

**НАЦИОНАЛЬНЫЙ АВИАЦИОННЫЙ УНИВЕРСИТЕТ.
КАФЕДРА АЭРОКОСМИЧЕСКИХ СИСТЕМ УПРАВЛЕНИЯ**



**Воронов С.И. ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО МАТЛАВ
ДЛЯ СТУДЕНТОВ ПЕРВОГО КУРСА.**
Учебное пособие
(СЕРИЯ - МЕТОДЫ И АЛГОРИТМЫ В ИНЖЕНЕРНЫХ ЗАДАЧАХ)

Киев 2018г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	6
Аннотация к тематической структуре практикума.	7
1. НАЧАЛЬНОЕ ЗНАКОМСТВО С MATLAB. (ЛАБ. РАБОТА 1)	10
1.1. ФОРМА БЭКУСА — НАУРА.	10
1.1.1. Термины БНФ.....	10
1.1.2. Специальный нетерминал <Пусто>.....	10
1.1.3. Оператор определения.....	10
1.1.4. Основное правило БНФ.	10
1.1.5. Оператор выбора.....	10
1.1.6. Рекурсивное правило БНФ.....	11
1.1.7. Специальные расширения стандарта БНФ.....	11
1.2. ЗНАКОМСТВО С ПРЕДСТАВЛЕНИЕМ ДАННЫХ В СИСТЕМЕ MATLAB.	12
1.2.1. Типы арифметических чисел в MATLAB.....	12
1.2.2. Знакомство с комментариями, константами и переменными, с простыми арифметическими операциями.	13
1.2.3. Выражение. Простое арифметическое выражение.	14
1.2.4. Синтаксис простого арифметического выражения.....	15
1.2.5. Примеры арифметических выражений.....	17
1.3. САМОСТОЯТЕЛЬНАЯ РАБОТА.....	18
2. MATLAB – ЭТО МАТРИЦЫ (ЛАБ. РАБОТА 2).....	20
2.1. СОЗДАНИЕ МАТРИЦ И МАНИПУЛЯЦИЯ ИХ ЭЛЕМЕНТАМИ.....	20
2.1.1. Определение одномерных матриц (векторов).....	20
2.1.2. Примеры создания одномерных матриц (векторов).....	21
2.1.3. Определение прямоугольных матриц (2D-матриц).....	23
2.1.4. Примеры создания прямоугольных матриц (2D-матриц).	23
2.1.5. Правила доступа к структурам внутри матриц.....	24
2.1.6. Примеры доступа к структурам внутри матриц.....	25
2.1.7. Матрицы MATLAB это динамические структуры.....	25
2.1.8. Примеры манипуляции строками и столбцами или субматрицами в 2D-матрицах.	26
2.1.9. Пример динамического понижения размеров и размерности исходной 2D-матрицы.....	27
2.2. МАТРИЦЫ С РАЗМЕРНОСТЬЮ БОЛЬШЕ ЧЕМ 2D.....	28
2.2.1. Пример создания матриц 3D.....	28
2.2.2. Пример манипуляции значениями и размерностью матриц 3D.....	29
2.2.3. Несколько слов о матрицах размерностью 4D и выше.....	30
2.2.4. Сервисные функции MATLAB для работы с матрицами.....	31
2.3. ОПЕРАЦИИ С МАТРИЦАМИ В MATLAB.....	32
2.3.1. Операция транспонирования.....	32
2.3.2. Поэлементные операции для арифметических матриц.....	33
2.3.3. Одноместные поэлементные операции для арифметических матриц любой размерности.....	33
2.3.4. Двухместные поэлементные операции для арифметических nD-матриц и простого арифметического операнда (скаляра).	34
2.3.5. Двухместные поэлементные операции для арифметических nD-матриц одинаковой размерности и размера.	36
2.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.....	37

3. ПОСТРОЕНИЕ ГРАФИКОВ (ЛАБ. РАБОТА 3)	38
3.1. PLOT - ПОСТРОЕНИЕ ГРАФИКОВ, ЗАДАННЫХ МАТРИЦАМИ.....	39
3.1.1. Функция PLOT. Синтаксис.....	39
3.1.2. Функция PLOT. Примеры.....	40
3.2. FPLOТ - ПОСТРОЕНИЕ ГРАФИКОВ, ЗАДАННЫХ АНАЛИТИЧЕСКИ.....	42
3.2.1. Функция FPLOТ. Синтаксис.....	42
3.2.2. Функция FPLOТ. Примеры.....	43
3.3. ФУНКЦИЯ AXIS.....	44
3.3.1. Функция AXIS. Синтаксис.....	44
3.3.2. Функция AXIS. Примеры.....	45
3.4. EZPLOТ. ПРОСТОЙ В ИСПОЛЬЗОВАНИИ ПЛОТТЕР.....	45
3.4.1. Функция EZplot. Синтаксис:.....	45
3.4.2. Функция EZplot. Описание.....	46
3.4.3. Функция EZplot. Примеры.....	46
3.5. EZPOLAR. ПРОСТОЙ В ИСПОЛЬЗОВАНИИ ПОЛЯРНЫЙ КООРДИНАТНЫЙ ПЛОТТЕР.....	47
3.5.1. Функция EZPOLAR. Синтаксис.....	47
3.5.2. Функция EZPOLAR. Описание.....	47
3.5.3. Функция EZPOLAR. Примеры.....	47
3.6. САМОСТОЯТЕЛЬНАЯ РАБОТА.....	48
4. УСЛОВНЫЕ ВЫРАЖЕНИЯ, ОПЕРАТОРЫ ВЕТВЛЕНИЯ И ЦИКЛЫ (ЛАБ. РАБОТА 4)	49
4.1. УСЛОВНЫЕ ВЫРАЖЕНИЯ (CONDITIONAL EXPRESSION).....	49
4.1.1. Операторы отношений (Relational operations).....	49
4.1.2. Примеры операторов отношений.....	49
4.1.3. Вычисление отношений алгоритмическим путем.....	51
4.1.4. Логические операторы и операции (Logical operations).....	52
4.1.5. Некоторые законы для логических операций.....	56
4.2. УСЛОВНЫЕ ОПЕРАТОРЫ (ИНСТРУКЦИИ).....	57
4.2.1. Простой условный оператор if.....	57
4.2.2. Селектор по различным условиям if..elseif.....	58
4.2.3. Селектор по различным значениям.....	60
4.3. Циклы.....	60
4.3.1. Цикл FOR.....	60
4.3.2. Цикл FOR. Примеры.....	63
4.3.3. Цикл WHILE.....	64
4.3.4. Цикл WHILE. Примеры.....	66
4.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.....	67
5. СКРИПТЫ И ФУНКЦИИ (ЛАБ. РАБОТА 5)	68
5.1. ВВЕДЕНИЕ.....	68
5.2. СКРИПТЫ.....	68
5.2.1. Правило оформления help для скриптов:.....	69
5.3. ФУНКЦИИ.....	70
5.3.1. Синтаксис функций.....	71
5.3.2. Глобальные и локальные переменные.....	72
5.3.3. Вызов функций.....	73
5.3.4. Основные правила работы с функциями.....	74
5.3.5. Функции. Примеры.....	74
5.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.....	77
6. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ. (ЛАБ. РАБОТА 6)	78

6.1. РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ КРАМЕРА.....	78
6.1.1. Основная идея метода.....	78
6.2. ТЕОРЕМА КРОНЕКЕРА-КАПЕЛЛИ.....	79
6.3. РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ГАУССА.	80
6.3.1. Основная идея метода.....	80
6.3.2. Метод исключения произвольной переменной из указанного уравнения в системе линейных уравнений.....	81
6.3.3. Метод построения треугольной матрицы путем последовательного исключения переменных.....	83
6.4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ. ПРИМЕРЫ	84
6.4.1. Реализация метода Крамера	84
6.4.2. Решение систем линейных уравнений средствами MATLAB. Примеры	86
6.4.3. Реализация метода Гаусса. Примеры	86
6.4.4. Сравнительные тесты методов решения SLE	88
6.5. САМОСТОЯТЕЛЬНАЯ РАБОТА.	89
7. АППРОКСИМАЦИЯ ТАБЛИЧНО ЗАДАННОЙ ФУНКЦИИ. (ЛАБ. РАБОТА 7). ...	91
7.1. Постановка задачи.....	91
7.2. ЗАДАЧА АППРОКСИМАЦИИ В ДИСКРЕТНОЙ ФОРМЕ	92
7.2.1. Построение условий для решения задачи аппроксимации в дискретной форме.	92
7.2.2. Решение задачи аппроксимации в дискретной форме	94
7.3. РЕАЛИЗАЦИИ МЕТОДОВ НАИМЕНЬШИХ КВАДРАТОВ.....	95
7.3.1. Метод наименьших квадратов для базисных функций x^n	95
7.3.2. Метод наименьших квадратов для произвольных базисных функций.....	97
7.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.	99
СПИСОК ЛИТЕРАТУРЫ	99
ПРИЛОЖЕНИЯ	101
ПРИЛОЖЕНИЕ А. Задача стыковки челнока и станции.....	101
Постановка задачи.....	101
Математический анализ задачи.....	101
Исходные данные и задание.....	103
Содержание работы	104
ПРИЛОЖЕНИЕ Б. 3D-ГРАФИКИ В MATLAB.....	105
Постановка задачи.....	105
1. Функция <i>meshgrid</i>	105
2. Функции <i>mesh</i> , <i>meshc</i> , <i>meshz</i>	106
3. Функция <i>colormap</i>	108
4. Функция <i>shading</i>	110
5. Функция <i>view</i>	110
6. Функция <i>surf</i>	111
Содержание работы	115
ПРИЛОЖЕНИЕ В. ЛОГИСТИЧЕСКАЯ ЗАДАЧА НА ПОВЕРХНОСТИ АСТЕРОИДОВ	116
Введение. Оценка перспективности и актуальности задачи.....	116
Постановка задачи и содержание работы.....	116
Анализ задачи о движении тела с постоянным ускорением.....	118
Анализ задачи о вертикальном движении тела в гравитационном поле	119
Вариант решения №1.....	123
Вариант решения №2.....	123

<i>Решение задачи о вертикальном движении тела в гравитационном поле в среде MATLAB.....</i>	<i>125</i>
<i>Задача о движении тела в гравитационном поле, брошенного под углом к горизонту.....</i>	<i>127</i>
<i>Движение тела брошенного под углом к горизонту при постоянном ускорении свободного падения.....</i>	<i>127</i>
<i>Движение тела брошенного под углом к горизонту с ускорением свободного падения, зависящим от высоты над поверхностью.....</i>	<i>129</i>
<i>Окончательное решение задачи в среде MATLAB.....</i>	<i>131</i>

ВВЕДЕНИЕ

Те, кто много лет занимается программированием, давно заметили, что наилучшим учителем программирования является среда написания, компиляции или интерпретации программных текстов, а также подсистемы HELP в этой среде. Второй составляющей успеха освоения нового языка или технологии программирования, являются опыт реализации в этой среде алгоритмов, которые наиболее часто используются и являются характерными для выбранной прикладной области. Настоящий практикум предлагает именно такой подход к начальному освоению программного инструментария в составе математической системы MATLAB.

Основная особенность программного инструментария в составе системы MATLAB обуславливается ее предназначением быть инструментом для исследования объектов и процессов различными математическими средствами. Очень часто, процесс такого исследования не может предварительно быть описан некоторым законченным алгоритмом. Действительно, промежуточные результаты процесса исследования могут породить новые гипотезы, которые первоначально либо еще не рассматривались, либо были отнесены к второстепенным решениям. Система MATLAB, предлагая в качестве основного режима работы режим интерпретации, то есть, выполнения математических и программных инструкций по факту их ввода в систему, позволяет гибко выстраивать или даже менять алгоритм решения поставленной задачи. В качестве итогового результата таких исследований можно рассматривать как формализованное математическое описание некоторого объекта или процесса, так и законченный алгоритм решения некоторой задачи. Кроме того, такой законченный алгоритм уже можно рассматривать как основу для разработки и компиляции прикладного программного обеспечения, ориентированного на конечного (как правило, менее математически квалифицированного) пользователя.

Алгоритмы в составе практикума

Исследование некоторого объекта или процесса, как правило, начинается с измерения его различных свойств. Измерение, как процесс получения числа из меры, позволяет представить объект или процесс как некоторую табличную функцию. Такая функция связывает воздействие на объект или процесс в виде аргумента функции и реакцию объекта или процесса как значение функции. Однако первичные математические сведения об объекте или процессе в виде числовых табличных функций, явно недостаточны для дальнейшего математического анализа и требуют нахождения аналитических эквивалентов для таких табличных функций. Аналитическая форма полученных функций, естественным образом открывает возможности для дальнейшего анализа использование таких разделов математики как алгебра, высшая математика и так далее.

Учитывая, что настоящий практикум ориентирован на студентов первого года обучения, а следовательно и соответствующую базу знаний студентов, ограничимся следующим списком простейших алгоритмов из раздела численных методов математики:

- Сумма элементов табличной функции;
- Среднее значение элементов табличной функции;
- Произведение табличной элементов табличной функции;
- Вычисление целочисленной степени числа;
- Вычисление факториала числа;
- Вычисление степенных полиномов;
- Нахождение действительного корня алгебраического уравнения методом бисекции (дихотомии);
- Вычисление элементарных функций степенными рядами (рядами Тейлора, Маклорена). Более детально смотрите источник [1];

- Вычисление интеграла от табличной функции квадратурами прямоугольников, трапеций и квадратурами Симпсона. Более детально смотрите источник [2];
- Поиск минимального и максимального значений табличной функции;
- Простейшая (пузырьковая сортировка) табличных функций;
- Решение систем линейных уравнений методами Крамера и Гаусса;
- Решение простейших задач интерполяции и экстраполяции;
- Нахождение аналитического вида функции методами наименьших квадратов. Более детально смотрите источник [3].

Дальнейший математический анализ объектов или процессов требует специальных прикладных знаний, которые студенты получают на старших курсах. По этой причине, такие возможности системы MATLAB как SIMULINK и различные TOOLBOX вынесены за скобки данного практикума и переадресованы в соответствующие учебные дисциплины на старших курсах.

В заключение отметим, что для самостоятельного освоения или расширения знаний по системе MATLAB, рекомендуются источники [4], [5], [6], [7], а для начального освоения системы с использованием литературы на английском языке [8]. Для изучения связей системы MATLAB с другими языками программирования, рекомендуется источник [8]

Аннотация к тематической структуре практикума.

Тема 1. Начальное знакомство с MATLAB. (Лаб. Работа 1)

Данная тема включает знакомство со средой управления системы MATLAB (главное меню системы, окна Command Window, Workspace, Command History), а также быстрые кнопки. Такое знакомство выполняется в режиме локальной видеоконференции и не входит в печатный материал практикума.

Поскольку большинство студентов профессионально осваивают программирование в первый раз, в данной теме предлагается начальное знакомство с основами формализации синтаксиса языков программирования. В качестве такой формализации рассматриваются [формы Бэкуса — Наура](#), далее БНФ. Выполняется демонстрация на примерах работы MATLAB в режиме [интерпретатора](#) простейших арифметических выражений (режим калькулятора). Практическая работа по теме заключается в освоении скалярных типов данных и операций над ними включая:

- Первичные арифметические типы данных (целые и целые со знаком, действительные типы в их регулярном и экспоненциальном формате, комплексные типы);
- Арифметические операции над скалярами и соответствующие арифметические выражения.

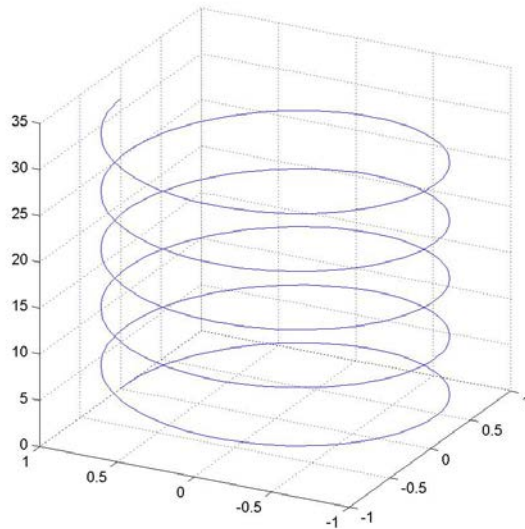
Итоговая работа по теме сформулирована в приложении (А) и выполняется студентами, как домашняя работа.

Тема 2. MATLAB – это матрицы. (Лаб. Работа 2)

В этой теме основное внимание уделяется технике создания и трансформации матриц. Рассматриваются различные виды конструкторов, таких как конструкторы столбцов и строк матриц, конструкторы диапазонов значений и конструкторы индексов. Приводятся примеры повышения и понижения размеров и размерностей матриц. На первый взгляд, это довольно скучная информация. Однако хорошее понимание структуры данных и логики доступа к ним, это залог минимизации ошибок и оптимальности алгоритмов.

Тема 3. Построение графиков. (Лаб. Работа 3)

Обычно эта тема и соответствующая ей лабораторная работа наиболее любима студентами. Действительно, ну кто не любит красивые картинки? Однако, суть ее в ином – это отображение функций заданных таблично или аналитически в наглядной форме. Например:



Такая замечательная картинка получается при использовании пакета инструкций следующего содержания:

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

В то же время, гораздо чаще необходимо строить графики для функций одной переменной. Освоению техники построения таких графиков и посвящается данная лабораторная работа. В ней вы познакомитесь с такими функциями как: plot, ezplot, fplot, polar, ezpolar. а так же некоторыми вспомогательными функциями.

Тема 4. Условные выражения, инструкции ветвления и циклов. (Лаб. Работа 4)

В теме 4 рассматриваются логические типы данных и логические выражения, которые в дальнейшем используются как условия для операторов ветвления и циклов. Определяются синтаксис, а так же блок – алгоритмические схемы операторов ветвления и циклов.

В качестве примеров предлагаются исходные тексты программ для реализации отдельных простых функций MATLAB, а также программы простейших примеров из раздела численных методов.

Например:

```
% Вычисление интеграла табличной функции f,
% квадратурами трапеций.заданной прис равномерном
% шаге step по оси X.
xb=0;
xe=1;
step=0.01;
f = [xb:step:xe];
k=1;
y=0;
while k <= length(f)-1
    y=y+(f(k)+f(k+1))/2;
```



```
k=k+1;  
end;  
y=y*step
```

Тема 5. Скрипты и функции. (Лаб. Работа 5)

Выполняя предшествующие работы, мы обычно использовали некоторый стандартный редактор текстов в Windows и последовательности инструкций для MATLAB переносили в окно Command Window путем копирования и вставки. Такая техника организации вычислительного процесса подчеркивала интерпретирующий характер системы MATLAB и позволяла гибко менять алгоритм выполнения задачи в процессе ее решения.

Фактически мы вручную выполняли инструкции или пакеты инструкций, которые можно выполнять с помощью обращения к имени файла, содержащего исходный текст таких пакетов. В этом заключается основная идея создания и применения скриптов.

В данной теме рассматриваются достоинства и недостатки такой техники программирования, а устранение недостатков логически обосновывает мощный инструмент практически всех языков программирования – функции.

В теме также частично формализуется синтаксис оформления последовательности инструкций в виде функции, а так же правила и синтаксис использования функций. В качестве примеров приведены исходные тексты функций пузырьковой сортировки вектора и вычисления функции $\sin(x)$ рядом Маклорена.

Тема 6. Решение систем линейных уравнений. (Лаб. Работа 6)

Решение систем линейных уравнений является одной из самых востребованных задач для анализа и создания систем управления. В этом смысле, тема и лабораторная работа 6 является важным практикумом для студентов технических специальностей. В работе рассмотрены минимальные теоретические сведения и исходные тексты соответствующих функций для решения систем линейных уравнений наиболее популярными методами, то есть:

- Решение системы линейных уравнений методом Крамера;
- Решение системы линейных уравнений методом Гаусса.

Кроме того, для решения систем линейных уравнений показано применение встроенных в MATLAB матричных операций на базе методов линейной алгебры.

Тема 7. Аппроксимация таблично заданной функции. (Лаб. работа 7)

Тема 7 посвящена преобразованию табличных функций к аналитическому виду.

Табличные функции являются, как правило, результатом выполнения разного рода экспериментов или измерений. Наиболее частым случаем табличной функции является таблица соответствий вида $Y_i = F(X_i)$ полученная при разных индексах (i). Очевидно, что подобные таблицы весьма неудобны для дальнейшего, особенно математического анализа. Таким образом, одной из важнейших задач во множестве технических приложений, является задача замены табличной функции аналитическим эквивалентом $y = f(x)$.

Традиционно такая задача решается как задача [аппроксимации](#), при которой в качестве аналитической функции рассматривается степенной ряд, коэффициенты которого определяются [методом наименьших квадратов](#).

Однако степенной ряд не всегда является наилучшей формой, которая используется для аппроксимации. В лабораторной работе 7, в качестве аппроксимирующего ряда показано применение суммы произвольных функций, для которых методом наименьших квадратов определяются масштабные коэффициенты.

1. НАЧАЛЬНОЕ ЗНАКОМСТВО С MATLAB. (ЛАБ. РАБОТА 1)

1.1. Форма бэкуса — наура.

Форма Бэкуса — Наура (сокращенно БНФ). Это формальная система описания синтаксиса, в которой одни синтаксические элементы формального языка последовательно определяются через другие элементы.

1.1.1. Термины БНФ

- *Терминал* (терминальный символ или обобщение понятия «буквы») — объект, непосредственно присутствующий в словах языка, соответствующего грамматике и имеющий конкретное, неизменяемое значение. В формальных компьютерных языках, в качестве терминалов обычно берут стандартные символы ASCII — малые и заглавные латинские буквы, цифры, а также специальные символы.
- *Нетерминал* (нетерминальный символ). Это объект обозначающий какую-либо сущность языка (например, это инструкция, формула, арифметическое выражение, и т.д.). Нетерминал не имеет конкретного символьного значения, однако обозначает некоторый смысл. Все нетерминалы записываются с помощью текста раскрывающего такой смысл и заключаются в треугольные скобки. Например:

<Натуральное число>

1.1.2. Специальный нетерминал <Пусто>

Используется для обозначения отсутствия всякого символьного значения.

1.1.3. Оператор определения.

Используется для присвоения значения терминалу или нетерминалу. Записывается как последовательность следующих терминалов:

`::=`

1.1.4. Основное правило БНФ.

Основное правило предполагает, что нетерминал, смысл которого определяется записывается слева от оператора определения, а справа записываются терминалы или нетерминалы смысл которых уже определен или будет определен далее. Например:

<Знак умножения> ::= *

1.1.5. Оператор выбора.

Используется как способ указать при определении нетерминала выбор только одного нетерминала или терминала из множества в правой части определения левой части определения.

Например:

<Цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

1.1.6. Рекурсивное правило БНФ.

Определение нетерминалов БНФ позволяет определяемый нетерминал записывать как в левой, так и правой части операции определения. Такое правило является рекурсивным и всегда предполагает, что в нем содержится некоторое начальное значение определяемого нетерминала. Например:

$$\begin{aligned} \langle \text{Цифра} \rangle & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \langle \text{Число} \rangle & ::= \langle \text{Цифра} \rangle \mid \langle \text{Число} \rangle \langle \text{Цифра} \rangle \end{aligned}$$

В данном примере часть правила $\langle \text{Цифра} \rangle$ задает начальное значение для рекурсии, а часть $\langle \text{Число} \rangle \langle \text{Цифра} \rangle$ показывает правило выполнения самой рекурсии. Наглядно, это можно представить следующими шагами:

Шаг	Часть правила, которая работает	$\langle \text{Цифра} \rangle$	$\langle \text{Число} \rangle$
1	$\langle \text{Цифра} \rangle$	5	5
2	$\langle \text{Число} \rangle \langle \text{Цифра} \rangle$	2	52
3	$\langle \text{Число} \rangle \langle \text{Цифра} \rangle$	7	527
4	$\langle \text{Число} \rangle \langle \text{Цифра} \rangle$	1	5271
5	$\langle \text{Число} \rangle \langle \text{Цифра} \rangle$	8	52718

Продолжая шаги с помощью части правила $\langle \text{Число} \rangle \langle \text{Цифра} \rangle$ можно сформировать цепочку изображающую натуральное число любой разрядности.

1.1.7. Специальные расширения стандарта БНФ

Помимо классической БНФ в научной литературе также используется расширенная форма. Эта форма вводит некоторые новые правила и обозначения.

В рамках данного практикума мы позаимствуем из расширенной формы следующее:

1.1.7.1. Условное вхождение.

Условное вхождение подразумевает возможность использовать или пропускать некоторую часть правила и записывается с помощью квадратных скобок. Квадратные скобки выделяют необязательный элемент правила, который может присутствовать, а может и отсутствовать. Например, правило вида $\langle A \rangle ::= \langle \text{Пусто} \rangle \mid [B]$ обозначает, что нетерминал A либо является пустым, либо состоит из одного символа B .

1.1.7.2. Специальное обозначение терминальных символов.

Поскольку для условного вхождения возникла необходимость в специальном смысле квадратных скобок, то терминальный вид квадратных скобок следует определить другим способом. Для такого определения можно воспользуемся следующей нетерминальной формой:

$$\begin{aligned} \langle ' [' \rangle \\ \langle '] ' \rangle \end{aligned}$$

Аналогичным образом можно определить терминальные символы для треугольных скобок:

```
<'<'>  
<'>'>
```

Кроме того, символ пробела, который легко теряется при использовании в обычной терминальной форме, будет удобно определить в виде:

```
<' '>
```

1.2. Знакомство с представлением данных в системе MATLAB.

1.2.1. Типы арифметических чисел в MATLAB.

1.2.1.1. Арифметическое число.

```
<Арифметическое число> ::= <Натуральное число>  
| <Целое со знаком>  
| <Действительное число>  
| <Комплексное число>
```

1.2.1.2. Натуральные числа.

```
<Натуральное число> ::= <Цифра> | <Натуральное число><Цифра>  
<Цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

1.2.1.3. Целые числа со знаком.

```
<Целое со знаком> ::= <integer>  
<integer> ::= [<Знак>]<Натуральное число>  
<Знак> ::= + | -
```

1.2.1.4. Действительные числа

```
<Действительное число> ::= <Real>  
<Real> ::= <Regular Real> | <Exponential Real>
```

1.2.1.5. Действительные числа в обычной форме.

```
<Regular Real> ::= <Целое со знаком> . <Число>
```

Например:

```
1.1  
1.38  
602.21412
```

1.2.1.6. Действительные числа в экспоненциальной форме.

```
<Exponential Real> ::= <Мантисса> e <Порядок>  
<Мантисса> ::= <Regular Real>  
<Порядок> ::= <Целое со знаком>
```

ВНИМАНИЕ! **e** <Порядок> это значение $10^{\text{Порядок}}$.

Например:

```
1,602176565e-19 == 1.602176565*10^-19 (это элементарный заряд)
1,380650424e-23 == 1.380650424*10^-23 (это Постоянная Больцмана)
6,02214129e23 == 6.02214129*10^23 (это число Авогадро)
```

1.2.1.7. Комплексные числа.

```
<Комплексное число> ::= [<Real> + ] <Real>i |
[<Real> + ] <Real>j
где: i^2 = -1; j^2 = -1;
```

Например:

```
1.1 + 6.2i
1.1 + 6.2j
```

1.2.1.8. Символы специальных значений:

```
<Символ специального значения> ::= [Знак]inf | NaN
```

Например:

```
inf      % infinity или бесконечность
NaN      % Not-a-Number или неопределенность
```

1.2.2. Знакомство с комментариями, константами и переменными, с простыми арифметическими операциями.

1.2.2.1. Комментарий.

Комментарий это произвольный текст, который не интерпретируется (игнорируется) парсером MatLab и может размещаться в виде отдельных строк или замыкать отдельные строки.

Формально комментарий можно определить так:

```
<Комментарий> ::= % '<' '><Произвольный текст>
```

Например:

```
% Это комментарий
```

1.2.2.2. Арифметические константы в MatLab.

Константы в MatLab, это самоопределенные терминалы. Другими словами это терминалы, тип и значение которых явно следует из их изображения.

Например:

```
-123      % Это числовое значение типа <целое со знаком>
```

Однако, кроме самоопределенных терминалов для обозначения констант используются также символические имена, за которыми закрепляются определенные численные значения. Таким образом, константы можно определить как:

```
<Арифметическая константа> ::=
    <Самоопределенный арифметический терминал>
    | <Имя арифметической константы>
```

```
<Имя арифметической константы> ::= pi | eps
    | realmin | realmax
    | i | j
```

где:

```
pi = значение числа ПИ
eps = 2^(-52) или 2.22e-16
realmin = 2^(-1022) или 2.2251e-308
realmax = 21024 или 1.7977e+308
```

К символическим именам констант можно также отнести корень из минус единицы, то есть, **i** или **j**, который использовался в определении комплексных чисел.

Значения арифметических констант можно переопределять и восстанавливать. Например:

```
>> pi = 10          % Переопределим значение pi
pi = 10
>> clear pi        % Восстановим значение pi
>> pi
ans = 3.1416

% Еще один пример

>> i = 10          % Переопределим значение i
i = 10
>> clear i         % Восстановим значение i
>> i
ans = 0 + 1.0000i
```

1.2.2.3. Переменные в MatLab.

Забегая несколько вперед, отметим, что все переменные MatLab являются матрицами (детальнее см. лабораторная работа №2). Иными словами, даже переменная, предназначенная для хранения обычного натурального числа это матрица размером 1x1, то есть, матрица с одной ячейкой. Все переменные в MatLab обозначаются именами.

```
<Имя переменной> ::= <Идентификатор>
<Идентификатор> ::= <Буква> | <Идентификатор><Буква>
    | <Идентификатор><Цифра>
<Буква> ::= <Малые и заглавные буквы латинского алфавита> | <_>
```

1.2.3. Выражение. Простое арифметическое выражение.

Выражения это важнейший способ представить в MatLab символическую запись формул, которые вычисляются для арифметических, логических, текстовых и других значений. Основой для определения синтаксиса выражений служат простые выражения порядок вычисления которых регулируется круглыми скобками или старшинством операций. Поскольку формы Бэкуса — Наура ориентированы только на синтаксическое

конструирование, а правила порядка вычисления выражений относятся к семантике (то есть смысловой нагрузке синтаксических форм) такие правила описываются либо дополнительными грамматиками либо реализуются в виде блок – алгоритмических схем. В части дополнительных языковых грамматик, полезно ознакомиться с языком Дика (англ. Duck language, https://ru.wikipedia.org/wiki/Язык_Дика), который удобно использовать при вычислении выражений в скобках.

В нашем случае, чтобы избежать тяжеловесных цепочек форм Бэкуса — Наура мы будем раскрывать общий синтаксис выражения постепенно. Итак, вначале формализуем общий синтаксис нетерминала выражение.

1.2.3.1. Общий синтаксис нетерминала <Выражение>

```

<Выражение> ::= <Простое выражение>
| <Одноместная операция><Простое выражение>
| <Выражение><Двухместная операция><Простое выражение>
| ( <Выражение> )

<Простое выражение> ::= <Операнд>
| <Одноместная операция><Операнд>
| <Операнд><'>
| <Операнд><Двухместная операция><Операнд>

<Операнд> ::= <Константа>
| <Матрица>
| <Вызов функции>

<Вызов функции> ::= <Имя функции>
| <Имя функции>( )
| <Имя функции> ( <Список параметров> )

<Имя функции> ::= <Идентификатор>

<Список параметров> ::= <Параметр>
| <Параметр> , <Список параметров>

```

Приведенная формализация не раскрывает символического определения нетерминалов <Одноместная операция> и <Двухместная операция>. Дело в том, что операции в MatLab обладают весьма высоким полиморфизмом, то есть, смысл их выполнения и даже состав существенно зависит от типа операндов. Ранее мы уже подчеркивали, что все переменные MatLab являются матрицами.

В рамках данной лабораторной работы мы пока будем рассматривать только арифметические выражения для простых арифметических операций, то есть, операций операндами которых, являются арифметические константы, арифметические матрицы размером не более чем 1x1 или соответствующие вызовы функций. Другими словами, наши операнды будут неотличимы от операндов с которыми работает обычный калькулятор.

1.2.4. Синтаксис простого арифметического выражения.

Синтаксис арифметического выражения будет совпадать с общим синтаксисом выражения, если заменить нетерминал <Операнд> на нетерминал <Простой арифметический операнд>, а также добавить нетерминалы <Простая одноместная

арифметическая операция> и <Простая двухместная арифметическая операция>. При этом, приставки «простой» или «простая» будут означать операнды размером не более чем размерностью 1x1 (скаляры) или операции над соответствующими операндами. Соответственно нетерминал <Матрица> заменим на нетерминал <Имя переменной (размерности 1x1)>:

```
<Простой арифметический операнд> ::= <Арифметическая константа>
| <Имя переменной (размерности 1x1)>
| <Вызов функции (с результатом размерности 1x1)>
```

<Простая одноместная арифметическая операция> ::= -

```
<Простая двухместная арифметическая операция> ::=
- | + | * | / | ^
```

При относительной тяжеловесности синтаксических определений, конечный результат их применения хорошо знаком нам еще со школьной скамьи. Приведем несколько примеров, которые можно проверить прямо в строке приглашения Command Window:

1.2.4.1. Примеры простейших операций без использования имен переменных.

```
>> 2+2
>> 2-2
>> 2*2
>> 4/2
>> 2^2 % возведение во вторую степень
>> sqrt(4) % корень квадратный из 4
```

1.2.4.2. Примеры простейших операций с использованием имен переменных.

```
>> x = 3.2
>> y = -2.5
>> x+y
>> x-y
>> x*y;
>> x/y;
>> x^2;
>> sqrt(x/y);
```

1.2.4.3. Популярные алгебраические функции (краткий список)

```
>> x = 3.2
>> sqrt(x);
>> exp(x);
>> log(x);
>> log10(x);
>> log2(x);
```

1.2.4.4. Константа pi и популярные тригонометрические функции(краткий список)

Аргументы прямых тригонометрических функций и результаты обратных представляются в радианах.

