пользователя) инструментальные средства обеспечивают интегрированную среду для проектирования фильтра, анализа и выполнения, а также для исследования сигнала и редактирования. Например, с помощью средств графического интерфейса пользователя можно:

- Использовать «мышь», чтобы графически отредактировать реакцию (характеристику) фильтра или измерить наклон сигнала с помощью визуальных (экранных) линеек;
- Исполнить (проиграть) сигнал на звуковых аппаратных средствах вашей системы, выбирая пункт меню или нажимая соответствующую комбинацию клавиш;
- Настроить параметры и метод вычисления спектра сигнала, используя раскрывающиеся меню.

2.4 Программа SIMULINK

В последние несколько лет SIMULINK стал наиболее широко используемым пакетом программ в науке и промышленности для моделирования и симуляции динамических систем.

Используя SIMULINK, можно легко формировать модели из образцов или добавлять компоненты к уже существующим моделям. Симуляция интерактивна, поэтому можно изменять параметры в процессе работы и немедленно смотреть результаты изменений. Имеется прямой доступ к всем инструментальным средствам анализа MATLAB®, так что можно получить результаты, проанализировать их и построить все необходимые графики.

С SIMULINK можно исследовать более реалистичные нелинейные модели, которые учитывают, например, трение, воздушное сопротивление, проскальзывание механизма, жесткие остановки и другие вещи, которые описывают реальные явления.

SIMULINK - пакет программ для моделирования, симуляции и анализа динамических систем. Этот пакет поддерживает линейные и нелинейные

системы, смоделированные в непрерывном времени, заданном интервале времени или сочетании обоих. Системы могут быть также многоскоростными, то есть иметь различные части, которые выбраны или обновлены с различными скоростями.

Для моделирования SIMULINK обеспечивает графический интерфейс пользователя для формирования модели как блок-схемы, с использованием операции мыши «click – and – drag». С этим интерфейсом можно «рисовать» модели также, как с использованием карандаша и бумаги (или как большинство учебников их изображает). Эта возможность далека от возможностей предыдущих пакетов моделирования, которые требовали, чтобы дифференциальные и разностные уравнения были сформулированы на определенном языке или в виде программы. SIMULINK включает библиотеку, в которой есть разные блоки: приемники, источники, линейные и нелинейные компоненты, соединители. Можно также настраивать и создавать собственные блоки.

Модели имеют иерархическую структуру, так что можно формировать модели, используя нисходящий и восходящий подходы. Можно рассматривать систему на высоком уровне, затем двойным щелчком на блоках понижаться сквозь уровни, чтобы получить доступ к увеличивающимся уровням деталей модели. Этот подход обеспечивает понимание того, как модель организована и как взаимодействуют ее части.

После того, как модель определена, ее можно запустить на исполнение (симулировать), используя выбор из методов интегрирования, или из меню SIMULINK, или вводя команды в командном окне MATLAB®. Меню особенно удобны для интерактивной работы, в то время как работа из командной строки очень полезна при выполнении пакетного моделирования (например, если идет моделирование метода Монте – Карло или нужно «пропустить» параметр через весь диапазон значений). Используя специальные демонстрационные блоки, можно видеть результаты симуляции, в то время как симуляция еще выполняется. Кроме того, можно изменять параметры и немедленно смотреть, какой результат это дает (исследование типа «А что, если . . . ?»). Результаты моделирования (симуляции) могут быть помещены в рабочее пространство MATLAB® для последующей обработки и визуализации.

Инструменты анализа моделей включают линеаризацию и средства подстройки, которые могут быть доступны через командную строку MATLAB®, плюс многие инструментальные средства MATLAB® и его библиотеки. Благодаря тому, что MATLAB® и SIMULINK интегрированы, можно моделировать, анализировать, и исправлять модели в среде в любой точке.

2.5 Программа STATEFLOW

STATEFLOW - мощный графический инструмент для проектирования и развития для сложных проблем управления и контроля. STATEFLOW поддерживает систему обозначений в блок-схемах также, как и систему обозначений в изменениях состояний. Используя STATEFLOW, можно:

- Визуально моделировать и симулировать комплексные реактивные системы, основанные на теории конечных автоматов;

- Проектировать и развивать детерминированные системы централизованного контроля;
- Использовать систему обозначений в блок-схемах и систему обозначений в изменениях состояний в одной и той же диаграмме STATEFLOW;
- Легко изменять проект, оценивать результаты и проверять поведение системы в любой стадии проекта;
- Использовать преимущество интегрированности с MATLAB® и SIMULINK, чтобы моделировать, симулировать и анализировать систему.

Система обозначений в блок – схемах, по существу, есть логика, представленная без использования состояний. В некоторых случаях использование системы обозначений в блок - схемах является представлением, более близким к логике системы, и позволяет избежать использования ненужных состояний. Система обозначений в блок – схемах - эффективный способ представить общие структуры кода, как циклы for и конструкции условного оператора if – then - else.

STATEFLOW также обеспечивает ясные, краткие описания поведения сложных систем, используя теорию конечных автоматов, систему обозначений в блок – схемах и диаграммы изменений состояний. STATEFLOW сводит ближе друг к другу технические требования к системе и ее проект. Это просто - создать проект, рассмотреть различные сценарии и выполнять итерации до тех пор, пока диаграмма STATEFLOW не смоделирует желаемое поведение.

Примеры приложений (применений) пакета STATEFLOW, которые используют возможности этой программы:

- Внедренные системы:
 - Авиация (самолеты);
 - Автомобильная промышленность (автомобили);
 - Передача данных (например, алгоритмы маршрутизации);
 - Коммерческие (компьютерные внешние устройства, приборы, и т.д.);
 - Программируемые логические контроллеры (управление производственным процессом);
- Человеко-машинный интерфейс:
 - Графический интерфейс пользователя;
- Гибридные системы:
 - Системы управления воздушным движением (Обработка цифровых сигналов + управление + человеко–машинный интерфейс);

STATEFLOW состоит из следующих компонентов:

- Графический редактор STATEFLOW;
- Проводник STATEFLOW;
- Средство поиска STATEFLOW;
- Генератор объектного кода моделирования STATEFLOW;
- Отладчик STATEFLOW.

3. Представление и организация данных в MATLAB®. Типы данных. Создание матриц

Как уже говорилось выше, единственным способом организации данных в MATLAB® является матрица. Скалярные выражения считаются матрицами размером 1x1. Определение, классификация и примеры различных типов матриц даны в лекции 1.

Рассмотрим способы создания матриц в MATLAB®. Существуют три основных способа создания матриц:

прямой ввод с клавиатуры;

загрузка из дисковых файлов;

генерация числовых последовательностей средствами самого MATLAB®.

3.1 Использование клавиатуры для ввода.

Матрица может быть определена различными выражениями MATLAB®. Ниже даны примеры для вектор-строки x размером 1x3 с такими элементами: $x(1) = 2$, $x(2) = 4$ и $x(3) = -1$.

$x = [2 \ 4 \ -1]$ или $x = [2,4, -1]$

(Нажатие клавиши [ENTER] следует за каждой из вышеупомянутых инструкций MATLAB®.)

Заметьте, что скобки должны использоваться, чтобы открывать и закрывать набор чисел. Также обратите внимание, что запятые или пробелы могут использоваться как разделители между полями, определяющими элементы матрицы. Пробелы, используемые вокруг знака равенства, унарного знака и скобок, лишние; однако они иногда делают инструкцию более читаемой.

Матрица y размером 2x4, чьи элементы - $y(1,1) = 0$, $y(1,2) = y(1,3) = 2$, $y(1,4) = 3$, $y(2,1) = 5$, $y(2,2) = -3$, $y(2,3) = 6$ и $y(2,4) = 4$, может быть определена так:

$y = [0 \ 2 \ 2 \ 3$
 $\quad \quad \quad 5 \ -3 \ 6 \ 4]$

или

$y = [0 \ 2 \ 2 \ 3 ; 5 \ -3 \ 6 \ 4]$

Точка с запятой ";" используется, чтобы разделять строки матрицы, когда они записаны на одной и той же строке при вводе данных.

Элементы матрицы могут быть определены алгебраическими выражениями, помещенными в соответствующее местоположение элемента. Таким образом, выражение

$a = [\sin(\pi/2) \ \sqrt{2} \ 3+4 \ 6/3 \ \exp(2)]$

определяет матрицу

$$a = [1.0000 1.4142 7.0000 2.0000 7.3891]$$

Матрица может быть определена путем увеличения предварительно заданной матрицы. Используя матрицу x , определенную ранее, запишем выражение:

$$x1 = [x 5 8]$$

Результатом этого выражения является матрица

$$x1 = [2 \ 4 \ -1 \ 5 \ 8]$$

Выражение $x(5) = 8$ создает матрицу

$$x = [2 \ 4 \ -1 \ 0 \ 8]$$

Обратите внимание, что элементу $x(4)$ присвоено значение "0", которое не было явно определено.

Используя определение матрицы, y , данное выше, выражения

$$c = [4 \ 5 \ 6 \ 3]$$

$$z = [y; c]$$

создают матрицу z , имеющую вид

$$\begin{bmatrix} 0 & 2 & 2 & 3 \\ 5 & -3 & 6 & 4 \\ 4 & 5 & 6 & 3 \end{bmatrix}$$

Обратите внимание, что каждый раз, когда матрица определена, и нажата клавиша [ENTER], MATLAB® отображает на экране результат. Отменить это «эхо» можно с помощью знака «;», поставленного после выражения, перед нажатием клавиши [ENTER].

$$z = [y; c];$$

3.2 Загрузка из дисковых файлов.

Матрицы можно формировать путем загрузки данных из дисковых файлов. Это можно выполнить при помощи команды `load`, формат которой дан ниже:

`load <имя_файла>`

Если параметр команды опущен, то данные будут загружаться из файла с именем `matlab.mat`.

Загружаемые данные могут быть предварительно сохранены как в текстовом (ASCII) формате, так и в двоичном (внутреннем формате MATLAB®).

Также есть возможность избирательной загрузки матриц в память из файлов. Для этого используется такой формат команды load:

`load <имя_файла> x y z`

По этой команде заданные в качестве параметров команды матрицы x, y и z загружаются из указанного файла в рабочее пространство (память).

3.3 Генерация числовых последовательностей средствами самого MATLAB®.

Для того, чтобы создать матрицу генерацией, можно использовать специальный оператор «`:`».

Если два целых числа отделены двоеточием, MATLAB® генерирует все целые числа между этими двумя целыми числами. Например, команда

`a = 1:8`

генерирует вектор-строку

`a = [1 2 3 4 5 6 7 8].`

Если три числа (целых или нет) отделены двумя двоеточиями, среднее число интерпретируется как шаг, а первое и третье интерпретируются как границы. Таким образом, команда

`b = 0.0: .2: 1.0`

генерирует вектор-строку

`b = [0.0 .2 .4 .6 .8 1.0]`

Оператор «`:`» можно также использовать для создания вектора из уже существующей матрицы. Таким образом, если

`x = [2 6 8
0 1 7
-2 5 -6],`

то команда `y = x(:, 1)` создает вектор-столбец

`y = [2
0
-2]`

и $y = x(:, 2)$ создает

$$y = [6
1
5]$$

Команда $z = x(1, :)$ создает вектор-строку

$$z = [2 6 8]$$

Оператор двоеточия полезен при извлечении меньших матриц из больших. Если матрица размером 4 x 3 определена как

$$c = [-1 0 0
1 1 0
1 -1 0
0 0 2]$$

тогда $d1 = c(:, 2 : 3)$ создает матрицу, для которой используются элементы всех строк от 2-ого до 3-ого столбца. Результат - матрица размером 4 x 2

$$d1 = [0 0
1 0
-1 0
0 2]$$

Команда $d2 = c(3 : 4, 1 : 2)$ создает матрицу размером 2 x 2, в которой строки определены 3-ей и 4-ой строками матрицы с и столбцы определены 1-ым и 2-ым столбцами матрицы с:

$$d2 = [1 -1
0 0]$$

3.4 Удаление матриц

В данном разделе речь пойдет об удалении матриц (или переменных) из рабочего пространства (памяти) при работе MATLAB®. (Удаление матриц или переменных, сохраненных в дисковых файлах, не представляет интереса с точки зрения программирования, поскольку это может быть выполнено при помощи средств операционной системы).

Для удаления матриц из рабочего пространства используется команда `clear`, формат которой следующий:

```
clear x
```

По этой команде из рабочего пространства (памяти) удаляется указанная в качестве параметра матрица `x`. (ПРИМЕЧАНИЕ: можно задавать несколько матриц, разделенных пробелом).

Команда

`clear`

(без параметров) стирает ВСЕ матрицы из оперативной памяти, ПО-ЭТОМУ БУДЬТЕ ОЧЕНЬ ВНИМАТЕЛЬНЫ ПРИ ИСПОЛЬЗОВАНИИ ЭТОЙ КОМАНДЫ.

3.5 Типы данных.

В MATLAB® используются следующие типы данных:

- числовой;
- строки и символы;
- объекты.

Также пользователь по необходимости может определить собственные типы данных.

Рассмотрим эти типы данных более подробно.

3.5.1 Числовой тип данных

В MATLAB® можно оперировать двумя типами чисел – действительными и комплексными. Комплексные числа представляются в форме $a + ib$, где действительные числа a и b – соответственно действительная и мнимая части данного комплексного числа, а символ « i » (также можно использовать символы « I », « j » и « J ») обозначает мнимую единицу.

Числа любого типа могут быть содержимым матриц, векторов и скалярных величин. Для того, чтобы обозначить комплексное число, его нужно записать в указанной выше форме. (Обратите внимание на то, что «признак комплексности» - символ « I », « J », « i » или « j » должен быть записан слева или справа от числа, обозначающего мнимую часть, БЕЗ ПРОБЕЛА. В противном случае MATLAB® выдаст сообщение об ошибке).

В памяти все числа хранятся в виде чисел двойной точности. Границы интервала, в котором могут быть определены числа, а также машинная точность, задаются системными переменными `realmin`, `realmax` и `eps`.

3.5.2 Строки и символы

В MATLAB® под строками понимается последовательность символов, ограниченная апострофами или кавычками. Примеры строк:

`a='qwerty'`

`b='MATLAB'`

Для того, чтобы объединить несколько строк, можно использовать квадратные скобки «[]», которые используются при определении матриц. Например, выражение

`str1=['This ', 'is ', 'string']`

дает строку

`str1='This is string'`

Некоторые функции для создания и обработки строк в MATLAB®:

`blanks(n)` – возвращает строку из n пробелов;

`int2str(n)` – превращает целое число в строку;

`num2str(n)` – превращает действительное число в строку;

`deblank(s)` – удаляет ненужные пробелы из указанной строки s;

`index(s,t)` – возвращает позицию, в которой подстрока t появляется в указанной строке s впервые (ПРИМЕЧАНИЕ: эта функция неприменима для массивов строк). Если заданной подстроки нет, то возвращается 0;

`rindex(s,t)` – возвращает позицию, в которой подстрока t появляется в указанной строке s последний раз (ПРИМЕЧАНИЕ: эта функция неприменима для массивов строк). Если заданной подстроки нет, то возвращается 0;

`strcmp(s1,s2)` – возвращает 1, если две указанные строки s1 и s2 одинаковы. В противном случае возвращает 0;

`strrep(s,x,y)` – заменяет все вхождения подстроки x в строке s на строку y;

Отдельно следует сказать о функциях преобразования чисел в различных системах счисления, записанных в виде строк, в числа в других системах. К таким функциям относятся:

`bin2dec(s)` – возвращает десятичное число, соответствующее двоичному числу, представленному в виде строки;

`dec2bin(m)` – возвращает двоичное число (в виде строки), соответствующее заданному неотрицательному десятичному числу;

`dec2hex(n)` – возвращает шестнадцатиричное число (в виде строки), соответствующее заданному неотрицательному десятичному числу;

`hex2dec(s)` – возвращает десятичное число, соответствующее шестнадцатиричному числу, представленному в виде строки;

`str2num(s)` – превращает строку s в число;

4. Операции над матрицами. Специальные матрицы

В MATLAB® над матрицами, помимо рассмотренных выше арифметических операций, можно еще выполнять также и специальные операции (или преобразования). Рассмотрим эти операции подробнее.

4.1 Транспозиция матриц

Транспонированную матрицу можно получить, если поменять местами строки и столбцы. Оператор MATLAB®, который выполняет транспозицию – это апостроф «.'». Используя матрицу $G = [1 \ 3 \ 5; \ 2 \ 4 \ 6].'$, получаем

$$G' = [1 \ 2 \\ 3 \ 4 \\ 5 \ 6]$$

Обратите внимание, что операция транспозиции создает из матрицы размером $m \times n$ матрицу размером $n \times m$.

4.2 Объединение массивов

Для того, чтобы объединить несколько массивов в одну матрицу, используется команда

$$C = \text{cat}(<\text{размерность}>, A_1, A_2, \dots)$$

Эта команда соединяет заданные массивы A_1, A_2 и т.д. вдоль указанной размерности.

Пример: команда $\text{cat}(2, A, B)$ эквивалентна команде $[A, B]$, а команда $\text{cat}(1, A, B)$ – команде $[A : B]$.

Примечание: использование при создании списков точки (например, $\text{cat}(<\text{разм.}>, C{:})$ или $\text{cat}(<\text{разм.}>, C.\text{поле})$) является удобным способом соединения (конкатенации) ячейки структурированного массива, содержащей числовые матрицы, в единую матрицу.

4.3 Поворот матрицы

Другой манипуляцией над матрицей является поворот матрицы. Для выполнения этого действия в MATLAB® существуют две команды:

$$B = \text{fliplr}(A)$$

и

$$B = \text{flipud}(A)$$

Команда $B=\text{fliplr}(A)$ выполняет поворот матрицы в направлении слева направо, а команда $B=\text{flipud}(A)$ – поворот в направлении снизу вверх.

Пример: задана матрица A

$$A = \\ 1 \ 4 \\ 2 \ 5 \\ 3 \ 6$$

Команда $B=flipud(A)$ производит матрицу B с такими элементами:

3 6
2 5
1 4

Команда $B=fliplr(A)$ (с тем же значением матрицы A) производит матрицу B с такими элементами:

4 1
5 2
6 3

4.4 Поворот матрицы на 90°

Отдельно следует сказать о возможности поворота матрицы на 90° . Для этого следует использовать команду

$B = rot90(A)$

Эта команда поворачивает указанную матрицу на 90° против часовой стрелки.

Пример: пусть дана матрица X

$X =$
1 2 3
4 5 6
7 8 9

После выполнения команды $Y=rot90(X)$ в качестве результата получаем матрицу

$Y =$
3 6 9
2 5 8
1 4 7

Помимо вышеназванных специальных операций, в MATLAB® существует также возможность создания так называемых специальных матриц, т.е. матриц с наперед заданной структурой и правилами генерации элементов.

Эти специальные матрицы можно условно поделить на две группы:

- простые матрицы с равными элементами;
- специальные матрицы, применяемые в классической математике.

Для создания специальных матриц используются отдельные команды, которые фактически и генерируют матрицу необходимого типа.

Рассмотрим команды для генерации матриц заданного типа или заданной структуры.

eye(m,n) – генерирует идентичную матрицу (т.е. матрицу, у которой элементы на главной диагонали – единицы, а остальные элементы – нули) размером m x n;

linspace(a,b[,n]) – генерирует матрицу с n элементами, которые равномерно распределены в интервале [a;b]. Если параметр n не указан, то по умолчанию он устанавливается равным 100;

ones(m,n) – генерирует матрицу размером m x n, у которой все элементы – единица;

rand(m,n) – генерирует матрицу размером m x n, элементы которой являются случайными числами, равномерно распределенными в интервале (0;1);

zeros(m,n) – генерирует матрицу размером m x n, у которой все элементы – нули;

: (двоеточие) – генерирует вектор, элементы которого распределены равномерно, с заданным шагом, внутри определенного интервала [a;b]. Пример использования оператора «**:**»:

A=23:0.7:45 – сгенерирован вектор A, элементы которого с шагом 0.7 равномерно распределены в интервале [23;45].

hilb(n) – генерирует матрицу Гильберта порядка n. (Элементы матрицы Гильберта описываются формулой $H(i, j) = 1 / (i + j - 1)$);

invhilb(n) – генерирует обратную матрицу Гильберта;

magic(n) – генерирует матрицу порядка n, представляющую собой «матический квадрат» - матрицу, у которой сумма элементов по строкам равна сумме элементов по столбцам;

pascal(n) – генерирует матрицу Паскаля – симметричную положительную матрицу, элементами которой являются целые числа из треугольника Паскаля;

При анализе полученных результатов пользователю бывает необходимо получить определенные сведения об интересующей его матрице (или массиве). Для этого в MATLAB® существует несколько специальных функций:

size(A) – возвращает вектор – строку с двумя элементами, которые показывают размерность заданной матрицы: первый показывает количество строк в матрице A, второй – количество столбцов;

length(A) – возвращает длину вектора A;

ndims(A) – возвращает число размерностей в матрице A;

isempty(A) – логический предикат, проверяет заданный массив на «пустоту»: возвращает «1», если массив «пуст» (т.е. не имеет элементов вообще), и «0» - в любом другом случае;

isequal(A, B) - логический предикат, проверяет два заданных массива на эквивалентность (массивы считаются эквивалентными, если они имеют одинаковый размер и одинаковое содержание). Возвращает «1» в случае эквивалентности массивов, и «0» - в любом другом случае;

isnumeric(A) – логический предикат, проверяет тип заданного массива A. Возвращает «1», если заданный массив имеет числовой тип, и «0» - во всех других случаях.

Определения и термины

MATLAB® – это профессиональный коммерческий пакет программ, разработанный компанией MathWorks и предназначенный для решения математических и технических задач любой сложности.

Матрица – набор числовых значений, которые организованы в определенное сочетание строк и столбцов.

Вектор-столбец - вектор из n измерений (или элементов), представленный матрицей $n \times 1$.

Вектор-строка - вектор из n элементов, представленный матрицей $1 \times n$.

Новые команды

matlab, help, lookfor, who, whos, diary, quit, exit, save, type, wavemenu, load, clear.

Контрольные вопросы

1. Что такое MATLAB®, каковы его функции? Как можно загрузить MATLAB® (в разных операционных системах)?
2. Приведите возможные форматы команды help и объясните их назначение.
3. Что такое матрица, вектор – строка и вектор – столбец?
4. Для чего служат команды who и whos?
5. Опишите команду diary.
6. Опишите команду save (полный формат, все параметры).
7. Для чего служат библиотеки в MATLAB®?
8. Опишите назначение библиотек Wavelet, Image Processing, Signal Processing.
9. Опишите назначение программ SIMULINK и STATEFLOW.
10. Перечислите методы создания матриц в MATLAB® и приведите примеры на использование каждого метода.
11. Как можно удалить переменную (-ые) из памяти?
12. Какие типы данных есть в MATLAB®? Что такое «транспозиция матрицы»? Приведите примеры транспонированных матриц.
13. Для чего служат команды flipud и fliplr?
14. Приведите все команды для создания матриц определенного типа (или определенной структуры).
15. Приведите все предикаты и функции, которые можно использовать для определения характеристик матрицы или массива.

Лекция 2: «Арифметические и логические операции. Основные математические функции»

План лекции:

1. Арифметические и логические операции в MATLAB®;
2. Основные математические функции в MATLAB®.

1. Арифметические и логические операции в MATLAB®

1.1 Арифметические операции

В MATLAB® арифметические операции бывают двух типов:

- скалярные – это обычные операции, выполняемые над скалярными величинами;
- матричные – это операции, выполняемые над матрицами. Правила их выполнения отличаются от правил выполнения обычных (скалярных) операций.

1.1.1 Скалярные вычисления

В MATLAB® используются обычные арифметические операции, используемые в электронных таблицах и языках программирования типа БЕЙСИК. Единственное отличие – это существование правого и левого делений.

Арифметические операции:

- + Сложение
- Вычитание
- * Умножение
- / Правое деление (a/b означает «деление a на b »)
- \ Левое деление ($a\b$ означает «деление b на a »)
- ^ Возвведение в степень

Когда отдельная строка кода включает больше, чем один из этих операторов, порядок (приоритет) вычислений следующий:

Приоритет	Операция
1	круглые скобки
2	возвведение в степень, слева направо
3	умножение и деление, слева направо
4	сложение и вычитание, слева направо

Эти правила применяются к скалярным величинам (т.е. к матрицам размером 1×1) обычным способом. (Ниже мы увидим, что нескалярные матрицы (с размерностью большей, чем 1×1) требуют дополнительных правил для применения этих операторов!!!). Примеры записи арифметических операций (команд) в MATLAB®:

Команда	Результат
$3*4$	$ans=12$
$4/5$	$ans = .8000$
$4\sqrt{5}$	$ans=1.2500$
$x = pi/2 ; y = sin(x)$	$y = 1$
$z = 0; w = exp (4*z) /5$	$w = .2000$

Обратите внимание, что многие программисты предпочтут записать выражение для w , данное выше, в формате

$$w = (exp (4*x)) /5,$$

которое дает тот же самый результат, но более понятно, что важно в больших строках с арифметическими операциями.

1.1.2 Матричные вычисления

Поскольку матрицы составлены из множества элементов, а не из одного числа (за исключением скалярных матриц размером 1×1), обычные коммутативный, ассоциативный и дистрибутивный законы выполнения арифметических операций выполняются не всегда.

