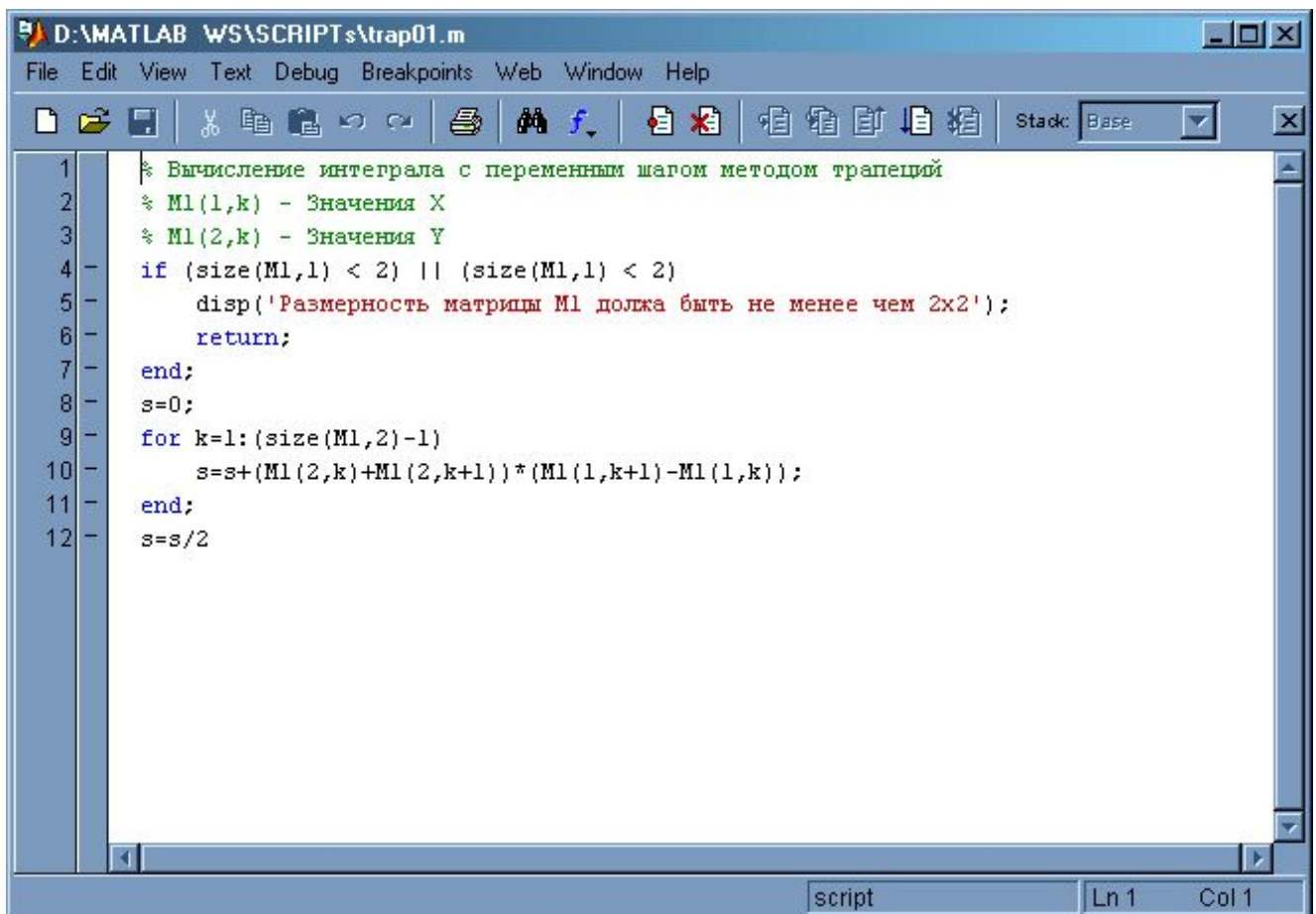


## ЛАБ. РАБОТА №5. СКРИПТЫ И ФУНКЦИИ

### Введение

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



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

### СКРИПТЫ

Как уже говорилось во введении, для выполнения последовательности инструкций MATLAB мы переносили в окно Command Window путем копирования и вставки. Когда инструкций не так много, это не вызывает затруднений. Но когда количество инструкций занимает несколько экранов такой подход становится обременительным. Естественным решением этой проблемы является размещение всей цепочки инструкций в отдельном файле и запуск такой цепочки путем ее вызова по имени такого файла. В этом и

