Лабораторная работа №4

Использование подпрограмм в системе MATLAB

Цель работы: Знакомство с организацией функций в MATLAB, особенностями их работы.

Согласно одному из принципов структурного программирования, повторяющиеся фрагменты программы, а также логически целостные ее фрагменты можно оформить в виде подпрограмм (процедур и функций). Подпрограммы представляют собой относительно самостоятельные фрагменты программы, для оформления и вызова которых используют специальные синтаксические средства. Подпрограмма может быть многократновызванаиз разных частей программы. В MATLAB подпрограммы реализованы в виде функций.

В отличие от программ-сценариев (скриптов) функция может иметь входные и выходные параметры, при вызове функция создает новую рабочую область. Напомним, что рабочей областью называют область памяти, в которой размещены переменные.Так как функция создает новую рабочую область, внутри функции не будут видны переменные, объявленные вне ее кода. Поэтому входные и выходные параметры используются для связи функции с внешним кодом.

Функции, так же как и скрипты,оформляются в отдельномm-файле. Файл, содержащий функцию и файл скрипта, вызывающего эту функцию, должны содержаться в одном каталоге.

В общем виде программа функции имеет следующую структуру:

```
function [y1,...,yN] = имя_функции(x1,...,xM)
оператор_1
оператор_2
…
оператор_n
end
```

Функция начинается с ключевого слова `function`, за которым следуют в квадратных скобках через запятую имена выходных переменных. Далее следует знак «=» и имя функции. Имя функции подчиняется тем же правилам, что и имена переменных. Имя функции обязательно должно совпадать с именем файла, в котором она определена. После имени функции в круглых скобках через запятую следуют имена входных параметров.Следующие строки содержат тело функции (любые допустимые выражения

MATLAB).Конец функции определяет ключевое слово `end`, однако оно не является обязательным, и его можно опустить. Если функция возвращает только один параметр, то его не обязательно заключать в квадратные скобки, например:

```
function s = triaArea( a, b )
% вычисление площади прямоугольного треугольника
% a, b - катеты треугольника
s = a * b / 2;
end
```

Если функция не имеет возвращаемых параметров, то после ключевого слова `function`следует имя функции, например:

```
function hellowWorld
% пример функции без входных и выходных параметров
disp('Helloworld!');
end
```

Для вызова функции применяется следующий синтаксис:

```
[k1, ..., kN] = имя_функции(z1,...,zM)
```

где `k1, ..., kN`– переменны, куда будут записаны выходные значения функции, а `z1,...,zM`– аргументы функции.

В случае, если функция возвращает только один параметр, квадратные скобки можно опустить, например:

```
Area = triaArea(1,2)
```

Фактические и формальные параметры функции

При написании и использовании функций различают фактические и формальные параметры функции:

- фактический параметр – аргумент, передаваемый в функцию при ее вызове;
- формальный параметр – аргумент, указываемый при определении функции.

Поясним различие между фактическими и формальными параметрами функции на примере:

Рассмотрим функцию, вычисляющую площадь и периметр параллелограмма. Функция содержится в файле`ParArea`.m:

```
function [s, p] = ParArea( a, b, alfa )
% Функция для вычисления площади и периметра параллелограммапо
% двум сторонам и углу между ними
```

```
% a, b, alfa - входные параметры (две стороны и угол), скаляры
% s, p - площадь и периметр треугольника соответственно, скаляры
% a, b, alfa, s, p являются формальными параметрами, то есть
% объявленные в определении функции.
% С входными параметрами можно работать, как с обычными
переменными.
% Выходным параметрам необходимо присвоить какое-либо значение

% Площадь параллелограмма
s = a*b*sin(alfa);
% Периметр параллелограмма
p  = 2*(a+b);
end
```

Пример использования данной функции:

```
>> a = 2;
>> b = 3;
>> [area, perimeter] = ParArea(a, b, pi/3)
area =
    5.1962
perimeter =
    10
```

При вызове функции `ParArea` параметры `a, b, pi/3, area, perimeter` являются фактическими. При вызове функции фактические параметры подставляются в формальные, поэтому их имена могут быть различными.

Для корректной работы функции при ее вызове имеет значение порядок следования переменных. Так, если в предыдущем примере поменять местами переменные `perimeter` и `area`, результат, который вернет функция, по смыслу будет неверным:

```
>> a = 2;
>> b = 3;
>> [perimeter, area] = ParArea(a, b, pi/3)
perimeter =
    5.1962
area =
    10
```

Рабочая область функции

В MATLAB рабочие области бывают двух типов:

- basework space – базовая рабочая область;
- function workspace – рабочая область функции.