1.1.2.1 Сложение и вычитание матриц

Операции сложения или вычитания могут быть выполнены только над матрицами ОДИНАКОВОГО ПОРЯДКА. Когда две матрицы одинакового порядка складываются или вычитываются в матричной алгебре, отдельные элементы этих матриц складываются или вычитаются. Таким образом, в данном случае дистрибутивное правило выполняется:

$$A + B = B + A \quad \text{и} \quad A - B = B - A$$

Если $C = A + B$, то каждый элемент C_{ij} определяется по формуле

$$C_{ij} = A_{ij} + B_{ij}$$

Определите A и B следующим образом:

$$A = [1 \ 2 \ 3 ; \ 3 \ 3 \ 3 ; \ 5 \ 3 \ 1]$$

$$B = [2 \ -3 \ 4 ; \ 2 \ -2 \ 2 ; \ 0 \ 4 \ 0]$$

Тогда сумма этих матриц может быть записана как

$$C = A + B \quad \text{и} \quad C = B + A$$

и результатом является матрица

$C =$

$$\begin{matrix} 3 & -1 & 7 \\ 5 & 1 & 5 \\ 5 & 7 & 1 \end{matrix}$$

Теперь определите вектор-строку $x = [3 \ 5 \ 7]$ и вектор-столбец $y = [4; -1; -3]$. В данном случае операция

$$z = x + y$$

невыполнима, потому что эти две матрицы имеют разный порядок (x - матрица размером 1×3 , а y - матрица размером 3×1). Сложение любого количества матриц размером 1×1 или скалярных величин допустимо и при этом выполняются обычные правила арифметики, потому что матрица размером 1×1 - скаляр. Сложение двух векторов допустимо, пока каждый – вектор-строка (матрица $1 \times n$) или вектор-столбец (матрица $n \times 1$). Конечно, любое число векторов может быть сложено или вычтено, при этом результатом является арифметическая сумма отдельных элементов в вектор-строках или вектор-столбцах. Квадратные матрицы могут складываться или вычитаться до тех пор, пока они имеют одинаковый порядок. Квадратная матрица размером 4×4 не может быть сложена с квадратной матрицей размером 3×3 , потому что они имеют разный порядок, хотя обе матрицы квадратны.

1.1.2.2 Умножение матриц

Умножение матриц, согласно определению, является более сложным, чем арифметическое умножение, потому что каждая матрица содержит множество элементов. Обращаясь вновь к векторному умножению, существование множества элементов в векторе привело к двум видам умножения матриц: скалярного произведения и векторного произведения. Матричное умножение также имеет собственный набор специальных правил.

При умножении матриц элементы произведения C двух матриц A и B рассчитываются по формуле

$$C_{ij} = \sum A_{ik} * B_{kj}$$

Для того, чтобы сформировать эту сумму, число столбцов первой (или левой) матрицы (A) должен быть равно числу строк во второй (или правой) матрице (B). Результирующее произведение, матрица C , имеет порядок, в котором число строк равняется числу строк первой (левой) матрицы (A) и число столбцов равно числу столбцов во второй (правой) матрице (B). Ясно, что произведение $A * B$ НЕ ОБЯЗАТЕЛЬНО РАВНО произведению $B * A$! Также ясно, что $A * B$ и $B * A$ существуют только для квадратных матриц!

Рассмотрите произведение двух квадратных матриц 2×2 .

$a = [1 \ 2; 3 \ 4];$

$b = [8 \ 7; 6 \ 5];$

Обращаясь к произведению этих матриц $c = a*b$

$$c_{11} = a_{11}*b_{11} + a_{12}*b_{21}$$

$$c_{12} = a_{11}*b_{12} + a_{12}*b_{22}$$

$$c_{21} = a_{21}*b_{11} + a_{22}*b_{21}$$

$$c_{22} = a_{21}*b_{12} + a_{22}*b_{22}$$

Выполните вычисления вручную и проверьте результат, используя MATLAB®. Затем рассмотрите следующее произведение матриц x размером 3×2 и y – размером 2×4 .

$x = [2 \ 3; 4 \ -1; 0 \ 7];$

$y = [5 \ -6 \ 7 \ 2; 1 \ 2 \ 3 \ 6];$

Прежде всего, обратите внимание на то, что произведение матриц $x*y$ существует, потому что число столбцов в x (2) совпадает с количеством строк в y (2). (Обратите внимание, что произведение $y*x$ НЕ существует!) Если произведение $x*y$ называется C , то эта матрица должен быть размером 3×4 . Снова выполните вычисления, вручную и проверьте результат, используя MATLAB®.

Обратите внимание, что ПРОИЗВЕДЕНИЕ СКАЛЯРНОЙ ВЕЛИЧИНЫ И МАТРИЦЫ – матрица, в которой каждый элемент матрицы был умножен на скаляр. Проверьте это, используя матрицу x , определенную выше, и вычислите произведение $3*x$ в MATLAB®. (Обратите внимание, что это может быть также записано $x*3$, потому что число 3 – скаляр.)

1.1.3 Произведение массивов

Выше уже говорилось, что сложение и вычитание матриц использовало сложение или вычитание отдельных элементов матриц.

Иногда необходимо просто умножить или разделить каждый элемент матрицы соответствующим элементом другой матрицы. В MATLAB® это называется 'операциями над массивом'. Операции над массивами (или поэлементные операции) выполняются в том случае, когда перед оператором ставится знак '!'. Таким образом,

$a.*b$ – умножает каждый элемент a на соответствующий элемент b

$a./b$ – делит каждый элемент a на соответствующий элемент b

$a.\b$ – делит каждый элемент b на соответствующий элемент a

$a.^b$ – возводят каждый элемент a в степень соответствующего элемента b

Например, если матрицы G и H определены как

$G = [1 \ 3 \ 5; 2 \ 4 \ 6];$

$H = [-4 \ 0 \ 3; \ 1 \ 9 \ 8];$

то поэлементное произведение имеет вид

$$G. * H = [-4 \ 0 \ 15 \\ 2 \ 36 \ 48]$$

1.1.4 Скалярное произведение двух векторов

Скалярное (или внутреннее) произведение двух векторов строки, G1 и G2, можно определить следующим образом. Создайте вектор-строки, разделив матрицу G, определенную выше.

$$G1 = G (1,:)$$
$$G2 = G (2,:)$$

Тогда внутреннее произведение вектор-строки G1 размером 1x3 и вектор строки G2 размером 1x3

$$G1 * G2' = 44$$

(здесь символ «'» обозначает операцию транспозиции).

Если эти два вектора - каждый вектор-столбцы, то внутреннее произведение должно быть сформировано матричным произведением транспозиции вектор-столбца на вектор-столбец. Таким образом, создается операция, в которой матрица 1xn умножена на матрицу nx1.

В заключение обратите внимание, что внутреннее произведение должно всегда быть произведением вектор-строки на вектор-столбец.

1.1.5 Внешнее произведение двух векторов

Если существуют два вектор-строки, например, G1 и G2 (как было определено выше), внешнее произведение определяется просто как

$$G1' * G2$$

{Обратите внимание, что G1' имеет размер 3x1 и G2 – размер 1x3}

Результат - квадратная матрица (в отличие от скалярного результата для внутреннего произведения). НЕ ПУТАЙТЕ ВНЕШНЕЕ ПРОИЗВЕДЕНИЕ С ВЕКТОРНЫМ ПРОИЗВЕДЕНИЕМ В МЕХАНИКЕ! Если эти два вектора – вектор-столбцы, то внешнее произведение должно быть сформировано произведением одного вектора на транспозицию второго!

1.2 Логические операции

Говоря о логических операциях, следует сказать:

а) об операциях отношения;

б) об истинно логических операциях.

1.2.1 Операции отношения

К операциям отношения относятся следующие:

- $A > B$ – операция «больше»;
- $A < B$ – операция «меньше»;
- $A \leq B$ – операция «меньше или равно»;
- $A \geq B$ – операция «больше или равно»;
- $A == B$ – операция «равно»
- $A ~=~ B$ – операция «не равно».

Операции отношения выполняют поэлементные сравнения между двумя массивами. Они возвращают массив того же самого размера, в котором элементы формируются по следующему правилу: истина (1) там, где проверяемое отношение между элементами истинно, и ложь (0) там, где отношение не выполняется.

Операции $<$, $>$, \leq и \geq используют только вещественную часть их операндов для сравнения. Операции $==$ и $~=~$ проверяют и действительные, и мнимые части.

Для проверки эквивалентности двух строк используется функция `strcmp`, которая позволяет сравнивать векторы с разной длиной.

Если один из operandов – скаляр, а другой матрица, то скаляр расширяется до размеров матрицы. Например, две пары инструкций:

```
X = 5;  
X >= [1 2 3; 4 5 6; 7 8 10]
```

```
X = 5*ones (3,3);  
X >= [1 2 3; 4 5 6; 7 8 10]
```

производят одинаковый результат:

`Ans =`

```
1 1 1  
1 1 0  
0 0 0
```

1.2.2 Логические операции

К логическим операциям в MATLAB® относятся следующие операции:

- $\&$ - операция «И»;
- $|$ - операция «ИЛИ»;
- \sim - операция «НЕ».

Логические операции выполняются поэлементно над массивами. При выполнении этих операций используется 0, представляющий ложь, и любое ненулевое число, представляющее логическую единицу (истину). Таблица истинности для всех операций дана ниже:

Переменная A	Переменная B	«И» A&B	«ИЛИ» A B	«ИСКЛЮЧАЮЩЕЕ ИЛИ» хор(A,B)	«ОТРИ- ЦАНИЕ» ~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Логические операции имеют самый низкий приоритет по отношению к арифметическим операциям и операциям отношения.

По отношению друг к другу логические операции имеют следующие правила приоритета:

- отрицание NOT имеет самый высокий приоритет;
- AND и OR имеют равный приоритет и вычисляются слева направо.

Следующие два скалярных выражения иллюстрируют отношения между приоритетами для арифметических операций, операций сравнения и логических операций:

$$\begin{aligned} 1 \& 0 + 3 \\ 3 > 4 \& 1 \end{aligned}$$

Их значениями являются 1 и 0 соответственно, и эти выражения эквивалентны следующим:

$$\begin{aligned} 1 \& (0 + 3) \\ (3 > 4) \& 1 \end{aligned}$$

Два скалярных выражения, представленных ниже, показывают отношения приоритета между самими логическими операциями:

$$\begin{aligned} 1 \mid 0 \& 0 = 0 \\ 0 \& 0 \mid 1 = 1 \end{aligned}$$

И операции отношений, и логические операции применяются как при вычислении обычных арифметических выражений, так и в операторе условного перехода IF...THEN.

2. Основные математические функции в MATLAB®

Как уже было сказано выше, основным назначением пакета MATLAB® является решение различных математических задач и выполнение операций над матрицами, векторами и скалярными величинами. По-

этому MATLAB® обладает набором математических функций, с помощью которых пользователь может выполнять все необходимые ему действия.

Рассмотрим более подробно эти функции. Условно их можно разделить на две группы:

а) элементарные – это функции, доступные в любом языке программирования высокого уровня;

б) специализированные – это функции, реализованные только в MATLAB® и предназначенные для вычисления специальных математических функций большой сложности.

2.1 Элементарные математические функции

2.1.1 Тригонометрические

`sin` – синус;

`sinh` – гиперболический синус;

`asin` – арксинус;

`asinh` – гиперболический арксинус;

`cos` – косинус;

`cosh` – гиперболический косинус;

`acos` – арккосинус;

`acosh` – гиперболический арккосинус;

`tan` – тангенс;

`tanh` – гиперболический тангенс;

`atan` – арктангенс;

`atanh` – гиперболический арктангенс;

`sec` – секанс;

`sech` – гиперболический секанс;

`asec` – аркsecанс;

`asech` – гиперболический аркsecанс;

`csc` – косеканс;

`csch` – гиперболический косеканс;

`acsc` – арккосеканс;

`acsch` – гиперболический арккосеканс;

`cot` – котангенс;

`coth` – гиперболический котангенс;

`acot` – арккотангенс;

`acoth` – гиперболический арккотангенс;

2.1.2 Степенные (показательные)

`exp` – экспонента;

`log` – натуральный логарифм (логарифм по основанию e);

`log10` – десятичный логарифм (логарифм по основанию 10);

`log2` – логарифм по основанию 2;

`pow2` – возведение числа «2» в степень (обратная к log2);

`sqrt` – квадратный корень (при отрицательном аргументе результатом является комплексное число);

`nextpow2` – при `nextpow2(n)` эта функция возвращает первое число Р такое, что $2^R \geq$ модуля n. Эта функция часто применяется при выполнении быстрого преобразования Фурье в задачах обработки сигналов;

2.1.3 Функции обработки чисел

`abs` – абсолютное значение (модуль) числа;
`angle` – угол (фаза) комплексного числа;
`conj` – комплексное дополнение;
`imag` – мнимая часть комплексного числа (равна «0» для действительных чисел);
`real` – действительная часть комплексного числа
`isreal` – предикат. Возвращает истину («1») для массива с действительными числами;

2.1.4 Округление и остатки

`fix` – округление в сторону нуля;
`floor` – округление в сторону $-\infty$;
`ceil` – округление в сторону $+\infty$;
`round` – округление в сторону ближайшего целого;
`mod(x, y)` – модуль - остаток после деления (число со знаком);
`rem(x, y)` – остаток после деления. Значения функций MOD и REM одинаковы, если x и y имеют одинаковые знаки, и различны, если x и y имеют различные знаки;
`sign` – функция определения знака числа. Имеет стандартное определение:

$$f(x) = \begin{cases} 1, & \text{если } x > 0 \\ 0, & \text{если } x = 0 \\ -1, & \text{если } x < 0 \end{cases}$$

2.2 Специализированные математические функции

2.2.1 Функции классической математики

`besselj` – функция Бесселя первого типа;
`bessely` – функция Бесселя второго типа;
`besselh` – функция Бесселя третьего типа (функция Ханкеля);
`besseli` – модифицированная функция Бесселя первого типа;
`besselk` – модифицированная функция Бесселя второго типа;
`beta` – бета – функция;
`betainc` – незавершенная бета – функция;
`betainc` – логарифм бета – функции;
`ellipj` – эллиптическая функция Якоби;

`ellipke` – завершенный эллиптический интеграл;
`erf` – функция ошибки;
`erfc` – дополнительная функция ошибки;
`erfcx` – масштабированная дополнительная функция ошибки;
`erfinv` – инверсия функции ошибки;
`gamma` – гамма – функция;
`gammainc` – незавершенная гамма – функция;
`gammaln` – логарифм гамма – функции;
`legendre` – связанная функция Лежандра;

2.2.2 Функции теории чисел

`factor(n)` – функция возвращает вектор, содержащий простые множители указанного в качестве параметра числа n;

`isprime` – логический предикат, возвращает значение истины для простых чисел;

`primes(n)` – функция возвращает список простых чисел, не превышающих указанное число n;

`gcd` – наибольший общий делитель (НОД);

`lcm` – наименьшее общее кратное (НОК);

`perms(1 : N)` (или `perms(U)`, если U – вектор длиной N) – создает матрицу, в которой N! строк и N столбцов. Эта матрица содержит все возможные перестановки из N элементов. Эта функция может быть практически применима только при небольших значениях N ($N < 15$);

Следует отметить, что все вышеуказанные функции применимы и к скалярным величинам, и к векторам. В случае векторов функция применяется к каждому из элементов массива.

Контрольные вопросы

1. Как можно условно разделить арифметические операции?
2. Какие существуют матричные арифметические операции? Как выполняется умножение матриц?
3. Какие логические операции существуют в MATLAB®?
4. Приведите таблицы истинности для функций AND и XOR
5. Как можно поделить (условно) математические функции, доступные в MATLAB®?
6. Опишите любые 5 функций из различных разделов.

Лекция 3: «Типы файлов в MATLAB®. Инструкции в MATLAB®»

План лекции:

1. Типы файлов в MATLAB®;
2. Управляющие инструкции в MATLAB®.

1. Типы файлов в MATLAB®

Иногда необходимо записать несколько строк кода MATLAB® (или команд MATLAB®) перед их выполнением. Одним из путей решения этой задачи является использование так называемых M – файлов. При использовании M - файлов можно записать несколько строк кода MATLAB® и сохранить это в файле с заданным именем и автоматически добавленным расширением *.m. Впоследствии командная строка или содержимое файла могут быть выполнены путем вызова файла в MATLAB®. Это приложение называется **РАБОЧИЙ ФАЙЛ** (или **ФАЙЛ СЦЕНАРИЯ (СКРИПТ)**).

В некоторых случаях удобнее представить функцию в MATLAB®, чем вычислять или определять специальные матрицы. Многие сохраненные функции MATLAB® типа $\sin(x)$ и $\log(x)$ могут служить примерами функций. Можно писать собственные (определенные пользователем) функции, которые решают специфические задачи. Это применение *.m – файлов называется **ФАЙЛАМИ ФУНКЦИЙ**.

Рассмотрим эти два вида M – файлов, их создание и применение более подробно.

1.1 Рабочие файлы (файлы – сценарии)

Файлы - сценарии – это просто файлы, содержащие последовательность инструкций MATLAB®. Рабочий файл (или файл – сценарий) должен быть подготовлен в текстовом редакторе в **ТЕКСТОВОМ ФОРМАТЕ** и сохранен в каталоге, из которого загружен MATLAB®. Имя может быть любым, допустимым для имени файла, с расширением .m.

Предположим, что нам нужно создать график функции

$$y = e^{-x/10} \sin(x), x \in [0 ; 10].$$

Чтобы сделать это, используем рабочий .m – файл. Вызовем файл `expot.m`, откроем в текстовом редакторе и введем код, данный ниже. (Обратите внимание на использование знака '%' в первой строке для создания комментария. Любой текст или команда, напечатанные на строке после знака '%', будут расценены как комментарий и проигнорированы в исполнимом коде.)

```
% A scratch m-file to plot exp(-x/10)sin(x)
x = [ 0.:2:10 ];
```

```

y = exp(-x/10) .* sin(x);
plot(x,y),...
title('EXPONENTIAL DAMPED SINE FUNCTION'),...
xlabel('x'),...
ylabel('y'),...
text(.6,.7,'y = exp(-x/10)*sin(x)','sc')

```

Далее нужно сохранить этот файл под именем `explot.m` в рабочем каталоге MATLAB®. Теперь, находясь в MATLAB®, можно в любой момент напечатать команду `explot`, и появится график «заглушенной показательной синусоидальной функции».

1.2 Файлы функций

Иногда у пользователя возникает необходимость в создании собственных функций, которые выполняли бы строго определенные, необходимые только ему действия. С точки зрения самого интерпретатора пользователь просто прибавляет новые функции к словарю MATLAB®, выражая их в терминах уже существующих функций. Команды и функции, которые составляют новую функцию, постоянно находятся в текстовом M - файле.

Функции используют собственные локальные переменные и допускают ввод параметров.

Имя M – файла начинается с алфавитного символа, и имеет расширение имени *.m. Имя M - файла, без расширения, - это то, что MATLAB® ищет, когда пользователь пробует использовать сценарий или функцию.

Верхняя строка функционального M - файла содержит синтаксическое определение. Имя функции, определенное в первой строке M - файла, должно быть таким же, как и имя файла без расширения *.m. Например, файл на диске с именем `stat.m` с кодом

```

function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x)/n;
stdev = sqrt(sum((x-mean).^2/n));

```

определяет новую функцию, называемую `stat`, которая вычисляет среднее и стандартное отклонение для вектора. Все переменные в пределах тела функции - локальные переменные.

Подфункция, видимая только для других функций в том же самом файле, создается определением новой функции с ключевым словом `function` после тела предшествующей функции или подфункции. Например, `avg` - подфункция в файле `stat.m`:

```

function [mean,stdev] = stat(x)
    n = length(x);
    mean = avg(x,n);
    stdev = sqrt(sum((x-avg(x,n)).^2)/n);

function mean = avg(x,n)
    mean = sum(x)/n;

```

Подфункции невидимы снаружи того файла, в котором они определены. Функции обычно возвращают управление при достижении конца функции. Для преждевременного возвращения следует использовать команду **return**.

В том случае, если MATLAB® не «узнает» функцию по имени, он ищет файл с таким же именем на диске. Если функция найдена, MATLAB® компилирует ее в память для последующего использования.

При вызове функционального M – файла из командной строки или из другого M - файла, MATLAB® анализирует (компилирует) функцию и сохраняет ее в памяти. Откомпилированная функция остается в памяти до тех пор, пока память не очищена командой **clear**, или не осуществлен выход из MATLAB®. Команда **rscode** выполняет синтаксический анализ и сохраняет результат на диске в виде P - файла, который может быть загружен позже.

Рассмотрим пример создания новой функции. Предположим, что нужно иметь в рабочем каталоге MATLAB® функцию, которая вычисляет синус угла, когда параметр задан в градусах. Функция **sin(x)** в MATLAB® требует, чтобы x был задан в радианах. Функция может быть написана таким образом, чтобы, используя функцию **sin(x)** MATLAB® при x в радианах, вычислить синус параметра, выраженного в градусах. Снова, как и в случае рабочего файла, мы должны подготовить файл в текстовом формате, используя текстовый редактор, и добавить расширение **.m** к файлу. Как уже говорилось выше, ИМЯ ФАЙЛА ДОЛЖНО БЫТЬ ОДИНАКОВЫМ С ИМЕНЕМ ФУНКЦИИ.

Представленный ниже код решает нашу задачу. Обратите внимание, что первая строка должна всегда начинать со слова **function**, за которым следует имя функции, выраженное как " **y = имя_функции** ". В данном конкретном примере имя функции - **sind (x)**. Это имя, по которому мы будем вызывать функцию, как только она будет сохранена.

```

function y = sind(x)
% Эта функция вычисляет синус для аргумента,
% заданного в градусах.
% Обратите внимание, что умножение и деление массивов позволяет
% этой функции оперировать над скалярными величинами,
% векторами и матрицами.
y = sin( x .* pi ./ 180 )

```

Обратите снова внимание на правило в записи функций, которое заключается в следующем: в первой строке файла должно быть слово `function`, за которым следует

`y = <вызов функции>.`

(В данном случае вызов функции – это '`sind`' с аргументом '`x`' в круглых скобках). Таким образом,

```
function y=sind(x)
```

Теперь каждый раз, когда мы напечатаем `sind(x)` в MATLAB®, мы получим значение функции синус, которая рассчитана с предположением, что `x` задано в градусах. (Конечно, в параметре могут быть написаны любая матрица или выражение, включающее матрицу).

2. Управляющие инструкции в MATLAB®

В MATLAB есть возможность создания пользователем программ. Эти программы затем могут быть использованы в качестве отдельных функций.

Для контроля и управления выполнением вычислений при программировании в MATLAB используются специальные инструкции, так называемые управляющие конструкции: `if`, `while`, `for` и `switch - case`.

Перед подробным рассмотрением этих инструкций по отдельности следует сказать, что управляющие конструкции могут быть вложенными, т.е. внутри одной конструкции может быть другая (такая же или другая).

Каждая управляющая конструкция должна закрываться соответствующим оператором `end`.

2.1 Оператор условного перехода (ветвления) IF

Оператор условного перехода `IF` используется в тех случаях, когда дальнейшее исполнение программы зависит от выполнения (или невыполнения) какого-то условия (или условий). По-другому конструкцию `IF` называют конструкцией принятия решений.

Формат оператора `IF` для самого общего случая:

```
IF <условие1>
  {операторы1}
ELSEIF <условие2>
  {операторы2}
ELSE
  {операторы3}
ENDIF
```

{`Операторы1`} выполняются в том случае, если действительная часть в `<условие1>` не равна нулю (т.е. если условие истинно). В противном случае (если `<условие1>` ложно) проверяется истинность для `<условие2>`. Если

оно истинно, то выполняются {операторы2}. Если оно тоже ложно, то выполняются {операторы3}.

Разделы ELSE и ELSEIF являются необязательными. Разделы ELSEIF могут использоваться как вложенные операторы IF.

При составлении условий, записанных в разделах IF и ELSEIF, могут использоваться алгебраические выражения, логические операции и операции отношения: ==, <, >, <=, >=, или ~=.