закljučается смысл и техника использования скриптов. Осталось только добавить что файлы скриптов по прежнему остаются текстовыми файлами, однако имеют расширение \*.m

Например, файл trap01.m, который представлен на ранее приведенном рисунке, в командной строке вызывается следующим образом:

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

Результат:

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

```
>> trap01
```

Результат:

```
s =  
    21.5000
```

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

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

Например:

```
>> help trap01
```

Результат:

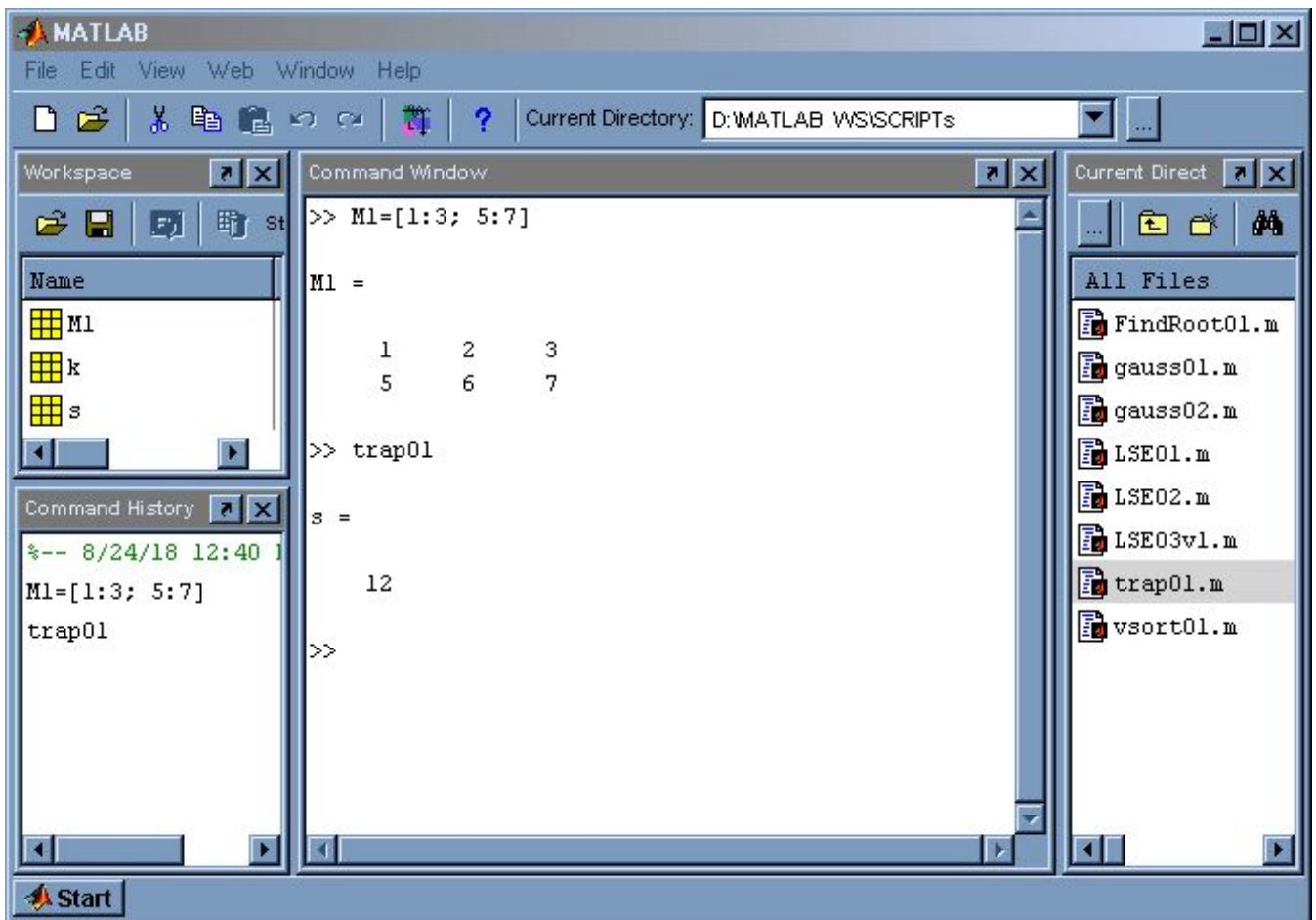
```
Вычисление интеграла с переменным шагом методом  
трапеций  
M1(1,k) - Значения X  
M1(2,k) - Значения Y
```