Как было отмечено ранее, при вызове функции создается новая рабочая область, которая существуют до конца работы функции. Все переменные, которые создаются в данной рабочей области, называются локальными. Они отличны от переменных других функций и переменных базовой рабочей области, даже если имеют одинаковые имена. Например, создадим следующую функцию:

```
function [] = demoLocaLVar()
% пример локальной переменной
x = 2;
end
```

И введем следующие команды в командное окно MATLAB:

```
>> x = 0
x =
     0
>>demoLocalVar()
>>x
x =
0
```

Как видно из примера, даже не смотря на то, что в функции была создана переменная `x` и ей было присвоено значение 2, это никак не повлияло на переменную `x`, созданную в базовой рабочей области.

После завершения выполнения функции рабочая область функции и содержащиеся в ней переменные удаляются. Выполнение функции завершается по выполнению последнего оператора, либо по команде `return`.

Программы-сценарии (скрипты) и функции

Программы-сценарии (скрипты) в MATLAB используют рабочую область программы, из которой они были вызваны, что можно использовать для передачи

параметров скриптам. Например напишем скрипт, вычисляющий гипотенузу прямоугольного треугольника:

```
% скрипт, вычисляющий гипотенузу прямоугольного треугольника
% a, b - катеты прямоугольного треугольника
% c - вычисляемая гипотенуза

% промежуточные расчеты
x = a^2 + b^2;

% вычисляем гипотенузу
c = sqrt(x);
```

Пример использования данного скрипта:

```
>> a = 3;
>> b = 4;
>>hypScr
>>c
c =
     5
>>x
x =
25
```

То есть, вызванные из консоли переменные `a` и `b` были созданы в базовой рабочей области. Далее из консоли вызывается программа-сценарий (скрипт) `hypScr`, которая в базовой рабочей области создаст переменные `c` и `x`, используя ранее созданные переменные `a` и `b`.

У данного подхода имеются следующие недостатки:

- необходимо заранее создать переменные с заранее заданными именами (`a`, `b`);
- после вычислений появится ненужная переменная `x`;
- все четыре переменные (`a`, `b`, `c` и `x`) могут использоваться в коде ранее и в них могут уже содержаться данные, которые нельзя терять, поэтому неосторожное использование скриптов может привести к возникновению ошибок.

Этих недостатков лишены функции:

```
function c = hypFun( a, b )
% функция, вычисляющая гипотенузу прямоугольного треугольника
% a, b - катеты прямоугольного треугольника, скаляры
```

```
% c - вычисляемая гипотенуза, скаляр

% промежуточные расчеты
x = a^2 + b^2;

% вычисляем гипотенузу
c = sqrt(x);

end
```

Пример использования данной функции:

```
>> a = 2; % некие важные данные, которые нужно сохранить
>> x = 3; % некие важные данные, которые нужно сохранить, со
скриптом это не получится
>> c = hypFun(3,4)
c =
     5
>>a
a =
2
>> x
x =
     3
```

Так как функция создает свою рабочую область, переменные `a` и `x`, содержащиеся в базовой рабочей области, переопределены не будут.

Передача параметров по значению

В MATLAB параметры всегда передаются по значению. Передача по значению означает, что вызывающая функция копирует в рабочую область функции непосредственное значение параметра. Изменение копии переменной, соответственно, оригинал не затрагивает.

Приведем пример. Функция demoTrFun:

```
function [ ] = demoTrFun( x )
% демонстрация передачи параметров по значению

% увеличим переменную x
```

```
x = x + 5;


end
```

Демонстрация использования данной функции:

```
>> x = 0; % создадим переменную x
>>demoTrFun(x); % вызовем тестовую функцию, чтобы показать
передачу по значению
>> x % покажем, что действительно была передана копия переменной
x (x не должна изменится)
x =
     0
```

