Министерство образования Республики Беларусь

Учреждение образования
"Гомельский государственный технический университет
имени П.О.Сухого"

Кафедра "Информационные технологии"

РЕШЕНИЕ ИНЖЕНЕРНЫХ ЗАДАЧ В СИСТЕМЕ MATLAB

ПРАКТИЧЕСКОЕ ПОСОБИЕ

по курсу "Информатика" для студентов технических специальностей дневного отделения

Гомель 2004

УДК 621.

Авторы-составители: В.В.Кротенок, Т.Л.Романькова, Т.А.Трохова
Рецензент:

Решение инженерных задач в системе matlab: Практическое пособие по курсу "Информатика" для студентов технических специальностей дневного отделения/ Авт.- сост. В.В.Кротенок, Т.Л.Романькова, Т.А.Трохова. – Гомель: ГГТУ им.П.О.Сухого, 2004. – 36с.

1 Программирование базовых алгоритмов в Matlab
1.1 Обработка М-файлов

М-файл представляет собой программу, состоящую из команд и выражений системы MatLab, хранящуюся на диске в виде файла с типом ***.m.*** Создать новый М-файл можно с помощью команд основного меню

File – New – M-file,

после чего на экране появляется окно редактора М-файлов.

Окно содержит сервисно-командную область (три верхние строки) и область ввода и редактирования М-файла. Для удобства отладки строки команд программы пронумерованы. Вид окна редактора М-файлов приведен на рисунке 1.1.

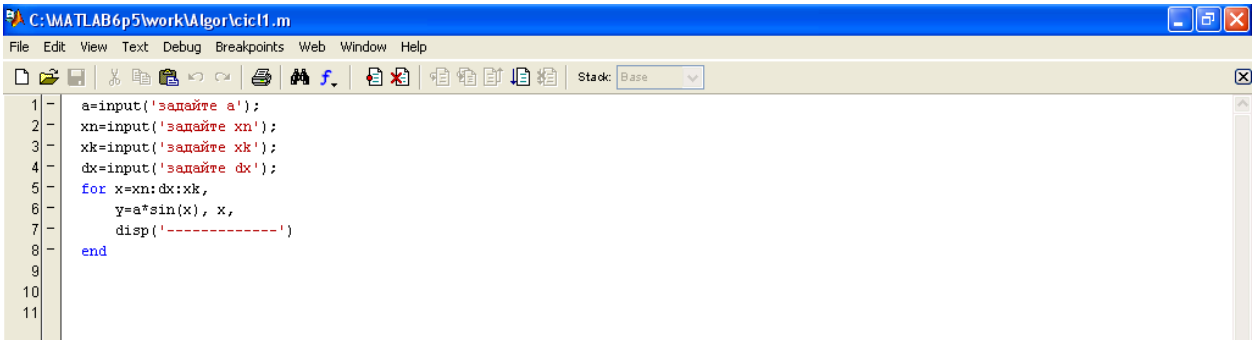


Рисунок 1.1 – Вид окна редактора М-файлов

Последовательность обработки М-файла такова.

1. Создать или отредактировать М-файл
2. Записать файл на диск с именем, содержащим тип **.m**
3. Запустить программу на выполнение, указав имя М-файла в командной строке окна команд Command Window
4. Если компиляция программы прошла успешно, то результаты выполнения программы будут отражены в командном окне.
5. Если в результате компиляции были найдены ошибки в программе, то необходимо вызвать программу в окно М-файла и повторить последовательность обработки программы, начиная с п.1, исправив ошибки
6. Вывести текст программы на принтер можно с помощью команд

   File – Print

   меню команд окна редактора М-файлов

Вывести результаты расчетов по программе можно с помощью команд основного меню рабочего стола Matlab:

File – Print (для вывода всей информации командного окна),

File – Print Selection (для вывода выделенной области командного окна).

При работе с программой пользователь может разместить окна на экране дисплея так, чтобы был виден текст программы (окно М-файла),

результаты расчетов (командное окно) и переменные, размещенные в памяти (окно рабочей области памяти). Пример такого удобного размещения окон приведен на рисунке 1.2.

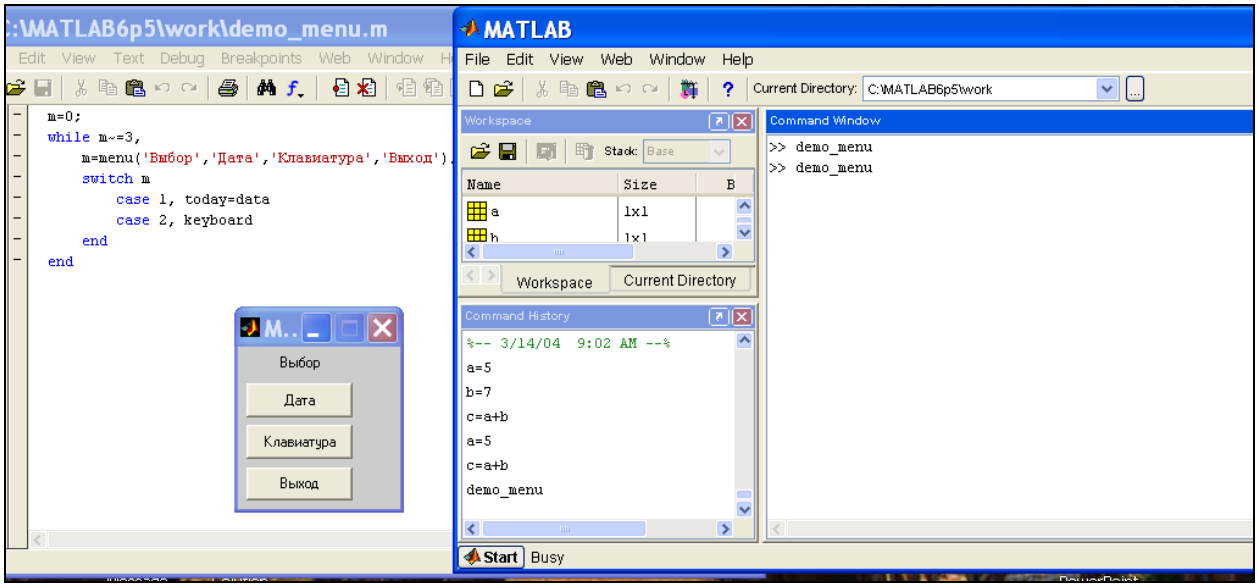


Рисунок 1.2 - Работа в командном режиме и в режиме создания М-файлов.

1.2 Программирование линейных алгоритмов

1.2.1 Оператор присваивания

При составлении линейных программ с помощью М-файлов в Matlab одним из основных операторов является оператор присваивания. В первой части практического пособия изложены приемы применения этого оператора при вычислениях в командном режиме Matlab. В программе этот оператор выполняет те же самые функции, т.е. присваивает переменной, стоящей слева от знака «=» значение выражения, стоящего справа.

Общий вид оператора присваивания:

Имя_переменной = Выражение

В качестве параметра **Имя_переменной** может выступать имя простой переменной, структурированной переменной (вектора, матрицы), имя функции. В качестве параметра **Выражение** применяется арифметическое, логическое или строковое выражение.

Тип переменной определяется системой автоматически по типу выражения, поэтому нет необходимости при программировании следить за соответствием типов данных в операторе присваивания. Если выражение содержит и арифметические и строковые элементы, то переменная будет численной, т.е. предпочтение отдается числовому типу данных.

Ниже приведены примеры правильной записи операторов присваивания.

A = cos(x)+c-d^2*p^2+4.92

N = 'номер формулы'

R = (x>5)&(x<=10)

В первом примере в переменную помещается арифметическое выражение, во втором – символьное, в третьем – логическое.

1.2.2 Программирование ввода и вывода данных

При программировании часто возникает необходимость вводить исходные данные не с помощью оператора присваивания, а с помощью операторов ввода в диалоговом режиме. Преимущества такого ввода очевидны: при вводе новых исходных данных нет необходимости вносить изменения в программу.

В MatLab в качестве оператора ввода используется функция **input**, которую, в силу ее значимости при программировании, принято называть оператором.

Она имеет следующий общий вид:

ИМЯ = input(Символьная константа)

Здесь **ИМЯ –** это имя простой переменной, **Символьная константа –** любой набор символов, заключенный в апострафы. Символьная константа, как правило, разъясняет смысловое назначение вводимой переменной.

Например:

S=input('Задайте площадь')

A=input('Задайте значение A=')

Оператор выполняется следующим образом:

- в командном окне выводится набор символов, стоящую в скобках после input (символьная константа);
- выполнение программы приостанавливается и компьютер переходит в режим ожидания;
- пользователь вводит константу;
- введенная константа помещается в оперативной памяти в переменную, стоящую слева в операторе input.

При запуске на выполнение программы, содержащей оператор ввода следует учитывать, что пока пользователь не ввел константу в ответ на запрос своей программы, оператор ввода продолжает свою работу. Система Matlab в это время блокирет выход и закрытие окна рабочего стола.

Если необходимо вывести данные на экран дисплея в определенной последовательности, отличной от последовательности их вычисления, применяется функция disp, которую принято называть оператором вывода.

Оператор имеет следующий общий вид:

disp(Выражение)

Здесь **Выражение –** это арифметическое, логическое или символьное выражение, частным случаем которого являются константы или переменные любого типа.

Примеры правильной записи оператора вывода приведены ниже.

```
disp('Результаты ')      a=[1,6,9,2];      x=7;
disp(summa)              disp(a)           disp((x>5)&(x<=10))
disp(5+6)
```

Следует помнить, что Matlab выводит в командном окне значение переменной, стоящей слева в операторе присваивания, если оператор не заканчивается символом «**;**», поэтому если в программе используется оператор вывода, нужно подавлять дублирование вывода, указывая символ «**;**» в конце оператора присваивания.

