МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. И.А. БУНИНА»

Е.В. Игонина

ПРОГРАММНЫЕ СРЕДСТВА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Учебное пособие

Елец — 2019

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А. Бунина
от ..... г., протокол №

Рецензенты:

М.А. Крутиков, кандидат педагогических наук, доцент кафедры информатики, информационных технологий и защиты информации ЛГПУ им. П.П. Семенова-Тян-Шанского.

А.В. Сидоров, кандидат физико-математических наук, доцент кафедры физики, радиотехники и электроники ЕГУ им. И.А. Бунина.

Е.В.Игонина

Программные средства математического моделирования: учебное пособие. – Елец: ЕГУ им. И.А. Бунина, 2019. – .. с.

Пособие посвящено вопросам применения современного программного средства Scilab для проведения математического моделирования и анализа реальных ситуаций и процессов. В нем даны теоретические основы работы в системе Scilab: главное меню системы, работа с арифметическими выражениями, многочленами, матричным аппаратом и массивами, построение графиков и численное интегрирование, представлены возможности программирования. Предложен комплекс заданий для выполнения лабораторных работ, а также приведены примеры программной реализации математического моделирования некоторых реальных систем.

Предназначено для студентов направления подготовки 01.03.02 Прикладная математика и информатика, также данное пособие будет интересно и полезно студентам IT-направлений: 09.03.01 Информатика и вычислительная техника, 09.03.02 Информационные системы и технологии, а также аспирантам и преподавателям вышеперечисленных направлений подготовки.

Содержание

Введение

Важной особенностью математических моделей, создаваемых в настоящее время, является их комплексность, связанная со сложностью моделируемых объектов и процессов. Это приводит к усложнению модели и необходимости совместного использования нескольких теорий из разных областей знания, применения современных вычислительных методов и вы-числительной техники, прикладных пакетов для получения и анализа результатов моделирования. В случае сложных объектов удовлетворить всем предъявляемым требованиям в одной модели обычно невозможно. Приходится создавать целый спектр моделей одного и того же объекта (в некоторых случаях – иерархическую совокупность «вложенных» одна в другую моделей), каждая из которых наиболее эффективно решает возложенные на нее задачи. Необходимость массового построения моделей требует разработки некоторой совокупности правил и подходов, которые позволили бы снизить затраты на разработку моделей и уменьшить вероятность появления трудно устранимых впоследствии ошибок. Подобную совокупность правил можно было бы назвать технологией создания математических моделей.

Процесс построения любой математической модели можно представить последовательностью выполнения следующих этапов [1]:

1. Обследование объекта моделирования и формулировка технического задания на разработку модели (содержательная постановка задачи);
2. Концептуальная и математическая постановка задачи;
3. Качественный анализ и проверка корректности модели;
4. Выбор и обоснование выбора методов решения задачи;
5. Поиск решения;
6. Разработка алгоритма решения и исследование его свойств, реализация алгоритма в виде программ;
7. Практическое использование построенной модели.

Разработка и использование надежного и эффективного программного обеспечения является также сложным и важным, как и всё вышеперечисленное, этапом создания математической модели. Успешное решение данной задачи возможно лишь при уверенном владении современными алгоритмическими языками и технологиями программирования, знаний возможностей вычислительной техники, имеющегося программного обеспечения, особенностей реализации методов вычислительной математики.

В последнее время для проведения математического моделирования реальных процессов и ситуаций активно используется пакет прикладных математических программ Scilab, который относится к группе “open source” (открытым бесплатным источникам). Scilab имеет внутренний алгоритмический язык, относящимся к языкам программирования высокого уровня, и может быть использован для обработки сигналов, статистического анализа данных, моделирования в гидродинамике, численной оптимизации и моделировании динамических систем, а также для выполнения символических, графических и вычислительных манипуляций [2–4]. Scilab является наиболее полной альтернативой категории очень популярной в мире программы прикладных пакетов (ППП) MATLAB. На основе и с помощью матричных вычислений и автоматического управления памятью многие числовые проблемы могут быть выражены в сокращенном количестве строк программного кода, по сравнению с аналогичными решениями с использованием традиционных языков программирования, таких как Fortran, C или C++. Это дает пользователям возможность быстро построить

модели для ряда математических проблем. Scilab-пакет содержит библиотеки операций высокого уровня, таких как корреляция и методы сложной многомерной арифметики и т.д.

Синтаксис Scilab подобен синтаксису MATLAB [5]: Scilab содержит исходный код конвертора для оказания помощи при преобразовании кода из MATLAB в Scilab и наоборот. Scilab, как это было отмечено выше предоставляется в бесплатной (open source) лицензии. Благодаря открытому исходному коду программного обеспечения, многие профессиональные пользователи имеют возможность для внесения своих разработок в виде ППП для интеграции их в основную программу. Scilab распространяется под GPL – совместимость CeCILL лицензии.

Пособие посвящено вопросам применения современного программного средства Scilab для проведения математического моделирования и анализа реальных ситуаций и процессов. В первой части описываются теоретические основы работы в системе Scilab: главное меню системы, работа с арифметическими выражениями, многочленами, матричным аппаратом и массивами, построение графиков и численное интегрирование, возможности программирования. В пособии приведено множество примеров, облегчающих восприятие теоретического материала и поэтапность проведения математического моделирования реальных ситуаций и процессов. Вторая часть пособия представлена комплексом заданий для выполнения лабораторных работ. Отметим, что номер работы соответствует номеру темы, поэтому перед выполнением заданий для самостоятельного решения следует изучить необходимый теоретический материал. Для более полного и легкого восприятия терминологии данного учебного пособия, рекомендуется предварительное изучение работы [1].

Предназначено для студентов направления подготовки *01.03.02 Прикладная математика и информатика*, также данное пособие будет интересно и полезно студентам IT-направлений: *09.03.01 Информатика и вычислительная техника*, *09.03.02 Информационные системы и технологии.*

Часть 1. Теоретические основы работы в системе Scilab

Тема 1. Знакомство с системой Scilab и ее возможностями

Scilab – мощная интерактивная система автоматизации инженерных, научных и математических расчетов, построенная на расширенном представлении и применении матричных операций.

С 1994 года распространяется в виде исходных кодов через интернет. Сейчас Scilab поддерживается компанией Scilab Consortium, созданной в 2003 году. Scilab использует язык программирования высокого уровня для организации технических вычислений и позволяет работать с элементарными и специальными функциями (Бесселя, Неймана, интегральные функции), имеет мощные средства работы с матрицами, полиномами (в том числе и символьно), производить численные вычисления (например, численное интегрирование) и решение задач линейной алгебры, оптимизации и симуляции, в его состав входят мощные статистические функции, а также средства для построения и работы с графиками.

Перечислим некоторые возможности системы.

- *В области математических вычислений:*
  - матричные, векторные, логические, условные операторы;
  - символьные вычисления;
  - полиномиальные и рациональные функции;
  - элементарные и специальные функции;
  - полиномиальная арифметика.
- В области реализации численных методов:
  - решение дифференциальных уравнений;
  - численное интегрирование;
  - поиск корней нелинейных алгебраических уравнений;
  - оптимизация функций нескольких переменных;
  - одномерная и многомерная интерполяция;
  - решение задач математической статистики.
- В области программирования:
  - свыше 500 встроенных математических функций;
  - интерфейс к Fortran, Tcl/Tk, C, C++, Java, LabView.
- В области визуализации результатов расчетов и графики:
  - возможности создания и редактирования двухмерных и трехмерных графиков;
  - проведение визуального анализа данных.
- Обработка сигналов
- Параллельная работа
- Статистика
- Работа с компьютерной алгеброй

Scilab имеет схожий с MATLAB язык программирования, в составе имеется утилита, позволяющая конвертировать документы Matlab ? Scilab. Программа доступна для различных операционных систем, включая GNU/Linux и Microsoft Windows.

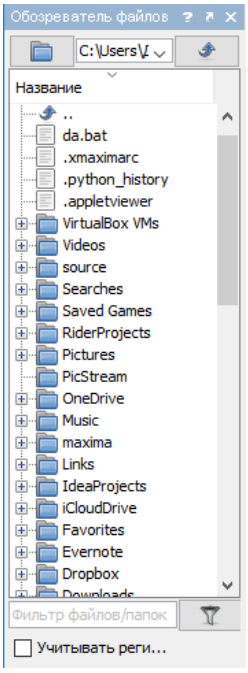
Отличия от некоторых коммерческих программ:

- Бесплатность
- Маленький размер (дистрибутив занимает менее 120Мб против более чем двухгигабайтного пакета MATLAB) Scilab состоит из 3-х частей:
- интерпретатор
- библиотека функций (Scilab-процедуры)
- библиотека Fortran и C процедур

Главное окно программного пакета

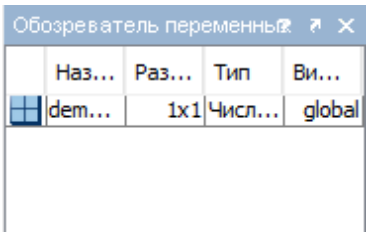
Главное окно программы состоит из:

1. Обозреватель файлов – служащий для отображения и работы с файлами. В случае если мы в программном коде используем для работы файлы, то тут мы можем отслеживать их изменение со временем.

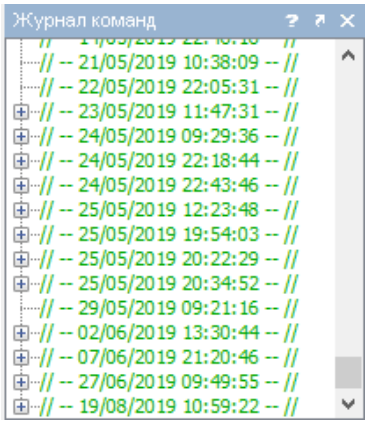


2. Командное окно – служит для отображения процессов работы программного кода, поисков ошибок, подгрузки дополнительных модулей, а также проведения дополнительных вычислений и отображений требуемых нам значений после вычисления.

3. Обозреватель переменных – отображает переменные и информацию о них, которые были введены в программном коде и занесены в память компьютера.



4. Журнал команд – отображает команды которые мы использовали в командном окне в определенный день и время.



5. Панель инструментов – отображает наиболее важные инструменты для работы с программным пакетом

5.1  «SciNotes» – встроенные текстовый редактор в программный пакет, который предназначен для написания программного кода и проведения вычислений.

5.2  «Открыть файл» - открывает необходимый нам для работы файл.

5.3  «Вырезать» - функция является аналогичной стандартной функции Windows «Вырезать». Вырезает часть текста из командного окна.

5.4  «Копировать» - функция копирует из команды необходимый нам текст.

5.5  «Вставить» - функция вставляет из буфера обмена текст который мы ранее копировали.

5.6  «Очистить командное окно» - очищает командное окно, но сохраняет все полученные результаты в памяти.

5.7  «Печать» - отправляет принтеру данные из командного окна, которые будут напечатаны.

5.8  «Управление модулями» - модули «Atoms» - это дополнительные возможности программного пакета, написанные сторонними разработчиками.

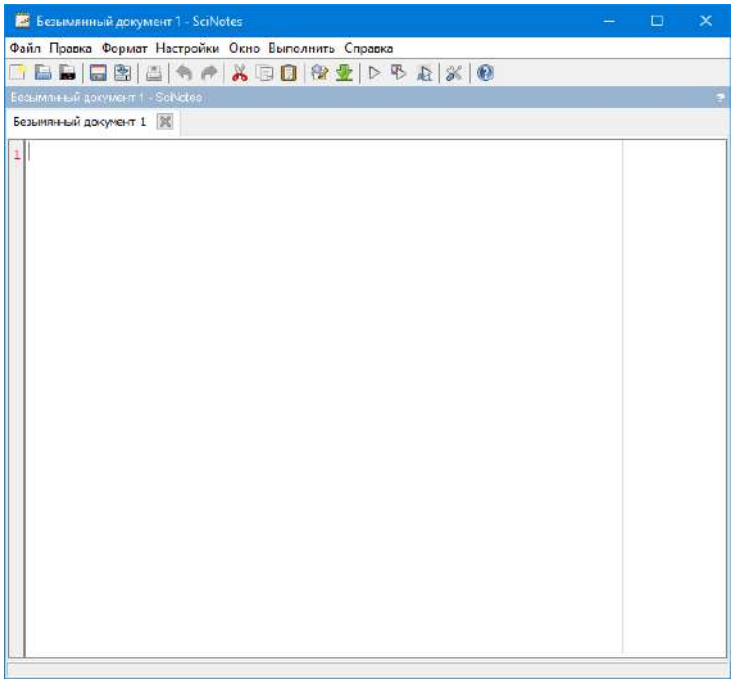
5.9  «Xcos» - это встроенная графическая интерактивная среда, в основе которой лежит блочное моделирование систем в области механики, гидравлики, электроники, а также систем массового обслуживания.

5.10  «Настройки Scilab» - окно настроек программного пакета

5.11  «Примеры» - примеры использования программного пакета, для проведения вычислений, от разработчиков.

5.12  «Справка» - справочная система по программному пакету и встроенному языку программирования.

Написания кода в Scilab происходит в SciNotes, рассмотрим его более подробно. При запуске текстового редактора мы увидим:



Текстовый редактор имеет:

1. Окно ввода, для набора программного кода.
2. Набор инструментов, для работы с программным кодом.

2.1  «Новый» - создает новый безымянный документ в текстовом редакторе.

2.2  «Открыть» - открывает ранее созданный файл

2.3  «Open» - открывает системную папку программного пакета с модулями и стандартными функциями и дает возможность просмотреть как они написаны и посмотреть их тестирования.

2.4  «Сохранить» - сохраняет программный код в файле с расширением .sci.

2.5  «Сохранить как» - сохраняет программный код в указанный нами каталог.

2.6  «Печать» - отправляет принтеру данные из окна программного кода, которые будут напечатаны.

2.7  «Назад» - отменяет последние изменения

2.8  «Повторить» - повторяет только что отмененные изменения

2.9  «Вырезать» - функция является аналогичной стандартной функции Windows «Вырезать». Вырезает часть текста из командного окна.

2.10  «Копировать» - функция копирует из команды необходимый нам текст.

2.11  «Вставить» - функция вставляет из буфера обмена текст который мы ранее копировали.

2.12  «Найти/Заменить» - функция поиска определенного слова или предложения в коде и имеющая возможность заменить все эти слова или предложения, или только лишь одно определенное место.

2.13  «Пошаговый поиск» - в программировании именуемый как «отладчик», пошаговый поиск определенных ошибок в ходе.

2.14  «Выполнить» - запуск программного кода.

2.15  «Сохранить и выполнить» - вначале производит сохранения, после чего происходит выполнения.

2.16  «Сохранить и выполнить все» - вначале сохраняет все открытые файлы, после чего производит их выполнение.

2.17  «Настройки Scilab» - окно настроек программного пакета

2.18  «Справка» - справочная система по программному пакету и встроенному языку программирования.

Работа с Scilab в режиме диалога

Сеанс работы с с Scilab по аналогии с Matlab будем называть сессией (**session**). В сессии имеются строки ввода, вывода и сообщений об ошибках. Входящие в сессию определения переменных и функций, расположенные в рабочей области памяти, можно записать на диск, используя команду **save**. Команда **load** позволяет считать с диска данные рабочей области. Фрагменты сессии можно оформить в виде дневника с помощью команды **diary**.

Основное окно системы Scilab – это командное окно (**Command Window**). В нем можно вводить команды, и в него Scilab выводит результат выполнения этих команд и свои служебные сообщения. Очистить командное окно можно, нажав клавишу F2.

Система Scilab позволяет любые вычисления выполнять в интерактивном режиме. Работа с системой в этом случае реализуется по принципу «задал вопрос – получил ответ». Пользователь набирает на клавиатуре вычисляемое выражение, редактирует (при необходимости) его и завершает ввод нажатием клавиши **Enter**.

Если система готова к вводу данных, в командном окне появляется символ –> в начале строки. Данные вводят с помощью простейшего строчного редактора. Для блокировки вывода результата вычислений некоторого выражения после него нужно поставить ; (точку с запятой). Если не указать имя переменной, которой надо присвоить значение результата вычислений, то Scilab присвоит этой переменной имя **ans**. В качестве знака присваивания

в системе используется знак равенства =. Встроенные функции (например, **sin**) вводят строчными буквами и указывают их аргументы в круглых скобках. Результат вычислений выводится в строках вывода (без знака –>). Переменные a и A в среде Scilab – это разные переменные.

Примеры:

```
–> v=[1 2 3 4]
v =
1. 2. 3. 4.
–> m=[1, 2; 3, 4]
m =
1. 2.
3. 4.
–> sin(v)
ans =
0.8414750 0.9092974 0.1411200 -0.7568025
–> 3*v
ans =
3. 6. 9. 12.
```

Две записи вектора v=[1 2 3 4] и v=[1, 2, 3, 4] являются идентичными.

В некоторых случаях вводимое математическое выражение может не уместиться в одной строке. Часть выражения можно перенести на новую строку с помощью знака двоеточие, например:

```
–> s=1+2+1/3+..
+4
s =
7.3333333
```

Математические выражения в Scilab

Математические выражения состоят из чисел, констант, переменных, операторов, функций и спецзнаков. Числа могут быть целыми, дробными, с фиксированной и плавающей точкой. Примеры: -3 2.453 123.12e-3. Последнее число – это $123.12*10^{-3}$, т.е. 0,12312. **Для разделения целой и десятичной части числа используется точка**. Числа могут быть вещественными и комплексными. Кроме того, в Scilab существуют так называемые системные переменные и символьные константы:

- **%i** – мнимая единица (%i=$\sqrt{-1}$);
- **%pi** – число π=3,1415927;
- **%e** – число exp=2,71828184;
- **%eps** – 2.22d-16;
- **%inf** – значение машинной бесконечности;
- **ans** – переменная, хранящая результат последней операции;
- **%nan** – указание на нечисловой характер данных (**not-a-number**).

Примеры:

```
–> 1/0
! – error 27
division by zero
–> 0/0
```

! – error 27
division by zero

Системные переменные нельзя переопределить, например, переменной %eps нельзя присвоить другое значение:

–>%eps = 0.1
! – error 13
redefining permanent variable

Символьная константа – это цепочка символов, заключенная в апострофы, например, 'Hello'.

В Scilab наглядность описания сложных выражений достигается с помощью текстовых комментариев, которые вводят с помощью символов //.

Переменная в Scilab может иметь имя, содержащее сколько угодно символов, но система запоминает и идентифицирует только первые 24 символа. Имя должно начинаться с буквы и может содержать буквы, цифры и символ подчеркивания _.

В памяти компьютера переменные занимают определенное место, называемое рабочим пространством (**Workspace**). Для очистки рабочего пространства используют функцию **clear**:

- **clear** – уничтожение определений всех переменных;
- **clear x** – уничтожение определения переменной x;
- **clear a, b, c** – уничтожение определений нескольких переменных.

Примеры:
–> v=[1 2 3 4 5];
–> clear v
–> v
!- - error 4
undefined variable : v

Операторы. Функции. Форматированный вывод

Большинство операций в Scilab являются матричными операциями, а соответствующие им операторы относятся к матричным операторам. Например, с помощью операторов умножения * и деления / вычисляют произведение и частное от деления двух массивов (векторов или матриц). Есть ряд спецоператоров, например, оператор \ используют для деления справа налево, а операторы .* и ./ – для поэлементного умножения и деления массивов.

Примеры:
–> v1=[2 4 6 8]; v2=[1, 4, 12, 24]; p=v1/v2, t=v1.*v2, r=v1./v2, h=v1.\v2
p =
0.3826323
t =
2 16 72 192
r =
2. 1. 0.5 0.3333333
h =
0.5 1. 2. 3.
–>(2*1+4*4+6*12+8*24)/(1^2+12^2+4^2+24^2)
ans =
0.3826323
Обратите внимание на результат операции:

–>x=[1 2 3 4 5 6];y=1/x

y =

0.0109890

0.0219780

0.0329670

0.0439560

0.0549451

0.0659341

В данном случае вычисляются не величины, обратные элементам вектора x, а каждый элемент вектора x делится на сумму квадратов всех элементов вектора. Результат же операции y=x^(-1) дает то, что надо:

–>x=[1 2 3 4 5 6];y=x^(-1)

y =

1. 0.5 0.3333333 0.25 0.2 0.1666667

Функции – это имеющие уникальные имена подпрограммы, выполняющие определенные преобразования над своими аргументами и при этом возвращающие результаты этих преобразований.

Функции (макросы) в Scilab похожи на те, что встречаются в других языках программирования. Функции могут иметь аргумент, сами являться аргументом другой функции, быть членом списка, участвовать в операциях сравнения, вызываться рекурсивно. Функция начинается со слова **function** и заканчивается словом **endfunction**.

Первая строка функции может быть следующей:

function var=my_ name(x1,...,xk),

где **var** - имя переменной, а **xi** - входные переменные.

Ниже приведен **пример** функции, вычисляющей сумму положительных элементов в массиве v.

```
function g=f(v)
s=0;  n=length(v);
for i=1:n
if  v(i)>0  then
s=s+v(i);
end
end
g=s;
endfunction
```

Для использования этой функции ее нужно сначала сохранить на диске в файле с именем **f.sci**, выполнить пункт меню Execute/Load into Scilab, а затем вызвать ее:

–> x=[1 2 5 -3 7 -9 12]; t=f(x)

t =

27

Если функция должна возвращать несколько значений, то ее надо определить в формате:

function[y1, y2, . . .] = func(x1,x2,. . .)

. . . . . тело функции

endfunction

Здесь **y1**, **y2**, . . . – список выходных аргументов, **x1**, **x2**, . . . – список входных аргументов, **func** – имя функции.

