Лабораторная работа № 9
Основы программирования в MatLab

Структура программы

Как правило, каждая программа в MatLab представляет собой функцию и начинается с ключевого слова function, за которым через пробел следует ее название. Например,

```
function Lab1
a = 5;
b = 2;
c = a*b;
```

Данная программа заключена в функции с именем Lab1 и вычисляет произведение двух переменных a и b. При сохранении программы в m-файл рекомендуется указывать имя файла, совпадающее с именем функции, т.е. в данном случае – Lab1.

Редактор/отладчик предоставляет как средства редактирования текста m-файла, так и средства пошаговой его отладки. Один из способов вызова редактора – вызов из командной строки MATLAB с помощью команды **edit.**

Редактор, используемый в системе, имеет синтаксическую раскраску, т.е. слово или символ по мере ввода приобретают тот цвет, который соответствует их типу. С помощью пункта меню Tools-Fonts можно настроить такие важные параметры, как используемый шрифт. Это особенно важно для работы с русским текстом, поскольку не все шрифты правильно воспроизводят русский текст. Редактор имеет стандартный набор возможностей (запуск М-файла, расстановка точек останова – breakpoints), так и специальные: переключение в Cell Mode (режим ячеек), публикация html, пункт Evaluate Selection, который позволяет вычислять значение выделенного выражения и помещать результат в консоль, пункт Edit-Paste to Workspace. В режиме ячеек легко отлаживать и читать программу; с той же целью введена пиктограмма Show Functions, позволяющая перескакивать по заголовкам вложенных функций.

Следует отметить, что в одном m-файле можно задавать множество дополнительных функций. Для этого достаточно написать в конце листинга основной программы еще одно ключевое слово function и задать ее имя, например,

```
function Lab1
a = 5;
b = 2;
c = a*b;
out_c(c);                  % вызов функции out_c()

function out_c(arg_c)    % определение функции out_c()
disp(arg_c);
```

Функцию out_c() можно вызывать в основной программе до ее определения.

Дополнительные функции можно оформлять и в отдельных m-файлах. Например, если есть необходимость какую-либо функцию описать в одном m-файле, а вызывать ее в другом, то это можно реализовать следующим образом.

1-й файл (Lab1.m)

```
function Lab1
a = 5;
b = 2;
c = square(a,b);          % вызов функции square()
out_c(c);                  % вызов функции out_c()

function out_c(arg_c)    % определение функции out_c()
disp(arg_c);
```

2-й файл (square.m)

```
function res=square(a, b)
res = a*b;
```

При выполнении функции Lab1 система MatLab вызовет функцию square из файла square.m. Это будет сделано автоматически, т.к. встроенные функции языка MatLab определены также и вызываются из файлов, имена которых, как правило, соответствуют именам вызываемых функций. Обратите также внимание на то, что функция square() не только принимает два аргумента a и b, но и возвращает их произведение с помощью переменной res. Представленный синтаксис следует использовать всякий раз, когда требуется возвратить результат вычислений основной программе.

Структуры в MatLab

Структуры задаются следующим образом:

```
S = struct('field1',VALUES1,'field2',VALUES2,...);
```

где field1 – название первого поля структуры; VALUES1 – переменная первого поля структуры, и т.д.

Приведем пример, в котором использование структуры позволяет эффективно представить данные. Таким примером будет инвентарный перечень книг, в котором для каждой книги необходимо указывать ее наименование, автора и год издания. Причем количество книг может быть разным. Для хранения информации об одной книге будем использовать структуру, которая задается в MatLab с помощью ключевого слова struct следующим образом:

```
S = struct('title','','author','','year',0);
```

В итоге задается структура с тремя полями: title, author и year. Каждое поле имеет свой тип данных и значение.

Для того, чтобы записать в эту структуру конкретные значения используется оператор ‘.’ (точка) для доступа к тому или иному полю структуры:

```
S.title = 'Евгений Онегин';
S.author = 'Пушкин';
S.year = 2000;
```

и таким образом, переменная S хранит информацию о выбранной книге.