```
pi = 3,1415926535 8979323846 2643383279 5028841971 6939937510...

sin(x); asin(x);
cos(x); acos(x);
```



```
tan(x); atan(x);
cot(x); acot(x);
```

Подробнее о функциях см. HELP (MATLAB Functions: Functions - By Category: Mathematics).

1.2.5. Примеры арифметических выражений.

1.2.5.1. Нахождение корней квадратного уравнения:

Алгебраическая запись:

$$a * x^2 + b * x + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 * a * c}}{2 * a}$$

Запись в MatLab:

```
a*x^2+b*x+c
x=(-b+sqrt(b^2-4*a*c))/(2*a)

% Корень действительное число
>> a=4
>> b=16
>> c=2
>> (-b+sqrt(b^2-4*a*c))/(2*a)
% Результат:
ans = -0.1292
% Корень мнимое число
>> a=4
>> b=2
>> c=2
>> (-b+sqrt(b^2-4*a*c))/(2*a)
% Результат:
ans = -0.2500 + 0.6614i
```

1.2.5.2. Проверка корней квадратного уравнения:

```
% Проверка с действительным корнем
>> a=4
>> b=16
>> c=2
>> x=-0.1292
>> a*x^2+b*x+c
% Результат:
ans = -4.2944e-004

% Проверка с мнимым корнем
>> a=4
>> b=2
>> c=2
>> x=-0.2500 + 0.6614i
>> a*x^2+b*x+c
% Результат:
ans = 2.0016e-004
```

1.2.5.3. Преобразование в радианы (R) угла заданного в градусах (D) и обратно.

```
>> R = pi*D/180
>> D = 180*R/pi
```

1.2.5.4. Нахождение проекций вектора заданного длиной L и углом поворота D (в градусах).

```
>> L=4
>> D=30
>> R = pi*D/180
>> x = L*cos(R)
>> y = L*sin(R)
% Результат:
R = 0.5236
x = 3.4641
y = 2.0000
```

1.2.5.5. Нахождение длины L и угла поворота D (в градусах) для вектора заданного проекциями (x, y) .

```
>> x=3.4641
>> y=2.0000
>> L=sqrt(x^2+y^2)
>> R=atan(y/x)
>> D=180*R/pi
% Результат:
L = 4.0000
R = 0.5236
D = 30.0000
```

1.3. Самостоятельная работа

1.3.1. Составьте собственные имена переменных и выполните примеры приведенные в разделе «Арифметические операции»

1.3.2. Вычислите корни квадратных уравнений, а также выполните несколько операций с проекциями произвольных векторов.

1.3.3. Вычислите с помощью MatLab объемы простейших геометрических тел

Фигура	Формула	Обозначения
Куб	c^3	c — ребро куба
Призма	Sh	S — площадь основания, h — высота призмы
Цилиндр	$\pi r^2 h$	r — радиус, h — высота цилиндра
Шар	$\frac{4}{3}\pi r^3$	r — радиус
Эллипсоид	$\frac{4}{3}\pi abc$	a, b, c — главные оси

Пирамида	$\frac{1}{3}Ah$	A — площадь основания, h — высота пирамиды
Конус	$\frac{1}{3}\pi r^2 h$	r — радиус основания, h — высота конуса

1.3.4. Выполните задание, которое сформулировано в приложении (А). В этом задании рассматривается задача по стыковке челнока с космической станцией при различных начальных параметрах движения челнока относительно станции.

2. MATLAB – ЭТО МАТРИЦЫ (ЛАБ. РАБОТА 2)

2.1. Создание матриц и манипуляция их элементами

На первых шагах рассмотрения матриц мы ограничимся двумя видами матриц, это матрица в виде строки или столбца, а также прямоугольная матрица. Далее, распространим рассмотренные правила и смыслы на многомерные матрицы (или, в другой терминологии, многомерные массивы).

Начнем с того, что, все матрицы MatLab являются динамическими структурами, то есть, они могут изменять свои размеры в процессе выполнения над ними операций.

Динамический характер матриц обусловил одно из важнейших свойств MatLab – любая переменная в MatLab является матрицей.

Все переменные в MATLAB это матрицы

Для проверки этого утверждения воспользуемся простейшим примером:

```
% Определим переменную X и присвоим ей комплексное
% значение 1 + 2j

>> X = 1+2j

% Обратимся к переменной X как к прямоугольной
% матрице с индексом строки и индексом столбца
% равными единице

>> X(1,1)
ans = 1.0000 + 2.0000i
```

2.1.1. Определение одномерных матриц (векторов).

Под одномерными матрицами будем подразумевать матрицы, состоящие из одной строки, содержащей несколько столбцов (вектор – строка) или состоящие из множества строк, где в каждой строке присутствует только один столбец (вектор столбец).

```
<Вектор > ::= < [ > < ] >
| <Вектор-строка>
| <Вектор-столбец>
```

```
<Вектор-строка> ::= < [ ><Список вектора-строки>< ] >
```

```
<Вектор-столбец> ::= < [ ><Список вектора-столбца>< ] >
```

Примечание. В некоторых случаях самоопределенные термы могут совпадать с элементами изображения термов в формах Бэкуса—Наура. В таких случаях они будут записываться с помощью термскобок. Например, знак «квадратная скобка» будет записан как < [> или <] >.

```
<Список вектора-строки> ::= <Значение вектора-строки>
| < Список вектора-строки > , <Значение вектора-строки>
| < Список вектора-строки > <' '> <Значение вектора-строки>
```

<Список вектора-столбца> ::= <Значение вектора-столбца>
| < Список вектора-столбца > ; <Значение вектора-столбца>

<Значение вектора-строки> ::= <Арифметическое значение>
| <Вычисленное логическое значение>
| <Имя вектора-строки>
| <Строка-диапазон значений>

<Значение для вектора-столбца> ::= <Арифметическое значение>
| <Вычисленное логическое значение>
| <Имя вектора-столбца>

<Арифметическое значение> ::= <Целое со знаком>
| <Действительное число>
| <Комплексное число>

Примечание. <Целое со знаком> <Действительное число> и <Комплексное число> было сделано в лабораторной работе №1

<Вычисленное логическое значение> ::= true | false

Примечание. Формальное определение логических выражений, которые порождают <Вычисленное логическое значение> мы сделаем в последующих лабораторных работах.

<Строка-диапазон значений> ::= <Начало>[:<Шаг>]:<Конец>

<Начало> ::= <Имя переменной> | <Значение для диапазона>

<Шаг> ::= <Имя переменной> | <Значение для диапазона>

<Конец> ::= <Имя переменной> | <Значение для диапазона>

<Значение для диапазона> ::= <Целое со знаком>
| <Действительное число>

Примечание. Численные значения начала, шага и конца диапазона следует выбирать так, чтобы выполнялось:

Начало < Конец

Начало + Шаг > Начало

Кроме того, если шаг не задан, то его значение равно единице.

И наконец:

<Имя вектора-строки> ::= <Идентификатор>

<Имя вектора-столбца> ::= <Идентификатор>

2.1.2. Примеры создания одномерных матриц (векторов).

2.1.2.1. Явное определение вектора-строки или вектора-столбца

```
>> % Явное определение вектора-строки (вариант 1)
>> X1=[1 2 3]
>> % Явное определение вектора-строки (вариант 2)
>> X2=[4, 5, 6]

>> % Явное определение вектора-столбца
>> Y1=[1; -2; 3; -1.12; 1.55e-2]
Y1= 1.0000
    -2.0000
     3.0000
    -1.1200
     0.0155
```

2.1.2.2. Определение вектора-строки с использованием имени вектора-строки

```
>> X3=[7 8 X2]
X3 = 7      8      4      5      6
```

2.1.2.3. Определение вектора-строки с использованием значения 1X1, вычисляемого функцией и имени вектора-строки.

```
>> X3=[sqrt(4), X1]
X3 = 2      1      2      3
```

2.1.2.4. Определение вектора-строки с использованием значения комплексного числа и имени вектора-строки.

```
>> X4=[1+2j, X1]
X4 = 1.0000 + 2.0000i    1.0000    2.0000    3.0000
```

2.1.2.5. Определение вектора-строки с использованием имени вектора-строки и вычисленного логического значения.

```
>> X5=[X1, true]
X5 = 1      2      3      1
```

2.1.2.6. Явное определение значений вектора-строки с помощью начального значения, шага и конечного значения.

```
>> X6=[1,2, 4:8]
X6 = 1      2      4      5      6      7      8

>> X6= [2.5 : 0.1 : 2.8]
X6= 2.5000    2.6000    2.7000    2.8000
```

Неявное определение значений вектора-строки с помощью начального значения, шага и конечного значения.

```
>> xb=2.5;
>> step=0.1;
>> xe=2.8;
>> X7=[xb:step:xe, 1,2]
X7 = 2.5000    2.6000    2.7000    2.8000    1.0000    2.0000
```

Мы уделили основное внимание синтаксису и способам создания векторов-строк и привели только один пример создания вектора-столбца (см. пример 2.1.1). Это

обусловлено тем, что преобразовать вектор-строку в вектор-столбец достаточно просто, используя операцию транспонирования. Воспользуемся результатом последнего примера (2.1.6) и преобразуем X7 в Y7:

2.1.2.7. Преобразование значений вектора-строки в значения вектора-столбца.

```
>> Y7 = X7'  
Y7 = 2.5000  
      2.6000  
      2.7000  
      2.8000  
      1.0000  
      2.0000
```

2.1.3. Определение прямоугольных матриц (2D-матриц).

Прямоугольные матрицы можно считать основной формой структуризации данных в MatLab. Такие матрицы получают объединением (конкатенацией) векторов-строк одинакового размера или объединением векторов-столбцов, также одинакового размера. Синтаксис определения прямоугольных матриц имеет вид:

```
<2D-матрица> ::= <[><Строки 2D-матрицы><]>  
              | <[><Столбцы 2D-матрицы><]>  
  
<Строки 2D-матрицы> ::= <Вектор-строка>  
                        | <Строки 2D-матрицы> ; <[><Вектор-строка><]>  
  
<Столбцы 2D-матрицы> ::= <Вектор-столбец>  
                        | <Столбцы 2D-матрицы> , <[><Вектор-столбец><]>
```

2.1.4. Примеры создания прямоугольных матриц (2D-матриц).

2.1.4.1. Явное определение прямоугольной матрицы из двух векторов-строк.

```
>> A1=[[1,2,3]; [4,5,6]]  
A1 = 1     2     3  
      4     5     6
```

2.1.4.2. Явное определение прямоугольной матрицы из трех векторов-строк, заданных одинаковыми диапазонами.

```
>> A2=[[1:3]; [4:6]; [8:10]]  
A2 = 1     2     3  
      4     5     6  
      8     9    10
```

2.1.4.3. Определение прямоугольной матрицы из трех векторов-строк, заданных именами X1 и X2.

```
>> X1=[1 2 3]  
% Результат  
X1= 1 2 3  
  
>> X2=[4, 5, 6]  
% Результат
```

```

X2= 4  5  6

>> A3=[X1; X2]
% Результат
A3 =  1  2  3
      4  5  6

```

Примечание: В некоторых случаях, как правило, когда вектора представлены переменными или диапазонами допускается опускать квадратные скобки, охватывающие отдельные вектора. Однако такая возможность сокращенного синтаксиса не является общим правилом и может вызывать ошибку «Error using ==> horzcat»

2.1.4.4. Явное определение прямоугольной матрицы из двух векторов-столбцов.

```

A4=[[1;2;3], [4;5;6]]
% Результат
A4 =  1  4
      2  5
      3  6

```

2.1.5. Правила доступа к структурам внутри матриц

Доступ к структурам внутри вектора или прямоугольной матрицы осуществляется с помощью указания координат таких структур. Самыми простейшими структурами внутри матриц являются отдельные ячейки содержимым, которых являются значения численных или других типов. Отдельные ячейки матриц нумеруются натуральными числами (индексами) начиная от единицы. Области внутри матриц (субматрицы) указываются диапазонами индексов.

Общий синтаксис доступа к внутренним структурам векторов и 2D матриц определяется следующими правилами:

```

<Субматрица> ::= <Субматрица вектора>
               | <Субматрица 2D-матрицы>

<Субматрица вектора> ::= <Имя матрицы>(<Диапазон индексов>)

<Имя матрицы> ::= <Идентификатор>

<Диапазон индексов> ::= <Индекс>
                       | <Индекс> : <Индекс>
                       | <Индекс> : end
                       | :

<Индекс> ::= <Индекс-переменная> | <Индекс-число>

<Индекс-переменная> ::= <Имя переменной>

```

Примечание: такая переменная должна содержать только значения, которые определяются как <Индекс-число>

```

<Индекс-число> ::= <Ненулевая цифра>
                 | <Индекс-число><Ненулевая цифра>

<Ненулевая цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```



```
< Субматрица 2d-матрицы> ::=  
    <Имя матрицы>( <Индексы-строки>, <Индексы-колонки> )  
  
<Индексы-строки> ::= <Диапазон индексов>  
<Индексы-колонки> ::= <Диапазон индексов>
```

2.1.6. Примеры доступа к структурам внутри матриц

2.1.6.1. Доступ к ячейке с индесами строки=2 и столбца=4

```
A = [[1,2,3,4,5];[6,7,8,9,10];[11,12,13,14,15]]  
B = [] % Обнуллим матрицу B  
B = A(2,4)
```

2.1.6.2. Доступ к субматрице (области внутри матрицы) с диапазонами по строкам от 2 до 3 и столбцами от 4 до 5

```
A = [[1,2,3,4,5];[6,7,8,9,10];[11,12,13,14,15]]  
B = ones(8,8)  
B(1:2,1:2) = A(2:3,4:5)
```

2.1.6.3. Доступ к субматрице (области внутри матрицы) с индексом строки=5 и диапазоном по столбцами от 3 до 5

```
A = [[1,2,3,4,5];[6,7,8,9,10];[11,12,13,14,15]]  
B = [] % Обнуллим матрицу B  
A(3,3:5)  
B(5,7:9) = A(3,3:5)
```

2.1.6.4. Доступ к субматрице (области внутри матрицы) с диапазонами по строкам от 2 до 3 и столбцами от 4 до последнего в матрице

```
A = [[1,2,3,4,5];[6,7,8,9,10];[11,12,13,14,15]]  
B = [] % Обнуллим матрицу B  
B = A(2:3,4:end)
```

2.1.6.5. Доступ к субматрице (области внутри матрицы) с диапазонами по всем строкам и столбцами от 4 до последнего в матрице

```
A = [[1,2,3,4,5];[6,7,8,9,10];[11,12,13,14,15]]  
B = [] % Обнуллим матрицу B  
B = A(:,4:end)
```

2.1.7. Матрицы MATLAB это динамические структуры

Динамическими называют данные, если в процессе выполнения над ними операций они изменяют не только свои значения, но и структуру представления. Для матриц MatLab минимальными структурами, над которыми реализуются динамические преобразования, являются вектора.

Примеры динамического расширения размеров 2D-матриц.

2.1.7.1. Создание пустой матрицы

```
>> A=[]  
A = []
```

Примечание. Такая матрица представлена только своим именем и не содержит ни одной строки и, соответственно, ни одного столбца.

2.1.7.2. Создание нулевой матрицы, заданного размера. Например, 3 строки, 4 столбца.

```
>> A(3,4)=0
A = 0     0     0     0
     0     0     0     0
     0     0     0     0
```

Примечание. Аналогичную операцию выполняет стандартная в MatLab функция `zeros(3,4)`.

2.1.7.3. Расширение существующей матрицы до заданного размера.

Пусть в процессе вычислений у нас получилась некоторая матрица, которую мы сформируем следующим образом:

```
A=[1:3; 4:6; 7:9]
A = 1     2     3
     4     5     6
     7     8     9
```

Если у нас возникла необходимость ее расширить, то достаточно задать всего одну ячейку за пределами ее текущей размерности, например:

```
>> A(4,2)=42
A = 1     2     3
     4     5     6
     7     8     9
     0    42     0
```

или, если необходимо выполнить еще более радикальное расширение, то:

```
>> A(6,7)=67
A = 1     2     3     0     0     0     0
     4     5     6     0     0     0     0
     7     8     9     0     0     0     0
     0    42     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     67
```

Примечание. Конкретные значения, которые мы присваивали ячейке, расширяющей матрицу, выбраны только из соображения наглядности и могут быть произвольными в рамках допустимых типов.

2.1.8. Примеры манипуляции строками и столбцами или субматрицами в 2D-матрицах.

Достаточно часто необходимо возникает задача изменить значения строки, столбца или субматрицы в рамках некоторой исходной матрицы. Проиллюстрируем решение этой задачи на примере перемещения двух субматриц внутри исходной матрицы без потери их значений.

2.1.8.1. Пример обмена данными между субматрицами в одной матрице.

Создаем из строк исходную матрицу по сокращенному синтаксису

```
>> A=[1:4; 5:8; 9:12]
```

```
A = 1 2 3 4
     5 6 7 8
     9 10 11 12
```

Выполняем сохранение значений первой субматрицы в буферной (временной) переменной TEMP.

```
>> TEMP=A(1:2, 1:2)
TEMP = 1 2
       5 6
```

Копируем вторую субматрицу на место первой

```
>> A(1:2, 1:2)=A(2:3, 3:4)
A = 7 8 3 4
    11 12 7 8
    9 10 11 12
```

Примечание. Размерности и размеры субматриц слева и справа от операции присвоения должны быть строго одинаковы.

Перемещаем первую (сохраненную) субматрицу на место второй

```
>> A(2:3, 3:4)=TEMP
A = 7 8 3 4
    11 12 1 2
    9 10 5 6
```

Динамически освобождаем временную переменную TEMP, то есть, удаляем ее из Workspace.

```
>> clear TEMP
```

Примечание. Функция clear в системе MATLAB позволяет динамически удалять из Workspace все, несколько или конкретную переменную. Синтаксис функции clear смотрите в HELP.

2.1.9. Пример динамического понижения размеров и размерности исходной 2D-матрицы.

Также достаточно часто (например, при построении миноров), необходимо удалить некоторую строку или столбец, что равносильно динамическому изменению размеров исходной матрицы. Покажем способ реализации такой операции

Создаем из строк исходную матрицу.

```
>> A=[[1:4]; [5:8]; [9:12]]
A = 1 2 3 4
     5 6 7 8
     9 10 11 12
```

Удалим в матрице второй столбец.

```
>> A(:,2)=[]
A = 1 3 4
     5 7 8
     9 11 12
```

Далее, удалим вторую строку:

```
>> A(2,:)=[]  
A = 1 3 4  
    9 11 12
```

Далее, удалим вторую строку (первое понижение размерности):

```
>> A(1,:)=[]  
A = 9 11 12
```

Далее, удалим первый столбец:

```
>> A(:,1)=[]  
A = 11 12
```

Наконец, удалим второй столбец (второе понижение размерности):

```
>> A(:,2)=[]  
A = 11
```

2.2. Матрицы с размерностью больше чем 2D

В материале, который мы рассматривали ранее, размерность матриц не превышала значения 2. Например, размерность векторов равна единице или 1D(one-dimensional array), размерность прямоугольных матриц равна двум или 2D (two-dimensional array). В отличие от термина «размер» (size), который определяет количество ячеек, термин «размерность» определяет необходимое число индексов (или координат) для однозначного обращения к одной ячейки матрицы.

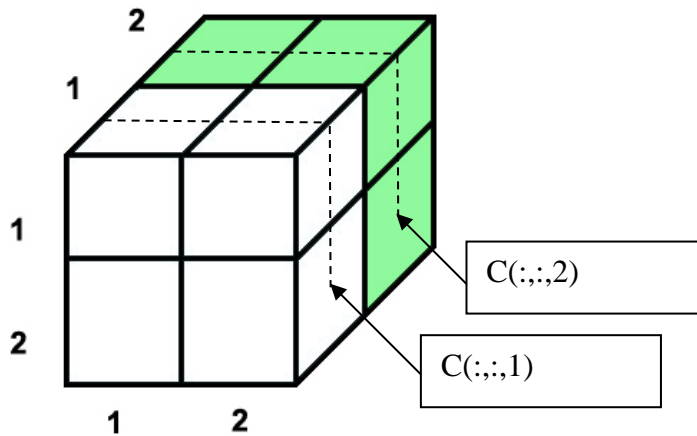
Система MatLab позволяет работать с матрицами размерность которых может превышать 2D. Такие матрицы называют многомерными матрицами(multi-dimensional array). Создание таких матриц синтаксически подобно созданию 2D-матриц.

2.2.1. Пример создания матриц 3D

2.2.1.1. В качестве примера покажем создание матрицы 3D как кубика с размерами две ячейки в каждой размерности:

```
C(2,2,2)=222  
% Результат:  
C(:,:,1) = 0 0  
           0 0  
C(:,:,2) = 0 0  
           0 222
```

Результат этой операции явно требует пояснений. Итак, представим заданную матрицу в виде следующего рисунка:



Тогда результат, обозначенный как $C(:,:,1)$, отобразит нам все значения ячеек ее первого слоя, а результат, обозначенный как $C(:,:,2)$, все значения ячеек ее второго слоя

Примечание. Такой способ создания можно назвать как создание матриц путем указания конечных индексов на главной диагонали.

2.2.2. Пример манипуляции значениями и размерностью матриц 3D

В 3D-матрице можно манипулировать не только значениями конкретных ячеек или векторов как 2D-матрице, но также можно изменять целые слои. Для этого конкретный слой необходимо зафиксировать в одной из размерностей, после чего заменить значения слоя.

2.2.2.1. Манипуляция значениями матрицы, в которой слои фиксируется по *третьей* размерности:

```
% Сформируем исходную матрицу
>> C(2,2,2)=222;

% Подготовим значения заменяющие значения слоя
>> V=[1,2; 3,4];

% Заменяем матрицей «В» первый слой матрицы «С»
>> C(:,:,1)=V
C(:,:,1) = 1     2
           3     4
C(:,:,2) = 0     0
           0    222

% Заменяем первую строку (вектор) во втором слое
>> C(1,:,2)=[5, 6]
C(:,:,1) = 1     2
           3     4
C(:,:,2) = 5     6
           0    222

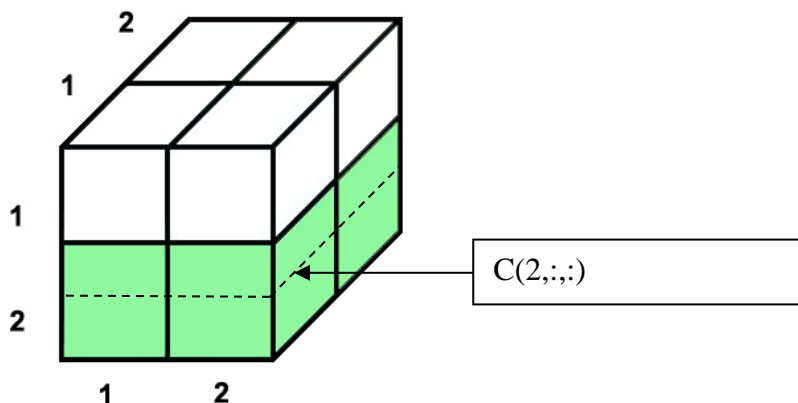
% В завершение удалим второй слой полностью
>> C(:,:,2)=[]
C = 1     2
     3     4
```

Примечание. Подчеркнем, что последняя операция в этом примере изменяет не только значения матрицы но и ее размерность.

2.2.2.2. Манипуляция значениями матрицы, в которой слои фиксируется по *первой* размерности:

```
% Сформируем исходную матрицу
>> C(2,2,2)=222;

% Покажем фиксацию второго слоя по первой размерности с помощью
следующего рисунка:
```



```
% Заменяем (явно заданной) 2D-матрицей зафиксированный слой матрицы «С»
```

```
>> C(2,:,:)=[[1,2]; [3,4]]
C(:,:,1) = 0     0
           1     3
C(:,:,2) = 0     0
           2     4
```

Примечание. Обратите внимание - не смотря на то, что операция осуществлялась в слое с фиксированной строкой, результат отображается слоями (сечениями) с фиксацией по третьей размерности. Эргономика такого отображения это достаточно субъективная территория или как шутят программисты - это чисто религиозный вопрос ☺

2.2.3. Несколько слов о матрицах размерностью 4D и выше

Я думаю, Вы уже догадались, что индексы четвертой размерности позволяют создавать и избирательно работать с множеством 3D-матриц. Такое правило работает и для более высоких размерностей.

К сожалению, с повышением размерности отображение результата становится все менее и менее читаемым. В этом легко убедиться, выполнив создание, к примеру, 8D-матрицы:

```
>> A(2,2,2,2,2,2,2,2)=0
```

Отображение результата такой операции представляет собой 64 сечения в виде 2D-матрицы каждое. Это создает заметные трудности при планировании обработки значений, особенно при работе со средними (в смысле размерностей) структурами внутри таких

матриц. Одним из выходов в такой ситуации можно считать переход к программным средствам обработки, которые мы рассмотрим в дальнейшем.

2.2.4. Сервисные функции MATLAB для работы с матрицами

Большинство функций MatLab имеют много вариантов списка их параметров, а соответственно и смысла каждого из них. Детальное описание таких параметров для каждой функции явно выходит за рамки лабораторной работы. По этой причине, мы ограничимся только именами функций и кратким смыслом их действий. Для получения полной информации обращайтесь в подсистему HELP.

Группа функций, которая позволяет получить размеры динамически изменяющихся матриц

№	Имя функции	Описание
1	length	Возвращает длину вектора в ячейках
2	size	Возвращает для каждой или выбранной размерности матрицы длину этой размерности в ячейках

Группа функций, которая позволяет создавать и инициализировать типичные варианты матриц.

№	Имя функции	Описание
1	zeros	Создает матрицу заданной размерности, ячейки которой заполнены нулями.
2	ones	Создает матрицу заданной размерности, ячейки которой заполнены единицами.
3	eye	Создает матрицу заданной размерности, ячейки главной диагонали которой, заполнены единицами, а остальные ячейки заполнены нулями.
4	rand	Создает матрицу заданной размерности, ячейки которой заполнены случайными значениями в диапазоне (0-1) в соответствии с равномерным законом распределения.
5	randn	Создает матрицу заданной размерности, ячейки которой заполнены случайными значениями в диапазоне (0-1) в соответствии с нормальным законом распределения.

Группа функций для выполнения типичных трансформаций в матрицах.

№	Имя функции	Описание
1	fliplr	Выполняет для векторов и 2D-матриц зеркальное отображение относительно вертикали.
2	flipud	Выполняет для векторов и 2D-матриц зеркальное отображение относительно горизонтали.
3	rot90	Выполняет для квадратных 2D-матриц вращение значений матрицы на углы кратные 90°.

Примечание. В приведенных таблицах представлены только наиболее часто используемые функции, условно разделенные на три категории. На самом деле, таких функций, особенно в категориях 2 и 3 значительно больше. Подсистема

HELP с помощью вкладки SEARCH и гиперссылок позволяет подобрать необходимую Вам функцию из множества функций MATLAB.

2.3. Операции с матрицами в MATLAB

Операции MATLAB над матрицами подразделяются на два класса операций – это поэлементные операции и матричные операции.

Поэлементные операции применимы к матрицам любой размерности. Главное при двухместных поэлементных операциях это использование одинаковых размеров в матрицах операндах.

Матричные операции, как правило, применяются для решения задач линейной алгебры и в данной лабораторной работе не рассматриваются. Рассмотрение таких операций будет выполняться по мере необходимости их использования в различных дисциплинах учебного процесса. В рамках данной дисциплины такие операции будут рассмотрены в дополнительных лабораторных работах, ориентированных на самостоятельное изучение.

2.3.1. Операция транспонирования

В рамках данной работы рассмотрим только одноместную матричную операцию – <транспонирование>, которая определена для векторов и прямоугольных матриц.

<транспонированная матрица> ::= <матрица>'

Данная операция выполняет перестановку местами строк и столбцов исходной матрицы, то есть все строки исходной матрицы становятся столбцами результирующей матрицы.

2.3.1.1. Пример транспонирования векторов и двумерных матриц:

```
% Пример транспонирования вектора - строки
>> A=[1 2 3 4]
>> B=A'
% Результат:
A =
     1     2     3     4
B =
     1
     2
     3
     4

% Пример транспонирования вектора - столбца
>> A=[1; 2; 3; 4]
>> B=A'
% Результат:
A =
     1
     2
     3
     4
B =
     1     2     3     4
```

2.3.1.2. Пример транспонирования двумерных матриц:

```
% Пример транспонирования прямоугольной матрицы
>> A=[1:3; 5:7; 9:11]
>> B=A'
```



```

% Результат:
A = 1     2     3
     5     6     7
     9    10    11
B = 1     5     9
     2     6    10
     3     7    11

```

2.3.2. Поэлементные операции для арифметических матриц

Ранее мы сформулировали обобщенное правило для нетерминала <Выражение>

```

<Выражение> ::= <Простое выражение>
              | <Одноместная операция><Простое выражение>
              | <Выражение><Двухместная операция><Простое выражение>
              | ( <Выражение> )

<Простое выражение> ::= <Операнд>
                       | <Одноместная операция><Операнд>
                       | <Операнд><'>
                       | <Операнд><Двухместная операция><Операнд>

<Операнд> ::= <Константа>
              | <Матрица>
              | <Вызов функции>

<Вызов функции> ::= <Имя функции>
                   | <Имя функции>( )
                   | <Имя функции> ( <Список параметров> )

<Имя функции> ::= <Идентификатор>

<Список параметров> ::= <Параметр>
                       | <Параметр> , <Список параметров>

```

Как уже говорилось, приданная формализация не раскрывает определения нетерминалов <Одноместная операция> и <Двухместная операция>. Кроме того подчеркивалось, что операции в MATLAB обладают весьма высоким полиморфизмом, то есть, смысл их выполнения и даже состав существенно зависит от типа операндов.

В рамках данной темы мы уделим основное внимание поэлементным арифметическим операциям, операндами которых являются матрицы различной размерности.

Поэлементными называются операции, которые выполняются над скалярными элементами матриц с одинаковыми индексами. Результатом являются скалярные элементы матрицы с индексами аналогичными индексам матриц операндов. Размеры матриц операндов должны быть равны.

2.3.3. Одноместные поэлементные операции для арифметических матриц любой размерности

Для арифметических матриц любой размерности определяется единственная одноместная операция (минус):

```

<Одноместная операция для nD матриц> ::= -

```

Практически, это означает изменение знака на противоположный для всех значений в ячейках матрицы, например:

```
% Пусть:
>>A
A(:, :, 1) = 2     2
             2     2
A(:, :, 2) = 2     2
             2     2

% Тогда:
>> -A
ans(:, :, 1) = -2    -2
              -2    -2
ans(:, :, 2) = -2    -2
              -2    -2
```

2.3.4. Двухместные поэлементные операции для арифметических nD-матриц и простого арифметического операнда (скаляра).

Напомним, что простой арифметический операнд мы уже рассматривали в лабораторной работе 1 в виде:

```
<Простой арифметический операнд> ::= <Арифметическая константа>
| <Имя переменной (размерности 1x1)>
| <Вызов функции (с результатом размерности 1x1)>
```

При этом под размерностью 1x1 мы подразумевали матрицу с единственной ячейкой. Следует также отметить, что простой арифметический операнд часто называют скаляром.

Для такого сочетания операндов двухместная арифметическая операция определяется правилом:

```
<двухместная арифметическая операция для nD-матрицы и скаляра> ::=
- | + | * | / | ^ | ./ | .* | .^
```

Обратим внимание, что помимо символов традиционных операций умножения, деления и возведения в степень ($*$ $/$ $^$), используются их двухсимвольные обозначения с лидирующей точкой. Такие двухсимвольные обозначения подчеркивают необходимость выполнять операцию поэлементно, то есть, выполнять соответственно для каждой ячейки nD-матрицы и ячейки простого арифметического операнда (**скаляра**). Необходимо подчеркнуть, что для операций между скаляром и скаляром, а также матрицей и скаляром операции ($*$ $/$ $^$) и ($.*$ $./$ $.^$) выполняются совершенно одинаково. Очевидно, что конечным результатом такой операции будет либо скаляр, либо nD-матрица.

Например:

2.3.4.1. Пусть имеются исходная кубическая матрица A. Для начала выполним ее создание:

```
>> A = ones(3,3,3)
>> A * 4
% Результат:
ans(:, :, 1) = 4     4
              4     4
```

```

      4      4
ans(:,:,2) = 4      4
              4      4
              4      4

```

2.3.4.2. На базе полученных в 2.3.4.1 значений матрицы A приведем еще один пример, который иллюстрирует возможные позиции операндов в рассматриваемой операции:

```

>> 4 / A
% Результат:
ans(:,:,1) = 1      1
              1      1
              1      1
ans(:,:,2) = 1      1
              1      1
              1      1

```

или (напомним что $\sqrt{4} = 2$)

```

>> (A.*2)./sqrt(4)
% Результат:
ans(:,:,1) = 1      1
              1      1
              1      1
ans(:,:,2) = 1      1
              1      1
              1      1

```

2.3.4.3. Еще один важный пример показывает, что комплексное число интерпретируется в MATLAB как арифметическое выражение, состоящее из суммы действительной и мнимой частей:

```

>> A = ones(3,3,3).*2
>> A .^ (2 + 2i)

% Результат:
ans(:,:,1) = 0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i

ans(:,:,2) = 0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i

ans(:,:,3) = 0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i
              0.7338 + 3.9321i    0.7338 + 3.9321i

```

Однако, учитывая старшинство операций при выполнении следующих строк, мы уже получим такой результат:

```

>> A = ones(3,3,3).*2
>> A .^ 2 + 2i

% Результат:
ans(:,:,1) = 4.0000 + 2.0000i    4.0000 + 2.0000i
              4.0000 + 2.0000i    4.0000 + 2.0000i
              4.0000 + 2.0000i    4.0000 + 2.0000i

ans(:,:,2) = 4.0000 + 2.0000i    4.0000 + 2.0000i
              4.0000 + 2.0000i    4.0000 + 2.0000i
              4.0000 + 2.0000i    4.0000 + 2.0000i

```

2.3.5. Двухместные поэлементные операции для арифметических nD-матриц одинаковой размерности и размера.

