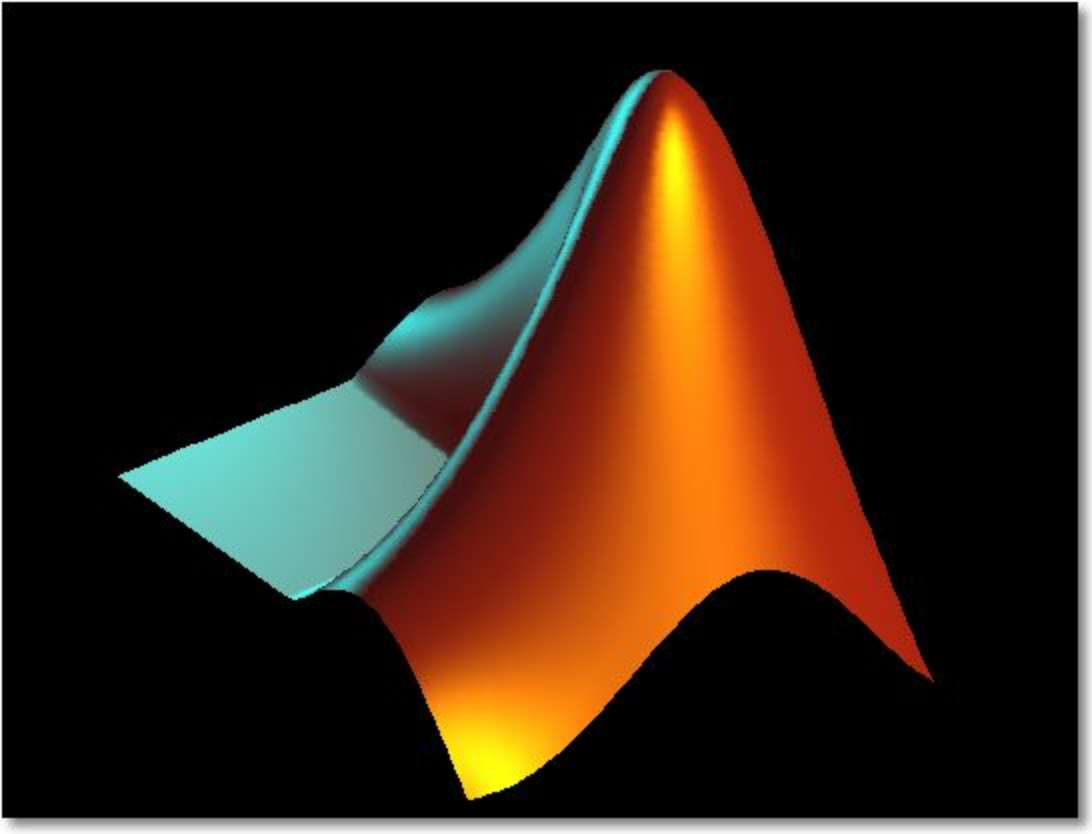


Лекция 5

Программирование в Matlab

Типы программных файлов

- Написание программ – это альтернатива работе в командной строке
- Программный код Matlab размещают в файлах с расширением «`m`» (m-файлах)
- m-файлы бывают двух видов:
  - *скрипты (scripts)*
  - *функции (functions)*
- К сожалению, Matlab плохо понимает кириллицу…

Скрипты

- Представляют собой последовательности команд Matlab
  - как если бы мы перенесли их из командного окна в отдельный файл
- Скрипт вызывается по имени через командную строку
- Скрипт выполняется в режиме интерпретатора

Функции

- Специальный вид m-файлов
- В отличие от скриптов могут принимать аргументы и возвращать значения
- Использование функций позволяет
  - структурировать программу
  - избежать повторения кода

Скрипты

- Полезны для автоматизации последовательности действий, которые выполняются многократно
- Не могут принимать параметры и возвращать аргументы
- Хранят значения своих переменных в рабочем пространстве
  - где переменные доступны для других скриптов и из командной строки

Функции

- Создание функции преследует целью расширение языка
- Переменные, определённые внутри функции являются *локальными*
  – то есть видны только внутри самой функции
- Функция имеет собственное имя
- Кроме того, с ней связано имя m-файла, в котором функция записана
  – будем соблюдать правило: имя функции и имя m-файла должны быть одинаковы