Каждый новый оператор **disp** выполняет вывод с новой строки командного окна, например:

Фрагмент прграммы	Командное окно
c=a-b+k*d;	**результат=**
disp ('результат=')	**28**
disp(c);	

Если необходимо вывести несколько данных в одной строке, нужно сформировать из них вектор-строку, который будет использоваться затем в выражении в операторе disp, например:

Фрагмент прграммы	Командное окно
c=a-b+k*d; r='результат='	**результат=28 при a=5**
r1=' при a='; x=[r, c, r1, a];	
disp(x);	

1.2.3 Пример линейной программы

В данном пособии примеры программирования задач рассматриваются по следующему плану: условие задачи, алгоритмический анализ задачи, графическая схема алгоритма решения задачи, программа, тесты. На все виды базовых алгоритмов предложены задачи прикладного содержания.

Задача 1

Условие задачи. Для динамической колебательной системы, содержащей массу ***m,*** пружину с коэффициентом жесткости ***k*** и демпфер с коэффициентом демпфирования ***c***, вычислить резонансную частоту системы по формуле:

$$fr = fn\sqrt{1-2s^2},$$

где коэффициент затухания системы **s** вычисляется по формуле:

$$s = \frac{c}{2\sqrt{km}}$$

Собственная частота незатухающих колебаний системы рассчитывается по формуле:

$$fn = \frac{1}{2\pi}\sqrt{\frac{k}{m}}$$

Алгоритмический анализ задачи

Исходные данные:
m – масса системы;
k - коэффициент жесткости пружины;
c - коэффициент демпфирования.
Результат:
fr - резонансная частота системы.
Промежуточные данные:
s - коэффициент затухания системы
fn - собственная частота незатухающих колебаний

Графическая схема алгоритма решения задачи

Графическая схема алгоритма решения задачи приведена на рисунке 1.3.

Текст программы

```
m=input(' Задайте массу системы ');
k=input(' Задайте коэффициент жесткости пружины ');
c=input(' Задайте коэффициент демпфирования ');
s=c/(2*sqrt(k*m));
fn=1/(2*pi)*sqrt(k/m);
fr=fn*sqrt(1-2*s^2);
disp ('Резонансная частота системы =')
disp(fr)
```

Тестовый пример:

Задайте массу системы 10
Задайте коэффициент жесткости пружины 10
Задайте коэффициент демпфирования 0.6
Резонансная частота системы =
0.1590

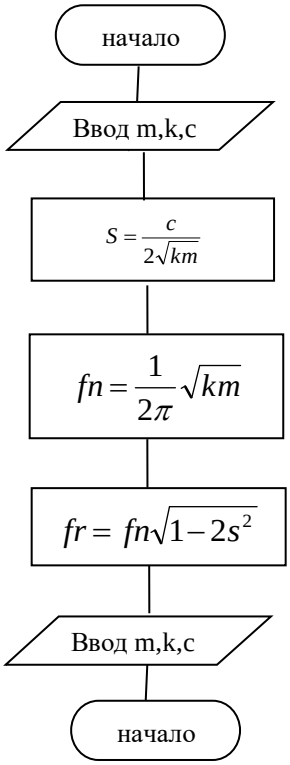


Рисунок 1.3 Графическая схема алгоритма решения задачи

1.3 Программирование разветвляющихся алгоритмов

1.3.1 Операторы условия

Для программирования разветвляющихся алгоритмов в Matlab существуют оператор условия и оператор выбора.

Условный оператор представлен в нескольких формах и имеет следующий общий вид:

Полная форма 1

```
if  ЛВ,  ОПЕРАТОР1,
        else   ОПЕРАТОР2,
end
```

Полная форма 2

```
if  ЛВ1,  ОПЕРАТОР1,
      elseif  ЛВ2, ОПЕРАТОР2,
      elseif  ЛВ3, ОПЕРАТОР3,
      …
                  else   ОПЕРАТОР,
  end
```

Краткая форма

if ЛВ, ОПЕРАТОР1
end

Здесь
ЛВ, ЛВ1, ЛВ2, ЛВ3 – логические выражения;
ОПЕРАТОР1, ОПЕРАТОР2, ОПЕРАТОР3, ОПЕРАТОР– любые операторы или группы операторов языка Matlab.

Разделителями в операторах могут быть запятая или точка с запятой. Если ОПЕРАТОР расположен в следующей строке, то разделитель ставить не обязательно.

Порядок выполнения оператора условия следующий.

Полная форма 1

- Логическое выражение вычисляется до константы (0 или 1)
- Если логическое выражение истинно, то выполняется оператор, стоящий после логического выражения, а затем следующий за оператором if оператор
- Если логическое выражение ложно, то выполняется оператор, стоящий после слова else, а затем следующий за оператором if оператор

Полная форма 2

- Логическое выражение первого оператора if вычисляется до константы (0 или 1)
- Если логическое выражение истинно, то выполняется оператор, стоящий после логического выражения, а затем следующий за оператором if оператор
- Если логическое выражение ложно, то вычисляется логическое выражение, стоящее после слова **elseif,** встречающегося первый раз
- Если логическое выражение истинно, то выполняется оператор, стоящий после логического выражения, а затем следующий за оператором if оператор
- Процесс повторяется столько раз, сколько слов **elseif** содержит оператор
- Если последнее логическое выражение ложно, то выполняется оператор, стоящий после слова else, а затем следующий за оператором if оператор

Краткая форма

- Логическое выражение вычисляется до константы (0 или 1)

- Если логическое выражение истинно, то выполняется оператор, стоящий после логического выражения, а затем следующий за оператором if оператор
- Если логическое выражение ложно, то выполняется следующий за оператором if оператор

Графические схемы алгоритмов выполнения различных форм оператора if приведены на рисунке 1.4.

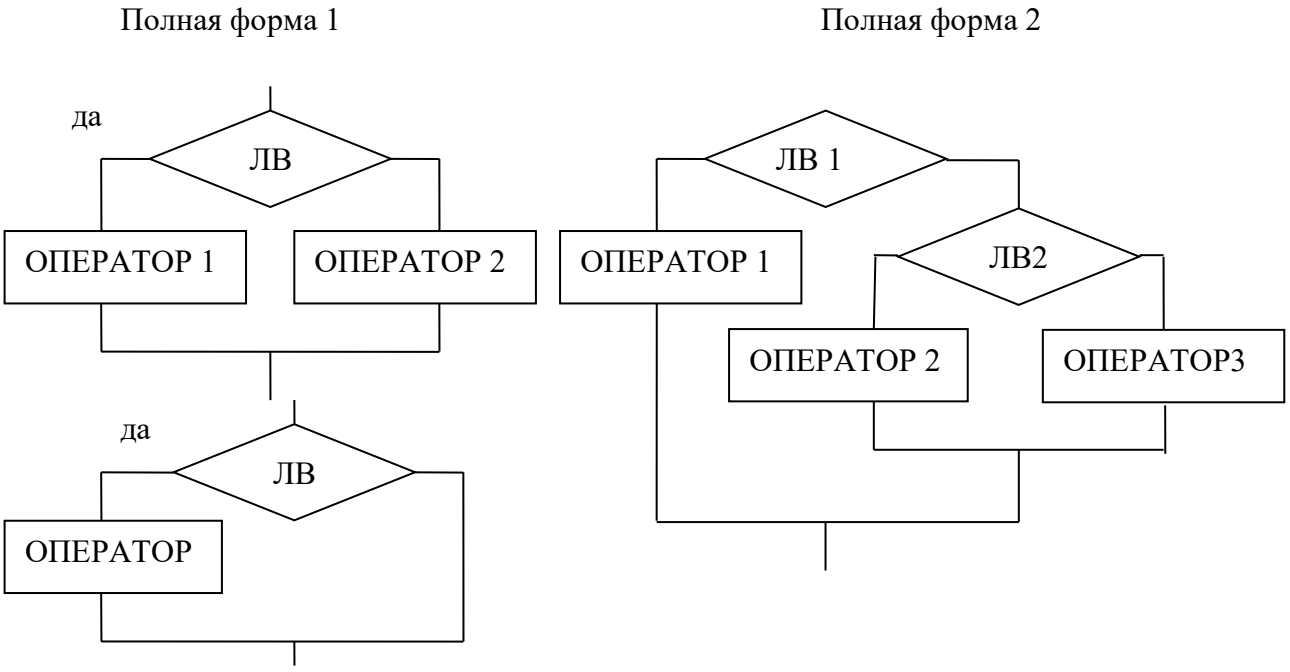


Рисунок 1.4. Графические схемы алгоритмов выполнения различных форм оператора if

Примеры правильной записи оператора условия (считается, что все переменные заданы корректно) приведены ниже.

```
if    b*b - 4*a*c < 0,
                  disp('нет корней'),
                                else
                   disp('есть корни')
  end


if    (a > 0) & (b>0),
                  disp(' числа положительные'),
   elseif    (a < 0) & (b<0) ,
```

```
                disp('числа отрицательные')
    else,
                disp('числа разные по знаку')
    end

if   D < 0,
                disp('Решения нет'),
                A=0
  end
```

1.3.2 Оператор выбора

Оператор выбора предназначен для управления разветвляющимся вычислительным процессом, когда необходимо сделать выбор из произвольного числа имеющихся вариантов.

Общий вид оператора следующий.

Полная форма

```
switch  ВС
      case ск1, оператор1;
      case ск2, оператор2;
      …
      case скN, операторN
       otherwise оператор
   end
```

Краткая форма

```
switch  ВС
      case ск1, оператор1;
      case ск2, оператор2;
      …
      case скN, операторN
   end
```

Здесь

ВС – выражение-селектор или выражение выбора

ск1, ск2, скN- константы выбора,

оператор, оператор1, оператор2, операторN – любые операторы или группы операторов.