Если операндами двухместной операции являются nD- матрицы одинаковой размерности и размера то результатом операции является nD-матрица аналогичной размерности и размера, а значения ее ячеек будут вычисляться как результат приенения операции к значениям ячеек nD-матриц операндов с одинаковыми индексами. Математически это можно записать следующим образом:

Для каждого полностью определенного варианта значений индексов nD-матриц:
 $inx1, inx2, \dots, inxN$
 поэлементно выполнить:
 $Y (inx1, \dots, inxN) = X1 (inx1, \dots, inxN) \langle \text{оператор} \rangle X2 (inx1, \dots, inxN)$

Назовем такую операцию nD-двухместной арифметической операцией.

`<nD-двухместная арифметическая операция > ::=`
`- | + | .* | ./ | .^`

Как видно из приведенного правила, операции * и / исключены из списка возможных. Это связано с тем, что исключенные символы умножения и деления используются в MATLAB не только для поэлементных, но и для матричных операций, которые определяются в линейной алгебре.

Приведем несколько примеров двухместных операций для двух арифметических nD-матриц одинаковой размерности и размера.

2.3.5.1. Пусть имеются две исходные кубические матрицы A и B одинаковой размерности. Для начала выполним их создание:

```
>> A = ones(3,3,3);
>> B = ones(3,3,3).*2;
% Пример поэлементной операции деления:
>> C = A ./ B
% Результат:
C(:,:,1) = 0.5000    0.5000
           0.5000    0.5000
           0.5000    0.5000
C(:,:,2) = 0.5000    0.5000
           0.5000    0.5000
           0.5000    0.5000
C(:,:,3) = 0.5000    0.5000
           0.5000    0.5000
           0.5000    0.5000
```

2.3.5.2. Пример, в котором значения матрицы C (см. пример 2.3.5.1) используются как набор показателей степени для каждого элемента матрицы B:

```
>> B .^ C
% Результат:
ans(:,:,1) = 1.4142    1.4142
            1.4142    1.4142
            1.4142    1.4142
ans(:,:,2) = 1.4142    1.4142
```

```
ans(:, : , 3) = 1.4142    1.4142
                1.4142    1.4142
                1.4142    1.4142
                1.4142    1.4142
```

2.4. Самостоятельная работа

2.4.1. Выполните все примеры из теоретической части.

2.4.2. Опробуйте создание и трансформации матриц с помощью типичных функций и для различных вариантов списка их параметров.

3. ПОСТРОЕНИЕ ГРАФИКОВ (ЛАБ. РАБОТА 3)

Представление функций заданных как аналитически, так и в табличной форме в виде графиков позволяет существенным образом повысить информативность при восприятии функции как некоторого процесса. При этом ясно проявляются такие особенности функций как:

- возрастающий или убывающий характер как в целом, так и на отдельных участках функции;
- проявляют себя участки функции с точками перегиба;
- появляется визуальная возможность оценить порядок первой производной функции на различных участках;
- в некоторых случаях достаточно четко проявляют себя особые точки, такие как: разрывы функции; неравенство первой производной слева и справа в некоторых точках (излом функции) и некоторые другие особенности.

Для отображения функций в виде графиков обычно используются два вида координат, это декартовы координаты или полярные координаты.

Чаще всего используются графики для функций одной переменной. В этом случае:

- Декартовы координаты позволяют представить график функции на плоскости в виде плавной линии где каждая отображаемая точка линии задается проекциями этой точки на ось X и ось Y . При этом значение проекции на ось X определяется аргументом функции, а значение Y определяется значением функции для такого аргумента.
- Полярные координаты позволяют представить график функции на плоскости в виде плавной линии где каждая отображаемая точка линии задается длиной и углом поворота вектора. При этом угол поворота определяется аргументом функции. а длина вектора определяется значением функции для такого аргумента. Другими словами, линию графика рисует кончик вращающегося вектора переменной длины.

Между декартовыми и полярными координатами имеет место взаимнооднозначное преобразование.

Переход от декартовых координат к полярным:

$$L = \sqrt{X^2 + Y^2}$$

$$Alpha = \arcsin(Y / L)$$

Переход от полярных координат к декартовым соответственно имеет вид:

$$X = L \cdot \sin(Alpha)$$

$$Y = L \cdot \cos(Alpha)$$

Для функций двух переменных осуществляется переход от плоскости к различным представлениям трехмерного пространства. При этом в декартовых координатах

добавляется еще одна ось для второго аргумента функции, а в полярных координатах соответственно еще один угол поворота.

Функции, которые отображают графики достаточно часто используются при выполнении работ в MATLAB. По этой причине будет полезнее представить их описание в том виде, который доступен в подсистеме HELP

3.1. PLOT - построение графиков, заданных матрицами.

Функция PLOT строит график в декартовых координатах.

3.1.1. Функция PLOT. Синтаксис

- PLOT (X, Y) отображает значения взятые из вектора Y как значения функции по оси Y, соответственно значения аргумента функции берутся из вектора X. Если X или Y - матрица, то график строится по строкам или столбцам матрицы. Если X - скаляр, а Y - вектор, то график представляет собой вертикальную цепочку точек или линию Y - проекции которой определяются значениями из вектора Y, а X – проекция определяется скаляром X (такое отображение называется отображением в отложенных точках).
- PLOT (Y) отображает значения из столбцов вектора Y, используя в качестве аргумента индексы этого вектора. Если Y комплексно, то PLOT (Y) эквивалентно PLOT (real (Y), imag (Y)). Во всех других применениях PLOT мнимая часть игнорируется.
- Различные типы линий, графические символы и цвета могут быть получены с помощью PLOT (X, Y, S), где S - символьная строка («LineStyle»), сформированная из символов:

ColorOrder - Первый символ строки S

Символ	Смысл	Расшифровка
b	blue	синий
g	green	зеленый
r	red	красный
c	cyan	Голубой(электрик)
m	magenta	пурпурный
y	yellow	желтый
k	black	черный
w	white	белый

Второй символ строки S

Символ	Смысл	Расшифровка
.	point	Точка
o	circle	Круг
x	x-mark	Значок X
+	plus	Значок +
*	star	Звездочка
s	square	Квадрат
d	diamond	Ромбик
v	triangle (down)	Треугольник (вниз)

^	triangle (up)	Треугольник (вверх)
<	triangle (left)	Треугольник (слева)
>	triangle (right)	Треугольник (справа)
p	pentagram	Пентаграмма
h	hexagram	Гексаграмма

LineStyleOrder - Остальные символ/символы строки S

Символы	Смысл	Расшифровка
-	solid	Непрерывная линия
:	dotted	точками
-.	dashdot	Пунктир-точка
--	dashed	Пунктир

Например:

PLOT (X, Y, 'c +:') отображает голубую пунктирную линию с плюсом в виде маркера точек '+' в каждой точке данных;

PLOT (X, Y, 'bd') отображает синий алмаз по каждому данным но не рисует ни одной линии.

- PLOT (X1, Y1, S1, X2, Y2, S2, X3, Y3, S3, ...) объединяет графики, определенные (X, Y, S) троек, где X и Y являются векторами или матрицами и S - это строки.

Например, PLOT (X, Y, 'y -', X, Y, 'go') отображает данные дважды, накладывая на сплошную желтую линию, зеленые круги в точках данных.

Команда PLOT, если цвет не указан, автоматически использует цвета, указанные свойством ColorOrder. ColorOrder определяется в таблице для цветных систем, где по умолчанию синий цвет используется для одной строки, а для нескольких строк применяется цикл через первые шесть цветов в таблице. Для монохромных систем, PLOT циклически перемещает по осям свойство LineStyleOrder.

Смотрите также функции:

SEMILOGX, SEMILOGY, LOGLOG, PLOTYY, GRID, CLF, CLC, TITLE, XLABEL, YLABEL, AXIS, AXES, HOLD, COLORDEF, LEGEND, SUBPLOT, STEM.

3.1.2. Функция PLOT. Примеры

3.1.2.1

```
% Очистка Command Window и прорисовка
% строки приглашения для ввода команд
help clc
clc
```

3.1.2.2

```
% Очистка (WorkSpace)
help clear
```

3.1.2.3

```
% Рисование графика для табличной функции
help plot

% Построим первый график
x=-32:0.01:32;
```



```
y=sin(x)./x;
plot(x,y);
```

3.1.2.4

```
% Специальные форматы plot отображения графиков, когда в качестве X
% используется матрица, либо матрицы используются в качестве X и Y
X1 = 1:0.2:2;
X2 = 3:0.2:4;
X3 = 6:0.2:7;
Z4 =[X1',X2',X3'];
plot (1,X1,'d') % Аргумент скаляр, Функция вектор
input('Для продолжения нажмите Enter ... ');
plot (X1) % Прорисовка вектора
input('Для продолжения нажмите Enter ... ');
plot (Z4) % Прорисовка матрицы
input('Для продолжения нажмите Enter ... ');
plot (Z4, Z4) % Аргумент матрица, Функция матрица
```

3.1.2.5

```
% Заморозка окна для рисования в нем многих графиков (hold on)
help hold

% Включение grid on / выключение сетки grid off
help grid

% Команда закрыть окно графика
help close
```

3.1.2.6.1

```
% Построим графики в замороженном (общем) окне.
% Порядок построения с возрастающим диапазоном по X
clear all;
close all;
hold on; % Заморозить окно
x=-2*pi:0.2:2*pi; % Диапазон 4*pi
y=sin(x)./x;
plot(x,y, '-*r');
input('Для продолжения нажмите Enter ...')
x=-3*pi:0.4:3*pi; % Диапазон 6*pi
y=2*sin(x);
plot(x,y, '-db');
grid on
hold off
% ВЫВОД: Общее окно автоматически масштабируется функцией
% с наибольшим диапазоном по X и Y
```

3.1.2.6.2

```
% Построим графики в замороженном (общем) окне.
% Порядок построения с убывающим диапазоном по X
clear all;
close all;
hold on; % Заморозить окно
x=-3*pi:0.4:3*pi; % Диапазон 6*pi
y=sin(x)./x;
plot(x,y, '-*r');
input('Для продолжения нажмите Enter ...')
x=-2*pi:0.2:2*pi; % Диапазон 4*pi
y=2*sin(x);
plot(x,y, '-db');
grid on
hold off
% ВЫВОД: Общее окно автоматически масштабируется функцией
% с наибольшим диапазоном по X и Y
```

3.2. FPLOТ - построение графиков, заданных аналитически.

Функция FPLOТ предназначена для построения графиков по их аналитическому описанию. Другими словами, вместо табличной функции, которая задается таблицами или векторами, для FPLOТ функция графика задается текстовой строкой, которая содержит аналитическую запись функции. Особенностью FPLOТ является ее самостоятельный выбор точек для аргумента, причем дистанция между такими точками определяется параметром толерантность. Кроме того FPLOТ может использоваться как генератор табличных функций по их аналитическому виду.

3.2.1. Функция FPLOТ. Синтаксис

- FPLOТ (FUN, LIMS) отображает функцию FUN, заданную в символьном виде. Для области определения оси X, указывается параметр LIMS = [XMIN XMAX]. Использование LIMS = [XMIN XMAX YMIN YMAX] позволяет также контролировать границы отображения FUN на оси Y.

Примечание. При работе FPLOТ функция FUN (X) создает вектор строки для каждого элемента из вектора X, который она вначале подготавливает исходя из диапазона [XMIN XMAX]. Например, если FUN возвращает несколько значений для одного аргумента [f1 (x), f2 (x), f3 (x)], то при вводе вектора аргументов функция должна возвращать матрицу:

```
f1 (x1) f2 (x1) f3 (x1)
f1 (x2) f2 (x2) f3 (x2)
```

- FPLOТ (FUN, LIMS, TOL), где TOL <1 указывает значение толерантности (специальной относительной погрешности). Данный параметр определяет детальность предварительного вычисления точек отображаемой функции. По умолчанию параметр TOL равен 2e-3, то есть 0,2 процента.

Примечание. Параметр толерантность определяет выбор приращения для очередной точки значения аргумента. Выбор осуществляется таким образом, чтобы очередное значение функции отличалось от предшествующего не менее чем на величину относительной погрешности.

- FPLOТ (FUN, LIMS, N) с параметром N >= 1 требует отображения функции с минимумом N + 1 точек. Значение по умолчанию N равно 1. Максимальный размер шага ограничен (1 / N) * (XMAX-XMIN).

Примечание. Параметр N проявляет себя только, если число затребуемых точек N + 1 больше, чем число точек определенное исходя из параметра толерантность.

- FPLOТ (FUN, LIMS, «LineSpec») с заданной спецификацией линии (см. . PLOT)
- FPLOТ (FUN, LIMS, ...) принимает комбинации необязательных аргументов TOL, N и 'LineSpec', в любом порядке.

- $[X, Y] = \text{FPLOT}(\text{FUN}, \text{LIMS}, \dots)$ возвращает X и Y такие, что $Y = \text{FUN}(X)$, при этом на экран график **не** выводится.
- $\text{FPLOT}(\text{FUN}, \text{LIMS}, \text{TOL}, \text{N}, \text{'LineSpec'}, \text{P1}, \text{P2}, \dots)$ позволяет использовать специальные параметры P1 , P2 и т. д. Такие параметры непосредственно передаются функции FUN : $Y = \text{FUN}(X, \text{P1}, \text{P2}, \dots)$. Чтобы использовать значения по умолчанию для TOL , N или «LineSpec», необходимо заменить их пустыми матрицами ($[]$).

3.2.2. Функция *FPLOT*. Примеры

```
% Рисование графика для аналитической функции
help fplot
```

3.2.2.1.

```
% Построим графики в замороженном (общем) окне.
% Порядок построения с возрастающим диапазоном по X
clear all;
close all;
hold on;                % Заморозить окно
fplot('sin(x)/x',[-2*pi 2*pi], '-r'); % Диапазон 4*pi
input('Для продолжения нажмите Enter ...')
fplot('2*sin(x)',[-3*pi 3*pi]);      % Диапазон 6*pi
grid on
hold off
% ВЫВОД: Общее окно автоматически масштабируется функцией
% с наибольшим диапазоном по X и Y
```

3.2.2.2

```
% Построим графики в замороженном (общем) окне.
% Порядок построения с убывающим диапазоном по X
clear all;
close all;
hold on;                % Заморозить окно
fplot('2*sin(x)',[-3*pi 3*pi]);      % Диапазон 6*pi
input('Для продолжения нажмите Enter ...')
fplot('sin(x)/x',[-2*pi 2*pi], '-r'); % Диапазон 4*pi
grid on
hold off
% ВЫВОД: Общее окно автоматически масштабируется функцией
% с наибольшим диапазоном по Y и последним диапазоном по X
```

3.2.2.3

```
% Проверим предшествующий вывод на примере функций
% с чередующимися диапазонами по X
close all;
hold on;
grid on;
fplot('log(x)',[1/4 8*pi], '-g');
input('Для продолжения нажмите Enter ...')
fplot('exp(x)',[-2*pi pi/4], '-c');
input('Для продолжения нажмите Enter ...')
fplot('0.5*sin(x)^2',[-2*pi 2*pi]);
input('Для продолжения нажмите Enter ...')
fplot('2*sin(x)/x',[-3*pi 3*pi], '-r');
hold off
% ВЫВОД: Общее окно автоматически масштабируется функцией
% с наибольшим диапазоном по Y и последним диапазоном по X
```

3.2.2.4

```

% Использование функции fplot для создания векторов аргумента и функции
clear all;
close all;
[X,Y]=fplot('[2*sin(x)]',[ -3*pi 3*pi]); % Диапазон 6*pi
input('Для продолжения нажмите Enter ...')
plot(X,Y); % Диапазон 6*pi
grid on
% ВЫВОД: При использовании функции fplot для создания векторов
% аргумента и функции вывод графика отсутствует.

```

3.2.2.5

```

% Использование функции с различными значениями TOL и N
clear all;
close all;
hold on;
% TOL = 1e-1
fplot('[2*sin(x)]',[ -3*pi 3*pi], 1e-1, 'rd-'); % Диапазон 6*pi
% TOL = 1e-1, N = 40
fplot('[2*sin(x)]',[ -3*pi 3*pi], 1e-1, 40, 'go-'); % Диапазон 6*pi
hold off;
grid on;

```

3.2.2.6

```

% Использование литералов при рисовании графиков
% аналитических функций
clear all;
close all;
hold on;
s1='[log(0.5*(1+sin(x)^2))]' ;
fplot(s1,[- 2*pi 2*pi], '-r');
s1='[2*abs(sin(x)/x)]' ;
fplot(s1,[-4*pi 4*pi]);
grid on;
hold off

```

3.2.2.7.

```

% Использование параметров при рисовании графиков
% аналитических функций
clear all;
close all;
% TOL = [], N = [], S= [] , Параметр P1
P1=10;
fplot('[(sin(x*P1)+1)]',[ -3*pi 3*pi], [],[],[], P1);
grid on;

```

3.3. Функция AXIS

3.3.1. Функция *AXIS*. Синтаксис.

Функция *AXIS* предназначена для масштабирования и установки внешнего вида осей координат. Функция имеет большое количество синтаксических форм и параметров. Многие применения являются весьма специфическими. По этой причине ограничимся только теми синтаксическими формами, которые наиболее часто применяются на начальных этапах освоения MATLAB.

- *AXIS* ([XMIN XMAX YMIN YMAX]) устанавливает масштабирование для осей *x* и *y* на текущем участке.
- *AXIS* ([XMIN XMAX YMIN YMAX ZMIN ZMAX]) устанавливает масштабирование для *x*-, *y*- и *z*-осей на текущем трехмерном графике.

- **AXIS AUTO** возвращает масштабирование оси по умолчанию, автоматически для каждого измерения выбираются «хорошие» лимиты в экстендах всех линий, поверхностей, патчей и изображений.

3.3.2. Функция *AXIS*. Примеры

3.3.2.1.

```
% Создать пустое окно с заданного математического размера
help axis
close all;
axis([-4 4 -2 2]);
```

3.3.2.2.

```
% Управление математическим размером
% окна просмотра в окне графика
close all;
x = 0:.025:pi/2;
plot(x,tan(x),'-ro')
input('Для продолжения нажмите Enter ...')
axis([0 pi/2 0 5]);
input('Для продолжения нажмите Enter ...')
axis([0 pi/4 0 2]);
input('Для продолжения нажмите Enter ...')
axis([0 pi/2 0 20]);
input('Для продолжения нажмите Enter ...')
close all;
```

3.3.2.3.

```
% Ручное управление окном графиков и его масштабами
close all;
axis manual;
axis([-4 4 -2 2]); % Установить размер окна
hold on; % Заморозить окно и его вертикальный масштаб
fplot('[3*sin(x)/x]', [-2*pi 2*pi], '-r');
fplot('[abs(sin(x))]', [-2*pi 2*pi]);
input('Для продолжения нажмите Enter ...')
axis([-4 4 -2 6]);
input('Для продолжения нажмите Enter ...')
hold off;
input('Для продолжения нажмите Enter ...')
axis auto;
```

3.4. EZplot. Простой в использовании плоттер.

3.4.1. Функция *EZplot*. Синтаксис:

- `EZplot (f)`
- `EZplot (f, [min,max])`
- `EZplot (f, [Xmin, Xmax, Ymin, Ymax])`

Где $f(x)$ – функция заданная в аналитическом виде

- `EZplot (x, y)`
- `EZplot (x, y, [tmin,tmax])`

Где x, y – функции заданные в аналитическом виде

3.4.2. Функция *EZplot*. Описание

`ezplot(f)` отображает выражение $f = f(x)$ по умолчанию: $-2\pi < x < 2\pi$.
`ezplot(f, [min, max])` графики $f = f(x)$ над областью аргумента: $\min < x < \max$.

Для неявно определенных функций двух переменных $f = f(x, y)$:

`ezplot(f)` отображает графики $f(x, y) = 0$ над областью аргумента по умолчанию $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$.

`ezplot(f, [xmin, xmax, ymin, ymax])` графики $f(x, y) = 0$ над областью аргумента $x_{\min} < x < x_{\max}$ и $y_{\min} < y < y_{\max}$.

`ezplot(f, [min, max])` графики $f(x, y) = 0$ по $\min < x < \max$ и $\min < y < \max$.

Если f является функцией переменных u и v (а не x и y), то конечные точки для области аргумента u_{\min} , u_{\max} , v_{\min} и v_{\max} сортируются по алфавиту. Таким образом, `ezplot('u ^ 2 - v ^ 2 - 1', [-3,2, -2,3])` отображает $u^2 - v^2 - 1 = 0$ над $-3 < u < 2$, $-2 < v < 3$,

`ezplot(x, y)` изображает параметрически определенную планарную кривую $x = x(t)$ и $y = y(t)$ для области аргумента $0 < t < 2$.

`ezplot(x, y, [tmin, tmax])` графики $x = x(t)$ и $y = y(t)$ по $t_{\min} < t < t_{\max}$.

3.4.3. Функция *EZplot*. Примеры

3.4.3.1.

```
% Простейший пример
ezplot('sin(x)')
```

3.4.3.2.

```
% Простейший пример с определенным диапазоном
ezplot('sin(x)', [-2 * pi 2 * pi])
```

3.4.3.3.

```
% В этом примере отображается неявно определенная
% функция,  $x:2 - y:4 = 0$  для области аргумента  $[-2, 2]$ :
ezplot('x ^ 2 - y ^ 4', [-2, 2])
```

3.4.3.4.

```
% Примеры аналогичные 3.4.3.3.
ezplot('x ^ 2 - y ^ 2')
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'sin(y)', [-2 * pi 2 * pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'cos(y)', [-2 * pi 2 * pi])
```

3.4.3.5.

```
ezplot('sin(x)/x')
```

3.4.3.6.

```
% Способ управления окном отображения
close all;
ezplot('sin(x)/x', [-16*pi 16*pi])
axis([-16*pi 16*pi -0.4 1])
```

3.4.3.7.

```

% Функции двух независимых переменных X и Y
% с одинаковым диапазоном (Например: фигуры Лиссажу).
ezplot('sin(x)', 'sin(y)', [-2*pi 2*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'cos(y)', [-2*pi 2*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'cos(2*y)', [-2*pi 2*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'cos(3*y)', [-2*pi 2*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x+pi/4)', 'cos(3*y)', [-2*pi 2*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(x)', 'cos(4*y)', [-2*pi 2*pi])

```

3.4.3.8.

```

% Функции двух переменных A и B, которые
% вычисляются на основе независимой переменной (t)
% с заданным диапазоном. . По умолчанию
% диапазон определяется как [-2*pi, 2*pi]
ezplot('t*cos(t)', 't*sin(t)', [0,4*pi])
input('Для продолжения нажмите Enter ...')
ezplot('sin(3*t)*cos(t)', 'sin(3*t)*sin(t)', [0,pi])

```

3.4.3.9.

```

% Функции переменных: X (независимая) и переменной Y,
% которая связана с значениями X уравнением Y(X)=0
ezplot('x^2 - y')
input('Для продолжения нажмите Enter ...')
ezplot('x^2 - y', [-1 20])
input('Для продолжения нажмите Enter ...')
ezplot('x^2 - y^2 - 1')
input('Для продолжения нажмите Enter ...')
ezplot('x^3 + y^3 - 5*x*y + 1/5', [-3,3])
input('Для продолжения нажмите Enter ...')
ezplot('1/y-log(y)+log(-1+y)+x - 1')

```

3.5. EZPOLAR. Простой в использовании полярный координатный плоттер

3.5.1. Функция EZPOLAR. Синтаксис

- ezpolar (f)
- ezpolar (f, [a, B])

Где f – функция заданная в виде строки

3.5.2. Функция EZPOLAR. Описание

- ezpolar (f) отображает полярную кривую $\rho = f(\theta)$ для области аргумента (по умолчанию $0 < \theta < 2\pi$).
- ezpolar (f, [a, b]) графики f для $a < \theta < b$.

3.5.3. Функция EZPOLAR. Примеры

3.5.3.1.

```

% В этом примере создается полярный график

```

```
% функции 1 + cos (t) по области [0, 2 * pi]:
ezpolar ( '1 + cos (t)')
```

3.5.3.2.

```
% Функции, которая отображается кончиком вектора длиной вычисляемой
% литералом и углом поворота (t) в заданном диапазоне. По умолчанию
% диапазон определяется как [0, 2*pi]
ezpolar('sin(t)',[0, 2*pi])
input('Для продолжения нажмите Enter ...')
ezpolar('1+cos(t)')
input('Для продолжения нажмите Enter ...')
ezpolar('1+cos(t)',[0, pi])
input('Для продолжения нажмите Enter ...')
ezpolar('sin(t)+cos(t)',[0, pi])
input('Для продолжения нажмите Enter ...')
ezpolar('sin(t)/t', [-6*pi,6*pi])
input('Для продолжения нажмите Enter ...')
ezpolar('sin(2*t)*cos(3*t)',[0,pi])
input('Для продолжения нажмите Enter ...')
ezpolar('sin(tan(t))')
input('Для продолжения нажмите Enter ...')
ezpolar('cos(5*t)')
input('Для продолжения нажмите Enter ...')
ezpolar('cos(8*t)')
```

3.6. САМОСТОЯТЕЛЬНАЯ РАБОТА

3.6.1. Выполните все примеры из теоретической части лабораторной работы и представьте полученные графики в виде отчета.

3.6.2. Используя в составе MATLAB подсистему HELP, самостоятельно опишите синтаксис и описание функции POLAR.

3.6.3. Сравните функции POLAR и PLOT.

3.6.4. Выполните следующие примеры:

3.6.4.1.

```
clear all;
close all;
x = 0:.01:2*pi;
polar(x,sin(2*x),'--r')
```

3.6.4.2.

```
% Прорисовка Функций в полярных координатах
clear all;
close all;
% An M-file script to produce          % Comment lines
% "flower petal" plots
theta = -pi:0.01:pi;                  % Computations
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for k = 1:4
    polar(theta,rho(k,:))              % Graphics output
    pause
end;
```

3.6.5. Самостоятельно изучите функции для построения 3D – графиков. Для этого используйте ПРИЛОЖЕНИЕ (Б) 3D-ГРАФИКИ В MATLAB

4. УСЛОВНЫЕ ВЫРАЖЕНИЯ, ОПЕРАТОРЫ ВЕТВЛЕНИЯ И ЦИКЛЫ (ЛАБ. РАБОТА 4)

4.1. Условные выражения (conditional expression)

Условные выражения предназначены для ответов на вопросы заданные с помощью операций (инструкций) отношений и логических операций (инструкций).

4.1.1. Операторы отношений (Relational operations)

- Операторы (инструкции) отношений всегда применяются к двум операндам, которые представлены в виде арифметических значений или отдельных текстовых символов. Результатом операции является булево значение **true** или **false**.
- Операторы отношений могут также применяться к матрицам одинакового размера и размерности, элементами (ячейками) которых, являются арифметические значения или отдельные текстовые символы. В этом случае результатом операции будет матрица, полученная поэлементным вычислением со значениями true или false в соответствующих элементах.
- Применение *всех* операторов отношений для комплексных чисел возможно только в отношении их реальных или мнимых частей. Для этого используются функции выделения реальной `real(x)` или мнимой `imag(x)` части таких чисел. Непосредственное сравнение двух комплексных чисел допускается только для операторов `==` (равно) и `~=` (не равно).

Синтаксис и семантика операторов отношений для двух элементов (ячеек) представлены в таблице №1.

Таблица 1

Операция	Описание результата операции
$A < B$	Истина (1 или true), если левый операнд <i>A строго меньше</i> правого операнда <i>B</i> , иначе ложь (0 или false).
$A > B$	Истина (1 или true), если левый операнд <i>A строго больше</i> правого операнда <i>B</i> , иначе ложь (0 или false).
$A \leq B$	Истина (1 или true), если левый операнд <i>A меньше или равен</i> правому операнду <i>B</i> , иначе ложь (0 или false).
$A \geq B$	Истина (1 или true), если левый операнд <i>A больше или равен</i> правому операнду <i>B</i> , иначе ложь (0 или false).
$A == B$	Истина (1 или true), если левый операнд <i>A строго равен</i> правому операнду <i>B</i> , иначе ложь (0 или false).
$A ~= B$	Истина (1 или true), если левый операнд <i>A строго не равен</i> правому операнду <i>B</i> , иначе ложь (0 или false).

4.1.2. Примеры операторов отношений

4.1.2.1.

```
% Пример для сравнения предварительно
% вычисленных арифметических выражений X1 и X2
```

```
X1 = 2.34;
X2 = 2;
(X1 > X2)
% Результат:
```

```

        ans = 1
(X1 < X2)
% Результат:
        ans = 0

```

4.1.2.2.

```

% Пример для сравнения предварительно
% вычисленных комплексных выражений X1 и X2
% по их реальной и мнимой части.

```

```

X1 = 1 + 2.34i;
X2 = 1 + 2i;
real(X1) == real(X2)
% Результат:
        ans = 1

```

```

imag(X1) == imag(X2)
% Результат:
        ans = 0

```

4.1.2.3.

```

% (непосредственное сравнение).
% Пример для сравнения предварительно
% вычисленных комплексных выражений X1 и X2

```

```

X1 = 1 + 2.34i;
X2 = 1 + 2i;
X1 == X2
% Результат:
        ans = 0

```

```

X1 ~= X2
% Результат:
        ans = 1

```

4.1.2.4.

```

% Сравнение двух символов

```

```

X1='a';
X2='A';
X1 == X2
% Результат:
        ans = 0

```

```

X1 < X2
% Результат:
        ans = 0

```

```

X1 > X2
% Результат:
        ans = 1

```

4.1.2.5.

```

% Сравнение двух арифметических массивов

```

```

X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
X1 == X2
% Результат:
ans =
     1     1     0
     1     0     1

```

```

X1 < X2

```

```
% Результат:
ans =
     0     0     1
     0     1     0
```

4.1.2.6.

```
% Поэлементное сравнение двух текстовых строк
X1='qwerty';
X2='QwErTy';
X1 == X2
% Результат:
ans = 0     1     0     1     0     1
```

4.1.2.7.

```
% Полное сравнение двух текстовых строк
X1='qwerty';
X2='QwErTy';
strcmp(X1,X2)
% Результат:
ans = 0

X1='qwerty';
X2='qwerty';
strcmp(X1,X2)
% Результат:
ans = 1
```

4.1.2.8.

```
% Полное сравнение двух текстовых строк или матриц
% одинакового размера и размерности с помощью функции isequal

X1='qwerty';
X2='QwErTy';
isequal(X1,X2)
% Результат:
ans = 0

X1='qwerty';
X2='qwerty';
isequal(X1,X2)
% Результат:
ans = 1

X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
isequal(X1,X2)
% Результат:
ans = 0

X1=[1 2 3; 4 5 6];
X2=[1 2 3; 4 5 6];
isequal(X1,X2)
% Результат:
ans = 1
```

4.1.3. Вычисление отношений алгоритмическим путем

4.1.3.1.

```
% Полное алгоритмическое сравнение двух текстовых строк
```

```

X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
% Вычислить произведение всех элементов Y1 посредством
% явного перечисления индекса в цикле
k=1;
for n = 1 : length(Y1)
    k=k * Y1(n);
end;
% Отобразить результат сравнения
if k == 0
    disp('Строки X1 и X2 НЕ равны');
end;

```

4.1.3.2.

% Полное алгоритмическое сравнение двух текстовых строк

```

X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
% Вычислить произведение всех элементов Y1 посредством
% косвенного перечисления индекса в цикле
n =1;
for k = Y1
    n = n * k;
end;
% Отобразить результат сравнения
if n == 0
    disp('Строки X1 и X2 НЕ равны');
end;

```

4.1.3.3.

% Полное алгоритмическое сравнение двух текстовых строк
% с использованием функций **prod** (перемножить элементы массива)
% и **double** (преобразовать элементы массива к арифметическому
% типу).

```

X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
% Вычислить произведение всех элементов Y1
n = prod(double(Y1));
% Отобразить результат сравнения
if n == 0
    disp('Строки X1 и X2 НЕ равны');
end;

```

4.1.4. Логические операторы и операции (Logical operations)

Логические операции «И», «ИЛИ», «НЕ» оперируют с *вычисленными отношениями* и (или) *логическими переменными*, которые принимают значения ИСТИНА (**true** или **1**) или ЛОЖЬ (**false** или **0**). Результатом логической операции также является булевское значение **true** или **false**.

Логическая операция «И».

Логическая операция **И** выполняет *логическое умножение двух переменных*, каждая из которых может содержать только булевское значение true или false. Варианты результатов операции «И» представлены в следующей таблице:

X Y	0	1
0	0	0
1	0	1

Другими словами результат операции **И** равен **true** только когда оба операнда X и Y равны **true**.

В **MATLAB** операция **И** обозначается символом **&** для матриц (поэлементное **И**) или **&&** для скалярных величин.

4.1.4.1.

```
% Явное вычисление

true & true           % матричная форма
% Результат:
ans = 1

true && true          % скалярная форма
% Результат:
ans = 1

true & false
% Результат:
ans = 0

false & true
% Результат:
ans = 0
```

4.1.4.2.

```
% Операция И для двух логических матриц Y1 и Y2

X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 = 1     1     0
     1     0     1

Y2=(X1~=X2)
% Результат:
Y2 = 0     0     1
     0     1     0

Z1=(Y1 & Y2)
% Результат:
Z1 = 0     0     0
     0     0     0
```

Логическая операция «ИЛИ».

Логическая операция **ИЛИ** выполняет *логическое сложение двух переменных*, каждая из которых может содержать только булевское значение true или false. Варианты результатов операции «ИЛИ» представлены в следующей таблице:

X \ Y	0	1
0	0	1
1	1	1

Другими словами результат операции **ИЛИ** равен **false** только когда оба операнда X и Y равны **false**.