Пример:

```
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```

При разработке приложений можно использовать сокращенные формы условного оператора:

a) IF <условие>
 {операторы}
 ENDIF

b) IF <условие>
 {операторы1}
 ELSE
 {операторы2}
 ENDIF

2.2 Оператор выбора SWITCH

Другим средством для выбора пути, по которому пойдет исполнение программы (или принятия решения) является оператор SWITCH, который так и называется оператором выбора.

Формат оператора SWITCH:

```
SWITCH <проверяемое_выражение>
CASE <значение>
    оператор, . . . , оператор;
CASE {значение1, значение2, значение3,...}
    оператор, . . . , оператор;
OTHERWISE,
    оператор, . . . , оператор;
END
```

Выполнение оператора CASE в MATLAB аналогично выполнению такого же оператора в языке Си.

Параметр <проверяемое_выражение> может быть скалярным выражением или строкой. При использовании скалярных величин равенство признается истинным, если <проверяемое_выражение> == <значение>. Равенство строк признается истинным, если функция strcmp(<проверяемое_выражение>, <значение>) возвращает 1 (истину).

Пример использования оператора (предположим, что переменная METHOD существует и является строковой переменной):

```
switch lower(METHOD)
case {'linear','bilinear'}, disp('Метод линейный')
    case 'cubic', disp('Метод кубический')
    case 'nearest', disp('Метод приближенный')
otherwise, disp('Неизвестный метод.')
end
```

Помимо операторов выбора и условного перехода, в MATLAB также существуют операторы циклов.

2.3 Оператор WHILE

Оператор цикла WHILE используется для повторения определенного набора команд и операторов неопределенное количество раз.

Формат оператора WHILE:

```
WHILE <выражение>
    {операторы}
END
```

{Операторы} выполняются до тех пор, пока действительная часть <выражения> имеет все ненулевые элементы.

При составлении <выражения>, аналогично оператору ветвления IF, используются операторы отношения ==, <, >, <=, >=, или ~=.

Пример использования оператора цикла WHILE (допустим, что A уже определено):

```
E = 0*A;
F = E + eye(size(E));
N = 1;
while norm(E+F-E,1) > 0,
    E = E + F;
    F = A*F/N;
    N = N + 1;
end
```

2.4 Оператор FOR

Другим оператором цикла, доступным в MATLAB, является оператор FOR. С помощью этого оператора можно повторять указанные операторы заданное количество раз.

Формат этого оператора:

```
FOR <сч. ц.> = <нач. знач.> : [<шаг>:] <конечн. знач. >
    оператор
    ...
    оператор
END
```

Операторы, составляющие тело цикла, выполняются столько раз, сколько нужно, чтобы переменная <сч. ц.> прошла все значения от начального (<нач. знач.>) до конечного (<конечн. знач. >) с заданным шагом <шаг>. Если значение шага не задано, то оно по умолчанию принимается равным 1.

Пример на использование оператора FOR (предположим, что N уже было присвоено значение):

```
FOR I = 1:N,
    FOR J = 1:N,
        A(I,J) = 1/(I+J-1);
    END
END
```

2.5 Команда управления BREAK

С помощью команды управления BREAK можно прервать исполнение циклов WHILE или FOR.

Для вложенных циклов команда BREAK выполняет выход из самого внутреннего цикла.

Определения и термины.

Графический файл (или файл сценария (скрипт)) – это текстовый файл, содержащий набор команд (инструкций) и операторов MATLAB®, который после создания может быть использован для выполнения каких – либо заданных действий.

Файл функции – это текстовый файл, содержащий последовательность команд и операторов MATLAB®, которые записаны в определенном формате и представляют собой реализацию какой – либо пользовательской функции.

Новые команды

if . . . endif, switch . . . case, while . . . end, for . . . end.

Контрольные вопросы

1. Что такое «файл сценария» («скрипт»)? Что такое «файл функции»?
2. Какое расширение присваивается файлам в MATLAB?
3. Как создать файл сценария?
4. Дайте формат описания пользовательской функции (внутри функционального файла).
5. Для чего используется команда RETURN?
6. Напишите программу, реализующую функцию $f(x, y) = x^2 + xy^3 + 4.57xy + y^5$
7. Дайте форматы оператора условного перехода IF . . . ENDIF (полный и сокращенные).
8. Дайте формат оператора цикла FOR. Объясните значение входящих в него параметров.
9. Дайте формат операторов WHILE и SWITCH.
10. Составьте программы на использование операторов IF, FOR и WHILE

Лекция 4: «Графика в MATLAB®»

План лекции:

1. Обзор графических возможностей MATLAB;
2. Простейшие команды двумерной графики;
3. Графики в декартовой системе координат;
4. Графики в полярных координатах;
5. Гистограммы;
6. Множественные графики;
7. Построение графиков пар данных;
8. Использование цветов при рисовании;
9. Распечатка графиков;
- 10.Функции преобразования координат;
- 11.Трехмерная графика;

1. Обзор графических возможностей MATLAB®

Одной из наиболее мощных особенностей MATLAB является способность создания графики. В данном курсе представлены элементарные, наиболее общие, команды для создания графиков двух векторов. Более сложные и мощные команды графики могут быть найдены в документации MATLAB.

MATLAB имеет возможность построения графиков в разных системах координат: прямоугольной (декартовой), сферической, цилиндрической. Имеется также возможность преобразования координат из одного вида в другой.

Также графики можно строить в двух- или трехмерной системах координат.

2. Простейшие команды двумерной графики

Перед рассмотрением особенностей построения графиков в той или иной системе координат будут представлены некоторые общие графические команды, которые применимы в любой системе.

plot(x, y) - создает график векторов x и y в декартовой плоскости;

plot(y) - создает график y против номеров элементов в y-векторе.

semilogx(x, y) – чертит график логарифма x против y

semilogy(x, y) – чертит график x против логарифма y

loglog (x, y) – чертит график логарифма x против логарифма y

grid - создает сетку на графике

title('текст') - размещает заголовок сверху графика

xlabel('текст') - записывает 'текст' под осью X графика

ylabel('текст') - записывает 'текст' слева от оси Y графика

text(x, y, 'текст') - записывает 'текст' в точке (x, y)

text(x, y, 'текст', 'sc') - записывает 'текст' в точке (x, y), предполагая, что нижний левый угол имеет координаты (0,0) и верхний правый - (1,1).

`polar(theta, r)` - создает полярный график векторов r и θ , где θ дана в радианах.

`bar(x)` - создает гистограмму вектора x . (Примечание: также можно использовать команду `stairs(x)`.)

`bar(x, y)` - создает гистограмму элементов вектора y , располагая области согласно элементам вектора x . (Примечание: также можно использовать команду `stairs(x, y)`.)

3. Декартовы (или $X - Y$) графики

Декартов (или ортогональный) (x, y) график основан на рисовании графика пар данных, принадлежащих указанным векторам (x, y). Ясно, что векторы x и y должны иметь одинаковое число элементов. Представьте, что нужно построить график показательной функции для значений x от 0 до 2.

```
x = 0 : .1 : 2;  
y = exp (x);  
plot(x, y)
```

ПРИМЕЧАНИЕ: выбранные функции типа `exp()` и `sin()` будут использоваться в этих обучающих программах просто для облегчения понимания, поскольку с их помощью можно легко создавать матрицы с однозначным соответствием между элементами.

Конечно, символы x, y выбраны произвольно. Если нужно составить график температуры (`temperature`) на ординате и времени (`time`) на абсциссе, и векторы для температуры и времени были загружены в MATLAB, то команда будет

```
plot(time , temperature)
```

Обратите внимание, что команда `plot(x, y)` открывает графическое окно. Если теперь выполнить команду `grid`, графическое окно будет перерисовано. (Обратите внимание, что нужно переместить курсор в командное окно перед вводом новой команды.). Чтобы избежать перерисовки окна, можно использовать знак продолжения строки. Предположим

```
plot(x, y), ...  
grid, ...  
title('Показательная функция'), ...  
xlabel ('x'), ...  
ylabel (' exp (x)'),  
text(.6, .4, ' y = exp (x) ', 'sc')
```

Обратите внимание, что если при печатании ряда строки программы сделана ошибка (как было упомянуто выше), использование знака «продолжение строки» может быть ошибочным при исполнения программы. Чаще всего графические команды включаются в рассмотренные ранее M – файлы (файлы – сценарии или файлы – функции). Этот подход

дает лучшую возможность для редактирования ошибок при помощи возврата к уже созданным файлам и программам.

Определив векторы x и y , создайте графики при помощи команд `semilog` и `loglog`, используя их или другие векторы, которые можно при желании создать. Обратите внимание на особенность применения логарифмического масштаба для отображения вектора, который имеет отрицательные или нулевые элементы. Здесь вектор x имеет нулевой элемент ($x(1) = 0$). Так как логарифм нуля или любого отрицательного числа неопределен, пара данных (x,y) , в которой обнаружен нулевой или отрицательный элемент, будет отвергнута и выдано предупреждающее сообщение.

Обратите особое внимание на различие в командах `plot(x)` (и/или `plot(y)`) и `plot(x,y)`.

Обратите внимание, что MATLAB рисует прямую линию между точками. Если необходимо построить относительно гладкую, главную кривую, построенную для быстроизменяющейся функции, то нужно включить много точек (пар данных). Например, рассмотрим тригонометрическую функцию $\sin(x_1)$ (x_1 выбран здесь как параметр, чтобы отличить его от вектора x , который был определен ранее.)

Начертим график $\sin(x_1)$ в интервале $0 \leq x_1 \leq \pi$. Сначала определим x_1 с 5 элементами,

```
x1 = 0 : pi/4 : pi
y1 = sin(x1)
plot(x1, y1)
```

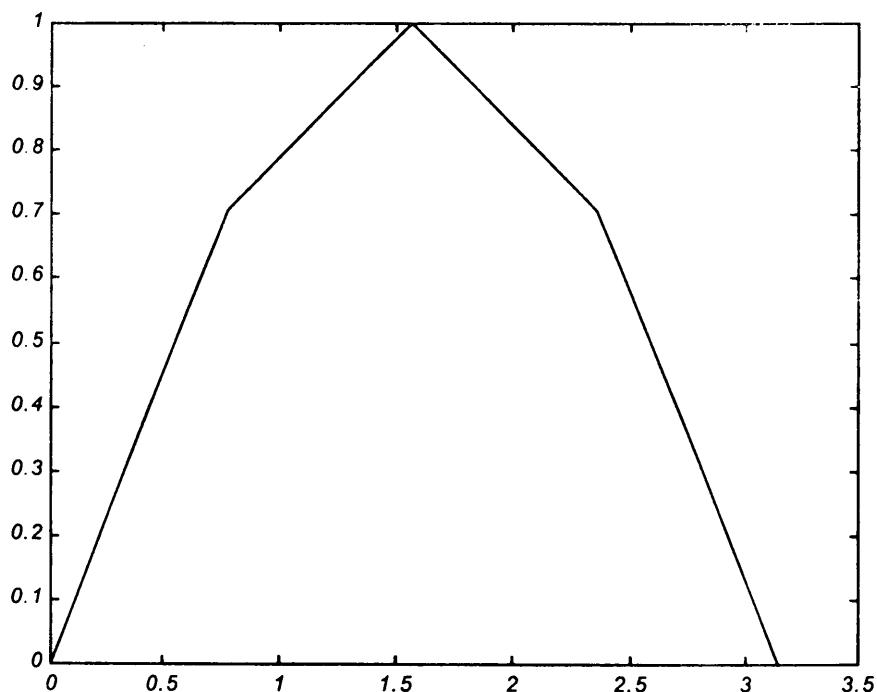


Рис.1 График функции $y=\sin(x)$ при $x \in [0;\pi]$ с шагом $\pi/4$

Теперь, повторим это после определения нового вектора для x_1 и y_1 с 21 элементами. (Обратите внимание на использование здесь точки с запятой для предотвращения печати и прокрутки на экране вектора или матрицы с большим количеством элементов!)

```
x1 = 0 : .05*pi : pi ;
y1 = sin(x1);
plot(x1, y1)
```

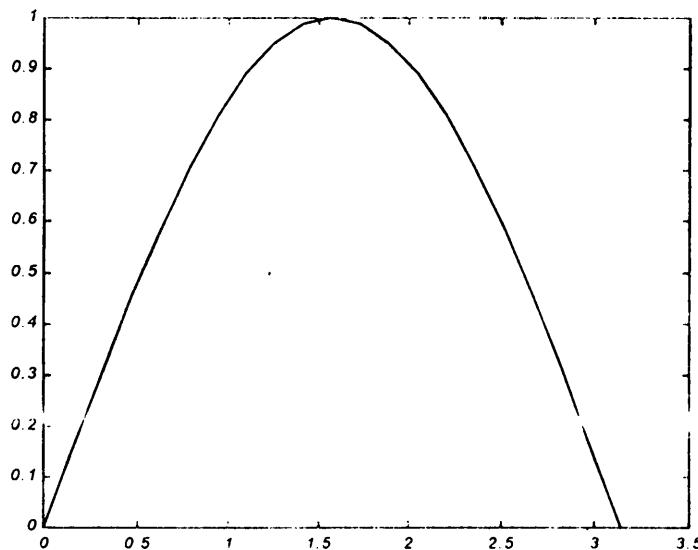


Рис.2 График функции $y=\sin(x)$ при $x \in [0;\pi]$ с шагом $\pi/20$

Заметно, что при использовании большего числа точек кривая получается более гладкой. (Однако следует помнить, что увеличение количества точек (отсчетов) при построении графика функции приводит к увеличению времени исполнения.)

3. Графики в полярных координатах

Графики в полярных координатах создаются аналогичным с графиками в декартовых координатах X-Y способом; однако в данном случае параметрами являются угол theta в радианах, отложенный от горизонтальной (положительной) оси x, и длина радиус - вектора, направленного от начала координат под этим углом.

Графики в полярных координатах не позволяют маркирование осей; однако, обратите внимание, что масштаб для радиус - вектора представлен по вертикальной оси и когда используется команда grid, сетка узлов представлена в виде сегментов по 15° . Команды title и text действительны для полярных координат.

```
angle = 0 : .1*pi : 3*pi;
radius = exp(angle/20);
polar(angle , radius), ...
title(' Пример графика в полярных координатах '), ...
grid
```

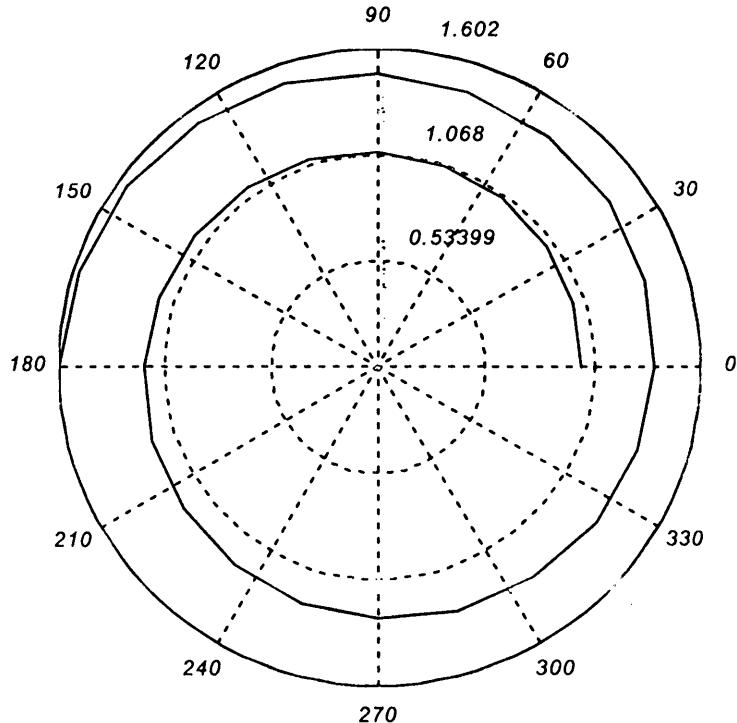


Рис 3 Пример графика в полярных координатах

Обратите внимание, что углы могут превышать один оборот в 2π .

4. Гистограммы

Для того, чтобы пронаблюдать, как MATLAB создает гистограммы, вернемся к векторам x и y , которые были определены ранее. Создайте прямоугольные и ступенчатые гистограммы, использующие эти векторы. Команды `title` и `text` также могут использоваться с гистограммами любого типа.

`bar(x,y)` и `bar(y)`
`stair(x,y)` и `stair(y)`

5. Множественные графики

На одном графике можно представить более чем рисунок. Один способ сделать это – сохранить графическое окно открытым с помощью команды `hold` и выполнить следующую графическую («рисующую») команду.

```
x1 = 0 : .05*pi : pi;
y1 = sin(x1);
plot(x1, y1)
hold
y2=cos(x1);
plot(x1, y2)
```

Команда `hold` останется активной до тех пор, пока не будет «выключена» командой `hold off`.

Вы можете создавать множественные диаграммы (графики), используя несколько аргументов. В дополнение к векторам `x` и `y`, созданным ранее, создадим векторы `a`, `b` и построим график обоих векторных наборов (одновременно) следующим образом:

```
a = 1 : .1 : 3;
b = 10*exp(-a);
plot(x, y, a, b)
```

Множественные графики могут быть также выполнены, если использовать в параметрах матрицы вместо простых векторов. Если параметры команды `plot` - матрицы, то СТОЛБЦЫ матрицы `y` рисуются по оси ординат против СТОЛБЦОВ матрицы `x` по оси абсцисс. Обратите внимание, что матрицы `x` и `y` должны иметь одинаковый порядок! Если `y` - матрица, и `x` - вектор, строки или столбцы `y` составляют график против элементов `x`. В данном случае число столбцов ИЛИ строк в матрице должно соответствовать числу элементов в `x`. Матрица `x` может быть вектор-строкой или вектор-столбцом!

Обратитесь повторно к вектор-строкам `x` и `y`, определенным ранее. Увеличьте вектор-строку `y` и создайте матрицу `yy` с 2 строками.

```
yy = [ y ; exp(1.2*x) ];
plot(x, yy)
```

6. Рисование пар (точек) данных. Другие полезные возможности

MATLAB соединяет пары данных, описанные векторами в команде `print`, прямой линией. Можно представить только точки – изображения данных и опустить любые соединительные линии между этими точками. Точки могут быть описаны различными символами (`.` , `+` , `*` , `o` и `x`). Следующая команда «рисует» график данных (`x`, `y`) как "кривую" из соединенных отрезков и, кроме того, размещает символ '`o`' в каждой из пары данных (`x1`, `y1`).

```
plot(x, y, x1, y1, 'o')
```

Строки могут быть окрашены и прерывистой для того, чтобы различать несколько строк. Разноцветные строки эффективны на цветных мониторах, принтерах или графопостроителях. Однако цвета бесполезны на

обычных принтерах или монохромных мониторах. Цвета обозначаются в MATLAB следующими символами:

r - красный,
g - зеленый,
b - синий,
w - белый,
i - невидимый.

Следующая команда рисует график данных (x , y) красной сплошной линией и данных (r , s) - зеленой прерывистой линией:

```
plot(x, y, 'r', r, s, '-- g')
```

7. Печать графиков

Команда `print` отправляет содержимое текущего графического окна на локальный принтер.

Созданный граяфик можно сохранить в графическом файле. Для того, чтобы сделать это, нужно просто добавить в конец команды `print` имя файла. Команда

```
print <имя_файла>
```

сохранит содержимое графического окна в файле, названном `имя_файла.ps`, в формате, называемом Postscript. Расширение `*.ps` можно опустить, поскольку MATLAB автоматически добавит его. При просмотре файлов в вашем рабочем каталоге удобно идентифицировать любые графические файлы, просто ища файлы с расширением `.ps`. Для того, чтобы напечатать один из этих файлов, можно удобно «перетащить» и поместить файл из окна Диспетчера файлов в значок принтера.

8. Преобразование координат

Как уже было сказано выше, в MATLAB пользователь имеет возможность преобразования координат. Для этого существуют специальные функции:

`cart2sph` – преобразование декартовых координат в сферические;
`cart2pol` – преобразование декартовых координат в полярные;
`pol2cart` – преобразование полярных координат в декартовы;
`sph2cart` – преобразование сферических координат в декартовы;

9. Трехмерная графика

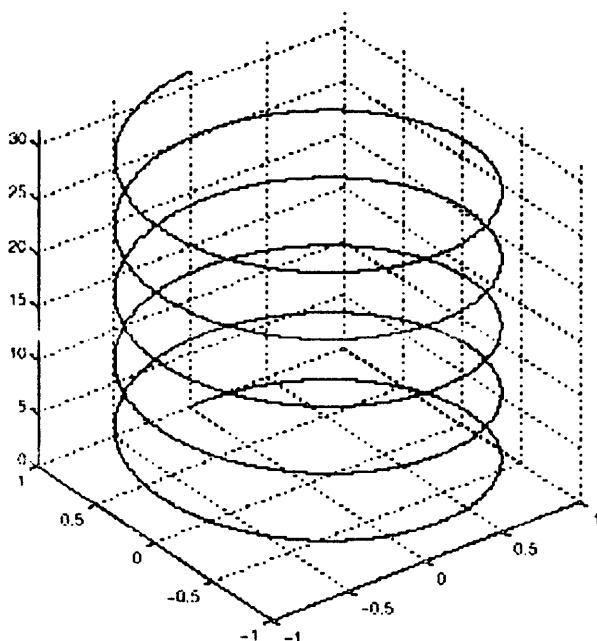
Говоря о графических возможностях MATLAB, следует отдельно сказать о трехмерной (3-D) графике, поскольку она более сложная по сравнению с двумерной, но чаще применима в реальных задачах.

Рассмотрим команды, которые являются специфическими именно для трехмерной графики:

`plot3` – аналогично команде `plot` в двумерной графике. Рисует линии и точки в трехмерном пространстве. Координаты задаются векторами x , y и z , которые должны иметь одинаковую длину. Например, программа

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t) .
axis square; grid on
```

рисует фигуру геликс:



`mesh` – эта команда рисует трехмерную «сеть»;

`surf` – рисует трехмерную окрашенную поверхность;

`fill3` – рисует трехмерные заполненные многоугольники;

Пример использования команд трехмерной графики: представим, что нам нужно построить график функции двух переменных $z = f(x, y)$. Первым шагом является генерация матриц X и Y , состоящих из повторенных строк и столбцов, соответственно, в области функции. Затем можно использовать эти матрицы для вычисления и рисования графика функции.

Функция `meshgrid` преобразовывает область, указанную двумя векторами x и y , в матрицы X и Y . Затем эти матрицы можно использовать для вычисления функции двух переменных. Строки матрицы X – копии вектора x , а столбцы матрицы Y – копии вектора y .

Чтобы проиллюстрировать использование команды `meshgrid`, рассмотрим функцию $\sin(r)/r$ или sinc . Для того, чтобы вычислить значения этой функции на интервале $[-8 ; 8]$ и для x , и для y , необходимо передать толь-

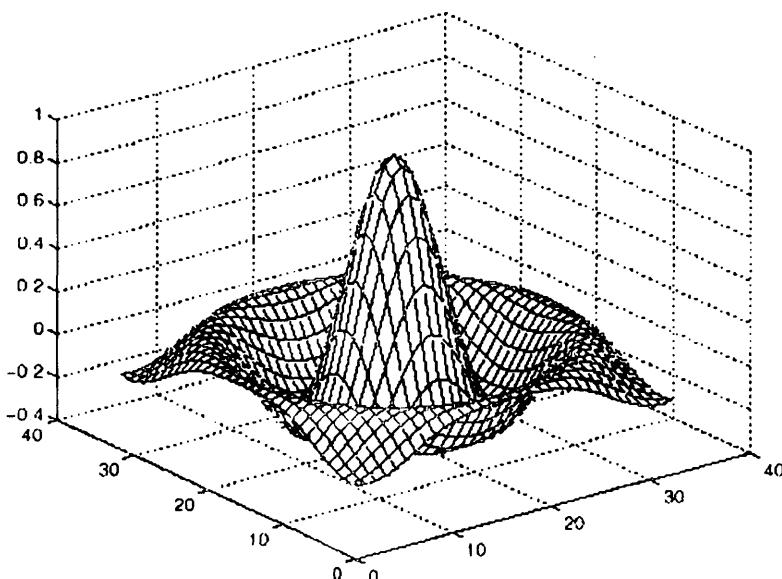
ко один вектор - аргумент в функцию `meshgrid`, который затем используется в обоих направлениях:

```
[X, Y] = meshgrid (-δ : .5 : δ);  
R = sqrt(X.^2 + Y.^2) + eps;
```

Матрица R содержит расстояние от центра матрицы, которая является началом координат. Добавление значения `eps` предотвращает деление на ноль (в следующем шаге), которое производит значение `NaN` в данных.

Формирование функции `sinc` и составления графика Z с сетью приводит к трехмерной поверхности:

```
Z = sin(R) ./ R;  
mesh(Z)
```



Контрольные вопросы

1. Для чего служат команды `plot`, `semilog`, `loglog`, `text`, `polar`?
2. В чем различие между командами `plot(x)` и `plot(x,y)`?
3. В чем особенность построения графиков в логарифмическом масштабе?
4. Как строятся графики в полярных координатах?
5. С помощью какой команды можно в одном графическом окне построить несколько графиков?
6. Какие символы можно использовать для обозначения (или выделения) точек на графиках в MATLAB? Приведите примеры команды `plot`, в которой используются различные типы для точек данных и соединительных линий.
7. Какой командой можно распечатать график из текущего графического окна на принтере? Как можно сохранить рисунок в файле?
8. Приведите команды преобразования координат из одной системы в другую.
9. Приведите команды, характерные для трехмерной графики в MATLAB.

Лекция 5: «Решение систем линейных уравнений»

План лекции:

1. Постановка задачи решения систем линейных уравнений (СЛУ);
2. Обзор методов решения СЛУ;
3. Функции MATLAB® для решения СЛУ;
4. Решение практических задач.

1. Постановка задачи решения систем линейных уравнений (СЛУ)

Задача решения систем линейных уравнений (СЛУ) очень часто возникает при решении многих практических проблем и задач. Рассмотрим постановку задачи решения СЛУ.

Пусть дана СЛУ вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \quad (1)$$

Здесь x_1, x_2, \dots, x_n - искомые переменные, $a_{11}, a_{12}, \dots, a_{nn}$ и b_1, b_2, \dots, b_n - действительные числа.

Решением данной СЛУ является такое множество значений переменных x_1, x_2, \dots, x_n , при которых система (1) обращается в тождество.

Система (1) может быть записана также и в матричном виде:

$$A \cdot x = B \quad (2)$$

Здесь:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad \text{- матрица размером (n x n), состоящая из коэффициентов при неизвестных;}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \text{- вектор - столбец размером (n x 1), состоящий из неизвестных;}$$

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \quad \text{- вектор - столбец размером (n x 1), состоящий из свободных членов системы (1).}$$

В дальнейшем, при рассмотрении всех методов решения, будут использоваться оба способа записи СЛУ.

2. Обзор методов решения СЛУ

В данном разделе будет дан обзор только численных методов решения СЛУ.

Наиболее известными являются методы Гаусса (или метод исключения переменных) и метод Крамера. Эти методы подробно описаны в [1] и [2].

Другим методом, который применяется реже, однако дает лучшие результаты, является так называемый метод наименьших квадратов (МНК).

МНК заключается в том, что подбираются значения переменных (в данном случае – значения x_1, x_2, \dots, x_n) таким образом, что квадрат разности между точным (заданным) и приближенным (вычисленным) должен быть минимальным. Более подробно МНК будет рассмотрен в лекции, посвященной методам статистической обработки данных.

3. Функции MATLAB® для решения СЛУ

Для решения СЛУ MATLAB® предлагает два способа. Первый заключается в применении так называемой операции «левого деления» («\»). При использовании этой операции решение СЛУ может быть найдено так:

$$x = A \setminus B$$

Другим способом является использование встроенной функции nnls (Nonnegative least squares), которая использует метод наименьших квадратов (МНК) для решения заданной системы, и результат этой функции – решение системы – является неотрицательным.

Функция nnls имеет несколько форматов:

$x = \text{nnls}(A, b)$
 $x = \text{nnls}(A, b, tol)$
 $[x, w] = \text{nnls}(A, b)$
 $[x, w] = \text{nnls}(A, b, tol)$

Рассмотрим каждый из этих форматов подробнее:

$x = \text{nnls}(A, b)$ – решает систему уравнений вида $A \cdot x = B$ методом наименьших квадратов с учетом ограничения, что решение – вектор x – не должно иметь отрицательных элементов.