Константы выбора могут быть объединены в множества с помощью {}, например, {5, 7, 8, 4}.

Графические схемы алгоритма выполнения оператора case приведены на рисунке 1.5.

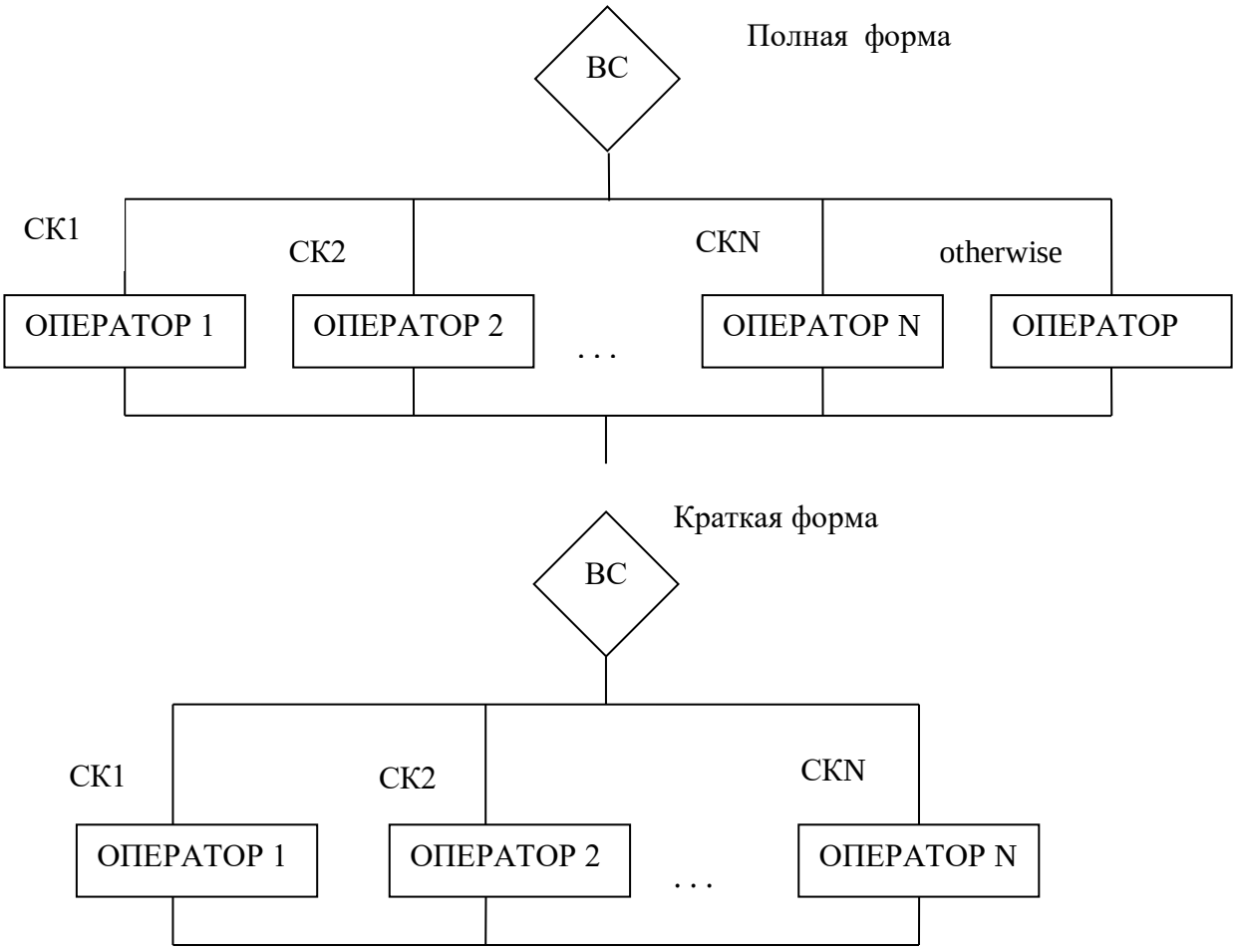


Рис.1.5. Графические схемы алгоритма выполнения оператора выбора

Порядок выполнения оператора case следующий.

Полная форма

- Вычисляется выражение-селектор
- Его значение сравнивается последовательно с константами выбора
- Если они равны, то выполняется оператор, стоящий после данной константы выбора
- 4.Если ни одна из констант выбора не равна значению выражения-селектора, то выполняется оператор, стоящий после слова otherwise.

Краткая форма

- Вычисляется выражение-селектор
- Его значение сравнивается последовательно с константами выбора
- Если они равны, то выполняется оператор, стоящий после данной константы выбора

- Если ни одна из констант выбора не равна значению выражения-селектора, то выполняется оператор, стоящий после оператора otherwise

Пример правильной записи оператора case приведен ниже.

```
switch n
case 5,    disp (‘Отличник’),
case {4,3}, disp (‘Неплохо’),
 case 2,   disp (‘Нужно подтянуться’),
 case1:   disp (‘Все пропало’)
 otherwise disp (‘Неверная оценка’)
end
```

1.3.3 Примеры программ разветвляющихся алгоритмов

Задача 2

Условие задачи Вычислить значение характеристики амортизатора передней подвески автомобиля по формуле:

$$S1 = \begin{cases} 2 \cdot \omega \cdot \phi 1 \cdot v & \text{ïðè} \quad v \le 0 \\ 2 \cdot \omega \cdot \phi 2 \cdot v & \text{ïðè} \quad v > 0 \end{cases}$$

$$\omega = \sqrt{9.8 \cdot \frac{k}{l}}$$

ω, φ1, φ2 – параметры амортизатора;
v – скорость деформации упругого элемента;
k – показатель политропы;
l – идеализированная высота столба газа.

Алгоритмический анализ задачи

Исходные данные:
φ1, φ2 – параметры амортизатора;
v – скорость деформации упругого элемента;
k – показатель политропы;
l – идеализированная высота столба газа.

Результат:
S1 - характеристика амортизатора.

Промежуточное данное:
ω- параметр амортизатора.

Графическая схема алгоритма решения задачи

Графическая схема алгоритма решения задачи приведена на рисунке 1.6.

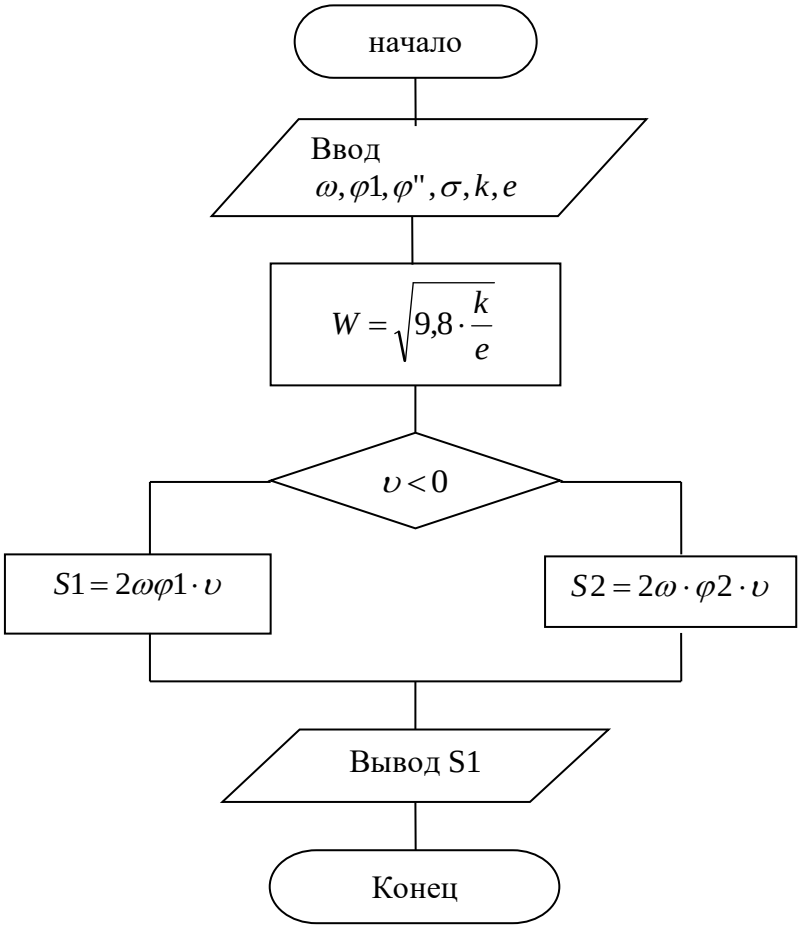


Рис.1.5. Графическая схема разветвляющегося алгоритма

Текст программы

```
v=input('Задайте скорость деформации упругого элемента:   ');
k=input('Задайте показатель политропы:   ');
l=input('Задайте высоту столба газа:  ');
f1=input('Задайте параметр амортизатора f1');
f2=input('Задайте параметр амортизатора f2');
w = sqrt(9.8*k/l);
if v <= 0,
   S1 = 2*w*f1*v;
     else S1 = 2*w*f2*v;
     end;
disp ('Характеристика амортизатора подвески:  '),
disp(S1)
```

Тестовый пример:

Тест 1	Тест 2
Задайте скорость деформации упругого элемента: -2 **Задайте показатель политропы: 1.25** **Задайте высоту столба газа: 0.3** **Задайте параметр амортизатора f1 1.2** **Задайте параметр амортизатора f2 2.1** **Характеристика амортизатора подвески:** **-30.6725**	**Задайте скорость деформации упругого элемента: 2** **Задайте показатель политропы: 1.25** **Задайте высоту столба газа: 0.3** **Задайте параметр амортизатора f1 1.2** **Задайте параметр амортизатора f2 2.1** **Характеристика амортизатора подвески:** **53.6768**

Задача 3

Условие задачи Подобрать значение крутящего момента Т для муфты, если задан ее диаметр и известна зависимость:

Диаметр	Крутящий момент
25…28	500
29…31	1000
39	1600

Алгоритмический анализ задачи

Исходные данные:

d – диаметр муфты;

Результат:

T - крутящий момент.