Структура функции

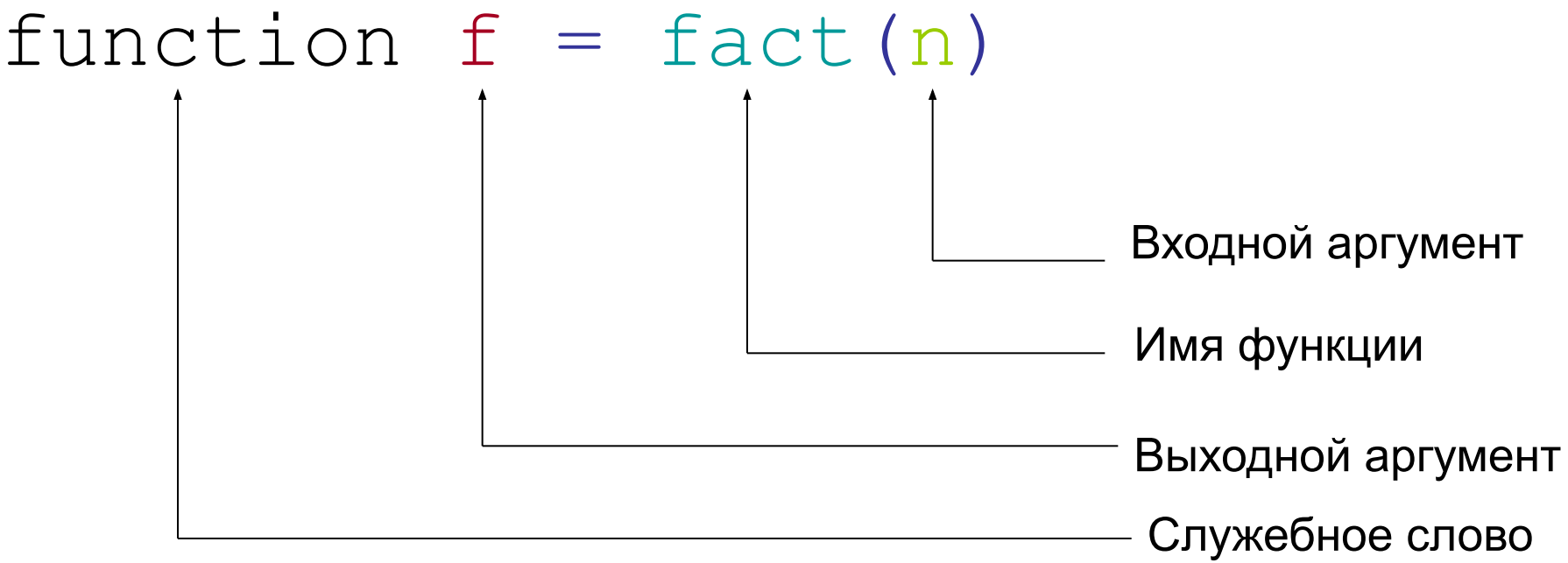
- Функция состоит из *заголовка* и *тела*

```
function f = fact(n)          Заголовок
% Вычисляет факториал.       Комментарий
% FACT(N) возвращает N!,     Комментарий

f = prod(1:n);                Тело функции
```

- Фактически, функция отличается от скрипта наличием заголовка и способом вызова

Заголовок функции



Комментарии

- Используются для
  - пояснения кода
  - временного исключения кода из текста
- Могут быть *строчными* и *блочными*
- Строчные начинаются с символа «%»
  - с этого места и до конца строки всё игнорируется компилятором % как в этом примере
- Блочные начинаются с символа «%{» и заканчиваются символом «%}»:

  %{
  - эти символы должны обязательно стоять в отдельных строках!

  %}

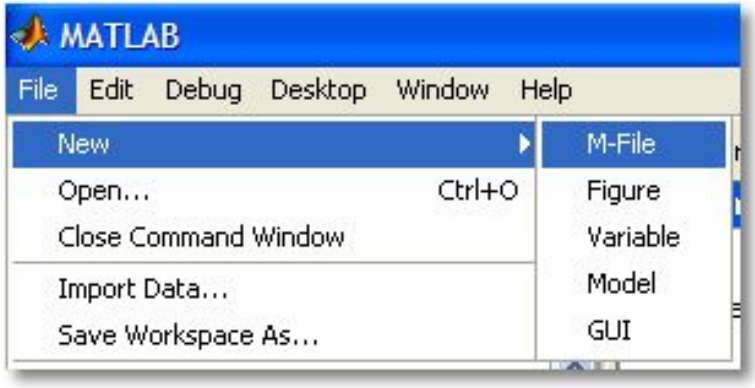
Комментарии

- Можно автоматически закомментировать блок текста. Для этого:
  - выделить блок
  - щёлкнуть правой кнопкой
  - выбрать Comment (или Ctrl+R)
- Снять комментарий:
  - выделить закомментированный блок
  - щёлкнуть правой кнопкой
  - выбрать Uncomment (или Ctrl+T)