$x = \text{nnls}(A, b, tol)$ – решает систему уравнений с заданной точностью tol.

$[x, w] = \text{nnls}(A, b)$ – помимо решения, возвращает также вектор w, в котором $w_i \leq 0$, если $x_i = 0$, и $w_i \geq 0$, если $x_i > 0$.

$[x, w] = \text{nnls}(A, b, tol)$ – решает систему уравнений, возвращает вектор w и определяет точность tol.

Алгоритм, использованный в данной функции, описан в главе 23 в [1]: Lawson, C. L. and R. J. Hanson, Solving Least Squares Problems, Prentice-Hall, 1974. Алгоритм начинается с набора возможных базисных векторов, вычисляет соответствующий вектор w, и выбирает базисный вектор, соответствующий максимальному значению в w для того, чтобы удалить его из базиса в обмен на другого возможного кандидата. Этот процесс повторяется до тех пор, пока значение не начнет выполнять условие $0 \geq w$.

4. Решение практических задач

Рассмотрим несколько примеров, которые покажут, как необходимо использовать оба способа решения СЛУ.

Пример 1: найти решение системы уравнений, используя операцию левого деления:

$$\begin{cases} r + s + t + w = 4 \\ 2r - s + w = 2 \\ 3r + s - t - w = 2 \\ r - 2s - 3t + w = -3 \end{cases}$$

Решение: запишем заданную систему в матричном виде (в форме (2)):

$$\left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 2 & -1 & 0 & 1 \\ 3 & 1 & -1 & -1 \\ 1 & -2 & -3 & 1 \end{array} \right] \cdot \left[\begin{array}{c} r \\ s \\ t \\ w \end{array} \right] = \left[\begin{array}{c} 4 \\ 2 \\ 2 \\ -3 \end{array} \right]$$

Теперь в командной строке MATLAB® зададим исходные матрицы:

```
>> A=[1 1 1 1 ; 2 -1 0 1 ; 3 1 -1 -1 ; 1 -2 -3 1]
>> B=[4 ; 2 ; 2 ; -3]
```

Теперь определим решение заданной системы с помощью операции «левого деления»:

```
>>x=A\B
```

Результат будет выдан в виде вектор – столбца:

x =

```
1
1
1
1
```

Таким образом, система имеет единственное решение:

$$x = [1 \ 1 \ 1 \ 1]$$

Пример 2: решить систему уравнений при помощи функции nnls:

$$\begin{cases} 2x_1 + x_2 - 4x_3 + 6x_4 + 3x_5 - 2x_6 = 16 \\ -x_1 + 2x_2 + 3x_3 + 5x_4 - 2x_5 = -7 \\ x_1 - 2x_2 - 5x_3 + 3x_4 + 2x_5 + x_6 = 1 \\ 4x_1 + 3x_2 - 2x_3 + 2x_4 + x_6 = -1 \\ 3x_1 + 3x_2 - 2x_3 + 2x_4 + 3x_5 + 6x_6 = -11 \\ 5x_1 + 2x_2 - 2x_3 + 3x_4 + x_5 + x_6 = 5 \end{cases}$$

Решение: запишем данную систему в матричном виде:

$$\left[\begin{array}{cccccc} 2 & 1 & -4 & 6 & 3 & -2 \\ -1 & 2 & 3 & 5 & -2 & 0 \\ 1 & -2 & -5 & 3 & 2 & 1 \\ 4 & 3 & -2 & 2 & 0 & 1 \\ 3 & 3 & -2 & 2 & 3 & 6 \\ 5 & 2 & -2 & 3 & 1 & 1 \end{array} \right] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 16 \\ -7 \\ 1 \\ -1 \\ -11 \\ 5 \end{bmatrix}$$

Вновь, как и в примере 1, в командной строке MATLAB® зададим исходные матрицы:

```
>>A=[2 1 -4 6 3 -2;-1 2 3 5 -2 0;1 -2 -5 3 2 1;4 3 -2 2 0 1;3 3 -2 2 3  
6;5 2 -2 3 1 1]
```

```
>>B=[16 ; -7 ; 1 ; -1 ; -11 ; 5]
```

Теперь найдем решение системы, вызвав функцию nnls:

```
>>x=nnls(A,B)
```

Результатом является вектор – столбец:

$x =$

```
0  
0  
0  
0.3610  
1.0258  
0
```

Решение, предложенное функцией nnls, не является единственным, но его положительной чертой является отсутствие в нем отрицательных элементов (что является иногда обязательным условием при решении некоторых задач).

Пример 3: написать функцию, которая возвращает решения произвольной системы уравнений, заданной в матричной форме, полученные и с помощью операции «левого деления», и с помощью функции nnls.

Решение: пусть СЛУ задана в матричном виде (2). Тогда известными являются матрицы А и В. Они будут передаваться в функцию в качестве параметров.

Текст функции (M – файла) представлен ниже:

```
function [left_div_sol,nnls_sol]=joint_solver(A,B)  
    left_div_sol=A\B;  
    nnls_sol=nnls(A,B);  
    return
```

Программа 1 (функция joint_solver)

Теперь рассмотрим, как обращаться к данной функции из командной строки MATLAB®. В качестве исходных значений для данной функции будет использована система из примера 2:

```
>>A=[2 1 -4 6 3 -2;-1 2 3 5 -2 0;1 -2 -5 3 2 1;4 3 -2 2 0 1;3 3 -2 2 3  
6;5 2 -2 3 1 1]
```

```
>>B=[16 ; -7 ; 1 ; -1 ; -11 ; 5]
```

```
>>[x1,x2]=joint_solver(A,B)
```

Результаты расчета – вектор – столбцы x_1 и x_2 – будут иметь такие значения:

$x_1 =$

1.8146
-0.7825
1.0966
-0.0149
3.4178
-3.6878

$x_2 =$

0
0
0
0.3610
1.0258
0

Как видно, результаты различаются. Поэтому при выборе метода решения СПУ следует иметь в виду также и условия (или ограничения), накладываемые на решение.

Контрольные вопросы

1. Приведите алгоритм проведения структурной идентификации на основе экспериментальных данных.
2. Опишите метод выбранных точек.
3. Опишите метод наименьших квадратов.
4. Приведите все команды статистической обработки данных в MATLAB®.

Список рекомендуемой литературы

1. И.М.Бронштейн, К.А.Семеняев Справочник по математике для инженеров и студентов ВТУЗов – М.: Наука, 1986 – стр. 522;
2. Н.И.Данилина, Н.С.Дубровская, О.П.Кваша, Г.Л.Смирнов, Г.И.Феклисов Численные методы: учебник для техникумов – М.: Высш. школа, 1976 – гл. VI;
3. Мак – Кракен Д., Дорн У. Численные методы и программирование на ФОРТРАНе – М.: Мир, 1977 – гл. 8

План лекции:

1. Введение в проблему статистической обработки;
2. Метод структурной идентификации;
3. Методы параметрической идентификации;
4. Основные функции MATLAB® для статистической обработки данных;
5. Примеры решения практических задач статистического анализа.

1. Введение в проблему статистической обработки

В общем виде задача первичной обработки экспериментальных данных может быть сформулирована следующим образом: пусть в результате исследования некоторой величины x значениям x_1, x_2, \dots, x_n поставлены в соответствие значения y_1, y_2, \dots, y_n некоторой величины y . Требуется подобрать вид аналитической зависимости $y=f(x)$, связывающей x и y .

Пусть аналитические зависимости, выявленные в результате эксперимента, называются эмпирическими.

Выявление эмпирических зависимостей можно разделить на 2 этапа – выбор эмпирической формулы (структурная идентификация) и определение коэффициентов выбранной формулы (параметрическая идентификация).

В качестве исходных данных для всех дальнейших исследований и объяснений будут использоваться 2 массива x и y , состоящие из n элементов $x(1), x(2), \dots, x(n)$ и $y(1), y(2), \dots, y(n)$ соответственно. При этом каждому элементу $x(i)$ ставится в соответствие элемент $y(i)$, $i = 1 \div n$.

2. Метод структурной идентификации

Пусть искомая функция y является функцией одной переменной и имеет два параметра a и b . Эмпирическая зависимость будет выбрана из следующих функций:

1. линейная функция $y = ax + b$;
2. показательная функция $y = ab^x$;
3. дробно – рациональная функция $y = \frac{1}{ax+b}$;
4. логарифмическая функция $y = a \ln x + b$;
5. степенная функция $y = ax^b$ (при $b > 0$ – это параболическая зависимость; при $b < 0$ – гиперболическая зависимость; при $b = 0$ – линейная зависимость);
6. гиперболическая зависимость $y = a + \frac{b}{x}$;
7. дробно – рациональная функция $y = \frac{x}{ax+b}$.

Начальным этапом проведения структурной идентификации является построение графика исходных данных (т.е. точек массивов x и y). Затем следует выполнить некоторые дополнительные вычисления. В массиве x выберем 2 точки, достаточно надежные и по возможности – далеко отстоящие друг от друга. Для простоты возьмем крайние точки $x(1)$ и $x(n)$.

Определим среднее арифметическое $x_{ap} = \frac{x(1)+x(n)}{2}$, среднее геометрическое

$x_{geom} = \sqrt{x(1) \cdot x(n)}$ и среднее гармоническое $x_{ гарм } = \frac{2 \cdot x(1) \cdot x(n)}{x(1)+x(n)}$. Затем по вычис-

ленным значениям независимой переменной x найдем из построенного графика соответствующие значения зависимой переменной y :

$$x_{ap} \rightarrow y_1^*$$

$$x_{geom} \rightarrow y_2^*$$

$$x_{ гарм } \rightarrow y_3^*$$

для пока еще неизвестной аналитической зависимости $y=f(x,a,b)$.

Аналогичные вычисления проведем для массива y : $y_{ap} = \frac{y(1)+y(n)}{2}$,

$y_{geom} = \sqrt{y(1) \cdot y(n)}$ и $y_{ гарм } = \frac{2 \cdot y(1) \cdot y(n)}{y(1)+y(n)}$. Имея значения y_{ap} , y_{geom} , $y_{ гарм }$, y_1^* , y_2^* и y_3^* , вычисляем такие погрешности:

$$\varepsilon_1 = |y_1^* - y_{ap}|$$

$$\varepsilon_2 = |y_1^* - y_{geom}|$$

$$\varepsilon_3 = |y_1^* - y_{ гарм }|$$

$$\varepsilon_4 = |y_2^* - y_{ap}|$$

$$\varepsilon_5 = |y_2^* - y_{geom}|$$

$$\varepsilon_6 = |y_3^* - y_{ap}|$$

$$\varepsilon_7 = |y_3^* - y_{ гарм }|$$

Теперь необходимо определить минимальную из полученных семи погрешностей:

$$\varepsilon = \min(\varepsilon_1, \varepsilon_2, \dots, \varepsilon_7)$$

Определив минимальную погрешность, можно провести структурную идентификацию по следующим правилам:

- 1) если $\varepsilon = \varepsilon_1$, то аналитическая зависимость – линейная функция $y = ax + b$;
- 2) если $\varepsilon = \varepsilon_2$, то аналитическая зависимость – показательная функция $y = ab^x$;

- 3) если $\mathcal{E} = \mathcal{E}_3$, то аналитическая зависимость – дробно – рациональная функция вида $y = \frac{1}{ax+b}$;
- 4) если $\mathcal{E} = \mathcal{E}_4$, то аналитическая зависимость – логарифмическая функция $y = a \ln x + b$;
- 5) если $\mathcal{E} = \mathcal{E}_5$, то аналитическая зависимость – степенная функция $y = ax^b$;
- 6) если $\mathcal{E} = \mathcal{E}_6$, то аналитическая зависимость – гиперболическая функция $y = a + \frac{b}{x}$;
- 7) если $\mathcal{E} = \mathcal{E}_7$, то аналитическая зависимость – дробно – рациональная функция вида $y = \frac{x}{ax+b}$.

3. Методы параметрической идентификации

После того, как определен внешний вид функции, которая связывает переменные x и y , необходимо определить значения параметров a и b , которые входят в данную функцию.

В принципе, существует несколько методов определения параметров a и b . Однако на практике, для решения реальных задач, чаще всего применяются следующие:

- а) метод выбранных точек;
- б) метод средних;
- в) метод наименьших квадратов (МНК);

Рассмотрим два из них: метод выбранных точек и МНК (метод средних подробно рассмотрен в [1] и [2]).

3.1. Метод выбранных точек

Метод выбранных точек является самым простым способом, который требует минимум вычислений. Однако его точность очень сильно зависит от точности построения графика, и в этом его реальный недостаток.

Метод выбранных точек заключается в следующем: на построенной кривой берутся две произвольные точки $M(x_1^*, y_1^*)$ и $N(x_2^*, y_2^*)$. Затем составляется система

$$\begin{cases} y_1^* = f(x_1^*, a, b) \\ y_2^* = f(x_2^*, a, b) \end{cases}$$

и решается относительно искомых параметров a и b .

3.2. Метод наименьших квадратов (МНК)

МНК дает более точные результаты (по сравнению с методом выбранных точек), не зависит от точности построения графика, однако требует большего количества вычислений, что иногда замедляет работу и является недостатком этого метода.

Перед рассмотрением МНК введем понятие уклонения Δ_i , которое определяется как разность между табличным (или опытным) значением y_i и истинным значением функции $f(x_i, a, b)$ в точке x_i , т.е.:

$$\Delta_i = y_i - f(x_i, a, b)$$

Согласно МНК, наилучшими параметрами a и b считаются те, для которых сумма квадратов уклонений минимальна:

$$F(a, b) = \sum_{i=1}^n (\Delta_i)^2 = \min$$

Определим частные производные функции $F(a, b)$ по параметрам a и b :

$$\frac{\partial F(a, b)}{\partial a} = -2 \sum_{i=1}^n \Delta_i f'_a(x_i, a, b)$$

$$\frac{\partial F(a, b)}{\partial b} = -2 \sum_{i=1}^n \Delta_i f'_b(x_i, a, b)$$

По условию экстремума функций многих переменных, наилучшими значениями параметров a и b служат те, при которых частные производные этой функции по этим параметрам равны нулю, т.е.

$$\begin{cases} \frac{\partial F(a, b)}{\partial a} = 0 \\ \frac{\partial F(a, b)}{\partial b} = 0 \end{cases}$$

После подстановки получаем:

$$\begin{cases} \sum_{i=1}^n \Delta_i f'_a(x_i, a, b) = 0 \\ \sum_{i=1}^n \Delta_i f'_b(x_i, a, b) = 0 \end{cases}$$

Решая эту систему относительно a и b , получаем искомые оптимальные значения.

Отдельно следует сказать о применении МНК для нахождения параметров эмпирической зависимости вида $y = ax^2 + bx + c$ (в данном случае нужно определять параметры a , b и c).

При использовании МНК необходимо минимизировать следующую функцию:

$$F = \sum_{i=1}^n (\Delta_i)^2 = \sum_{i=1}^n (y_i - (ax_i^2 + bx_i + c))^2 = \min$$

Пользуясь необходимым условием экстремума функции нескольких переменных, получаем следующую систему:

$$\begin{cases} \frac{\partial F}{\partial a} = 2 \sum_{i=1}^n \Delta_i \cdot (-x_i)^2 = 0 \\ \frac{\partial F}{\partial b} = 2 \sum_{i=1}^n \Delta_i \cdot (-x_i) = 0 \\ \frac{\partial F}{\partial c} = 2 \sum_{i=1}^n \Delta_i \cdot (-1) = 0 \end{cases}$$

После преобразований получаем окончательную систему:

$$\begin{cases} a \sum_{i=1}^n x_i^4 + b \sum_{i=1}^n x_i^3 + c \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i^2 y_i \\ a \sum_{i=1}^n x_i^3 + b \sum_{i=1}^n x_i^2 + c \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i \\ a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i + cn = \sum_{i=1}^n y_i \end{cases}$$

Решение этой системы и дает необходимые значения параметров a, b и c.

4. Основные функции MATLAB® для статистической обработки данных

Для выполнения статистических операций над данными в MATLAB® можно применять следующие функции:

mean(x) - возвращает среднее значение элементов вектора или, если x - матрица, то возвращает вектор-строку, чьи элементы - среднее значение элементов каждого столбца матрицы;

median(x) – аналогично mean(x), только возвращает значение медианы вектора (или матрицы);

std(x) - возвращает среднеквадратичное отклонение элементов вектора или, если x - матрица, вектор-строку, чьи элементы - среднеквадратичные отклонения каждого столбца матрицы;

Для построения гистограмм можно использовать следующие функции:

hist(x) - рисует гистограмму элементов вектора, x. Десять точек масштабируются между максимумом и минимумом.

hist(x,n) - рисует гистограмму с n точками, масштабированными между максимумом и минимумом элементов.

Для сортировки и отбора данных существуют такие функции:

max(x) - возвращает максимальное значение среди элементов в векторе или, если x - матрица, то возвращает вектор-строку, чьи элементы - максимальные значения из каждого соответствующего столбца матрицы;

min(x) - возвращает минимум среди элементов x (аналогично max(x));

sort(x) - сортирует значения в векторе x (или столбцах матрицы) в порядке возрастания. Обратите внимание, что эта команда уничтожит любую зависимость, которая может существовать между элементами в строке матрицы x.

Для нахождения суммы и/или произведения элементов массива существуют такие функции:

sum(x) – возвращает сумму элементов вектора или, если x - матрица, возвращает сумму элементов каждого соответствующего столбца матрицы;

prod(x) – аналогично **sum(x)**, только возвращает произведение элементов.

5. Примеры решения практических задач статистического анализа

Для того, чтобы продемонстрировать использование вышеуказанных функций, рассмотрим несколько практических примеров.

Пример № 1: определить вид эмпирической зависимости для указанных табличных значений:

X	1	2	3	4	5	6	7	8	9
Y	521	308	240,5	204	183	171	159	152	147

Решение: для начала построим график по заданным табличным значениям (рис. 1):

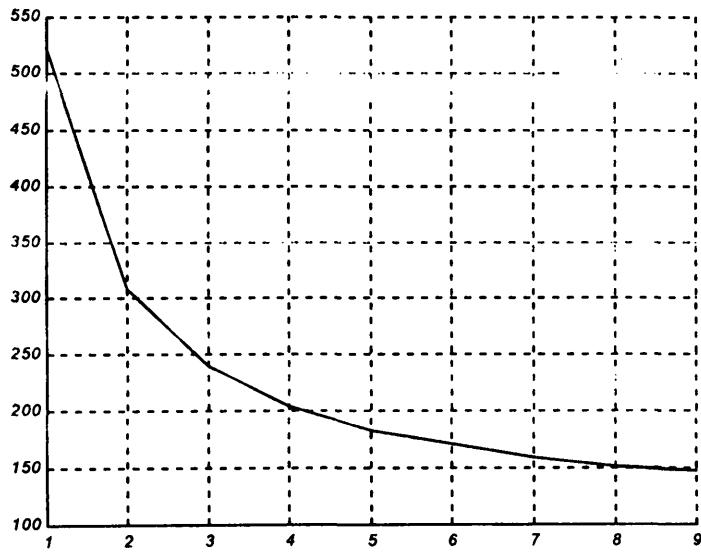


Рис.1 График исходных данных

Затем выполним дополнительные вычисления для массива x:

$x_ar = 5$

$x_geom = 3$

$x_garm = 1.8$

После вычисления дополнительных значений строим их на графике данных (рис. 2):

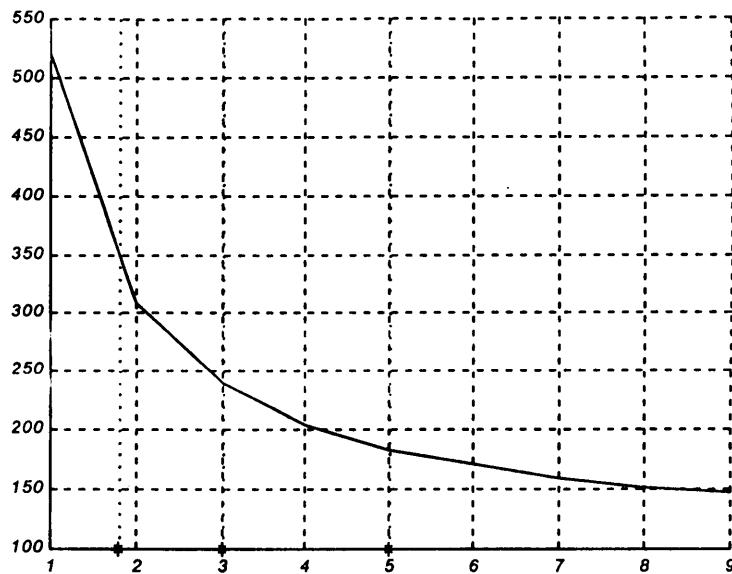


Рис. 2 Вычисленные значения

Теперь приблизительно определим значения переменной y , соответствующие значениям x_{ar} , x_{geom} и x_{garm} :

$$y_1 \approx 180$$

$$y_2 \approx 242$$

$$y_3 \approx 350$$

Теперь вычисляем значения семи разностей по указанным в разделе 3 формулам и определяем минимальную из них. Минимальная ε равна ε_6 , следовательно, эмпирическая зависимость может быть описана функцией 6-го типа - гиперболическая функция $y = a + \frac{b}{x}$.

Приведенная ниже программа выполняет указанные вычисления и оценку:

```

x=[1:9]
y=[521;308;240.5;204;183;171;159;152;147]

n=length(x);

hold on
plot(x,y)
grid

x_ar=(x(1)+x(n))/2;
x_geom=sqrt(x(1)*x(n));
x_garm=(2*x(1)*x(n))/(x(1)+x(n));

disp(['x_ar = ',num2str(x_ar)]);
disp(['x_geom = ',num2str(x_geom)]);
disp(['x_garm = ',num2str(x_garm)]);

pause

plot(x_ar,100,'r*')
plot(x_geom,100,'r*')
plot(x_garm,100,'r*')

plot(x_ar,[100:10:550],'r.')
plot(x_geom,[100:10:550],'r.')
plot(x_garm,[100:10:550],'r.')

pause

y_1=input('Znachenie dlya x_ar = ');
y_2=input('Znachenie dlya x_geom = ');
y_3=input('Znachenie dlya x_garm = ');

y_ar=(y(1)+y(n))/2;
y_geom=sqrt(y(1)*y(n));
y_garm=(2*y(1)*y(n))/(y(1)+y(n));

disp(['y_ar = ',num2str(y_ar)]);
disp(['y_geom = ',num2str(y_geom)]);
disp(['y_garm = ',num2str(y_garm)]);

e=zeros(1,7);

e(1)=abs(y_1-y_ar);
e(2)=abs(y_1-y_geom);
e(3)=abs(y_1-y_garm);
e(4)=abs(y_2-y_ar);
e(5)=abs(y_2-y_geom);
e(6)=abs(y_3-y_ar);
e(7)=abs(y_3-y_garm);

[e_min, pos]=min(e)

disp(['Empiricheskaya zavisimost opisyvaetsya funkciей vida ',num2str(pos)])

```

Программа 1

Выявив вид эмпирической зависимости, определим параметры этой зависимости – коэффициенты a и b . Воспользуемся для этого методом выбранных точек (в качестве таких точек возьмем опытные данные (1;521) и

(4;204)). Решая систему уравнений

$$\begin{cases} 521 = a + \frac{b}{1} \\ 204 = a + \frac{b}{4} \end{cases}, \text{ получаем искомые значения:}$$

$a \approx 98.3333$
 $b \approx 422.6667$. Теперь построим график полученной функции
 $y = 98.3333 + \frac{422.6667}{x}$ и сравним его с графиком исходных данных. Для выполнения такого построения воспользуемся следующей программой:

```
a=98.3333
b=422.6667

x=[1:9]
y=[521;308;240.5;204;183;171;159;152;147]
y1=a+b./x

hold on
plot(x,y)
plot(x,y1, 'r.-')
zoom
```

Программа 2

Результаты работы этой программы представлены на рисунках 3,4.

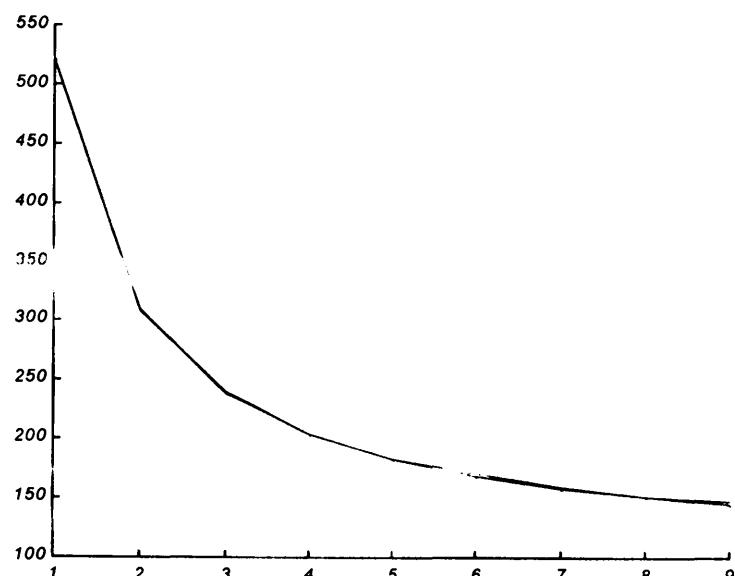


Рис. 3 Совместный график исходных данных (синий) и значений определенной функции (красный).