Графическая схема алгоритма решения задачи

Графическая схема алгоритма решения задачи приведена на рисунке 1.7.

Текст программы

```
d=input('задайте диаметр');
switch d
   case {25,26,27,28}, km=500;
   case {29,30,31}, km=1000;
   case 39, km=1600;
otherwise, km=0;
```

```
disp('неверно задан диаметр'),
end;
disp (['крутящий момент=', km]),
disp (['при d=' ,d])
```

Тестовый пример:

Тест 1	Тест 2
задайте диаметр 26	**задайте диаметр** 40
крутящий момент= 500	**неверно задан диаметр**
при d=26	

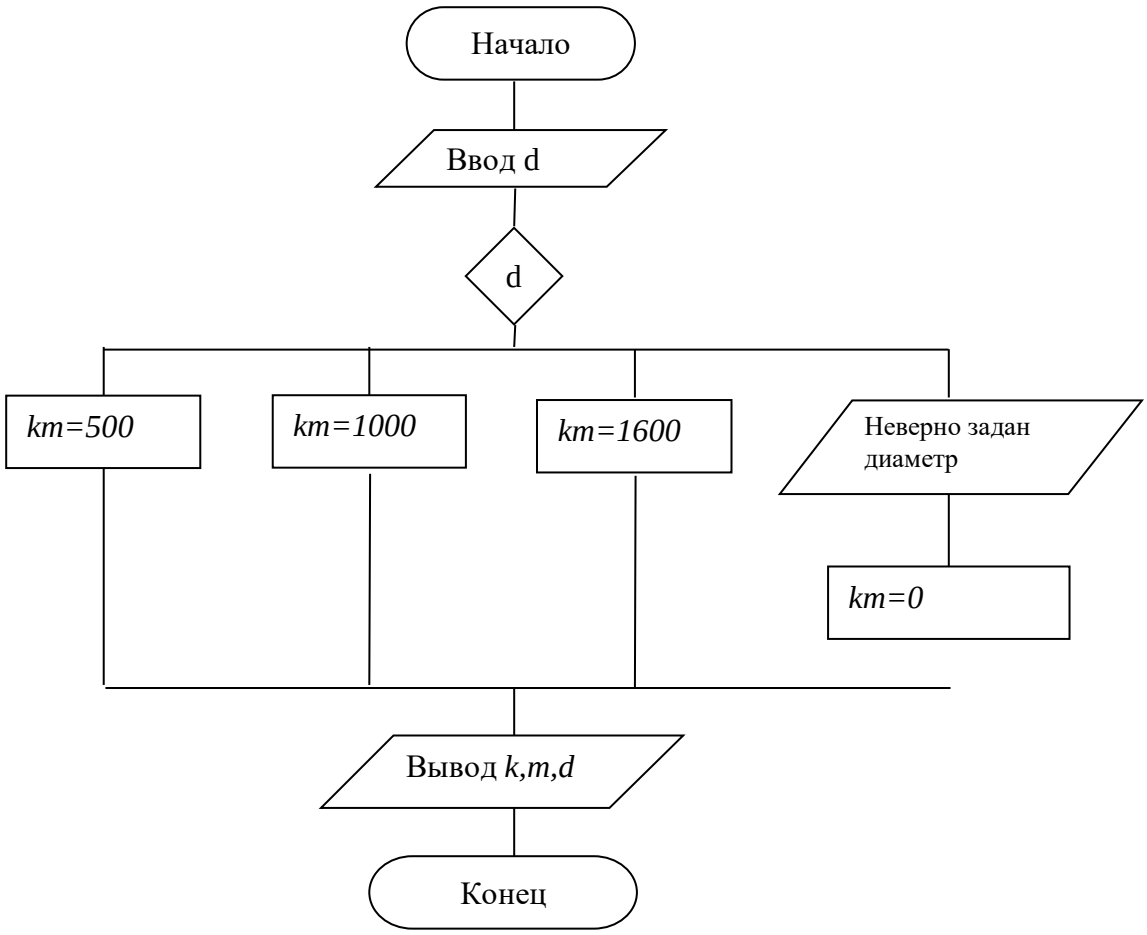


Рис.1.7. Графическая схема алгоритма

1.4 Программирование циклических алгоритмов

1.4.1 Назначение и классификация операторов цикла

Операторы цикла предназначены для программирования циклических алгоритмов, они изменяют естественный ход выполнения программы и относятся к операторам управления. Операторы являются

взаимозаменяемыми, выбор для применения того или иного оператора зависит от программиста.

Операторы цикла в Matlab можно классифицировать следующим образом:

- Оператор цикла с параметрами for
- Оператор цикла с предусловием while

При классификации циклических алгоритмов отмечалось, что цикл может управляться явно выраженной переменной цикла, имеющей параметры, либо цикл не содержит переменной цикла и управляется иным способом. Операторы цикла можно разделить по тому же принципу:

- для программирования циклов с переменной цикла и параметрами;
- для программирования циклов без явно выраженной переменной цикла.

1.4.2 Оператор цикла с параметрами

Оператор цикла **for** предназначен для программирования циклических алгоритмов, когда переменная цикла явно выражена и изменяется от начального значения до конечного значения с постоянным шагом.

Общий вид оператора цикла for (1 форма)

```
for   x= xn : dx : xk
                ОПЕРАТОР,
                end
```

Общий вид оператора цикла for (2 форма):

```
for   x= xn : xk
                ОПЕРАТОР,
                end
```

Здесь :

x – переменная цикла,
xn – начальное значение переменной цикла,
dx – шаг изменения переменной цикла,
xk – конечное значение переменной цикла,
ОПЕРАТОР – любой оператор или группа операторов рабочей части цикла.

Во 2 форме оператора for шаг изменения переменной цикла равен 1.

Порядок выполнения оператора цикла(1 форма):

- Проверяется условие xn<=xk..
- Если условие не выполняется, то оператор цикла прекращает свою работу, и выполняется следующий за ним оператор программы.
- Если условие выполняется, то переменной цикла присваивается ее начальное значение x:=xn.

- Выполняется оператор рабочей части цикла.
- До тех пор, пока x< = xk переменная цикла увеличивается на шаг dx и выполняется рабочая часть цикла.

Порядок выполнения оператора цикла(2 форма) тот же, только переменная цикла увеличивается на шаг, равный 1.

Параметры цикла могут быть константами, переменными или выражениями. Если параметры цикла заданы выражениями, все переменные, входящие в состав выражений должны быть определены заранее. Если параметры цикла заданы некорректно, то оператор цикла не выполняется ни разу. Корректное задание параметров цикла:

xn<=xk при dx>0,

xn>=xk при dx<0.

Количество повторений цикла вычисляется по формуле:

|xk – xn|/dx +1.

Примеры правильной записи оператора for

Фрагмент прграммы	Командное окно
for k=1 : 5 **disp(k), end**	**1** **2** **3** **4** **5**

Вид экрана после выполнения этого оператора будет следующий:

Фрагмент прграммы	Командное окно
for s=10 :-1: 6 **disp(s), end**	**10** **9** **8** **7** **6**

```
c=8; d=11;
for i=c +d : 50
        a=i+5
        b=a+i
  end
```

1.4.3 Оператор цикла с предусловием

Оператор while предназначен для программирования любых циклов, где проверка условия повторения цикла выполняется перед выполнением рабочей части цикла.

Общий вид:

while ЛВ,
ОПЕРАТОР,
end

где

ЛВ – логическое выражение,

ОПЕРАТОР – любой оператор или группа операторов рабочей части цикла.

Порядок выполнения оператора следующий:

- Проверяется истинность логического выражения.
- До тех пор, пока оно истинно, выполняется оператор рабочей части цикла.
- Если логическое выражение стало ложным, то выполняется следующий за оператором цикла оператор программы.

Графическая схема алгоритма выполнения оператора **while** приведена на рисунке 1.8.

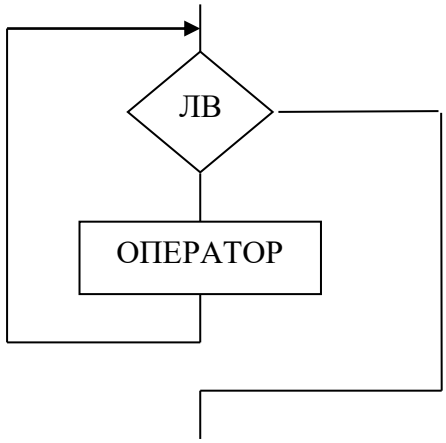


Рис.1.8. Графическая схема алгоритма выполнения оператора While

Рекомендации по программированию и использованием этого оператора цикла приведены ниже.

Если переменная цикла явно выражена, то:

- нужно присвоить ей начальное значение до оператора цикла;
- нужно каким-либо образом изменять ее в рабочей части цикла;
- она должна использоваться в логическом выражении условия повторения цикла.

Если логическое выражение в заголовке цикла while заведомо ложно, то цикл может не выполниться ни разу.

Примеры правильной записи оператора цикла while приведены ниже.