Создание функции

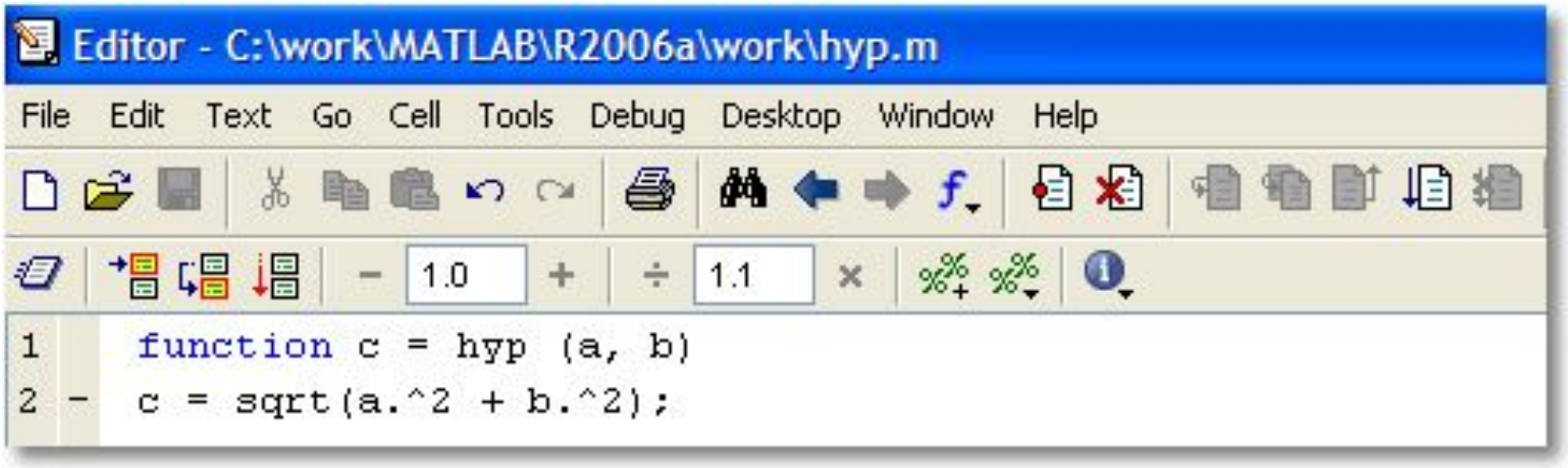
- m-файл можно создать в любом текстовом редакторе
- Например, во встроенном редакторе
  - при помощи меню
  - или командой

```
edit <имя файла>
```



Использование функции

- Функция вызывается по своему имени (которое совпадает с именем её m-файла)




```
>> a = [1:3]; b = [5:7];
>> z = hyp(a, b)

z =

    5.0990    6.3246    7.6158
```

Входные и выходные параметры

- При написании функций в Matlab можно (и желательно!) проводить проверку количества входных и выходных параметров
- Для этого в описании функции используют служебные слова:
  - `nargin`: количество входных параметров
  - `nargout`: количество выходных параметров

Входные и выходные параметры (пример)

```
 1   function [u, v] = wh (a, b)
 2
 3 - switch nargin
 4 -     case 1
 5 -         if nargout == 1
 6 -             u = a.^2;
 7 -         else
 8 -             u = a;
 9 -             v = 1/a;
10 -         end
11 -     case 2
12 -         if nargout == 1
13 -             u = a.^2 + b.^2;
14 -         else
15 -             u = a + b;
16 -             v = 1./(a + b);
17 -         end
18 - end
```

```
>> x = wh(2)

x =

     4

>> [x, y] = wh(2)

x =

     2


y =

    0.5000

>> x = wh(2, 3)

x =

    13

>> [x, y] = wh(2, 3)

x =

     5


y =

    0.2000
```

