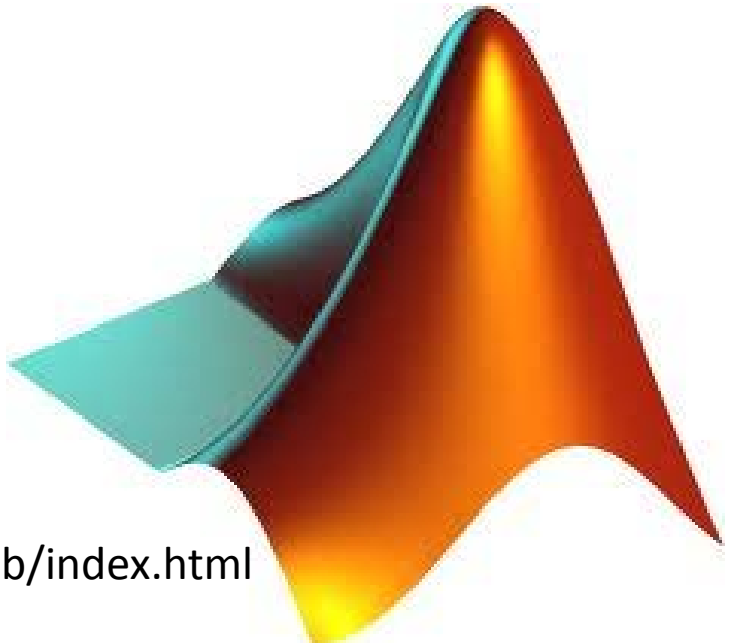
Matlab

Kap. II - Programmierung



http://www.pci.tu-bs.de/aggericke/Matlab/index.html

Eine Function ist eine Blackbox

Output ← *function* ← Input

Eine Funktion ist eine Blackbox.
Sie verbirgt den Code und den Arbeitsbereich and kommuniziert mit der Welt nur über die Ein- und Ausgabevariablen

2

Matlab

- **Einfachste Ein-/Ausgabe**

```
>> x = input(‘Zahl eingeben:‘);   % Einfache Eingabe
>> x                              % Einfache Ausgabe
x =                               % Für Eingabe x = 5
  5
>> sprintf('Zahl = %d\n',x)       % Formatierte Ausgabe
  Zahl = 5
>> disp(‘Hello World‘);           % Einfache Text Ausgabe
  Hello World
```

Matlab

- **Programmierung**
  - Bedingungen

```
if  BEDINGUNG
  ...Anweisungen
end
```

```
if BEDINGUNG
 ...Anweisungen werden ausgeführt, falls Bedingung erfüllt ist.
else
 ...Anweisungen werden ausgeführt, falls Bedingung NICHT erfüllt ist.
end
```

Matlab

- **Programmierung**
  - Bedingungen

```
if      1.BEDINGUNG
 ...Anweisungen werden ausgeführt, falls 1.Bedingung erfüllt ist.
elseif  2. Bedingung
 ...Anweisungen werden ausgeführt, falls 2.Bedingung erfüllt ist.
else
 ...Anweisungen werden ausgeführt, falls KEINE Bedingung erfüllt ist.
end
```

Matlab

- **Programmierung**
  - Bedingungen „switch“.

    Beispiel: s ist ein String und je nach Wert wird der entsprechende „case“ angesprungen und die dortigen Anweisungen ausgeführt (hier s=‘Meier‘)

```
switch  s
   case ‘Schulz‘
      ...Anweisungen werden ausgeführt, falls 1.Fall (also falls s=‘Meier‘ wäre)
   case ‘Meier‘
      ...Anweisungen werden ausgeführt, falls 2.Fall (was hier zutrifft)
    ....
    ....
   otherwise
      ...Anweisungen werden ausgeführt, falls KEIN Fall gefunden wurde;
                                                    z.B. Für s = ‘Charly‘
end
```