Функцию средствами пакета Scilab можно создать так:

С помощью команды **deff**.

Пример. Создадим в редакторе функцию с именем **fun** двух аргументов **t** и **y**, результатом которой будет трехмерный вектор, первый элемент которого равен **t+y**, второй элемент равен **t-y**, а третий элемент равен **t*y**.

```
deff('[w]=fun(t,y)',[
'w(1)=t+y;';
'w(2)= t-y;'; 'w(3)= t*y;']) //Вызовем эту функцию
q=fun(5,7)
```

В математических выражениях часто встречается оператор : (**двоеточие**), имеющий следующий формат:

Начальное_ значение: Шаг: Конечное_ значение

Если Шаг не указан, то считается, что он равен 1.

Примеры:

```
-> 1:3
ans =
1. 2. 3.
-> j=10:-2:2
j =
10. 8. 6. 4. 2.
-> x=1:.2:1.4, sin(x)
x =
1. 1.2 1.4
ans =
0.8415470 0.9320391 0.9854497
```

Если в математическом выражении допущена ошибка или предписываемые вычисления некорректны, Scilab выводит в командное окно соответствующие сообщения. Для возвращения к ранее набранным строкам с целью их корректировки используют клавиши ↑ и ↓.

По умолчанию Scilab представляет результат вычислений с 8 значащим цифрами. Для того, чтобы контролировать количество выводимых разрядов числа на печать, можно использовать команду printf с заданным форматом.

Примеры:

```
->c=678.5556696777888899 // Будет напечатано 8 цифр
c =
678.55567
->printf("%4.8f",c)
678.55566968
```

Аналогичный результат дадут команды printf("%1.8f",c), printf("%5.8f",c)

```
->printf("%4.1f",c)
678.6
->d=56.6789
->printf('%3.6f',d) // 6 знаков после запятой
56.678900
->c=678.55566969;d=56.6789;
->printf('%4.3f %4.1f',c,d)
678.556 56.7
```

Для завершения работы с системой можно использовать команды **Quit** и **Exit**.

Тема 2. Матричные операторы линейной алгебры

Основы редактирования и отладки sci-файлов

Для набора, редактирования, отладки и запуска sci-файлов служит специальный редактор, который можно вызвать из командной строки (команда **scipad();**), либо командой Editor (кнопка **SciNotes** в пятой версии). Для запуска файла его необходимо предварительно записать на диск, используя команду **Save As** в меню **File** редактора. После записи файла на диск его надо загрузить в среду Scilab командой редактора **Execute/Load into Scilab** или из главного меню Scilab вызвать команду **Exec** и указать имя файла-сценария.

С помощью редактора, в частности, можно устанавливать в тексте файла специальные метки – точки прерывания.

Операторы и функции

Некоторые специальные символы:

a(: , j) – j-й столбец матрицы **a**;

a(i , :) – i-я строка матрицы **a**; **a(j : k)** – элементы a_j, a_{j+1}, . . . , a_k; **a(:)** – записывает все элементы массива **a** в виде столбца; **a(m , :) =[ ]** – удаляет из матрицы строку **m**; **a'** – транспонированная матрица **a**; **a.'** – транспонирование массива; **prod(a)** – произведение элементов массива; **prod(a, dim)** – произведение элементов столбцов (**dim=1**) или строк (**dim=2**) ; **sum(a)** – сумма элементов массива; **sum(a, dim)** - сумма элементов столбцов (**dim=1**) или строк (**dim=2**).

Матричные операции линейной алгебры

- **det(a)** – возвращает определитель квадратной матрицы **a**;
- **rank(a)** – возвращает ранг матрицы;
- **norm(a)** – возвращает норму матрицы **a**;
- **b=orth(a)** – возвращает ортонормальный базис матрицы **a**;
- **inv(a)** – возвращает матрицу, обратную матрице **a**;
- **spec(a)** – возвращает вектор собственных значений матрицы **a**.

Примеры:

Пусть $d = \begin{pmatrix} 2 & 9 & 9 & 4 \\ 2 & -3 & 12 & 8 \\ 4 & 8 & 3 & -5 \\ 1 & 2 & 6 & 4 \end{pmatrix}$, $c = \begin{pmatrix} 1 & -2 & -3 & 0 \\ 2 & 3 & 8 & 7 \\ -1 & 1 & 1 & -1 \end{pmatrix}$, $x = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \\ 4 & 1 & 3 \end{pmatrix}$.

Вычислить: определитель матрицы d; ранг матрицы c; обратную матрицу к матрице x.

Решение:

В командном окне в строках ввода набираем:

–> d=[2 9 9 4; 2 -3 12 8; 4 8 3 -5; 1 2 6 4];

–> c=[1, -2, -3, 0; 2, 3, 8, 7; -1, 1, 1, -1];x=[3 2 1; 1 0 2; 4 1 3];

–> x1=det(d), x2=rank(c), x3=inv(x)

В строках вывода получаем:

```
x1 =
       147.
x2 =
        2.
x3 =
       -0.4    -1.    0.8
        1.       1.     -1.
        0.2     1.    -0.4
```

Решение систем линейных уравнений

Пример: решить систему линейных уравнений

$$\begin{cases} 2x_1 + x_2 + x_4 = 8 \\ x_1 - 3x_2 + 2x_3 + 4x_4 = 9 \\ -5x_1 - x_3 - 7x_4 = -5 \\ x_1 - 6x_x + 2x_3 + 6x_4 = 0 \end{cases}$$

Решение возможно одним из способов (s_1, s_2, s_3 или s_4 – см. приведенную ниже программу).

```
--> a=[2, 1, 0, 1;  1, -3, 2, 4;   -5, 0, -1, -7;  1, -6, 2, 6];  b=[8 9 -5 0];
--> s1=b/a',  s2=a\b',  s3=b*a'^(-1),  s4=b*inv(a')
s1  =
    8.1481481  - 1.5185185    11.703704  - 6.7777778
 s2  =
    8.1481481
  - 1.5185185
    11.703704
  - 6.7777778
 s3  =
    8.1481481  - 1.5185185    11.703704  - 6.7777778
 s4  =
    8.1481481  - 1.5185185    11.703704  - 6.7777778
```

Решить систему линейных уравнений вида $a\vec{x} - \vec{b} = 0$ можно с помощью функции linsolve : linsolve(a,b). В нашем примере:

```
a=[2, 1, 0, 1;  1, -3, 2, 4;   -5, 0, -1, -7;  1, -6, 2, 6];
b=[8; 9 ;-5 ;0]; x=linsolve(a,-b)
x  =
8.1481481
   - 1.5185185
   11.703704
   - 6.7777778
```

Если система уравнений имеет бесчисленное множество решений, то выводится одно из них:

```
a=[2, 1; 4, 2];  b=[-7; -14 ];x=linsolve(a,b)
x  =
    2.8
    1.4
```

Если система не имеет решений:

```
a=[2, 1; 4, 2];  b=[-7; -13 ];x=linsolve(a,b)
WARNING:Conflicting linear constraints!
 x  =
     []
```

Вычисление корней полинома

Функция **roots(c)** возвращает вектор-столбец из корней полинома **c**.

Пример: решить уравнение $7x^3 + 12x + 23 = 0$

```
--> x=[7, 0, 12, 23];  d=roots(x)
d  =
    0.5564046 + 1.6257442i
    0.5564046 - 1.6257442i
  - 1.1128093
```

Примечание: Коэффициенты полинома следует вводить в порядке убывания степеней переменной x. Если в уравнении отсутствует слагаемое, содержащее, например, x2, то в векторе коэффициентов на соответствующем месте надо ввести 0.

Решение нелинейных уравнений вида f(x)=0

Уравнения бывают алгебраическими и трансцендентными. Алгебраическим называют уравнение вида $a_0+a_1x+a_2x^2+...+a_mx^n = 0$. Если уравнение нельзя свести к алгебраическому заменой переменных, то его называют трансцендентным. Пример: $x^3 \sin x - \log x - 24 = 0$

Для решения уравнений, в том числе трансцендентных, в Scilab применяют функцию fsolve(x0,f), где x0 - начальное приближение, f - функция, описывающая левую часть уравнения f(x)=0.

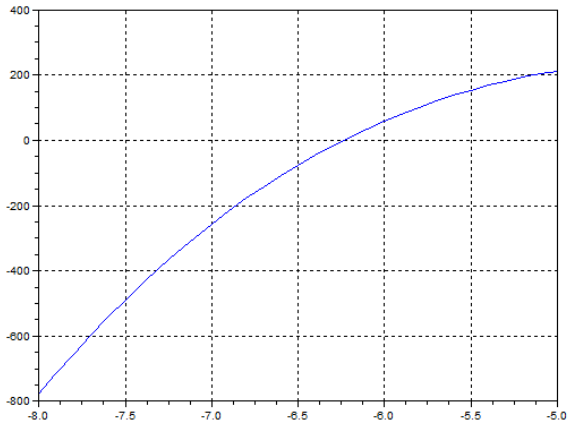
Пример: решить уравнение $7x^3 + 45x^2 + 12x + 23 = 0$. Набираем в окне редактора файл:

```
function y=f(x)
y=7*x.^3+45*x.^2+12*x+23;
endfunction
```

и сохраняем его под именем **f.sci**. Загружаем его в Scilab (**Execute/Load into Scilab**).

Для нахождения отрезка [a, b], на котором отделен корень данного уравнения, построим график функции $y = 7x^3 + 45x^2 + 12x + 23$.

->x=-8:0.1:-5; plot(x, f(x)); xgrid()



Из графика видно, что корень отделен на отрезке [-6.5 , -6]. Найдем его, используя функцию **fsolve**:

```
-->x0=-6.5;x1= fsolve(x0,f)
Получаем:
 x1  =
  - 6.2381997
```

Систему нелинейных уравнений также можно решить, используя функцию **fsolve**.

$$\begin{cases} x^2 + y^2 = 1 \\ x^3 - y = 0 \end{cases}$$

```
clc
function [y]=ff(x)
  y(1)=x(1)^2+x(2)^2-1;
  y(2)=x(1)^3-x(2);
endfunction
t=fsolve([-.5,-.5],ff)
t  =
  - 0.8260314  - 0.5636242
```

Поиск минимума функции y=f(x) на интервале [a, b]

Функция **[f,xopt]=optim(costf,x0)** возвращает локальный минимум функции **costf**.

Функция возвращает минимум функции (значение f) и точку, в которой этот минимум достигается (xopt).

Главной особенностью функции optim является структура функции costf, которая должна быть такой:

function [f,g,ind]=costf(x,ind)
f=gg(x);//функция, минимум которой мы ищем
g=numdiff(gg,x); //градиент функции f
endfunction

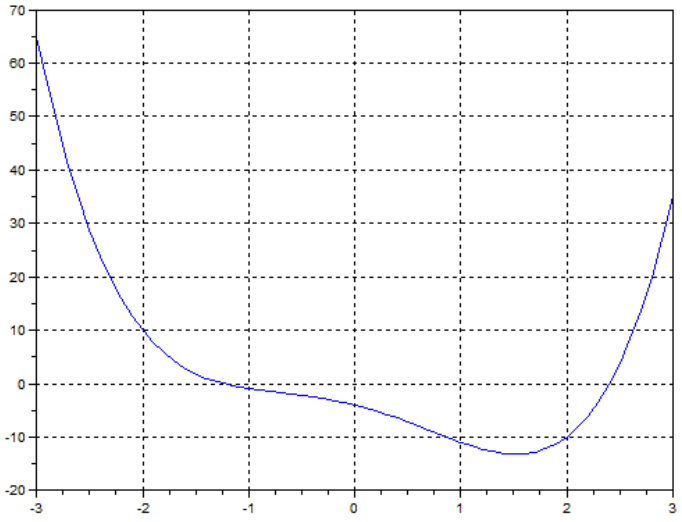
Если возвращаемое сформированной функцией *costf* значение *ind* равно 2, 3 или 4, то функция *costf* обеспечивает поиск минимума, т.е. в качестве результата функции *optim* возвращается f и xopt. Если *ind*=1, то в функции *optim* ничего не считается, условие

ind<0 означает, что минимум f(x) не может быть оценен, а *ind*=0 прерывает оптимизацию. Вообще говоря, значение параметра *ind* является внутренним параметром для связи между *optim* и *costf*, для использования *optim* необходимо помнить, что параметр *ind* должен быть определен в функции *costf*.

Пример: найти минимум функции $y = x^4 - 3x^2 - 5x - 4$.

Решение. Построим график функции для определения интервалов [a, b], на которых находятся экстремумы этой функции.

*–>x=-3:.1:3; y=x.^4-3*x.^2-5*x-4; plot(x, y); xgrid()*



Из графика видно, что это отрезок [1, 2]. Набираем в окне редактора и отправляем на выполнение файл

```
function [f,g,r]=z(x,r)
f=x.^4-3*x.^2-5*x-4
g=4*x.^3-6*x-5
endfunction
x0=1;
[fmin,xmin]=optim(z,x0)
```

Получаем

```
--> xmin  =
1.5233402
fmin  =
- 13.193373
```

Возможен другой вариант, без ручного вычисления производной:

```
function y=gg(x)
y=x.^4-3*x.^2-5*x-4;
endfunction
function [f,g,r]=z(x,r)
f=gg(x)
g=numdiff(gg,x)
```

```
endfunction
x0=1;
[fmin,xmin]=optim(z,x0)

xmin  =
1.5233402
fmin  =
- 13.193373
```

В случае функции двух переменных: (Поиск минимума функции Розенброка $f(x,y) = 100(y-x^2)^2 + (1-x)^2$)

```
clc
x0=[-2;2];
function y=gg(x)
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
function [f,g,r]=z(x,r)
f=gg(x)
g=numdiff(gg,x)
endfunction
[fmin,xmin]=optim(z,x0)
xmin  =
0.9999955
    0.9999910
fmin  =
     2.010D-11
```

Тема 3. Построение графиков функций одной переменной и графиков трехмерных поверхностей

Построение графиков в прямоугольной системе координат

Пусть x принимает значения от 0 до 10 с шагом 0,1. Чтобы построить график функции *y=sin(x)*, можно набрать следующую строку:

–>x=0:0.1:10; plot(x, sin(x))

Для построения графиков сразу нескольких функций (например, $y_1 = \sin x$; $y_2 = \frac{\sin x}{x+1}$) **в одном графическом окне** можно воспользоваться разновидностью команды **plot**:

–> x=0:0.1:2%pi; y1=sin(x); y2=sin(x)./(x+1); plot(x, y1, x, y2)*

График будет построен линиями разного цвета. Аналогичный результат можно получить, используя два оператора plot вместо одного:

–>x=0:0.1:2%pi; y1=sin(x); y2=sin(x)./(x+1); plot(x, y1), plot(x, y2)*

Линии в этом случае будут одного цвета.

Очистить графическое окно можно операторами xbasc(), xclear() или clf().

Пример:

–>xbasc(1:3) //очищаются три окна с номерами 1, 2 и 3.

–>xbasc([1,3,5]) // очищаются окна с номерами 1, 3 и 5.

Оператором xbasc() очищается текущее окно.

Уничтожаются изображения, а не сами окна.

Для построения графиков функций в разных графических окнах используют оператор xset.

Пример:

–>x=0:.1:2%pi; y=sin(x); plot(x,y); xset('window',1); z=cos(x); plot(x, z)*

Заголовок текущего окна можно изменить командой xname.

Пример:

В графическом окне есть своя панель меню, позволяющая открывать новые окна (File/New), записывать содержимое окна в родном формате **.scg** (**File2„ - Save**), конвертировать в другие форматы (File - Export), загружать в окно картинки в формате **.scg** с диска (**File - Load**), печатать на принтере и др.

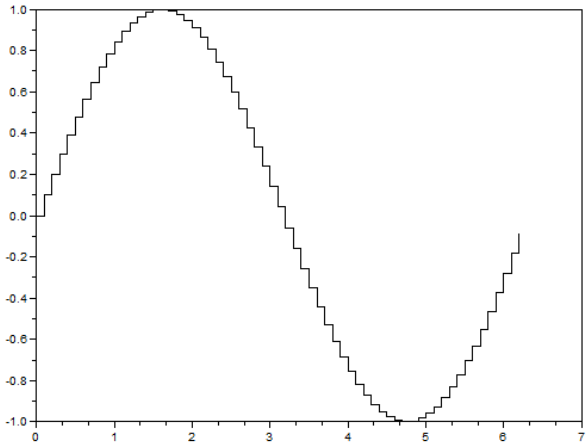
Команды меню **Zoom** и **UnZoom** служат для увеличения или уменьшения масштаба в графическом окне.

Команда меню **3D Rot** позволяет вращать 3D-объекты.

В результате работы программы

```
x=[0:0.1:2*%pi]';
xbasc()
plot2d2(x,sin(x))
```

получим



Добавить к графику сетку можно командой **xgrid**.

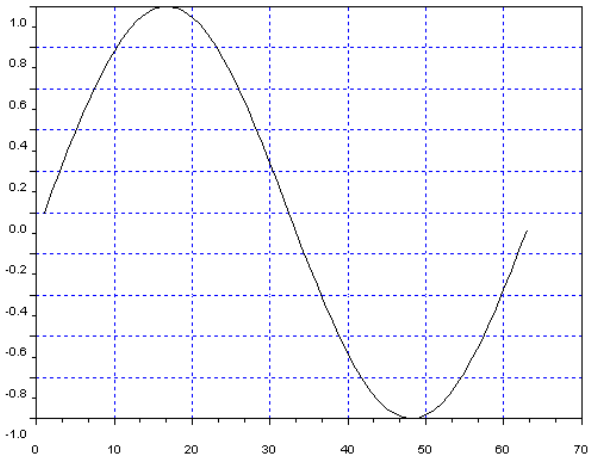
Синтаксис **xgrid([style])**

Параметры

style : целое число. Указывает на стиль изображения вспомогательной сетки. Это - цвет или тип черной штрихованной линии, предварительно задаваемый с помощью команды **xset()** для текущего графического окна.

Пример:

```
xbasc()
x=[0:0.1:2*%pi]';
plot2d(sin(x));
xgrid(2)
```



Для редактирования графическое окно имеет специальный инструментарий, который содержится в разделе **Edit** главного меню окна

- **Figure Properties** – свойства графика;
- **Current Axes Properties** – установка параметров осей координат;
- **Erase figure** - стереть график;

Выполнив команду **Figure Properties**, мы получаем окно, в котором можем задать, что именно мы хотим редактировать: **figure**, **axes**, **compound**, **polyline**.

Выбрав вариант **figure**, мы можем поменять название графика, его размещение на экране (X position=0, Y position=0 – левый верхний угол); размеры графика, сжав его по любой из осей; задать цвет заднего плана, поменять цветовую гамму.

Выбрав редактирование осей **axes**, мы можем редактировать каждую из осей X, Y, Z: задать название оси (**Label:**); цвет (при этом меняется цвет у названия оси, сетки и разметки оси); задать размещение названия оси: наверху (**: top**) или внизу (**: bottom**) или в середине (**: middle**)– для оси X , слева, в середине или справа – для оси Y; цвет метки, угол поворота, ее размер, фонт, задать расстояние между линиями сетки, установить насечку между линиями, задать заголовок графика, его цвет, размер, стиль, установить цвета разметки осей, цвет внутренней части графика, размер и вид сетки. Пункты меню **Aspect Isoview** и **Tight limits** позволяют менять расположение графика относительно прямоугольной области, внутри которой он построен. Пункт меню **Boxes** позволяет задать рамку для этой области, **Viewpoint** – повернуть график на определенный угол.

Выбрав последовательно подпункты меню **Figure Properties axes** и **compound**, можно обнаружить подпункт **polyline**, в котором можно поменять вид и размеры маркеров и аппроксимировать наш график различными кривыми.

В меню **File** есть команда **Copy to clipboard** – копирование рисунка в буфер обмена, откуда потом его можно вставить в Word или другой текстовый редактор.

Рассмотрим другие варианты команды plot:

- **plot(x,y,s)** строит график функции **y(x)**, координаты точек **(x,y)** которой берутся из векторов одинакового размера **x** и **y**. Тип линии графика можно задать с помощью строковой константы **s**;
- **plot(x1, y1, s1, x2, y2, s2, ...)** – строит в одном графическом окне несколько графиков.

Пример:

```
-->x=-2*%pi:.1*%pi:2*%pi;

-->y1=sin(x);y2=sin(x).^2;plot(x,y1, '-m', x, y2, '-.+r')
```

График функции y_1 строится сплошной фиолетовой линией, график функции y_2 строится штрихпунктирной линией с точками в виде знака «+» красного цвета.

Графики в полярной системе координат

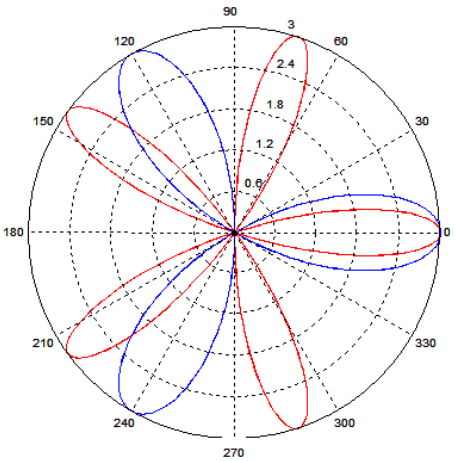
В полярной системе координат любая точка представляется как конец радиус-вектора, исходящего из начала системы координат, имеющего длину ρ и угол θ. Для построения графика функции $\rho(\theta)$ используется команда **polarplot**:

- **polarplot(theta, rho,[key1=value1,. . . ,keyn=valuen])** – строит график, представляющий собой положение конца радиуса-вектора длиной **rho** и углом **theta**.