Вывести на экран дисплея четные целые числа от 2 до 12.

```
k=2;
while k<=20
      disp(k)
```

```
    k=k+2;
end
```

Выводить на экран дисплея сообщение «Нажмите клавишу 5» до тех пор, пока эта клавиша не будет нажата.

```
p=0;
while p ~= 5
    p=input(‘ Нажмите 5’);
end;
```

1.4.4 Пример программы циклического алгоритма

Задача 4

Условие задачи. Определить множество значений координаты Y ведущего звена кривошипа при равномерном вращении по формуле YA(t)=r·sin(ω·t) где r-длина кривошипа, ω-угол поворота, t- время вращения, которое изменяется от начального значения tn до конечного tk с определенным шагом dt.

Исходные данные:

r-длина кривошипа;

ω-угол поворота;

tn – начальное значение аргумента;

tk – конечное значение аргумента;

xt – шаг изменения аргумента.

Результат:

YA - значение координаты Y ведущего звена кривошипа в текущий момент времени.

Промежуточное данное:

t - текущее значение времени.

Графическая схема алгоритма решения задачи

Графическая схема алгоритма решения задачи приведена на рисунке 1.9.

Текст программы

```
r=input('задайте радиус кривошипа');
ω =input('задайте угол поворота кривошипа');
tn=input('задайте tn');
tk=input('задайте tk');
dt=input('задайте dt');
for t=tn:dt:tk,
   YA=r*sin(w*t);
   disp('    t        YA')
   disp([t,YA])
```

```
    disp('--------------------')
end
```

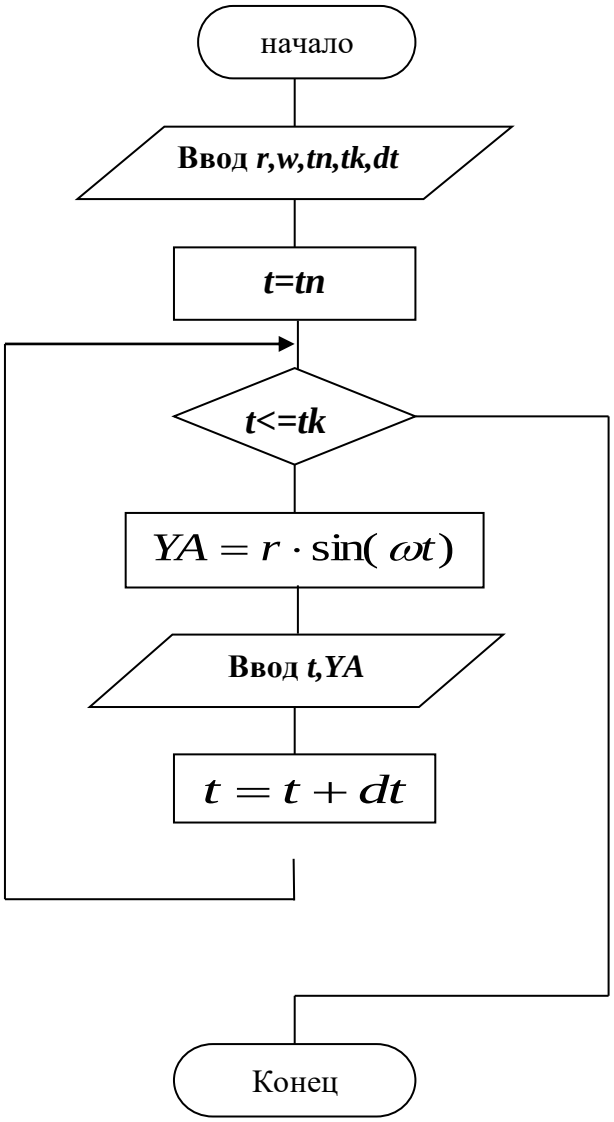


Рис.1.9. Графическая схема циклического алгоритма

Тестовый пример:

Ввод исходных данных	Фрагмент результата
задайте радиус кривошипа 0.5 **задайте угол поворота кривошипа 1.3** **задайте tn 1** **задайте tk 5** **задайте dt 0.5**	**t YA** **1.0000 0.4818** **----------** **t YA** **1.5000 0.4645** **----------** **t YA** **2.0000 0.2578** **----------** **t YA** **2.5000 -0.0541** **----------**

4 РАЗРАБОТКА ГРАФИЧЕСКОГО ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

4.1 Интегрированная среда разработки графического пользовательского интерфейса

Разработку графического интерфейса в Matlab можно выполнять с помощью специального набора команд и функций, а можно использовать специализированное интерактивное средство **GUIDE** (Graphical User Interface Development Environment). Второй подход очень нагляден, удобен и не требует высокой квалификации разработчика. Поэтому в данном практическом пособии рассматриваются возможности применения подсистемы **GUIDE** для программирования графического интерфейса пользователя.

Запуск подсистемы **GUIDE** осуществляется по команде **guide,** введенной в командном окне системы Matlab, или по команде **File – New – GUI** главного меню. При появлении окна ***GUIDE Quick Start*** нужно выбрать один из предлагаемых шаблонов интерфейса или пустое графическое окно ***Blank GUI*** (рекомендуется).

В результате на экран будет выведено окно, показанное на рисунке 4.1, которое содержит меню подсистемы, панель инструментов, палитру графических элементов и редактируемое поле (окно-заготовку) для размещения элементов интерфейса.

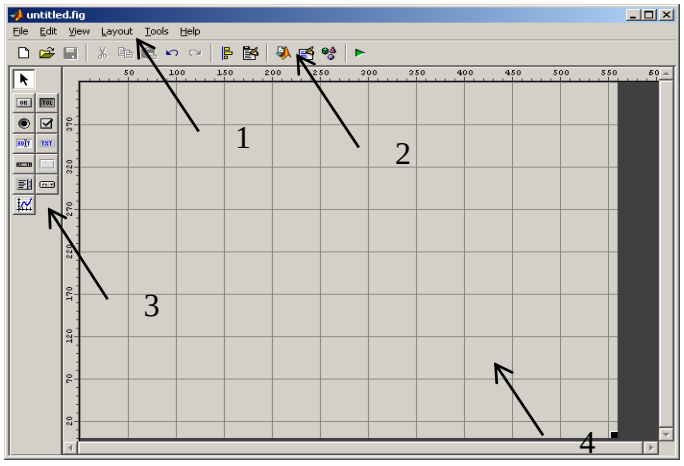


Рисунок 4.1 – Окно подсистемы GUIDE:
1- меню; 2 – панель инструментов; 3 – палитра графических элементов; 4 – редактируемое поле.

Все элементы интерфейса на палитре 3 снабжены наглядными рисунками на соответствующих кнопках, а также всплывающими подсказками. С помощью мыши эти элементы можно «перетаскивать» в создаваемое собственное графическое окно 4. Линейки по краям

проектируемого графического окна и нанесенная на редактируемое поле сетка помогают точно располагать элементы интерфейса и выравнивать их друг относительно друга. Если линейка или сетка не видны, то их можно вывести с помощью команды меню **Tools-Grid and Rulers…**

Когда элемент управления уже размещен в графическом окне, можно изменить его свойства с помощью редактора свойств **Property Inspector.** Окно редактора свойств открывается двойным щелчком мышью по элементу. Здесь расположен список всех свойств данного графического элемента и показаны все значения этих свойств по умолчанию, которые при необходимости можно откорректировать.

Сохранить текущее состояние работы можно, выполнив команду меню **Save** или щелкнув по кнопке **Save** на панели инструментов. При первом сохранении необходимо задать имя файла. Обычно при этом создаются сразу два файла с одинаковым именем и с расширениями **.fig** и **.m.**

Открыть созданный ранее с помощью **GUIDE** проект можно, выполнив команду главного меню системы Matlab **File – New – GUI**. При появлении окна ***GUIDE Quick Start*** нужно выбрать закладку ***Open Existing GUI***, а затем выбрать нужный файл. То же самое можно сделать с помощью команды **Open** меню подсистемы **GUIDE.**

Чтобы **запустить на выполнение** созданный проект, нужно выполнить команду **Tools-Run** меню подсистемы **GUIDE** или щелкнув по кнопке  на панели инструментов.

4.2 Основные элементы графического интерфейса и их свойства

В системе Matlab имеются следующие основные типы элементов для проектирования графического интерфейса:

 кнопка вызова ***push button***;

 кнопка переключения ***toggle button***;

 кнопка выбора отклика ***radio button***;

 окно контроля ***check box***;

 окно редактируемого текста ***edit text***;

 окно фиксированного текста ***static text***;

 скользящая шкала ***slider***;

 панель для размещения элементов управления ***frame***;

 ниспадающее меню ***popup menu***;

 графическая область ***axes***.

Все элементы графического интерфейса в системе Matlab характеризуются присущим им набором свойств. Меняя последние, можно модифицировать их внешний вид и поведение. В таблице 4.1 приведены некоторые свойства основных элементов интерфейса.

Таблица 4.1 – Свойства элементов графического интерфейса

Свойство	Описание
Задание стилей и внешнего вида	
BackgroundColor	Цвет фона
ForegroundColor	Цвет текста
SelectionHighlight	Выделение объекта при его выборе
String	Строка символов, используемая для вывода текста в поле элемента
Visible	Видимость/ невидимость объекта
FontAngle	Наклон шрифта
FontName	Название шрифта
FontSize	Размер шрифта
FontWeight	Шрифт по толщине (нормальный, жирный, полужирный)
Horizontal Alignment	Выравнивание текста в поле элемента
Общая информация об элементе	
Enable	Элемент доступен/ недоступен
Style	Тип элемента
Tag	Метка (имя) элемента
TooltipString	Строка всплывающей подсказки
Управление выполнением процедур ответного вызова	
Callback	Имя процедуры ответного вызова
CreateFcn	Процедура ответного вызова при создании элемента
DeleteFcn	Процедура ответного вызова при удалении элемента

Каждому элементу графического интерфейса соответствует свой **дескриптор** (число, уникально идентифицирующее конкретный объект).