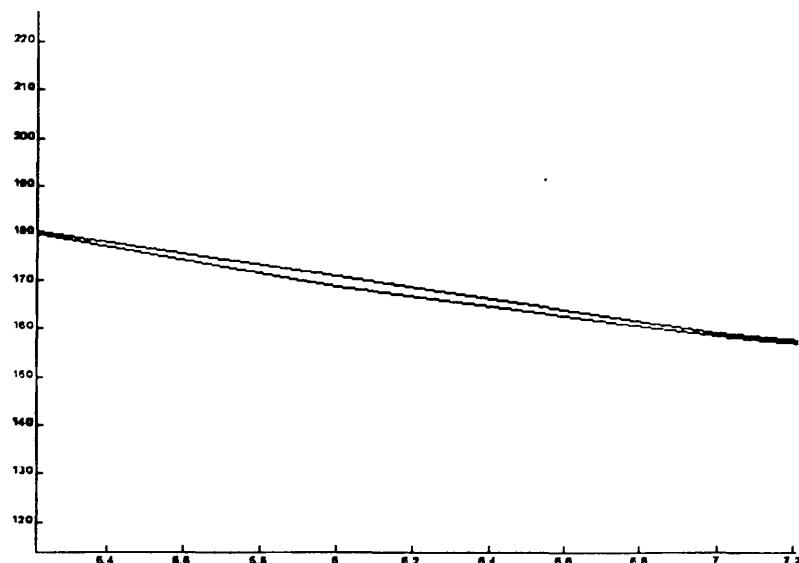


Рис. 4 Фрагмент рис.3, показывающее отклонения между опытными и аналитическими данными на интервале [5.4; 7.2].

Пример № 2: определить параметры a , b и c эмпирической зависимости вида $y = ax^2 + bx + c$ методом наименьших квадратов. Исходные данные заданы следующей таблицей:

X	2	2.2	2.4	2.6	2.8	3
Y	0.3010	0.3424	0.3802	0.4150	0.4472	0.4771

Решение: для решения поставленной задачи воспользуемся следующей программой:

```

x=2:0.2:3;
y=[.3010,.3424,.3802,.4150,.4472,.4771];

A=zeros(3);
B=zeros(3,1);

A(1,1)=sum(x.^4);
A(1,2)=sum(x.^3);
A(1,3)=sum(x.^2);
B(1,1)=sum((x.^2).*y);

A(2,1)=A(1,2);
A(2,2)=A(1,3);
A(2,3)=sum(x);
B(2,1)=sum(x.*y);

A(3,1)=A(2,2);
A(3,2)=A(2,3);
A(3,3)=length(x);
B(3,1)=sum(y);
% A*x=B => x=A\B
x=A\B;
disp(['a= ',num2str(x(1))]);
disp(['b= ',num2str(x(2))]);
disp(['c= ',num2str(x(3))]);

```

Программа 3

Результаты, полученные при помощи этой программы:

$$\begin{aligned}a &= -0.03567 \\b &= 0.35402 \\c &= -0.26414\end{aligned}$$

Таким образом, эмпирическая функция имеет такой вид:

$$y = -0.03567x^2 + 0.35402x - 0.26414$$

Контрольные вопросы

5. Приведите алгоритм проведения структурной идентификации на основе экспериментальных данных.
6. Опишите метод выбранных точек.
7. Опишите метод наименьших квадратов.
8. Приведите все команды статистической обработки данных в MATLAB®.

Список рекомендуемой литературы

1. И.М.Бронштейн, К.А.Семендяев Справочник по математике для инженеров и студентов ВТУЗов – М.: Наука, 1986 – стр. 522;
2. Н.И.Данилина, Н.С.Дубровская, О.П.Кваша, Г.Л.Смирнов, Г.И.Феклисов Численные методы: учебник для техникумов – М.: Высш. школа, 1976 – гл. VI;
3. Е.Н.Львовский Статистические методы построения эмпирических формул: учебное пособие для ВУЗов – М.: Высш. школа, 1988;
4. Д.Мак-Кракен, У.Дорн Численные методы и программирование на ФОРТРАНе – М.: Мир, 1977 – стр. 327 (практический пример 10).

Лекция 7: «Решение задач оптимизации функций одной или нескольких переменных»

План лекции:

1. Математическая постановка задачи оптимизации функций;
2. Методы оптимизации функций (обзор);
3. Функции MATLAB® для решения задач оптимизации;
4. Примеры решения практических задач;

1. Математическая постановка задачи оптимизации функций

При решении многих теоретических и практических задач возникает необходимость найти экстремум (минимальный или максимальный) скалярной функции $q(x_1, x_2, \dots, x_n)$ в n переменных. В общем виде эту задачу можно представить так:

$$q(x_1, x_2, \dots, x_n) = \min_x \quad ,$$

где x - множества значений независимых переменных x_1, x_2, \dots, x_n , а x^* - вектор оптимальных значений x_1, x_2, \dots, x_n (т.е. таких значений, при которых функция $q(x_1, x_2, \dots, x_n)$ принимает минимальное значение).

Следует также сказать, что задачу максимизации $q(x_1, x_2, \dots, x_n)$ можно заменить задачей минимизации. Для этого заданную скалярную функцию $q(x_1, x_2, \dots, x_n)$ заменяют функцией $g(x_1, x_2, \dots, x_n) = -q(x_1, x_2, \dots, x_n)$. Затем осуществляется поиск минимума для новой функции $g(x_1, x_2, \dots, x_n)$. Найденное оптимальное решение x^* , минимизирующее функцию $g(x_1, x_2, \dots, x_n)$, одновременно максимизирует функцию $q(x_1, x_2, \dots, x_n)$.

2. Методы оптимизации функций (обзор)

На сегодняшний день в математике существует большое количество методов для оптимизации функций различных типов. Однако все эти методы можно поделить на две группы – по подходу, используемому для решения поставленной задачи.

К первой группе относятся так называемые косвенные методы. Эти методы сводят решение задачи оптимизации к решению системы нелинейных уравнений, которая является следствием условия экстремума функции многих переменных – в точке экстремума x^* все первые производные функции по независимым переменным равны нулю:

$$\left. \frac{\partial q}{\partial x_i} \right|_{x=x^*} = 0, i = 1 \dots n$$

Примерами «косвенных» методов могут служить метод хорд, метод касательных, метод Ньютона.

Реальным недостатком этих методов является сложность при решении систем нелинейных уравнений. Поэтому для решения практических задач используются другие, т.н. прямые, методы, суть которых заключается в следующем – при решении задачи оптимизации создается последовательность векторов x^0, x^1, \dots, x^n , таких, что

$$q(x^0) > q(x^1) > \dots > q(x^n) > \dots$$

Начальная точка x^0 может быть выбрана произвольно, однако стремятся использовать всю имеющуюся информацию о поведении функции $q(x_1, x_2, \dots, x_n)$ для того, чтобы разместить эту начальную точку x^0 как можно ближе к точке минимума (или максимума).

3. Функции MATLAB® для решения задач оптимизации

Говоря о функциях, используемых для решения задач оптимизации, следует отдельно говорить о функциях, реализованных в самом MATLAB® (ядре пакета), и о возможностях, реализованных в специальной библиотеке (наборе инструментов) Optimization. В данном разделе подробно будут рассмотрены функции, содержащие в ядре (как доступные в любом случае) и дан краткий обзор возможностей библиотеки (набора инструментов).

Однако перед рассмотрением (или описанием) любых функций необходимо упомянуть о дополнительном элементе, присутствующем в MATLAB®. Таким элементом является массив с именем `foptions`, в котором хранятся так называемые «параметры по умолчанию», которые используются оптимизационными процедурами.

Рассмотрим назначение каждого элемента этого массива:

`options(1)` – параметр отображения. По умолчанию равен 0. При установке в 1 отображает некоторые результаты;

`options(2)` – точность прекращения вычислений для x . По умолчанию $-1e-4$;

`options(3)` – точность прекращения вычислений для F . По умолчанию $-1e-4$;

`options(4)` – критерий прерывания по нарушению ограничения. По умолчанию $-1e-6$;

`options(5)` – алгоритм: Стратегия: Используется не всегда;

`options(6)` – алгоритм: Оптимизатор: Используется не всегда;

`options(7)` – алгоритм: Алгоритм линейного поиска. (По умолчанию - 0);

`options(8)` – значение функции (Лямбда в целевой функции)

`options(9)` – этот параметр должен быть установлен в 0, если нужно проверять градиенты, предложенные пользователем;

`options(10)` – число оценок функции и ограничений;

`options(11)` – число оценок градиента функции;

`options(12)` – число оценок ограничений;

`options(13)` – число ограничений на равенство;

`options(14)` – максимальное число оценок функции. (По умолчанию - $100 * \text{количество переменных}$);

`options(15)` – использовано в целевой функции для специальных целей;

`options(16)` – минимальное изменение в переменных для градиентов с конечной разностью;

`options(17)` – максимальное изменение в переменных для градиентов с конечной разностью;

`options(18)` – длина шага. (По умолчанию – 1 или меньше).

Различные оптимизационные процедуры используют при своей работе различные параметры из этого массива. Поэтому для каждой оптимизационной процедуры при ее описании будут отдельно оговорены параметры, которые должны быть заданы, и параметры, которые возвращают какой – то результат (или меняют свое значение).

После рассмотрения настройки параметров можно приступать к рассмотрению самих процедур и функций, используемых для оптимизации.

3.1 Функции ядра MATLAB®

В ядре MATLAB® существуют две функции для решения задачи оптимизации.

Первой функцией является функция `fmin`, предназначенная для минимизации функции одной переменной. У этой функции есть несколько форм записи (синтаксиса):

```
x = fmin('fun',x1,x2)
x = fmin('fun',x1,x2,options)
x = fmin('fun',x1,x2,options,P1,P2, ...)
[x,options] = fmin(...)
```

Описание каждого из этих форматов:

`x = fmin('fun',x1,x2)` – вызов функции в таком формате возвращает значение `x`, которое минимизирует заданную функцию `fun(x)` на интервале `x1 < x < x2`.

`x = fmin('fun',x1,x2,options)` – вызов функции в таком формате выполняет то же, что и описанный выше формат, но использует дополнительные контрольные (управляющие) параметры.

`x = fmin('fun',x1,x2,options,P1,P2,...)` – вызов функции в таком формате выполняет то же, что и описанные выше, однако передает в качестве параметров в функцию `fun(x)` значения `P1, P2, ...` (т.е. в результате получаем `fun(x,P1,P2,...)`). Передает пустую матрицу для опций для того, чтобы использовать значение по умолчанию.

`[x,options] = fmin(...)` – при вызове в таком формате функция возвращает в параметре `options(10)` количество выполненных шагов.

Для систематизации описанных выше параметров составим таблицу:

<code>x1,x2</code>	Интервал, в котором минимизируется функция <i>function</i> .
<code>P1,P2...</code>	Аргументы, передаваемые в функцию <i>function</i> .
<code>fun</code>	Строка, содержащая имя минимизируемой функции.
<code>options</code>	Вектор управляющих параметров. Только три из 18-ти компонентов используются функцией <code>fmin</code> . библиотека <code>Optimization</code> ис-

пользует остальные. Эти тремя управляющими параметрами являются:

- options(1) – если не равен 0, то показываются промежуточные шаги при решении. По умолчанию – 0;
- options(2) – точность прекращения вычислений. По умолчанию - 1.e-4;
- options(14) – максимальное число шагов. По умолчанию - 500.

Алгоритм, примененный в данной функции, основан на методе «золотого сечения» и параболической интерполяции. Программа на языке Фортран, реализующая тот же алгоритм, может быть найдена в : Forsythe, G. E., M. A. Malcolm, and C. B. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, 1976.

Другой функцией, предназначеннной для решения задач оптимизации, является функция fmins, предназначенная для оптимизации функций нескольких переменных. Данная функция также имеет несколько синтаксических форм:

```
x = fmins('fun',x0)
x = fmins('fun',x0,options)
x = fmins('fun',x0,options,[],P1,P2, ...)
[x,options] = fmins(...)
```

Описание каждого из этих форматов:

$x = \text{fmins}('fun',x0)$ – в этом формате функция возвращает вектор x , который является локальным минимумом функции $\text{fun}(x)$ в окрестности точки x_0 .

$x = \text{fmins}('fun',x0,options)$ – в этом формате функция выполняет ту же операцию, что и в описанном выше, однако использует дополнительные управляющие параметры.

$x = \text{fmins}('fun',x0,options,[],P1,P2,...)$ - вызов функции в таком формате выполняет то же, что и описанные выше, однако передает в качестве параметров в функцию $\text{fun}(x)$ значения $P1, P2, \dots$ (т.е. в результате получаем $\text{fun}(x,P1,P2,\dots)$). Передает пустую матрицу для опций для того, чтобы использовать значение по умолчанию.

$[x,options] = \text{fmins}(\dots)$ - при вызове в таком формате функция возвращает в параметре $\text{options}(10)$ количество выполненных шагов.

Для систематизации описанных выше параметров составим таблицу:

x_0	Начальный вектор (стартовая точка)
$P1,P2\dots$	Аргументы, передаваемые в функцию fun .
$[]$	Аргумент, необходимый для обеспечения совместимости с функцией $fminu$ в библиотеке Optimization.

Fun	Строка, содержащая имя минимизируемой функции. Функция <code>fun(x)</code> – скалярная функция от векторной переменной.
options	<p>Вектор управляющих параметров. Только четыре из 18-ти компонентов используются функцией <code>fmins</code>; библиотека Optimization использует остальные. Используемыми параметрами являются:</p> <ul style="list-style-type: none"> • <code>options(1)</code> – если этот параметр не равен 0, то показываются промежуточные шаги при решении. По умолчанию – 0; • <code>options(2) and options(3)</code> -- точность прекращения вычислений (для x и для $function(x)$). По умолчанию (оба значения) - 1.e-4; • <code>options(14)</code> -- максимальное число шагов. По умолчанию - 500.

Алгоритм, используемый в данной функции, называется симплекс – поиском Нелдера – Мида (Nelder-Mead simplex search) и описан в: [1] Nelder, J. A. and R. Mead, "A Simplex Method for Function Minimization," Computer Journal, Vol. 7, p. 308-313, и [2] Dennis, J. E. Jr. and D. J. Woods, "New Computing Environments: Microcomputers in Large-Scale Computing," edited by A. Wouk, SIAM, 1987, pp. 116-122. Это метод прямого поиска, который не требует градиентов или иной производной информации. Если n – длина вектора x , симплекс в n – мерном пространстве характеризуется $(n+1)$ различными векторами, которые являются его вершинами. В двумерном пространстве симплекс – треугольник, в трехмерном – пирамида.

На каждом шаге поиска генерируется новая точка в текущем симплексе или рядом с ним. Значение функции в новой точке сравнивается со значением функции в вершинах (симплекса) и, обычно, одна из вершин заменяется новой, которая затем дает новый симплекс. Этот шаг продолжается до тех пор, пока диаметр симплекса не станет меньше заданной точности.

Таковы функции оптимизации, доступные в ядре MATLAB®. Рассмотрим теперь кратко возможности библиотеки Optimization.

3.2 Обзор возможностей библиотеки (набора инструментов)

Optimization

В библиотеке Optimization собраны все современные функции, предназначенные для оптимизации линейных и нелинейных функций. Основные свойства библиотеки:

- Безусловная оптимизация нелинейных функций;
- Метод наименьших квадратов и нелинейная интерполяция;
- Решение нелинейных уравнений;
- Линейное программирование;
- Квадратичное программирование;
- Условная минимизация нелинейных функций;
- Метод минимакса;
- Многокритериальная оптимизация.

Функции в данной библиотеке реализуют такие алгоритмы:

- Безусловная оптимизация: метод симплексного поиска Нелдера – Мида;
- Условная, многокритериальная оптимизация и метод минимакса: различные варианты метода последовательного квадратичного программирования;
- Методы линейного и квадратичного программирования: метод проекций;
- Возможность выбора метода оптимизации и стратегии линейного поиска.

Также в библиотеку включены примеры, позволяющие посмотреть, как одна и та же задача может быть решена разными методами.

4. Примеры решения практических задач

Для того, чтобы показать практическое применение рассмотренных выше функций `fmin` и `fmins`, решим несколько задач.

Пример 1: рассмотрим решение задачи оптимизации функции одной переменной на примере задачи 2.23 из [1]: в структуре капитальных вложений на развитие химического завода важное место занимают затраты на приобретение и монтаж труб, а также затраты на установку насосов. Рассмотрите проект трубопровода длиной L фут., который должен обеспечивать подачу жидкости со скоростью Q гал/мин. Выбор наиболее экономичного диаметра трубы D (в дюймах) осуществляется на основе минимизации функции затрат на приобретение труб, насосов и прокачивание жидкости. Известно, что функция затрат в единицу времени (при определенных типах труб и насосов) записывается так:

$$f = 0.45L + 0.245LD^{1.5} + 325(hp)^{0.5} + 61.6(hp)^{0.925} + 102,$$

где $hp = 4.4 \cdot 10^{-8} \frac{LQ^3}{D^5} + 1.92 \cdot 10^{-9} \frac{LQ^{2.68}}{D^{4.68}}$.

Сформулируйте соответствующую задачу оптимизации с одной переменной для проектирования трубопровода длиной 1000 фут., который должен обеспечивать подачу жидкости со скоростью 20 гал/мин. Диаметр трубы должен быть заключен в пределах от 0.25 до 0 дюймов.

Решение: решение данной задачи начнем с написания M – файла, в котором описана минимизируемая (целевая) функция. Этот M – файл будет иметь вид:

```
function y=f1(D,L,Q)
hp=4.4e-8*((L*Q^3)./(D.^5))+1.92e-9*((L*Q^2.68)./(D.^4.68));
y=0.45*L+0.245*L.* (D.^1.5)+325*sqrt(hp)+61.6*(hp.^0.925)+102;
return
```

Программа 1 (функция f)

(Обратите внимание, что вообще-то функция затрат зависит от трех параметров – длины трубопровода L , скорости прокачки жидкости Q и диаметра трубы D . Однако в нашем случае, по условию задачи, значения параметров L и Q заданы. Поэтому можно говорить о функции затрат как

с функции одной переменной, а заданные значения передать в функцию при ее вызове). После создания данной программы мы можем воспользоваться командой `fmin`. Для этого в командной строке MATLAB нужно ввести такую команду:

```
>> D_optim=fmin('f1',0.25,6,[],1000,20)
```

Искомое оптимальное значение диаметра будет присвоено переменной `D_optim`.

Функция выдаст такой результат:

```
D_optim =
```

1.1173

Таким образом, затраты будут минимальными, если диаметр трубы равен 1.1173 дюйма.

Для завершения исследования построим график функции затрат от значения диаметра. Диаметр будет изменяться в интервале от 0.25 до 6 (дюймов). Шаг изменения – 0.1 дюйма.

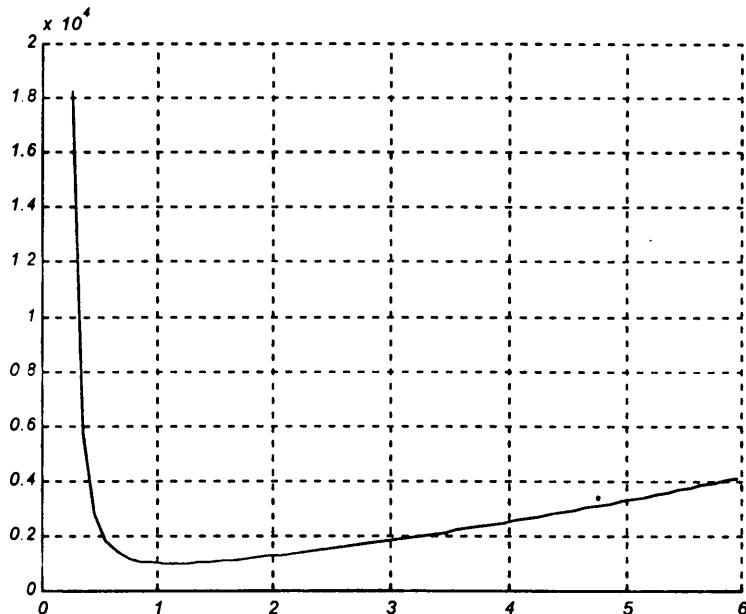


Рис. 1 График поведения функции затрат на интервале [0.25;6]

На рис. 2 приведен фрагмент графика в области точки минимума. Видно, что найденная точка 1.1173 действительно является локальным минимумом заданной функции.

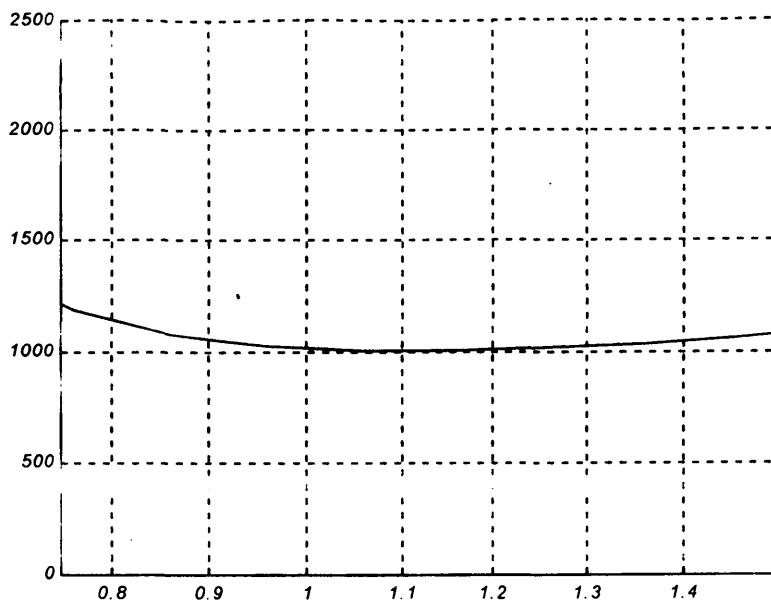


Рис. 2 Фрагмент графика в области точки минимума

Следующие примеры будут посвящены минимизации функций нескольких переменных (и, соответственно, применению функции `fmins`).

Пример 2: этот пример взят из [1], задача 3.18: рассматривается функция Розенброка $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Задана начальная точка $x^0 = [-1.2; 0]^T$. Найти точку x^* , которой соответствует минимальное значение $f(x^*)$.

Решение: для начала создадим M – файл, содержащий описание заданной функции (аналогично примеру 1):

```
function y=f2(x)
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
return
```

Программа 2 (функция f)

После сохранения функции в командной строке MATLAB вводится такая строка:

```
>>X_optim=fmins('f2',[-1.2 0])
```

Искомое оптимальное значение будет сохранено в переменной X_optim. Оно равно:

X_optim =

1.0000 1.0000

Пример 3: этот пример взят из [1], задача 3.17: пусть требуется переправить 400 ярд³ сыпучего материала через реку. Для перевозки груза нужно построить контейнер. Известны следующие данные: стоимость каждого рейса на противоположный берег реки и обратно равна 4.2 долл.; стоимость материалов для изготовления дна контейнера составляет 20 долл/ярд², боковых стенок – 5 долл/ярд², крышки – 20 долл/ярд².

Сконструировать контейнер так, чтобы минимизировать полные затраты на перевозку груза.

Пример: так как в данной задаче не сформулирована минимизируемая функция, мы должны это сделать сами. Для этого выведем функцию полных затрат.

Допустим, что контейнер имеет форму прямоугольного параллелепипеда (рис. 3). Размеры этого параллелепипеда – x , y и z .

Затраты на изготовление контейнера выражаются так:

$$Z_K = Z_D + Z_{CT} + Z_{KP},$$

где:

Z_K – затраты на изготовление контейнера;

Z_D - затраты на изготовление дна;

Z_{CT} - затраты на изготовление стенок;

Z_{KP} - затраты на изготовление крышки.

Используя исходные данные, получаем:

$$Z_D = 20 \cdot S_D = 20xy$$

$$Z_{CT} = 5 \cdot S_{CT} = 5 \cdot (zx+zy) \cdot 2 = 10z(x+y)$$

$$Z_{KP} = 20 \cdot S_{KP} = 20xy$$

Таким образом, суммарные затраты на изготовление контейнера равны:

$$Z_K = 20xy + 10z(x+y) + 20xy = 40xy + 10z(x+y) = 10(4xy + z(x+y))$$

Пусть для перевозки всего груза требуется сделать n рейсов. Тогда суммарные затраты на перевозку Z_{PER} выражаются как

$$Z_{PER} = n \cdot Z_0 + Z_K,$$

где Z_0 – затраты на один рейс (в нашем случае $Z_0 = 4.2$ долл.). После подстановки получаем:

$$Z_{PER} = 4.2n + 10(4xy + z(x+y))$$

Пусть за один рейс перевозится V_0 ярд³ материала. Тогда за n рейсов будет перевезено $n \cdot V_0$ ярд³. По условию, общий объем материала $V = 400$ ярд³. Тогда:

$$n \cdot V_0 = V|_{V=400} = 400$$

Если контейнер – прямоугольный параллелепипед (по допущению), то $V_0 = xyz$. После подстановки получаем, что:

$$n \cdot xyz = 400 \Rightarrow n = \frac{400}{xyz}$$

Итоговое выражение имеет вид:

$$Z_{PER} = 4.2 \cdot \frac{400}{xyz} + 10(4xy + z(x+y)) = \frac{1680}{xyz} + 10(4xy + z(x+y)) \quad (1)$$

Таким образом, задача минимизации сводится к нахождению таких значений для величин x , y и z , которые минимизируют функцию затрат Z_{PER} .

После формулировки задачи, как обычно, создаем M – файл, описывающий минимизируемую функцию:

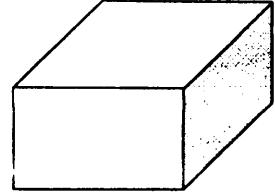


Рис. 3

```
function y=f3(x)
x=x(1);
% v'2'
z=x(3);
y=(1680/(x*y*z))-10*(4*x*y+z*(x+y));
return
```

Программа 3 (функция f3)

Затем в командной строке MATLAB вызываем функцию fmins. В качестве начальной точки возьмем вектор [1 1 1]:

```
>> X_optim=fmins('f3',[1 1 1])
```

Функция вернет такой результат:

```
X_optim =
```

```
1.3932    1.3933    5.5730
```

Таким образом, значение данной функции затрат будет минимальным, если контейнер будет иметь размеры 1.39 x 1.39 x 5.57.