Пример: Построить графики двух функций:
$\rho = 3\cos 5\phi$ и $\phi_1 = 3\cos 3\phi$

Программа:

```
fi=0:.01:2*%pi;
ro=3*cos(5*fi);ro1=3*cos(3*fi);
polarplot(fi,ro,style=color("red"));
polarplot(fi,ro1,style=color("blue"));
```



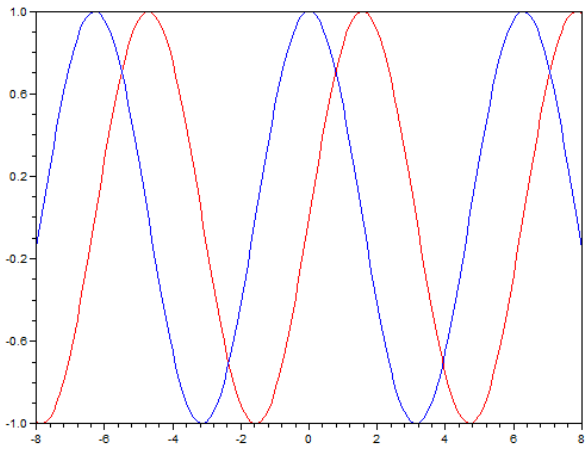
Приведем еще несколько примеров построения двумерных графиков.

Построить графики функций y=sin(x) и y1=cos(x) . Модифицировать масштаб координатных осей графика.

Сформируем массив X приняв, что x изменяется в диапазоне [-8:8] с шагом 0,1, затем совместно сформируем массивы значений заданных функций с помощью следующей записи y=[*sin(x)*; *cos(x)*].

С помощью функции *plot2d* построим графики функций y=sin(x) и y1=cos(x), установив значение параметра *nax*=[4,9,3,6]. Таким образом, ось X будет разбита 9 основными делениями (засечками), каждое основное 4 промежуточными, а ось Y соответственно - 6 и 3.

```
x=[-8:0.1:8];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),color("blue")],axesflag=1, nax=[4,9,3,6]);
```



axesflag - значение параметра *keyn=valuen* функции plot2d - определяет наличие рамки вокруг графика. Необходимо выделить следующие базисные значения этого параметра:

0 - нет рамки, нет изображения осей;

1 или 4 - изображение рамки, ось y слева (по умолчанию);

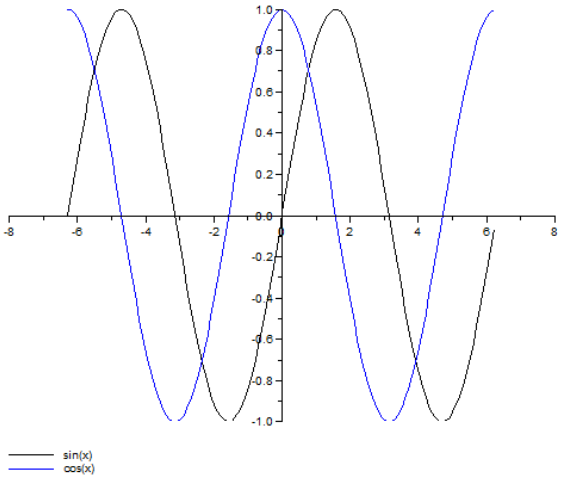
2 - изображение рамки, изображения осей нет;

3 - изображение рамки нет, ось y справа;

5 - изображение осей, проходящих через точку (0,0).

Построим графики функций y=sin(x) и $y1$=cos(x) с пересечением осей X и Y в точке (0,0) - значение параметра *axesflag*=5, выведем легенду с подписями для обеих кривых

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',axesflag=5, leg="sin(x)@cos(x)");
```



Построение графиков трехмерных поверхностей

Для построения трехмерных графиков используют операторы **plot3d**, **plot3d1** и **mesh**.

Обращение к ним следующее:

plot3d(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen]),

plot3d1(x,y,z,[theta,alpha,leg,flag,ebox][keyn=valuen])

Здесь x - вектор - столбец значений абсцисс;

y - вектор - столбец значений ординат;

z - матрица значений функции;

theta, *alpha* - действительные числа, которые определяют в градусах сферические координаты угла зрения на график. Попросту говоря, это угол, под которым наблюдатель видит отображаемую поверхность;

leg - подписи координатных осей графика - символы, отделяемые знаком @. Например , 'X@Y@Z'.

flag – массив, состоящий из 3 целочисленных параметров [*mode*,*type*,*box*]. Здесь *mode* устанавливает цвет поверхности.

Значения параметра mode

>0 - поверхность имеет цвет «mode», выводится прямоугольная сетка.

0 - выводится прямоугольная сетка, заливка отсутствует (белый цвет).

<0 - поверхность имеет цвет «mode», отсутствует прямоугольная сетка.

По умолчанию, равен 2 - цвет заливки синий, прямоугольная сетка выводится.

type - позволяет управлять масштабом графика, по умолчанию имеет значение 2.

Значения параметра *type*

0 - применяется способ масштабирования, как у ранее созданной графики.

1 - границы графика указываются вручную с помощью параметра ebox.

2 - границы графика определяют исходные данные.

Box - определяет наличие рамки вокруг отображаемого графика. По умолчанию равен 4.

Значения параметра box

0 и 1 - нет рамки

2 - только оси, находящиеся за поверхностью

3 - выводится рамка и подписи осей

4 - выводится рамка, оси и их подписи.

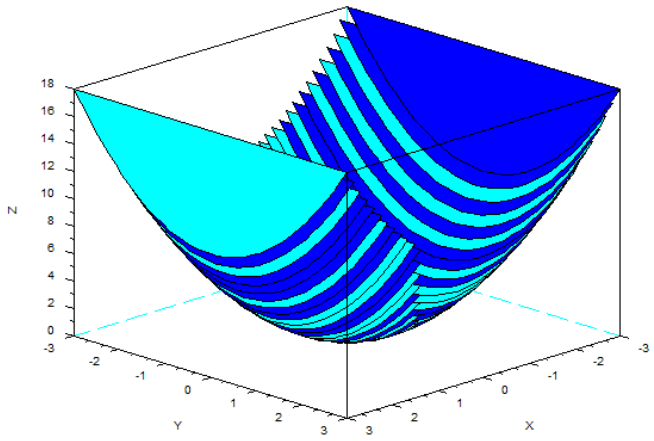
ebox - определяет границы области, в которую будет выводиться поверхность, как вектор [*xmin*,*xmax*,*ymin*,*ymax*,*zmin*,*zmax*]. Этот параметр может использоваться только при значении параметра *type*=1.

keyn=*valuen* - последовательность значений свойств графика *key1*=*value1*,*key2*=*value2*, ...,*keyn*=*valuen*, таких как толщина линии, ее цвет, цвет заливки фона графического окна, наличие маркера и др. Таким образом, функции *plot3d* (*plot3d1*) в качестве параметров необходимо передать прямоугольную сетку и матрицу значений в узлах сетки.

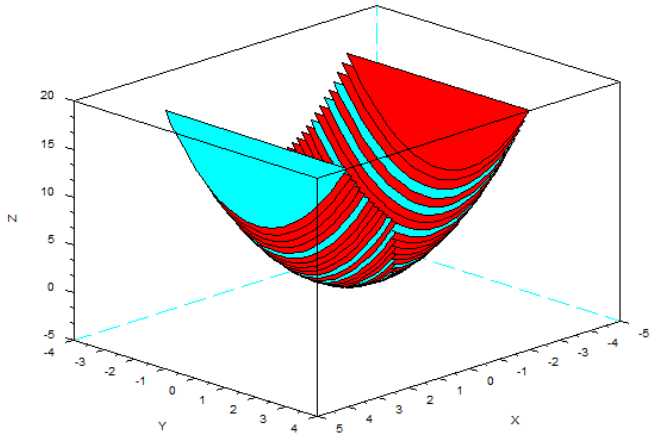
Примеры:

```
-->[x, y]=meshgrid([-3:0.15:3]);  z=x.^2+y.^2;  plot3d(x, y, z)
```

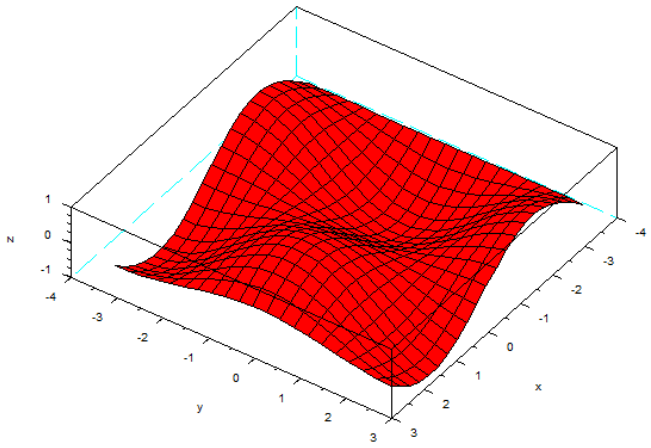
Задается опорная плоскость для построения трехмерной поверхности, переменные x, y меняются от –3 до 3 с шагом 0,15.




```
xbasc()
[x, y]=meshgrid([-3:0.15:3]);  z=x.^2+y.^2;  plot3d(x, y, z,flag=[5,2,3])
Цвет заливки - красный.
xbasc();[x, y]=meshgrid([-3:0.15:3]);  z=x.^2+y.^2;  plot3d(x, y,z,flag=[5,1,4],
ebox=[-5,5,-4,4,-1,20])
```




```
xbasc()
t=-%pi:.3:%pi;
plot3d(t,t,sin(t)'*cos(t),35,45,'x@y@z',[5,2,4]);
```



Для формирования прямоугольной сетки впервые в Scilab 4.0 появилась функция *meshgrid*. Обращение к ней имеет вид:

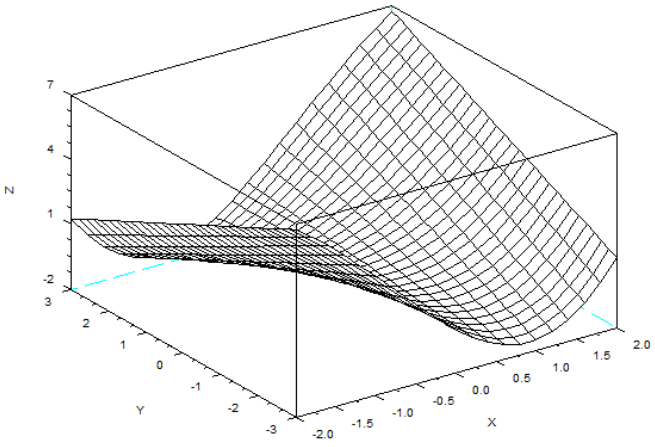
[X,Y[,Z]]=meshgrid(x,y[,z])

здесь *x,y* [,z] - массивы 2(3) исходных параметров X, Y (,Z), указываемые через запятую; X, Y [,Z] - матрицы в случае 2 и массивы в случае 3 входных величин.

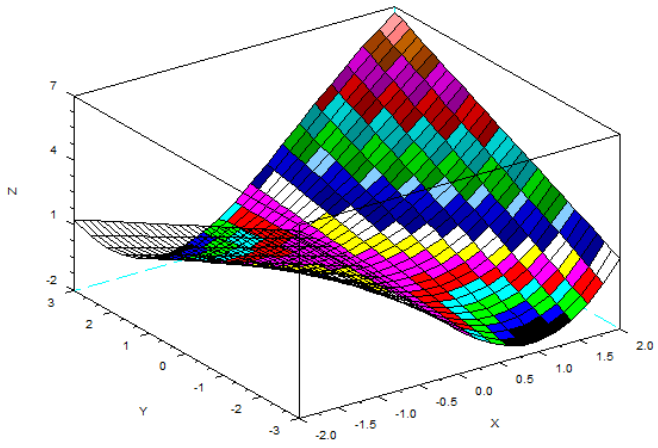
После формирования сетки вывести в нее графику можно с помощью функции *surf* либо *mesh*. Так же, как и в случае с функциями *plot3d* и *plot3d1*, *surf* строит поверхность, заливая каждую ячейку цветом, который зависит от конкретного значения функции в узле сетки, а mesh заливает ее одним цветом.

Таким образом, *mesh* является полным аналогом функции surf со значением параметров *Color mode=индекс белого цвета* в текущей палитре цветов и *Color flag*=0.

```
-->[x, y]=meshgrid(-2:0.2:2, -3:.2:3);  z=x.^2+y.*sin(x);  mesh(x,y,z)
```

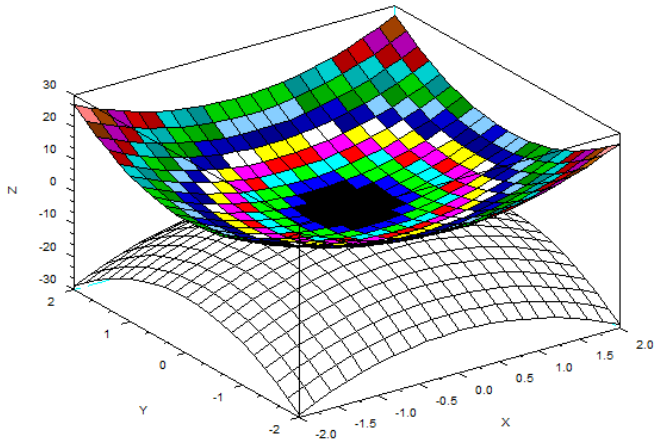



```
-->[x, y]=meshgrid(-2:0.2:2, -3:.2:3);  z=x.^2+y.*sin(x);  surf(x,y,z)
```



Программа, приведенная ниже, показывает, как можно в одном окне построить графики двух трехмерных функций (оператор mtlb_ hold('on');). Правда, и без этого оператора график остается таким же.

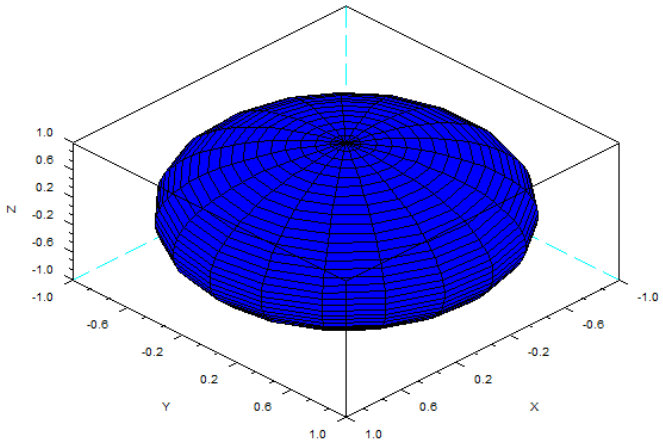
```
[x y]=meshgrid(-2:0.2:2,-2:0.2:2);
z=3*x.^2+4*y.^2-1;
z1=-3*x.^2-4*y.^2-1;
surf(x,y,z);
mtlb_hold('on');
mesh(x,y,z1);
```



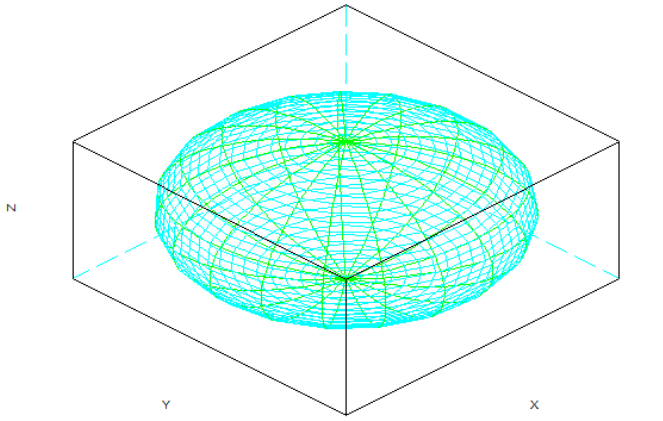
Сферу можно построить так:

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);
```

Оператор v = linspace(0,2*%pi,20);задает вектор v, состоящий из 20 значений с шагом $\phi/10$.



Замена последней строки программы на plot3d3(X,Y,Z); дает



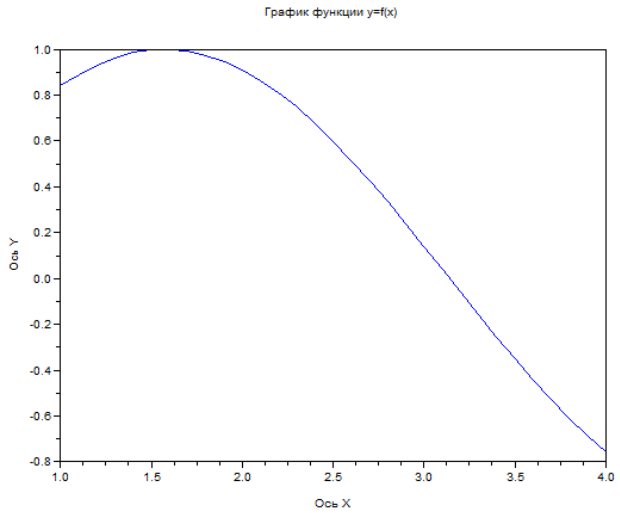
Оформление и комбинирование графиков

После того, как график построен, Scilab позволяет выполнить его форматирование. Для установки над графиком титульной надписи используется команда **title**

title('string')

Пример:

```
-->title('График функции sin(x)')
clc
xbasc()
x=1:.1:4;plot(x,sin(x))
xtitle("График функции y=f(x)", "Ось X","Ось Y")
```



Для установки надписей возле осей x, y и z используются команды **label('string')**:

xlabel('Ось X') ylabel('Ось Y') zlabel('Ось Z')

Для создания легенды используются различные варианты команды **legend**.
Одна из них:

legend(['string1','string2','string3',....],a=pos)

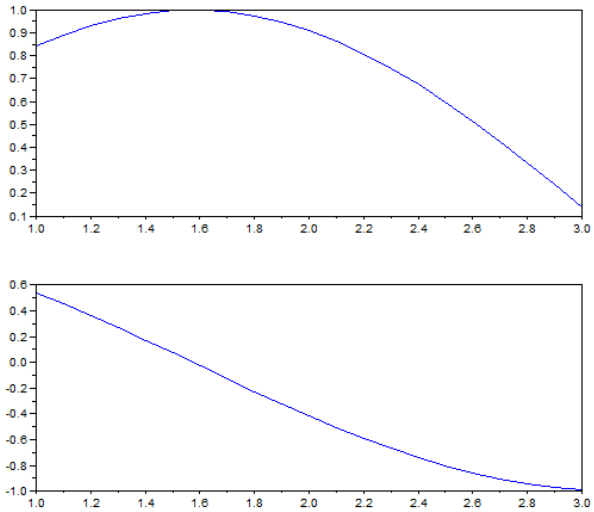
Если pos=1, то легенда помещается в верхний правый угол,
pos=2, то в левый верхний угол,
pos=3, то в левый нижний угол,
pos=4, то в нижний правый угол,
pos=5, легенда перетаскивается мышью.
Пример:
->legend(['sin(x)';'cos(x)';'sin(x)'],a=3);

Для расположения нескольких графиков в одном окне без наложения их друг на друга используется команда **subplot**. Она должна стоять до оператора построения графика.

- subplot(m,n,p) или subplot(mnp) : m окон по горизонтали, n – по вертикали, p – номер окна, в которое будет выводиться текущий график (нумерация ведется по строкам).
- clf reset удаляет все подокна и возвращают графическое окно в обычное положение.

Пример.

```
x=1:.1:3;
subplot(211),plot(x,sin(x));subplot(212),plot(x,cos(x))
```



Построение графиков в виде ступенчатой линии

Для изображения графика в виде ступенчатой линии в Scilab существует функция *plot2d2(x,y)*. Она полностью совпадает по синтаксису с функцией *plot2d*. Главное отличие состоит в том, что X и Y могут быть независимыми друг от друга функциями, важно лишь, чтобы массивы X и Y были разбиты на одинаковое количество интервалов.

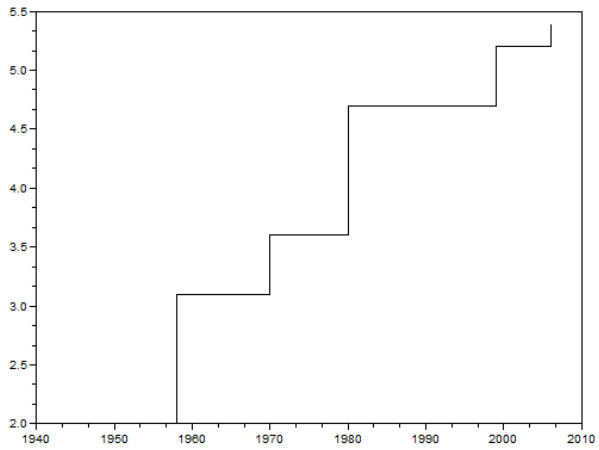
ЗАДАЧА

Имеются детальные наблюдения за ростом народонаселения на планете за период с 1947 по 2006 годы, млн. чел. Построить график, отражающий динамику процесса на основании данных 1947, 1958, 1970, 1980, 1999, 2006 года.