Для определения дескриптора элемента существуют следующие специальные функции:

Gcf – возвращает дескриптор текущего графического окна;

Gcbf - возвращает дескриптор графического окна, элемент которого выполняет процедуру ответного вызова;

Findobj - позволяет определить дескриптор элемента по заданному значению пары *имя/ значение* некоторого свойства. Синтаксис функции следующий:

Findobj(<дескриптор_графического_окна>, '<имя_свойства>', <значение_свойства>).

Например, функция **findobj(gcf,'Tag','edit1')** возвращает дескриптор элемента интерфейса, размещенного в текущем окне и имеющего имя edit1.

Для определения значения свойства элемента используется функция

Get(<дескриптор>,' <имя_свойства>').

Например, функция **Get(gcf, 'BackgroundColor')** возвращает цвет фона текущего графического окна.

Для придания свойству элемента определенного значения используется команда

Set(<дескриптор>,' <имя_свойства>', <значение_свойства>).

Например, после выполнения набора команд

h=findobj(gcf, 'Tag','Text1')

set(h,'String','С новым годом !')

в поле элемента с именем **Text1** активного графического окна будет выведен текст **«С новым годом !».**

4.3 Особенности реализации графического интерфейса с помощью подсистемы GUIDE

4.3.1 Ввод и вывод данных с помощью элементов интерфейса

Для ввода значения переменных рекомендуется использовать такой элемент управления как окно редактируемого текста ***edit text***. Этот элемент при выполнении программы ведет себя как обычный однострочный текстовый редактор, при этом текст, набранный в окне ввода, помещается в свойство **String**.

Чтобы набранное в окне ввода значение присвоить переменной, нужно с помощью функции **findobj** определить дескриптор элемента ***edit text***, а затем с помощью функции **Get** присвоить переменной значение свойства **String** . При необходимости следует также применить одну из функций преобразования типов.

Ниже приведен фрагмент m-файла, в котором вводится значение числовой переменной *x*.

```
h1=findobj(gcf,'tag','edit1'); %  Определение дескриптора элемента с меткой
                               %  edit1
x=str2num(get(h1,'string')); %   Функц. get возвращает значение свойства
                               %  string  элемента с дескриптором h1,
                               %  a str2num преобразует строку в число.
```

Для вывода данных рекомендуется использовать такой элемент управления как окно фиксированного текста ***static text***. Текст, отображаемый в поле элемента, помещается в свойство **String**.

Чтобы вывести значение переменной в поле элемента ***static text*** в графическом окне, нужно с помощью функции **findobj** определить дескриптор элемента ***static text***, а затем с помощью функции **Set** в свойство **String** поместить значение переменной. При необходимости следует также применить одну из функций преобразования типов.

В приведенном ниже фрагменте m-файла осуществляется вывод значений переменных *x* и *y*.

```
x=3.05;   y=sin(x)+cos(x);
h=findobj(gcf,'tag','text1');
set(h,'string', strcat('x=', num2str(x), ' y=', num2str(y)))
```

Здесь функция **num2str** преобразует числа в строки, а функция **strcat** осуществляет конкатенацию строк.

Результатом выполнения указанных команд будет строка в поле элемента

x=3.05 y= -0.90434

4.3.2 Построение графиков с помощью элемента axes

Для того чтобы в графическое окно вывести графики функций, необходимо разместить в редактируемом поле среды GUIDE графическую область **axes,** после чего с помощью редактора свойств **Property Inspector,** если это нужно**,** изменить значения некоторых свойств элемента. Затем в m-файле ответного вызова следует использовать одну из команд построения графика.

В таблице 4.2 приводятся некоторые дополнительные свойства объекта **axes,** отсутствующие в таблице 4.1.

Таблица 4.2 – Дополнительные свойства элемента **axes**

Свойство	Описание
Box	Контур координатных осей: отображать (on), не отображать (off).
GridLineStyle	Типы линий для сетки
LineStyleOder	Типы линий для вывода нескольких графиков
Xgrid, Ygrid, ZGrid	Включить/ выключить координатную сетку по соответствующей оси
Title	Заголовок
XLabel, YLabel, ZLabel	Метки осей
Xcolor, Ycolor, ZColor	Цвета линий координатных осей, маркеров, меток и сетки
Xscale, Yscale, ZScale	Масштаб по осям: линейный(linear),

4.3.3 Порядок разработки графического интерфейса

Для создания графического пользовательского интерфейса рекомендуется выполнить следующие действия:

- Создать отдельную папку для разрабатываемого проекта;
- Запустить утилиту **GUIDE** системы Matlab**;**
- Разместить в графическом окне необходимые элементы интерфейса;
- С помощью редактора **Property Inspector** установить нужные значения свойств для каждого элемента;
- Создать m-файлы, содержащие программы ответных вызовов, которые будут выполняться при активизации элементов (например, при нажатии на кнопку ***push button***) и сохранить их в созданной папке;
- В свойство **Callback** элементов, для которых создавались программы обратных вызовов, записать имена соответствующих m-файлов;
- Сохранить полученный проект в созданной папке;
- Запустить созданное приложение на выполнение и проверить правильность его работы.

Для примера рассмотрим реализацию задачи построения графика функции sin(x) на интервале изменения аргумента от $x_{нач.}$ до $x_{кон.}$ с шагом Δx.

На рисунке 4.2 приводится вид редактируемого поля GUIDE с размещенными на нем элементами интерфейса, а на рисунке 4.3 - графическое окно в процессе выполнения спроектированного приложения.

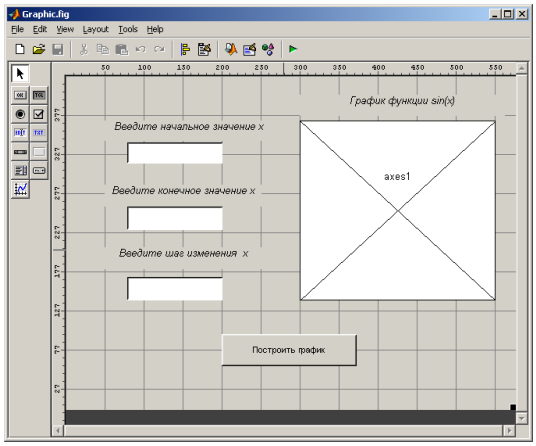


Рисунок 4.2

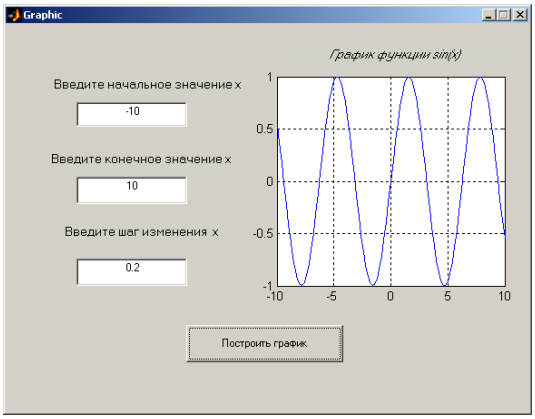


Рисунок 4.3

Чтобы после ввода исходных данных по нажатию кнопки **Построить график** в графическую область окна выводился график функции, был создан m-файл ответного вызова, имя которого было записано в свойство кнопки **Callback**. Текст этого файла приведен ниже.

```
h1=findobj(gcf,'tag','edit1'); xn=str2num(get(h1,'string')); % ввод исходных
h2=findobj(gcf,'tag','edit2'); xk=str2num(get(h2,'string')); % данных
h3=findobj(gcf,'tag','edit3'); dx=str2num(get(h3,'string'));
x=xn:dx:xk;  y=sin(x);  plot(x,y); grid on                    % построение графика
```

3 Приемы моделирования в Simulink

3.1 Общие сведения о пакете Simulink

Пакет Simulink является приложением к системе MATLAB. При моделировании с использованием Simulink реализуется принцип визуального программирования, в соответствии с которым, пользователь из библиотеки стандартных блоков создает модель устройства и осуществляет расчеты.

Simulink является автономным пакетом MATLAB. Модели, созданные в Simulink, построены на основе внутреннего языка MATLAB, которые могут быть откорректированы в MATLAB, и данные из MATLAB могут быть переданы для обработки в Simulink. Часть входящих в состав пакетов имеет инструменты, встраиваемые в Simulink (например, LTI-Viewer приложения Control System Toolbox – пакета для разработки систем управления). Имеются также дополнительные библиотеки блоков для разных областей применения (например, Power System Blockset – моделирование электротехнических устройств, Digital Signal Processing Blockset – набор блоков для разработки цифровых устройств и т.д.).

При работе с Simulink пользователь имеет возможность модернизировать библиотечные блоки, создавать свои собственные, а также составлять новые библиотеки блоков.

При моделировании пользователь может выбирать метод решения дифференциальных уравнений, а также способ изменения модельного времени (с фиксированным или переменным шагом). В ходе моделирования имеется возможность следить за процессами, происходящими в системе. Для этого используются специальные устройства наблюдения, входящие в состав библиотеки Simulink. Результаты моделирования могут быть представлены в виде графиков или таблиц.

Преимущество Simulink заключается также в том, что он позволяет пополнять библиотеки блоков с помощью подпрограмм написанных как на языке MATLAB, так и на языках C + +, Fortran и Ada.