Подфункции

- В файлах-функциях Matlab могут быть реально описаны несколько функций
- Синтаксически это оформляется как две (или более) функций, записанных в одном файле
- При вызове такого m-файла происходит запуск самой первой функции
  - её имя должно совпадать с именем файла
- Описание следующих функций локально
  - обычно они используются как вспомогательные для первой функции

Подфункции (пример)

```
function [avg, med] = newstats(u) % Primary function
% NEWSTATS Find mean and median with internal functions.
n = length(u);
avg = mean(u, n);
med = median(u, n);

function a = mean(v, n)              % Subfunction
% Calculate average.
a = sum(v)/n;

function m = median(v, n)            % Subfunction
% Calculate median.
w = sort(v);
if rem(n, 2) == 1
   m = w((n+1) / 2);
else
   m = (w(n/2) + w(n/2+1)) / 2;
end
```

Вложенные функции

- Помимо последовательного вложения в один файл функция может быть описана непосредственно в теле другой функции
- Такая функция называется *вложенной*
- Вложенная функция, в свою очередь, может содержать другие вложенные функции

Вложенные функции (примеры)

```
function x = A(p1, p2)
...
   function y = B(p3)
   ...
   end

   function z = C(p4)
   ...
   end
...
end
```

```
function x = A(p1, p2)
...
   function y = B(p3)
   ...
      function z = C(p4)
      ...
      end
   ...
   end
...
end
```

Создание p-кода

- При вызове m-файла сравнительно много времени тратится на его компиляцию
- Чтобы сократить время выполнения можно предварительно перевести m-файл в p-код («пи-код»)
  - команда `pcode <имя m-файла>`
- Откомпилированный в псевдокод файл получает расширение «p»
- Такой файл будет выполняться быстрее, чем обычный m-файл

Интерактивный ввод данных

- Используется при написании скриптов
- Для ввода числовых данных применяют функцию `input` по формату

```
x = input('строка приглашения')
```

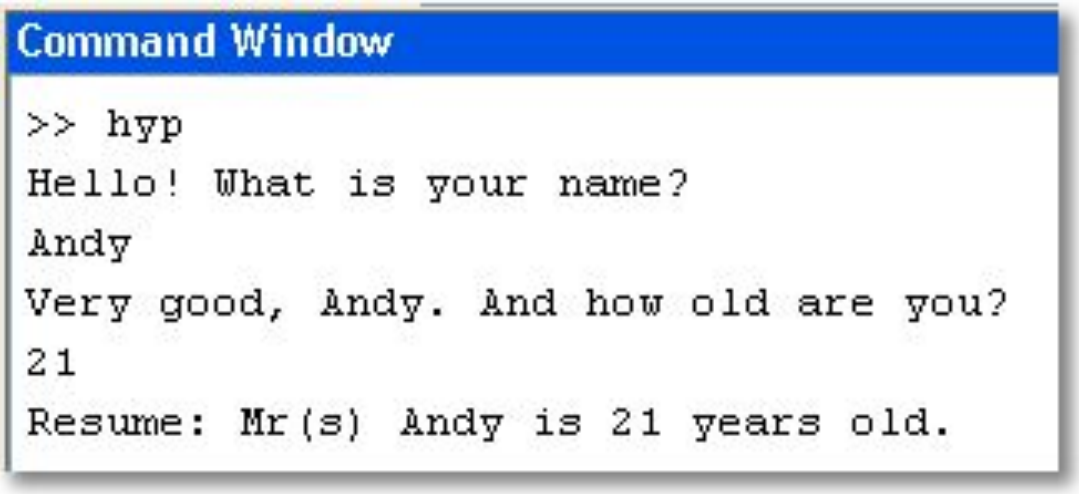
- Введённое пользователем значение сохранится в переменной x
- Для ввода строковых данных функция `input` вызывается с дополнительным параметром:

```
c = input('строка приглашения','s')
```

- Кроме того, имеется Си-подобная функция `sscanf`

Пример использования `input`

```
1 -  name = input ('Hello! What is your name?\n', 's');
2 -  y = input (['Very good, ', name, '. And how old are you?\n']);
3 -  disp(['Resume: Mr(s) ', name, ' is ', int2str(y), ' years old.'])
```