Поэлементно введем массивы X и Y и воспользуемся функцией *plot2d2(x,y)*: x=[1947 1958 1970 1980 1999 2006];

y=[2.003 3.1 3.6 4.7 5.2 5.4];

plot2d2(x,y);

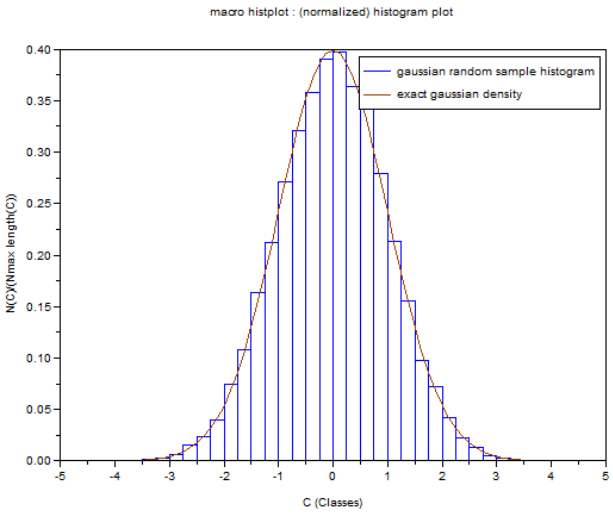


Гистограмму можно построить с помощью оператора histplot.

Пример:

```
histplot([-4.5:0.25:4.5],rand(1,20000,'n'),style=2);
deff('[y]=f(x)','y=exp(-x.*x/2)/sqrt(2*%pi);');
```

```
x=-4.5:0.125:4.5;x=x';plot2d(x,f(x),26,"000");
titre= 'macro histplot : (normalized) histogram plot';
xtitle(titre,'C (Classes)','N(C)/(Nmax length(C))');
legends(['gaussian random sample histogram' 'exact gaussian density'],[2 26],1)
```



Для построения трехмерных гистограмм в Scilab используется функция *hist3d*:
hist3d(f,[theta,alpha,leg,flag,ebox])

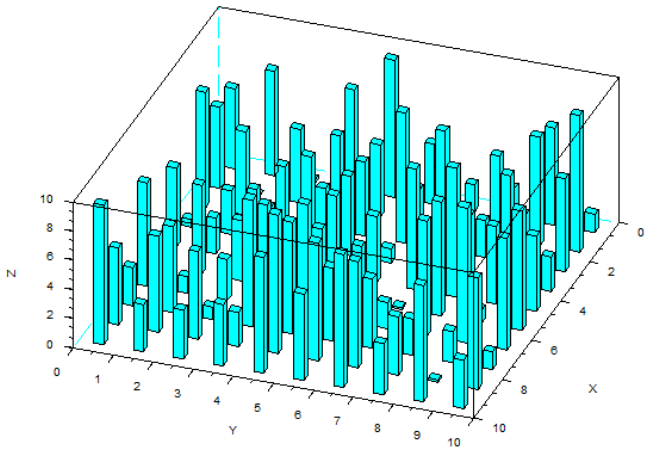
Здесь *f* - матрица (m:n), задающая гистограмму *f (i , j)=F(xi,yj)* Параметры *theta,alpha, leg,flag,ebox* управляют теми же свойствами, как и у функции *plot3d.*

Пример. Построить трехмерную гистограмму при помощи команды *hist3d.*

Для формирования матрицы входных данных воспользуемся командой *rand.* Напомним, чтобы создать матрицу размером *(m,n)*, необходимо использовать конструкцию *rand(m,n)*

Полученная гистограмма изображена на рисунке.

*hist3d(9.7*rand(10,10),20,35);*



Тема 4. Основы программирования в системе Scilab

В Scilab встроен мощный язык программирования с поддержкой объектов. Работа в Scilab может осуществляться как в режиме командной строки, так и в программном режиме. Для создания программы (программу в Scilab иногда называют сценарием) необходимо:

1. Вызвать команду **Editor** из меню.
2. В окне редактора **Scipad** набрать текст программы.
3. Сохранить текст программы с помощью команды **File /Save** в виде файла с расширением **sce**, например **file.sce**.
4. После этого программу можно будет вызвать, набрав в командной строке **exec**, например **exec("file.sce")**, или вызвав команду меню **File/Exec**..., или, находясь в окне Scipad, выполнить команду **Execute/Load into Scilab (Ctrl+l)**.

Программный режим достаточно удобен, так как он позволяет сохранить разработанный вычислительный алгоритм в виде файла и повторять его при других исходных данных в других сессиях. Кроме обращений к функциям и операторов присваивания, в программных файлах могут использоваться операторы языка программирования Scilab (язык программирования Scilab будем называть sci-языком).

Основные операторы sci-языка

Функции ввода-вывода в Scilab

Для организации простейшего ввода в Scilab можно воспользоваться функциями

x=input('title');

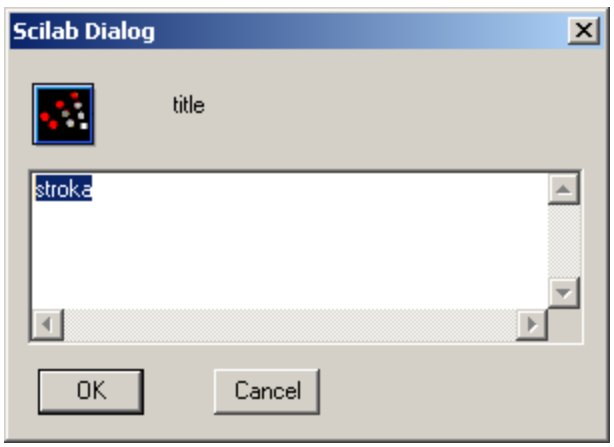
или

x=x_ dialog('title', 'stroka');

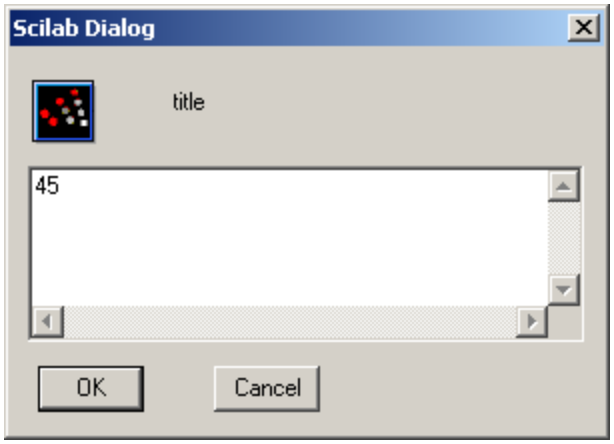
Функция *input* выводит в командной строке Scilab подсказку *title* и ждет пока пользователь введет значение, которое в качестве результата возвращается в переменную x.

```
-->x=input('title');
title-->4
-->x
 x  =
        4.
```

Функция x_ *dialog* выводит на экран диалоговое окно с именем *title*, после чего пользователь может щелкнуть **OK** и тогда *stroka* вернется в качестве результата в переменную x, либо ввести новое значение вместо *stroka*, которое и вернется в качестве результата в переменную x.



Набрав вместо stroka число 45 и распечатав x, получим:




```
-->x
 x  =
        45
```

Функция *input* преобразовывает введенное значение к числовому типу данных, а функция *x_ dialog* возвращает строковое значение. Поэтому при использовании функции *x_ dialog* для ввода числовых значений, возвращаемую ею строку следует преобразовать в число с помощью функции evstr. Поэтому можно предложить следующую форму использования функции *x_ dialog* для ввода числовых значений.

```
x=evstr(x_dialog('title', 'stroka'));
Можно сразу набрать
->x=evstr(x_dialog('title', '46'));
-->x+6
 ans  =
           52.
```

Для вывода в текстовом режиме можно использовать функцию *disp* следующей структуры *disp(b)*. Здесь *b* - имя переменной или заключенный в кавычки текст.

```
-->disp('Привет')
 Привет
-->disp("Привет")
 Привет
```

Оператор присваивания

Оператор присваивания имеет следующую структуру *a=b*, здесь *a* –имя переменной или элемента массива, *b* - значение или выражение. В результате выполнения оператора присваивания переменной *a* присваивается значение выражения *b*.

x=5;y=sin(x);

Условный оператор

Одним из основных операторов, реализующим ветвление в большинстве языков программирования, является условный оператор *if*. Существует обычная и расширенная формы оператора *if* в Scilab. Обычный *if* имеет вид

```
if Логическое_выражение then
       Инструкции_1
       else
```

```
    Инструкции_2
end
```

Инструкции в списке разделяют оператором, (**запятая**) или ; (**точка с запятой**).

Логические выражения

Логические выражения состоят из констант, переменных и функций, соединенных операциями отношения: больше (>), больше или равно (>=), равно (==), не равно (~=) , меньше (<), меньше или равно (<=). Например: x^2>y. Логические выражения принимают значения «истина» или «ложь». Например, при x=2 и y=8 значение выражения x^2>y является ложью. Несколько логических выражений могут быть объединены в одно операторами & и |. Например: x.^2>y | y>7. Если несколько логических выражений соединены оператором & , то значение такого выражения является истиной, если каждое логическое выражение, входящее в него, является истиной. Если несколько логических выражений соединены оператором |, то значение такого выражения является истиной, если хотя бы одно логическое выражение, входящее в него, является истиной.

Зачастую при решении практических задач недостаточно выбора выполнения или невыполнения одного условия. В этом случае можно, конечно, по ветке *else* написать новый оператор *if*, но лучше воспользоваться расширенной формой оператора *if*.

```
if  Логическое_выражение_1 then
      Инструкции_1
elseif    Логическое_выражение_2 then
       Инструкции_2
else
     Инструкции_3
end
```

В этом случае оператор *if* работает так: если логическое выражение_ 1 истинно, то выполняются инструкции_ 1, а затем инструкции, стоящие после оператора **end**. Если логическое выражение_ 2 является истиной, то выполняются инструкции 2, а затем инструкции, стоящие после оператора **end**. Если ни одно из логических выражений не является истиной, то выполняются инструкции, стоящие после слова **else**.

Частные случаи:

1. **if** Логическое_ выражение **then** Инструкции , **end**

   Примеры:

   ```
   -->if x>5 then y=8, end
   -->if x>5 then y=8;t=6; end
   ```

2. **if** Логическое_ выражение **then** Инструкции_ 1, **else** Инструкции_ 2, **end**

   Примеры:

   ```
   -->if x>5 then y=8,t=6; else y=x^2 ,end
   ```

3. ```
   if Логическое_выражение then
         Инструкции
      end
   ```
   ```

*Замечание: слово **then** в составе условного оператора **if** не является обязательным.*

Пример. Вычислить функцию y при x=0.397, если

$$y=\begin{cases}\sin(3x) & x^2 \le 25\cos x^3 \\ \dfrac{2x-tg(x)}{\sqrt{x^2+2}} & x^2 > 25\cos x^3\end{cases}$$

Решение:

```
x=0.397;
if x.^2<=25*cos(x.^3) then
    y=sin(3*x)
else
    y=(2*x - tan(x))./sqrt(x.^2+2)
end
```

Набрав этот файл в редакторе Scipad и запустив его на выполнение командой Execute/Load into Scilab, получим:

```
y  =
    0.9287402
```

В качестве примера программирования разветвляющегося процесса рассмотрим решение квадратного уравнения $ax^2 + bx + c = 0$ Входными данными этой задачи являются коэффициенты квадратного уравнения a, b, c. Выходными данными являются корни уравнения x1 , x2 или сообщение о том, что действительных корней нет.

Алгоритм состоит из следующих этапов:

1. Ввод коэффициентов уравнения a, b и c.
2. Проверка, является ли уравнение квадратным (a≠0).
3. Вычисление дискриминанта уравнения d.
4. Если d=0, то выводится сообщение: уравнение имеет два равных корня и печатается значение корня.
5. Если d>0, определяются x1 и x2
6. Если d<0, то выводится сообщение «Корней нет».

Программа решения квадратного уравнения

```
clc
a=input('a=');
b=input('b=');
c=input('c=');
if a==0 then
disp("Уравнение не является квадратным")
else
// Вычисляем дискриминант.
d=b*b-4*a*c;
// Если дискриминант отрицателен,
if d<0
// то вывод сообщения,
```

```
disp(' Действительных корней нет');
elseif d==0
//иначе-вычисление корней соответствующего
// квадратного уравнения.
disp("уравнение имеет два равных корня") ;x=-b/(2*a)
else
x1=(-b+sqrt(d))/2/a
x2=(-b-sqrt(d))/2/a
end
end

a=-->1
b=-->2
c=-->-5
 x1  =
    1.4494897
 x2  =
  - 3.4494897
```

Найти все корни квадратного уравнения можно и без оператора if, воспользовавшись тем, что в Scilab определены операции над комплексными числами.

```
clc
a=input('a=');
b=input('b=');
c=input('c=');
d=b*b-4*a*c;
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
disp(x1,x2);

a=-->1
b=-->2
c=-->3

  - 1. - 1.4142136i
  - 1. + 1.4142136i
```

Получили комплексные корни квадратного уравнения.

Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор *select* альтернативного выбора следующей структуры:

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
...
else операторы
end
```

Оператор *select* работает следующим образом: если значение параметра равно *значению1*, то выполняются *операторы1*, иначе если параметр равен *значению2*, то выполняются

операторы2; в противном случае, если значение параметра совпадает со *значением3*, то выполняются *операторы3* и т.д. Если значение параметра не совпадает ни с одним из значений в группах *case*, то выполняются операторы, которые идут после служебного слова *else*.

Конечно, любой алгоритм можно запрограммировать без использования *select*, используя только *if*, но использование оператора альтернативного выбора *select* делает программу более компактной.

Рассмотрим использование оператора *select* на примере решения следующей задачи.

Пример. Вывести на печать название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и 1-е число - понедельник.

Для решения задачи воспользуемся условием, что 1-е число - понедельник. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник (первое, восьмое, пятнадцатое, двадцать второе и двадцать девятое числа), двойке - вторник, тройке - среда и так далее. Вычислить остаток от деления числа x на k можно по формуле x-*int(x/k)*x*. Следовательно, при построении алгоритма необходимо использовать семь условных операторов.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором select.

```
D=input('Enter a number from 1 to 31');
//Вычисление остатка отделения D на 7, сравнение его с числами
// от 0 до 6.
select D-int(D/7)*7
case 1 then disp('Monday');
case 2 then disp('Tuesday');
case 3 then disp('Wednesday');
case 4 then disp('Thursday');
case 5 then disp('Friday');
case 6 then disp('Saturday');
else
disp('Sunday');
end
```

Рассмотрим операторы цикла в Scilab. В sci-языке есть два вида цикла - оператор цикла с предусловием while и оператор for.

Оператор while

Оператор цикла *while* имеет вид

while условие

операторы

end

Здесь *условие* - логическое выражение; *операторы* будут выполняться циклически, пока логическое *условие* истинно.

Оператор цикла *while* обладает значительной гибкостью, но не слишком удобен для организации «строгих» циклов, которые должны быть выполнены заданное число раз.

Оператор цикла for используется именно в этих случаях.

Оператор for

Оператор цикла *for* имеет вид

for x=xn:hx:xk

операторы

end

Здесь x - имя скалярной переменной - параметра цикла, xn - начальное значение параметра цикла, xk - конечное значение параметра цикла , hx - шаг цикла . Если шаг цикла равен 1, то hx можно опустить, и в этом случае оператор *for* будет таким.

```
for x=xn:xk
операторы
end
```

Выполнение цикла начинается с присвоения параметру начального значения (x=xn). Затем следует проверка, не превосходит ли параметр конечное значение (x>xk). Если x>xk, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. Если же $x \leq xk$, то выполняются операторы в цикле (тело цикла). Далее параметр цикла увеличивает свое значение на hx (x=x+hx). После чего снова производится проверка значения параметра цикла, и алгоритм повторяется.

Пример. Протабулировать функцию $y = xln^2x$, x принимает значения от 1 до с шагом 0,2.

```
clc
for x=1:.2:2
  y=x*log(x)^2;
  printf("%1.1f   %2.2f",x,y)
  disp('')
  end
1.0   0.00
1.2   0.04
1.4   0.16
1.6   0.35
1.8   0.62
2.0   0.96
```

Циклы **while** и **for** могут быть прерваны с помощью оператора **break**.

Пример.

```
-->a=0; for i=1:5:100, a=a+1; if i>10 then break, end;end
-->a
a = 3.
```

Обработка одномерных массивов и матриц в Scilab

Массив – это совокупность однородных элементов, имеющих одно имя. Одномерный массив – это вектор $\vec{a} = (a_1, a_2, .., a_n)$. Двумерный массив – это матрица

$$b = \begin{pmatrix} b_{11}\, b_{12} \ldots b_{1m} \\ b_{21} b_{22} \ldots b_{2m} \\ \text{----------} \\ b_{n1}\, b_{n2} \ldots b_{nm} \end{pmatrix}$$

размерности n*m.

В системе Scilab принято элементы массивов записывать следующим образом: a(1), a(n), b(3, 7), b(n, m).

Для того, чтобы определить количество элементов в одномерном массиве **x**, используют функцию **length** вида

n=length(x)

Для того, чтобы определить количество строк (**n**) и столбцов (**m**) матрицы b, можно воспользоваться функцией **size**:

[n, m]=size(b)

Рассмотрим возможности sci-языка для обработки массивов и матриц. Особенностью программирования задач обработки массивов (одномерных, двумерных) на sci-языке является возможность как поэлементной обработки массивов (как в любом языке программирования), так и использование функций Scilab для работы массивами и матрицами.

Рассмотрим основные алгоритмы обработки массивов и матриц и их реализацию на sci-языке.

Ввод-вывод массивов и матриц

Ввод массивов и матриц может быть организован как в режиме диалога:

```
clc
N=input('N=');
disp("Ввод вектора X");
for i=1:N
x(i)=input('X=');
end
disp(x);
//Ввод матрицы
N=input('N=');
M=input('M=');
disp(' Ввод матрицы ');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);

N=-->3
 Ввод вектора X
X=-->4
X=-->-3
X=-->46

    4.
  - 3.
    46.

N=-->2
M=-->3
  Ввод матрицы
-->-1
-->6
-->0
-->34
-->3
-->65

  - 1.     6.    0.
    34.    3.    65.
```

так и непосредственно в программе:

```
x=[4 -3  46];
a=[-1, 6, 0;34, 3,65];
```

Вычисление суммы и произведения элементов массива (матрицы)

Рассмотрим алгоритм нахождения суммы, который заключается в следующем: вначале сумма равна 0 (s=0), затем к s добавляем первый элемент массива и результат записываем опять в переменную s, далее к переменной s добавляем второй элемент массива и результат записываем в s и далее аналогично добавляем к s остальные элементы массива. При нахождении суммы элементов матрицы последовательно суммируем элементы всех строк.

Алгоритм нахождения произведения следующий: на первом начальное значение произведения равно 1 (p=1), затем последовательно умножаем p на очередной элемент, и результат записываем в p и т.д.

Пример. Программа вычисления суммы элементов массива

```
//Записываем в переменную s число 0.
s=0;
//Перебираем все элементы массива
for i=1:length(x)
//накопление суммы
s=s+x(i);
end
```

Пример. Программа вычисления произведения элементов массива

```
p=1;
for i=1:length(x)
p=p*x(i);
end
```

Пример. Программа вычисления суммы элементов матрицы

```
s=0;
//Вычисляем количество строк n и столбцов m  матрицы a.
[n,m]=size(a);
for i=1:n
for j=1:m
s=s+a(i,j);
end
end
disp(s);
```

Пример. Программа вычисления произведения элементов матрицы

```
//Начальное значение произведение (p) равно 1.
p=1;
//Вычисляем количество строк N и столбцов M матрицы a.
[N,M]=size(a);
//Перебираем все строки матрицы.
for i=1:N
//Перебираем все столбцы матрицы.
```

```
for j=1:M
Умножаем значение p на текущий элемент матрицы.
p=p*a(i,j);
end
end
```

Поиск максимального (минимального) элемента массива (матрицы)

Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Пусть в переменной с именем *Max* хранится значение максимального элемента массива, а в переменной с именем *Nmax* - его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную *Max*, а в *Nmax* - его номер (1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную *Max*, а в переменную *Nmax* - текущее значение индекса *i*.

Пример. Фрагмент программы поиска максимума:

```
//Записываем в Max значение первого элемента массива.
Max=x(1);
//Записываем в Nmax  номер максимального элемента
//массива, сейчас это число 1.
Nmax=1;
//Перебираем все элементы массива, начиная со второго.
for i=2:N
//Если текущий элемент массива больше Max,
if x(i)>Max
//то текущий элемент массива объявляем максимальным,
Max=x(i);
//а его номер равен i.
Nmax=i;
end;
end;
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в операторе *if* знак поменяется с $>$ на $<$.

Программа поиска минимального элемента матрицы и его индексов: *Nmin* - номер строки , *min* - номер столбца минимального элемента.

Обратите внимание, что при поиске минимального (максимального) элемента матрицы циклы по *i* и *j* начинаются с 1. Если написать с двух, то при обработке элементов будет пропущена первая строка или первый столбец при сравнении *a(i , j)* с min.

Пример. Программа поиска минимального элемента матрицы и его индексов

```
//Записываем в Min a(1,1), в Nmin и Lmin число 1.
Min=a(1,1); Nmin=1; Lmin=1;
for i=1:N
for j=1:M
//Если текущий элемент матрицы меньше Min,
if a(i,j)<Min
//то текущий элемент массива объявляем минимальным,
Min=a(i,j);
//а его индексы равны i и j.
Nmin=i;
Lmin=j;
```

```
end;
end;
end;
```

Пример: вычислить количество отрицательных элементов вектора x=(-2, 5, -7, 9, -15).