3.2 Интерфейс пакета Simulink

Для запуска программы необходимо предварительно запустить пакет MATLAB. После открытия основного окна программы MATLAB нужно запустить программу Simulink. Это можно сделать одним из трех способов:

- Нажать кнопку (Simulink)на панели инструментов командного окна MATLAB.
- В командной строке главного окна MATLAB напечатать Simulink и нажать клавишу Enter на клавиатуре.
- Выполнить команду Open… в меню File и открыть файл модели (mdl - файл).

Последний вариант удобно использовать для запуска уже готовой и отлаженной модели, когда требуется лишь провести расчеты и не нужно добавлять новые блоки в модель. Использование первого и второго способов приводит к открытию окна обозревателя разделов библиотеки Simulink (рисунок 4.1).

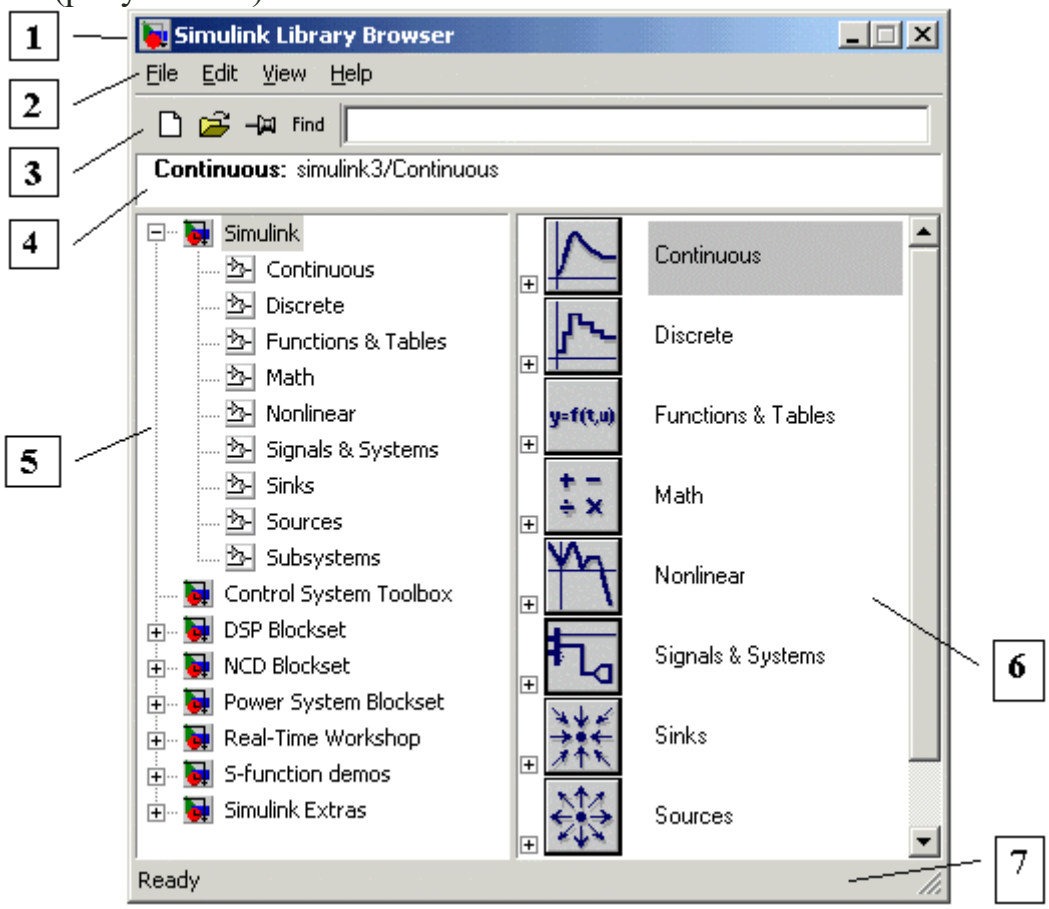


Рисунок 4.1. Окно обозревателя разделов библиотеки Simulink

Окно обозревателя библиотеки блоков содержит следующие элементы (рисунок 4.1):

1 - Заголовок, с названием окна – Simulink Library Browser.

2 - Меню, с командами File, Edit, View, Help.

3 - Панель инструментов, с ярлыками наиболее часто используемых команд.

4

- Окно комментария для вывода поясняющего сообщения о выбранном блоке.

- Список разделов библиотеки, реализованный в виде дерева.

5 - Окно содержимого раздела библиотеки (список вложенных разделов библиотеки или блоков)

6 - Строка состояния, содержащая подсказку по выполняемому действию.

7 На рисунке 4.1 выделена основная библиотека Simulink (в левой части окна) и показаны ее разделы (в правой части окна).

Библиотека Simulink содержит следующие основные разделы:

Continuous – линейные блоки.
Discrete – дискретные блоки.
Functions & Tables – функции и таблицы.
Math – блоки математических операций.
Nonlinear – нелинейные блоки.
Signals & Systems – сигналы и системы.
Sinks - регистрирующие устройства.
Sources — источники сигналов и воздействий.
Subsystems – блоки подсистем.

При выборе соответствующего раздела библиотеки в правой части окна отображается его содержимое (рисунок 4.2).

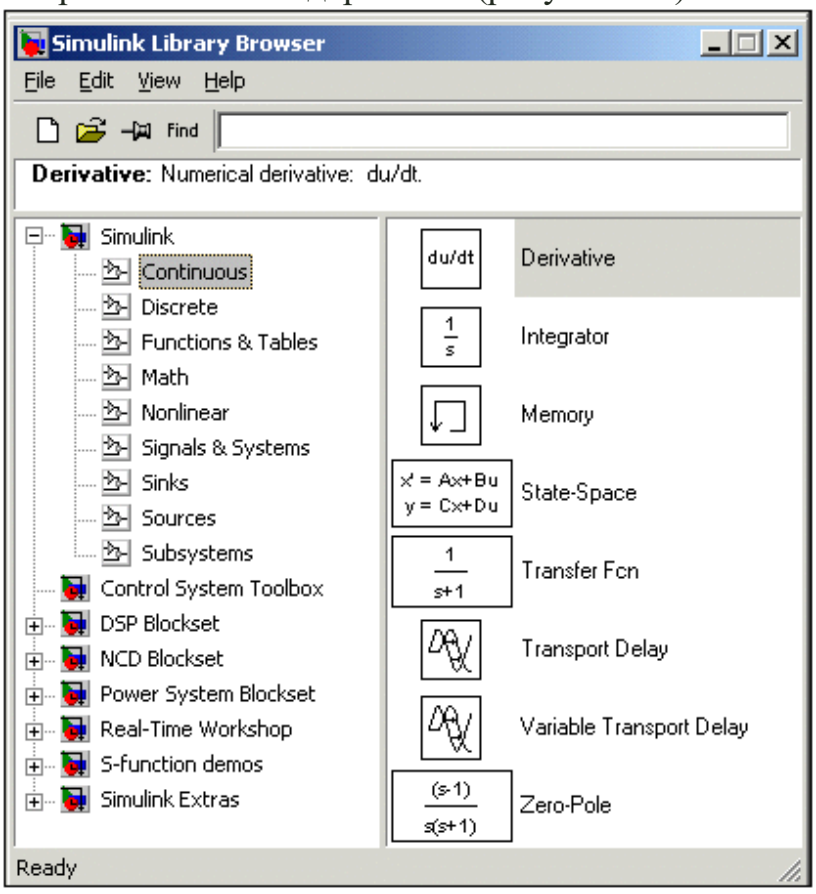


Рисунок 4.2. Окно обозревателя с набором блоков раздела библиотеки Continuous

3.3 Создание и редактирование модели в Simulink

Для создания модели в среде SIMULINK необходимо последовательно выполнить ряд действий.

Создание файла модели. Создать новый файл модели с помощью команды File/New/Model, или используя кнопку на панели инструментов.

Выбор блоков модели. Расположить блоки в окне модели. Для этого необходимо открыть соответствующий раздел библиотеки (Например, Sources - Источники). Далее, указав курсором на требуемый блок и нажав на левую клавишу “мыши” - “перетащить” блок в созданное окно. *Клавишу мыши нужно держать нажатой*. На рисунке 4.3 показано окно модели, содержащее блоки.

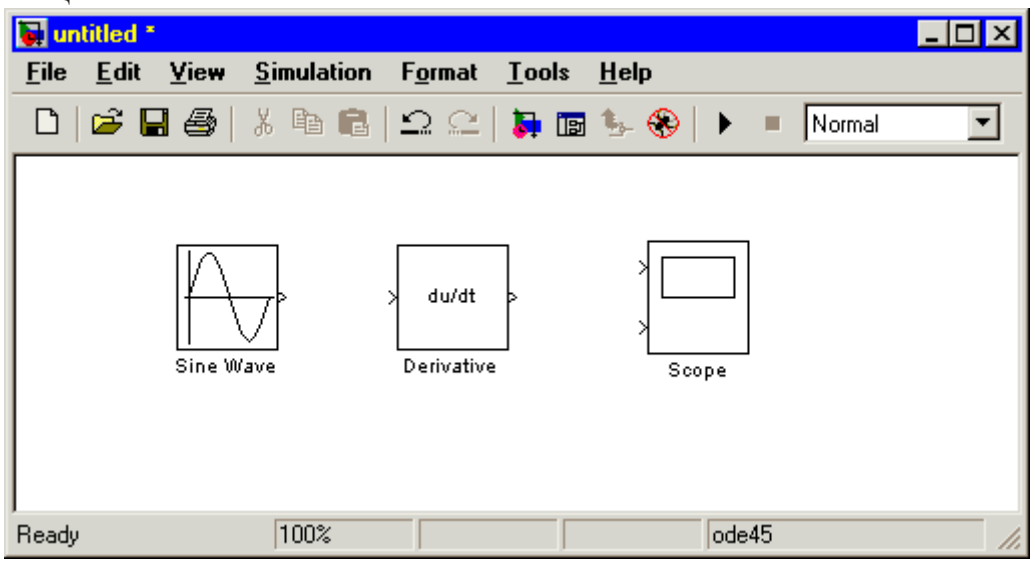


Рисунок 4.3. Окно модели, содержащее блоки

Редактирование блоков модели. Для удаления блока необходимо выбрать блок (указать курсором на его изображение и нажать левую клавишу “мыши”), а затем нажать клавишу Delete на клавиатуре.