Matlab

- **Programmierung**
  - Schleifen
    - for (variablenname) = Von:Schrittweite:Bis
    - o
```
for i=1:5
   sprintf ('Round %d\n',i)
end
```
    - o
```
for i=5:-1:1
   sprintf ('Round %d\n',i)
end
```
    - o
```
for i=[1 4 2 1]
   sprintf ('Round %d\n',i)
end
```

Matlab

- **Programmierung**
  - Schleifen
    - while Bedingung,...Anweisungen,.....end

```
ii=1;                                % Startwert
 while ii<6                          % Bedingung
   sprintf ('Round %d\n',ii)         % Anweisung
   ii=ii+2;                          % nächster Wert
end
```

What does Matlab code look like?

A simple example:

```
a = 1
while length(a) < 10
a = [0 a] + [a 0]
end
```

which prints out Pascal's triangle:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

(with "a=" before each line).

Matlab

- **Programmierung**
  - **break**: Abbruch von Schleifen (auch für while-Schleife)

```
>> for i=1:10
>>     x=2*i
>>     if (x>5)
>>         break;          % bricht Schleife komplett ab
>>     end
>> end
```

  - **continue**: Überspringen aller Anweisungen bis zum Schleifenende

```
>> for i=1:10
>>     if (i==5)
>>         continue        % Spring ans Schleifenende
>>     end
>>     x=2*i               % Sprung ans Schleifenende;
>> end                     % keine Berechnung für i=5(x=10)
```

Matlab

- **Programmierung**
  - **try** und **catch**

```
>> x = 'A';
>> try                                          % Versuch ok?
>>    log(x)                                    % wohl kaum
>> catch
>>   disp('no logarithm of strings')            % Nimm dies
>> end
>> lasterr                                      % Fehlermeldung
```

Matlab

- **Spezielle Datentypen**
  - Arrays

```
matrix = ['Peter','Hans']
   PeterHans
matrix = ['Peter';'Hans']
   geht nicht!
```

Lösung

  - Cells

```
cell = {'Peter','Hans'}
m = char(cell)
   Peter
   Hans
```

```
>> cell(2,:)
    Hans
>> cell{2}
    Hans
```

Matlab

- **Noch mehr Datentypen**
  - Cells -> beliebiege Datentypen

```
>> cell = {[6,8,0],99,[1,2,3;4,5,6],'Charly','ist'}
>> cell(4:5)
   'Charly'   'ist'
>> cell2 = {12,cell,'Peter','Hans'}
>> c3 = cell2(3)
   'Peter'                    % Ergebnis ist vom Typ „Cell“ !
```

Matlab

- **Noch mehr Datentypen**
  - **Structures**

  Ähnliche Idee bei **Struct** wie bei **cell**

```
struct1.a = cell2;
struct1.b = rand(10,10);
struct1.a{2}{4}              % Zugriff auf Charly
struct1.a{2}{3}(2,:)         % Zugriff 2. Matrixzeile 4 5 6
struct1.b(4,3)               % Zugriff auf das Element
                               (4,3) der Zufallsmatrix
```

Matlab

- **Noch mehr Datentypen**

MATLAB Help:
handle = @functionname returns a handle to the specified MATLAB® function.
The handle class is the superclass for allclasses that follow handle semantics. A handle isa reference to an object. If you copy an object's handle, MATLAB® copiesonly the handle and both the original and copy refer to the same objectdata. If a function modifies a handle object passed as an input argument,the modification affects the original input object.

– Handles

A function handle is a callable association to a MATLAB function. It contains an association to that function that enables you to invoke the function regardless of where you call it from. This means that, even if you are outside the normal scope of a function, you can still call it if you use its handle.