Контрольные вопросы

1. Сформулируйте задачу оптимизации функции в общем виде.
2. Дайте классификацию методов оптимизации. Приведите пример для каждой группы методов.
3. Поясните назначение каждого компонента управляющего стандартного вектора foptions.
4. Приведите все форматы функции fmin.
5. Приведите все форматы функции fmins.

Список рекомендуемой литературы

1. Г.Реклейтис, А.Рейвиндран, К.Рэгсдел «Оптимизация в технике» - М: Мир, 1986 – т.1;
2. Численные методы оптимизации – конспект лекций для бакалавров по направлению В522600 – «Информатика и информационные технологии».

Лекция 8: «Решение задач цифровой обработки сигналов»

План лекции:

1. Обзор цифровой обработки сигналов (ЦОС);
2. Средства MATLAB® для решения задач ЦОС;
3. Примеры решения некоторых стандартных задач ЦОС;

1. Обзор цифровой обработки сигналов (ЦОС)

В современном мире люди окружены всеми типами сигналов в различных формах. Некоторые сигналы – природные, однако большинство из них «созданы» человеком. Некоторые сигналы необходимы (речь), некоторые – приятны (музыка), в то время, как многие – нежелательны или не нужны в данной, конкретной ситуации. С точки зрения инженера сигналы – это носители информации (и нужной, и нежелательной). Следовательно, извлечение или усиление полезной информации из «смеси» противоречащей информации есть простейшая форма обработки сигнала. В более общем виде, обработка сигналов есть операция, созданная для извлечения, усиления, хранения, и передачи полезной информации. Различие между полезной и нежелательной информацией зачастую как объективно, так и субъективно. Поэтому обработка сигналов зачастую является зависимой от приложения.

Рассмотрим, как происходит ЦОС.

Сигналы, которые встречаются на практике, в основном являются аналоговыми сигналами. Эти сигналы изменяются непрерывно во времени и по амплитуде, и обрабатываются с помощью схем, содержащих активные и пассивные элементы. Этот подход известен как **аналоговая обработка сигналов (АОС)** – например, радиоприемники и телевизоры.

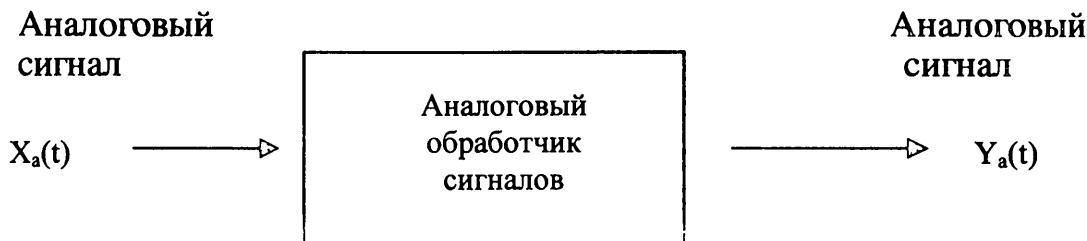


Рис. 1 АОС (из [2])

Эти сигналы также могут быть обработаны с помощью цифровых аппаратных средств, содержащих сумматоры, умножители и логические элементы, или с помощью микропроцессоров специального назначения. Однако это требует преобразования аналоговых сигналов в форму, подходящую для цифрового аппаратного обеспечения. Эта форма сигнала называется **цифровым сигналом**. Этот сигнал (цифровой) принимает одно значение из конечного их числа в конкретный момент времени и, следовательно, может быть представлен двоичными цифрами, или битами. Обработка цифровых сигналов называется **цифровой обработкой сигналов (ЦОС)**. В виде блок – схемы ЦОС можно представить так:



Рис. 2 Блок – схема процесса ЦОС (из [2])

В этой блок – схеме:

ВхФ – входной фильтр, отделяет необходимый сигнал;

АЦП – аналого – цифровой преобразователь, который генерирует поток битов из аналогового сигнала;

Цифровой обработчик сигналов – «сердце» ЦОС, может представлять из себя компьютер общего назначения (например, настольный ПК), процессор специального назначения, цифровую схему и т.д.;

ЦАП – цифро – аналоговый преобразователь, который генерирует ступенчатую волну из последовательности двоичных цифр. Это первый шаг к порождению аналогового сигнала.

ВыхФ – выходной фильтр, предназначен для сглаживания ступенчатой волны в желаемый аналоговый сигнал.

(Подробнее о различных сигналах и взаимосвязи между ними, а также о компонентах системы ЦОС см. [1] пп. 1.1, 1.2, 2.1, 2.2, 2.9)

Исходя из описанных выше подходов к обработке сигналов, может показаться, что ЦОС более сложна и содержит больше компонентов по сравнению с «выглядящей проще» АОС. Однако можно задать вопрос: зачем нужна ЦОС? Ответ лежит в тех преимуществах, которые предлагает ЦОС.

Главный недостаток АОС – это ограниченные возможности для работы в сложных приложениях обработки сигналов. Это отражается на «гибкости» обработки и сложности проектирования систем. В конце концов, это ведет к дороговизне окончательных продуктов и приложений. С другой стороны, используя ЦОС, возможно превратить недорогой ПК в мощный обработчик сигналов. К основным преимуществам ЦОС можно отнести:

1. Системы, использующие ЦОС, могут быть разработаны с использованием программного обеспечения, «работающего» на компьютерах общего назначения. Поэтому система ЦОС относительно удобна для разработки и тестирования. Также в этом случае программное обеспечение становится переносимым;
2. Операции ЦОС основаны только на сложениях и умножениях, что ведет к исключительной стабильности обработки – например, независимость от температур;

3. Операции ЦОС могут быть легко модифицированы. В реальном времени, зачастую – простым изменением в программе или перезагрузкой регистров;
4. Системы ЦОС имеют меньшую стоимость благодаря постоянному удешевлению аппаратных компонентов (памяти, шин, микропроцессоров и т.д.)

Главным недостатком ЦОС является скорость действий и операций, особенно на больших частотах. В основном из-за вышеуказанных преимуществ ЦОС сейчас становится основным используемым решением во многих технологиях и приложениях, таких, например, как бытовая электроника, коммуникации, мобильная связь.

Большинство операций ЦОС могут быть разделены или на задачи анализа сигналов, или на задачи фильтрации (см. рис 3).

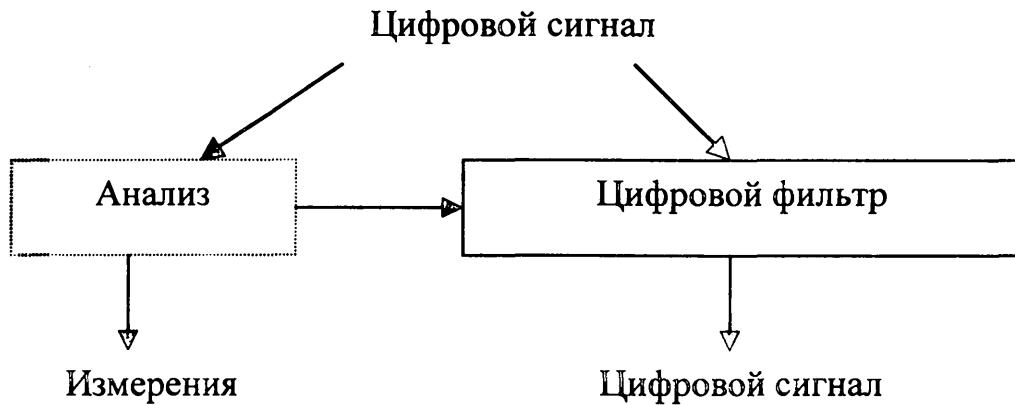


Рис. 3 Классификация задач ЦОС (из [2])

1.1 Анализ сигналов

Задача анализа сигналов связана с измерением свойств сигнала. В принципе - это операция в частотной области. Некоторые приложения, где это необходимо:

- спектральный (частотный и/или фазовый) анализ;
- распознавание речи;
- определение целей;

1.2 Фильтрация сигналов

Задача фильтрации сигналов характеризуется ситуацией «сигнал на входе – сигнал на выходе». Системы, которые выполняют эти задачи, в общем случае называются **фильтрами**. Обычно (но не всегда) это операция во временной области. Некоторые приложения фильтрации:

- удаление нежелательного фонового шума;
- удаление помех;
- выделение частотных полос;
- формирование спектра сигнала.

В некоторых приложениях (например, синтез голоса) сигнал сначала анализируется для изучения его характеристик, которые затем используются в цифровой фильтрации для генерации синтетического голоса.

2. Средства MATLAB® для решения задач ЦОС

Собственно, само ядро MATLAB® не имеет никаких специальных средств для реализации ЦОС. Поэтому пользователям, заинтересованным в таких средствах, нужно самостоятельно их разрабатывать.

Однако, как уже говорилось выше, MATLAB® для решения задач обработки сигналов предлагает две библиотеки – Signal Processing и Wavelet, в которых уже реализовано большинство необходимых функций (причем выбраны оптимальные алгоритмы). Кроме того, данные библиотеки поставляются вместе с ядром MATLAB® «по умолчанию» поэтому практически любой пользователь при установке получает доступ к ним.

Библиотека Signal Processing непосредственно предназначена для решения задач обработки сигналов. В этой библиотеке содержится более 100 функций, выполняющих все необходимые операции. Все функции сгруппированы по смыслу в разделы. Вот краткое описание этих разделов:

- «Генерация сигналов и их графическое изображение» - содержит функции для генерации сигналов заданной формы (синусоидальный (с постоянной амплитудой и затухающий), пилообразный, прямоугольные импульсы и др.);
- «Анализ и реализация фильтра» - функции этого раздела реализуют некоторые стандартные алгоритмы фильтрации. Исходные данные (входной сигнал), а также параметры «фильтров», реализованных в виде функций, передаются при вызове соответствующих функций;
- «Линейные преобразования систем» - содержит функции, выполняющие преобразования систем, представленных в виде полиномов, из одного описания в другие;
- «Прямая и классическая разработка фильтра с бесконечной импульсной характеристикой (БИХ)» - этот раздел содержит функции, позволяющие разрабатывать некоторые классические модели фильтров – фильтры Бесселя, Чебышева (I и II типов), Баттерворта;
- «Выбор порядка фильтра с БИХ» - содержит функции для выбора порядка фильтра Баттерворта, фильтров Чебышева (I и II типов) и эллиптического фильтра;
- «Разработка фильтра с конечной импульсной характеристикой (КИХ)» - содержит функции для проектирования фильтров с КИХ методом окон, методом наименьших квадратов и некоторыми другими стандартными методами;
- «Преобразования» - содержит функции, для выполнения преобразований Фурье (прямого и обратного), Гильберта и z – преобразований;
- «Статистическая обработка сигналов» - функции этого раздела выполняют статистическую обработку сигналов: определение взаимокорреляционной функции, определение спектральной плотности энергии, некоторых других статистических параметров;

- «Окна» - содержит реализации различных методов окон (окна Бартлетта, Чебышева, Кайзера и др.);
- «Параметрическое моделирование» - функции этого раздела позволяют проводить идентификацию фильтров на основании данных (во временной и частотной областях);
- «Специализированные операции» - содержит функции, выполняющие дополнительные операции над данными (например, деление многочленов или модуляцию/демодуляцию сигнала);
- «Разработка аналогового прототипа» - функции этого раздела позволяют разработать аналоговые прототипы классических фильтров (Бесселя, Баттервортса, Чебышева (I и II типов));
- «Преобразование частоты» - содержит функции для преобразования аналогового фильтра низких частот в другие (полосовой, задерживающий, фильтр высоких частот, фильтр низких частот);
- «Дискретизация фильтра» - эти функции используются для преобразования фильтра из аналогового в цифровой;
- «Интерактивные средства» - этот раздел содержит функцию `sptool`, которая загружает интерактивное визуальное средство для выполнения обработки сигналов, фильтрации и спектрального анализа.

Более подробно, с форматом и описанием параметров, некоторые функции этой библиотеки будут рассмотрены дальше, при описании решения задач ЦОС.

3. Примеры решения некоторых стандартных задач ЦОС

Рассмотрим одну из стандартных задач ЦОС - определение спектра сигнала (использование преобразования Фурье)

Для того, чтобы рассматривать пути решения этой задачи, следует дать определение понятию «спектр сигнала».

Если какой – либо колебательный процесс представляется в виде суммы гармонических колебаний различных частот (т.н. «гармоник»), то спектром колебательного процесса называется функция, описывающая распределение амплитуд по различным частотам. Спектр показывает, какого рода колебания преобладают в данном процессе, какова его внутренняя структура.

Для определения спектра сигнала используется аппарат преобразования Фурье (прямого и обратного), которое используется для описания сигналов в частотной области.

Спектром $X_a(j\omega)$ аналогового сигнала $x_a(t)$ называют прямое преобразование Фурье

$$X_a(j\omega) = \int_0^{\infty} x_a(t) e^{-j\omega t} dt \quad (1)$$

При помощи обратного преобразования Фурье можно через спектр выразить сам сигнал:

$$x_a(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X_a(j\omega) e^{j\omega t} d\omega \quad (2)$$

Спектром $X_a(j\omega T)$ дискретного сигнала $x(nT)$ называют прямое преобразование Фурье:

$$X(e^{j\omega T}) = \Phi\{x(nT)\} = \sum_{n=0}^{\infty} x(nT)e^{-j\omega nT} \quad (3)$$

Сигнал $x(nT)$ можно выразить через спектр при помощи обратного преобразования Фурье:

$$x(nT) = \Phi^{-1}\{X(e^{j\omega T})\} = \frac{T}{2\pi} \int_{-\pi/T}^{\pi/T} X(e^{j\omega T}) d\omega \quad (4)$$

Определение преобразования Фурье для непрерывной функции (т.е. для аналогового сигнала) можно найти в [3]. Дискретное преобразование Фурье можно определить так [1]: пусть $x(nT)$ - периодическая последовательность с периодом NT (период - N отсчетов), т.е. $x(nT) = x(nT + mNT)$, m - целое. Дискретным преобразованием Фурье (ДПФ) называют пару взаимно – однозначных преобразований:

$$X(k) = X(k\Omega) = \sum_{n=0}^{N-1} x(nT)e^{-jkn\Omega T}, \quad k = 0, 1, \dots, N-1 \quad (5)$$

$$x(n) = x(nT) = \frac{1}{N} \sum_{k=0}^{N-1} X(k\Omega)e^{jkn\Omega T}, \quad n = 0, 1, \dots, N-1 \quad (6)$$

Выражение (5) определяет прямое ДПФ, а выражение (6) – обратное ДПФ.

В этих преобразованиях $\Omega = \frac{2\pi}{NT}$ – основная частота преобразования (бинар ДПФ). Обозначая т.н. поворачивающий множитель $e^{-j\Omega T} = e^{-j2\pi/N} = W_N$, ДПФ и ОДПФ можно записать так:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \quad k = 0, 1, \dots, N-1 \quad (7)$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)W_N^{-kn}, \quad n = 0, 1, \dots, N-1 \quad (8)$$

ДПФ $X(k)$, как и сама последовательность $x(n)$, является периодической функцией по аргументу k с периодом N , т.к. $W_N^{kn} = W_N^{(k+mN)n}$, где m – целое. ДПФ может быть использовано и для представления последовательности $x(nT)$ конечной длины N , определенной при $n=0, 1, 2, \dots, N-1$ и равной нулю вне интервала $[0; N-1]$. Действительно, такую последовательность можно рассматривать как один период соответствующей периодической последовательности и использовать преобразования (7) и (8); следует только считать, что вне интервала $[0; N-1]$ $X(k)$ и $x(n)$ равны нулю.

При сравнении спектра конечного дискретного сигнала, определяемого формулой (3) (с учетом того, что $x(nT)=0$ при $n < 0$ и $n > N-1$), и ДПФ этого же сигнала (формула (5)) видно, что ДПФ – это N отсчетов спектра, взятых на периоде с интервалом дискретизации по частоте, равным $\Omega = 2\pi/NT$.

В библиотеке Signal Processing для выполнения ДПФ (прямого и обратного) существуют две функции:

$y = \text{fft}(x, N)$ – вычисляет N – точечное ДПФ. Если длина вектора x меньше, чем N , то x дополняется нулями. Если аргумент N опущен, то

длина ДПФ равна длине вектора x . Если x – матрица, то N – точечное ДПФ выполняется для каждого столбца x .

$Y = \text{ifft}(x, N)$ – вычисляет N – точечное ОДПФ. Параметры этой функции аналогичны параметрами функции $\text{fft}(x, N)$.

Отличительной чертой этих функций является то, что они написаны на машинном языке (т.е. они недоступны в виде M – файлов). Кроме того, они реализуют специальные алгоритмы, называемые алгоритмами быстрого преобразования Фурье (БПФ) (отсюда и название функции – Fast Fourier Transform – FFT), которые значительно ускоряют выполнение ДПФ. Все это в сумме (машинная реализация + специальные алгоритмы) дает очень высокую скорость выполнения ДПФ при использовании функций fft и ifft .

Рассмотрим примеры использования функции $\text{fft}()$ для определения спектра дискретных сигналов.

Пример 1: сигнал описывается выражением $s=3.5\cos(0.3\pi t)$. Время дискретизации $T_s=0.3$ сек, количество отсчетов $N=30$. Построить график сигнала на заданном отрезке времени. Выполнить преобразование Фурье над сигналом. Построить графики абсолютного значения (амплитуды) и фазы преобразованного сигнала (т.е. построить графики АЧХ и ФЧХ).

Скрипт – файл, решающий эту задачу, дан ниже:

```
clear
clc
T_s=0.3;
N=30;
time=[0:N-1]*T_s;
S=3.5*cos(0.3*pi.*time);
figure
plot(time,S); % figure #1
grid
xlabel('Time in sec')
ylabel('Signal s(t)')
title ('Periodical signal (time domain)')
zoom
S_FT=fft(S);
freq_plot=[0:N-1]./(N-1);
figure % figure #2
grid
plot(freq_plot,real(S_FT)), grid
title('Absolute value of transformed signal (frequency domain)');
ylabel('Abs. val. of signal')
xlabel('Frequency (in pi units)')
figure % figure #3
grid
plot(freq_plot,imag(S_FT)), grid
title('Phase (angle)of transformed signal (frequency domain)');
ylabel('Phase of signal')
xlabel('Frequency (in pi units)')
```

Результаты выполнения этого файла (т.е. решение задачи) показано в виде графиков на рис. 4 – 6.

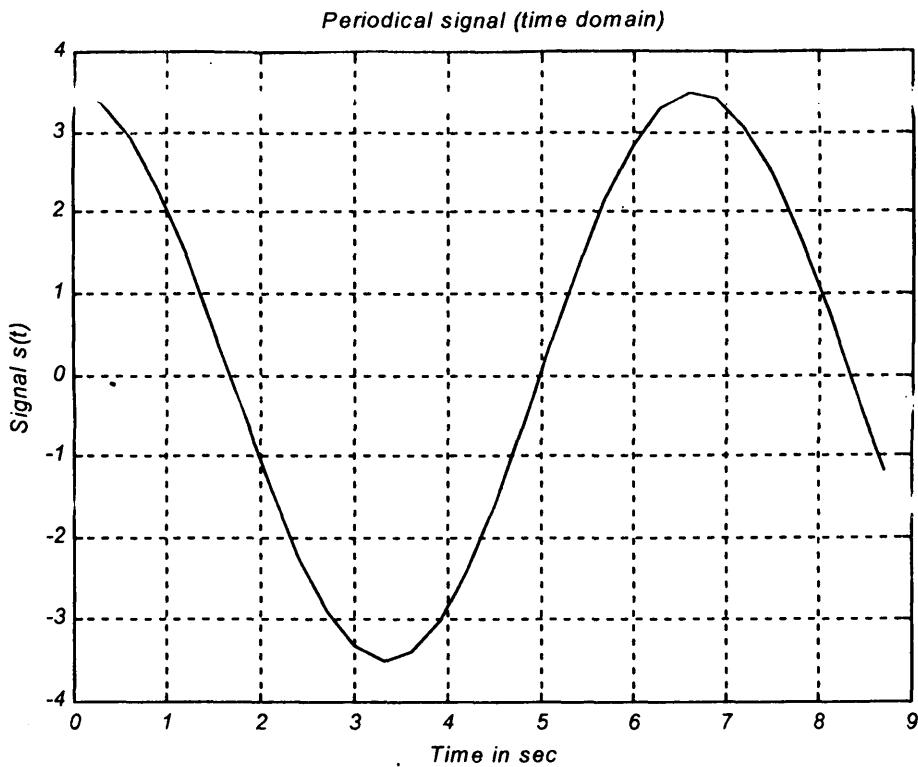


Рис 4 График сигнала на заданном временном отрезке

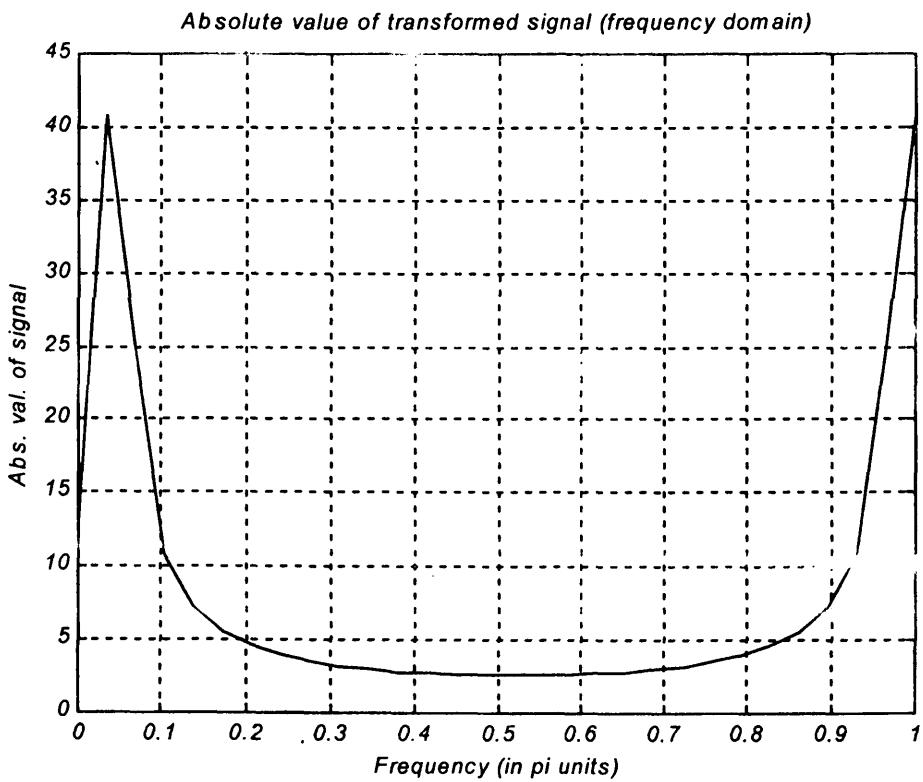


Рис 5 График амплитуды сигнала в частотной области

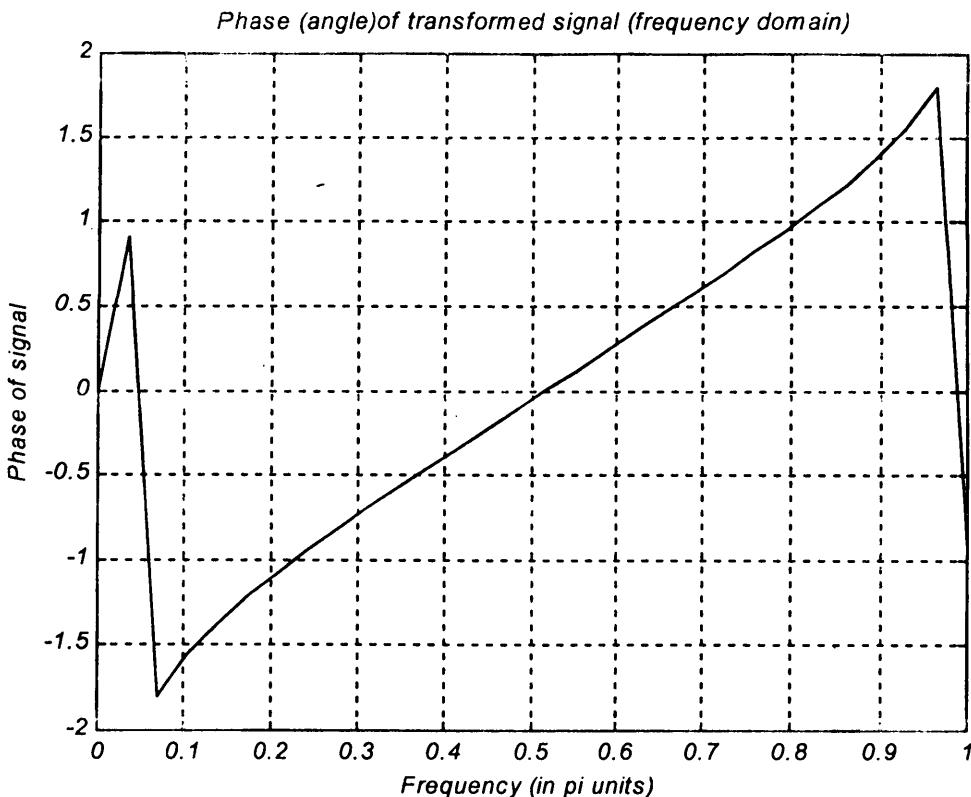


Рис 6 График фазы сигнала в частотной области

Пример 2: сигнал описывается выражением $s=3.91e^{-t}$. Время дискретизации $T_s=0.1$ сек, количество отсчетов $N = 110$. Построить график сигнала на заданном отрезке времени. Выполнить преобразование Фурье над сигналом. Построить графики абсолютного значения (амплитуды) и фазы преобразованного сигнала (т.е. построить графики АЧХ и ФЧХ).