Решение.

```
x=[-2, 5, -7, 9, -15];
n=length(x);
 k=0;
for i=1:n
    if x(i)<0 then
k=k+1;
    end
end
t= k
```

Пример: вычислить произведение положительных элементов матрицы

$$a=\begin{pmatrix}1 & 3 & 5 & 7\\ 9 & -2 & 0 & 5\end{pmatrix}$$

Решение.

```
a=[1  3 5 7
     9 -2 0 5];
[n, m]=size(a);
s=1;
for i=1:n
    for j=1:m
        if a(i, j)>0 then
s=s*a(i, j);
end
    end
end
s
```

Структура функций

В пакете есть возможность использовать функции. Функции играют роль подпрограмм. Это позволяет создавать интегрированные в Scilab специализированные программы и использовать библиотеки. Рекомендуемое расширение для файлов подпрограмм **sce**, а для библиотечных функций **sci**.

Существуют два типа sci-файлов: файл-сценарий и файл-функция.

Файл-сценарий (Script-файл) представляет собой последовательность команд без входных и выходных параметров. Он имеет следующую структуру:

// Комментарий

Тело файла с любыми выражениями

Пример:

```
//Plot with color red
// Строит график синусоиды линией красного цвета
// в интервале [0, pi]
x=0: .1: %pi; plot(x, sin(x), 'r')
```

Файл-функция имеет следующую структуру:

```
function[y1,...,yn]=fun(x1,...,xm)
  тело функции
endfunction
```

Здесь
fun - имя функции,
xi - входные аргументы функции (их m штук),
yi - выходные аргументы функции (их n штук).
Пример. Вычисление факториала.

```
function [x]=fact(k)
k=int(k)
if k<1 then k=1, end
x=1;
for j=1:k,x=x*j;end
endfunction
```

Наберем этот текст в любом текстовом редакторе и сохраним его в файле с именем fact.sci. Расширение ***.sci** является для Sclab "родным но не обязательным. Затем следует вызвать эти файлы из Scilab с помощью команд **getf(filename)** или **exec(filename,-1)**; Те же операции можно произвести с помощью команд меню **File-getf** или **File-exec**.

До вызова функции желательно проверить, не была ли уже загружена такая функция ранее. Для этого:

exists('fact')

Результат:

ans =

0.

После загрузки файла

–>exec('C:\fact');

набираем

–>x=fact(5)

x =

120

Пример:

```
function z=fun(x, y)
// Определение функции
z=x.^2 + y.^2;
endfunction
```

Сохраним эту функцию под именем fun.sci. После загрузки функции в Scilab мы можем обратиться к ней:

```
-->fun(1,2)
 ans  =
    5.
```

Пример. Даны два массива X и Y. Вывести в командное окно имя массива, содержащего наибольшее число элементов, кратных числу три.

Решение. Составим программу и сохраним ее под именем t.sci в каталоге C:\DOCUME~1\user\.

```
function g=t(v)
s=0; n=length(v);
for i=1:n
    if  modulo(v(i), 3)==0   then
        s=s+1;
    end
end
g=s;
endfunction
```

Далее сохраним в корневом каталоге C:\ под именем tot, а затем выполним программу

```
x=[1 2 3 5 7 6 12]; y=[5 7 24 15]; d1=t(x), d2=t(y)
if d1>d2 then
    disp('x')
elseif  d1==d2 then
    disp('в обоих массивах это число одинаково')
else
    disp('y')
end
```

Получим ответ:

```
d1  =
    3.
 d2  =
    2.
 x
```

Запуск файла на выполнение осуществляется так:

```
-->scipad('C:\DOCUME~1\user\t.sci');
-->scipad('C:\tot');
-->exec('C:\tot');
```

Файл-функция был сохранен в 'C:\DOCUME~1\user\t.sci', файл-сценарий – в 'C:\tot'

Это эквивалентно выполнению следующих операций: запускаем Scilab.

Далее: открываем файл-функцию File/Open... Далее Execute/Load into Scilab; Открываем файл-сценарий (выполняемый файл) File/Open...Далее File/Exec ...

Открываются файлы в окне Scilab.

Если выходных параметров несколько, то их надо указать в квадратных скобках после слова **function**.

```
 function [var1, var2, ...] = fname(список_параметров)
// Основной комментарий
// Дополнительный комментарий
Тело файла с любыми выражениями
var1=выражение
var2=выражение
...
endfunction
```

Пример. Задан вектор $\vec{x} = (x_1, x_2, ..., x_n)$ Вычислить $\vec{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ и $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \vec{x})^2}$

Решение. Набираем и сохраняем в редакторе под именем **statv.sci** файл

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

Далее в командном окне набираем

```
--> v=[1, 2, 3, 4, 5];
--> [a, m]=statv(v)
```

В строках вывода получаем:

```
m  =
    1.4142136
 a  =
    3.
```

Краткий комментарий к программе.

Во второй строке файла **statv.sci** вычисляется количество элементов в массиве **x**; **sum(x)** – суммируются все элементы массива **x**; **mean** – вычисляется $\vec{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$; **stdev** – вычисляется $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \vec{x})^2}$ В итоге получаем среднее значение $a = \vec{v} = \frac{1}{n}\sum_{i=1}^{n} v_i = 3$, $m = s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \vec{v})^2}$.

Для создания временной задержки выполнения программы служит команда **xpause**, которая приостанавливает текущий процесс на число микросекунд, указанных в скобках

- **xpause(microsecs)**

Пример:

```
xbasc();
xset("color",12);
xstring(1,1,"Blue");
xpause(5.e6); // временная задержка на 5 секунд
xbasc();
xset("color",5);
xstring(0,1,"Red");
```

В результате получили в графическом окне вначале надпись "Blue которая затем, и появляется новая сдвинутая по горизонтали надпись "Red".

Некоторые полезные функции Scilab для вычисления целой части числа:

```
int(1.9999)=1=int(1.1)   int(-1.9999)=-1=int(-0.00001)
floor(1.5)=1=floor(1.9)=floor(1.1)  floor(-1.1)=-2=floor(-1.8)
ceil(1.1)=ceil(1.9)=2   ceil(-1.9)=-1=ceil(-1.2)
fix(-1.2)=-1=fix(-1.7)  fix(1.7)=1=fix(1.2)
```

Тема 5. Примеры решения задач на массивы

Некоторые полезные сведения для решения задач

Среднее арифметическое $\frac{1}{n}\sum x_i$

Среднее геометрическое $\sqrt[n]{x_1 x_2 ... x_n}$

Целое число – число, у которого целая часть числа совпадает с самим числом. Например, целая часть числа 7,3 равна 7 и не совпадает с самим числом (7,3).

Число, кратное 7 – число, делящееся на 7 без остатка.

Для нахождения целой части числа в scilab можно использовать функцию int.

Для нахождения остатка от деления a на b используется функция modulo(a,b).

Пример 1. Найти среднее арифметическое положительных элементов массива в= [2 , 8.4, -3 ,7, -5.34, 8].

```
clc
s=0;m=0;
 b= [2 , 8.4,  -3,7,  -5.34,  8];
n=length(b);
for i=1:n
if b(i)>0 then
s=s+b(i); m=m+1;
end
end
t=s/m

t  =

    6.35
```

Пример 2. Найти среднее геометрическое целых положительных элементов вектора p=[4.4, 5, -3, 12, 6, 1.6].

```
clc
w=1;m=0;
 p=[4.4, 5, -3, 12, 6, 1.6];
n=length(p);
for i=1:n
if p(i)>0 & int(p(i))==p(i) then
w=w*p(i); m=m+1;
end
end
t=w^(1/m)
t  =
    7.1137866
```

Пример 3. Определить количество элементов массива t=[23, 21, 6,5,3,33], кратных 3.

```
clc
m=0;
t=[23, 21, 6,5,3,33 ];
n=length(t);
for i=1:n
if modulo(t(i),3)==0 then
```

```
m=m+1;
end
end
disp(m)
```

4.

Пример 4. Даны два одномерных массива X и Y. Вывести в командное окно имя массива, содержащего наибольшее число элементов, кратных числу три.

Решение задачи разобьем на несколько этапов.

1. Забудем на время, что массивов – два и решим такую задачу: Дан одномерный массив v. Определить, сколько в нем элементов, кратных числу 3.
2. Оформим эту программу в виде функции, аргументом которой является массив v, а результатом – переменная g. Пусть имя этой функции t.
3. Зададим значения векторов X и Y и дважды обратимся к этой функции. Пусть результат первого обращения (число элементов, кратных 3, в первом массиве) – это *d1*, а результат второго – *d2*.
4. Сравнивая *d1* и *d2*, печатаем либо букву x (имя первого массива), либо букву y, либо сообщение '*в обоих массивах это число одинаково*' в случае равенства *d1* и *d2*.

1 этап.

```
v=[ ]; //задаем конкретные значения элементов
s=0; n=length(v);
for i=1:n
    if  modulo(v(i), 3)==0   then
        s=s+1;
    end
end
g=s
```

Здесь g – число элементов, кратных 3, в массиве v.

2 этап.

Убираем первую строку и оформляем оставшийся кусок программы в виде функции:

```
function g=t(v)
s=0; n=length(v);
for i=1:n
    if  modulo(v(i), 3)==0   then
        s=s+1;
    end
end
g=s;
endfunction
```

3 этап:

```
x=[1 2 3 5 7 6 12]; y=[5 7 24 15]; d1=t(x), d2=t(y)
```

4 этап:

```
if d1>d2 then
    disp('x')
elseif  d1==d2 then
    disp('в обоих массивах это число одинаково')
else
    disp('y')
end
```

Окончательный вид программы:

```
function g=t(v)
s=0; n=length(v);
for i=1:n
    if  modulo(v(i), 3)==0   then
        s=s+1;
    end
end
g=s;
endfunction
x=[1 2 3 5 7 6 12]; y=[5 7 24 15]; d1=t(x), d2=t(y)
if d1>d2 then
    disp('x')
elseif  d1==d2 then
    disp('в обоих массивах это число одинаково')
else
    disp('y')
end
Получим ответ:
d1  =
    3.
 d2  =
    2.
 x
```

Для того, чтобы не присваивать заранее значения элементам одномерного массива, можно воспользоваться функцией input:

```
clc
N=input('N=');
disp("Ввод вектора X");
for i=1:N
x(i)=input('X=');
end
disp(x);
```

Пример 5. Ввести элементы вектора x=[3 -6 8].

После сохранения и запуска файла в scilab 5.4.1 в командном окне увидим N= ... Вводим 3(количество элементов одномерного массива) и нажимаем Enter. Далее появляется текст Ввод вектора X и в следующей строке надпись X= ... вводим первый элемент вектора (3) и нажимаем клавишу ввода. Снова появляется надпись X=... вводим -6 Enter X=... 8 Enter. После этого оператором disp распечатываются значения вектора X в виде столбца из трех элементов.

```
N=3
 Ввод вектора X
X=3
X=-6
X=8
     3.
  - 6.
    8.
```

Аналогично можно ввести матрицу.

Пример 6. Ввести матрицу a=[-1, 6, 0;34, 3,65];

```
//Ввод матрицы
N=input('N=');
M=input('M=');
disp(' Ввод матрицы ');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);

N=2
M=3

  Ввод матрицы
-1
6
0
34
3
65

  - 1.     6.    0.
    34.    3.    65.
```

Здесь N – количество строк матрицы, M – количество столбцов, далее после появления текста Ввод матрицы вводятся сами элементы матрицы построчно. Для того ввода матрицы по столбцам программа выглядит так:

```
//Ввод матрицы
N=input('N=');
M=input('M=');
disp(' Ввод матрицы ');
for j=1:M
for i=1:N
a(i,j)=input('');
end
end
disp(a);
```

```
N=2
M=3
   Ввод матрицы
-1
34
6
3
0
65

  - 1.     6.    0.
    34.    3.    65.
```

Пример 7. Программа вычисления суммы элементов матрицы

```
s=0;
//Вычисляем количество строк n и столбцов m  матрицы a.
[n,m]=size(a);
for i=1:n
for j=1:m
s=s+a(i,j);
end
end
disp(s);
Программа вычисления произведения элементов матрицы
//Начальное значение произведение (p) равно 1.
p=1;
//Вычисляем количество строк N и столбцов M матрицы a.
[N,M]=size(a);
//Перебираем все строки матрицы.
for i=1:N
//Перебираем все столбцы матрицы.
for j=1:M
Умножаем значение p на текущий элемент матрицы.
p=p*a(i,j);
end
end
```

Пример 8. Поиск максимального (минимального) элемента массива (матрицы)

Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Пусть в переменной с именем *Max* хранится значение максимального элемента массива, а в переменной с именем *Nmax* - его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную *Max*, а в *Nmax* - его номер (1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную *Max*, а в переменную *Nmax* - текущее значение индекса i. Фрагмент программы поиска максимума:

```
Реализация алгоритма поиска максимума
//Записываем в Max значение первого элемента массива.
Max=a(1);
//Записываем в Nmax  номер максимального элемента
//массива, сейчас это число 1.
```

```
Nmax=1;
//Перебираем все элементы массива, начиная со второго.
for i=2:N
//Если текущий элемент массива больше Max,
if x(i)>Max
//то текущий элемент массива объявляем максимальным,
Max=x(i);
//а его номер равен i.
Nmax=i;
end;
end;
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в операторе *if* знак поменяется с $>$ на $<$.

Программа поиска минимального элемента матрицы и его индексов: *Nmin* - номер строки , *min* - номер столбца минимального элемента.

Обратите внимание, что при поиске минимального (максимального) элемента матрицы циклы по i и j начинаются с 1. Если написать с двух, то при обработке элементов будет пропущена первая строка или первый столбец при сравнении *a(i , j)* с min.

```
Программа поиска минимального элемента матрицы и его индексов
//Записываем в Min a(1,1), в Nmin и Lmin число 1.
Min=a(1,1); Nmin=1; Lmin=1;
for i=1:N
for j=1:M
//Если текущий элемент матрицы меньше Min,
if a(i,j)<Min
//то текущий элемент массива объявляем минимальным,
Min=a(i,j);
//а его индексы равны i и j.
Nmin=i;
Lmin=j;
end;
end;
end;
```

Тема 6. Численное интегрирование

При вычислении определенных интегралов от функций, заданных в виде таблицы или в явном виде ($\int_a^b f(x)dx$), одним из численных методов используют функции intsplin, inttrap, integrate, intg.

Способ 1.

С помощью команды **intsplin**. Это интегрирование экспериментальных данных с помощью сплайн-интерполяции. Известно значение интегрируемой функции в дискретных точках (узлах).

Пример: вычислить интеграл от таблично заданной функции

x		.4	.8	.2	.6		.4	.8	.2	.6	
y=	.6487	.3771	.1972	.0833	.0202		.0202	.0833	.1972	.3771	.6487

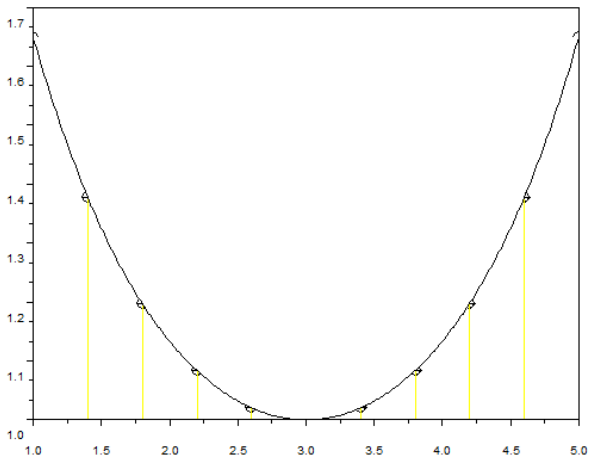
```
-->x=1:.4:5;
-->y=exp((x-3).^2/8)//значения y в таблице получены табулированием этой функции
-->v=intsplin(x,y)
Получаем:
 v  =
    4.7799684
```

Или:

```
-->x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5];
-->y=[1.6487 1.3771 1.1972 1.0833 1.0202 1 1.0202 1.0833 1.1972 1.3771 1.6487];
-->v=intsplin(x,y)
 v  =
    4.7799328
```

```
clc
x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5];
y=[1.6487 1.3771 1.1972 1.0833 1.0202 1 1.0202 1.0833 1.1972 1.3771 1.6487];
plot2d(x,y,-4);//График экспериментальных данных
koeff=splin(x,y);

 plot2d(x,y,-3); //Нанесение точек на график
//Построение кубического сплайна
t=min(x):0.01:max(x);
ptd=interp(t,x,y,koeff);
plot2d(t,ptd);
```



Способ 2.

С помощью команды **inttrap**. Интегрирование экспериментальных данных с помощью трапецеидальной интерполяции (метод трапеций).

При вычислении интеграла между соседними узлами функция интерполируется линейно. Этот метод вычислений называется методом трапеций.

Вычислим интеграл от той же самой функции

```
-->x=1:.4:5;
-->y=exp((x-3).^2/8)
-->v=inttrap(x,y)
```

Получаем:

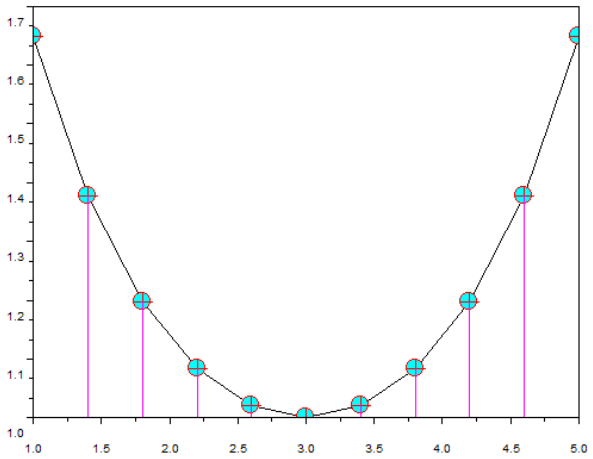
```
 v  =
    4.8017553
```

Или

```
x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5];
y=[1.6487 1.3771 1.1972 1.0833 1.0202 1 1.0202 1.0833 1.1972 1.3771 1.6487];
v=inttrap(x,y)
```

Получаем:

```
 v  =
    4.80172
```



В первых двух способах вычисляется площадь заштрихованных фигур.

Способ 3. С помощью команды **integrate**. Это интегрирование по квадратуре. Может задаваться требуемая точность вычислений.

Синтаксис **[x]=integrate(expr,v,x0,x1 [,ea [,er]])**

Параметры

expr : подынтегральная функция;

v : переменная интегрирования;

x0, x1 : пределы интегрирования;

ea, er : действительные числа. Первое из них – **ea** - предельная абсолютная ошибка. По умолчанию принимает значение **0**. Второе - **er** -, - предельная относительная ошибка. По умолчанию принимает значение **1.d-8**.

Пример: вычислить $\int\limits_1^5 e^{\frac{(x-3)^2}{8}} dx$

```
-->integrate('exp((x-3)^2/8)','x',1,5)
 ans  =
    4.7798306
```

Можно это сделать и так. Набираем и сохраняем в окне редактора под именем **a.sci** файл

```
function g=a(x)
g=exp((x-3).^2/8);
endfunction
```

Загружаем этот файл в среду Scilab (Load into Scilab).

Далее в строке ввода набираем:

```
-->integrate('a','x',1,5)
```

Получаем:

```
 ans  =
    4.7798306
```

Способ 4.

С помощью команды **intg**. Интегрируемая функция задана извне: либо в виде набора дискретных точек, либо вычисляется с помощью внешней подпрограммы. Может задаваться требуемая точность вычислений.

Синтаксис

[v,err]=intg(a,b,f [,ea [,er])

Параметры

a, b : действительные числа;

f : внешняя функция или список строк;

ea, er : - параметры, описанные выше;

err : оценка абсолютной ошибки результата.

Вычисляется определенный интеграл от **a** до **b** от **f(t)dt**.

Вычислим тот же интеграл: $\int\limits_1^5 e^{\frac{(x-3)^2}{8}}dx$

```
function g=a(x)
g=exp((x-3).^2/8);
endfunction
intg(1,5,a)
```

Вызвав этот файл на выполнение (Execute/Load into Scilab), получим ответ:

```
ans  =
    4.7798306
```

Тема 7. Примеры математического моделирования реальных ситуаций и процессов в программной среде Scilab

Использование программного пакета Scilab для проведения математических расчетов

Рассматривается задача: автомобиль находится перед гаражом на некотором расстоянии от него (рис. 1). Требуется поставить автомобиль в гараж и сделать это, по возможности, наилучшим образом.

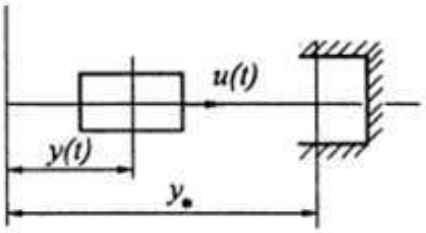


Рис. 1. Автомобиль перед гаражом

При решении будем руководствоваться алгоритмом системного анализа [2], представленного на рис.2.

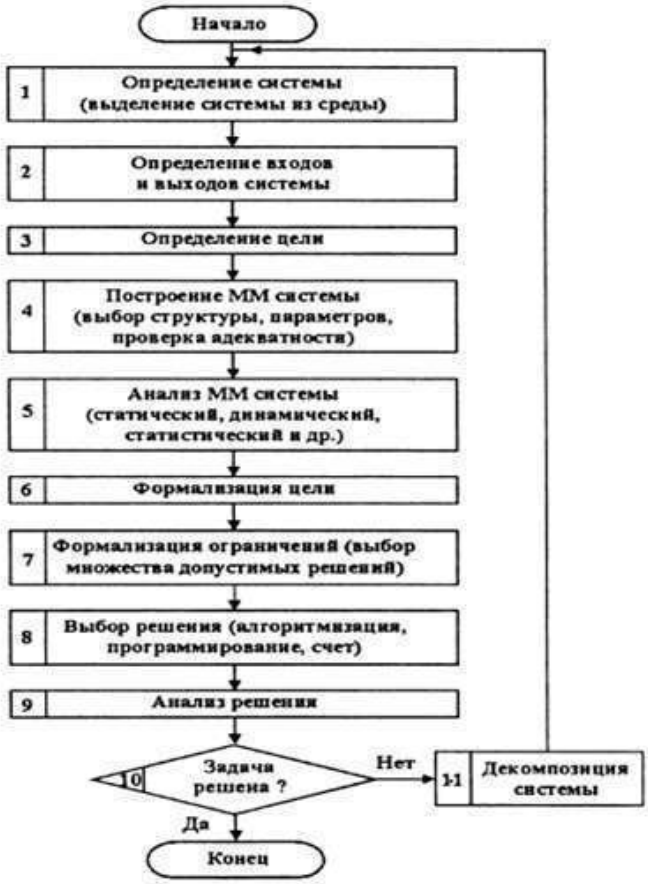


Рис. 2. Алгоритм проведения системного анализа

Для рассматриваемой задачи этапы решения примут вид:

Этап 1. Система «Автомобиль-гараж».

Этап 2. Вход – сила тяги двигателя. Выход – пройденный путь.

Этап 3. Цель – автомобиль должен проехать заданный путь и затормозить.

Этап 4. Построение математической модели исследуемой системы начинается с обозначения всех величин (переменных и постоянных), существенных для задачи. Введем следующие обозначения:

$u(t)$ - сила тяги в момент времени t (входная переменная);

$y(t)$ – путь, пройденный к моменту t (выходная переменная);

y_* - расстояние от автомобиля до гаража (параметр).

Выпишем уравнения и соотношения, существующие между введенными величинами. Воспользуемся уравнением динамики (второй закон Ньютона)

$$m\frac{d^2y(t)}{dt^2} = u(t) \tag{1}$$

где m – масса автомобиля, а также учтем начальные условия

$$y(0) = 0, y(0) = 0. \tag{2}$$

Этап 5. Синтезируемая модель адекватна, если можно пренебречь размерами автомобиля, ограничением на его мощность, силами трения и сопротивления и другими более второстепенными факторами.

Этап 6. Простейший вариант формализации цели задается равенством

$$y(t_*) = y_*, \tag{3}$$

где t_* - момент остановки, который оказывается неудовлетворительным, поскольку в (3) не формализовано само требование остановки $\dot{y}(t_*) = 0$, и, значит, не ясно, как система будет вести себя при $t > t_*$. Правильнее задать цель соотношением

$$y(t_*) = y_*, t > t_*, \tag{4}$$

Из которого следует, в частности, что $\dot{y}(t) = 0$ при $t > t_*$. Заметим, что однозначного решения поставленная задача не имеет, т.к. существует бесконечно много способов для достижения цели (4). Выберем то решение, которое позволит быстрее достичь цел. Формально новую цель можно записать так:

$$\min\{t_* : y(t) = y_* \text{ при } t \geq t_*\}. \tag{5}$$

Введем дополнительно ограничения на силу тяги:

$$-a \leq u(t) \geq b. \tag{6}$$

Этап 8. Воспользуемся геометрической интерпретацией задачи, т.к. естественная интерпретация, в координатах «время – пройденный путь» не позволяет в удобной форме представить ограничения на допустимые траектории движения автомобиля. Введем новую переменную $v(t) = y(t)$ (скорость). Уравнение (1) и условия (2) примут вид

$$-a \leq u(t) \geq b. \tag{7}$$

цель (5) :

$$\min\{t_* : \int_0^t v(s)ds = y_* \text{ при } t > t_*\}, \tag{8}$$

ограничения (6) превратятся в ограничения на скорость изменения новой переменной:

$$-\frac{a}{m} \leq \dot{v}(t) \geq \frac{b}{m}. \tag{9}$$

Геометрическая интерпретация динамики системы (7) с учетом условий - (8)–(9) в плоскости v, t изображена на (Рис. 3). Получили, что для решения задачи нужно найти кривую $v(t)(t \geq 0)$ с заданной площадью фигуры F под ней и наименьшей возможной координатой правого конца t_*, лежащую в треугольнике OMK с заданными углами наклона ϕ_1, ϕ_2 боковых сторон (в соответствии с (9) $\tan\phi_1 = \frac{b}{m}$ при $\tan\phi_2 = \frac{a}{m}$).

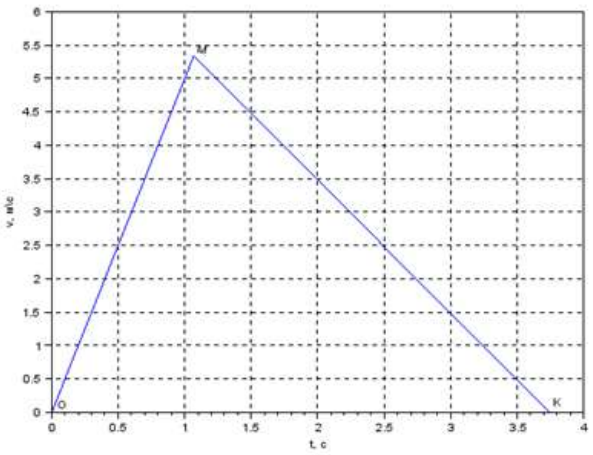


Рис. 3. Геометрическая интерпретация системы (7)

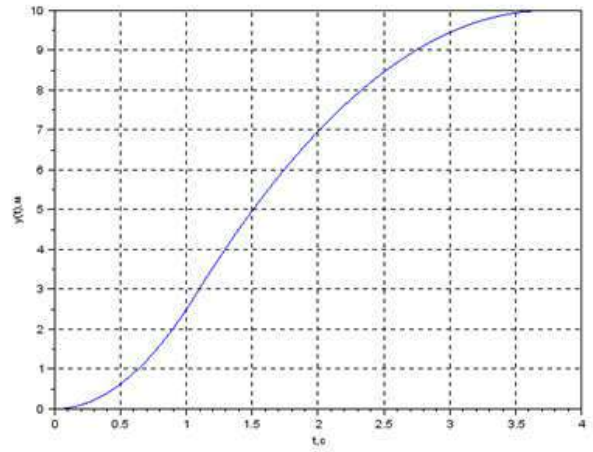


Рис. 4. «Время-пройденный путь»

Геометрическое решение очевидно: фигура F должна заполнять весь треугольник. Это значит, что автомобиль должен двигаться с максимальным ускорением до некоторого момента t_1, затем включить максимальное торможение и в момент t_* выключить двигатель. Формулы для определения момента переключения имеют вид

$$t_1 = \sqrt{\frac{2may_*}{b(a+b)}},\ t_* = \sqrt{\frac{2m(a+b)y_*}{ab}} \tag{10}$$

Рассмотренная геометрическая модель позволяет решать и более сложные задачи. Например, если по соблюдению безопасности нужно учесть ограничение на максимальную скорость: $|\dot{y}(t)| \leq v$, то легко усмотреть решения из (Рис. 3). График оптимальной траектории движения автомобиля представляет собой трапецию.

Программная реализация моделируемой системы «Автомобиль-гараж» в пакете Scilab выглядит следующим образом:

```
//Ввод исходных данных
a=2000; //Левое ограничение силы тяги
b=5000; //Правое ограничение силы тяги
m=1000; //Масса автомобиля
y_T=10; // Расстояние от автомобиля до гаража
```

```
u=1;
t1 = sqrt((2*m*a*y_T)/(b*(a+b))); // уравнения 10
T = sqrt((2*m*(a+b)*y_T)/(a*b)); // уравнения 10
x0=zeros(2,1); // Задание нулевой матрицы
t0=0;
t=[t0:0.001:T];
deff('[dx]=fp1_2(t,x)',.. // Подпрограмма функции
'if t<t1 then u=b;..
\indent    else u=-a; end ..
\indent     dx(1) = u/m;..
\indent     dx(2)=x(1);'); // задания функции для решения ДУ
x = ode(x0,t0,t,fp1_2); // решения ДУ
letter1 = 'O' //Вершина O
letter2 = 'M' //Вершина M
letter3 = 'K' //Вершина K
plot(t,x(1,:),'t','v') // Построение графика 1
xtitle('','t, c','v, м\c'); // Обозначения осей на графике 1
xstring(0.01,0.02,letter1,10,15); // Обозначение вершины O на графике
xstring(1.065,5.345,letter2,10,15); // Обозначение вершины M на графике
xstring(3.75,0.05,letter3,1,3); // Обозначение вершины K на графике
xgrid() //определяем сетку на графике и ее стиль(в нашем случае стиль стандартный)
xset("window",1) // Открытие нового окна
plot(t,x(2,:)) // Построение графика 2
xtitle('','t,c','y(t),м'); // Обозначение осей на графике 2
xgrid() //определяем сетку на графике и ее стиль(в нашем случае стиль стандартный)
```

Верификация синтезированной модели системы «автомобиль-гараж» представлена на (Рис. 3) и (Рис. 4).

Результаты данного раздела получены студентом второго курса Головиным Дмитрием направления подготовки Прикладная математика и информатика под руководством автора настоящего пособия Игониной Е.В. и представлены в работе[8].

Использование программной среды Scilab для нахождения спектра сигнала по дискретной выборке на основе преобразования Фурье

Рассмотрим использование вычислительных и графических возможностей программной среды Scilab для проведения спектрального анализа квазипериодических колебаний на основе дискретного преобразования Фурье.

Колебания – один из самых распространенных процессов в природе и технике. Несмотря на разнообразие видов: механические, электромагнитные, химические, термодинамические и др. колебания имеют между собой много общего. До начала XX века основная модель, описывающая колебательный процесс, представлялась в виде линейных дифференциальных уравнений:

$$\ddot{y}(x) = \omega^2 y(t) = 0, \quad 0 \le t < \infty \tag{11}$$

решением, которых являются гармонические колебания

$$y(t) = A_0 \sin(\omega t) + A_1 \cos(\omega t) \tag{12}$$

с круговой частотой ω и периодом $T = \frac{2\pi}{\omega}$, амплитуда $A = \sqrt{A_0^2 + A_1^2}$ зависит от начальных условий: $A_1 = y(0), A_0 = \frac{\dot{A}(0)}{\omega}$. Частотный спектр функции (2) дискретен и состоит из одной точки $\frac{\omega}{2\pi}$.

В настоящее время формы описания колебательных процессов постоянно развиваются и совершенствуются. Например, для описания колебаний более сложной формы можно соединять модели вида (11) с различными частотами колебаний $\omega_1, ..., \omega_r$. Если данные частоты являются соизмеримыми, то колебания будут с периодом равным $\frac{2\pi}{\omega_0}$. Если же эти частоты несоизмеримы, то колебания будут непериодическими, относящиеся к классу квазипериодических.

Как известно [6], любой периодический сигнал можно представить в виде комбинации гармонических колебаний. Совокупность амплитуд и частот гармоник, составляющих сигнал, называют спектром сигнала. Спектральный анализ основывается на преобразовании Фурье и заключается в том, что имеющийся сигнал разлагается на частотные или спектральные составляющие. В дальнейшем оцениваются спектральные характеристики сигнала – фаза, амплитуда, спектральная плотность мощности и другие. Наиболее часто рассматриваются энергетические характеристики: мощность и энергия. Мгновенная мощность $\rho(t)$ сигнала $y(t)$ определяется как квадрат его мгновенного значения: $\rho(t) \triangleq y(t)^2$ Энергия P сигнала на интервале $[t_1, t_2]$ находится как интеграл от мгновенной мощности $P \triangleq \int\limits_{t_1}^{t_2} y(t)^2 dt$. Отношение $\frac{P}{t_2 - t_2} = \frac{1}{t_2 - t_1} \int\limits_{t_1}^{t_2} y(t)^2 dt$ выражает среднюю (на интервале $[t_1, t_2]$) мощность сигнала (обозначим ее через $y(t)^2$)

Рассмотрим применение преобразования Фурье [6] для исследования квазипериодических колебаний на основе указанных характеристик. Отметим, что для периодических процессов $y(t)$ с периодом T, ряд Фурье запишется в виде:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos k\frac{2\pi}{T} t + b_k \sin k\frac{2\pi}{T} t) \tag{13}$$

где $a_0 = \frac{2}{T}\int_0^T y(t)dt, \quad a_k = \frac{2}{T}\int_0^T y(t)\cos(k\frac{2\pi}{T}t)dt, b_k = \frac{2}{T}\int_0^T y(t)\sin(k\frac{2\pi}{T}t)dt,$
$(k = 1, 2, ...)$.

Совокупность величин $s_0 \triangleq \frac{|a_0|}{2}, s_k \triangleq \sqrt{a_k^2 + b_k^2} (k = 1, 2, ...)$ называется амплитудным частотным спектром периодической функции $y(t)$ Значения s_k представляют собой амплитуды гармоник с частотой $\omega_k = k\Omega (\Omega = \frac{2\pi}{T})$ в разложении процесса в ряд (13). Указанные значения зависят от номера гармоники k и графически представимы в виде отрезков высотой s_k, проведенных в точках ω_k оси частот (поэтому спектр периодической функции называют линейчатым, или дискретным). Полученный спектр является носителем информации о частотных свойствах сигнала: если сигнал имеет выраженные колебания на некоторых частотах, то его спектр на этих частотах содержит пики.

Обобщением ряда Фурье на непериодические процессы является интеграл Фурье, при котором используется представление

$$y(t) = \frac{1}{\pi} \int_0^{\infty} (V(\omega)\sin(\omega t) + U(\omega)\cos(\omega t))dt \tag{14}$$

где

$U(\omega) = \int\limits_{-\infty}^{\infty} y(t)\cos(\omega t)dt, \quad V(\omega) = \int\limits_{-\infty}^{\infty} y(t)\sin(\omega t)dt).$

Аналогично вводится частотный спектр процесса $y(t)$ в виде функции от непрерывного аргумента ω: $S(\omega) \triangleq \sqrt{U(\omega)^2 + V(\omega)^2}$.

Поскольку при цифровом моделировании исходной является дискретная реализация процесса и нахождение интеграла выполняется конечным суммированием, то при числовом гармоническом анализе вместо непрерывного преобразования (14) выполняется дискретное преобразование Фурье (ДПФ)[7]. Для этого исследуемый процесс $y(t)$ (длительностью T) заменяется выборочной дискретной функцией (т.е. последовательностью) $y[k] \triangleq y(t_k)$, где

$t_k = kT_0(k = 1, 2, ..., N), N$ - заданное число точек, $T_0 = \frac{T}{N}$ - шаг дискретности (квантования). Далее вычисляется функция

$$Y(k) = \sum_{n=1}^{N} y[n] \exp(-j2\pi(k-1)\frac{n-1}{N}), \quad 1 \leq k \geq N \tag{15}$$

(«изображение по Фурье»), принимающая комплексные значения.

В математическом пакете SCILAB ДПФ выполняется процедурой **fft**. Для ускорения вычислений число точек определяется формулой $N = 2^v$, где v - некоторое натуральное число. В этом случае программой реализуется так называемое «быстрое преобразование Фурье». Обратный переход от изображения к исходной функции выполняется по формуле

$$y[n] = \frac{1}{N} \sum_{n=1}^{N} Y(k) \exp(j2\pi(k-1)\frac{n-1}{N}), \quad 1 \leq n \geq N. \tag{16}$$

При выборе параметров вычисления спектра следует учитывать, что диапазон существенных частот исследуемого процесса не должен выходить за частоту Найквиста $\omega_N \triangleq \frac{\pi}{T_0}$. Несоблюдение этого условия может привести к значительным ошибкам при определении характеристик процесса.

Поскольку рассматриваемый процесс $y(k)$ в общем случае не является периодическим с частотой $\Omega = \frac{2\pi}{T}$, вычисление его спектра с помощью рассматриваемой процедуры является приближенным. Как видно из формулы (15), соседние точки по частоте отстоят на величину $\delta\omega \triangleq \frac{2\pi}{T_0 N}$. Учитывая, что $N = \frac{T}{T_0}$, получим $\delta\omega = \frac{2\pi}{T}$. Таким образом, длительность исследуемой реализации должна быть достаточно большой для получения спектра с заданным шагом дискретности по частоте ($T >> \frac{1}{\delta\omega}$).

Перейдем к задаче нахождения спектра сигнала по дискретной выборке на основе преобразования Фурье с использованием математического пакета Scilab. Воспользуемся вышеприведенными рассуждениями, а именно вернемся к рассмотренному выше квазипериодическому процессу, представляющему собой сумму гармоник с несоизмеримыми частотами:

$$y(t) = \sin(t) + 0.75\sin(5\pi^{-1}t). \tag{17}$$

Зададим интервал дискретности $T_0 = 0.05$ и число точек отсчета $N = 2^{10}$. Это соответствует длине реализации $T = 51.2$.

Код программы, реализующей поставленную задачу, выглядит следующим образом:

```
dt = 0.05;  //Интервал дискретности
N=2^10;  //Точки отсчета
t=0:dt:(N-1)*dt;  //Ось t
y=sin(t)+0.75*sin(5/%pi*t);  //Квазипериодический процесс
Y=fft(y,-1);  //Дискретное преобразование Фурье
plot2d(t,y)  //Построение графика y(t)
xtitle('y(t)')  //Обозначение на графике
PY=abs(Y).^2/N;  //Обратный переход от изображения к функции
wn2=2*%pi/dt;  //Частота
dw=wn2/N;  //Вычисление интервала между соседними точками
w=0:dw:5;  //Ось омега
lw=length(w);  //Вычисление длины вектора w
xset('window',1)  //Новое окно для графика
plot2d(w,PY(1:lw))  //Построение графика S(w)
xtitle('S(w)')  //Обозначение на графике
```

На рис. 5 представлен график квазипериодического сигнала, описываемого функцией (17), а на рис. 6 дана графическая визуализация результата вычисления спектра сигнала по дискретной выборке на основе преобразования Фурье.

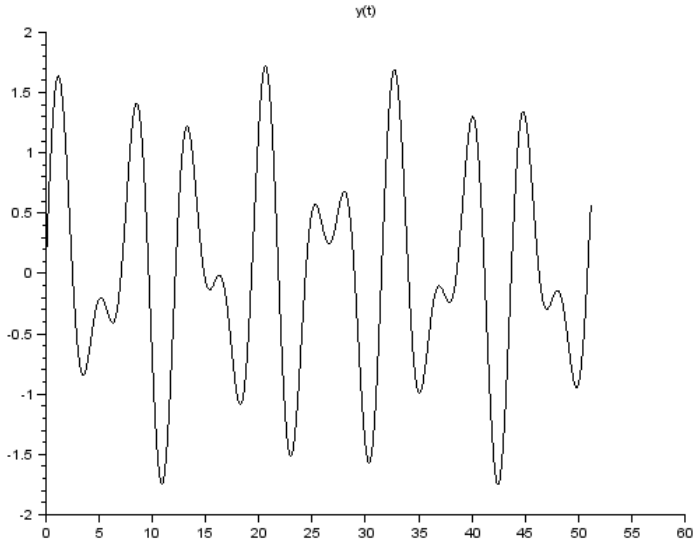


Рис. 5. Квазипериодический сигнал

Задача оценивания спектра по выборке имеет важную особенность: оценка спектральной плотности, получаемая по ДПФ, является смешенной, т.е. неограниченное уменьшение шага дискретности не приводит к неограниченному убыванию погрешности оценивания. Средним значением оценки является неточное значение спектральной плотности на заданной частоте, а некоторое сглаженное значение, получаемое усреднением функции. Удачный выбор такой функции позволяет уменьшить погрешность оценивания при заданном конечном шаге дискретности.

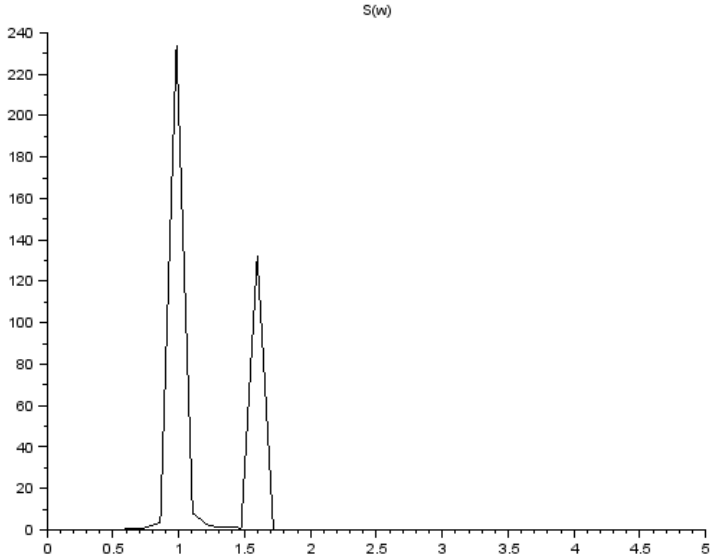


Рис. 6. Спектр квазипериодического сигнала

Полученные в настоящей работе результаты по изучению функциональных возможностей программного пакета Scilab для проведения спектрального анализа могут быть использованы в радиотехнике, медицине, в различных отраслях промышленности и науки.

Часть 2. Лабораторный практикум

Лабораторная работа №1. Арифметические выражения в Scilab

Цель работы

1. Ознакомиться с работой системы в командном и программном режимах.
2. Освоить запись арифметических выражений.
3. Изучить арифметические операции и математические функции.

Краткие методические указания

1. В командном окне задать значения переменным, затем записать выражение на языке SciLab. Для вывода значения выражения не ставить после него точки с запятой.
2. Добиться правильной записи выражения без синтаксических ошибок. Открыть окно редактора SciNotes, скопировать в него исходные переменные и правильный вариант выражения из командного окна.
3. Сохранить содержимое окна редактора в sce-файле и запустить его на выполнение с отображением команд.

Задание

Вычислить значение арифметического выражения с учетом значений входящих переменных

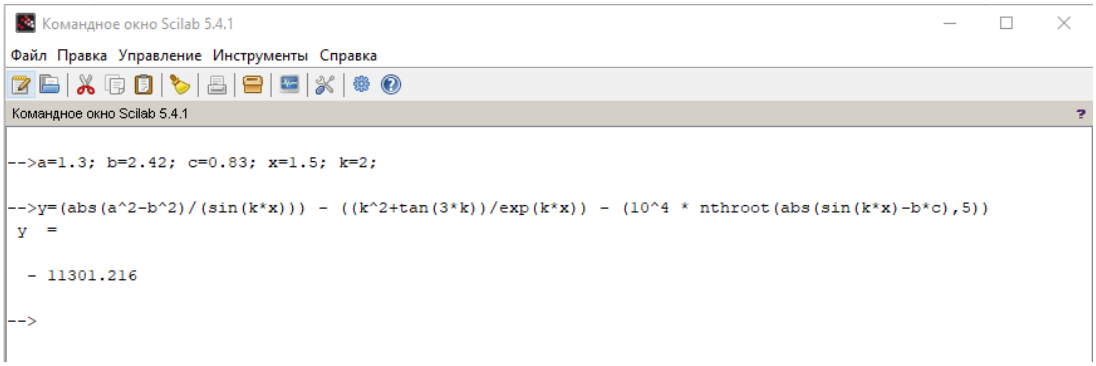
$a = 1,3; b = 2,2; c = 0,83; x = 1,5; k = 2$

$y = \frac{|a^2-b^2|}{\sin kx} - \frac{k^2+\tan 3k}{e^{kx}} - 10^4 \sqrt[5]{|\sin kx - bc|}.$

Код программы