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

#### **Правило оформления help для скриптов:**

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

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



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

## ***ФУНКЦИИ***

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

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

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

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

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

$$\langle \text{список параметров} \rangle ::= \langle \text{имя переменной} \rangle =$$

$$| [ \langle \text{список имен} \rangle ] =$$

&lt;имя функции&gt;::= &lt;идентификатор&gt;

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

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

Приведем несколько примеров заголовков функций:

(Пример №1)

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

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

(Пример №2)

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

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

(Пример №3)

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

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

## ***Глобальные и локальные переменные***

### **Глобальные переменные и константы**

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

#### **Примечание .**

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

## **Локальные переменные функции**

Все переменные, которые создаются инструкциями в теле функции, являются локальными. Главная особенность локальных переменных это время их существования. Локальные переменные существуют только пока выполняется тело функции и автоматически удаляются при завершении функции. Другими словами, до начала работы функции они еще не существуют, после завершения работы они уже не существуют. Особо подчеркнем, что при таком режиме существования, локальные переменные должны размещаться и действительно размещаются в отдельном блоке памяти никак не связанном с Workspace. Названные особенности называют изоляцией или инкапсуляцией переменных внутри функции. Изоляция локальных переменных от Workspace позволяет свободно выбирать идентификаторы для имен локальных переменных и тем самым реализовывать алгоритм функции без оглядки на имена глобальных переменных. Другими словами, при работе инструкций в теле функции поиск любой используемой переменной вначале обращается именам в локальной области и только, если имя в локальной области не найдено, то начинают рассматриваться имена в глобальной области, то есть, в Workspace. Это важнейшее свойство локальных переменных позволяет рассматривать функции как истинно универсальные инструменты, которые можно создавать в отрыве от конкретного места их применения в дальнейшем.

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

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

## **Вызов функций**

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

Например, если мы имеем глобальную матрицу A1, которая описывает некоторую таблично заданную функцию и у нас есть функция MATLAB, которая предназначена для вычисления определенного интеграла таких таблично заданных функций

```

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

```

то для вызова такой функции MATLAB необходимо выполнить вызов:

```
ftrap01(A1)
```

Для начала убедимся в этом, выполнив следующий пример:

```

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

```

Результат:

```

A1 =
     1     2     3     4
     1     4     9    16

```

Интеграл с переменным шагом методом трапеций

```
ans = 21.5000
```

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

### **Основные правила работы с функциями**

- Все переменные, которые создаются в теле функции, являются локальными и существуют в течение времени, пока выполняются инструкции функции.

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

## ПРИМЕРЫ

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

### ***ПРИМЕР 1. Пузырьковая сортировка вектора – строки или вектора – столбца.***

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



```

        end;
    end;
    n=n-1;
end;

```

Покажем пример работы с этой функцией:

```

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

```

Результат:

```

B1 =      1      1      2      3      3      4      6      8

```

## **ПРИМЕР 2. Вычисление функции SIN(x) рядом Маклорена.**

Математическая постановка задачи.

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

Традиционно ряд Тейлора определяют следующим образом:

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

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

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

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

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

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

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

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

Рассмотрим к вачестве примера вычисление функции  $SIN(x)$  рядом Маклорена.

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

Выполним вызов этой функции и сравним ее ответ с библиотечной функцией MATLAB `sin(x)`:

```
x=pi/2;  
y=f*sin01(x);  
y=sin(x)
```

Результат:  
ans = 2.2204e-016

В качестве значений параметров можно передавать не только переменные `n` и явные значения:

```
f sin01(3.14)
```

Результат:  
ans = 0.0016

## ПРАКТИЧЕСКАЯ РАБОТА.

1. Выполните все примеры из теоретической части лабораторной работы.
2. Восстановите блок – алгоритмическую схему пузырьковой сортировки и выполните текстовое описание работы алгоритма.

3. Напишите функцию сортировки для сортировки вектора по убыванию.
4. Напишите функцию сортировки для сортировки для двумерной матрицы, в которой первая строка содержит аргумент таблично заданной функции, а вторая значение такой функции для соответствующего аргумента.
5. Добавьте в список входных параметров функции `fsin01` параметр, с помощью которого можно указывать в радианах или градусах задается значение  $x$ . Выполните необходимые доработки функции.
6. Вычислите в нуле некоторое количество производных необходимого порядка для функции  $\cos(x)$  и модифицируйте `fsin01` для вычисления косинусов.

---

Финальная редакция материала 23.08.2018г.  
Воронов С.И.