В **MATLAB** операция **ИЛИ** обозначается символом | для матриц (поэлементное **ИЛИ**) или || для скалярных величин.

4.1.4.3.

```
% Явное вычисление

true | false           % матричная форма
% Результат:
ans = 1

true || false          % скалярная форма
% Результат:
ans = 1

true | true
% Результат:
ans = 1

false | false
% Результат:
ans = 0
```

4.1.4.4.

```
% Операция ИЛИ для двух логических матриц Y1 и Y2

X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 = 1     1     0
      1     0     1

Y2=(X1~=X2)
% Результат:
Y2 = 0     0     1
      0     1     0

Z1=(Y1 | Y2)
% Результат:
Z1 = 1     1     1
      1     1     1
```

Логическая операция «НЕ».

Логическая операция **НЕ** выполняет *инвертирование значения переменной*, которая может содержать только булевское значение true или false. Варианты результатов операции «НЕ» представлены в следующей таблице:

X	НЕ(X)
0	1
1	0

Другими словами результат операции **НЕ** равен **false** только когда операнд X равен **true**.

В **MATLAB** операция **НЕ** обозначается символом \sim как для матриц (поэлементное **НЕ**) так и для скалярных величин.

4.1.4.5.

```
% Явное вычисление
```

```
~true
```

```
% Результат:
```

```
ans = 0
```

```
~false
```

```
% Результат:
```

```
ans = 1
```

4.1.4.6.

```
% Операция НЕ для логической матрицы Y1
```

```
X1=[1 2 3; 4 5 6];
```

```
X2=[1 2 8; 4 6 6];
```

```
Y1=(X1==X2)
```

```
% Результат:
```

```
Y1 = 1     1     0
      1     0     1
```

```
Z1=~Y1
```

```
% Результат:
```

```
Z1 = 0     0     1
      0     1     0
```

Логическая операция «ИСКЛЮЧАЮЩЕЕ ИЛИ».

Логическая операция **ИСКЛЮЧАЮЩЕЕ ИЛИ** выполняет *двоичное сложение без переноса для двух переменных*, каждая из которых может содержать только булево значение true или false. Варианты результатов операции **ИСКЛЮЧАЮЩЕЕ ИЛИ** представлены в следующей таблице:

X \ Y	0	1
0	0	1
1	1	0

Другими словами результат операции **ИСКЛЮЧАЮЩЕЕ ИЛИ** равен **false** только когда оба операнда X и Y равны **false** или **true**.

В **MATLAB** операция **ИСКЛЮЧАЮЩЕЕ ИЛИ** выполняется только с помощью функции $Z = \text{xor}(X,Y)$ как для матриц (поэлементное **ИСКЛЮЧАЮЩЕЕ ИЛИ**), так и для скалярных величин.

4.1.4.7.

```

% Явное вычисление

xor(true,false)
% Результат:
ans = 1

xor(true,true)
% Результат:
ans = 0

xor(false,false)
% Результат:
ans = 0

```

4.1.4.8.

```

% Операция ИСКЛЮЧАЮЩЕЕ ИЛИ для двух логических
% матриц Y1 и Y2

X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 = 1     1     0
     1     0     1

Y2=(X1~=X2)
% Результат:
Y2 = 0     0     1
     0     1     0

Z1=xor(Y1, Y2)
% Результат:
Z1 = 1     1     1
     1     1     1

```

4.1.5. Некоторые законы для логических операций**Законы поглощения**

$$X \text{ И } (X \text{ ИЛИ } Y) = X$$

$$X \text{ ИЛИ } (X \text{ И } Y) = X$$

Штрих Шеффера — бинарная логическая операция, булева функция над двумя переменными. Введена Генри Шеффером в 1913 г.

Штрих Шеффера, эквивалентен операции **И-НЕ**

Стрелка Пирса — бинарная логическая операция, булева функция над двумя переменными. Введена Чарльзом Пирсом в 1880—1881 г.г.

Стрелка Пирса, эквивалентна операции **ИЛИ-НЕ**

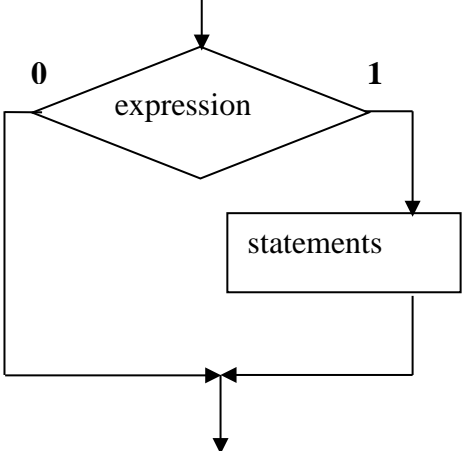
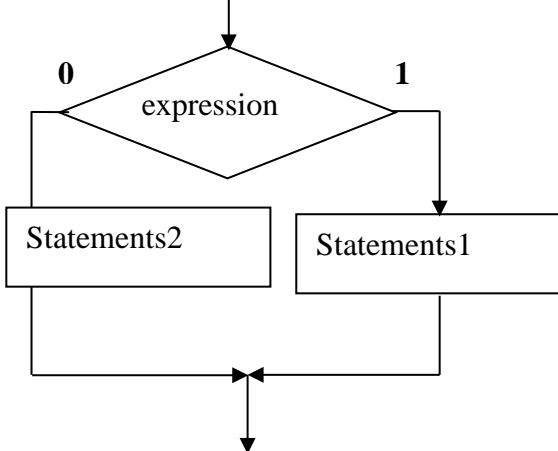
Законы де Моргана

$$\text{НЕ}(X \text{ ИЛИ } Y) = \text{НЕ}(X) \text{ И } \text{НЕ}(Y)$$

$$\text{НЕ}(X \text{ И } Y) = \text{НЕ}(X) \text{ ИЛИ } \text{НЕ}(Y)$$

4.2. Условные операторы (инструкции)

4.2.1. Простой условный оператор *if*

Простой условный оператор с одной ветвью по true	Простой условный оператор с двумя ветвями по true и false
 <p><u>Синтаксис:</u></p> <pre>if expression statements end</pre>	 <p><u>Синтаксис:</u></p> <pre>if expression statements1 else statements2 end</pre>

4.2.1.1.

```
% Условный оператор с одной ветвью по true
x=pi/7;
if sin(x) < cos(x)
    disp('Для заданного x верно sin(x) < cos(x)');
end;
```

4.2.1.2.

```
% Условный оператор с ветвями true и false
x=pi/7;
if sin(x) >= cos(x)
    disp('Для заданного x верно sin(x) >= cos(x)');
else
    disp('Для заданного x верно sin(x) < cos(x)');
end;
```

4.2.1.3.

```
% Проверка на попадание заданного числа X
% в диапазоны X1..X2 и X3..X4
clc;
clear all;
x1=1; x2=3; x3=5; x4=7;
x=input('Введите некоторое число >>');
y1=(x>=x1)&(x<=x2);
y2=(x>=x3)&(x<=x4);
y= y1|y2;
if y
```

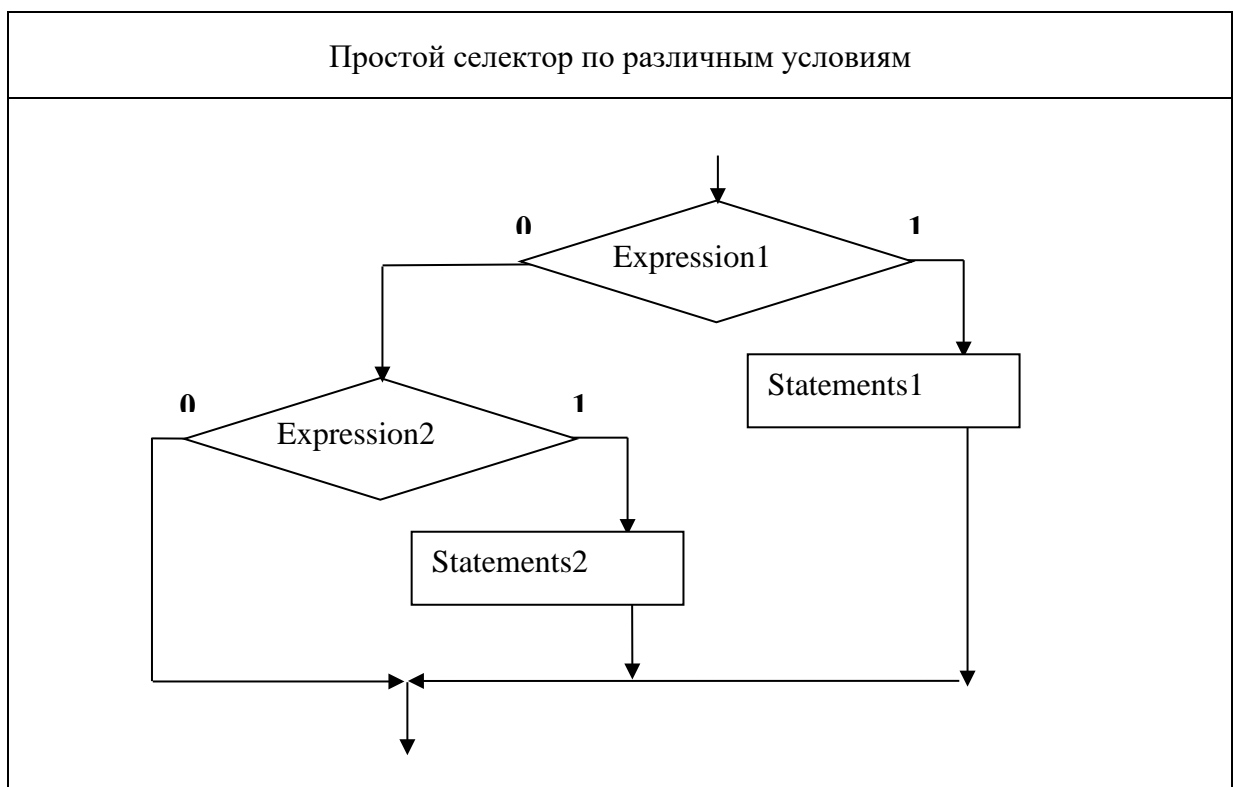
```

disp('число в одном из диапазонов');
else
disp('число ВНЕ диапазонов');
end;

```

4.2.2. Селектор по различным условиям *if..elseif*

Простой селектор по различным условиям (синтаксис)	Селектор по различным условиям с блоком else (синтаксис)
<p><u>Синтаксис:</u></p> <pre> if expression1 statements1 elseif expression2 statements2 . . . end </pre>	<p><u>Синтаксис:</u></p> <pre> if expression1 statements1 elseif expression2 statements2 . . . else statements3 end </pre>



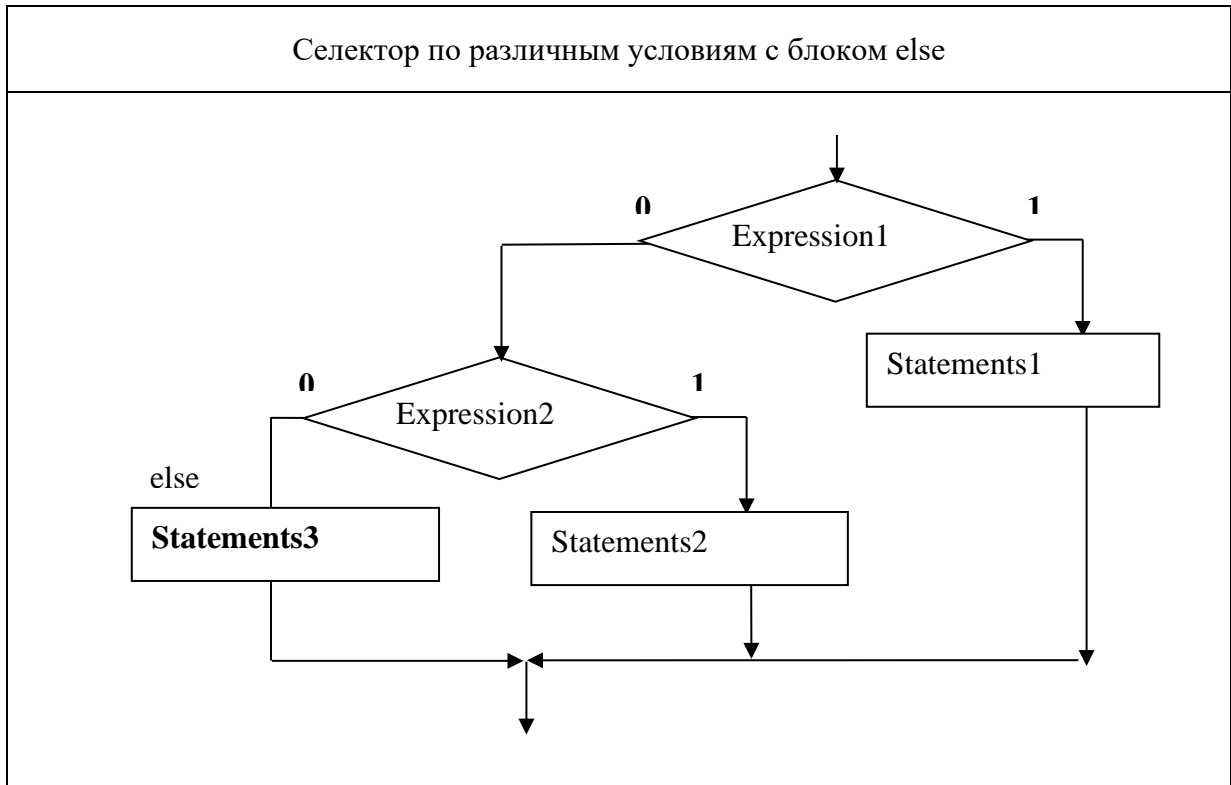
4.2.2.1.

```

% Простой селектор по различным условиям
x1='A';
x2='B';
if (x1 == 'A') & (x2 == 'B')
    disp('Заданы символы А и В');
elseif (x1 ~= 'A') & (x2 == 'B')
    disp('Задан только символ А');

```

```
elseif (x1 == 'A') & (x2 ~= 'B')
    disp('Задан только символ B');
end;
```



4.2.2.2.

```
% Селектор по различным условиям с блоком else
x1='C';
x2='D';
if (x1 == 'A') & (x2 == 'B')
    disp('Заданы символы A и B');
elseif (x1 ~= 'A') & (x2 == 'B')
    disp('Задан только символ A');
elseif (x1 == 'A') & (x2 ~= 'B')
    disp('Задан только символ B');
else
    disp('Символы A и B НЕ заданы');
end;
```

Пример с if	Пример с if и elseif
<pre> if A x = a else if B x = b else if C x = c else x = d end end end end </pre>	<pre> if A x = a elseif B x = b elseif C x = c else x = d end </pre>

4.2.3. Селектор по различным значениям

селектор по различным значениям (синтаксис)

```

switch switch_expr
  case case_expr
    statement,...,statement
  case {case_expr1,case_expr2,case_expr3,...}
    statement,...,statement
  ...
  otherwise
    statement,...,statement
end

```

4.2.3.1.

```

% селектор по различным значениям строк
str1 = 'Bilinear';
switch lower(str1)
  case {'linear','bilinear'}
    disp('Заявлено имя linear методов');
  case 'cubic'
    disp('Заявлено имя cubic метода');
  case 'nearest'
    disp('Заявлено имя nearest метода');
  otherwise
    disp(' Заявлено имя Unkownn метода. ');
end;

```

4.2.3.2.

```

% селектор по различным значениям диапазонов
x1 = 12;
switch x1
  case 1
    disp('Обнаружено x1 = 1');
  case {2,3,4,5,6,7,8}
    disp('Обнаружено x1 в диапазоне [2:1:8]');
  case {9,10,11,12,13,14,15,16}
    disp('Обнаружено x1 в диапазоне [9:1:16]');
  otherwise
    disp(' x1 ВНЕ областей обнаружения. ');
end;

```

4.3. Циклы

4.3.1. Цикл FOR

Синтаксис

```

for variable = expression
  statements
end

```

Общий формат

```

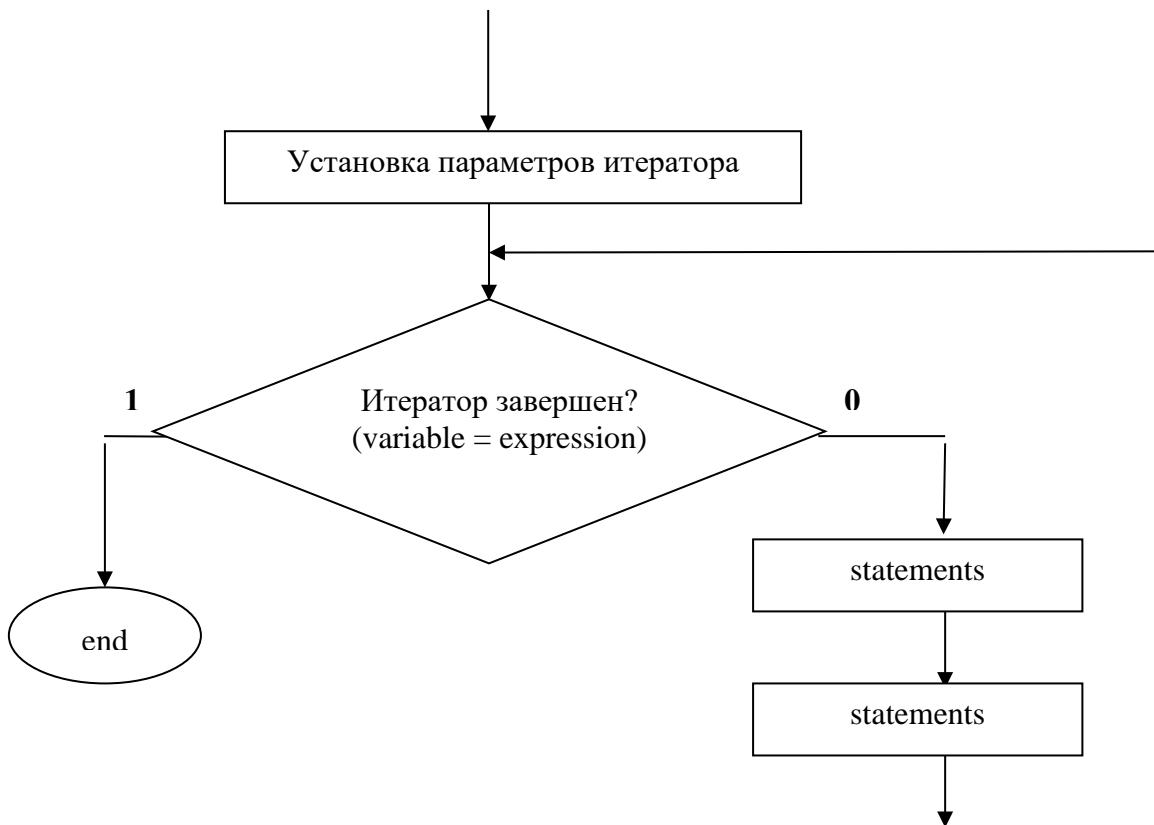
for variable = expression
    statement
    ...
    statement
end

```

Цикл предназначен повторять инструкции (statements) определенное количество раз. Для этого в заголовке цикла используется конструкция, которая в современных языках программирования обобщена термином - итератор:

```
variable = expression
```

В MATLAB эта конструкция определяет переменную (variable), которая может использоваться в теле цикла, а также определяет собой количество повторений тела цикла.



Столбцы выражения (expression), даже если они состоят всего из одного элемента, сохраняются по одному в переменной (variable) и доступны для каждой инструкции (statements) в теле цикла. Заметим, что на практике выражение (expression) почти всегда имеет вид scalar, и в этом случае его столбцы являются просто скалярами. Область действия оператора for всегда заканчивается соответствующим end.

Смотрите также функции: size, rand:

help size

help rand

4.3.1.1.

```
% Вложенные циклы с единичным шагом. (Матрица Гильберта):
```

```
k = 8;
a = zeros(k,k); % матрица предварительного выделения
for m = 1:k
for n = 1:k
    a(m,n) = 1/(m+n -1);
end;
end;
plot(a);
```

4.3.1.2.

```
% Цикл for с заданным диапазоном 1.0 .. 0.0 и шагом -0.1
a=[];
ind = 1;
for S = 1.0: -0.1: 0.0
    a(ind) = S;
    ind = ind + 1;
end
plot(a);
```

4.3.1.3.

```
% Последовательная выборка в тело цикла элементов из вектора строки
B = [3, 4, 5];
for S = B
    S
end;
```

4.3.1.4.

```
% Выборка в тело цикла вектора столбца (полностью)
B = [3; 4; 5];
for S = B
    S
end;
```

4.3.1.5.

```
% Выборка в тело цикла очередного вектора столбца из двумерной матрицы
A=[[1,2,3]; [4,5,6]; [7,8,9]]
for S = A
    S
end;

% Эквивалентная форма примера (4.3.1.5)
A=[[1,2,3]; [4,5,6]; [7,8,9]]
[d1,d2] = size(A)
for k = 1:d2
    V = A(:,k)
end
% Эквивалентная форма примера (5)
A=[[1,2,3]; [4,5,6]; [7,8,9]]
[d1,d2] = size(A)
for k = 1:d2, V = A(:,k)
    V
end;
```

4.3.1.6.

```
% Последовательная установка значений sqrt(m)
% вместо единиц в единичных n-векторах:
n=5;
m = 2;
for B = eye (n)
    B.*sqrt(m)
```

```
end;
```

4.3.1.7.

```
% Публикация столбцов в слоях 3D матрицы
A=rand(2,2,2)
for S = A
    S
end;
```

4.3.2. Цикл FOR. Примеры

Смотрите также функции управления циклами: **break**, **continue**, **isempty**:

help break

help continue

help isempty

4.3.2.1.

```
% Создание и инициализация вектора строки
% Вариант 1
for m=1:10
    x(m)=m;
end

% Вариант 2
for m=1:10
    x(1,m)=m;
end
```

4.3.2.2.

```
% Создание и инициализация вектора столбца
for m=1:10
    y(m,1)=m;
end
```

4.3.2.3.

```
% Инициализация вектора строки
x=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
for m=1:length(x)
    x(m)=m;
end
```

4.3.2.3.

```
% Суммирование элементов вектора строки
x=rand(1,5);
sm = 0;
for m=1:length(x)
    sm=sm+x(m);
end
```

4.3.2.3.

```
% Суммирование элементов вектора столбца
x=rand(1,5); % Вектор строка
y=x'         % Вектор столбец
sm = 0;
for m=1:length(y)
    sm=sm+y(m);
end
```

4.3.2.4.

```
% Создание и инициализация двухмерного
% массива в режиме диалога
r = input('Введите число строк или по умолчанию просто Enter >> ');
if isempty(r)
    r = 10; % Значение по умолчанию
end; ...
c = input('Введите число столбцов или по умолчанию просто Enter >> ');
if isempty(c)
    c = 10;
end; ...
for n=1:r
    for m=1:c
        M1(n,m) = 0;
    end
end
end
```

4.3.2.5.

```
% Поиск максимального элемента в массиве
% Вначале выполним копию массива Y в массив WM
WM=input('Input 2D-array name (example: Y) >>');
mx =WM(1,1); ir=1; ic=1;
for r=1:size(WM,1)
    for c=1:size(WM,2)
        if WM(r,c) > mx
            mx =WM(r,c); ir=r; ic=c;
        end
    end
end;
r, c, mx % Отобразить результаты
```

4.3.2.6.

```
% Поиск максимального элемента в массиве с входным контролем
% Вначале выполним копию массива Y в массив WM
WM=input('Input 2D-array name (example: Y) >>');
if isempty(WM) || (size(WM,2)< 1)
    disp('< array is incorrect > : end')
else
    mx=WM(1,1); ir=1; ic=1;
    for r=1:size(WM,1)
        for c=1:size(WM,2)
            if WM(r,c) > mx
                mx =WM(r,c); ir=r; ic=c;
            end
        end
    end
end;
r, c, mx % Отобразить результаты
end;
```

4.3.3. Цикл WHILE

Выполняет тело цикла пока условное выражение в его заголовке (expression) вычисляет истинное значение

Синтаксис

```
while expression
    statements
end
```


Цикл `while` повторяет инструкции (statements) своего тела неопределенное количество раз. Инструкции выполняются, если действительная часть условного выражения в его заголовке (expression) имеет все ненулевые элементы, то есть, true (истина).

Простые условные выражения в его заголовке (expression) обычно имеют вид:

(условное выражение) `rel_op` (условное выражение)

где:

`<rel_op> ::= = | == | < | > | <= | > = | ~ =`

Сфера действия оператора `while` всегда заканчивается соответствующим `end`.

Expression (условное выражение)

expression – как условное выражение MATLAB, обычно состоящее из переменных или меньших выражений, соединенных реляционными операторами (например, `count < limit`) или логическими функциями (например, `isreal (A)`).

Простые выражения могут быть объединены логическими операторами (`&`, `|`, `~`) в составные выражения. MATLAB оценивает составные выражения слева направо, придерживаясь правил приоритета оператора.

```
(count < limit) & ((height - offset) >= 0)
```

statements

statements - это одно или несколько инструкций MATLAB, которые должны выполняться только в том случае, если условное выражение в его заголовке (expression) истинно или отличное от нуля.

Замечания

Если вычисленное условное выражение в его заголовке (expression) дает не скалярное значение, то каждый элемент этого значения должен быть истинным или ненулевым, чтобы все выражение считалось истинным. Например, оператор, тогда как `(A < B)` является истинным, только если каждый элемент матрицы A меньше его соответствующего элемента в матрице B. См. Пример 2 ниже.

Пример 1 - Простая инструкция

Переменная `eps` - это толерантность, используемая для определения таких вещей, как близость сингулярности и ранга. Его вычисленное значение является `epsilon` машины, или расстоянием от 1.0 до следующего по величине числа с плавающей запятой на вашем компьютере.

```
eps = 1;  
while (1+eps) > 1  
    eps = eps/2;  
end  
eps = eps*2
```

Пример 2 - Нескалярные матрицы A и B

Пусть:

A =	B =
1 0	1 1
2 3	3 4

Тогда условное выражение оценивается как:

Expression	Вычислено	Пояснение
A < B	False	Если A (1,1) больше B (1,1), то далее уже можно не вычислять.
A < (B + 1)	True	Каждый элемент A меньше, чем тот же элемент B с добавленным 1.
A & B	False	A (1,2) & B (1,2) равно false.
B < 5	True	Каждый элемент B меньше 5.

4.3.4. Цикл WHILE. Примеры

4.3.4.1.

```
% Вычисление суммы элементов вектора строки
vs = rand(1,8)
k=1;
y=0;
while k <= length(vs)
    y=y+vs(k);
    k=k+1;
end;
y
```

4.3.4.2.

```
% Вычисление интеграла функции f, заданной с равномерным
% шагом h по оси X методом трапеций
f = [0 1 2 3 4 5]
h = 1
k=1;
y=0;
while k <= length(f)-1
    y=y+(f(k)+f(k+1))/2;
    k=k+1;
end;
y=y*h
```

4.3.4.3.

```
% Вычисление основания натурального логарифма (e)
% как результата функции exp(1) вычисляемой
% рядом Маклорена.
Series_max = 32; % Максимальная длина ряда
x=1; % Аргумент функции exp(x)
y=0; % Текущий результат
delta = 1; % Начальная абсолютная ошибка
k=1; % Счетчик попыток
p=1; % Текущая степень
f=1; % Текущий факториал
```

```
// Вычислим ряд для функции exp(1)
while (k < 32)
    delta = p/f;
    y=y + delta;
    f=f*k;
    p=p*x;
    k=k+1;
    if (abs(delta) < 1e-10)
        break;
    end;
end;
% Вывод результата и значение ошибки
y
abserror=y-exp(1)
```

4.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.

4.4.1. Выполните все примеры из теоретической части лабораторной работы.

4.4.2. Используя технику диалогового ввода исходных данных (см. пример 4.3.2.5.), написать исходные тексты для следующих алгоритмов:

- 4.4.2.1. Сумма элементов табличной функции;
- 4.4.2.2. Среднее значение элементов табличной функции;
- 4.4.2.3. Вычисление интеграла от табличной функции квадратурами прямоугольников, трапеций и квадратурами Симпсона;
- 4.4.2.4. Произведение табличной элементов табличной функции;
- 4.4.2.5. Вычисление целочисленной степени числа;
- 4.4.2.6. Вычисление факториала числа;
- 4.4.2.7. Вычисление степенных полиномов;
- 4.4.2.8. Нахождение действительного корня алгебраического уравнения методом бисекции (дихотомии);
- 4.4.2.9. Вычисление элементарных функций степенными рядами (рядами Тейлора, Маклорена);
- 4.4.2.10. Поиск минимального и максимального значений табличной функции.
- 4.4.2.11. Простейшая (пузырьковая сортировка) табличных функций.

5. СКРИПТЫ И ФУНКЦИИ (ЛАБ. РАБОТА 5)

5.1. Введение

Выполняя предшествующие работы, мы обычно использовали некоторый стандартный редактор текстов в Windows и последовательности инструкций MATLAB переносили в окно Command Window путем копирования и вставки. В то же время MATLAB содержит свой редактор текстов, который можно вызвать через главное меню MATLAB как File/New/M-File. Данный редактор позволяет выполнить как операции редактирования, так и некоторый сервис пошаговой отладки инструкций. Внешний вид такого редактора представлен на рисунке ниже:

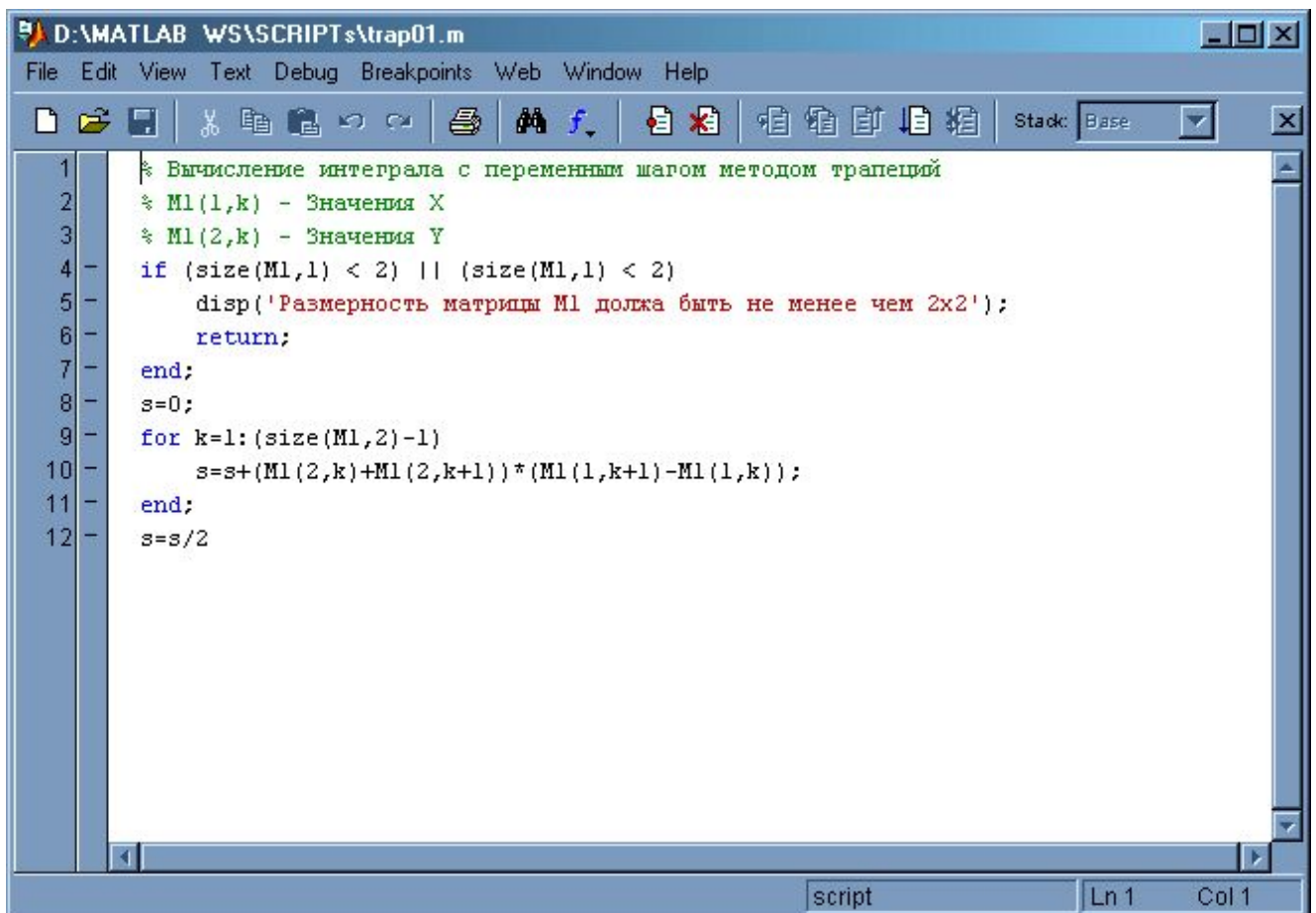


Рис. 5.1. Образ редактора-отладчика в MATLAB версии 6.5.

Однако основная его задача редактора инструкций MATLAB это поддержка скриптов и функций MATLAB.