Вывод данных в командное окно

- Для этого используют команду `disp` (от *display*) по формату

  `disp(<выводимая строка>)`
- Если выводимое значение – число, то вначале его преобразуют к строковому типу при помощи функций `int2str` или `num2str`
- Конкатенацию строк производят как для одномерных векторов-строк

```
>> x = 2 + pi;
>> disp(['x = ', num2str(x)] )
x = 5.1416
```

- Кроме того, имеется Си-подобная функция `sprintf`

Основные языковые конструкции

- Как и любой процедурный язык высокого уровня, Matlab позволяет использовать при написании программ
  - следование
  - ветвление
  - циклы
  - пользовательские функции

Следование

- Реализуется перечислением каждого из операторов в отдельной строке
- Либо в одной строке через запятую (или точку с запятой)

Ветвление

- Реализуется в двух вариантах:
  - при помощи оператора `if`
  - при помощи оператора `switch`

Оператор `if`

- Простейшая форма:

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

```
if rem(a, 2) == 0
    disp('a is even')
    b = a/2;
end
```

Полный формат оператора `if`

- В полном варианте оператора могут использоваться слова `else` и `elseif`
- Слово `elseif` может использоваться в одном операторе многократно с указанием условия
- Слово `else` – только один раз в конце оператора и без условия

```
if n < 0                % If n negative, display error message.
    disp('Input must be positive');
elseif rem(n,2) == 0 % If n positive and even, divide by 2.
    A = n/2;
else
    A = (n+1)/2;        % If n positive and odd, increment and divide.
end
```

Циклы

- В Matlab имеется два вида циклов:
  - цикл с параметром `for`
  - цикл с предусловием `while`
- Также имеются
  - оператор досрочного выхода из цикла `break`
  - оператор перехода к следующей итерации `continue`

Цикл с параметром

```
for index = start:increment:end
    statements
end
```

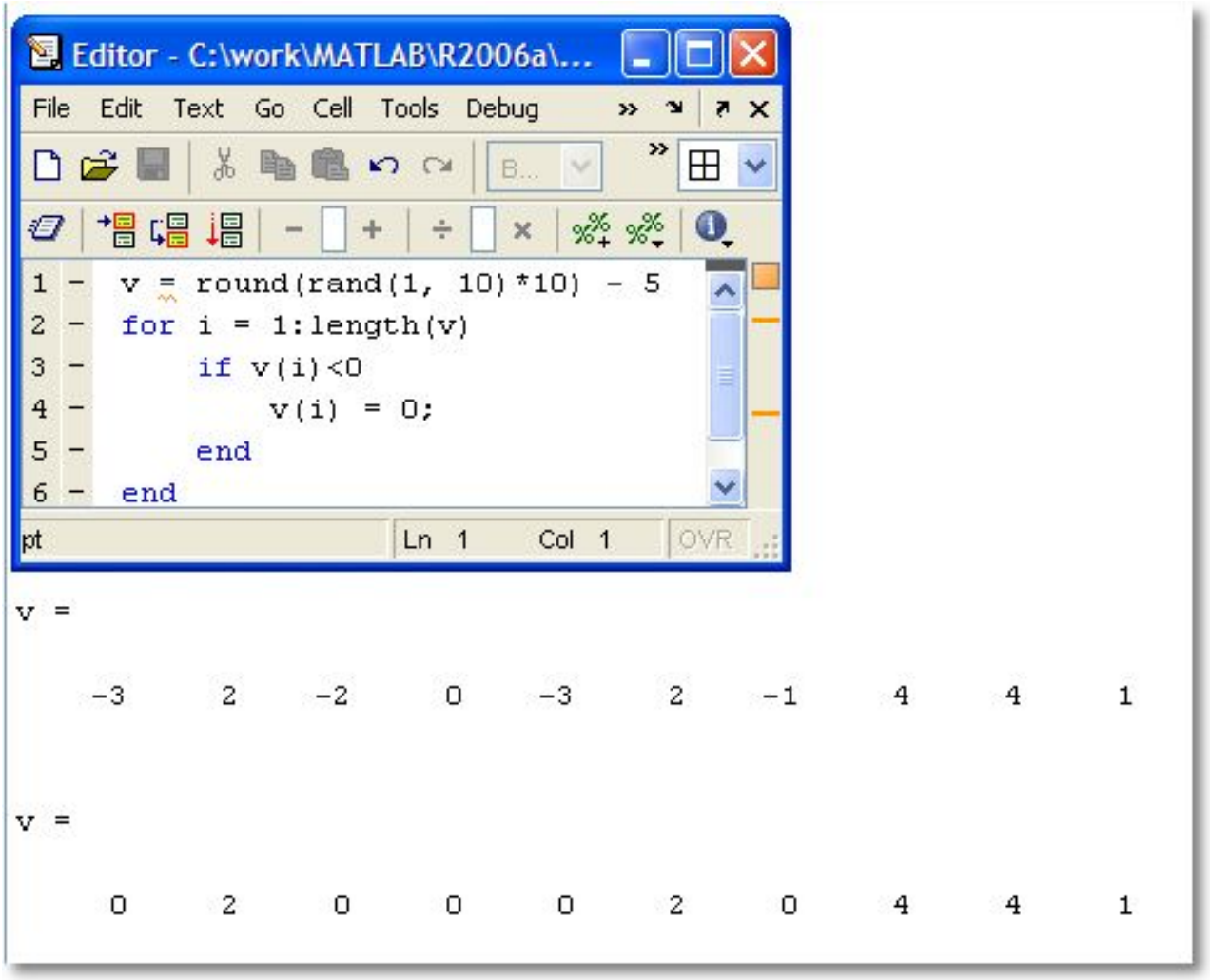
```
for n = 2:6
    x(n) = 2 * x(n - 1);
end
```

```
for m = 1:5
    for n = 1:100
        A(m, n) = 1/(m + n - 1);
    end
end
```