Однако по условиям задачи необходимо осуществлять запись не по одной, а по 100 книгам. В этом случае целесообразно использовать вектор структур lib, который можно задать следующим образом:

```
lib(100,1) = struct('title','','author','','year',0);
и записывать информацию о книгах так:
lib(1).title = 'Евгений Онегин';
lib(1).author = 'Пушкин';
lib(1).year = 2000;
```

При работе со структурами полезными являются следующие функции:

isstruct(S) – возвращает истину, если аргумент структура

isfield(S, 'name') – возвращает истину, если имеется такое поле

fieldnames(S) – возвращает массив строк с именами всех полей

которые позволяют программно определить всю необходимую информацию о той или иной структуре и корректно выполнять обработку ее полей.

Ячейки в MatLab

Ячейки также как и структуры могут содержать разные типы данных, объединенные одной переменной, но в отличие от вектора структур, вектор ячеек может менять тип данных в каждом элементе. Таким образом, вектор ячеек является универсальным контейнером – его элементы могут содержать любые типы и структуры данных, с которыми работает MatLab – векторы чисел любой размерности, строки, векторы структур и другие (вложенные) векторы ячеек.

Методы создания вектора ячеек похожи на методы создания вектора структур. Как и в случае структур, векторы ячеек могут быть созданы либо путём последовательного присваивания значений отдельным элементам массива, либо созданы целиком при помощи специальной функции cell(). Однако в любом случае важно различать ячейку (элемент вектора ячеек) и её содержимое. Ячейка – это содержимое плюс некоторая оболочка (служебная структура данных) вокруг этого содержимого, позволяющая хранить в ячейке произвольные типы данных любого размера.

Приведем пример создания вектора ячеек хранения разных типов данных.

```
book = struct('title','Онегин','author','Пушкин','year',2000);
```

```
MyCell(1)={book};
MyCell(2)={'Пушкин'};
MyCell(3)={2000};
```

Здесь задан вектор ячеек MyCell с тремя элементами. Первый элемент соответствует структуре, второй – строке, а третий – числу. В этом и заключается особенность организации данных с помощью ячеек: у каждого элемента свой тип данных.

Для обращения к содержимому той или иной ячейки используются фигурные скобки, внутри которых ставится индекс элемента, с которым предполагается работа:

```
MyCell{1}
```

выведет на экран

```
 title: 'Онегин'
 author: 'Пушкин'
 year: 2000
```

Если же используются круглые скобки, то будет возвращена структура данных вместо отдельных значений, например

```
MyCell(1)
```

выведет

```
[1x1 struct]
```

Для того чтобы задать вектор или матрицу ячеек с пустыми (неопределенными) значениями, используется функция cell() как показано ниже.

```
MyCellArray = cell(2, 2);
```

задается матрица размером 2x2. Данную инициализацию целесообразно выполнять, когда нужно определить большой вектор или матрицу ячеек и в цикле задавать их значения. В этом случае MatLab сразу создает массивы нужных размеров, в результате чего повышается скорость выполнения программ.

В заключении рассмотрим возможность программирования функции с произвольным числом аргументов благодаря использованию ячеек. Для этого в качестве аргумента функции указывается ключевое слово varargin, которое интерпретируется внутри функции как вектор ячеек с переданными аргументами:

```
function len = SumSquare( varargin )
n= length( varargin );
len = 0;
for k = 1 : n
    len = len + varargin{ k }(1)^2 +varargin{ k }(2)^2;
end
```

Данная функция вычисляет сумму квадратов чисел, которые передаются ей следующим образом:

```
SumSquare( [ 1 2], [3 4] )              % ответ 30
SumSquare( [ 1 2], [3 4], [ 5 6] )          % ответ 91
```

Таким образом, благодаря использованию ячеек функции SumSquare() можно передавать произвольное число двумерных векторов.

Самостоятельно: выполните передачу двумерного массива в качестве аргумента функции SumSquare(), заранее размещенного в Workspace:

1) Mas1|2x2|;
2) Mas2|3x3|.