Скрипт – файл, решающий данную задачу, дан ниже:

```

clear
clc
T_s=.1;
N=110;
time=[0:N]*T_s;
S=3.91*exp(-time);
plot(time,S); % figure #1
grid
xlabel('Time in sec')
ylabel('Signal s(t)')
title ('Exponential signal (time domain)')
zoom
S_FT=fft(S);
freq_plot=[0:N]./N;
figure % figure #2
grid
subplot(2,1,1), plot(time,abs(S_FT)), grid, zoom
title('Absolute value of transformed signal (time domain)');
ylabel('Abs. val. of signal')
xlabel('Time (in sec)')
subplot(2,1,2), plot(freq_plot,abs(S_FT)), grid, zoom
title('Absolute value of transformed signal (frequency domain)');
ylabel('Abs. val. of signal')
xlabel('Frequency (in pi units)')

```

```

figure % figure #3
zoom
subplot(2,1,1), plot(time,angle(S_FT)), grid, zoom
title('Phase (angle) of transformed signal (time domain)');
ylabel('Phase of signal')
xlabel('Time (in sec)')
subplot(2,1,2), plot(freq_plot,angle(S_FT)), grid, zoom
title('Phase (angle) of transformed signal (frequency domain)');
ylabel('Phase of signal')
xlabel('Frequency (in pi units)')

```

Результаты в виде графиков даны на рис 7 – 9

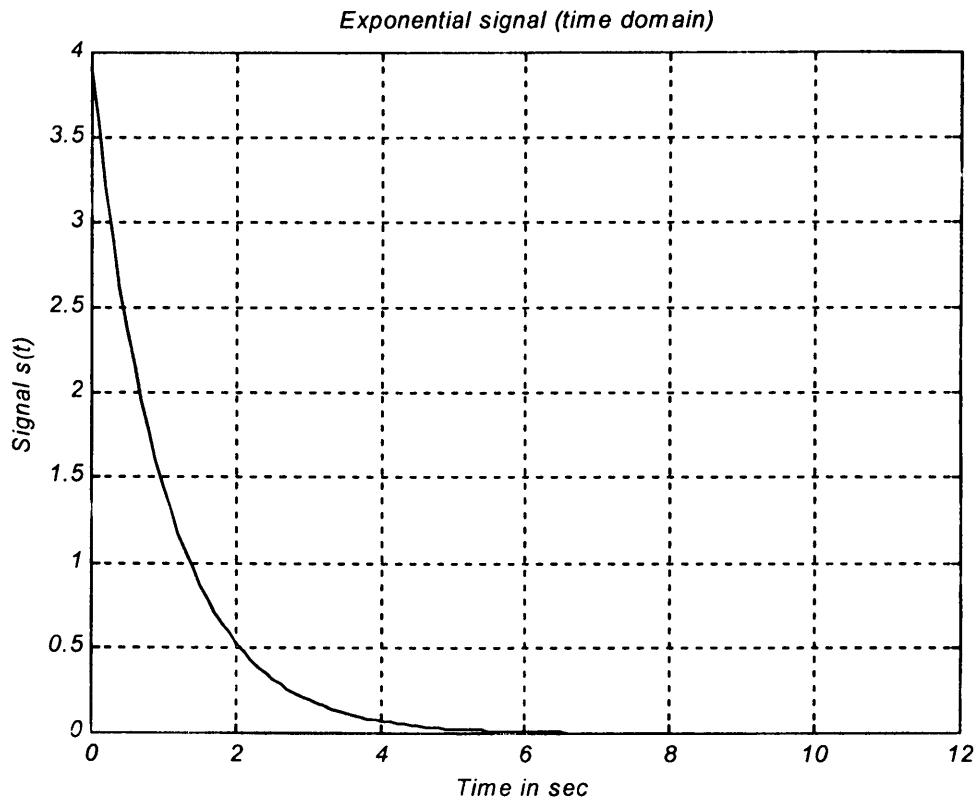


Рис 7 График сигнала во временной области

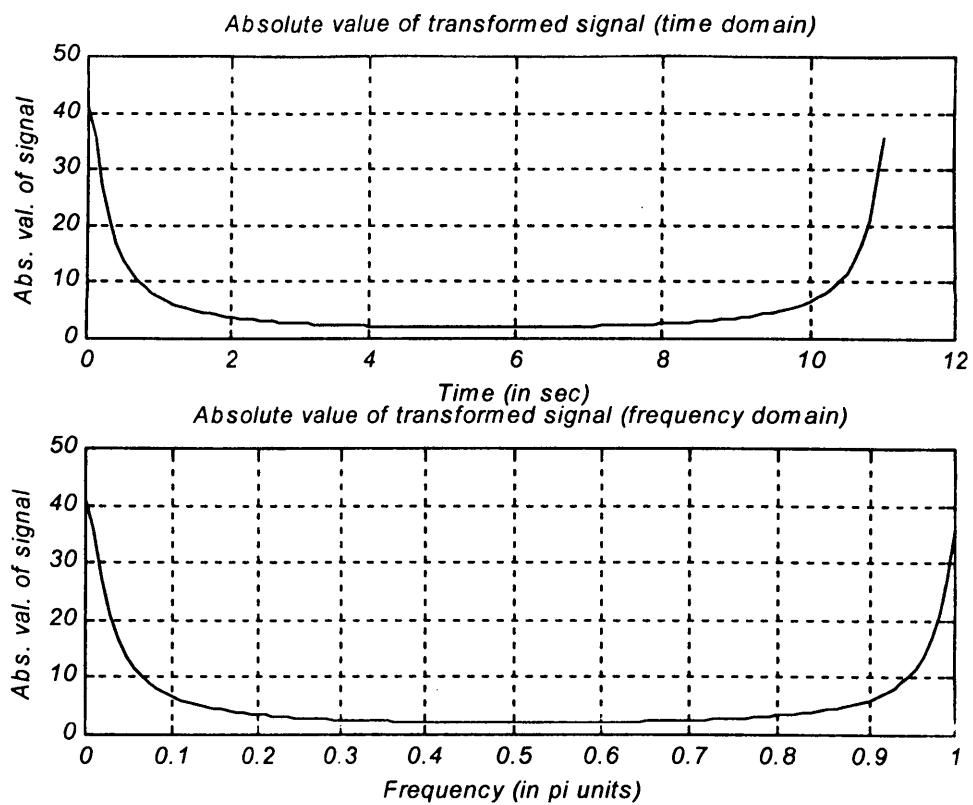


Рис 8 График амплитуды сигнала (во временной и в частотной областях)

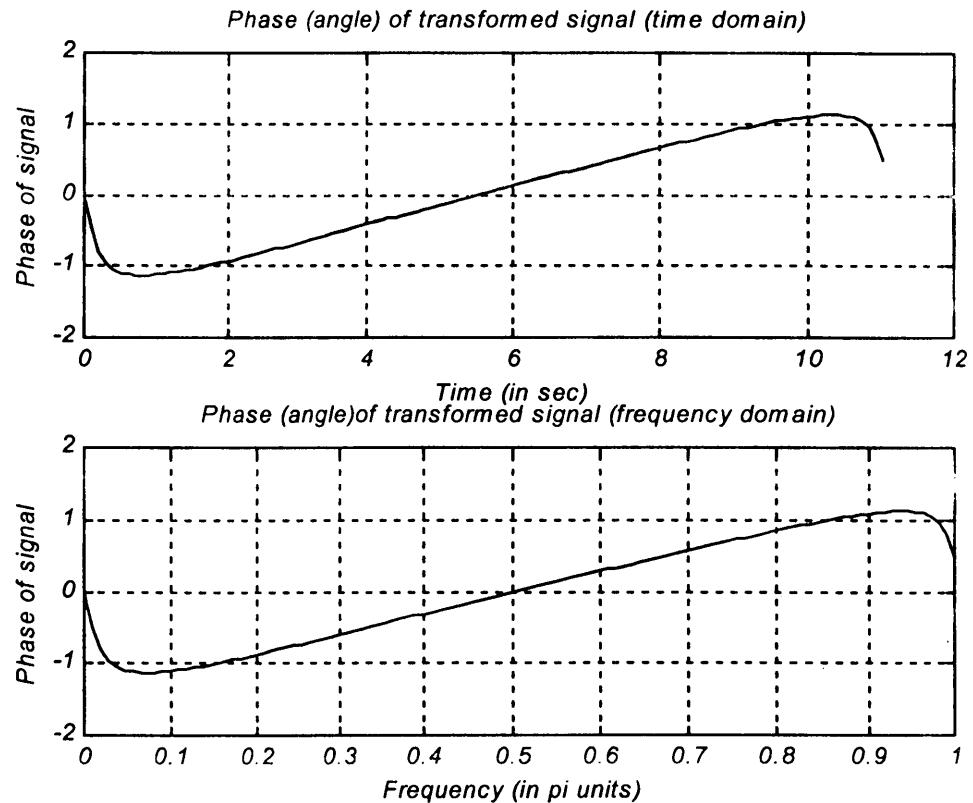


Рис 9 График фазы сигнала (во временной и в частотной областях)

Определения и термины

Спектром колебательного процесса называется функция, описывающая распределение амплитуд по различным частотам.

Контрольные вопросы

1. Чем отличаются аналоговая обработка сигналов и цифровая обработка сигналов?
2. Нарисуйте блок – схему ЦОС.
3. Укажите преимущества и недостатки ЦОС.
4. На какие разделы условно можно поделить функции библиотеки Signal Processing?
5. Что такое спектр?
6. Что такое преобразование Фурье? Для чего оно служит?

Список рекомендуемой литературы

1. Голбденберг Л.М., Мтюшкин Б.Д., Поляк М.Н. – Цифровая обработка сигналов (учебное пособие для ВУЗов) – М.: Радио и связь, 1990;
2. Vinay K. Ingle, John G.Proakis Digital Signal Processing using MATLAB® - BROOKS/COLE, 2000;
3. Бронштейн И.Н., Семеняев К.А. Справочник по математике для инженеров и учащихся ВТУЗов - 13-е изд-е – стр. 425

Лекция 9: «Решение дифференциальных уравнений с помощью MATLAB®»

План лекции:

1. Математическое описание дифференциальных уравнений, их классификация;
2. Численные методы решения дифференциальных уравнений (обзор);
3. Решение дифференциальных уравнений средствами MATLAB®.
4. Примеры решения практических задач.

1. Математическое описание дифференциальных уравнений, их классификация

Уравнения, содержащие производную функции одной переменной, возникают во многих областях прикладной математики. В принципе, любая физическая ситуация, в которой рассматривается степень изменения одной переменной по отношению к другой переменной, описывается дифференциальным уравнением (ДУ) – уравнением, в которое неизвестная функция входит под знаком производной (или дифференциала).

Если неизвестная функция, входящая в ДУ, зависит только от одной независимой переменной, то ДУ называется обыкновенным.

Если же функция, входящая в ДУ, является функцией нескольких переменных, то ДУ называется уравнением в частных производных.

Порядком ДУ называется наивысший порядок производной, входящей в уравнение. В данной лекции будут рассматриваться только обыкновенные ДУ.

Обыкновенное ДУ n -го порядка в самом общем случае содержит независимые переменные, неизвестную функцию и ее производные или дифференциалы до n -го порядка включительно и имеет вид:

$$F(x, y, y^{'}, y^{''}, \dots, y^{(n)}) = 0 \quad (1)$$

В этом уравнении x – независимая переменная, y – неизвестная функция, $y^{'}, y^{''}, \dots, y^{(n)}$ – производные этой функции.

ДУ n -го порядка, разрешенное относительно старшей производной, может быть записано в виде:

$$y^{(n)} = f(x, y, y^{'}, \dots, y^{(n-1)}) \quad (2)$$

Решением (или интегралом) уравнения (2) называется всякая дифференцируемая функция $y = \varphi(x)$, удовлетворяющая этому уравнению, т.е. такая, после подстановки которой в уравнение (2) оно обращается в тождество.

График решения обыкновенного ДУ называется интегральной кривой этого уравнения.

Решение ДУ, содержащее столько независимых произвольных постоянных (параметров), каков его порядок, называется общим решением (или общим интегралом) этого уравнения.

Геометрически общее решение ДУ представляет собой семейство интегральных кривых этого уравнения.

Частным решением ДУ называется всякое решение, которое может быть получено из общего при определенных числовых значениях произвольных постоянных, входящих в общее решение.

Произвольные постоянные, входящие в общее решение, определяются из т.н. начальных условий.

В общем виде задача с начальными условиями ставится так: найти решение $y = \varphi(x)$ уравнения $y^{(n)} = f(x, y, y', \dots, y^{(n-1)})$, удовлетворяющее дополнительным условиям, состоящим в том, что решение $y = \varphi(x)$ должно принимать вместе со своими производными до $(n-1)$ -го порядка заданные числовые значения $y_0, y'_0, y''_0, \dots, y_0^{(n-1)}$ при заданном числовом значении $x = x_0$ независимой переменной x :

$$y = y_0, y' = y'_0, y'' = y''_0, \dots, y^{(n-1)} = y_0^{(n-1)} \text{ при } x = x_0 \quad (3)$$

Условия (3) называются начальными условиями, числа $x_0, y_0, y'_0, y''_0, \dots, y_0^{(n-1)}$ - начальными данными решения, а задача отыскания решения $y = \varphi(x)$, удовлетворяющего начальным условиям (3), - задачей с начальными условиями, или задачей Коши.

В случае уравнений 1-го порядка, т.е. при $n = 1$, получаем задачу Коши для уравнения $y' = f(x, y)$ с начальными условиями $x = x_0, y = y_0$.

2. Численные методы решения дифференциальных уравнений (обзор)

Точное решение ДУ возможно только для достаточно небольшого класса уравнений, встречающихся на практике. Поэтому существуют методы приближенного решения ДУ. Эти методы, в зависимости от формы представления решения, можно разделить на 2 группы:

1) аналитические методы – дают приближенное решение ДУ в виде аналитического выражения;

2) численные методы – дают приближенное решение в виде таблицы;

В данном разделе будут кратко охарактеризованы численные методы решения ДУ: метод Эйлера и его модификации, методы Рунге – Кутта и Адамса. Более подробно эти методы описаны в [1] и [2].

Перед рассмотрением методов следует дать определение численному решению ДУ. Решить дифференциальное уравнение $y' = f(x, y)$ численным методом – это значит для заданной последовательности аргументов x_0, x_1, \dots, x_n и числа y_0 , не определяя функцию $y = F(x)$, найти такие значения y_1, y_2, \dots, y_n , что $y_i = F(x_i)$ ($i = 1, 2, \dots, n$) и $F(x_0) = y_0$. Таким образом, численные методы позволяют вместо нахождения функции $y = F(x)$ получить таблицу значений этой функции для заданной последовательности аргументов. Величина $h = x_k - x_{k-1}$ называется шагом интегрирования.

Условно численные методы можно поделить на 2 класса:

- одноступенчатые методы – в них используется только информация о самой кривой в одной точке и не производятся итерации. Эти мето-

- ды являются прямыми (т.е. без итераций), однако требуют много-кратных повторных вычислений функции;
- многостепенные методы – в них следующую точку кривой можно найти, не производя так много повторных вычислений функции, как в одноступенчатых методах.

Для систематизации сведений о численных методах решения ДУ можно составить такую таблицу:

Класс	Представители (некоторые конкретные методы)	Достоинства	Недостатки
Одноступенчатые	Методы Рунге – Кутта, метод Эйлера	<p>а) т.к. в них используется информация только об очередной точке и не используется информация о ранее найденных точках, то с помощью этих методов можно начинать решение;</p> <p>б) используя информацию только об очередной точке решения, эти методы позволяют очень легко менять величину шага h;</p>	<p>а) при использовании этих методов много-кратно приходится вычислять функцию $f(x,y)$ и затрачивать на это много машинного времени;</p> <p>б) при использовании этих методов весьма отрудно получить оценку для ошибки ограничения;</p>
Многостепенные		<p>а) они более экономичны (по сравнению с одноступенчатыми) в смысле затрат машинного времени, т.к. в этих методах вместо вычисления $f(x,y)$ используется информация о ранее вычисленных точках;</p> <p>б) в результате можно получить также и оценку ошибки ограничения;</p>	<p>а) их нельзя использовать для начала решения, т.к. они используют предыдущие вычисленные значения;</p> <p>б) в большинстве случаев после изменения величины шага h приходится временно возвращаться к методам Рунге – Кутта.</p>

Сравнивая свойства этих групп методов, видно, что они удачно дополняют друг друга, поэтому чаще всего при решении практических задач используют комбинацию этих методов.

3. Решение дифференциальных уравнений средствами MATLAB®

3.1 Формулирование задачи решения ДУ

Перед тем, как использовать функции для решения ДУ, пользователь должен сформулировать саму задачу, которую необходимо решить. Для облегчения этой задачи MATLAB® предлагает вспомогательное средство. Это раздел встроенной системы помощи MATLAB®, в котором по шагам описан процесс создания функции (по умолчанию эта функция называется `odefile`, однако пользователь может задать любое имя, которое пожелает), которая затем может быть использована при вызове функций – «решателей» ДУ.

Пользуясь этим стандартным вспомогательным средством, можно определять ДУ таких видов: $y' = F(t, y)$ или $My' = F(t, y)$. Здесь: t – скалярная независимая величина (обычно представляет время); y – вектор зависимых переменных; F – функция от t и y , возвращающая вектор – столбец такой же длины, как и y ; M – константа.

Для того, чтобы использовать вспомогательное средство, следует выполнить следующее:

1. ввести команду `help odefile` в командной строке MATLAB® для того, чтобы получить на экране «шаблон» функции;
2. скопировать текст («шаблон») функции `odefile` в отдельный файл;
3. отредактировать новый файл (с текстом «шаблона») так, чтобы устранить все случаи, не применимые в данном конкретном случае;
4. задать те сведения, которые даны и известны.

Ниже показан текст «шаблона» функции и указаны те параметры, которые должны быть изменены (или заданы) при создании конкретной функции.

```

function [out1,out2,out3] = odefile(t,y,flag,p1,p2) 1
% ODEFILE The template for ODE files.
%
```

```

if nargin < 3 | isempty(flag) % Return dy/dt = F(t,y) 2
    out1 = < Insert a function of t and/or y, p1, and p2 here >;
else

```

```

switch(flag) 3
case 'init'           % Return default [tspan, y0, and options]

```

```

    out1 = < Insert tspan here >; 4
    out2 = < Insert y0 here >;
    out3 = < Insert options = odeset(...) or [] here >;

```

```

case 'jacobian'      % Return matrix J(t,y) = dF/dy 5
    out1 = < Insert Jacobian matrix here >;

```

```

case 'jpattern'       % Return sparsity pattern matrix S 6
    out1 = < Insert Jacobian matrix sparsity pattern here >

```

```

case 'mass'           % Return mass matrix M(t) or M 7
    out1 = < Insert mass matrix here >;

```

```

case 'events'          % Return event vector and info
    out1 = < Insert event function vector here >;

```

```

    out2 = < Insert logical isterminal vector here >; 8
    out3 = < Insert direction vector here >;

```

```

otherwise
    error(['Unknown flag'' ' flag ''']);
end
end

```

Рис 1. «Шаблон» функции odefile

Теперь следует пояснить каждый из пронумерованных шагов создания собственного описания задачи:

1. odefile должен принять векторы t и y от «решателя» ДУ и должен вернуть вектор – столбец такой же длины, как и y . Дополнительный аргумент ввода $flag$ определяет тип результата (весовая матрица, Якобиан, и т.д.), который возвращается из odefile;
2. «Решатели» циклически вызывают odefile для вычисления ДУ (или системы ДУ) различное количество раз. Необходимое условие – ДУ

- (или система ДУ), которая должна быть решена, должна быть определена;
3. Конструкция `switch` определяет тип требуемого результата на выходе так, чтобы `odefile` мог передать соответствующую информацию «решателю» (см. шаги 4 – 9);
 4. По умолчанию, начальные условия (промежуток времени, начальные условия, дополнительные параметры) передаются из `odefile` в функцию – «решатель». Если отменить это, то все указанные выше параметры следует задавать из командной строки;
 5. В блоке “`Jacobian`” `odefile` возвращает якобиан в «решатель». Этот параметр используется только в том случае, когда нужно улучшить результаты «жестких» функций – «решателей» `ode15s` и `ode23s`;
 6. В блоке “`jpattern`” `odefile` возвращает в «решатель» «разреженный образцовый якобиан». Этот блок следует использовать только в том случае, когда требуется, чтобы разреженные якобианы были сгенерированы численно для «жестких» «решателей»;
 7. В блоке “`mass`” `odefile` возвращает весовую матрицу в «решатель». Этот блок должен существовать только тогда, когда необходимо решать ДУ (или систему ДУ) в виде $M\dot{y} = F(t, y)$ или $M(t)\dot{y} = F(t, y)$;
 8. В блоке “`events`” `odefile` возвращает в «решатель» значения, которые нужны ему для вычисления каждой точки. Когда свойство `Events` установлено в 1, «решатели» проверяют каждый элемент вектора `event` на смещения к нулю, от нуля или прохождение через нуль. Если соответствующий элемент логического вектора `isterminal` установлен в 1, интегрирование будет прервано, когда определен переход через нуль. Элементы вектора `direction` – это числа -1, 1 или 0, определяющие, что соответствующее событие должно быть уменьшающим, увеличивающим, или что должен быть определен любой переход;
 9. Неопознанный флаг генерирует сообщение об ошибке.

Пример: уравнение Ван дер Пола $\ddot{y}_1 - \mu(1 - \dot{y}_1^2)\dot{y}_1 + y_1 = 0$ эквивалентно

системе ДУ первого порядка
$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \mu(1 - y_1^2)y_2 - y_1 \end{cases}$$
. М – файл, определяющий данную систему, выглядит так (при $\mu = 1$):

```
function out1 = vdp1(t,y)
out1 = [y(2); (1-y(1)^2)*y(2) - y(1)];
```

Для того, чтобы решить уравнение Ван дер Пола на интервале времени $[0; 20]$ при начальных условиях (в момент времени $t = 0$) $y(1)=2$ и $y(2)=0$ и построить график решения, используем такие команды:

```
[t,y] = ode45('vdp1',[0 20],[2; 0]);
plot(t,y)
```

График решения показан на рис 2.

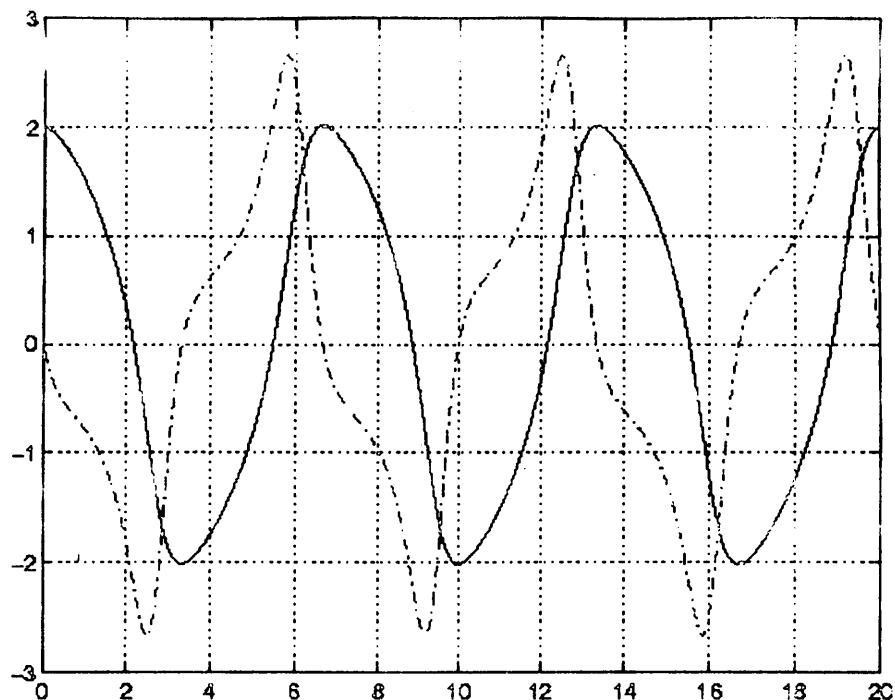


Рис 2. График решения уравнения Бах дер Поля (при заданных начальных условиях)

3.2 Функции для решения ДУ

В данном разделе будут рассмотрены именно те функции – «решатели», которые используются для собственно решения ДУ (и которые упоминались в предыдущем разделе).

Существует несколько форматов для вызова таких функций:

$[T, Y] = \text{solver}('F', tspan, y0)$

$[T, Y] = \text{solver}('F', tspan, y0, options)$

$[T, Y, TE, YE, IE] = \text{solver}('F', tspan, y0, options)$

$[T, X, Y] = \text{solver}('model', tspan, y0, options, ut, p1, p2, ...)$

Параметры этих форматов:

F	Имя оdefile, функции MATLAB® от t и y, возвращающей вектор – столбец. Все «решатели» могут решить системы в форме $y' = F(t, y)$. Оба «решателя» ode15s и ode23s могут решать уравнения в форме $My' = F(t, y)$. Только «решатель» ode15s может решать уравнения в форме $M(t)y' = F(t, y)$.
tspan	Вектор, определяющий интервал интегрирования в форме [t0 tfinal]. Для того, чтобы получить решения в конкретные моменты времени, следует использовать tspan = [t0,t1, ..., tfinal].
y0	Вектор начальных условий
options	Дополнительные аргументы интегрирования, созданные при помощи функции odeset.

p1, p2, ...	Дополнительные параметры, которые должны быть переданы в F.
T, Y	Матрица решения Y, в которой каждая строка соответствует времени, возвращенному в вектор – столбце T.