5.2. Скрипты

Как уже говорилось во введении, для выполнения последовательности инструкций MATLAB мы переносили в окно Command Window путем копирования и вставки. На этом этапе освоения MATLAB, это было удобно в силу наглядности выполнения инструкций. Последовательность выполнения инструкций была связана в Command Window с визуализацией промежуточных результатов и интерпретация останавливалась на любой ошибочной или аварийно завершившейся инструкции. Когда инструкций не так много,

это не вызывает затруднений и для начинающих программистов такая наглядность является ощутимой поддержкой.. Но когда количество инструкций занимает несколько экранов такой подход становится обременительным. Естественным решением этой проблемы является размещение всей цепочки инструкций в отдельном файле и запуск такой цепочки путем ее вызова по имени такого файла. В этом и заключается смысл и техника использования скриптов. Осталось только добавить что файлы скриптов по прежнему остаются текстовыми файлами, однако имеют расширение *.m

Например, файл trap01.m, который представлен на рисунке (Образ редактора-отладчика в MATLAB версии 6.5), в командной строке вызывается следующим образом:

```
>> M1 = [1 2 3 4; 1 4 9 16]
```

Результат:

```
M1 =
     1     2     3     4
     1     4     9    16
```

```
>> trap01
```

Результат:

```
s =
    21.5000
```

Как видно из примера, инструкции скрипта могут использовать и используют переменные из Workspace, а также все константы, которые определены в MATLAB. Кроме того, переменные и константы, определенные внутри скрипта, также создаются и заполняются в Workspace после чего становятся доступными для других скриптов или инструкций, непосредственно вводимых в строку приглашения. Отметим, что такие переменные и константы называются глобальными.

Еще одной важной особенностью скриптов в MATLAB является возможность получение подсказки о скрипте с помощью команды help.

Например:

```
>> help trap01
```

Результат:

```
Вычисление интеграла с переменным шагом методом трапеций
M1(1,k) - Значения X
M1(2,k) - Значения Y
```

Как вы уже знаете, подсказки (помощь) является весьма удобным инструментом. Таким образом, сформулируем правило их оформления:

5.2.1. Правило оформления help для скриптов:

Все строки комментариев MATLAB в скрипте предшествующие первой исполняемой инструкции MATLAB, являются содержимым, которое выводится в Command Window по команде help <имя файла скрипта без расширения>

В файловой системе компьютера имеет место множество файлов. Более того, некоторые файлы в разных директориях имеют одинаковое имя и расширение. Каким образом ограничить MATLAB в поиске необходимого нам скрипта? Для этого в MATLAB предусмотрен навигатор по директориям, который ограничивает поиск файла текущей директорией (папкой). Для вызова навигатора директорий используется главное меню MATLAB (версия 6.5), а в нем выбирается позиция View\Current Directory. На рисунке, приведенном ниже (крайняя справа), показана дополнительная панель Current Directory, в которой отображаются доступные на данный момент m-файлы.

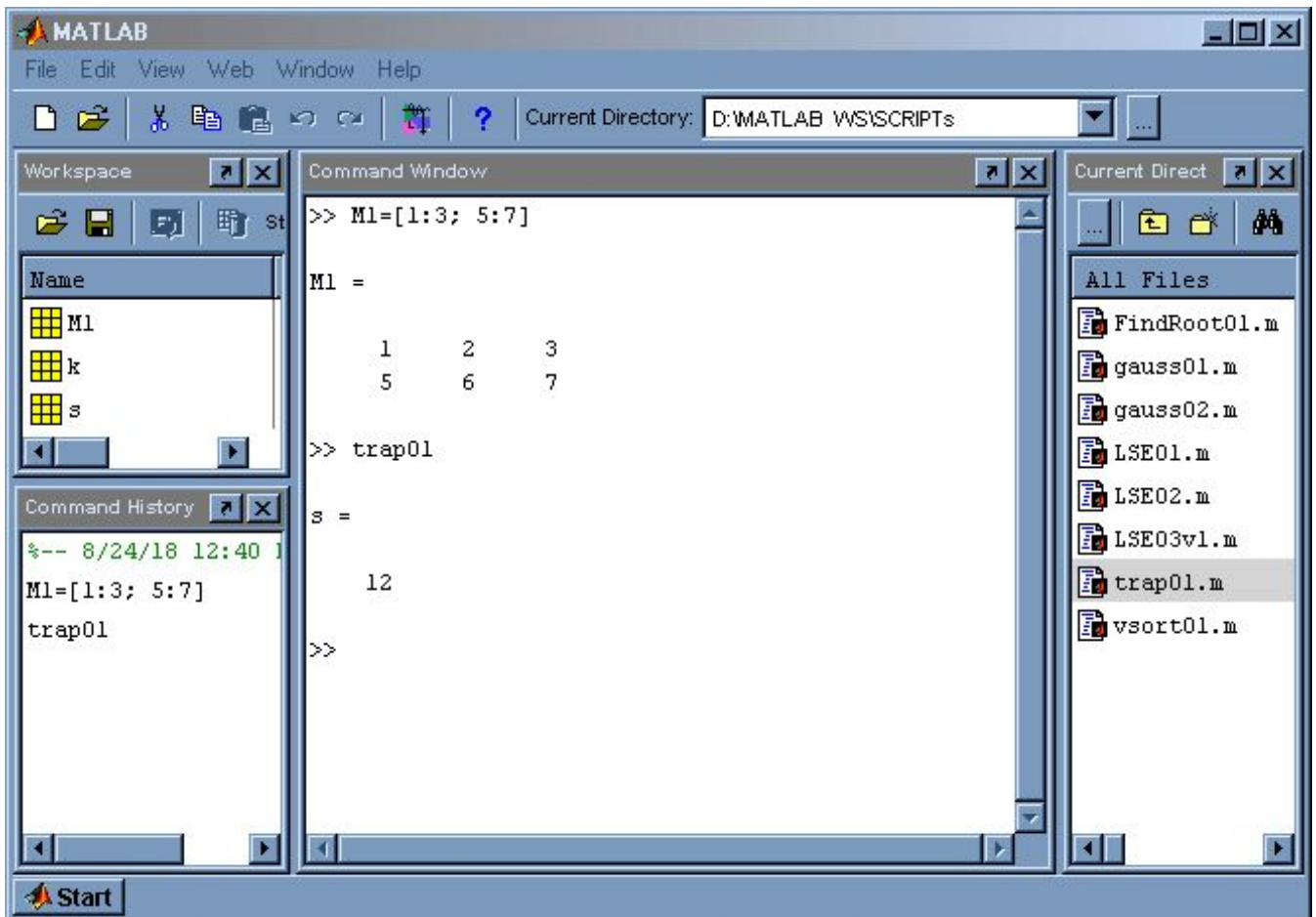


Рис. 5.2. Образ экрана с окном навигации для m-файлов

Двойной клик по любому из этих файлов, позволяет автоматически вызвать и загрузить этим файлом редактор m-файлов MATLAB. Кроме того, панель содержит много полезных инструментов в собственном контекстном меню. Если панель отключена установку текущей директории (Current Directory) можно выполнить с помощью панели быстрых кнопок, которая расположена под строкой главного меню.

5.3. Функции

Основным недостатком скриптов является привязка имен в инструкциях тела скрипта к именам в Workspace.

Например, если скрипт используется в процессе вычислений однократно, то не представляет особого труда подготовить исходные данные с теми именами, которые он использует. Однако, если в процессе вычислений скрипт используется многократно, а

данные, которые являются для него исходными имеют разные имена, то это приводит значительным неудобствам.

Например, в процессе вычислений у нас некоторая матрица A требует вычисления интеграла, а скрипт trap01 воспринимает исходные данные как матрицу с именем M1. В этом случае приходится до вызова скрипта выполнять предварительное копирование матрицы A в матрицу M1:

```
A = [1 2 3 4; 1 4 9 16];
M1=A;
trap01;
```

В принципе, с такими дополнительными затратами, можно смириться (особенно если таких затрат малое количество). Однако ситуация явно выходит из под контроля, если в теле скрипта используется вместо имени M1 имя A. В этом случае выполнение скрипта меняет содержимое матрицы A, которое, как правило, необходимо для дальнейших расчетов. Ситуация значительно усугубляется если таких перекрытий имен несколько, а скрипт достаточно большой.

Вполне понятно, что названная проблема привела к мысли изолировать внутренние переменные и константы скрипта от внешнего мира (Workspace), а для связи с внешним миром предложить некоторый интерфейс, который будет выполнять копирование внешних переменных во внутренние и наоборот. Сформулированную идею можно рассматривать как основную идею для инструкций типа <функция>.

5.3.1. Синтаксис функций

Итак, синтаксис определения функции средствами БНФ можно записать следующим образом:

```
<функция> ::= <заголовок функции> <тело функции>

<заголовок функции> ::= function
    <выходные параметры>
    <имя функции>
    <входные параметры>

<выходные параметры> ::= <пусто>
    | < параметр > =
    | [ <список параметров> ] =

<список параметров> ::= <имя переменной>
    | <список имен>

<список имен> ::= <имя переменной>
    | <список имен> , <имя переменной>

<имя функции> ::= <идентификатор>

<входные параметры> ::= (<пусто>) | (<список параметров>)

<тело функции> ::= <инструкция MATLAB>
    | <тело функции> <инструкция MATLAB>
```

Приведем несколько примеров заголовков функций:

(Вариант 1)

```
function [Var01,Var02]=FunctionName01()
```

Функции с таким заголовком обычно используются для создания и заполнения некоторых переменных в Workspace. В данном примере это переменные (скаляры или матрицы) с именами Var01,Var02

(Вариант 2)

```
function [Var01]=FunctionName02(Var02,Var03,Var04)
```

Функции с таким заголовком используются наиболее часто. Они обращаются к входным параметрам из Workspace (в данном примере это переменные Var02,Var03,Var04) и используя их значения вычисляют выходные параметры

(Вариант 3)

```
function [Var01]=FunctionName01(Var01)
```

Функции с таким заголовком обычно используются для различных преобразований или обработки переменных из Workspace (в данном примере это переменная Var01)

5.3.2. Глобальные и локальные переменные

Глобальные переменные и константы

О глобальных переменных и константах мы уже немного говорили при рассмотрении скриптов. Это переменные и константы, которые представлены в Workspace или MATLAB и доступны инструкциям MATLAB (включая функции) либо в течении от старта до выхода из MATLAB (в течении сеанса MATLAB), либо в течении времени, между их созданием и удалением в процессе сеанса.

Примечание.

Не рекомендуется применять глобальные переменные в теле функции, однако если вы их все же применяете, то их имена не должны совпадать с именами локальных переменных функции. Данная рекомендация не распространяется на константы MATLAB (например, константу Pi)

Локальные переменные функции

Все переменные, которые создаются инструкциями в теле функции, являются локальными. Главная особенность локальных переменных это время их существования. Локальные переменные существуют только пока выполняется тело функции и автоматически удаляются при завершении функции. Другими словами, до начала работы функции они еще не существуют, после завершения работы они уже не существуют. Особо подчеркнем, что при таком режиме существования, локальные переменные должны размещаться и действительно размещаются в отдельном блоке памяти никак не связанном

с Workspace. Такие особенности локальных переменных называют изоляцией или инкапсуляцией переменных внутри функции. Изоляция локальных переменных от Workspace позволяет свободно выбирать имена для локальных переменных без оглядки на их возможное совпадение с именами для глобальных переменных. Другими словами, при работе инструкций в теле функции поиск любой используемой переменной вначале обращается именам в локальной области и только, если имя в локальной области не найдено, то начинают рассматриваться имена в глобальной области. Это важнейшее свойство локальных переменных позволяет рассматривать функции как истинно универсальные инструменты, которые можно создавать в отрыве от конкретного места их применения в дальнейшем.

Например, если в Workspace существует глобальная переменная с именем V01 и некоторая инструкция в собственном теле создает и изменяет (как правило, путем присвоения значения) переменную с таким же именем V01, то такое создание и изменение никоим образом не скажется на содержимом глобальной переменной в Workspace. В то же время, пока работают инструкции тела функции, их обращение к V01 будет обеспечиваться значениями локальной переменной.

Как видно из сказанного, функции разрешают наиболее острую проблему скриптов, обусловленную перекрытием имен. Другая проблема скриптов, связанная с необходимостью настройки скрипта на имена обрабатываемых переменных (в скрипте мы ее решали путем копирования), разрешается в функциях при обработке входных параметров при вызовах функции на исполнение.

5.3.3. Вызов функций

При вызове функции на исполнение, в качестве входных и выходных параметров указываются имена глобальных переменных, для обработки которых и применяется ее вызов.

Например, мы имеем глобальную матрицу A1, которая описывает некоторую таблично заданную функцию. Пусть нам необходимо вычислить интеграл такой таблично заданной функции. Если у нас есть функция, которая предназначена для вычисления определенного интеграла таких таблично заданных функций:

```
% Вычисление интеграла с переменным шагом методом трапеций
% M1(1,k) - Значения X
% M1(2,k) - Значения Y
% s      - Значение интеграла
function s=ftrap01(M1)
s=0;
for k=1:(size(M1,2)-1)
    s=s+(M1(2,k)+M1(2,k+1))*(M1(1,k+1)-M1(1,k));
end;
s=s/2;
```

то для вызова такой функции в Command Window необходимо выполнить вызов:

```
ftrap01(A1)
```

Для начала убедимся в этом, выполнив следующий пример:

```
A1 = [1 2 3 4; 1 4 9 16]
ftrap01(A1)
```

```
Результат:  
A1 =  
     1     2     3     4  
     1     4     9    16  
  
ans = 21.5000
```

Обратим внимание - тело функции написано для переменной с именем M1, но при вызове в позиции этого параметра мы указали глобальную переменную A1. Отсюда следует вывод, что инструкция заголовка функции MATLAB при вызове такой функции на выполнение автоматически реализует копирование внешней переменной A1 во внутреннюю локальную переменную M1, после чего начинают работать инструкции тела функции MATLAB с этой копией. Такой механизм передачи внешних данных в тело функции получил название позиционной передачи параметров. Под позиционностью подразумевается следующее – если входных параметров несколько, то их позиции в заголовке функции определяют позиции, по которым указываются внешние переменные при вызове. И еще один важный момент. Под внешними переменными подразумеваются не только глобальные переменные из Workspace. То есть, если некоторая функция вызывается в теле другой функции, то локальные переменные вызывающей функции еще существуют и могут, как внешние, использоваться вызываемой функцией. А теперь объединим сказанное в несколько итоговых правил:

5.3.4. Основные правила работы с функциями

- Все переменные, которые создаются в теле функции, являются локальными и существуют в течение времени, пока выполняются инструкции функции.
- Поиск имен, которые используются в инструкциях функции, начинается с ее локальных переменных и, в случае отсутствия, расширяется в порядке обратном вложенности вызовов на внешние переменные вплоть до глобальных переменных.
- При вызове функции внешние переменные указываются в том порядке, в котором описаны локальные переменные в списке входных параметров в заголовке функции, после чего значения внешних переменных копируются в локальные переменные, которые и используются в инструкциях в теле функции. В качестве значений параметров можно передавать не только переменные но и явные значения.
- Для сохранения результатов работы функции используется список выходных параметров функции. Переменные во внешней среде, предназначенные для размещения выходных значений вызываемой функцией, создаются функцией, как правило, автоматически, однако могут и переопределяться, если они во внешней среде уже существуют.
- Аналогично скриптам, в функциях можно разместить несколько строк инструкций – комментариев, которые будут отображаться по help команде MATLAB. Такие комментарии рекомендуется размещать в m-файле функции до инструкции заголовка.

5.3.5. Функции. Примеры

Рассмотрим несколько вариантов реализации функций для наиболее популярных алгоритмов обработки данных.

5.3.5.1. Пузырьковая сортировка вектора – строки или вектора – столбца.

```

% ФУНКЦИЯ СОРТИРОВКИ ДЛЯ ВЕКТОРА x
% Синтаксис:
%   x=fvsort(x)
% Например, вариант №1:
%   x=rand(1,8);
%   x=fvsort(x);
% Например, вариант №2:
%   x=rand(8,1);
%   x=fvsort(x);
function x=fvsort(x)
n=length(x);
while (n >= 2)
    for k=1:(n-1)
        if (x(k) > x(k+1))
            buf = x(k+1);
            x(k+1)=x(k);
            x(k)=buf;
        end;
    end;
    n=n-1;
end;

```

Покажем пример работы с этой функцией:

```

B1 = [1 3 6 3 2 8 4 1];
B1=fvsort(B1)

```

Результат:

```

B1 =      1      1      2      3      3      4      6      8

```

5.3.5.2. Вычисление функций рядом Маклорена.**Математическая постановка задачи.**

Ряды Тейлора являются степенными рядами, которые используются для аппроксимации различных функций, что в ряде случаев значительно упрощает анализ и преобразование таких функций.

Традиционно ряд Тейлора определяют следующим образом:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!} (x-a)^k$$

при этом предполагается, что функция $f(x)$ бесконечно дифференцируемая в окрестности точки a .

С математической точки зрения бесконечная длина такого ряда не является препятствием для его рассмотрения и дальнейших аналитических преобразований. С точки зрения реальных вычислений, приходится ограничиваться некоторой конечной длиной ряда. Иными словами, приходится ограничивать верхний индекс ряд Тейлора некоторым конечным значением N , что приводит ряд к следующему виду:

$$f(x) = \sum_{k=0}^N \frac{f^{(k)}(a)}{k!} (x-a)^k + R_N$$

где R_N называют остаточным членом, то есть, некоторым числом, которое отражает ошибку представления функции ограниченным рядом, а его значение оценивают (как правило, в форме Лагранжа) следующим образом:

$$R_N = \frac{f^{(N+1)}(\delta)}{(N+1)!} (x-a)^{N+1}, \quad a < \delta < x$$

При выполнении условия $a = 0$, когда все производные вычисляются в нулевой точке, ряд Тейлора приобретает вид, известный как ряд Маклорена:

$$f(x) = \sum_{k=0}^N \frac{f^{(k)}(0)}{k!} (x)^k + R_N$$

Реализация функции SIN(x) рядом Маклорена (вариант 1).

```
% Вычисление функции SIN(x) рядом Маклорена
% x- аргумент вычисляемой функции
% Например:
%     x=pi/2;
%     y=fsin01(x);
%
function y=fsin01(x)
y=0; % Предусстановка результата
p=1; % x - в очередной степени
f=1; % Очередное значение факториала
% d-derivatives (производные функции вычисленные в нуле)
d=[0 1 0 -1 0 1 0 -1 0 1 0 -1 0 1 0 -1 0 1 0 -1];
n=length(d);
for k=1:n
    y=y+(d(k)/f)*p;
    f=f*k;
    p=p*x;
end;
```

Реализация функции SIN(x) рядом Маклорена (вариант 2).

```
% Вычисление функции SIN(alpha) рядом Маклорена, где alpha задан в
% радианах.
% Доработки функции по отношению к варианту 1.
% 1. Оптимизация алгоритма с ограничением длины ряда по достижению
% ошибки delta
% 2. Перенос угла alpha диапазон -2*pi 2*pi
% 3. Оптимизация алгоритма по значениям производных для ряда
% произвольной длины (переход к патернам)
% x- аргумент вычисляемой функции
% Например:
%     x=pi/2;
%     y=f_sin03(alpha,1e-16);
%
function y=f_sin03(alpha,delta)
% Перенос угла alpha диапазон -2*pi 2*pi
x=alpha-2*pi*fix(alpha/(2*pi));
y=0; % Предусстановка результата
p=1; % x - в очередной степени
f=1; % Очередное значение факториала
% d-derivatives (Повторяющаяся группа производных функции вычисленная в
% нуле)
d=[0 1 0 -1];
```

```

s=realmax;
n=128;
m=1;
for k=1:128
    if d(m)~=0
        y=y+(d(m)/f)*p;
        if abs(s-y)<delta
            disp(strcat('Длина ряда k=', int2str(k), ', макс. длина n=',
int2str(n)));
            return;
        end;
        s=y;
    end;
    m=m+1;
    if m > length(d)
        m=1;
    end;
    f=f*k;
    p=p*x;
end;

```

5.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.

5.4.1. Выполните все примеры из теоретической части лабораторной работы.

5.4.2. Преобразуйте в функции исходные тексты для следующих алгоритмов:

- 5.4.2.1. Сумма элементов табличной функции;
- 5.4.2.2. Среднее значение элементов табличной функции;
- 5.4.2.3. Вычисление интеграла от табличной функции квадратурами прямоугольников, трапеций и квадратурами Симпсона;
- 5.4.2.4. Произведение табличной элементов табличной функции;
- 5.4.2.5. Вычисление целочисленной степени числа;
- 5.4.2.6. Вычисление факториала числа;
- 5.4.2.7. Вычисление степенных полиномов;

5.4.3. Восстановите блок – алгоритмическую схему пузырьковой сортировки и выполните текстовое описание работы алгоритма.

5.4.4. Напишите функцию сортировки для сортировки вектора по убыванию.

5.4.5. Напишите функцию сортировки для сортировки для двухмерной матрицы, в которой первая строка содержит аргумент таблично заданной функции, а вторая значение такой функции для соответствующего аргумента.

5.4.6. Добавьте в список входных параметров функции fsin01 параметр, с помощью которого можно указывать в радианах или градусах задается значение x . Выполните необходимые доработки функции.

5.4.7. Вычислите в нуле некоторое количество производных необходимого порядка для функции $\cos(x)$ и модифицируйте fsin01 для вычисления косинусов.

5.4.8. Рассмотрите пример постановки и частичного решения задачи баллистической доставки грузов на поверхности астероидов с малой гравитацией (Приложение). Завершите этот проект по сформулированному для него заданию.

6. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ. (ЛАБ. РАБОТА 6)

6.1. Решение системы линейных уравнений методом Крамера.

6.1.1. Основная идея метода

Пусть задана исходная система линейных уравнений:

$$\begin{cases} a_{00}X_0 + a_{01}X_1 + a_{02}X_2 + \dots + a_{0n}X_n = b_0 \\ a_{10}X_0 + a_{11}X_1 + a_{12}X_2 + \dots + a_{1n}X_n = b_1 \\ a_{20}X_0 + a_{21}X_1 + a_{22}X_2 + \dots + a_{2n}X_n = b_2 \\ \dots \dots \dots \dots \dots \dots \dots \\ a_{n0}X_0 + a_{n1}X_1 + a_{n2}X_2 + \dots + a_{nn}X_n = b_n \end{cases} \quad (6.1.1.1)$$

Для применения метода Крамера исходную систему линейных уравнений обычно представляют как две матрицы в виде:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0n} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1n} \\ a_{20} & a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n0} & a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (6.1.1.2)$$

$$B = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} \quad (6.1.1.3)$$

Основной идеей метода является вычисление главного и дополнительных определений (детерминантов) матрицы A

Главный определитель D вычисляется с помощью функции **det** в составе функций **MATLAB**:

$$D = \det(A); \quad (6.1.1.4)$$

Вычисление дополнительных определителей D_i вначале требует формирование дополнительных матриц Ab_i , которые получаются путем замены i - столбца матрицы A на матрицу B :

$$\begin{matrix}
 & a_{00} & \dots & b_0 & \dots & a_{0n} \\
 & a_{10} & \dots & b_1 & \dots & a_{1n} \\
 ABi = & a_{20} & \dots & b_2 & \dots & a_{2n} \\
 & \dots & \dots & \dots & \dots & \dots \\
 & a_{n0} & \dots & b_i & \dots & a_{nn}
 \end{matrix}
 \quad (6.1.1.5)$$

Соответственно значения дополнительных определителей можно с помощью **MATLAB** получить следующим образом:

$$Di = \det(ABi); \quad (6.1.1.6)$$

Если главный определитель не равен нулю, то корни системы линейных уравнений вычисляются по формуле:

$$Xi = \frac{Di}{D} \quad (6.1.1.7)$$

В случае равенства нулю главного определителя, решение отсутствует, а соответствующую систему линейных уравнений называют **плохо определенной**.

Система называется **совместной**, если она имеет хотя бы одно решение, и **несовместной**, если у неё нет ни одного решения. Решения считаются различными, если хотя бы одно из значений переменных не совпадает. Совместная система с единственным решением называется **определённой**, при наличии более одного решения — **недоопределённой**.

6.2. Теорема Кронекера-Капелли

Матрица, получающаяся конкатенацией матрицы **A** и столбца правых частей **B** путем конкатенации по столбцам называется расширенной матрицей **AB** системы линейных уравнений.

$$\begin{matrix}
 & a_{00} & a_{01} & a_{02} & \dots & a_{0n} & b_0 \\
 & a_{10} & a_{11} & a_{12} & \dots & a_{1n} & b_1 \\
 AB = & a_{20} & a_{21} & a_{22} & \dots & a_{2n} & b_2 \\
 & \dots & \dots & \dots & \dots & \dots & \dots \\
 & a_{n0} & a_{n1} & a_{n2} & \dots & a_{nn} & b_n
 \end{matrix}
 \quad (6.2.1)$$

Теорема [Кронекер, Капелли]. Система совместна тогда и только тогда, когда ранг матрицы этой системы совпадает с рангом ее расширенной матрицы:

$$rank(A) = rank(AB) \quad (6.2.2)$$

$$X_n = \frac{b_n^*}{a_{nn}^*} \quad (6.3.1.3)$$

После чего, последовательно от $k = n-1$ до $k = 0$ применяется формула, которая использует уже вычисленные значения корней:

$$X_k = \frac{b_k^* - \sum_{i=k+1}^n a_{ki}^* X_i}{a_{kk}^*}, \quad \forall k : 0 \leq k < n \quad (6.3.1.4)$$

Таким образом, основной задачей решения системы уравнений становится задача по последовательному исключению независимых переменных из строк системы уравнений, причем таким образом, чтобы после необходимых исключений новая система (2) оставалась эквивалентной исходной (1). Метод Гаусса предполагает решать названную задачу исключения последовательно, исключая в каждой последующей строке системы уравнений очередную независимую переменную. Такой метод становится возможным, если вспомнить, что любое равенство сохраняется если:

- обе части равенства умножить или поделить на одну и ту же величину;
- к одному равенству добавить или вычесть другое равенство.

Если такие операции применить к некоторой строке системы линейных уравнений и полученным результатом заменить исходную строку, то образованная система уравнений будет эквивалентна исходной, то есть, будет иметь такие же корни, как и исходная. Рассмотрим сформулированную идею метода Гаусса по отдельным шагам.

6.3.2. Метод исключения произвольной переменной из указанного уравнения в системе линейных уравнений.

Для определенности обозначим смысл используемых индексов:

- n – Число строк в системе линейных уравнений;
- m – Индекс независимой переменной X , которая подлежит исключению;
- k – Индекс строки системы уравнений, в которой исключается переменная X_m .

Для исключения переменной X_m из уравнения с k - тым строчным индексом последовательно выполним следующее:

Шаг 1.

Умножим уравнение с m - тым строчным индексом на величину:

$$\frac{a_{km}}{a_{mm}} \quad (6.3.2.1)$$

В результате такого умножения получим:

$$a_{m0} \frac{a_{km}}{a_{mm}} X_0 + a_{m1} \frac{a_{km}}{a_{mm}} X_1 + \dots + a_{mm} \frac{a_{km}}{a_{mm}} X_k + \dots + a_{mn} \frac{a_{km}}{a_{mm}} X_n = b_m \frac{a_{km}}{a_{mm}}$$

или в компактном виде:

$$\sum_{i=0}^n a_{mi} \frac{a_{km}}{a_{mm}} X_i = b_k \frac{a_{km}}{a_{mm}}, \quad \forall : 0 < k \leq n \quad (6.3.2.2)$$

Шаг 2.

Вычтем уравнение, которое мы получили в результате выполнения шага 1 из исходного уравнения с k – тым строчным индексом. В результате вычитания одного равенства из другого, получим следующее:

$$(a_{k0} - a_{m0} \frac{a_{km}}{a_{mm}}) X_0 + \dots + (a_{km} - a_{mm} \frac{a_{km}}{a_{mm}}) X_k + \dots + (a_{kn} - a_{mn} \frac{a_{km}}{a_{mm}}) X_n = b_k - b_m \frac{a_{km}}{a_{mm}}$$

или в компактном виде:

$$\sum_{i=0}^n (a_{ki} - a_{mi} \frac{a_{km}}{a_{mm}}) X_i = b_k - b_m \frac{a_{km}}{a_{mm}} \quad (6.3.2.3)$$

Если, для наглядности, ввести обозначения:

$$a_{ki}^* = a_{ki} - a_{mi} \frac{a_{km}}{a_{mm}} \quad (6.3.2.4)$$

$$b_k^* = b_k - b_m \frac{a_{km}}{a_{mm}} \quad (6.3.2.5)$$

То выражение (6.2.2.3) можно записать в еще более компактной форме:

$$\sum_{i=0}^n a_{ki}^* X_i = b_k^*, \quad \forall : 0 < k \leq n \quad (6.3.2.6)$$

Обратим внимание, что коэффициент при переменной X_m в этом выражении всегда будет равен нулю:

$$(a_{km} - a_{mm} \frac{a_{km}}{a_{mm}}) \equiv 0 \quad (6.3.2.7)$$

Поскольку этот нулевой коэффициент является множителем для переменной X_m , то переменная X_m фактически исключается из выражения (6.3.2.6).

Очевидно, что финишная подсистема разрешима как обычное уравнение, а итоговая система уравнений, включающая финишную подсистему, будет иметь вид:

$$\left\{ \begin{array}{l} a_{00}X_0 + a_{01}X_1 + a_{02}X_2 + \dots + a_{0n}X_n = b_0 \\ \quad a_{11}^*X_1 + a_{12}^*X_2 + \dots + a_{1n}^*X_n = b_1^* \\ \quad \quad a_{22}^*X_2 + \dots + a_{2n}^*X_n = b_2^* \\ \quad \quad \quad \dots \quad \dots \quad \dots \\ \quad \quad \quad \quad a_{nn}^*X_n = b_n^* \end{array} \right. \quad (6.3.3.4)$$

Таким образом, в результате последовательного исключения одной переменной в каждой последующей подсистеме мы получили ответ на задачу и теперь можем вычислить все корни системы линейных уравнений

6.4. Решение систем линейных уравнений. Примеры

6.4.1. Реализация метода Крамера

6.4.1.1.

```
% Пошаговая реализация метода Крамера
% Исходные данные
a=[-1 2 7; 4 3 1; 7 5 -8]
x1=[1; 2; 3]
b=a*x1
% Метод Крамера
% Пошаговая реализация
d=det(a)
% Вычисление x1
aw=a;
aw(:,1)=b;
d1=det(aw);
x1=d1/d
% Вычисление x2
aw=a;
aw(:,2)=b;
d2=det(aw);
x2=d2/d
aw=a;
% Вычисление x3
aw(:,3)=b;
d3=det(aw);
x3=d3/d
```

6.4.1.2.

```
% Универсальная реализация метода Крамера
% Solving Linear Systems of Equations (KRAMER)
% РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ КРАМЕРА
% Синтаксис:
% [x]=fkramer01(a,b)
%
% a - Исходная матрица коэффициентов системы линейных
% уравнений вида:
%
% a11 a12 a13 a14 .... a1N
```

```

% a21 a22 a23 a24 .... a2N
% .....
% aN1 aN2 aN3 aN4 .... aNN
%
% b - Исходная матрица свободных членов системы
% линейных уравнений вида:
%
% b1
% b2
% .....
% bN
%
% x - Вычисляемый вектор корней
%
function [x]=fkramer01(a,b)
if isempty(a) || isempty(b)
    disp('Одна из матриц не задана');
    x=NaN;
    return
end;
if (size(a,1) ~= size(a,2)) || (size(a,1) ~= size(b,1))
    disp('Размеры матриц системы линейных уравнений заданы
неправильно');
    x=NaN;
    return
end;
d=det(a);
if d ~= 0
    for n=1:size(a,2)
        buf=a(:,n);
        a(:,n)=b;
        x(n,1)=det(a)/d;
        a(:,n)=buf;
    end;
else
    x=NaN;
    if (sum(abs(b))==0) && (d == 0)
        x=zeros(size(a,2),1);
        disp('Система имеет тривиальное решение');
    else
        disp('Матрица плохо обусловлена');
    end;
end;
end;

```

6.4.1.2.

```

% Тест точности метода КРАМЕРА
function maxerr=ftest01kramer(n)
if n < 2
    n=2
end;
if n > 500
    n=500
end;
% Создать случайную матрицу и случайные корни
A = rand(n,n);
x1=rand(n,1);
% Вычислить соответствующие свободные члены
b=A*x1;
% ВЫПОЛНИТЬ Тест
x2 = fkramer01(A,b);
err=x2-x1;
maxerr=max(abs(err));
grid on;

```

```
plot(err);
```

6.4.2. Решение систем линейных уравнений средствами MATLAB. Примеры

6.4.2.1. Тестирование ($x=a\backslash b$) метода MatLab

```
% Тест точности метода (x=a\b)
function maxerr=ftest01LSE01(n)
if n < 2
    n=2
end;
if n > 500
    n=500
end;
% Создать случайную матрицу и случайные корни
a = rand(n,n);
x1=rand(n,1);
% Вычислить соответствующие свободные члены
b=a*x1;
% ВЫПОЛНИТЬ Тест
x2 = a\b;
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

6.4.2.2. Тестирование ($x=inv(a)*b$) метода линейной алгебры

```
% Тест точности метода (x=inv(a)*b)
function maxerr=ftest01LSE02(n)
if n < 2
    n=2
end;
if n > 500
    n=500
end;
% Создать случайную матрицу и случайные корни
a = rand(n,n);
x1=rand(n,1);
% Вычислить соответствующие свободные члены
b=a*x1;
% ВЫПОЛНИТЬ Тест
x2 = inv(a)*b;
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

6.4.3. Реализация метода Гаусса. Примеры

6.4.3.1. Реализация метода

```
% Solving Linear Systems of Equations (GAUSS)
% РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ГАУССА
% Синтаксис:
% [x]=fgauss01(ab)
%
```

```

% ab - Исходная матрица системы линейных уравнений вида :
%
%   a11 a12 a13 a14 .... a1N b1
%   a21 a22 a23 a24 .... a2N b2
%   .....
%   aN1 aN2 aN3 aN4 .... aNN bN
%
% x - Вычисляемый вектор корней
%
function [x]=fgauss01(ab)
if isempty(ab)
    disp('Матрица системы линейных уравнений не задана');
    x=NaN;
    return
end;
nb = size(ab,2);
nx = size(ab,1);
if (nb < 2) || (nb ~= (nx+1))
    disp('Размеры матрицы системы линейных уравнений заданы
неправильно');
    x=NaN;
    return
end;
% Построение Гауссовой матрицы
for nM=1:(nx-1)
    % Установка на главную диагональ строки с максимальным элементом
ab(nM,nM)
    [v,n]=max(abs(ab(:,nM)));
    if (n > nM)
        s=ab(n,:);
        ab(n,:) = ab(nM,:);
        ab(nM,:)=s;
    end;
    % Ловушка неразрешимых ситуаций
    if ab(nM,nM) == 0
        disp('Матрица плохо обусловлена');
        x=NaN;
        return;
    end;
    % Построение треугольной Гауссовой матрицы
    for row =(nM + 1):nx
        koef =ab(row,nM)/ab(nM,nM);
        for n=1:nb
            if n == nM
                ab(row,n)=0;
            else
                ab(row,n)=ab(row,n)-ab(nM,n)*koef;;
            end;
        end;
    end;
end;
% Вычисление корней
% Ловушка неразрешимых ситуаций
if ab(nx,nx) == 0
    disp('Матрица плохо обусловлена');
    x=NaN;
    return;
end;
x=zeros(nx,1);
row = nx;
x(nx)=ab(row,nb)/ab(row,nx); % Последний корень
while row > 1
    row=row-1;
    s=0;

```

```

    for n=1:nx
        if ab(row,n) ~= 0
            s=s+ab(row,n)*x(n);
        end;
    end;
    x(row)=(ab(row,nb)-s)/ab(row,row); % Очередной корень
end;

```

6.4.3.2. Тестирование реализации метода ГАУССА

```

% Тест точности метода ГАУССА
function maxerr=fctest01gauss(n)
if n < 2
    n=2
end;
if n > 500
    n=500
end;
% Создать случайную матрицу и случайные корни
a = rand(n,n);
x1=rand(n,1);
% Вычислить соответствующие свободные члены
b=a*x1;
% Создать матрицу линейных уравнений
ab=cat(2,a,b);
% ВЫПОЛНИТЬ Тест
x2 = fgauss01(ab);
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);

```

6.4.4. Сравнительные тесты методов решения SLE

6.4.4.1. Сравнительный тест точности методов SLE

```

% Сравнительный тест точности методов SLE
function [maxerr]=fctest01LSE_All(n)
if n < 2
    n=2
end;
if n > 100
    n=100
end;
% Создать случайную матрицу и случайные корни
a = rand(n,n);
x1=rand(n,1);
b=a*x1;
ab=cat(2,a,b);
% Тест 1
x2 = a\b;
maxerr(1,1)=max(abs(x2-x1));
% Тест 2
x2 = inv(a)*b;
maxerr(2,1)=max(abs(x2-x1));
% Тест 3 (Kramer)
x2 = fkramer01(a,b);
maxerr(3,1)=max(abs(x2-x1));
% Тест 4 (Gauss)
x2 = fgauss01(ab);
maxerr(4,1)=max(abs(x2-x1));
% Отчет

```



```

[v,ind]=max(abs(maxerr));
switch ind
case 1
disp(strcat('Мак. ошибка при x=A\b. Ошибка : ', num2str(v)));
case 2
disp(strcat('Мак. ошибка при x=inv(A)*b. Ошибка : ',
num2str(v)));
case 3
disp(strcat('Мак. ошибка при x=fkramer01(A,b). Ошибка : ',
num2str(v)));
case 4
disp(strcat('Мак. ошибка при x=fgauss01(ab). Ошибка : ',
num2str(v)));
end;

```

6.4.4.2. Сравнительный тест СКОРОСТИ методов GAUSS и KRAMER

```

% Сравнительный тест СКОРОСТИ методов GAUSS и KRAMER
function ftest02_Gauss_Kramer(nm)
if (nm < 100) || (nm > 200)
nm=150;
disp(strcat('Параметр вызова заменен на :', num2str(nm)));
end;
disp('Сравнительный тест СКОРОСТИ методов GAUSS (blue) и KRAMER (red)');
disp(strcat('Сечас будет выполнено вычисление : ', num2str(2*nm), '
систем уравнений'));
disp('ОЖИДАЙТЕ ЗАВЕРШЕНИЯ длительной операции');
pproc=0;
nproc=0;
for n = 1:nm
A = rand(n,n);
b = rand(n,1);
AB = cat(2,A,b);
x(n)=n;
tic
y2 = fgauss01(AB);
t1(n) = toc;
tic
y2 = fkramer01(A,b);
t2(n) = toc;
% Процент выполнения
nproc=100*(n-rem(n,10))/nm;
if nproc > pproc
disp(strcat('Выполнено :', num2str(nproc), '%'));
pproc=nproc;
end;
end;
clf; % Очистка графиков
disp('ГОТОВО! Смотрите графики ...');
hold on;
plot(x,t1); % GAUSS
plot(x,t2,'-r'); % KRAMER

```

6.5. САМОСТОЯТЕЛЬНАЯ РАБОТА.