Условные операторы и циклы в MatLab

Условный оператор if

В самом простом случае синтаксис данного оператора if имеет вид:

```
if <выражение>
<операторы>
end
```

В табл. 1 представлены варианты простых логических выражений оператора if.

Таблица 1

Простые логические выражения

if a < b	Истинно, если переменная a меньше переменной b и ложно в противном случае.
if a > b	Истинно, если переменная a больше переменной b и ложно в противном случае.
if a == b	Истинно, если переменная a равна переменной b и ложно в противном случае.
if a <= b	Истинно, если переменная a меньше либо равна переменной b и ложно в противном случае.
if a >= b	Истинно, если переменная a больше либо равна переменной b и ложно в противном случае.
if a ~= b	Истинно, если переменная a не равна переменной b и ложно в противном случае.

Ниже представлен пример реализации функции sign(), которая возвращает +1, если число больше нуля, -1 – если число меньше нуля и 0, если число равно нулю:

```
function my_sign
x = 5;
if x > 0
disp(1);
end
if x < 0
disp(-1);
end
if x == 0
disp(0);
end
```

Анализ приведенного примера показывает, что все эти три условия являются взаимоисключающими, т.е. при срабатывании одного из них нет необходимости проверять другие. Реализация именно такой логики позволит увеличить скорость выполнения программы. Этого можно добиться путем использования конструкции

```
if <выражение>
<операторы1>        % выполняются, если истинно условие
else
<операторы2>        % выполняются, если условие ложно
end
```

Тогда приведенный выше пример можно записать следующим образом:

```
function my_sign
x = 5;
if x > 0
disp(1);
else
     if x < 0
     disp(-1);
     else
     disp(0);
     end
end
```

Приведенный выше пример можно записать в более простой форме, используя еще одну конструкцию оператора if языка MatLab:

```
function my_sign
x = 5;
if x > 0
     disp(1);        % выполняется, если x > 0
```

```
elseif x < 0
     disp(-1);       % выполняется, если x < 0
else
     disp(0);        % выполняется, если x = 0
end
```

Самостоятельно:

1. Измените функцию, добавьте ей аргумент и проверьте правильность выполнения на разном наборе данных.

2. Рассмотрите пример использования составных условий. Пусть требуется проверить попадание переменной x в диапазон от 0 до 2.

3. Выполните проверку на не принадлежность переменной x диапазону от 0 до 2.

4. Переменная x попадает в диапазон от -5 до 5, но не принадлежит диапазону от 0 до 1.

Оператор цикла while

Приведем пример работы цикла while для подсчета суммы ряда $S=\sum_{i=1}^{20} i$:

```
function sum_i
S = 0;                  % начальное значение суммы
i=1;                    % счетчик суммы
while i <= 20           % цикл (работает пока i <= 20)
     S=S+i;             % подсчитывается сумма
     i=i+1;             % увеличивается счетчик на 1
end                     % конец цикла
disp(S);                % отображение суммы 210 на экране
```

Самостоятельно: Подсчитайте сумму ряда $S=\sum_{i=1}^{20} i$, пока $S \le 20$.

Работу любого оператора цикла, в том числе и while, можно принудительно завершить с помощью оператора break.

Самостоятельно: Требуется подсчитать сумму элементов массива a = [1 2 3 4 5 6 7 8 9]; исключая элемент с индексом 5.

Оператор цикла for

Рассмотрим работу данного цикла на примере реализации алгоритма поиска максимального значения элемента в векторе:

```
function search_max
a = [3 6 5 3 6 9 5 3 1 0];
m = a(1);                   % текущее максимальное значение
for i=1:length(a)           % цикл от 1 до конца вектора с
                            % шагом 1 (по умолчанию)
    if m < a(i)                 % если a(i) > m,
         m = a(i);              % то m = a(i)
    end
end                         % конец цикла for
disp(m);
```

В данном примере цикл for задает счетчик i и меняет его значение от 1 до 10 с шагом 1. Обратите внимание, что если величина шага не указывается явно, то он берется по умолчанию равным 1.