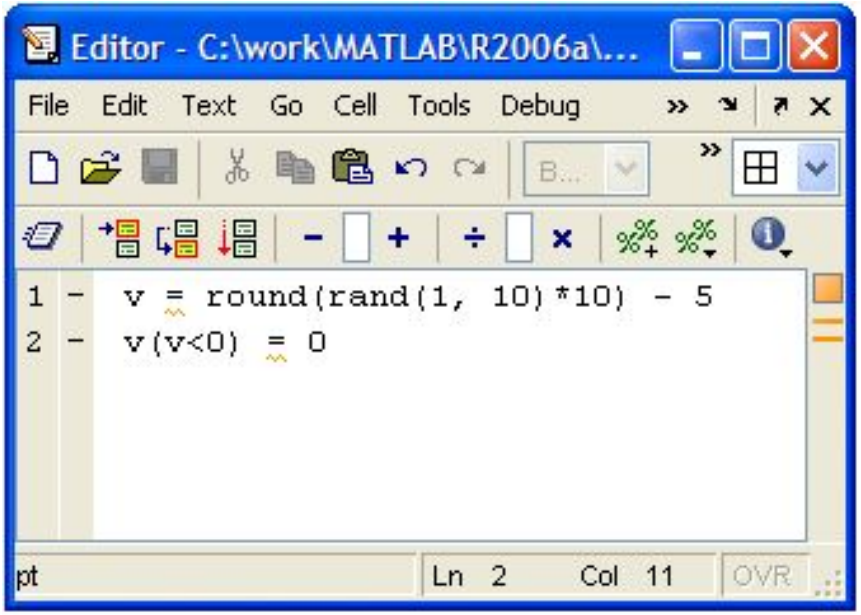
Замечание по использованию цикла с параметром

- Обычно цикл `for` используется для обработки массивов
- Важно помнить, что если есть возможность обойтись без этого цикла (применить матричные или векторные операции), то **лучше избавиться от явного цикла**
- В этом случае программа будет работать на порядок быстрее

Пример: замена отрицательных элементов вектора на нули (с циклом)



Пример: замена отрицательных элементов вектора на нули (без цикла)




```
v =

     2    -2     3     1    -1     2     0    -1     2     1


v =

     2     0     3     1     0     2     0     0     2     1
```