```
//Лабораторная работа №1
a=1.3;b=2.2;c=0.83;x=1.5;k=2;
y=(abs(a^2-b^2)/(sin(k*x))) - ((k^2+tan(3*k))/exp(k*x)) -
(10^4 * nthroot(abs(sin(k*x)-b*c),5))
```

Результат тестирования



Задания для самостоятельной работы

Задание №1:
$x = 0,29;\ a = -2.4;\ k = 3;\ c = 1,52;$
$y = \frac{\sqrt[3]{lnx+a^2}}{0,47x^2} - |0,47x^2 - \frac{10^4}{7}\cos^2 k| - \frac{c}{k}.$
Контрольный счет: -1363.4871
Задание №2:
$a = -2,5;\ b = 1,35;\ x = 2,75;\ i = 3;\ c = -0,72;$
$y = \frac{1,5(a-b)^2}{|a-b|c} + i/5 + 10^3\sqrt{|a-b|} - \frac{2,5(a+x^2)\sin 7}{ix^2+a^2bc}$
Контрольный счет: 1954.2203
Задание №3:
$a = 3,5;\ i = 2;\ b = -0,7;\ x = 0,8;$
$y = 10^4 \sin^2 i - \frac{0,32x^3+4x+b}{\cos ia}\sqrt[4]{0,32x^3-b} + |b|.$
Контрольный счет: 8265.4699
Задание №4:
$a = 4,72;\ b = 1,25;\ d = -0,01;\ x = 2,25;\ i = 2;\ k = 3;$
$y = \frac{ax^2+|d|}{(a+b)^2} - 10^4\sqrt[5]{\frac{kx}{(a+b)^2}} - \frac{\cos i}{\sin kx}.$
Контрольный счет: -7167.61
Задание №5:
$a = -3,25;\ x = 8,2;\ k = 4;\ b = 0,05;\ d = 0,95;$
$y = (x-a)\cos k + \frac{\sqrt[5]{|x+a|}}{2,4b}e^3 + 10^{-4}\frac{(x+a)^3+x^4d}{k(x-a)^3}.$
Контрольный счет: 222.99054
Задание №6:
$x = 0,48;\ b = -0,31;\ c = 1,72;\ a = 2,01;\ k = 3;$
$y = \sqrt[5]{|ax^2-b^3|} + lnkx - \frac{e^{kx}+c^2}{\sin kx} - 10^{-3}\sqrt{2157}.$
Контрольный счет: -6.0546836

Лабораторная работа №2. Форматный вывод в командное окно

Цель работы

1. Повторить запись арифметических выражений.
2. Научиться создавать и запускать программы.
3. Освоить функцию mprintf форматного вывода.

Краткие методические указания

1. Создать программу SciLab в sce-файле (редактор SciNotes), в котором задать путем присваивания необходимые исходные данные. Переменным дать названия, совпадающие с указанными в варианте задания.
2. Записать выражения для решения задачи. Вывести результаты расчета и исходные данные в командное окно SciLab с помощью функции mprintf форматного вывода. Вывод осуществить именно так, как указано в варианте задания.
3. Вывести все заданные и рассчитанные числовые значения с 2-3 знаками после десятичной точки. Ввод и вывод угловых значений осуществить в градусах. Вывести рядом с числовыми значениями названия единиц измерения.

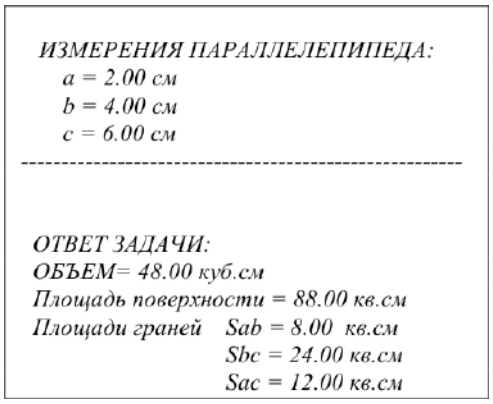
Задание

Задать длины сторон a, b, c прямоугольного параллелепипеда. Вычислить его объем V, площади трех граней Sab, Sbc, Sac и площадь поверхности S по формулам:

$V = a \cdot b \cdot c$

$$Sab = a \cdot b, \quad Sbc = b \cdot c, \quad Sca = c \cdot a,$$

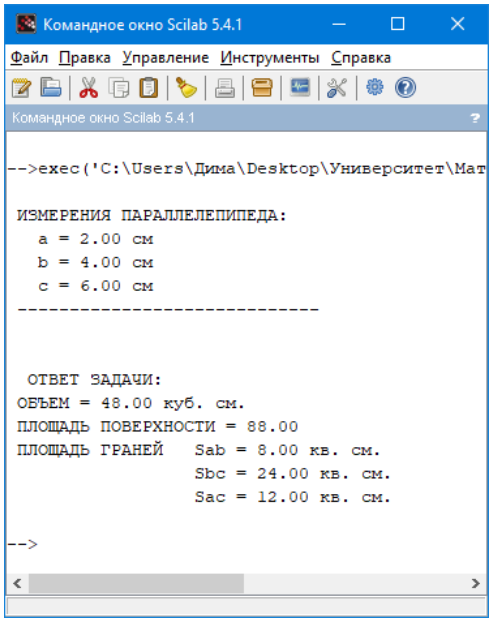
$S = 2 \cdot (Sab + Sbc + Sca)$.
Вывести исходные и рассчитанные значения в виде:



Код программы

```
//Лабораторная работа №2
//Исходные данные
a=2.00;b=4.00;c=6.00;
 //Расчет
V=a*b*c;
Sab=a*b;Sbc=b*c;Sca=c*a;
S=2*(Sab+Sbc+Sca);
 //Выводим в командное окно
mprintf('\n')
mprintf(' ИЗМЕРЕНИЯ ПАРАЛЛЕЛЕПИПЕДА:\n')
mprintf('   a = %.2f см\n   b = %.2f см\n   c = %.2f см\n',a,b,c)
mprintf(' ----------------------------\n\n\n ')
mprintf(' ОТВЕТ ЗАДАЧИ:\n')
mprintf(' ОБЪЕМ = %.2f куб. см.\n',V)
mprintf(' ПЛОЩАДЬ ПОВЕРХНОСТИ = %.2f \n',S)
mprintf(' ПЛОЩАДЬ ГРАНЕЙ   Sab = %.2f кв. см.\n',Sab)
mprintf('                  Sbc = %.2f кв. см.\n',Sbc)
mprintf('                  Sac = %.2f кв. см.\n',Sca)
```

Результат тестирования



Задания для самостоятельной работы

Задание №1:

Задать длины полуосей a, b эллипса.

Вычислить длину эллипса по приближенной формуле:

$$l = 2\pi\left(\frac{3}{2} \cdot \frac{a+b}{2} - \frac{1}{2}\sqrt{ab}\right).$$

Вывести полученные и исходные значения в виде:

ИСХОДНЫЕ ДАННЫЕ

Полуоси эллипса: a= 3.00 мм b= 6.00 мм

РЕЗУЛЬТАТЫ РАСЧЕТА

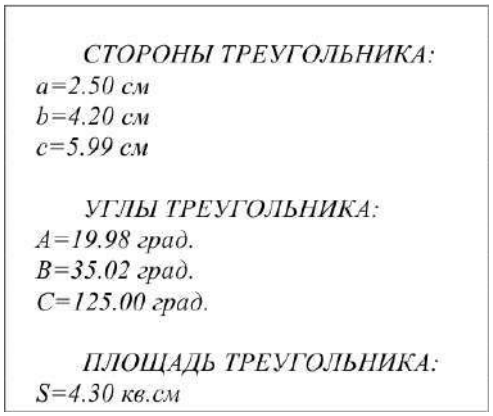
Длина эллипса = 29.08 мм

Задание №2:

Задать две стороны a и b треугольника и угол C между ними. Найти третью сторону c, два других угла A, B и площадь треугольника S:

$$c = \sqrt{a^2 + b^2 - 2ab\cos C}, \quad A = \arccos\frac{b^2+c^2-a^2}{2bc}$$
$$B = 180^\circ - (A + C), \quad S = \frac{1}{2}ac\sin B.$$

Вывести полученные и исходные значения в виде:

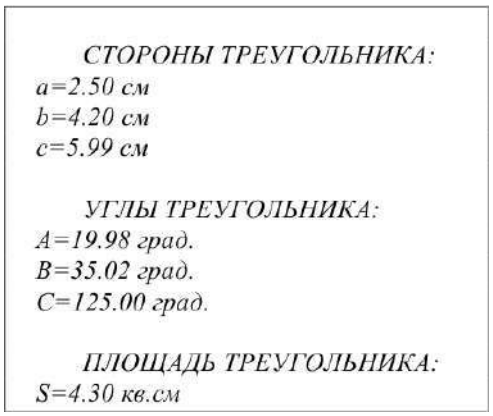


Задание №3:

Задать стороны треугольника $a, b, .$ Вычислить углы A, B, C и площадь S треугольника:

$$A = \arccos\frac{b^2+c^2-a^2}{2bc},\ B = \arccos\frac{a^2+c^2-b^2}{2ac},$$
$$C = 180^\circ - (A+B),\ S = \frac{1}{2}ab\sin C.$$

Вывести рассчитанные значения и исходные данные в виде::



Лабораторная работа №3. Построение и оформление графиков функций

Цель работы

1. Приобрести навыки построения графиков в SciLab.
2. Изучить возможности оформления графиков и графических окон.
3. Закрепить знания и навыки по работе с массивами

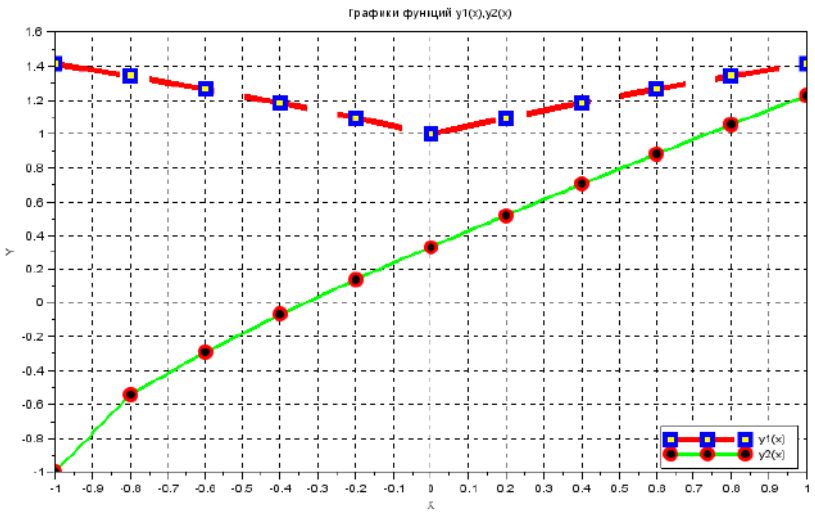
Краткие методические указания

1. В программе создать массив значений аргумента, и получить массивы значений двух функций. Использовать поэлементные операции с массивами.
2. Построить графики двух функций в одном окне. Задать такие шаг и диапазон изменения аргумента, которые показаны в примере.
3. Применить все возможности оформления линий, маркеров и окон.

Задание

$y_1(x) = \sqrt{1+|x|},\ y_2(x) = \frac{1+3x}{\sqrt[3]{1+x}+2}.$

Контрольный результат

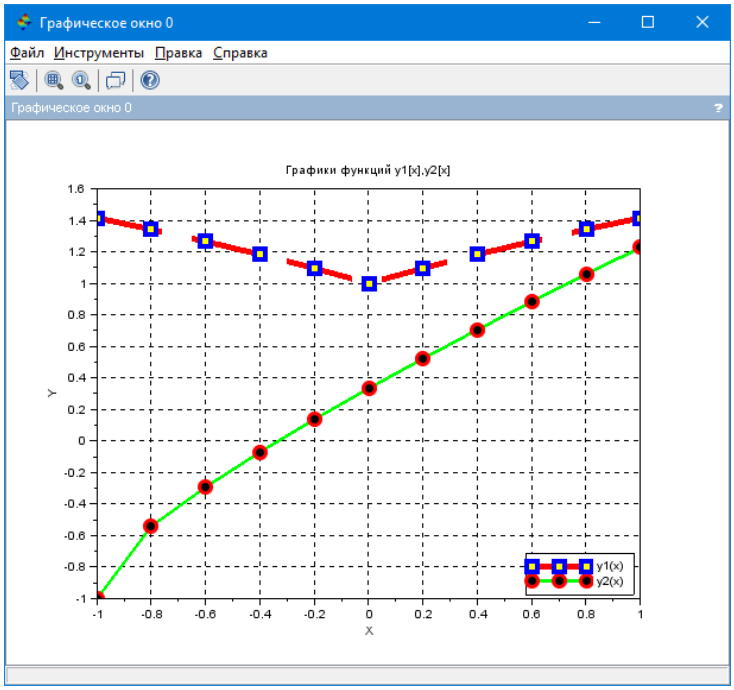


Программа

```
//Лабораторная работа №3
clear //очистка памяти
clc //очистка команданого окна
clf //очистка графического окна
//Задаем вектор значений аргумента x
x=[-1:0.2:1]
//Получаем вектор значений первой функции y1
y1=sqrt(1+abs(x))

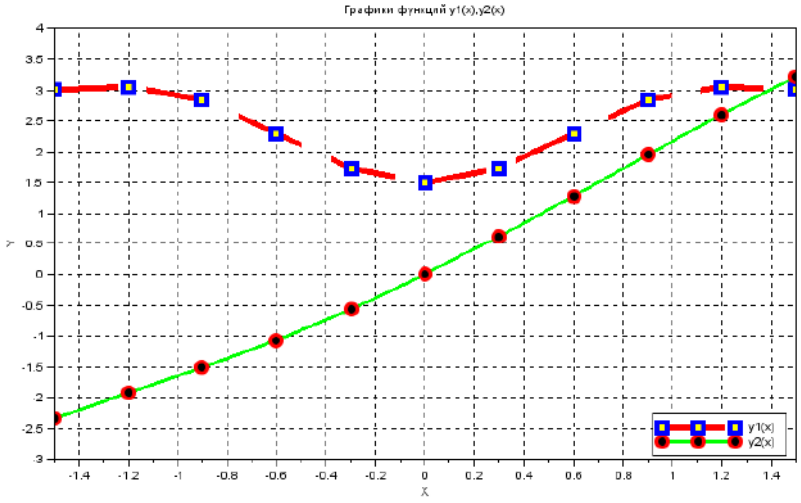
//Строим график первой функции
plot(x,y1,'LineStyle','--','Color','r','Thickness',5,...
'Marker','s','MarkerEdgeColor','b','MarkerFaceColor','y',...
'MarkerSize',10)
//Получаем вектор значений второй функции y2
y2 = (1+3 .*x)./((nthroot((1+x),3)) + 2)
//Строим график второй функции и оформляем линию и маркеры
plot(x,y2,'LineStyle','-','Color','g','Thickness',3,...
'Marker','o','MarkerEdgeColor','r','MarkerFaceColor','k',...
'MarkerSize',10)
//Выводим подписи к области графика и к осям, легенду и сетку
xtitle('Графики функций y1[x],y2[x]','X','Y')
legend('y1(x)','y2(x)',4)
xgrid
```

Результат тестирования

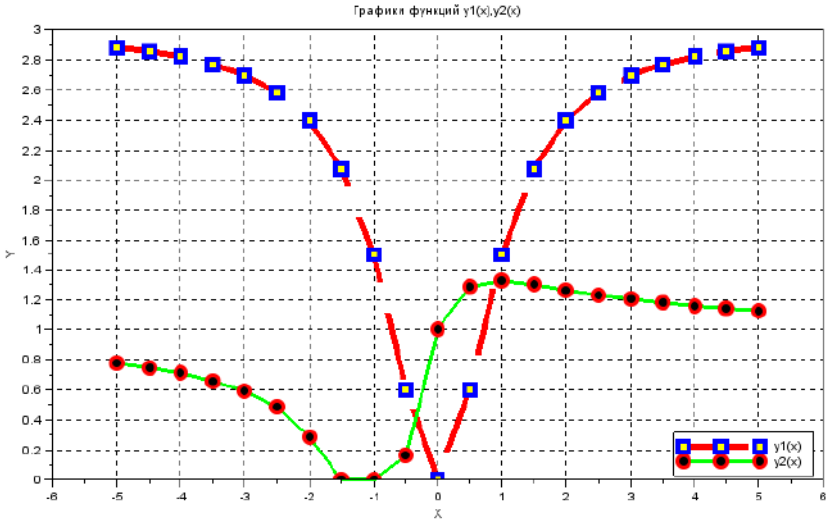


Задания для самостоятельной работы

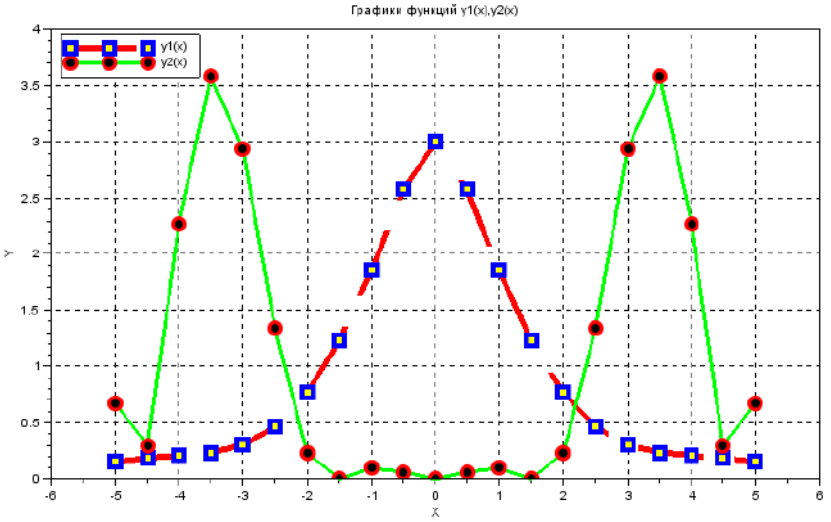
Построить графики следующих функций:
$y_1(x) = \frac{3+\sin^2 2x}{1+\cos^2 x}, \quad y_2(x) = 2x + \frac{\sin^2 x}{3+x}.$
Контрольный график



$$y1(x) = \frac{3x^2}{1+x^2}, \quad y2(x) = \sqrt{1 + \frac{2x}{e^{0,5x}+x^2}}.$$



$$y1(x) = \frac{3+\sin^2 x}{1+x^2}, \quad y2(x) = \frac{1}{3}x^2 \cos^2 x.$$



Лабораторная работа №4. Условные операторы и оператор цикла с условием

Цель работы

1. Ознакомиться с условным оператором и оператором цикла.
2. Приобрести навыки написания программ при разветвленных и циклических вычислениях.
3. Повторить приоритеты операций в выражениях, функцию mprintf.

Краткие методические указания

1. Первая часть задания соответствует условным операторам, а вторая часть — операторам цикла с условием.
2. Для первой части задать значения переменных, применить условные операторы и mprintf. Проверить вычисления по разным направлениям с разными исходными данными.

3. Для второй части задать x, организовать цикл расчета члена и суммы ряда по рекуррентному соотношению, пока не будет достигнута точность 10^{-5}. Вывести номер члена, его значение и сумму.

Задание

Заданы четыре переменные. Наименьшую из них заменить на сумму остальных. Вычислить сумму ряда:

$H_1 = 6x, ..., H_n = -H_{n-1} * \frac{n+2}{n}x.$

Вывести исходные и полученные данные в виде:

Исходные значения: x1=-12, x2=33, x3=-54, x4=47
Полученные значения: x1=-12, x2=33, x3=68, x4=47

x=0.1

n:	*H:*	*S:*
1	*0.6000000000*	*0.6000000000*
2	*-0.1200000000*	*0.4800000000*
3	*0.0200000000*	*0.5000000000*
4	*-0.0030000000*	*0.4970000000*
5	*0.0004200000*	*0.4974200000*
6	*-0.0000560000*	*0.4973640000*
7	*0.0000072000*	*0.4973712000*

Сумма ряда S(x) = 0.4973712000

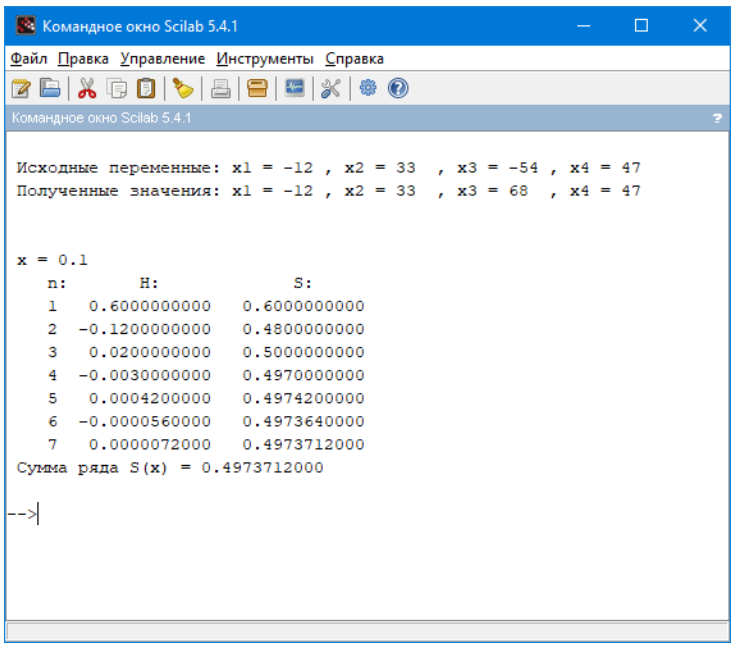
Код программы