В следующем примере рассмотрим реализацию алгоритма смещения элементов вектора вправо, т.е. предпоследний элемент ставится на место последнего, следующий – на место предпоследнего, и т.д. до первого элемента:

```
function queue
a = [3 6 5 3 6 9 5 3 1 0];
disp(a);
```

```
for i=length(a):-1:2                    % цикл от 10 до 2 с шагом -1
     a(i)=a(i-1);                  % смещаем элементы вектора a
end                                % конец цикла for
disp(a);
```

Результат работы программы

```
3  6  5  3  6  9  5  3  1  0
3  3  6  5  3  6  9  5  3  1
```

Рассмотрим работу оператора цикла for на примере моделирования случайной последовательности с законом изменения

$$x_i = r x_{i-1} + \xi_i,$$

где r - коэффициент от -1 до 1; ξ_i - нормальная случайная величина с нулевым математическим ожиданием и дисперсией

$$\sigma_\xi^2 = \sigma_x^2 (1 - r^2),$$

где σ_x^2 - дисперсия моделируемого случайного процесса. При этом первый отсчет x_1 моделируется как нормальная случайная величина с нулевым математическим ожиданием и дисперсией σ_x^2. Программа моделирования имеет следующий вид:

```
function modeling_x
r = 0.95;                          % коэффициент модели
N = 100;                           % число моделируемых точек
ex = 100;                          % дисперсия процесса

et = ex*(1-r^2);                   % дисперсия случайной добавки ξi
x = zeros(N,1);                        % инициализация вектора x
x(1) = sqrt(ex)*randn;             % моделирование 1-го отсчета
for i=2:N                          % цикл от 2 до N
     x(i)=r*x(i-1)+sqrt(et)*randn;          % моделирование СП
end                                % конец цикла
plot(x);                           % отображение СП в виде графика
```

При выполнении данной программы будет показана реализация смоделированной случайной последовательности $\mathbf{x}$.

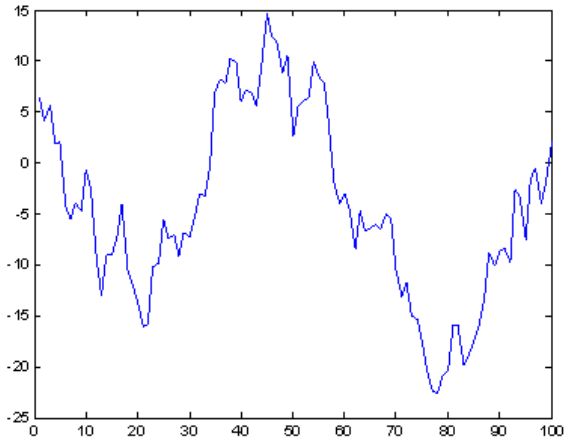

Рис. 1. Результат моделирования случайной последовательности.

Работа с графиками в MatLab

Оформление графиков

Пакет MatLab позволяет отображать графики с разным цветом и типом линий, показывать или скрывать сетку на графике, выполнять подпись осей и графика в целом, создавать легенду и многое другое. В данном параграфе рассмотрим наиболее важные функции, позволяющие делать такие оформления на примере двумерных графиков.

Функция plot() позволяет менять цвет и тип отображаемой линии. Для этого, используются дополнительные параметры, которые записываются следующим образом:

```
plot(<x>, <y>, <'цвет линии, тип линии, маркер точек'>);
```

Обратите внимание, что третий параметр записывается в апострофах и имеет обозначения, приведенные в таблицах 2-4. Маркеры, указанные ниже записываются подряд друг за другом, например, ‘ko’ – на графике отображает черными кружками точки графика.