Рассмотрим теперь каждый из указанных выше форматов:

[T,Y] = *solver*('F',tspan,y0) при tspan=[t0 tfinal] интегрирует систему ДУ вида $y' = F(t,y)$ от точки t0 до точки tfinal с начальными условиями y0. 'F' – это строка, содержащая имя odefile. Функция F(t,y) должна возвращать вектор – столбец. Каждая строка в массиве решений y соответствует времени, возвращенному в вектор – столбце t. Для того, чтобы получить решения в отдельных точках времени t0, t1, ..., tfinal (все возрастающие или убывающие), следует использовать tspan = [t0 t1 ... tfinal].

[T,Y] = *solver*('F',tspan,y0,options) решает задачу также, как и в описанном выше формате, только параметры интегрирования «по умолчанию» заменяются значениями свойств, указанными в options – аргументе, созданном функцией *odeset* (описание этой функции см. ниже). Общепринятые свойства включают допустимую скалярную относительную ошибку RelTol (по умолчанию 1e-3) и вектор допустимых абсолютных ошибок AbsTol (по умолчанию все значения равны 1e-6).

[T,Y] = *solver*('F',tspan,y0,options,p1,p2,...) выполняется также, как описано выше, только дополнительные параметры p1, p2, ... передаются в М-файл с именем F при каждом его вызове. Если параметр options не используются, следует использовать пустую матрицу ("[]") на месте параметра options.

[T,Y,TE,YE,IE] = *solver*('F',tspan,y0,options) при установленом в "on" свойстве Event, выполняется так, как описано выше, регистрируя также переходы через нуль функции события, определенной в odefile. Odefile должен быть написан так, чтобы возвращать нужную информацию (см. описание средства создания odefile). Выходной вектор TE – это вектор – столбец моментов времени, в которые случились события, строки YE – это соответствующие решения (по отношению к вектору TE), а индексы в векторе IE определяют произошедшее событие.

Когда любой из форматов вызова «решателей» используется без выходных аргументов, вызванный «решатель» вызывает выходной аргумент «по умолчанию» - функцию *odeplot* для того, чтобы построить вычисленное решение. Дополнительный способ – это установить свойству OutputFcn значение "odeplot".

Теперь следует рассмотреть более подробно сами «решатели», их характеристики, свойства, алгоритмы.

Для систематизации описания воспользуемся такой таблицей:

«Решатель»	Тип проблемы	Порядок точности	В каких случаях используется
ode45	«Нежесткие»	Средний	В большинстве случаев.
ode23	«Нежесткие»	Низкий	При использовании гру-

			бых допустимых ошибок или решения умеренно «жестких» проблем.
ode113	«Нежесткие»	От низкого к высокому	При использовании строгих допустимых ошибок или решения ДУ с интенсивными вычислениями.
ode15s	«Жесткие»	От низкого к среднему	Если ode45 «работает» медленно («жесткие» системы) или присутствует матрица масс.
ode23s	«Жесткие»	Низкий	При использовании грубых допустимых ошибок для решения «жестких» систем, или при постоянной матрице масс.

Алгоритмы, используемые в функциях – «решателях», различаются в порядке точности и типе решаемых систем («жесткие» или «нежесткие»). Ниже в таблице приведены функции и те алгоритмы, которые они реализуют:

Функция	Реализуемый алгоритм
ode45	Явный алгоритм Рунге – Кутта (одноступенчатый «решатель»)
ode23	Явный алгоритм Рунге – Кутта (одноступенчатый «решатель»)
ode113	Алгоритм Адамса – Башфорта – Моултона переменного порядка (многоступенчатый «решатель»)
ode15s	Формулы численного дифференцирования переменного порядка (многоступенчатый «решатель»)
ode23s	Модифицированная функция Розенброка второго порядка (одноступенчатый «решатель»)

Более подробное описание всех алгоритмов, а также ссылки на литературу можно найти на странице `matlab\help\techdoc\ref\ode45.html` справочной системы MATLAB®.

3.3 Настройка параметра options функций решения ДУ

Различные функции – «решатели» принимают и используют различные параметры в списке `options`. Рассмотрим, какие параметры принимает тот или иной «решатель».

Параметры	ode45	ode23	ode113	ode115s	ode23s
RelTol, AbsTol	✓	✓	✓	✓	✓
OutputFcn, OutputSel, Refine, Stats	✓	✓	✓	✓	✓
Events	✓	✓	✓	✓	✓
MaxStep, InitialStep	✓	✓	✓	✓	✓
JConstant, Jacobian, JPATTERN, Vectorized	--	--	--	✓	✓
Mass, MassConstant	--	--	--	✓	✓
MaxOrder, BDF	--	--	--	✓	--

Для настройки передаваемых свойств следует использовать функцию odeset. Формат этой функции:

options = odeset('name1',value1,'name2',value2,...)

С помощью этой команды создается структура параметров интегратора, в которой указанные (именованные) свойства получают указанные значения. Функция odeset устанавливает все неуказанные (неопределенные) свойства в значение «пустая матрица» («[]»).

Более подробно описание всех вышеуказанных свойств дано на странице matlab\help\techdoc\ref\odeset.html встроенной системы помощи MATLAB®.

4. Примеры решения практических задач

Для того, чтобы изучить применение перечисленных выше функций, рассмотрим решение нескольких практических задач с использованием этих функций.

Пример 1: примером «нежесткой» системы является система уравнений, описывающая движение твердого тела без внешних сил:

$$\begin{cases} \dot{y}_1 = y_2 y_3 \\ \dot{y}_2 = -y_1 y_3 \\ \dot{y}_3 = -0.51 y_1 y_2 \end{cases} \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 1 \\ y_3(0) = 1 \end{cases}$$

Для того, чтобы смоделировать эту систему, следует создать функцию (M – файл) rigid, содержащую уравнения:

```
function dy = rigid(t,y)
dy = zeros(3,1); % a column vector (вектор – столбец)
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

В этом примере будут изменены допустимые ошибки (командой `odeset`) и система будет решена на интервале [0 12] с вектором начальных условий [0 1 1] в момент времени 0. Для этого используем команды:

```
options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5]);
[t,y] = ode45('rigid',[0 12],[0 1 1],options);
```

Строя график столбцов массива – результата \mathbf{Y} против T , можно увидеть решение (рис 3) :

```
plot(T,Y(:,1),'-',T,Y(:,2),'.',T,Y(:,3),':')
```

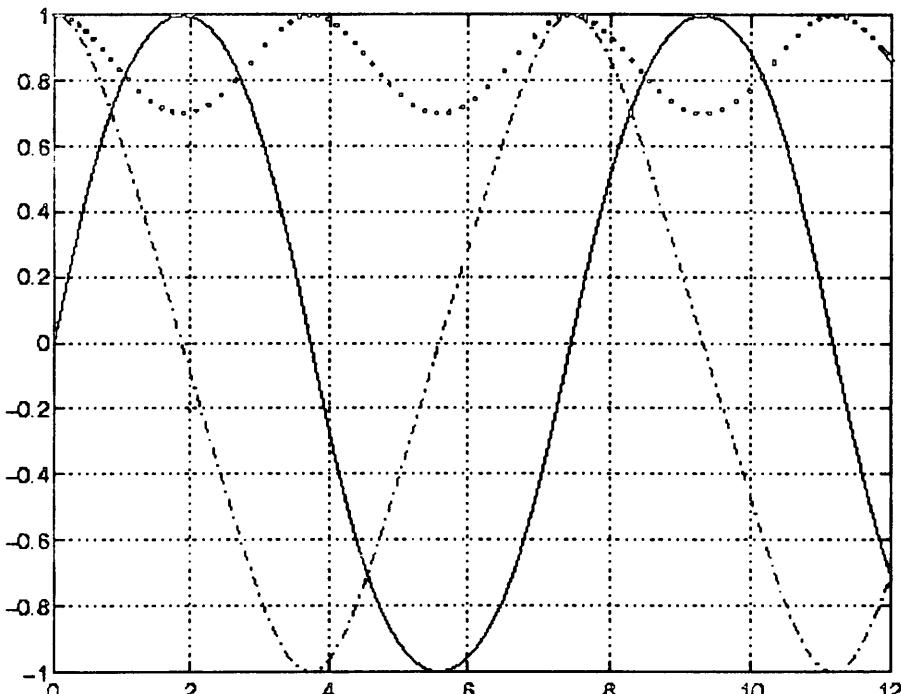


Рис 3 График решения системы ДУ из примера 1

Пример 2: примером «жесткой» системы является уже упоминавшееся уравнения Ван дер Пола, управляемые колебаниями релаксации. Предельный цикл имеет части, где компоненты решения изменяются медленно и проблема весьма «жесткая», чередующиеся с областями очень крутого изменения, где проблема уже «нежесткая».

Следует решить такую систему:
$$\begin{cases} y_1' = y_2 \\ y_2' = 1000(1 - y_1^2)y_2 - y_1 \end{cases} \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 1 \end{cases}$$

Для моделирования этой системы используем М – файл `vdp1000`, содержащий уравнения:

```
function dy = vdp1000(t,y)
dy = zeros(2,1); % a column vector
dy(1) = y(2);
dy(2) = 1000*(1 - y(1)^2)*y(2) - y(1);
```

Для решения этой задачи будут использованы допустимые относительные и абсолютные ошибки, установленные «по умолчанию». Интервал решения [0 3000], вектор начальных условий - [2 0] в момент времени $t = 0$.

Решение осуществляется командой

```
[T,Y] = ode15s('vdp1000',[0 3000],[2 0]);
```

Строя график первого столбца массива – результата Y против T , можно увидеть решение (рис 4) :

`plot(T,Y(:,1),'-o');`

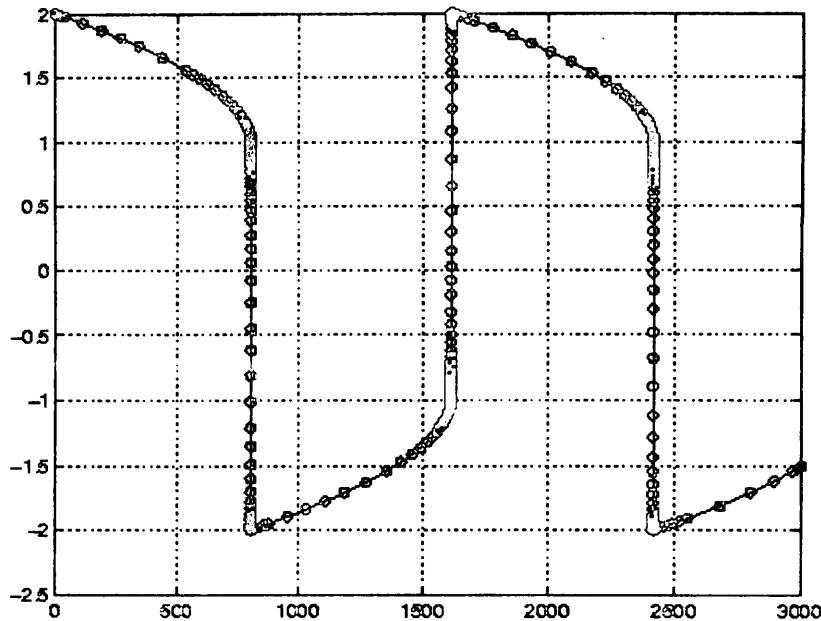


Рис 4. График решения системы ДУ из примера 2.

Определения и термины

Дифференциальное уравнение (ДУ) – уравнение, в котором неизвестная функция находится под знаком производной (или дифференциала).

Порядком ДУ называется наивысший порядок производной, входящей в уравнение.

Интегральная кривая - график решения обыкновенного ДУ.

Общее решение (или общий интеграл) – это решение ДУ, содержащее столько независимых произвольных постоянных (параметров), каков его порядок.

Частным решением ДУ называется всякое решение, которое может быть получено из общего при определенных числовых значениях произвольных постоянных, входящих в общее решение.

Контрольные вопросы

4. Укажите общий вид ДУ p – ого порядка.
5. Что называют общим и частным решениями ДУ?
6. Сформулируйте в общем виде задачу Коши.
7. Что значит «решить ДУ численно»?
8. Как классифицируются численные методы решения ДУ?
9. Какое средство MATLAB® помогает сформулировать задачу решения ДУ?
10. Какие функции существуют в MATLAB® для решения ДУ? Какие алгоритмы они реализуют?

Список рекомендуемой литературы

1. Н.И.Данилина, Н.С.Дубровская, О.П.Кваша, Г.Л.Смирнов,
Г.И.Феклисов Численные методы: учебник для техникумов – М.:
Высш. школа, 1976 – гл. VI;
2. Мак – Кракен Д., Дорн У. Численные методы и программирование
на ФОРТРАНе – М.: Мир, 1977.

Лекция 10: «Работа с полиномами. Интерполяция данных»

План лекции:

1. Описание функций MATLAB® для работы с полиномами;
2. Описание функций MATLAB® для интерполяции;

В данной лекции будут рассмотрены некоторые стандартные функции MATLAB®, которые применяются для выполнения операций над полиномами и матричного анализа. Данные операции не являются какой – то отдельной частью техники, однако зачастую являются промежуточным этапом при решении многих прикладных задач.

MATLAB®, являясь высокоуровневым языком для технических вычислений, предоставляет пользователям целый набор стандартных функций, которые и будут рассмотрены и описаны ниже.

1. Описание функций MATLAB® для работы с полиномами

Перед рассмотрением функций для действий над полиномами следует сказать о том, в каком виде MATLAB® хранит полиномы.

Пусть дан полином вида $a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$. Здесь *коэффициенты полинома* a_i – это произвольные действительные числа. MATLAB® хранит полиномы в виде векторов, содержащих именно эти коэффициенты. Таким образом, полином степени N представляется вектором коэффициентов длиной (N+1) элементов. Поэтому в описании всех функций в дальнейшем будет использоваться векторное описание полиномов.

Рассмотрим последовательно операции с полиномами, доступные в MATLAB®:

1. умножение полиномов – для выполнения этой операции используется функция $c = \text{conv}(a,b)$, где a и b – векторы длиной M и N элементов, содержащие коэффициенты полиномов – множителей. Результатом выполнения этой операции является вектор с длиной ($M*N - 1$) элементов, содержащий коэффициенты полинома – произведения.

Пример: пусть даны полиномы $p_1 = 2x^3 - 4x^2 + 0.56x - 1$ и $p_2 = -4x^2 + 7x - 2.6$. Найдем их произведение.

Векторы $a = [2 -4 0.56 -1]$ и $b = [-4 7 -2.6]$ описывают заданные полиномы. После исполнения команды $c = \text{conv}(a,b)$ вектор c содержит коэффициенты полинома – произведения: $c = [-8.0000 30.0000 -35.4400 18.3200 -8.4560 2.6000]$. Таким образом, искомый полином имеет вид $p_3 = -8x^5 + 30x^4 - 35.44x^3 + 18.32x^2 - 8.456x + 2.6$

2. деление полиномов – для выполнения этой операции используется функция $[q,r] = \text{deconv}(a,b)$ (параметры a и b – аналогично параметрам умножения). Эта функция возвращает два вектора q и r , которые являются частным от деления и остатком от деления, соответственно. При этом выполняется такое равенство: $a = \text{conv}(b,q)+r$, т.е. $a = b*q+r$.

Пример: разделить полиномы, описанные в предыдущем примере.

Выполняя команду $[q,r] = \text{deconv}(a,b)$, можно получить результаты: $q = [-0.5000 \quad 0.1250]$ и $r = [0 \quad 0 \quad -1.6150 \quad -0.6750]$.

3. определение производной полинома – для выполнения этой операции можно воспользоваться функцией `polyder`, которая имеет несколько форматов, которые показаны в таблице:

Формат	Действия, выполняемые функцией
$k = \text{polyder}(p)$	Возвращает производную полинома, описанного вектором p
$k = \text{polyder}(a,b)$	Возвращает производную от произведения полиномов, описанных векторами a и b
$[q,b] = \text{polyder}(b,a)$	Возвращает числитель q и знаменатель d производной от частного полиномов a/b .

4. построение полиномов по экспериментальным данным – эта задача может быть решена при помощи функции $p = \text{polyfit}(x,y,n)$, где x и y – это векторы экспериментальных данных длиной в k элементов, а число n – это степень того полинома, который должен аппроксимировать заданный набор данных. При вычислении коэффициентов результирующего полинома используется метод наименьших квадратов.

Пример: пусть векторы $x = [0.5 \quad 0.56 \quad 0.78 \quad 0.95 \quad 1.01 \quad 1.23]$ и $y = [45.78 \quad 46 \quad 46.23 \quad 47.23 \quad 47.99 \quad 48.11]$ описывают поведение какой – то величины. Аппроксимировать эту величину полиномом 6 – го порядка.

Сначала следует ввести векторы x и y в рабочую память среды MATLAB®. Исполнив команду $k = \text{polyfit}(x,y,6)$, получаем вектор из коэффициентов результирующего полинома: $k = [-184.4804 \quad 547.4283 \quad -83.1772 \quad 0 \quad 206.6074 \quad -97.2532 \quad 58.7287]$.

4. представление частного полиномов в виде суммы дробей, а также превращение суммы дробей в частное полиномов – эти взаимно обратные операции можно осуществить при помощи команды `residue`. Эта команда также имеет несколько форматов, которые представлены в таблице:

Формат	Действия, выполняемые функцией
$[r,p,k] = \text{residue}(b,a)$	Переводит частное полиномов b/a в сумму дробей и слагаемых.
$[b,a] = \text{residue}(r,p,k)$	Переводит полином из суммы дробей и слагаемых в два полинома числитель и знаменатель.

Следует подробнее сказать о представлении полиномов в виде суммы дробей и слагаемых.

Пусть частное полиномов a и b задано в таком виде:

$$\frac{b(x)}{a(x)} = \frac{b_1 + b_2 x^{-1} + b_3 x^{-2} + \cdots + b_{m+1} x^{-m}}{a_1 + a_2 x^{-1} + a_3 x^{-2} + \cdots + b_{n+1} x^{-n}}$$

С другой стороны, то же самое частное двух полиномов можно представить в таком виде:

$$\frac{b(x)}{a(x)} = \frac{r_1}{x - p_1} + \frac{r_2}{x - p_2} + \dots + \frac{r_n}{x - p_n} + k(x)$$

Указанное выше представление и есть представление в виде суммы дробей и слагаемых.

Число полюсов n определяется как $n = \text{length}(a) - 1$. Вектор коэффициентов прямого компонента $k(x)$ является пустым, если $\text{length}(a) < \text{length}(b)$. В противном случае $\text{length}(k) = \text{length}(b) - \text{length}(a)$.

В заключение приведем список аргументов этой функции и пояснения к каждому аргументу:

Аргумент	Назначение аргумента
b, a	Векторы, которые содержат коэффициенты полиномов
r	Вектор – столбец остатков
p	Вектор – столбец полюсов
k	Вектор – строка прямых слагаемых

5. получение полинома с заданными корнями – эта задача может быть решена при помощи функции $r = \text{poly}(a)$, где r – коэффициенты искомого полинома, a – вектор, содержащий корни полинома.

4. определение корней полинома – эта задача решается при помощи функции $r = \text{roots}(c)$. В векторе r содержатся корни полинома, коэффициенты которого заданы в векторе c .

Пример: пусть задан полином $x^2 - 2x + 1 = (x - 1)^2$. Определить его корни.

Для начала надо занести коэффициенты этого полинома в рабочую память MATLAB®. Для этого используется вектор $a = [1 -2 1]$. После исполнения команды $r = \text{roots}(a)$ вектор r содержит искомые корни заданного полинома:

$$\begin{matrix} r \\ 1 \\ 1 \end{matrix}$$

Как видно, вектор содержит два одинаковых корня, что и следовало ожидать.

2. Описание функций MATLAB® для интерполяции

Задача интерполяции данных часто возникает при обработке экспериментальных данных в реальных приложениях. MATLAB® для интерполяции данных предлагает такие функции:

Название функции	Формат функции	Описание функции
Одномерная интерполяция (табличный подход)	$y_i = \text{interp1}(x, Y, x_i)$	Возвращает вектор y_i , содержащий элементы, соответствующие элементам x_i и определенные интерполяцией в пределах векторов x и Y . Вектор x определяет точки, в которых даны значения вектора Y . Если Y – матрица, тогда интерполяция выполняется для каждого столбца Y и размер y_i будет $[\text{length}(x_i) \times \langle\text{количество столбцов в } Y\rangle]$

Двумерная интерполяция (табличный подход)	$ZI=interp2(X,Y,Z,Xi, Yi)$	Возвращает матрицу ZI , содержащую элементы, соответствующие элементам Xi и Yi , и определенные интерполяцией в пределах двумерной функции, описанной матрицами X , Y и Z . X и Y должны быть монотонными и иметь одинаковый формат («сетка») (такой, как возвращает функция $meshgrid$). Матрицы X и Y определяют точки, в которых дана информация матрицы Z . Эта функция предполагает, что $X = 1 : n$ и $Y = 1 : m$, где $[m,n] = size(Z)$
Интерполяция кубическими сплайнами.	$yi = spline(x,y,xi)$	Функция использует векторы x и y , которые содержат грубо размещенные данные, и вектор xi , который содержит новые, более точно размещенные, координаты (ось абсцисс), для интерполяции данных в новых точках. При интерполяции используются кубические сплайны.

Отдельно следует сказать о функциях, которые используются для создания массивов и матриц, которые в дальнейшем могут использоваться в функциях трехмерной графики и интерполяции многомерных функций.

К таким функциям относятся следующие функции:

Название функции	Формат функции	Описание функции
Создание матриц X и Y для трехмерной графики	$[X, Y]=meshgrid(x, y)$ $[X, Y]=meshgrid(x)$ $[X, Y, Z]=meshgrid(x, y, z)$	<p>Функция преобразовывает область, определенную векторами x и y, в массивы X и Y, которые могут быть использованы для вычисления функции двух переменных и трехмерной графики. Строки выходного массива X – это копии вектора x; столбцы выходного массива Y – это копии вектора y.</p> <p>Данная функция использует для построения только вектор x (т.е. $[X, Y]=meshgrid(x, x)$)</p> <p>Функция генерирует трехмерные массивы, используемые для вычисления функций трех переменных и трехмерного объемного рисования.</p>
Генерация	$[X1, X2, X3, \dots] =$	Функция преобразовывает про-

матриц для многомерных функций и интерполяций	<code>ndgrid(x1,x2,x3,...)</code>	пространство, определенное векторами x_1, x_2, x_3, \dots , в массивы X_1, X_2, X_3, \dots , которые могут быть использованы для вычисления функций многих переменных и многомерного интерполирования. i –ое измерение выходного массива X_i есть копии элементов вектора x_i .
---	-----------------------------------	---

Пример: построить поверхность, описываемую функцией

$$z = \sqrt{\frac{0.91x^2 + 1.45y^2}{4.51}}$$

(конус). Переменная $x \in [-5;5]$, переменная $y \in [-7;8]$.

Решение: для построения поверхности нужно создать массивы входных переменных данной функции. Это можно сделать с помощью команды $[X, Y] = meshgrid(x, y)$, где x и y – это векторы, содержащие значения переменных в заданных интервалах (шаг в обоих случаях равен 0.1). Теперь в рабочей памяти MATLAB® находятся два массива X и Y , элементы которых теперь можно использовать для вычисления z – координат требуемой поверхности. Для вычислений следует использовать указанную в задании формулу.

После того, как массив (матрица) z – координат вычислена, можно с помощью команды `mesh` построить требуемую поверхность (в качестве координатных осей выбраны векторы x и y). Построенная поверхность показана на рис. 1.

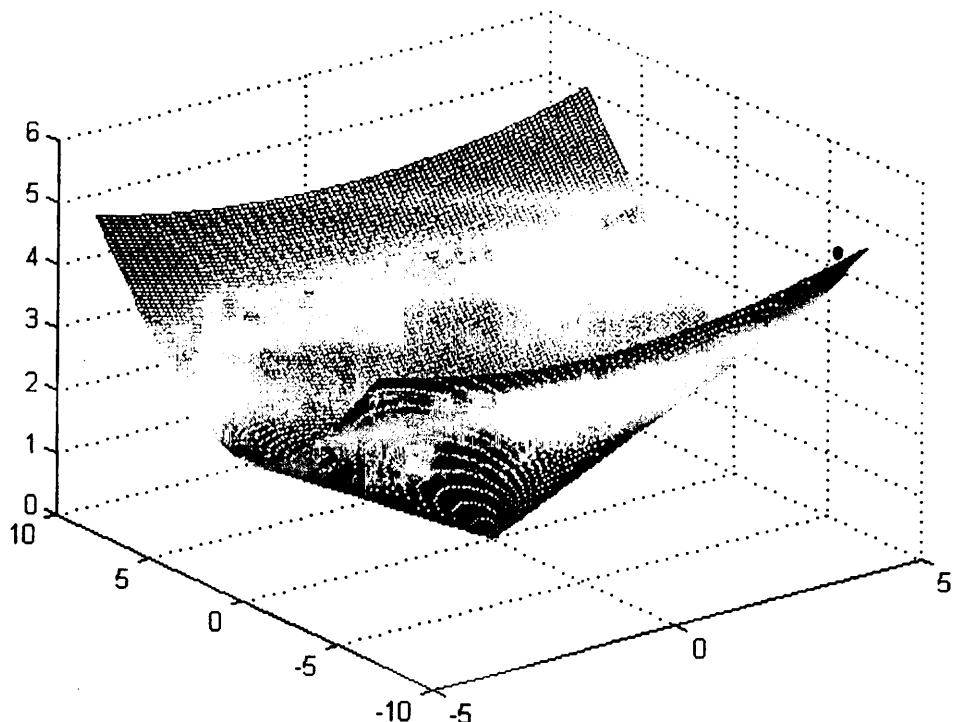


Рис 1 Пример поверхности (конус).

Контрольные вопросы

1. Укажите общий вид полинома n -ой степени. Что называется коэффициентами полинома?
2. Перечислите операции с полиномами.
3. Какие функции выполняют в MATLAB® действия с полиномами?
4. Что такое интерполяция?
5. Какие функции MATLAB® решают задачи интерполяции? Приведите примеры.