Проверка корректности входных переменных

Как правило, функция должна выполнять проверку корректности своих параметров. В случае, если параметры имеют некорректное значение, функция должна каким-то образом сообщить об этом вызывающей программе. В MATLAB как правило используется функция `error`, принимающая самом простом случае текст, описывающей ошибку, и прерывающая работу текущей функции и, если ошибка не перехвачена, то и работу программы. Приведем пример использования функции `error`:

```
function [ res ] = myGCD( a, b )
% вычисление наибольшего общего делителя чисел a и b

% проверка, что числа a и b не имеют дробных частей
if a ~= round(a) || b ~= round(b)
error('Входные аргументы должны быть целыми');
end

while a ~= b
if a > b
        a = a - b;
else
        b = b - a;
end
end
```

```
% сохранить результат в выходную переменную
res = a;

end
```

Пример использования этой функции:

```
>>myGCD(78, 66)
ans =
     6



>>myGCD(32, 48.5)
Error using myGCD (line 6)
Входные аргументы должны быть целыми
```

Анонимные функции

В MATLAB существует еще один вид функций, так называемые анонимные функции. Эти функции не имеют файла определения, а лишь ассоциированы с переменной, имеющей тип `function_handle`. Обращение к анонимным функциям (АФ) производится с помощью данных переменных.

Создадим АФ, вычисляющую квадрат числа:

```
>>sqr = @(x) x.^2;
```

Теперь квадрат 5 можно вычислить следующим образом

```
>>sqr(5)
ans =
    25
```

Ограничением для функций является то, что при определении тела АФ может быть использовано только одно выражение. Внутри АФ могут вызываться другие функции, в том числе и другие анонимные. Нельзя использовать управляющие конструкции условия и цикла.

При написании тела функции могут использоваться переменные, уже существующие в рабочем пространстве. Значения данных переменных будут сохранены в определении АФ и использоваться независимо от своих глобальных определений. Даже

если вы удалите переменные после того, как определили АФ, значения данных переменных будут использованы при выполнении функции.

```
>> a = -1.3; b = .2; c = 30;
>>parabola = @(x) a*x.^2 + b*x + c;
>> clear a b c
>> x = 1;
>> y = parabola(1)

y =
31.5000
```

Если есть необходимость изменить значения используемых переменных, то придется создать АФ заново.

```
>> a = -1.3; b = .2; c = 30;
>>parabola = @(x) a*x.^2 + b*x + c;
>> a = -3.9; b = 52; c = 0;
>>parabola = @(x) a*x.^2 + b*x + c; % Создание АФ с новыми значениями a, b, c
>> x = 1;
>> y = parabola(1)

y =
   48.1000
```

Сравнения таких объектов MATLAB, как скрипт, функция и анонимная функция, приведено в таблице 1.

Таблица 1. Сравнение программ-сценариев, функций и анонимных функций

	Объявление	**Вызов**	**Рабочая область**	**Где хранится**
Программа-сценарий (скрипт)	Построчное написание	`name`	`base`	Файл с расширением `.m`
Функция	`function []=name()` `end`	`[]=name()`	`function`	Файл с расширением `.m`
Анонимная функция	`name=@(x)f(x)`	`a=name()`	`function`	Переменная типа `function_handle`

Построение двух графиков в рамках одного окна

Приведем пример скрипта для построения двух графиков в рамках одного окна в MATLAB:

```
% скрипт, строящий два графика в рамках одного окна

%координаты точек для построения графика по оси абсцисс
x = 0 : 0.05 : 6*pi;

%координаты точек для построения графика по оси ординат
y1 = ( sin( 2 * x ) + 1 ) .* ( x .^ 2 );
y2 = x.^2;

% строим два графика
plot(x, y1, x, y2);

% заголовок графика
title('Построение двух графиков в рамках одного окна')

% легенда
legend('f1(x) = (sin(2*x)+1).*(x.^2)',' f2(x) = x.^2 ')
```

В результате выполнения данного скрипты появится следующее окно с графиком:

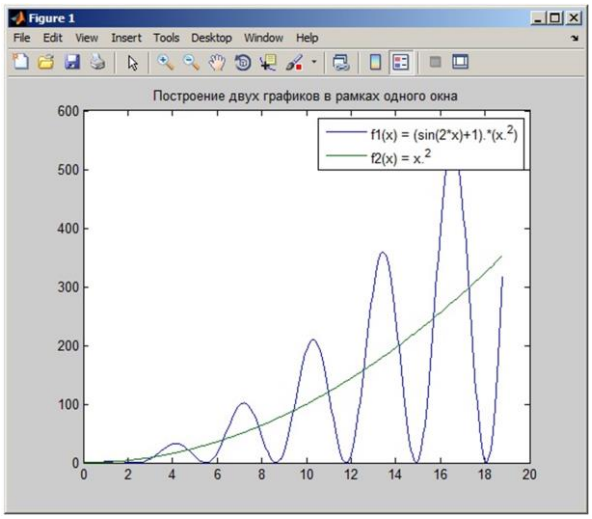

Рисунок 1. График в MATLAB

В простейшем случае для построения графиков используется функция plot, принимающая два входных аргумента, которыми являются векторы одинаковой длинны, задающие координаты точек для построения графика. Если возникает необходимость построить два графика на одной координатной плоскости, в функцию plot подставляются четыре входных аргумента. В первом аргументе содержится значение координат точек по оси абсцисс для первого графика, во втором — по оси ординат для первого графика, в третьем — по оси абсцисс для второго графика, во втором — по оси ординат для второго графика.

Для того чтобы задать заголовок графика используется функция `title`, принимающая заголовок в виде строки текста и выводящая его над графиком.

Функцию `legend` позволяет вывести легенду к графику. Данную функцию удобно использовать тогда, когда на одной координатной плоскости строится несколько графиков, т.е. в случаях подобных нашему примеру. Эта функция принимает подписи к графикам через запятую в виде текста.

Задание на лабораторную работу №4

1. Написать функцию f1, которая будет рассчитывать значение кусочно заданной функции (см. лабораторную работу №3). Входным параметром функции является скаляр – аргумент кусочно заданной функции. Выходным параметром функции является скаляр - значение кусочно заданной функции в точке-аргументе. Если аргумент функции не принадлежит области определения функции, должно быть выведено соответствующее

сообщение и работа функции должна быть остановлена. Текст функции сопроводить комментариями.

2. Написать программу-сценарий (скрипт), которая сформирует вектор `x` с использованием оператора двоеточия со значениями от `xMin` до `xMax` с шагом `dx=0.1`, где `xMin` и `xMax` – левая и правая границы интервала, на котором определена функция, соответственно; для каждого элемента созданного вектора `x` вычислит значения с использованием функции `f1` и запишет результат в вектор `y1`;вызовет скрипт `definition_anfun` с определением анонимной функции `f2` (см. табл. 4, функция для задания №1, лабораторная работа №1), для каждого элемента вектора `x` вычислит значения с использованием анонимной функции `f2` и запишет результат в вектор `y2`, построит графики двух заданных функций в рамках одного окна, используя векторы `x`, `y1` и `y2`, добавит к графику заголовок и легенду. При необходимости, использовать масштабирование для более наглядного отображения графиков функций.

3. Весь написанный программный код необходимо сопроводить комментариями.

4. Используя результаты лабораторной работы №3, сделать выводы по использованию программ-сценариев (скриптов) и функций для решения одной и той же задачи.

Контрольные вопросы

1. Дайте определение понятиям «программа-сценарий (скрипт)», «функция» и «анонимная функция»? В чем их разница?
2. Какие рабочие области в MATLAB вы знаете? Дайте определение «локальной переменной».
3. Что означает «передача параметров по значению»?
4. Напишите синтаксис объявления функции. Приведите пример объявления и вызова функции.
5. В чем разница между фактическими и формальными параметрами функции?
6. Напишите синтаксис объявления анонимной функции. Приведите пример объявления и вызова функции.
7. Для чего нужно документирование функций и чем оно отличается от комментирования?
8. Дайте определение понятиям «отладка» и «точка останова».
9. В чем состоят основные отличия программ-сценариев (скриптов) и функций?

10. Что выполняет функция `error`?

Требования к содержанию отчета

Отчет по лабораторной работе оформляется в любом текстовом редакторе и предоставляется в печатном виде. Отчет должен состоять из следующих разделов:

1. Титульный лист. На титульном листе необходимо указать номер и название лабораторной работы, номер варианта, ФИО и группу исполнителя, ФИО преподавателя.
2. Цель работы.
3. Задание на лабораторную работу в соответствии с номером варианта.
4. Ход работы:
   - Листинг функции f1;
   - Листинг Скрипта definition_anfun с объявлением анонимной функции
   - Скрипт вызова функций f1 и f2
   - Графики функций
5. Выводы по работе.

   К отчету прилагаются:
   - файл с текстом отчета;
   - m-файл с текстом программы

   Файл отчета необходимо разместить в личном кабинете.