Для изменения размеров блока требуется выбрать блок, установить курсор в один из углов блока и, нажав левую клавишу “мыши”, изменить размер блока (курсор при этом превратится в двухстороннюю стрелку).

Для редактирования блока, нужно изменить параметры блока, установленные пакетом “по умолчанию”. Для этого необходимо дважды щелкнуть левой клавишей “мыши”, указав курсором на изображение блока. Откроется окно редактирования параметров данного блока. При задании численных параметров следует иметь в виду, что в качестве

десятичного разделителя должна использоваться точка, а не запятая. После внесения изменений нужно закрыть окно кнопкой OK. На рисунке 4.4 в качестве примера показаны блок, моделирующий синусоидальный сигнал с заданной частотой, амплитудой, фазой и смещением, и окно редактирования параметров данного блока.

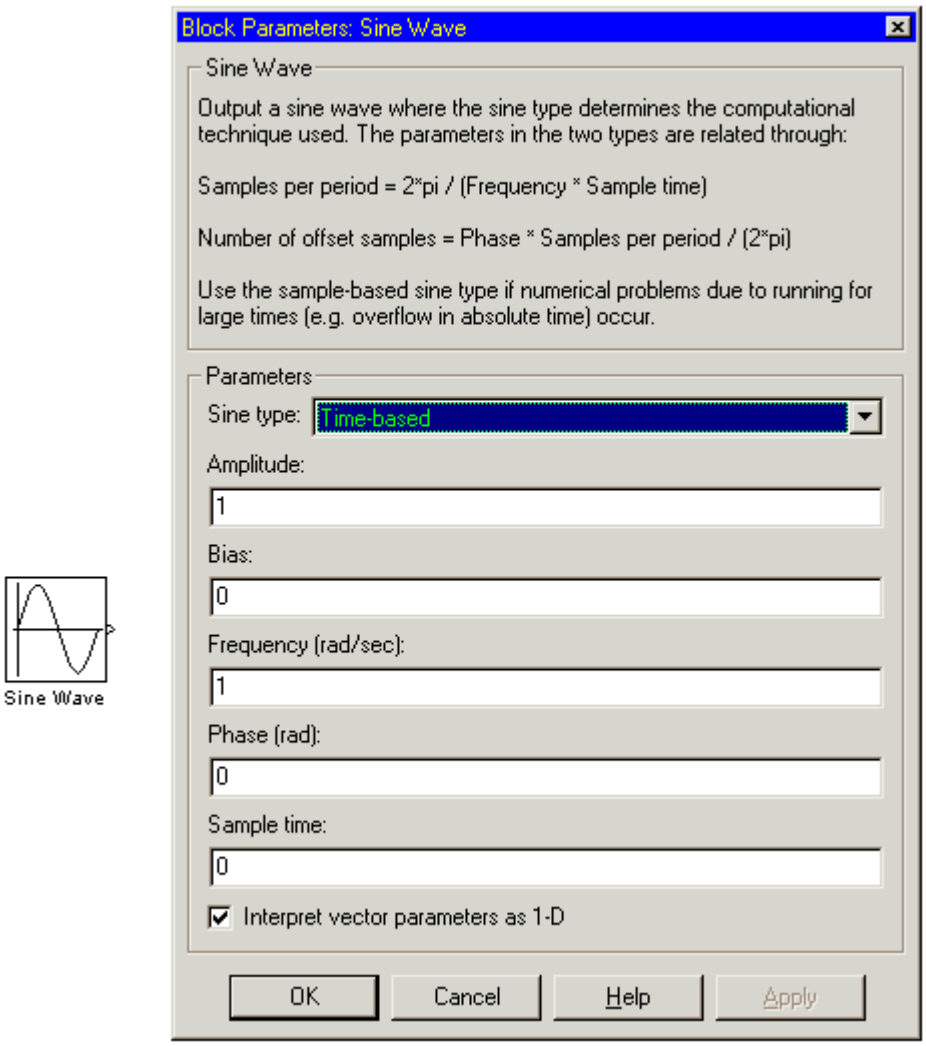


Рисунок 4.4. Блок, моделирующий синусоидальный сигнал с заданной частотой, амплитудой, фазой и смещением, и окно редактирования параметров блока

Соединение блоков модели. После установки на схеме всех блоков из требуемых библиотек нужно выполнить соединение элементов схемы. Для соединения блоков необходимо указать курсором на “выход” блока, а затем, нажать и, не отпуская левую клавишу “мыши”, провести линию к входу другого блока. После чего отпустить клавишу. В случае правильного соединения изображение стрелки на входе блока изменяет цвет. Для создания точки разветвления в соединительной линии нужно подвести курсор к предполагаемому узлу и, нажав *правую* клавишу “мыши”, протянуть линию. Для удаления линии требуется выбрать линию (так же, как это выполняется для блока), а затем нажать клавишу Delete на

клавиатуре. Схема модели, в которой выполнены соединения между блоками, показана на рисунке 4.5.

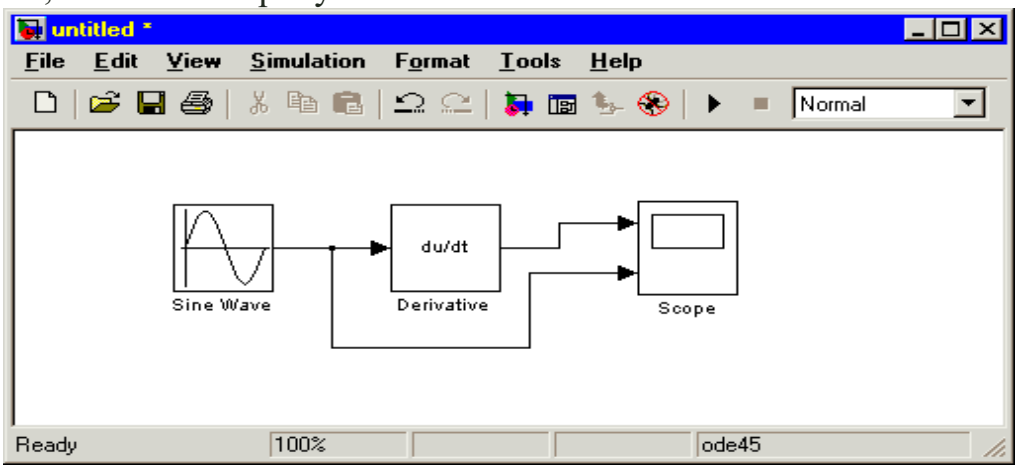
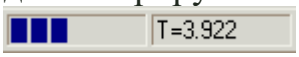


Рисунок 4.5. Схема модели

3.4 Запуск модели, анализ результатов

Запуск модели на выполнение осуществляется нажатием на панели инструментов кнопки (Start simulation). При этом в строке состояния демонстрируется текущее время расчета и процентное состояние расчета



Получение результатов модели сводится к записи в рабочую область MATLAB - Workspase данных в численном виде, или отображение исходных данных и результата в виде графика рисунок 4.6.

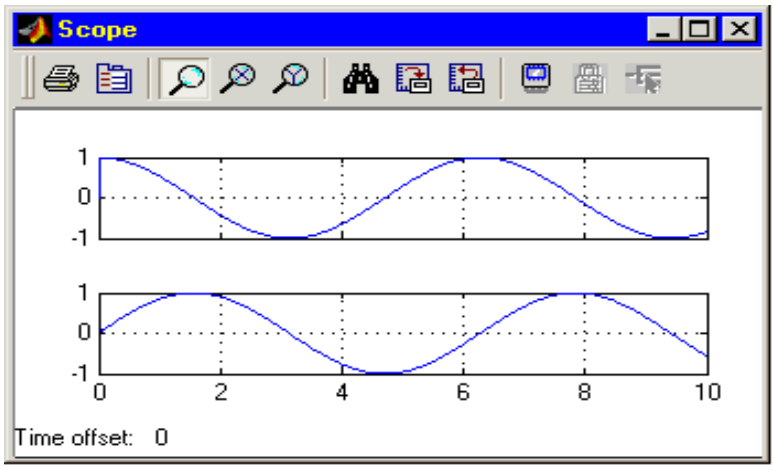


Рисунок 3.6 – Графическое отображение исходных данных и результата

Для сохранения расчетной схемы необходимо выбрать пункт меню File/Save As... в окне схемы и указав папку и имя файла. Следует иметь в виду, что имя файла не должно превышать 32 символов, должно начинаться с буквы и не может содержать символы кириллицы и спецсимволы. Это же требование относится и к пути файла (к тем папкам,

в которых сохраняется файл). При последующем редактировании схемы можно пользоваться пунктом меню Fille/Save. При повторных запусках программы SIMULINK загрузка схемы осуществляется с помощью меню File/Open... в окне обозревателя библиотеки или из основного окна MATLAB.

Список использованных источников

1 Дьяконов В. MATLAB 6: учебный курс.- СПб.: «Питер», 2001.

2 Говорухин В., Цибулин В. Компьютер в математическом исследовании.- СПб.: «Питер», 2001.

3 Мартынов Н.Н. Введение в MATLAB 6.- М.: КУДИЦ-ОБРАЗ, 2002.-352с.

СОДЕРЖАНИЕ