Цикл с предусловием

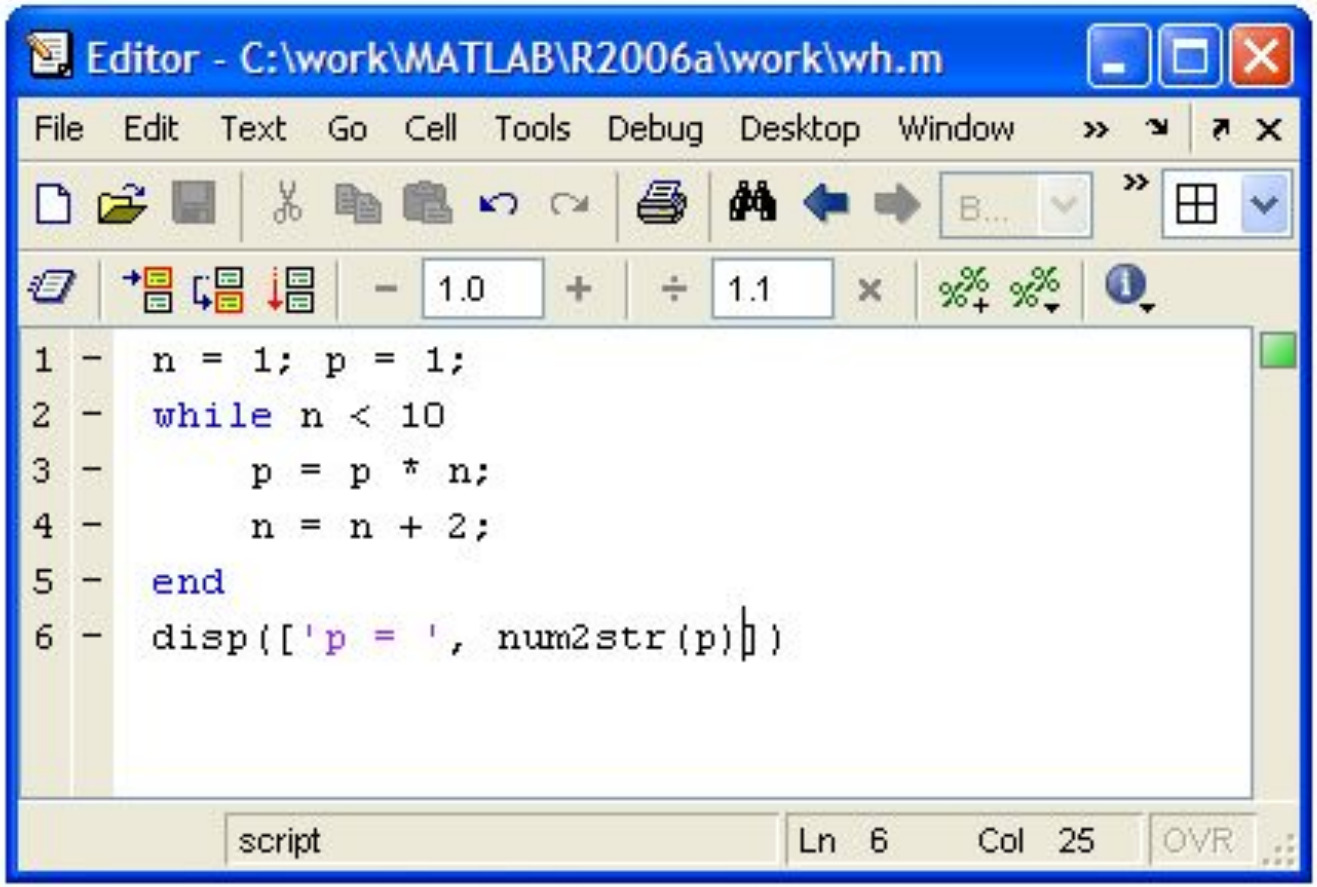
- Синтаксис:

```
while <логическое выражение>
    <операторы>
end
```

- Операторы выполняются, пока логическое выражение есть истина (true)

Цикл с предусловием (пример)

```
>> wh
p = 945
>>
```



Операторы `break` и `continue`

- Аналогичны одноимённым операторам Паскаля
- `Break` производит досрочный выход из цикла `for` или `while`
- `Continue` прекращает выполнение текущей итерации и переходит к следующей

Операторы `break` и `continue` (пример)

- Написать скрипт, который вводит с клавиатуры произвольное количество чисел. Если число положительное, то оно прибавляется к сумме, если отрицательное, то пропускается. Ноль – признак окончания работы

Операторы `break` и `continue` (решение)

```
Input a number: 1
Input a number: 2
Input a number: -3
Input a number: 3
Input a number: 0
s = 6
>>
```

```
 1 -  s = 0; p = 1;
 2 -  while p
 3 -      x = input('Input a number: ');
 4 -      if x == 0
 5 -          break
 6 -      end
 7 -      if x < 0
 8 -          continue
 9 -      end
10 -      s = s + x;
11 -  end
12 -  disp(['s = ', num2str(s)])
```