Таблица 2

Обозначение цвета линии графика

Маркер	Цвет линии
c	голубой
m	фиолетовый
y	желтый
r	красный
g	зеленый
b	синий
w	белый
k	черный

Таблица 3

Обозначение типа линии графика

Маркер	Цвет линии
-	непрерывная
--	штриховая
:	пунктирная
-.	штрих-пунктирная

Таблица 4

Обозначение типа точек графика

Маркер	Цвет линии
.	точка
+	плюс
*	звездочка
o	кружок
x	крестик

Ниже показаны примеры записи функции plot() с разным набором маркеров.

```
x = 0:0.1:2*pi;
y = sin(x);
subplot(2,2,1); plot(x,y,'r-');
subplot(2,2,2); plot(x,y,'r-',x,y,'ko');
subplot(2,2,3); plot(y,'b--');
subplot(2,2,4); plot(y,'b--+');
```

Представленный пример показывает, каким образом можно комбинировать маркеры для достижения требуемого результата. Следует особо отметить, что в четвертой строчке программы отображаются два графика: первый рисуется красным цветом и непрерывной линией, а второй черными кружками заданных точек графика. Остальные варианты записи маркеров очевидны.

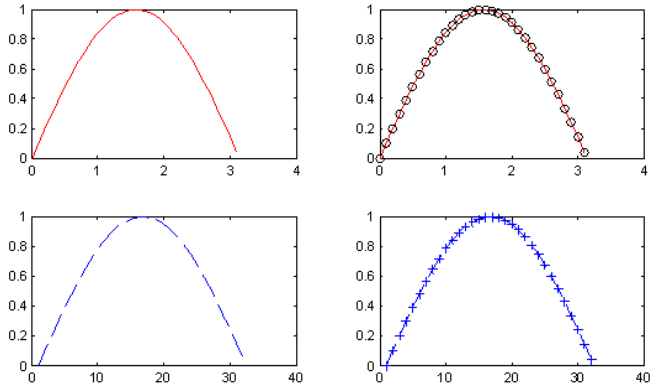

Рис. 2. Примеры отображения графиков с разными типами маркеров

Из примеров видно, что масштаб графиков по оси Ox несколько больше реальных значений. Дело в том, что система MatLab автоматически масштабирует систему координат для полного представления данных. Однако такая автоматическая настройка не всегда может удовлетворять интересам пользователя. Иногда требуется выделить отдельный фрагмент графика и только его показать. Для этого используется функция axis() языка MatLab, которая имеет следующий синтаксис:

```
axis( [xmin, xmax, ymin, ymax] ),
```

где название указанных параметров говорят сами за себя.

Воспользуемся данной функцией для отображения графика функции синуса в пределах от 0 до 2π:

```
x = 0:0.1:2*pi;
y = sin(x);
subplot(1,2,1);
plot(x,y);
axis([0 2*pi -1 1]);
subplot(1,2,2);
plot(x,y);
axis([0 pi 0 1]);
```

Из результата работы программы (рис. 3) видно, что, несмотря на то, что функция синуса задана в диапазоне от 0 до 2π, с помощью функции axis() можно отобразить как весь график, так и его фрагмент в пределах от 0 до π.

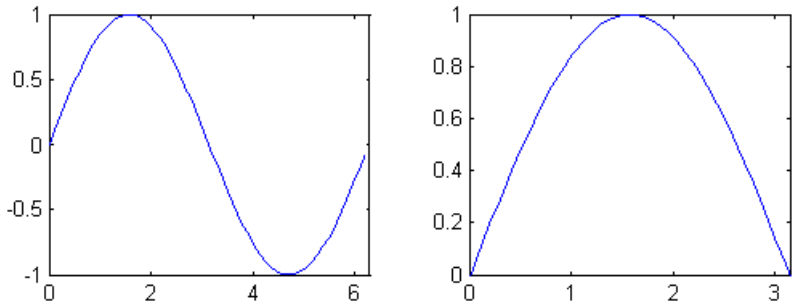

Рис. 3. Функция синуса

В заключении данного параграфа рассмотрим возможности создания подписей графиков, осей и отображения сетки на графике. Для этого используются функции языка MatLab, перечисленные в табл. 5.

Таблица 5

Функции оформления графиков

Название	Описание
grid [on, off]	Включает/выключает сетку на графике
title(‘заголовок графика’)	Создает надпись заголовка графика
xlabel(‘подпись оси Ox’)	Создает подпись оси Ox
ylabel(‘подпись оси Oy’)	Создает подпись оси Oy
text(x,y,’текст’)	Создает текстовую надпись в координатах (x,y).