6.5.1. Выполните все примеры из теоретической части лабораторной работы.

6.5.2. Восстановите блок – алгоритмическую схему решения системы линейных уравнений методом Гаусса.

6.5.3. Напишите функцию MATLAB для простейшей интерполяции некоторой таблично заданной функции степенным рядом.

Примечание:

Идея интерполяции степенным рядом заключается в нахождении коэффициентов некоторого полинома $P(X)$, который на конкретном участке табличной функции гарантированно проходит через заданные ее значения $F(X_i)$. Коэффициенты A_i такого полинома можно определить путем составления системы линейных уравнений. Подчеркнем, что неизвестными в такой системе линейных уравнений будут искомые нами коэффициенты A_i .

Рассмотрим пример составления такой системы линейных уравнений для полинома $P(X)$ второй степени:

$$P(X) = A_0 + A_1 \cdot X + A_2 \cdot X^2$$

Если значения полинома $P(X)$ при X_i будут совпадать со значениями $F(X_i)$ в точках X_i , то это представимо системой линейных уравнений, которая должна иметь вид:

$$\begin{cases} Y_i &= A_0 + A_1 \cdot X_i + A_2 \cdot X_i^2 \\ Y_{i+1} &= A_0 + A_1 \cdot X_{i+1} + A_2 \cdot X_{i+1}^2 \\ Y_{i+2} &= A_0 + A_1 \cdot X_{i+2} + A_2 \cdot X_{i+2}^2 \end{cases}$$

Осталось только разрешить эту систему относительно ее неизвестных A_i . Теперь у нас появилась возможность вычислять $F(X)$ для произвольных значений X внутри области определения ($X_i \dots X_{i+2}$), что и является основной целью интерполяции.

7. АППРОКСИМАЦИЯ ТАБЛИЧНО ЗАДАННОЙ ФУНКЦИИ. (ЛАБ. РАБОТА 7).

7.1. Постановка задачи

Пусть некоторую функцию $F(x)$, представленную табличными значениями, на аргументах $(X_0, \dots, X_m, \dots, X_M)$ в области (X_b, X_e) , необходимо наилучшим образом (то есть наиболее точным и, по возможности, наиболее компактным образом) представить в аналитическом виде.

Такая задача получила наименование – «задача аппроксимации таблично заданной функции».

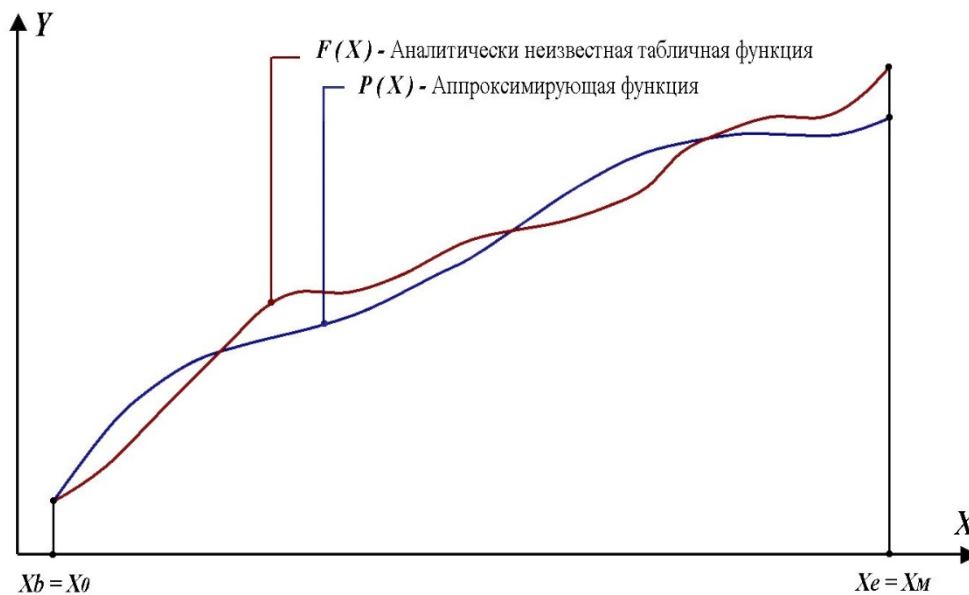


Рис. 7.1.1. Постановка задачи аппроксимации таблично заданной функции.

Эта задача, как правило, решается с помощью *аппроксимирующей* функции $P(x)$ путем определения ее коэффициентов $(a_0, \dots, a_i, \dots, a_n)$ таким образом, чтобы достигнуть минимального различия между исходной таблично заданной функцией $F(x)$ и ее аналитическим эквивалентом, то есть:

$$F(x) \cong P(x) = \sum_{n=0}^N a_n v_n(x)$$

Где:

$v_n(x)$ - является одной из заранее выбранных аналитических функций, которые называют базисными функциями;

a_n - является искомыми коэффициентом при этой функции.

Как вы уже догадались, для того, чтобы найти решение поставленной задачи, как минимум, необходимо математически определить **условия**, при которых достигается равенство:

$$F(x) \cong P(x)$$

Следует также подчеркнуть, что успех решения задачи аппроксимации зависит также от того, будут ли такие условия конструктивны с точки зрения достижения поставленной цели.

7.2. Задача аппроксимации в дискретной форме

Классическим примером условий, которые позволяют решить задачу аппроксимации, являются условия, сформулированные в методе наименьших квадратов. Построение таких условий мы рассмотрим ниже.

7.2.1. Построение условий для решения задачи аппроксимации в дискретной форме.

Для начала, исходя из предположения о существовании аналитического вида функций, графически представим функцию $F(x)$, а также некоторую аппроксимирующую функцию $P(x)$, которая пока еще не вполне удовлетворяет решению нашей задачи.

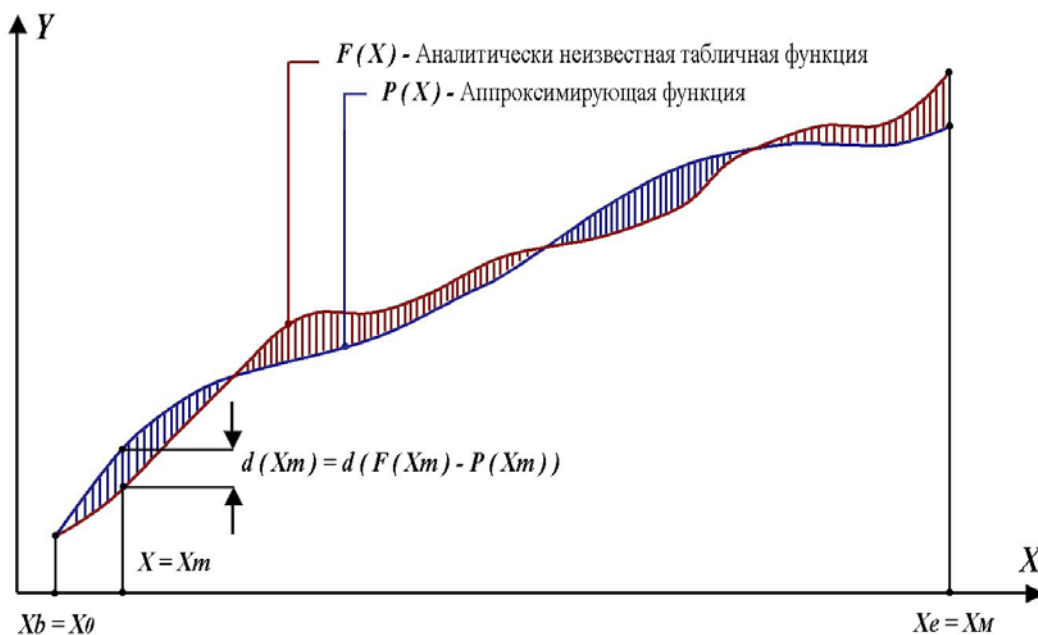


Рис. 7.2.1.1. Отклонения между аппроксимируемой и аппроксимирующей функциями.

Шаг 1. Дискретные отклонения и их сумма

Выводы, которые непосредственно доступны из графического представления функций $F(x)$ и $P(x)$ можно сформулировать следующим образом:

- **Вывод 1.** Если бы нам сразу удалось достигнуть равенства функций, то для каждого значения независимой переменной x , отклонение d между соответствующими значениями этих функций было бы нулевым.

- **Вывод 2.** Сумма всех отклонений будет тем ближе к нулю, чем ближе к нулю будет каждое из составляющих сумму отклонений. Однако, данное утверждение будет справедливым только в том случае, если все отклонения d мы будем рассматривать без учета их знака.
- **Следствие** – Если сумма всех отклонений (без учета их знака) стремится к нулю,

$$\sum_{m=0}^M d(x_m) \rightarrow 0$$

то к нулю будут стремиться все входящие в нее отклонения. Таким образом, если мы построим формальный метод, вычисляющий коэффициенты ($a_0, \dots, a_i, \dots, a_K$), при этом значения этих коэффициентов позволит наилучшим образом приблизить такую сумму к нулевому значению, то это будет равнозначно решению задачи аппроксимации:

$$F(x) \cong P(x)$$

Шаг 2. Отклонения в форме модуля

Для начала, определим отклонение $d(x)$ между функциями $F(x)$ и $P(x)$ как простую разность между этими функциями:

$$d(x) = F(x) - P(x)$$

К сожалению, простые разности, полученные при различных значениях x , могут иметь как положительные, так и отрицательные значения. В итоге, суммируя все положительные и отрицательные отклонения между функциями $F(x)$ и $P(x)$, вполне возможно получить нулевой результат при очевидном неравенстве этих функций. Для этого достаточно, чтобы сумма всех положительных отклонений случайным образом оказалась равна сумме всех отрицательных отклонений.

На первый взгляд, этот недостаток легко устранить - достаточно разность между функциями записать в форме модуля, и проблема будет устранена:

$$d_{\text{mod}}(x_m) = \left| F(x_m) - P(x_m) \right|$$

Однако применение модуля вносит в такое выражение особые точки, то есть точки, в которых производные, взятые слева и справа от точки не равны между собой. Появление таких особых точек (а их количество явно непредсказуемо) сразу вносит существенные сложности в методы поиска минимумов, которые нам далее придется использовать.

Шаг 3. Отклонения в квадратичной форме

Проблема особых точек, обнаруженная нами на шаге 2, ставит перед нами простой вопрос – что для нас более важно, удобная и надежная оценка равенства функций $F(x)$ и $P(x)$ или привязка конструируемой оценки к отклонению в виде *простой разницы* между этими функциями?

Ответ очевиден – важно получить однозначную, всюду положительную, непрерывную функцию оценки, избавленную от особых точек. Самый простой способ получить такую функцию оценки, это представить отклонение как квадрат разности функций $F(x)$ и $P(x)$:

$$d_{sq}(x_m) = (F(x_m) - P(x_m))^2$$

В этом случае, сумма всех отклонений определенная как:

$$\sum_{m=0}^M d_{sq}(x_m) \rightarrow 0$$

будет располагать всем, необходимым для нас, набором свойств:

- Сумма будет гарантировано положительна и ограничена снизу нулевым значением, которое является индикатором равенства функций $F(x)$ и $P(x)$
- Операции, которые использовались при конструировании суммы, не будут вносить особых точек, препятствующих применению методов минимизации.

Шаг 4. Формальное условие для решения задачи аппроксимации

На основании результатов шага 3, запишем окончательное выражение для формальной оценки успешности решения задачи аппроксимации можно записать следующим образом:

$$D = \sum_{m=0}^M ((F(x_m) - P(x_m)))^2 \rightarrow 0$$

Или:

$$D = \sum_{m=0}^M ((F(x_m) - \sum_{n=0}^N a_n v_n(x_m)))^2 \rightarrow 0$$

7.2.2. Решение задачи аппроксимации в дискретной форме

В рамках лабораторной работы мы опустим все шаги, которые последовательно приводят нас к построению следующей системы линейных уравнений относительно неизвестных a_n , решение которой позволяет найти аппроксимирующую функцию $P(x)$:

$$\begin{cases} \sum_{n=0}^N a_n \cdot C_{0,n} = B_0 \\ \dots \\ \sum_{n=0}^N a_n \cdot C_{N,n} = B_N \end{cases}$$

где:

$$B_k = \sum_{m=0}^M F(x_m) \cdot v_k(x_m)$$

$$C_{k,n} = \sum_{m=0}^M v_k(x_m) \cdot v_n(x_m)$$

Легко заметить, что в эти суммы входят функции, которые, либо даны нам как табличные значения для всех своих аргументов x , либо, учитывая известный аналитический вид функций $v(x)$, могут быть вычислены как значения для такого же набора аргументов x . Таким образом, рассматриваемые суммы B_k и $C_{k,n}$, можно рассматривать как константы, значения которых зависят только от индексов k и n .

7.3. РЕАЛИЗАЦИИ МЕТОДОВ НАИМЕНЬШИХ КВАДРАТОВ (Least Squares_Method)

7.3.1. Метод наименьших квадратов для базисных функций x^n

7.3.1.1 Реализация метода

```
% Least Squares_Method
% МЕТОД НАИМЕНЬШИХ КВАДРАТОВ ДЛЯ БАЗИСНЫХ ФУНКЦИЙ x^n
% Синтаксис вызова:
%
%                               [A]=LSM01(F,nV)
%
% F - Исходная матрица табличная функция вида:
%
%   x1 x2 x3 x4 .... xM
%   y1 y2 y3 y4 .... yM
%
% nV - Длина рапроксимирующего степенного ряда
%
% Например (подготовка тестовых данных):
%
%   x=[0:pi/100:2*pi];
%   y=eval('x+sin(x)-log(x+1)');
%
%   F=[x;y];
%
% Вызов метода:
% [A]=LSM01(F,10)
%
% Результат:
% Максимальная абсолютная ошибка : 0.00028535
% A =
% -0.0003
%  1.0068
%  0.4619
% -0.4028
%  0.1001
% -0.0232
%  0.0070
% -0.0013
%  0.0001
% -0.0000
%
```

```

function [A]=LSM01(F,nV)
if isempty(F)
    disp('Матрица [F] не задана');
    A=NaN;
    return
end;
% Минимальный входной контроль
if (size(F,1) ~= 2)
    disp('Размеры матрицы [F] заданы неправильно');
    A=NaN;
    return
end;
if (size(F,2) < nV)
    disp('Недостаточно точек в [F] для определения коэффициентов для
[V]');
    A=NaN;
    return
end;
% Подготовка рабочих матриц
C=zeros(nV,nV);
B=zeros(nV,1);
% Построим систему линейных уравнений для аппроксимации
% Для всех строк матрицы 'C'
for row=1:nV
    % Для всех 'x' матрицы 'F'
    B(row,1)=0;
    for k=1:size(F,2)
        x=F(1,k);
        B(row,1)=B(row,1)+ (F(1,k)^(row-1))*F(2,k);
    end;
    % Для всех колонок матрицы 'C'
    for col=1:nV
        % Для всех 'x' матрицы 'F'
        C(row,col)=0;
        for k=1:size(F,2)
            x=F(1,k);
            C(row,col)=C(row,col)+(F(1,k)^(row-1))*(F(1,k)^(col-1));
        end;
    end;
end;
% Вычислим коэффициенты для аппроксимирующего выражения
ab=cat(2,C,B);
A = fgauss01(ab);
% Ниже представлены варианты, которые иногда выдают предупреждения
% A=inv(C)*B; % A=C\B;

% Вычислим значения функции по аппроксимирующему выражению
% Для всех 'x' матрицы 'F'
for k=1:size(F,2)
    x=F(1,k);
    y(1,k)=0;
    % Для всех строк матрицы 'V'
    for row=1:nV
        y(1,k)=y(1,k)+A(row,1)*(F(1,k)^(row-1));
    end;
end;

% Построим график исходной табличной функции
% plot(F(1,:),F(2,:));
% Построим график по результатам аппроксимации
% plot(F(1,:),y);

% Построим график абсолютной ошибки
delta=F(2,:)-y;

```



```

maxdelta=max(abs(delta));
disp(strcat('Максимальная абсолютная ошибка : ', num2str(maxdelta)));
plot(F(1,:),delta);

```

7.3.1.2 Тест метода наименьших квадратов для базисных функций x^n

```

% SCRIPT Test01_LSM
disp('-----');
disp('ПОДГОТОВКА ТЕСТОВОГО ПОЛИНОМА');
x=[0:0.1:2];
c=[0.5 2 -1 0.3 1.7]
disp('Коэффициенты перечисляются от старших степеней к младшим');
y=polyval(c,x);
plot(x,y);
replay=input('Для продолжения нажмите Enter...');

disp('-----');
disp('АППРОКСИМАЦИЯ СТЕПЕННЫМ РЯДОМ');
F=[x;y];
[A1]=LSM01(F,5);
A1'
disp('Коэффициенты перечисляются от младших степеней к старшим');
replay=input('Для продолжения нажмите Enter...');

disp('-----');
disp('НАЛОЖЕНИЕ ШУМА С МАКСИМАЛЬНОЙ АМПЛИТУДОЙ');
y=polyval(c,x);
ns=rand(1,length(x));
maxns=max(abs(ns))
y=y+ns;
F=[x;y];
disp('АППРОКСИМАЦИЯ СТЕПЕННЫМ РЯДОМ');
[A2]=LSM01(F,5);
A2'
disp('Коэффициенты перечисляются от младших степеней к старшим');

```

7.3.2. Метод наименьших квадратов для произвольных базисных функций

7.3.2.1 Реализация метода

```

% Least Squares Method
% МЕТОД НАИМЕНЬШИХ КВАДРАТОВ ДЛЯ СВОБОДНО ОПРЕДЕЛЯЕМЫХ БАЗИСНЫХ ФУНКЦИЙ
% Синтаксис вызова:
%
%                               [A]=LSM01Free(F,V)
%
% F - Исходная матрица табличная функция вида:
%
%   x1 x2 x3 x4 .... xM
%   y1 y2 y3 y4 .... yM
%
% V - Литеральный вектор столбец описывающий базисные функции :
%
%   v1
%   v2
%   .....
%   vN
%
% Например (подготовка тестовых данных):
%
%   x=[0:pi/100:2*pi];

```

```

%   y=eval('x+sin(x)-log(x+1)');
%
%   F=[x;y];
%   V = ['x^0      '; 'x      '; 'sin(x)  '; 'log(x+1)']
%
% ВНИМАНИЕ! Длина строк всех литералов должна быть строго одинаковой и
% при необходимости расширяться пробелами справа.
%
% Вызов метода:
% [A]=LSM01(F,V)
%
% Результат:
% Максимальная абсолютная ошибка : 3.5527e-014
% A =
%   0.0000
%   1.0000
%   1.0000
%  -1.0000
%
function [A]=LSM01Free(F,V)
if isempty(F) || isempty(V)
    disp('Одна из входных матриц не задана');
    A=NaN;
    return
end;
% Минимальный входной контроль
if (size(F,1) ~= 2)
    disp('Размеры матрицы [F] заданы неправильно');
    A=NaN;
    return
end;
nV=size(V,1);
if (size(F,2) < nV)
    disp('Недостаточно точек в [F] для определения коэффициентов для
[V]');
    A=NaN;
    return
end;
% Подготовка рабочих матриц
C=zeros(nV,nV);
V=zeros(nV,1);
% Построим систему линейных уравнений для аппроксимации
% Для всех строк матрицы 'C'
for row=1:nV
    % Для всех 'x' матрицы 'F'
    V(row,1)=0;
    for k=1:size(F,2)
        x=F(1,k);
        V(row,1)=V(row,1)+eval(V(row,:))*F(2,k);
    end;
    % Для всех колонок матрицы 'C'
    for col=1:nV
        % Для всех 'x' матрицы 'F'
        C(row,col)=0;
        for k=1:size(F,2)
            x=F(1,k);
            C(row,col)=C(row,col)+eval(V(row,:))*eval(V(col,:));
        end;
    end;
end;
% Вычислим коэффициенты для аппроксимирующего выражения
A=C\V;

% Вычислим значения функции по аппроксимирующему выражению

```

```

% Для всех 'x' матрицы 'F'
for k=1:size(F,2)
    x=F(1,k);
    y(1,k)=0;
    % Для всех строк матрицы 'V'
    for row=1:nV
        y(1,k)=y(1,k)+A(row,1)*eval(V(row,:));
    end;
end;

% Построим график исходной табличной функции
% plot(F(1,:),F(2,:));
% Построим график по результатам аппроксимации
% plot(F(1,:),y);

% Построим график абсолютной ошибки
delta=F(2,:)-y;
maxdelta=max(abs(delta));
disp(strcat('Максимальная абсолютная ошибка : ', num2str(maxdelta)));
if max(abs(delta)) > 1e-16
    plot(F(1,:),delta);
end;

```

7.3.2.2 Тест метода наименьших квадратов для произвольных базисных функций

```

% SCRIPT Test02_LSM
hold on;
x=[0:pi/100:2*pi];
y=eval('x+sin(x)-3*log(x+1)');
% plot(x,y);
% ИСХОДНЫЕ ДАННЫЕ
F=[x;y];
% ТЕСТ1
V = ['x^0      '; 'x      '; 'sin(x)  '; 'log(x+1)'];
[A2]=LSM01Free(F,V)
replay=input('Для продолжения нажмите Enter...');
% ТЕСТ2
[A1]=LSM01(F,20)

```

7.4. САМОСТОЯТЕЛЬНАЯ РАБОТА.

7.4.1. Выполните все примеры из теоретической части лабораторной работы.

7.4.2. Восстановите блок – алгоритмическую схему метода наименьших квадратов для базисных функций x^n .

7.4.3. Восстановите блок – алгоритмическую схему метода наименьших для произвольных базисных функций.

СПИСОК ЛИТЕРАТУРЫ

1. Воронов С.И. Методы и алгоритмы для вычисления элементарных функций. – Киев: Национальный авиационный университет, 2015. – 23с., - http://sv52blog.ho.ua/calculus_ua/

2. Воронов С.И. Квадратурные методы для вычисления интегралов табличных функций с постоянным шагом по аргументу (методы и алгоритмы). – Киев: Национальный авиационный университет, 2014. – 20с., - http://sv52blog.ho.ua/calculus_ua/
3. Воронов С.И., Воронова О.С. Аппроксимация таблично заданных функций в инженерных задачах (методы и алгоритмы). – Киев: Национальный авиационный университет, 2015. – 47с., - http://sv52blog.ho.ua/calculus_ua/
4. В.Дьяконов. MATLAB 6: УЧЕБНЫЙ КУРС. - СПб.: Питер, 2001. — 592 с.
5. Половко А. М., Бутусов П. Н. MATLAB для студента. — СПб.: БХВ-Петербург, 2005. —320 с.
6. А. Ф. Дашенко, В. Х. Кириллов, Л. В. Коломиец, В. Ф. Оробей. MATLAB. В инженерных и научных расчетах. - Одесса: «Астропринт», 2003. - 261 с.
7. Кетков Ю. Л., Кетков А. Ю., Шульц М. М. MATLAB 7: программирование, численные методы. — СПб.: БХВ-Петербург, 2005. — 752 с.
8. Gayev Ye.A., Nesterenko B.N. MATLAB for Math and Programming: Textbook, 2nd eddition.– Kyiv: Nat. Aviation Univ, 2015. – 99 p. - <http://sv52blog.ho.ua/matlab-2/>
9. Смоленцев Н. К. Создание Windows приложений с использованием математических процедур MATLAB. – М.: ДМК Пресс, 2008. – 456 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. Задача стыковки челнока и станции

Постановка задачи

На расстоянии s_0 находятся космический челнок и космическая станция. Начальная скорость челнока относительно станции составляет v_0 , а двигатели челнока обеспечивают максимальное ускорение равное a_0 . Необходимо рассчитать минимальное время необходимое для стыковки челнока и станции t_0 .

Кроме того, необходимо получить ответы на вопросы, которые показаны на рисунке, то есть, вычислить t_1, t_2, s_1, s_2, v .

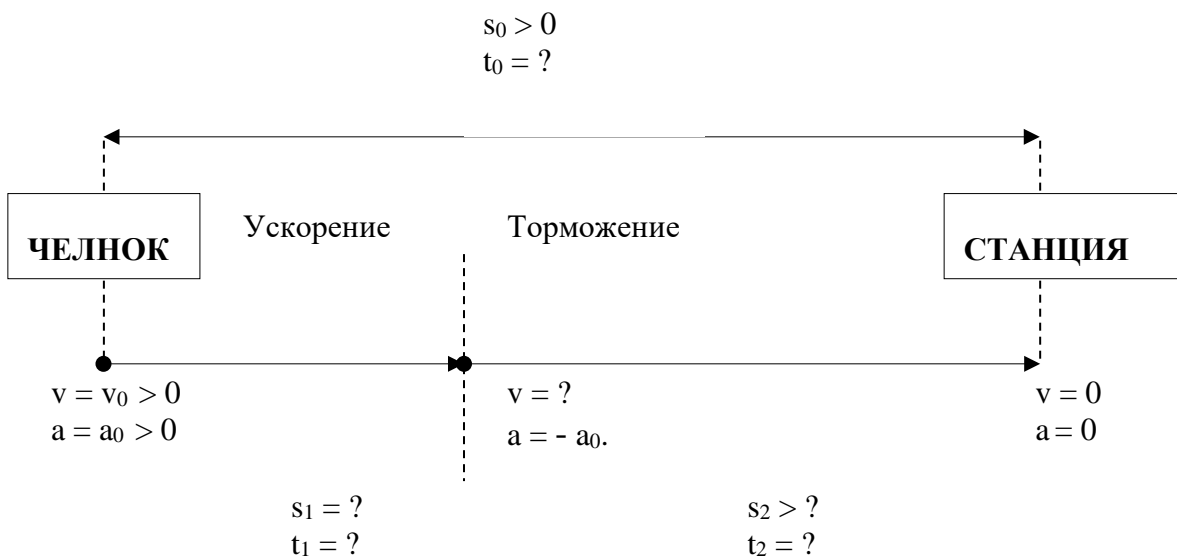


Рис. 1. Параметры сближения челнока и станции

Математический анализ задачи

Поскольку в точке стыковки относительная скорость челнока и станции должна быть равной нулю, то скорость, достигнутую к концу участка ускорения можно представить первым уравнением следующей системы уравнений:

$$\begin{cases} v = v_0 + a_0 \cdot t_1 = a_0 \cdot t_2 \\ s_0 = s_1 + s_2 = v_0 \cdot t_1 + \frac{a_0 \cdot t_1^2}{2} + \frac{a_0 \cdot t_2^2}{2} \end{cases} \quad (1)$$

Второе уравнение отображает составляющие исходного расстояния между челноком и станцией. Для решения системы выразим время торможения из уравнения и подставим это значение в уравнение

$$t_2 = \frac{v_0 + a_0 \cdot t_1}{a_0}$$

В результате подстановки уравнение (1) примет вид, при котором в его состав будет входить только одна неизвестная величина:

$$s_0 = v_0 \cdot t_1 + \frac{a_0 \cdot t_1^2}{2} + \frac{a_0}{2} \cdot \frac{(v_0 + a_0 \cdot t_1)^2}{a_0^2} \quad (2)$$

Упростим выражение (3)

$$s_0 = v_0 \cdot t_1 + \frac{a_0 \cdot t_1^2}{2} + \frac{a_0}{2} \cdot \frac{v_0^2 + 2 \cdot v_0 \cdot a_0 \cdot t_1 + (a_0 \cdot t_1)^2}{a_0^2}$$

После приведения подобных членов мы получим квадратное уравнение:

$$a_0 \cdot t_1^2 + 2 \cdot v_0 \cdot t_1 + \left(\frac{v_0^2}{2 \cdot a_0} - s_0 \right) = 0$$

Для наглядности введем следующие обозначения:

$$a = a_0$$

$$b = 2 \cdot v_0$$

$$c = \frac{v_0^2}{2 \cdot a_0} - s_0$$

Теперь квадратное уравнение принимает классический вид:

$$a \cdot t_1^2 + b \cdot t_1 + c = 0$$

Разрешим это уравнение относительно времени ускорения:

$$t_1 = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Теперь мы имеем возможность вычислить необходимое время торможения:

$$t_2 = \frac{v_0 + a_0 \cdot t_1}{a_0}$$

Полное время необходимое для стыковки:

$$t_0 = t_1 + t_2$$

Максимальную относительную скорость:

$$v = a_0 \cdot t_2$$

Длину участка ускорения:

$$s_1 = v_0 \cdot t_1 + \frac{a_0 \cdot t_1^2}{2}$$

Длину участка торможения:

$$s_2 = \frac{a_0 \cdot t_2^2}{2}$$

А также выполнить контроль вычислений по сумме длин участка ускорения и торможения, которая должна равняться исходному расстоянию между челноком и станцией

$$s_0 = s_1 + s_2$$

Исходные данные и задание

```
N =      ;      % Индивидуальный номер задания
s0=1000*N; % метры
v0=10*N;   % метры деленные на секунду
```

`a0=0.1*N; % метры деленные на квадрате секунду в квадрате`

Содержание работы

- Подготовить блок – алгоритмическую схему решения задачи;
- Подготовить тестовые значения;
- Выполнить численное моделирование задачи с помощью пошагового применения инструкций MATLAB;
- Сформулировать выводы.