```
//Лабораторная работа №4
clear //очистка памяти
clc //очистка командного окна

//Исходные значения для 1 задания
x1=-12; x2=33; x3=-54; x4=47;
mprintf('\n Исходные переменные: x1 = %-4.0f, x2 = %-4.0f, x3 = %-4.0f,
x4 = %-4.0f',x1,x2,x3,x4)
//Находим минимум
min_zn = min(min(x1,x2),min(x3,x4))
if min_zn == x1 then
    x1 = x2+x3+x4
end
if min_zn == x2 then
    x2 = x1+x3+x4
end
if min_zn == x3 then
    x3 = x1+x2+x4
end
if min_zn == x4 then
    x4 = x1+x2+x3
end
mprintf('\n Полученные значения: x1 = %-4.0f, x2 = %-4.0f, x3 = %-4.0f,
x4 = %-4.0f\n\n\n',x1,x2,x3,x4)
```

```
//Значени x для 2-й части
x =0.1;
mprintf(' x = %g\n',x)
//Начальные значения члена, номера и суммы ряда
n = 1; H = 6*x; S = H;
mprintf('%7s%10s%15s\n','n: ','H:  ','S:  ');
mprintf('%5d%15.10f%15.10f\n',n,H,S);
//цикл расчета
while abs(H) > 10^(-5)
    n = n + 1;
    H = (-H) * ((n+2)/n) * x;
    S = S + H;
    mprintf('%5d%15.10f%15.10f\n',n,H,S);
end
mprintf(' Сумма ряда S(x) = %.10f\n',S);
```

Результат тестирования



Задания для самостоятельной работы

Задание №1:
Найти сумму положительных из четырех заданных значений.
Вычислить сумму ряда:

$$H_1 = x, ..., H_n = H_{n-1} \cdot \frac{4n-7}{4n-3} x^4.$$

Вывести исходные и полученные данные в виде:

```
x1=-2,  x2=4,  x3=-4,  x4=7
Сумма положительных из x1, x2, x3, x4 равна 11

x=0.7
n:        H:             S:
 1  0.7000000000  0.7000000000
 2  0.0336140000  0.7336140000
 3  0.0044837341  0.7380977341
 4  0.0007453001  0.7388430342
 5  0.0001368415  0.7389798757
 6  0.0000265974  0.7390064731
 7  0.0000053643  0.7390118374
Сумма ряда S(x) = 0.7390118374
```

Задание №2:

Определить нмоер квадранта (четверти) на координатной плоскости, в которой находится точка с заданными координатами.

Вычислить сумму ряда:

$$H_1 = -x, ..., H_n = -H_{n-1} \cdot \frac{x^2}{(2n-1)(2n-2)}.$$

Вывести исходные и полученные данные в виде:

```
Координаты точки:  x=5,  y=-13
Точка находится в IV квадранте (четверти)

x=2
n:        H:             S:
 1 -2.0000000000 -2.0000000000
 2  1.3333333333 -0.6666666667
 3 -0.2666666667 -0.9333333333
 4  0.0253968254 -0.9079365079
 5 -0.0014109347 -0.9093474427
 6  0.0000513067 -0.9092961360
 7 -0.0000013156 -0.9092974515
Сумма ряда S(x) = -0.9092974515
```

Задание №3:

Найти все пары одинаковых значений среди четырех переменных.

Вычислить сумму ряда:

$$H_1 = 1, ..., H_n = -H_{n-1} \cdot \frac{(2n^2+1)x^2}{8n^4-20n^3+20n^2-6n}.$$

Вывести исходные и полученные данные в виде:

```
x1=2,  x2=-6,  x3=2,  x4=-6
Пары одинаковых значений:
x1=x3=2
x2=x4=-6

x=2.1
n:        H:              S:
  1   1.0000000000   1.0000000000
  2  -1.1025000000  -0.1025000000
  3   0.3421425000   0.2396425000
  4  -0.0467969906   0.1928455094
  5   0.0035438085   0.1963893178
  6  -0.0001694680   0.1962198499
  7   0.0000055689   0.1962254188
Сумма ряда S(x) = 0.1962254188
```

Лабораторная работа №5. Работа с числовыми массивами в Scilab

Цель работы

1. Приобрести навыки применения арифметических, логических и операций отношения к массивам (векторам и матрицам).
2. Освоить множественную и логическую индексацию массивов.
3. Изучить функции обработки данных в массивах.

Краткие методические указания

1. Исходные значения задавать прямо в файле с программой. Для первой части работы задать одномерные массивы (вектора), а для второй части — двумерные массивы (матрицы).
2. Для вывода отдельных значений и пояснений использовать функцию mprintf, а для вывода числовых массивов использовать функцию disp. Вывод осуществить именно так, как указано в варианте задания.
3. Программа должна быть универсальной, т. е. выполняться для произвольного количества элементов исходного вектора и произвольного числа строк и столбцов исходной матрицы.

Задание

Среди n первых элементов вектора найти сумму отрицательных элементов. Элементы квадратной матрицы ниже главной диагонали уменьшить на x_1, а элементы выше главной диагонали увеличить на x_2.

Вывести исходные и полученные данные в виде:

```
Исходный вектор:
 - 2.   9.  - 13.   1.   12.  - 15.   9.   11.  - 8.
Число n = 6
Сумма отрицательных среди 6 первых элементов  равна -30

 Исходная матрица:
   1.    10.   10.   10.
 - 10.   9.    10.   10.
 - 10.  - 10.  4.    10.
 - 10.  - 10.  - 10.  13.
x1 = 2  x2 = 4
 Полученная матрица:
   1.    14.   14.   14.
 - 12.   9.    14.   14.
 - 12.  - 12.  4.    14.
 - 12.  - 12.  - 12.  13.
```

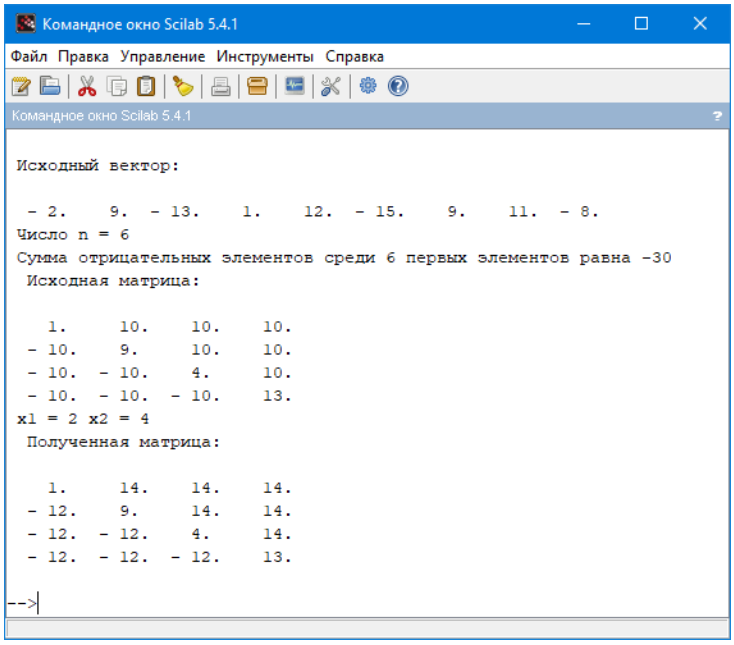
Код программы

```
//Лабораторная работа №5
clear
clc
//1.
//Исходный вектор
V = [-2 9 -13 1 12 -15 9 11 -8];
//Вывод вектора
disp(V,'Исходный вектор:')
n = 6
mprintf(' Число n = %d \n',n)
//Найдем первые 6 элементов
V1 = V(1:1:6)
// Выделим из них только отрицательные
L = V1 < 0
// Получим вектор отрицательных
V2 = V1(L)
// Теперь запишем сумму
Sum = sum(V2)
//Выводим на экран
mprintf(' Сумма отрицательных элементов среди %d первых элементов равна %d',n,Sum)

//2.
//Задаем матрицу
M = [1 10 10 10
    -10 9 10 10
    -10 -10 4 10
    -10 -10 -10 13];
//Выводим Матрицу
disp(M,' Исходная матрица: ')
//Задаем значения переменным x1, x2
x1 = 2; x2 = 4;
mprintf(' x1 = %d x2 = %d',x1,x2)
//Создадим 2 единичные матрицу
M1 = ones(M)
M2 = ones(M)
//Увеличим числа ниже главной диагонали на x1
```

```
M1 = x1*tril(M1,-1)
//Увеличим числа выше главной диагонали на x2
M2 = x2*triu(M2,1)
//Найдем разницу между исходной матрицей и матрицей M1, а также сумму
//между матрицей M и M2
M = M - M1 + M2
disp(M,' Полученная матрица: ')
```

Результат тестирования



Задания для самостоятельной работы

Задание №1:

Найти сумму отрицательных элементов вектора.

В матрице определить произведение элементов, расположенных на нечетных местах в каждой строке. Заменить полученными значениями элементы первого столбца матрицы.

Вывести исходные и полученные данные в виде:

Исходный вектор:
2. - 1. 6. 8. - 4. - 6.
Сумма отрицательных элементов вектора: -11

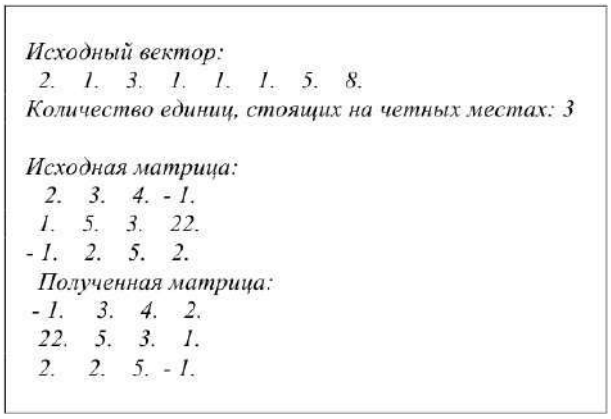
Исходная матрица:
2. 3. 7.
- 1. 2. - 6.
6. - 7. 9.
Полученная матрица:
14. 3. 7.
6. 2. - 6.
54. - 7. 9.

Задание №2:

Подсчитать количество «единиц» на четных местах вектора.

В матрице поменять столбец, в котором находится максимальный элемент с первым столбцом.

Вывести исходные и полученные данные в виде:

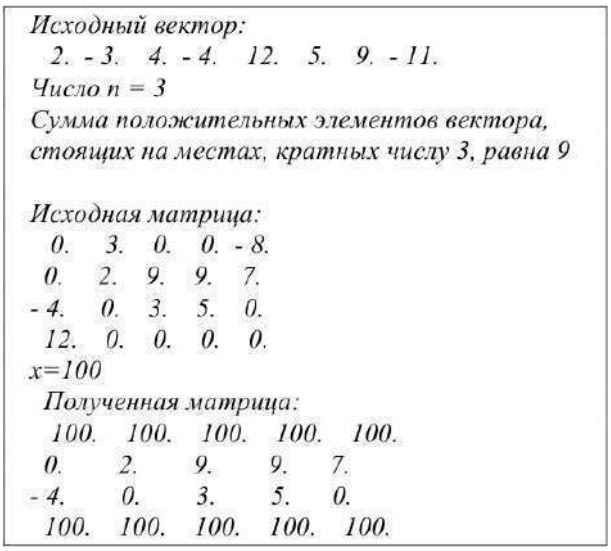


Задание №3:

Найти сумму положительных элементов вектора, стоящих на местах, кратных числу n.

В матрице определить строки, в которых расположено более чем два элемента, равных нулю. Заменить все элементы этих строк на x.

Вывести исходные и полученные данные в виде:



Лабораторная работа №6. Циклы с параметром и обработка массивов

Цель работы

1. Ознакомиться с оператором цикла с заданным числом повторений
2. Приобрести навыки поэлементной обработки векторов и матриц с использованием операторов цикла и условных операторов.
3. Повторить условные операторы.

Краткие методические указания

1. Для первой части задать одномерные массивы (вектора), а для второй части — двумерные массивы (матрицы). Вывести исходные и полученные данные поэлементно с использованием циклов и функции mprintf именно так, как приведено в варианте задания.

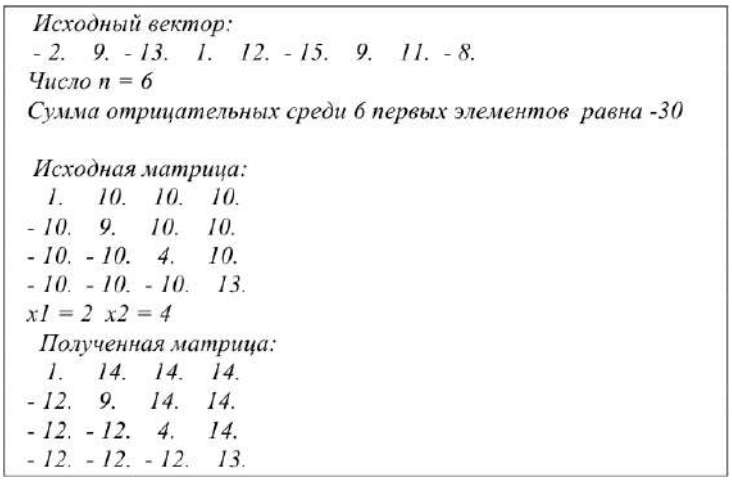
2. Задачу выполнить с использованием операторов цикла и условных операторов, не используя возможности SciLab по обработке массивов. Программа должна выполняться для произвольного количества элементов исходного вектора и числа строк и столбцов исходной матрицы

3. Варианты заданий взять из лабораторной работы №3.

Задание

Среди n первых элементов вектора найти сумму отрицательных элементов. Элементы квадратной матрицы ниже главной диагонали уменьшить на x_1, а элементы выше главной диагонали увеличить на x_2.

Вывести исходные и полученные данные в виде:



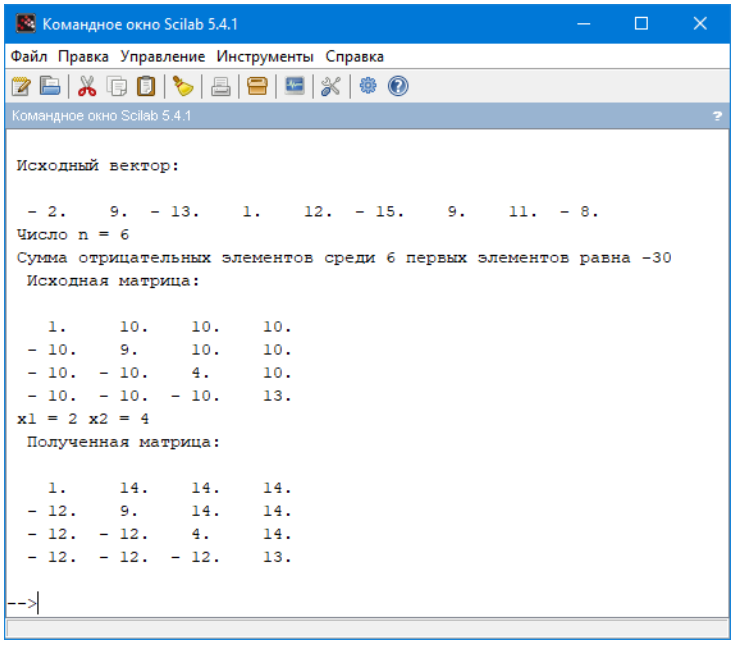
Код программы

```
//Лабораторная работа №6
clear
clc
//Исходный вектор
V = [-2 9 -13 1 12 -15 9 11 -8];
//Опеределяем количество элементов в вектор
kol = length(V);
//Выводим вектор
mprintf('Исходный вектор: \n')
for i=1:kol do
    mprintf('%5g',V(i))
end
mprintf('\n')
//Решение 1 части
n = 6;
mprintf(' Число n = %g\n',n)
sum_negative_number = 0;
for i=1:n do
```

```
    if V(i) < 0 then
        sum_negative_number = sum_negative_number + V(i)
    end
end
//Полученные значения
mprintf(' Сума отрицательных среди 6 первых элементов равна %g\n\n',sum_negative_number)

//Вторая часть задания
//Задаем матрицу
M = [1 10 10 10
    -10 9 10 10
    -10 -10 4 10
    -10 -10 -10 13];
//Выводим матрицу
kstr=size(M,'r'); kcol=size(M,'c');
//Выводим исходную матрицу
mprintf(' Исходная матрица:\n')
for i=1:kstr do
    for j=1:kcol do
        mprintf('%5g',M(i,j))
    end
    mprintf(' \n')
end
//Зададим значения x1 и x2
x1=2; x2=4;
mprintf('\n x1 = %g x2 = %g\n\n',x1,x2)
//Увеличим значения матрицы выше главной диагонали на x2
for i=1:kcol do
    for j=i+1:kstr do
        M(i,j) = M(i,j) + x2
    end
end
//Уменьшим значения матрицы ниже главной диагонали на x1
for i=1:kcol do
    for j=i+1:kstr do
        M(j,i) = M(j,i) - x1
    end
end
//Полученная матрицы:
mprintf(' Полученная матрица: \n')
for i=1:kstr do
    for j=1:kcol do
        mprintf('%5g',M(i,j))
    end
    mprintf(' \n')
end
```

Результат тестирования



Задания для самостоятельной работы

Выполнить решение заданий №1-№3 из лабораторной работы №5, используя циклы с параметром и обработку массивов.

Литература

1. Введение в математическое моделирование: Учеб. пособие / Под ред. П. В. Трусова. М.: Университетская книга, Логос, 2007. - 440с.

2. Алексеев Е. Р. Scilab: Решение инженерных и математических задач / Е. Р. Алексеев, О. В. Чеснокова, Е. А. Рудченко. – М. : ALT Linux ; БИНОМ. Лаборатория знаний, 2008 – 269 с. : ил. ; 8 с. цв. вклейки.– (Библиотека ALT Linux).

3. Бородин И. А., Решение инженерных и математических задач в Scilab. – Ярославль:ЯГТУ, 2009

4. Тропин И. С., Михайлова О. И., Михайлов А. В. Численные и технические расчеты в среде Scilab (ПО для решения задач численных и технических ПО для решения задач численных и технических вычислений): Учебное пособие. — Москва: 2008 – 64 с.

5. Андриевский Б. Р., Фрадков А. Л. Элементы математического моделирования в программных средах MatLab 5 и Scilab. — СПб.: Наука, 2001 — 286 с.

6. Александров, В. А. Преобразование Фурье: Учебное пособие. / В. А. Александров. – Новосибирск: НГУ, 2003. – 61 с.

7. Серегин Н. И. Особенности использования дискретного преобразования Фурье при спектральном анализе: Учебное электронное текстовое издание / Н. И. Серегин. – УПИ: Екатеринбург, 2006. – 36 с.

8. Головин Д. А. Использование программного пакета Scilab для проведения математических расчетов // Системы управления, технические системы: устойчивость, стабилизация, пути и методы исследования: материалы V Международной научно-практической конференции, посвященной 70-летию со дня рождения профессора Ю.Н. Меренкова. – Елец: Елецкий государственный университет им. И.А. Бунина, 2019. – С. 188–193.

Учебное издание

Елена Викторовна Игонина

Программные средства математического моделирования

Учебное пособие

Технический редактор —
Техническое исполнение —

Лицензия на издательскую деятельность
ИД №06146. Дата выдачи 26.10.01
Формат 60 x 84/16. Гарнитура Times. Печать трафаретная
Печ.л 5,3 Уч.-изд.л.5,1
Тираж 100 экз. (1-й завод 1 – 30 экз.). Заказ 100

Отпечатано с готового оригинал-макета на участке оперативной полиграфии
ФГБОУ ВО «Елецкий государственный университет им. И.А. Бунина»

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Елецкий государственный университет им. И.А. Бунина»
399770, г. Елец, ул. Коммунаров, 28, 1