Рассмотрим работу данных функций в следующем примере:

```
x = 0:0.1:2*pi;
y = sin(x);
plot(x,y);
axis([0 2*pi -1 1]);
grid on;
title('The graphic of sin(x) function');
xlabel('The coordinate of Ox');
ylabel('The coordinate of Oy');
text(3.05,0.16,'\leftarrow sin(x)');
```

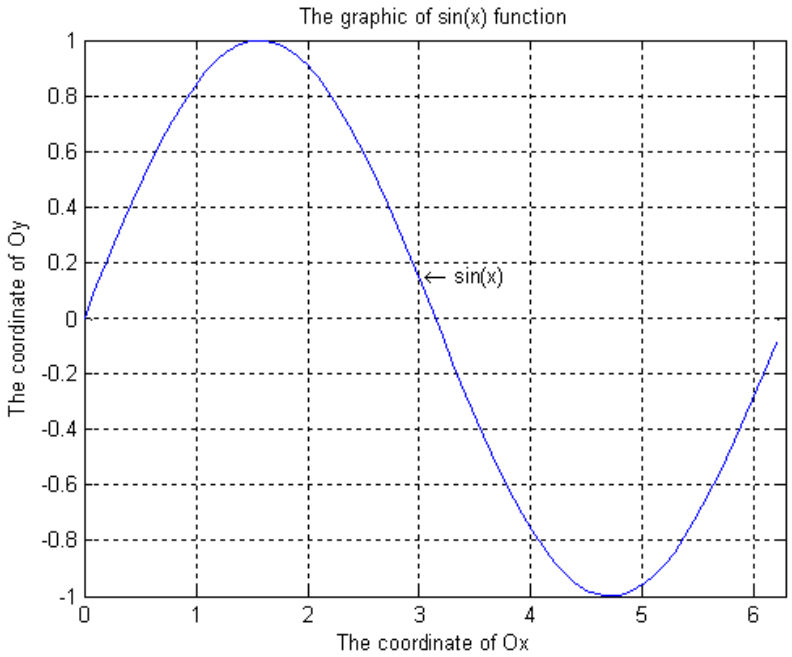

Рис. 4. Пример работы функций оформления графика

Программирование функций в MatLab

Синтаксис для определения собственных функций в MatLab имеет следующий вид:

```
function [RetVal1, RetVal2,…] = FunctionName(arg1, arg2,…)
<тело функции>
```

где RetVal1, RetVal2,... – набор возвращаемых значений функцией (результаты работы);

arg1, arg2,... – набор входных аргументов;

тело функции – набор операторов (программа), которые выполняются при вызове функции.

Рассмотрим пример реализации функции для вычисления евклидового расстояния:

```
function length = euqlid(x1, y1, x2, y2)
length = sqrt((x1-x2)^2+(y1-y2)^2);
```

Продемонстрируем возможность возвращения нескольких параметров на примере вычисления ширины и высоты прямоугольника, заданного координатами левого верхнего угла (x1,y1) и правого нижнего (x2,y2):

```
function [width, height] = RectangleHW(x1,y1,x2,y2)
width = abs(x1-x2);
height = abs(y1-y2);
```

Данную функцию можно записать еще и с таким набором параметров:

```
function [width, height] = RectangleHW(P1, P2)
width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
```

где P1 и P2 – векторы (массивы) размером в 2 элемента и описывают точку в двумерном пространстве. В этом случае при вызове функции, значения координат точек можно передавать таким образом:

```
[W, H] = RectangleHW([0 0], [10 20]);
```

Если же программист сделает ошибку и при вызове функции передаст неверный размер вектора, например, так

```
[W, H] = RectangleHW(0, [10 20]);
```

то выполнение функции завершится с ошибкой и выполнение всего алгоритма остановится. Чтобы избежать этой ситуации MatLab позволяет проводить проверку корректности переданных аргументов и корректно завершать работу функции без остановки работы всего алгоритма. Следующий пример записи функции демонстрирует работу такой проверки:

```
function [width, height] = RectangleHW(P1, P2)
if length(P1) < 2 | length(P2) < 2
     error( 'Bad 1st or 2nd parameter' );
end
```

```
width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
```