ПРИЛОЖЕНИЕ Б. 3D-графики в MATLAB

Постановка задачи

Самостоятельно изучить основные функции 3D-ГРАФИКИ В MATLAB

1. Функция *meshgrid*

Генерирует матрицы X и Y для трехмерных графиков

Syntax

```
[X,Y] = meshgrid(x,y)
[X,Y] = meshgrid(x)
[X,Y,Z] = meshgrid(x,y,z)
```

Описание

$[X, Y] = \text{meshgrid}(x, y)$ преобразует область, указанную векторами x и y , в массивы X и Y , которые могут использоваться для вычисления функций двух переменных, отображаемых в виде трехмерных сетчатых / поверхностных графиков. Строки выходного массива X являются копиями вектора x ; столбцы выходного массива Y являются копиями вектора y .

$[X, Y] = \text{meshgrid}(x)$ совпадает с $[X, Y] = \text{meshgrid}(x, x)$.

$[X, Y, Z] = \text{meshgrid}(x, y, z)$ создает трехмерные массивы, используемые для вычисления функций трех переменных и трехмерных объемных графиков.

Примечания

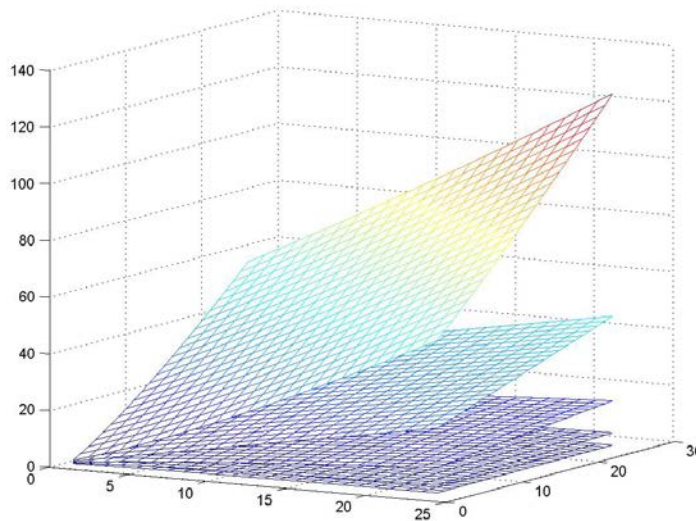
Функция *meshgrid* похожа на *ndgrid*, за исключением того, что порядок первых двух входных и выходных аргументов переключается. То есть утверждение $[X, Y, Z] = \text{meshgrid}(x, y, z)$ дает тот же результат, что и $[Y, X, Z] = \text{ndgrid}(y, x, z)$.

Как следствие, *meshgrid* лучше подходит для задач в двумерном или трехмерном декартовом пространстве, а *ndgrid* лучше подходит для многомерных задач, которые не основаны на пространстве. *meshgrid*, которое ограничено двумерным или трехмерным декартовым пространством.

- $[X, Y] = \text{meshgrid}(x, y)$ преобразует область, указанную векторами x и y ,
- в массивы X и Y , которые могут использоваться для оценки функций двух
- переменных и трехмерных сетчатых / поверхностных графиков. Строки выходного
- массива X являются копиями вектора x ; столбцы выходного массива Y являются
- копиями вектора y .
- $[X, Y] = \text{meshgrid}(x)$ совпадает с $[X, Y] = \text{meshgrid}(x, x)$.
- $[X, Y, Z] = \text{meshgrid}(x, y, z)$ создает трехмерные массивы, используемые для вычисления функций трех переменных и трехмерных объемных графиков. Размеры всех векторов должен быть не менее чем 3. Например:

```
[x,y,z] = meshgrid(1:24,1:24,1:5);
u=(x+y).^ (z/4);
[d1,d2,d3]=size(u);
hold on
for k=1:d3
    mesh(x(:,:,k),y(:,:,k),u(:,:,k));
end;
```

```
grid on;
hold off;
```



После прорисовки, график необходимо развернуть в удобное положение или выполнить поворот программно:

```
az = 30;
el = 30;
view(az, el);
```

2. Функции *mesh*, *meshc*, *meshz*

Прорисовка функций двух переменных X, Y в виде сетки Z

Syntax

```
mesh(X, Y, Z)
mesh(Z)
mesh(..., C)
mesh(..., 'PropertyName', PropertyValue, ...)
meshc(...)
meshz(...)
h = mesh(...)
h = meshc(...)
h = meshz(...)
```

Описание

Функции *mesh*, *meshc* и *meshz* создают параметрические поверхности в виде каркаса (сетки), заданные X , Y и Z , с цветом, заданным C .

mesh(X, Y, Z) рисует каркасную сетку с цветом, который определяется по значениям Z , поэтому цвет пропорционален высоте поверхности,

Если X и Y - векторы, $\text{length}(X) = n$ и $\text{length}(Y) = m$, то $[m, n] = \text{size}(Z)$. В этом случае пересечения линий сетки каркаса; X и Y соответствуют столбцам и строкам Z соответственно.

Если X и Y - матрицы, то пересечения линий сетки каркаса. *mesh(Z)* рисует каркасную сетку, используя $X = 1:n$ и $Y = 1:m$, где $[m, n] = \text{size}(Z)$. Высота, Z , является однозначной функцией, определенной над прямоугольной сеткой. Цвет пропорционален высоте поверхности.

mesh(..., C) рисует каркасную сетку с цветом, определяемым матрицей C . MATLAB выполняет линейное преобразование данных в C для получения цветов из текущей

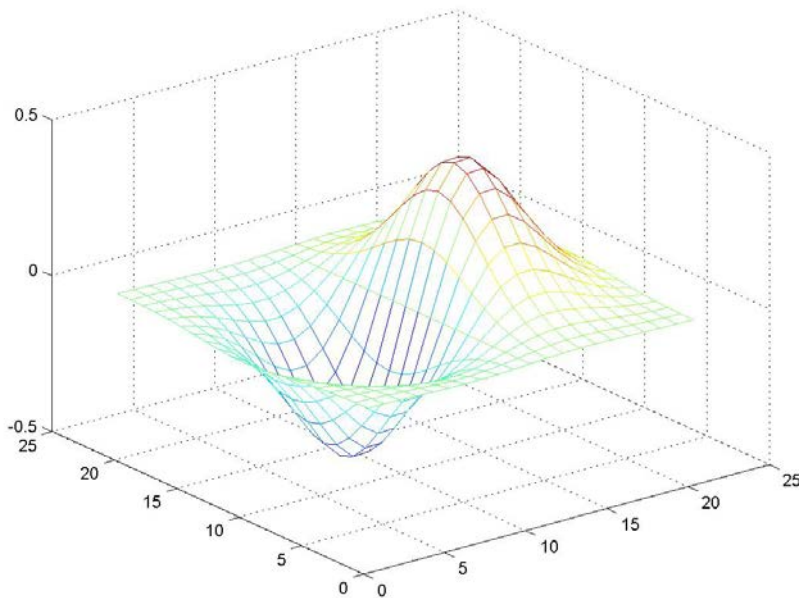
цветовой карты. Если X , Y и Z являются матрицами, они должны быть того же размера, что и C .

- `mesh(...,'PropertyName',PropertyValue,...)` задает значение указанного свойства поверхности. Множественные значения свойств могут быть заданы с помощью одного оператора.
- `meshc(...)` рисует контурный график под сеткой.
- `meshz(...)` рисует график занавеса (то есть опорную плоскость) вокруг сетки.
- `h = mesh(...)`, `h = meshc(...)` и `h = meshz(...)` возвращает дескриптор объекта поверхности.

2.1.

`% Пример mesh`

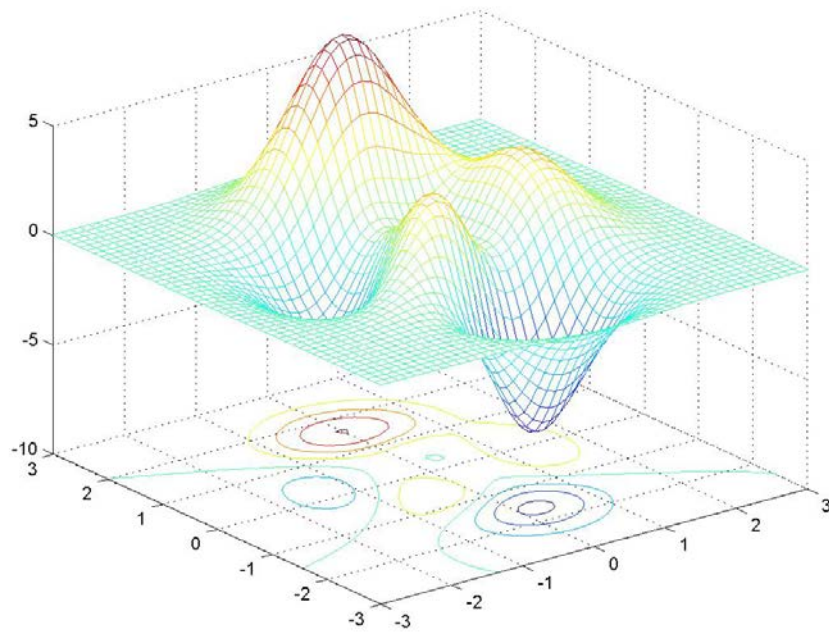
```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
mesh(Z)
```



2.2.

`% Пример meshc`

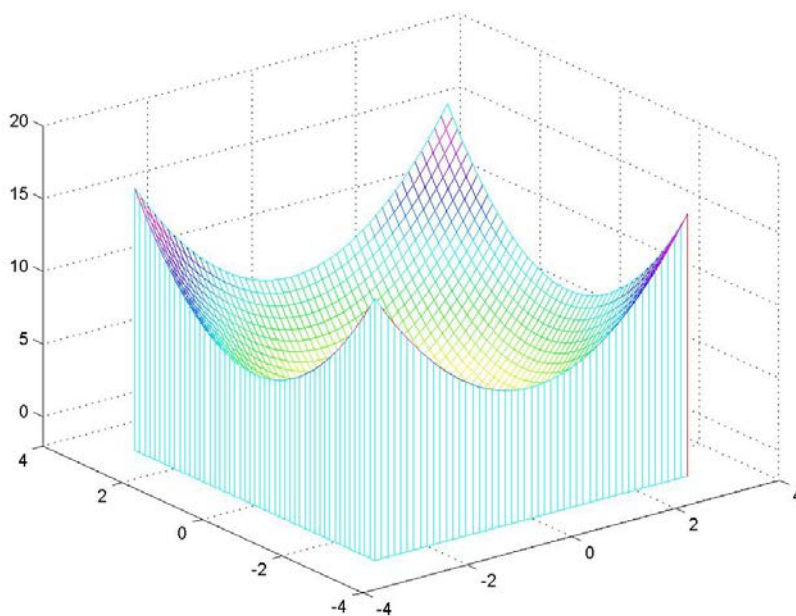
```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```



2.3.

% Пример meshz

```
[X,Y] = meshgrid(-3:.125:3);  
Z = -2+X.^2+Y.^2;  
meshz(X,Y,Z);  
colormap hsv  
axis([-4 4 -4 4 -2 20])
```



3. Функция colormap

MATLAB поддерживает несколько цветовых палитр.

Syntax

```
colormap(map)
colormap('default')
map = colormap
```

Палитры (map) для формата colormap(map):

- **autumn** - меняется плавно от красного, от оранжевого до желтого.
- **bone** - это цветовая палитра в оттенках серого с более высоким значением для синего компонента. Этот набор цветов полезен для добавления «электронного» вида к изображениям в оттенках серого.
- **colorcube** - содержит как можно больше цветных цветов в цветовом пространстве RGB, пытаясь обеспечить больше шагов серого, чистого красного, чистого зеленого и чистого синего.
- **cool** - состоит из цветов, которые являются оттенками голубого и пурпурного. Он изменяется плавно от голубого до пурпурного.
- **copper** – цвет плавно меняется от черной до яркой меди.
- **flag** - состоит из красных, белых, синих и черных цветов. Эта цветовая палитра полностью изменяет цвет с каждым приращением индекса.
- **gray** - возвращает линейную цветовую шкалу серого. горячий плавно меняется от черного, через оттенки красного, оранжевого и желтого до белого.
- **hsv** -изменяет компонент оттенка цветовой модели оттенка насыщенности. Цвета начинаются с красного, проходят через желтый, зеленый, голубой, синий, пурпурный и возвращаются к красному. Цветовая палитра особенно подходит для отображения периодических функций. $hsv(m)$ совпадает с $hsv2rgb([h \text{ единиц}(m, 2)])$, где h - линейная рампа, $h = (0: m-1) / m$.
- **jet** - колеблются от синего до красного и проходят через цвета, голубые, желтые и оранжевые. Это вариация цветовой схемы hsv. Каркас струи связан с аэрофизическим моделированием струй жидкости из Национального центра суперкомпьютерных приложений. См. Раздел «Примеры».
- **lines** - генерируют цветовую палитру цветов, определяемую свойством ColorOrder, а также оттенком серого.
- **pink** - содержит пастельные оттенки розового. Розовая цветовая палитра обеспечивает сепию цветовую окраску оттенков серого.
- **prism** - повторяет шесть цветов: красный, оранжевый, желтый, зеленый, синий и фиолетовый.
- **spring** - состоит из цветов, которые являются оттенками пурпурного и желтого.
- **summer** - состоит из цветов, оттенков зеленого и желтого.
- **white** - это белая монохромная цветовая палитра.
- **winter** - состоит из цветов, которые являются оттенками синего и зеленого.

Описание

Цветовая палитра представляет собой матрицу действительных чисел (m-by-3)- matrix, принимающую значения между 0.0 и 1.0. Каждая строка представляет собой вектор RGB, который определяет один цвет. k-я строка цветовой палитры определяет k-й цвет, где $map(k, :) = [r(k) \ g(k) \ b(k)]$ определяет интенсивность красного, зеленого и синего.

`colormap(map)` устанавливает цветовую карту в матричную карту. Если любые значения на карте находятся за пределами интервала [0 1], MATLAB возвращает ошибку: 'colormap must have values in [0,1]'.

`colormap('default')` устанавливает текущую цветовую палитру в стандартную цветовую палитру.

`map = colormap`; извлекает текущую цветовую палитру в матрицу `map`. Возвращаемые значения находятся в интервале $[0 \ 1]$.

Примеры для формата `colormap(map)`:

```
colormap(jet);  
colormap(gray);
```

4. Функция *shading*

Установка свойства затенения цвета

Syntax

```
shading flat  
shading faceted  
shading interp
```

Описание

Функция затенения управляет цветовой штриховкой поверхностей и патчей графических объектов.

shading flat -затенение каждого сегмента сетки и поверхности имеет постоянный цвет, определяемый значением цвета в конечной точке сегмента или углом поверхности, имеющим наименьший индекс или индексы.

shading faceted -затенение граненое плоское затенение с наложенными черными линиями сетки. Это режим затенения по умолчанию.

shading interp изменяет цвет в каждом сегменте линии и поверхности, путем интерполяции индекса `colormap` или истинного значения цвета по линии или поверхности.

Пример

```
shading flat  
shading faceted  
shading interp
```

5. Функция *view*

Установка точки просмотра

Syntax

```
view(az,el)  
view([az,el])  
view([x,y,z])  
view(2)  
view(3)  
view(T)
```

```
[az,el] = view  
T = view
```

Описание

Позиция зрителя (точка обзора) определяет ориентацию осей. Вы указываете точку зрения в терминах азимута и возвышения, или точкой в трехмерном пространстве.

`view (az, el)` и `view ([az, el])` задают угол обзора для трехмерного графика. Азимут, `az`, представляет собой горизонтальное вращение вокруг оси `z`, измеренное в градусах от отрицательной оси `y`. Положительные значения указывают вращение точки зрения против часовой стрелки. `el` - вертикальное возвышение точки зрения в градусах. Положительные значения высоты соответствуют движению над объектом; отрицательные значения соответствуют перемещению ниже объекта.

- `view ([x, y, z])` задает точку зрения на декартовы координаты `x`, `y` и `z`. Величина (`x`, `y`, `z`) игнорируется.
- `view (2)` устанавливает двумерный вид по умолчанию, `az = 0`, `el = 90`.
- `view (3)` задает трехмерный вид по умолчанию, `az = -37.5`, `el = 30`.
- `view (T)` задает вид согласно матрице преобразования `T`, которая представляет собой матрицу 4 на 4, такую как перспективное преобразование, созданное `viewmtx`. `[az, el] = view` возвращает текущий азимут и высоту.
- `T = view` возвращает текущую матрицу преобразования 4 на 4.

Пример

```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
az = 0;
el = 30;
view(az,el);
mesh(Z);
% Для немедленного завершения цикла нажмите Ctrl+Break
for k=1:8
    pause(1);
    az= az+10;
    view(az,el);
end;
```

6. Функция *surf*

Прорисовка функций двух переменных `X`, `Y` в виде поверхности `Z`.

Размеры `X`, `Y` и `Z` должны быть одинаковыми и не менее чем 3 на 3.

Syntax

```
surf (Z)
surf (X, Y, Z)
surf (... , 'цвет')
surf (... , s)
surf (X, Y, Z, S, k)
h = surf (...)
```

Описание

Функция `surf` отображает заштрихованную поверхность, основанную на комбинации окружающих, диффузных и зеркальных моделей освещения.

`surf (Z)` и `surf (X, Y, Z)` создают трехмерные затененные поверхности, используя направление по умолчанию для источника света и коэффициенты освещения по

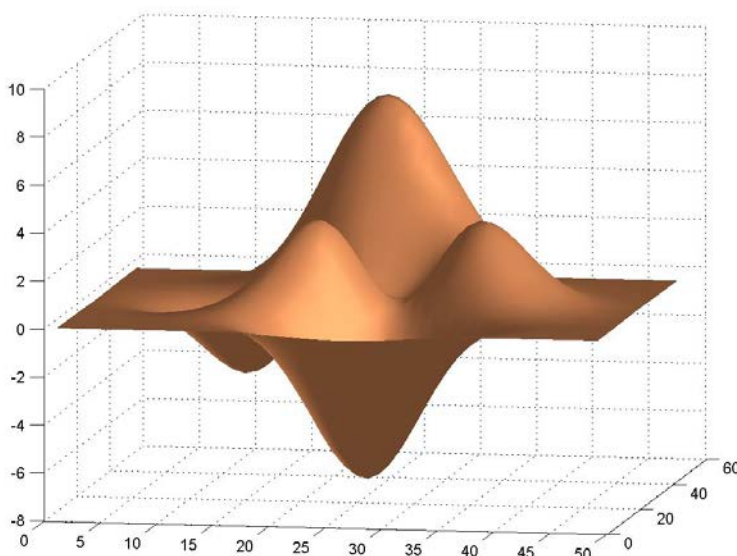
умолчанию для модели затенения. X, Y и Z - векторы или матрицы, которые определяют компоненты x, y и z поверхности.

- `surf` (... , 'light') создает цветную освещенную поверхность с использованием объектива MATLAB. Это дает результаты, отличные от метода освещения по умолчанию, `surf` (... , 'cdata'), который изменяет цветовые данные для поверхности как отражательную способность поверхности.
- `surf` (... , s) определяет направление источника света. s - двух- или трехэлементный вектор, определяющий направление от поверхности к источнику света. $s = [s_x \ s_y \ s_z]$ или $s = [\text{азимутальная высота}]$. Значение по умолчанию s составляет 45° против часовой стрелки от текущего направления просмотра.
- `surf` (X, Y, Z, s, k) определяет константу отражения k. k представляет собой четырехэлементный вектор, определяющий относительные вклады окружающего света, диффузного отражения, зеркального отражения и коэффициента зеркального блеска. $k = [k_a \ k_d \ k_s \ \text{shine}]$ и по умолчанию - [.55, .6, .4,10].
- `h = surf` (...) возвращает дескриптор объекта поверхности.

Примеры:

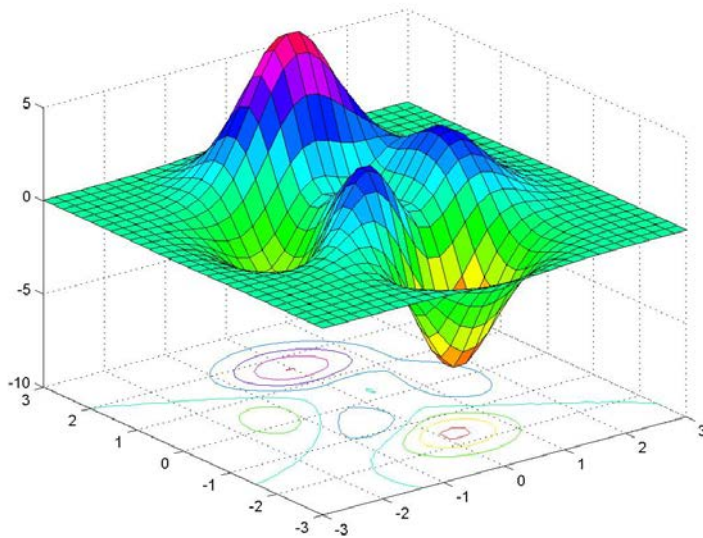
6.1.

```
view([10 10])
grid on
hold on
surf(peaks)
shading interp
colormap copper
hold off
```



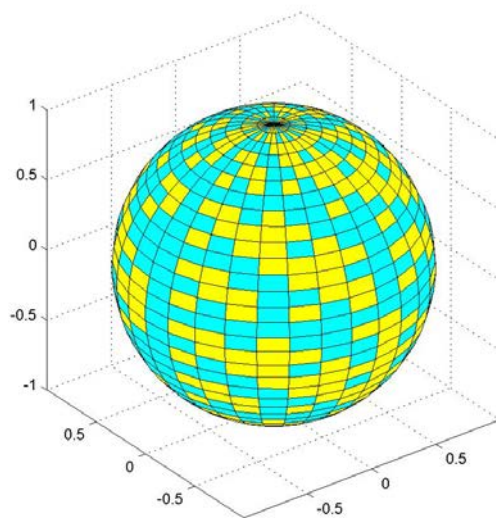
6.2.

```
[X,Y,Z] = peaks(30);
surf(X,Y,Z)
colormap hsv
axis([-3 3 -3 3 -10 5])
```

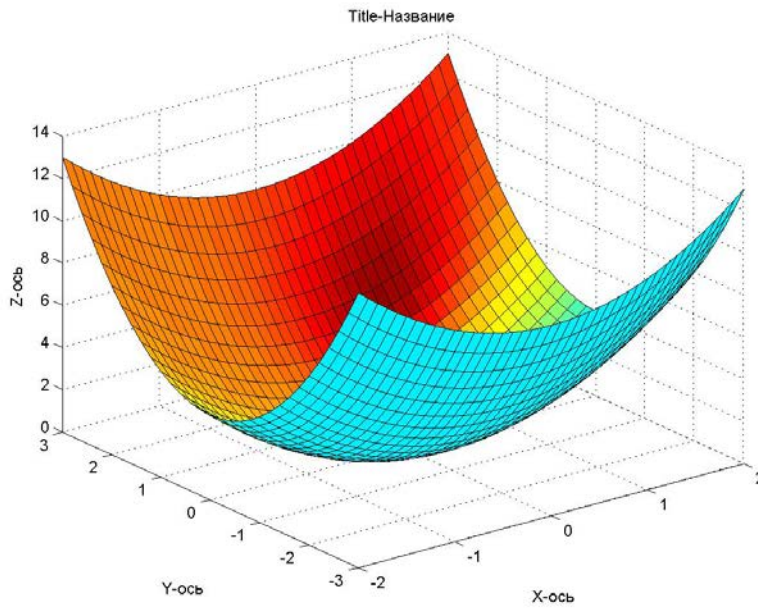
6.3.

```
k = 5;
n = 2^k-1;
[x,y,z] = sphere(n);
c = hadamard(2^k);
surf(x,y,z,c);
colormap([1 1 0; 0 1 1]);
axis equal
```



6.4.

```
[x,y] = meshgrid(-2:0.1:2, -3:0.2:3);
z = x.^2+y.^2;
surf1(x,y,z);
shading faceted
colormap(jet);
xlabel('X-ось');
ylabel('Y-ось');
zlabel('Z-ось');
title('Title-Название');
```



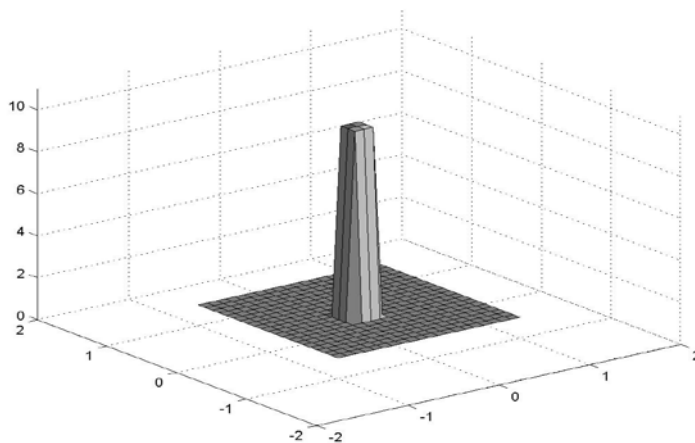
6.5.

```
[X,Y] = meshgrid(-1:0.1:1);
[d1,d2]=size(X);
Z=ones(d1,d2);
a=10;
Z(9,9)=a;
Z(9,10)=a;
Z(9,11)=a;

Z(10,9)=a;
Z(10,10)=a;
Z(10,11)=a;

Z(11,9)=a;
Z(11,10)=a;
Z(11,11)=a;

% mesh(X,Y,Z);
% colormap jet;
surf1(X,Y,Z);
colormap gray;
axis([-2 2 -2 2 0 11]);
```



6.6.

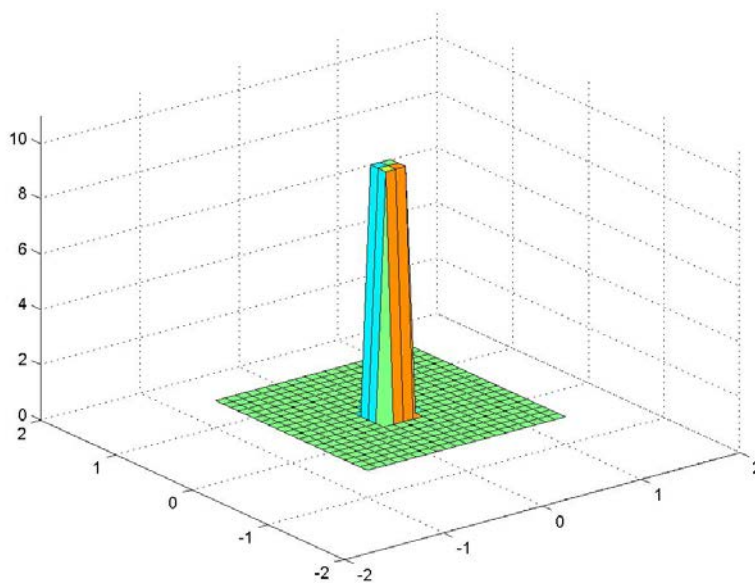
```
[X,Y] = meshgrid(-1:0.1:1);
[d1,d2]=size(X);
Z=ones(d1,d2);
a=10;
c1=fix(d1/2);
c2=fix(d2/2);

Z(c1-1,c2-1)=a;
Z(c1-1,c2)=a;
Z(c1-1,c2+1)=a;

Z(c1,c2-1)=a;
Z(c1,c2)=a;
Z(c1,c2+1)=a;

Z(c1+1,c2-1)=a;
Z(c1+1,c2)=a;
Z(c1+1,c2+1)=a;

% mesh(X,Y,Z);
% colormap jet;
surf1(X,Y,Z);
colormap jet;
axis([-2 2 -2 2 0 11]);
```

***Содержание работы***

Для каждого формата построения 3D – графика (примеры приведенные выше) реализовать собственный пример используя функции, которые вычисляют значения по третьей размерности. Такие функции должны быть отличными от функций, которые использовались в примерах.

ПРИЛОЖЕНИЕ В. Логистическая задача на поверхности астероидов

Введение. Оценка перспективности и актуальности задачи

В последнее время, все чаще рассматриваются перспективы добычи полезных ископаемых на астероидах в солнечной системе. Одним из аспектов проявляемого интереса к такой проблеме, является гипотеза о том, что целый ряд крупных астероидов могут быть обломками планет, следовательно, содержать полезные ресурсы, включая конструкционные, редкие или драгоценные металлы. Освоение такой ресурсной базы открывает возможности поэтапного создания индустрии космических полетов непосредственно в космосе или на малых планетах солнечной системы. Экономическая эффективность подобной индустрии будет определяться исключением дорогостоящей доставки технических грузов с поверхности Земли в космос, использованием обилия солнечной энергии (например, на поверхности Луны или околоземных орбитах), возможностью межпланетных стартов без необходимости преодоления земной гравитации.

Кроме того, до последнего времени такая задача рассматривалась скорее как чисто теоретическая задача с ее ориентацией на практическое воплощение в районе 2050 - 2080 годов. Однако, новая программа НАСА по освоению Луны, а также успех посадки роботов миссии "Хаябуса-2" на астероид Рюгу, позволяют сделать вывод о необходимости форсировать теоретические исследования этой задачи

Из ленты новостей за 24 сентября. 2018 г.

«Роботы MINERVA-III (1a и 1b) успешно сели на поверхность астероида Рюгу. Оба аппарата в отличном состоянии и уже передали фотографии и данные. Мы также подтверждаем, что они двигаются по поверхности», — сообщает официальная страница миссии в Twitter.

Роботы разработаны таким образом, чтобы передвигаться по поверхности астероида с помощью прыжков. В прыжке движение роботов гироскопически стабилизируются специальным вращающимся маховиком. Привычное перемещение роботов по поверхности астероида, например, с помощью колес, невозможно из-за низкой гравитации».

Прыжковый характер перемещения роботов MINERVA-III (1a и 1b), как решение задачи перемещения в условиях низкой гравитации, позволяет отнести задачу «Расчет параметров движения тела, брошенного под углом к горизонту, с учетом изменяющегося с высотой ускорения свободного падения» к задачам актуальным уже сегодня.

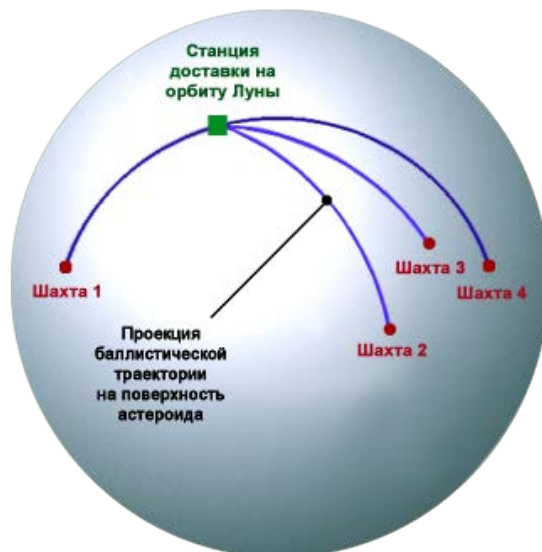
Постановка задачи и содержание работы

- 1) Внимательно проанализировать теоретический материал. На основании изучения этого материала необходимо:
 - Составить полный список допущений и ограничений, использованных при анализе и решении задачи;
 - Составить список самых важных ограничений;
 - Написать скрипт для решения задачи для выбранного вами астероида или астероида, определенного Вам в задании на работу.
 - Выполнить тестовый прогон скрипта для углов запуска (30, 45, 60 градусов) и начальных скоростей (10, 50, 90 м./сек.). Результаты тестового прогона свести в таблицу:

Угол запуска в градусах	Скорость запуска в метрах в секунду	Дальность полета по горизонтали в метрах	Относительная ошибка между энергией пуска и контакта
30	10		
	50		
	90		
45	10		
	50		
	90		
60	10		
	50		
	90		

2) Применительно для конкретного астероида:

- Вычислить угол пуска и начальную скорость контейнера для баллистической доставки груза от условных шахт добычи ресурсов до станции доставки грузов на орбиту Луны;



- Для вычисления начальной скорости при выбранном угле пуска (рекомендуется 45 градусов) использовать ручную реализацию метода численного решения уравнений методом бисекции (дихотомии) до достижения точности доставки в один метр. Результаты расчета отобразить в таблице:

№	Скорость (левая граница)	Скорость (правая граница)	Среднее значение скорости	Дальность полета по средней скорости	Ошибка доставки (м.)

1					
N					<=1

Анализ задачи о движении тела с постоянным ускорением

Эта задача устанавливает зависимость между такими параметрами движения как:

- (t) – время движения
- (a) -ускорение движения
- $(V(t))$ – скорость движения
- $(S(t))$ – путь

При постоянном ускорении (a) задача является классической и ее решение (3) можно связать с решением простейших дифференциальных уравнений (1) и (2)

$$\frac{d}{dt}(V(t)) = a \quad (1)$$

$$d(V(t)) = a dt$$

$$V(t) = \int a dt = at + V_0$$

$$\frac{dS(t)}{dt} = V(t) \quad (2)$$

$$dS(t) = V(t) dt$$

$$S(t) = \int (at + V_0) dt = \frac{at^2}{2} + V_0 t + S_0 \quad (3)$$

В общем случае эту задачу можно записать через простейшее дифференциальное уравнение второго порядка:

$$\frac{d}{dt} \left(\frac{dS(t)}{dt} \right) = a \quad (4)$$

Уравнение (4) достаточно просто разрешимо средствами MATLAB:

```
s=dsolve('D2s=a','s(0)=0')
```

```
% Результат:
```

```
% s = 1/2*a*t^2+C1*t
```

```
% где C1 = V0
```

Анализ задачи о вертикальном движении тела в гравитационном поле

В случае различного статического положения тела относительно сферического источника гравитации, значение ускорения определяется:

- (**G**) - Гравитационной постоянной
- (**M**) - Массой объекта, создающего гравитационное поле
- (**R**) - Радиусом такого объекта
- (**S**) – Радиальной высотой над поверхностью такого объекта.

То есть:

$$a(S) = \frac{G \cdot M}{(R + S)^2} \quad (5)$$

При этом предполагается, что вся масса (**M**) сосредоточена в центральной точке сферы радиуса (**R**).

Такое предположение ограничивает применение зависимости (5) объектами значительной массы и радиуса которые, как правило, достаточно однородны по плотности для фиксированных внутренних радиусов. В случае малых объектов (особенно малых астероидов), их внутренняя структура может быть достаточно неоднородна, а следовательно, зависимость (**a(S)**) должна рассматриваться как векторная сумма ускорений, созданная множеством тяготеющих элементов внутри такого объекта. Очевидно, что такая уточненная зависимость (**a(S)**) существенно усложняет решение задачи. По этой причине, для дальнейшего анализа, остановимся на выборе объектов достаточно больших, чтобы выполнялось исходное предположение. Основным признаком таких объектов является их сферическая форма и масса не менее 1.0e18 (кг.)

Если от статической зависимости ускорения от высоты над поверхностью перейти к движению вертикально вверх или вниз по отношению к поверхности, то ускорение и путь приобретают зависимость от времени:

$$\frac{d}{dt} \left(\frac{dS(t)}{dt} \right) = a(S, t) = \frac{G \cdot M}{(R + S(t))^2} \quad (6)$$

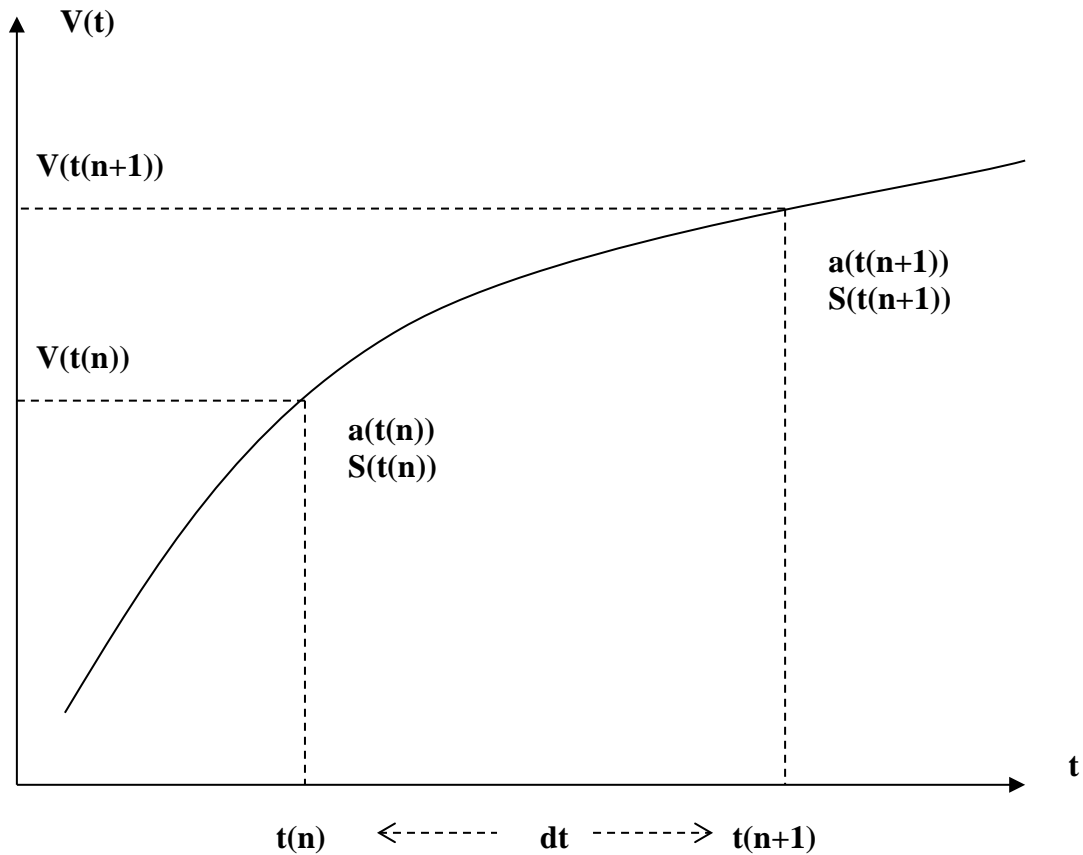
Дифференциальное уравнение (6) уже далеко не является простейшим и даже не является простым, что подтверждается попыткой его решения с помощью MATLAB:

```
s=dsolve('D2s=GM/((R+s)^2)', 's(0)=0')
```

```
% Результат:
% ??? Error using ==> dsolve
% Error, (in solve/rec2) too many levels of recursion
% или «Ошибка (в solve/rec2) слишком много уровней рекурсии»
```

Однако численными методами, путем разбиения всего пути движения на участки, решение все же может быть получено. Кроме того, сообщение MATLAB, дает подсказку по выбору методики решения этой задачи.

Рассмотрим изменение скорости $V(t)$ на некотором участке (n) траектории движения длительностью (dt):



Если устремлять (dt) к нулю, то соответственно приращение ускорения и пути на участке можно сделать сколь угодно малым. Обычно для оценки взаимосвязи приращений используют первую производную. Рассмотрим первую производную ускорения по приращению пути:

```
% Первая производная ускорения (ans) по приращению пути (x)
syms GM R x;
f=GM/(R+x)^2;
diff(f)

% Результат:
% ans = -2*GM/(R+x)^3
```

или

$$\frac{da}{ds} = -\frac{2 \cdot G \cdot M}{(R + S)^3} \quad (7)$$

Для приведения зависимости (7) к удобному для анализа виду, разделим числитель и знаменатель на (R^3) , а также выполним простейшие преобразования:

$$\frac{da}{ds} = -\frac{\frac{2GM}{R^3}}{\left(\frac{R+S}{R}\right)^3} = -\frac{\frac{2GM}{RR^2}}{\left(\frac{R+S}{R}\right)^3} = -\frac{\frac{2g}{R}}{\left(1+\frac{S}{R}\right)^3} = -\frac{2g}{R} \cdot \frac{1}{\left(1+\frac{S}{R}\right)^3}$$

В итоге получим зависимость приращения (**da**) от (**ds**), где (**g**) ускорение свободного падения на уровне поверхности планеты:

$$da = -\frac{2g}{R} \cdot \frac{1}{\left(1+\frac{S}{R}\right)^3} \cdot ds \quad (8)$$

Из приведенной зависимости легко заметить, что при стремлении (**S**) к бесконечности влияние планеты на приращения ускорения полностью исключается. Также очевидно, что такое влияние будет максимальным у самой поверхности, то есть, на первом участке траектории. Это позволяет сформулировать вопрос – можно ли выбрать некоторую длину участка или некоторый постоянный интервал времени для участка, при которых изменением ускорения допустимо пренебречь, а само ускорение на этом участке считать пусть разной, но постоянной величиной? Ответ на этот вопрос может значительно упростить решаемую задачу.

Итак, предположим, что на каждом участке вертикального движения ускорение можно считать пусть разной, но постоянной величиной. Поскольку на первом участке начальная вертикальная скорость максимальна, то длина этого участка будет наибольшей, следовательно, зависимость ускорения от высоты будет проявлять себя максимально. В этом случае, нам необходимо получить оценку возможной относительной ошибки (**gamma**) при допущении о постоянстве ускорения на участке.

Вычислим такую относительную ошибку (**gamma**) в процентах (**(da/g)*100**) при прохождении первого участка вертикальной траектории (**ds**) с некоторой средней скоростью (**V**) за интервал времени (**dt**)

$$\gamma = \frac{da}{g} 100 = -100 \cdot \frac{2}{R} \cdot \frac{1}{\left(1+\frac{0}{R}\right)^3} \cdot ds = -\frac{200}{R} \cdot V dt \quad (9)$$

Поскольку ошибка будет тем больше, чем меньше радиус планеты, рассмотрим динамику ошибки для наименьшего объекта из списка предложенных для исследования, а именно для астероида «Флора». Для астероида «Флора» первая космическая скорость (скорость выхода на круговую орбиту) составляет приблизительно 87 (м./сек). Построим график зависимости относительной ошибки для участков длительностью прохождения от 0,1 до 5 миллисекунд:

```
% Астероид Флора
dt=[0.0001: 0.0001: 0.005]; % Интервал первого участка (сек.)
V=87; % Первая космическая скорость (м./сек)
R=74e3; % Радиус (м.)
gamma=-200*V.*dt/R; % Относительная ошибка
```

```
plot(dt, gamma)
grid on
```

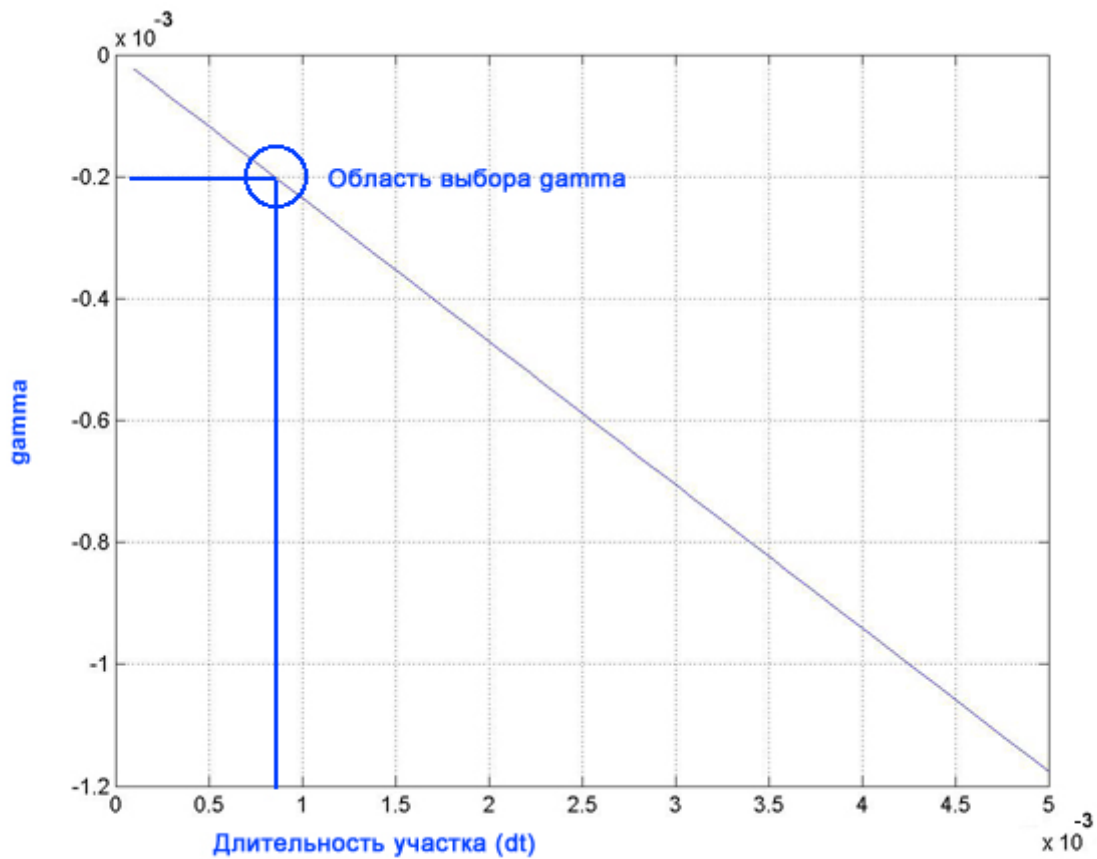


Рис.1. Зависимость временной длительности участка от gamma

Из графика легко заметить, что диапазон относительных ошибок в этом случае находится в границах от 0.1×10^{-3} до 1.2×10^{-3} процента. Учитывая, что большинство технических расчетов допустимо выполнять с относительной ошибкой меньше или равно 1×10^{-2} процента, допущение о постоянстве ускорения свободного падения на участке длительностью одна миллисекунда может быть вполне оправдано.

Кроме того, если задаться некоторой относительной величиной (**gamma**) или допуском на непостоянство ускорения на участке, то с учетом того, что первая космическая скорость это граница невозвращения объекта на поверхность, можно получить оценку времени для каждого участка (**dt**).

$$dt = \text{abs}\left(-\frac{\gamma \cdot R}{200 \cdot V_{S1}}\right) \quad (10)$$

где первая космическая скорость вычисляется по формуле:

$$V_{S1} = \sqrt{\frac{G \cdot M}{R}} \quad (11)$$

Такая оценка позволяет рассматривать постоянные интервалы времени (**dt**), как удобный способ разбиения траектории движения на участки, для которых можно применять

зависимости для движения тела с постоянным ускорением. Однако выбор минимального временного размера участка, полезно ограничить некоторым значением, таким, чтобы количество участков не стремилось к бесконечности. В соответствии с данными графика минимальным значением (**gamma**) целесообразно выбрать значение **0.002** процента.

Вариант решения №1

Таким образом, рассматривая ускорение на каждом участке как постоянную величину, можно записать рекурсивную систему правил для нахождения пути на каждом участке для некоторого объекта, брошенного вертикально вверх со скоростью (**V0**).

(Правило 1) Путь предшествующий первому участку (м.)

$$dt = 0.001$$

(Правило 2) Путь предшествующий первому участку (м.)

$$S_0 = 0$$

(Правило 3) Начальная вертикальная скорость (м./сек.)

$$V_0 = V_0$$

(Правило 4) Ускорение свободного падения на поверхности (м./сек.²)

$$a_0 = -\frac{G \cdot M}{R^2}$$

(Правило 5) Длина очередного участка (м.)

$$S_{n+1} = S_n + V_n \cdot dt + \frac{a_n \cdot dt^2}{2}$$

(Правило 6) Скорость в конце очередного участка (м./сек.)

$$V_{n+1} = V_n + a_n \cdot dt$$

(Правило 7) Ускорение в конце очередного участка (м./сек.²)

$$a_{n+1} = -\frac{G \cdot M}{(R + S_{n+1})^2}$$

(Правило 8) Переход на правило (Правило 5) или при (**Sn+1 <= 0**) завершение рекурсии.

Вариант решения №2

Правило 5 позволяет получить приближенную длину очередного участка. Такая приближенная длина позволяет также уточнить оценку ускорения в конце участка. Таким образом, точность вычисления параметров движения на участке можно улучшить:

(Правило 5-1) Оценка пути в конце текущего участка (м.)

$$Sr_n = S_n + V_n \cdot dt + \frac{a_n \cdot dt^2}{2}$$

(Правило 5-2) Оценка ускорения в конце текущего участка (м./сек.²)

$$ar_n = -\frac{G \cdot M}{(R + Sr_n)^2}$$

(Правило 5-3) Среднее ускорение на текущем участке (м./сек.²)

$$am_n = \frac{ar_n + a_n}{2}$$

(Правило 5-4) Уточненное значение пути в конце текущего участка (м.)

$$S_{n+1} = S_n + V_n \cdot dt + \frac{am_n \cdot dt^2}{2}$$

Вполне вероятно, что при постоянном шаге по времени (**dt**), последний участок пути может частично оказаться как бы ниже поверхности (см. рисунок 2).

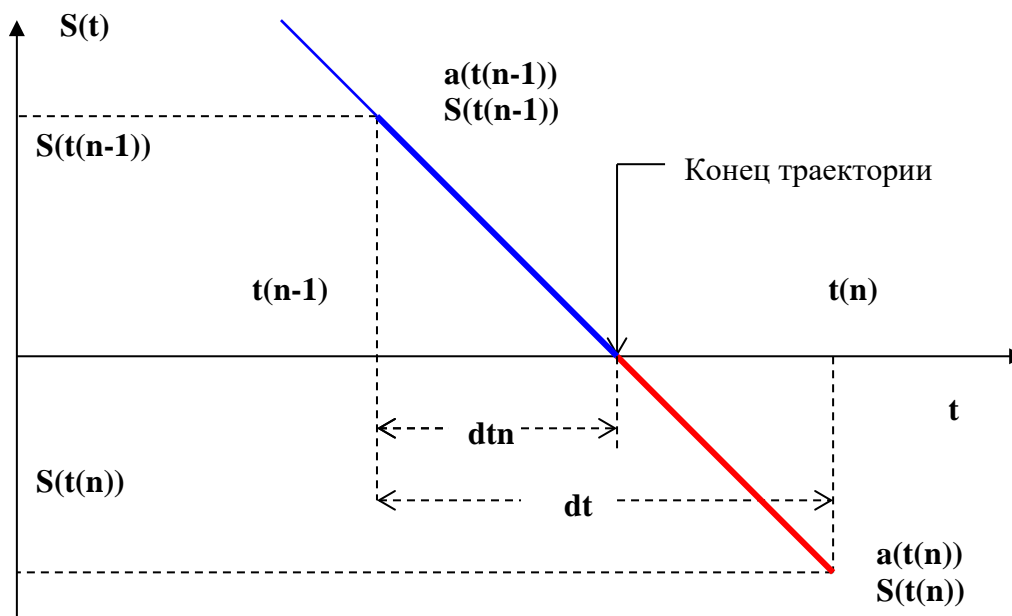


Рис.2 Путь последнего участка, пересекающий уровень поверхности.

Признаком такого события является отрицательное значение пути на очередном участке, что нашло свое отражение (правило 8) как признак конца рекурсии.

В этом случае, для получения правильного результата, нам необходимо найти фактическую длину пути до контакта с поверхностью для такого (последнего) участка, соответствующий интервал времени (**dt_n**), вычислить полный путь и скорректировать скорость контакта с поверхностью.

Итак, пусть правило 5 вычисляет полный путь для очередного участка, например (**S_n**). Если такой путь становится отрицательным, то правило 8 не предусматривает добавление приращение пути на таком участке к полному пути. Однако если для определения полного

пути предварительно длина такого участка (dS_n), то участок пути до поверхности (dpS_n) достаточно просто получить следующим образом:

$$dpS_n = dS_n - S_n \quad (12)$$

Следовательно, полный алгебраический путь (S_e) по всей траектории (или высота объекта над поверхностью) в точке контакта, будет иметь вид:

$$S_e = dS_n + dpS_n \quad (13)$$

Поскольку ускорение на последнем участке мы по прежнему считаем постоянной величиной, то скорость на этом участке будет изменяться по линейному закону, что позволяет получить оценку ее значения (v_e) в точке контакта с поверхностью:

$$V_e = V_{n-1} + (V_n - V_{n-1}) \cdot (dpS / dS) \quad (14)$$

а также вычислить оценку средней скорости:

$$V_m = (V_{n-1} + V_n) / 2$$

и оценку длительности участка во времени (dpt):

$$dpt = abs(dpS / V_m) \quad (15)$$

Для окончательного усреднения ошибок от допущений на последнем участке, вычислим скорость в точке контакта с поверхностью по среднему ускорению:

$$V_e = V_{n-1} + a_m \cdot dpt \quad (16)$$

Таким образом, полное время от старта до контакта с поверхностью может быть вычислено следующим образом:

$$T_n = T_{n-1} + dpt \quad (17)$$

Решение задачи о вертикальном движении тела в гравитационном поле в среде MATLAB

```
% -----
% Вычисление пути тела брошенного вертикально
% в верх по среднему ускорению на участке.
% -----
% Входные параметры:
% M - Масса планеты или астероида (кг.)
% R - Радиус планеты или астероида (м.)
% Vy - Стартовая вертикальная скорость (м./сек.)
% dt - Шаг вычисления по времени (сек.)
```

```

% Выходные параметры:
% T - Вектор отметок текущего времени (сек.)
% Sy - Вектор отметок пути по отметкам времени (м.)
% smax - Максимальная высота на траектории
% tmax - Время максимальной высоты на траектории
% vye - Вертикальная скорость в точке контакта с поверхностью (м./сек.)
%
function [T,Sy,smax,tmax,vye]=f_Vy2Sy(M,R,Vy,dt);
%
% -----
G = 6.6740831e-11; % Гравитационная постоянная
GM=G*M; % Вспомогательная переменная
g= -GM/R^2; % Ускорение свободного падения
% на поверхности

% Начальные параметры цикла
n=1; t=0; se=0; % Начальный индекс; Начальное время;
% Начальный путь
Sy(n)=se; T(n)=t; % Первая точка в векторах результата
n=n+1;
ae=g; % Ускорение до начала первого участка
ve=Vy; % Скорость до начала первого участка
smax=0; % Начальная максимальная высота на траектории
tmax=0; % Начальное время максимальной
% высоты на траектории

% Цикл вычислений
while se >=0
    ab=ae; % Ускорение в начале участка
    vb=ve; % Скорость в начале участка
    sb=se; % Полный путь в начале участка
    % -----
    % Предварительная оценка
    sr=sb+dt*(vb+0.5*ab*dt); % Полный путь в конце участка
    ar=-GM/(R+sr)^2; % Ускорение в конце участка
    a=(ab+ar)/2; % Среднее ускорение на участке
    % -----
    % Окончательная оценка
    dS=dt*(vb+0.5*a*dt); % Расчетная длина участка
    se=sb+dS; % Полный путь в конце участка
    ae=-GM/(R+se)^2; % Ускорение в конце участка
    a=(ab+ae)/2; % Среднее ускорение на участке
    ve=vb+a*dt; % Скорость в конце участка
    if se >=0
        Sy(n)=se;
        t=t+dt; % Полное время в конце участка
        T(n)=t;
        if se > smax
            smax=se; % Текущая максимальная высота
            % на траектории
            tmax=t; % Текущее время максимальной
            % высоты на траектории
        end;
        vye=ve; % Скорость у поверхности
    else
        % Последний участок
        dpS=dS-se; % Уточненная длина последнего участка
        se=dS+dpS; % Полный путь в конце участка
        ve=vb+(ve-vb)*dpS/dS; % Предварительная скорость при контакте
        vm=(ve+vb)/2; % Средняя скорость при контакте
        dpt=abs(dpS/vm); % Оценка времени последнего участка
        vye=vb+a*dpt; % Уточненная скорость в точке контакта
        t=t+dpt; % Полное время от старта до контакта
        Sy(n)=Sy(n-1)+dpS;
        T(n)=t;
        break;
    end
end

```

```

end;
% -----
% for next loop
n=n+1;
end;

```

Задача о движении тела в гравитационном поле, брошенного под углом к горизонту

По аналогии с решением задачи для постоянного ускорения, когда вертикальное движение рассматривается независимо от горизонтального движения, что возможно только при отсутствии дополнительных внешних воздействий (например, отсутствию торможения атмосферой), вертикальное и горизонтальное движение в гравитационном поле можно также считать независимыми движениями.

Для начала рассмотрим классическую задачу движения под углом к горизонту при постоянном ускорении, равным ускорению свободного падения у поверхности.

Движение тела брошенного под углом к горизонту при постоянном ускорении свободного падения

На первой фазе траектории тело, брошенное вертикально вверх со скоростью (V_{yb}), теряет свою скорость под воздействием ускорения свободного падения до нулевого значения.

Для более простой формы записи выражений, будем рассматривать значения скоростей и ускорения по абсолютной величине. Это позволяет записать уравнение, в котором разная направленность векторов скорости и ускорения отображена знаком минус:

$$V_{yb} - g \cdot t_{UP} = 0$$

или вычислить время достижения телом наивысшей точки траектории:

$$t_{UP} = \frac{V_{yb}}{g} \quad (18)$$

На второй фазе вертикального движения тело начиная с нулевой скорости, будет падать и у поверхности достигнет скорости (V_{ye}), то есть:

$$0 - g \cdot t_{DN} = V_{ye}$$

или

$$t_{DN} = \frac{V_{ye}}{g} \quad (19)$$

В силу того, что должен выполняться закон сохранения энергии, то есть, кинетическая энергия тела в начале движения должна быть равной кинетической энергии в конце

движения, то при одинаковой потенциальной энергии старта и финиша справедливо записать:

$$\frac{m \cdot V_{yb}^2}{2} = \frac{m \cdot V_{ye}^2}{2} \quad (20)$$

Следовательно, можно заключить, что также должно выполняться равенство:

$$V_{yb} = V_{ye}$$

или, учитывая выражения (18) и (19)

$$t_{UP} = t_{DN}$$

Таким образом, полное время от старта до финиша будет равно:

$$t = 2 \cdot \frac{V_{yb}}{g} \quad (21)$$

При независимом движении, если начальная скорость горизонтальной составляющей равна (V_{xb}), то полный путь тела по горизонтали или дальность движения по горизонтали составит:

$$L = V_{xb} \cdot t = 2 \cdot \frac{V_{yb} \cdot V_{xb}}{g} \quad (22)$$

Соответственно максимальная высота траектории будет иметь вид:

$$H = \frac{g \cdot t_{UP}^2}{2} = \frac{V_{yb}^2}{2 \cdot g} \quad (23)$$

Подчеркнем, что если скорости и ускорение свободного падения рассматривать без учета знака (то есть, как положительные значения) то от применения функции **abs** можно отказаться.

Если в качестве исходных данных такой задачи рассматривать полный вектор начальной скорости (V_0), а также угол **alpha** этого вектора относительно перпендикуляра к вертикали, то вертикальную и горизонтальную начальные скорости можно записать следующим образом:

$$V_{yb} = V_0 \cdot \sin(\alpha) \quad (24)$$

$$V_{xb} = V_0 \cdot \cos(\alpha) \quad (25)$$

Если записать выражение (17) с учетом (19), также известной из тригонометрии формулы

$$\sin(2 \cdot \alpha) = 2 \cdot \cos(\alpha) \cdot \sin(\alpha),$$

то выражение (17) примет вид:

$$L = \frac{V_0^2 \cdot \sin(2 \cdot \alpha)}{g} \quad (26)$$

Полезно также отметить, что для угла **alpha** равного **45 градусов** синус **двух alpha** или **90 градусов**, принимает свое максимальное значение равное **единице**, то есть:

$$L = \frac{V_0^2}{g} \quad \text{їđè} \quad \alpha = 45^0 \quad (27)$$

Движение тела брошенного под углом к горизонту с ускорением свободного падения, зависящим от высоты над поверхностью

Ранее мы сформулировали систему правил (вариант 1 уточненную вариантом 2 и формулами для последнего участка), систему, которая позволила нам вычислять основные параметры вертикального движения с учетом ускорения свободного падения и его зависимости от высоты над поверхностью:

```
% Вычисление пути тела брошенного вертикально в верх
% по среднему ускорению на участке
% Входные параметры:
% M - Масса планеты или астероида (кг.)
% R - Радиус планеты или астероида (м.)
% Vy - Стартовая вертикальная скорость (м./сек.)
% dt - Шаг вычисления по времени (сек.)
% Выходные параметры:
% T - Вектор отметок текущего времени (сек.)
% Sy - Вектор отметок пути по отметкам времени (м.)
% smax - Максимальная высота на траектории
% tmax - Время максимальной высоты на траектории
% vye - Вертикальная скорость в точке контакта с поверхностью (м./сек.)
%
function [T,Sy,smax,tmax,vye]=f_Vy2Sy (M,R,Vy,dt);
```

С учетом принципа независимости движения, можно вычислить полное время от старта до контакта с поверхностью:

```
n=length(T); % Число точек на оси времени
Tx=T(n); % Время от старта до контакта с поверхностью
```

Для вычисления горизонтальной дальности от точки старта до точки контакта с поверхностью нам необходимо указать угол пуска и выполнить разложение вектора начальной скорости (скорости пуска) на вертикальную и горизонтальную составляющие.

Такую задачу, в среде MATLAB, достаточно просто реализовать следующей вспомогательной диалоговой функцией:

```
% -----
% Диалог по разложению вектора начальной скорости на
% вертикальную и горизонтальную составляющие.
% -----
% Входные параметры:
% VS1 - Первая космическая скорость (м./сек.)
% Выходные параметры:
% V0 - Абсолютное значение скорости пуска (м./сек.)
% Vxb - Абсолютное значение горизонтальной скорости пуска (м./сек.)
% Vyb - Абсолютное значение вертикальной скорости пуска (м./сек.)
% alpha - Угол пуска (радианы)
%
function [V0,Vxb,Vyb,alpha]=f_dialogVyVx(VS1)
%
% -----
PR=50; % Процент допустимой вертикальной скорости от первой космической
clear V0;
V0=input('Введите СКОРОСТЬ ПУСКА (м./сек.), для значения 50 м./сек. просто
Enter >>');
if isempty(V0)
    V0=50;
end;
clear alpha;
alpha=input('Введите УГОЛ ПУСКА (градусы), для значения 45 град. просто Enter
>>');
if isempty(alpha)
    alpha=45;
end;
disp(strcat('Угол пуска (град.) =', num2str(alpha)));
alpha=alpha*pi/180; % Перевод alpha в радианы
Vxb=V0*cos(alpha); % Горизонтальная составляющая скорости
Vyb=V0*sin(alpha); % Вертикальная составляющая скорости
% Ограничение вертикальной скорости по проценту от первой космической
if Vyb > PR*VS1/100
    Vyb = PR*VS1/100;
    V0 = sqrt(Vxb^2+Vyb^2);
    disp('-----');
    disp('                ВНИМАНИЕ!');
    disp('При заданных значениях угла и скорости пуска, вертикальная');
    disp(strcat('составляющая скорости пуска будет > ',num2str(PR),'% первой
космической.'));
    disp(' ДЛЯ ДАЛЬНЕШЕГО РАСЧЕТА СКОРОСТЬ ПУСКА ОТКОРРЕКТИРОВАНА');
    disp(strcat('так, чтобы вертикальная составляющая =',num2str(PR),'% первой
космической.'));
    disp('-----');
end;
disp(strcat('Скорость пуска (м./сек.) =', num2str(V0)));
```

Получив горизонтальную составляющую, вычислим дальность пуска и полную скорость в точке контакта с поверхностью :

```
Lx=Vxb*Tx; % Оценка дальности пуска
Vxy=sqrt(Vxb^2+vye^2); % Полная скорость контакта с поверхностью
```

В завершение решения задачи, вычислим относительную ошибку между полной энергией пуска и полной энергией контакта в процентах, что даст нам оценку точности решения всей задачи.

Итак, в идеальном случае, если потенциальные энергии точек пуска и контакта равны (то есть, радиальные от центра масс высоты для точек пуска и контакта равны), кинетические энергии также должны быть равными или:

$$\frac{m \cdot V_0^2}{2} = \frac{m \cdot V_{xy}^2}{2} \quad (28)$$

или

$$V_0^2 = V_{xy}^2$$

Если они отличаются, то следует вычислить относительную ошибку, которая будет характеризовать это отличие:

$$\gamma = \frac{V_{xy} - V_0}{V_0} \cdot 100 \quad (\%) \quad (29)$$

Окончательное решение задачи в среде MATLAB

Для решения задачи используется функция MATLAB с именем **f_asteroid**, представленная в файловом приложении к теории. Заголовок этой функции имеет вид:

```
% -----
% Вычисление параметров движения тела,
% брошенного под углом к горизонту.
% -----
% Входные параметры:
% M      - Масса планеты или астероида (кг.)
% R      - Радиус планеты или астероида (м.)
% gamma  - Допуск на непостоянство ускорения на участке (%)
% Выходные параметры:
% T      - Вектор отметок текущего времени (сек.)
% Sy     - Вектор отметок пути по отметкам времени (м.)
% tmax   - Время максимальной высоты на траектории
% smax   - Максимальная высота на траектории
% vye    - Вертикальная скорость в точке контакта с поверхностью
%
function [T,Sy,smax,tmax,vye]=f_asteroid(M,R,gamma)
%
% -----
```

Покажем результат работы этой функции на консоли MATLAB при ее запуске с помощью следующего скрипта:

```
close all;
clear all;
clc;
disp('Спутник земли Луна');
M=7.34776e22;           % Масса в (кг.)
R=1738.7e3;            % Средний радиус в (м.)
gamma=0.002;          % Допустимая ошибка изменчивости ускорения на участке в
                      % процентах
```

```
[T,Sy]=f_asteroid(M,R,gamma);
```

Результат:

```
Спутник земли Луна
Средний радиус (км.) =1738.7
Масса (кг.) =7.347760e+022
Ускорение свободного падения на поверхности (м./сек.^2) =1.6222
Первая космическая скорость (м./сек) =1679.4263
Длительность шага моделирования (сек.) =0.010353
Введите СКОРОСТЬ ПУСКА (м./сек.) или для значения 50 м./сек. просто Enter
>>65
Введите УГОЛ ПУСКА (градусы) или для значения 45 град. просто Enter >>48
Угол пуска (град.) =48
Скорость пуска (м./сек.) =65
-----
      ОЖИДАЙТЕ! Выполнение требует некоторого времени ...
      Это время будет тем больше, чем меньше шаг моделирования.
      Для немедленного завершения нажмите Ctrl+Break
-----
Максимальная высота траектории (м.) =719.4921
Время до максимальной высоты траектории (сек.) =29.7958
Время до контакта с поверхностью (сек.) =59.588
Дальность до контакта с поверхностью (м.) =2591.6919
Вертикальная скорость пуска (м./сек.) =48.3044
Вертикальная скорость контакта с поверхностью (м./сек.) =-48.3044
Полная скорость контакта с поверхностью (м./сек.) =65
Относительная ошибка между энергией пуска и контакта (%) =1.3183e-009
-----
      РАСЧЕТ при неизменном ускорении на всех участках = g
      (g=const) Максимальная высота траектории (м.) =719.1945
      (g=const) Время до максимальной высоты траектории (сек.) =29.7776
      (g=const) Дальность до контакта с поверхностью (м.) =2590.2626
      (g=const) Время до контакта с поверхностью (сек.) =59.5552
-----
Время выполнения программы (сек.) =0.922
ВЫПОЛНЕНИЕ ЗАВЕРШЕНО ...
```