Для проверки числа переданных аргументов и числа ожидающих возвращаемых значений используются переменные nargin и nargout. Ниже приведен пример функции, использующей проверку корректности числа входных и выходных аргументов.

```
function [width, height] = RectangleHW(P1, P2)
if nargin ~= 2
    error( 'Bad number of parameters' );
end
if nargout ~= 2
    error(  'Must be 2 return values' );
end
if length(P1) < 2 | length(P2) < 2
    error( 'Bad 1st or 2nd parameter' );
end
width = abs(P1(1)-P1(2));
height = abs(P2(1)-P2(2));
```

При этом проверки корректности параметров функции будут срабатывать в следующих ситуациях:

```
[W, H] = RectangleHW([0 0]);      % Bad number of parameters
[W, H, V] = RectangleHW([0 0], [10 20]); % Must be 2 return
                                          % values
[W, H] = RectangleHW(0, [10 20]); % Bad 1st or 2nd parameter
```

Область видимости переменных

Следует отметить, что переменные, объявленные внутри функций, имеют область видимости только в пределах функции, и за ее пределами уже не доступны (не видны). Следующий пример программы демонстрирует механизм области видимости имен переменных в MatLab:

```
function MyFunc
x = 10;
disp(x);
MyFunc2();

function MyFunc2()
disp(x);
```

В результате на экране будет отображено

```
10
??? Undefined function or variable 'x'.
```

Этот пример показывает, что переменная с именем x, объявленная в функции MyFunc, не доступна в функции MyFunc2. Это сделано с расчетом, чтобы переменные в разных функциях не влияли друг на друга, даже если они имеют одни и те же имена. Однако в некоторых случаях требуется, чтобы переменная была видна за пределами функции, в которой объявлена. Это достигается путем обращения к переменной как к глобальной с помощью ключевого слова global, за которым следует имя глобальной переменной. Перепишем пример, представленный выше с использованием глобальной переменной:

```
function MyFunc
x = 10;
disp(x);
MyFunc2();
function MyFunc2()
global x;
disp(x);
```

Обратите внимание, что ключевое слово global написано в функции MyFunc2 и говорит о том, что переменная x уже объявлена ранее и нужно ее использовать внутри текущей функции.

Самостоятельные задания:

1. Заданы четыре переменные. Наименьшую из них заменить на сумму остальных. Вывести её с указанием имени переменной.

2. Есть две фигуры: квадрат задан длиной стороны, а круг – длиной радиуса. Определить, какая их них имеет большую площадь и больший периметр (и во сколько раз).

3. На плоскости заданы три точки

C1(x1, y1) , C2 (x2 , y2) , C3 (x3 , y3)

Найти расстояние этих точек от начала координат.

Вывести информацию в виде:

а) Вывести слова:

ИСХОДНЫЕ ДАННЫЕ

б) Пропустить строку;

в) Вывести значения исходных данных в виде:

КООРДИНАТЫ ТОЧЕК

x1= … x2=… x3=…

y1= … y2=… y3=…

г) Для подчеркивания вывести строку из дефисов

д) Пропустить 2 строки и вывести ответ в виде:

РАССТОЯНИЕ ОТ НАЧАЛА КООРДИНАТ

ТОЧКА C1 - … ТОЧКА C2 - … ТОЧКА C3 - …

4. Реализовать функцию myfun1, имеющую два аргумента (вектор и строку) и один результат – скаляр. Вычисление скаляра зависит от вида строки (предусмотреть три варианта расчета) – например, может рассчитываться среднее арифметическое.

Осуществить проверку корректности переданных аргументов. Для проверки числа переданных аргументов и числа ожидающих возвращаемых значений использовать переменные nargin и nargout.

Протестировать М-файл**.**

5. Построить графики функции (продумать самостоятельно параметры и их значения). Составить М-файл-сценарий. Использовать функции pol2cart (полярные координаты), plot3, comet3.