With function handles, you can:

- Pass a function to another function
- Capture data values for later use by a function
- Call functions outside of their normal scope
- Save the handle in a MAT-file to be used in a later MATLAB session

```
>> a=0; b=5;
>> integral(@log, a, b)
   3.0472
>> fct = @sin;
>> integral(fct, a, b)
   0.7163
```

Builtin Functions

- **Exponential**
  - **exp(x)**
  - **expm(A)** **% Matrix exponential A = expm(logm(A))**
  - **sqrt(x)**
  - **sqrtm(A)** **% Matrix square root X*X= A**

- **Logarithmen**
  - **log(x)** **% natürlicher Logarithmus (*ln)* zur Basis e.**
  - **log10(x)** **% Logarithmus zur Basis 10.**
  - **log2(x)** **% Logarithmus zur Basis 2.**
  - **reallog** **% *ln* für positive Argumentwerte**
  - **logm(A)** **% Matrix logarithm: logm(expm(A)) = A**

Builtin continued

- **numeric**
  - round(x) % round to nearest integer
  - ceil(x) % round to nearest integer towards +∞
  - fix(x) % round to nearest integer towards 0
  - floor(x) % round to nearest integer towards -∞
  - sign(x) % +1, 0 or -1
  - rem(x,y) % finds remainder of x/y

- **complex**
  - complex(a,b) % makes a complex number: Z = a + b i
  - abs(z) % absolute value: sqrt(a^2+b^2)
  - angle(z) % complex plane: R=abs(z), theta=angle(z)
  - conj(z) % conjugate complex i -> -i
  - imag(z)
  - real(z)

Builtin continued

- **Trigonometrische Functionen und deren Inverse**
  - `cos(x)` `acos(x)`
  - `sin(x)` `asin(x)`
  - `tan(x)` `atan(x)`
  - `cot(x)` `acot(x)`
  - `csc(x)` `acsc(x)`
  - `sec(x)` `asec(x)`
  - `atan2(x,y)`

  mit einem “d” am Ende der Funktion erhält man die Grad-Werte;
  z.B. `cosd(90) = 0`

  Vierquadranten inv. tan im Interval [-pi,pi],

- **Hyperbolische Funktionen und deren Inverse**
  - `cosh(x)` `acosh(x)`
  - `coth(x)` `acoth(x)`
  - `csch(x)` `acsch(x)`
  - `sech(x)` `asech(x)`
  - `sinh(x)` `asinh(x)`
  - `tanh(x)` `atanh(x)`

Remarks

- All trigonometric functions require the use of radians and not degrees; however you can use these d-fcts, like tand, atand,…
- All builtin functions can also be used with arrays of any dimension.
- All builtin functions can also be used with complex numbers.
- To understand the meaning with complex numbers go back to basic definitions:
  - exp(i x) = cos(x) + i sin(x)
  - sin(x) =(exp(i x) - exp(-i x))/(2 i)
  - sinh(x) = (exp(x) – exp(-x))/2
- Basic example
  - exp(a + i b) = exp(a) exp(i b) = exp(a) (cos(x) + i sin(x))

- Since a trigonometric function is periodic each value has many inverse values Matlab will return only one value, the so called 'principle value‘.
- For engineering applications it is important to check that this is the correct value.
- Plotting a complex valued function z =f(x+iy) is **not** possible. It requires a four dimensional space (x,y,u,v) since z = u+iv

Builtin Functions for vectors

- **max(x)**
  - returns largest value in vector x
- **[a,b] = max(x)**
  - returns largest value in a and index where found in b
- **max(x,y)**
  - x and y arrays of same size, returns vector of same length with larger value from corresponding positions in x and y
- **min(x,y)**
  - same type of functions are available for

Builtin functions for vectors

- **sort(X)**
- **mean(X)**
- **std(X)**
- **var(X)**
- **median(X)**
- **sum(X)**
- **prod(X)**
- **cumsum(X)**
  - returns vector of same size with cumulative sum of x, i.e. x=[4,2,3] returns [4,6,9]
- **cumprod(X)**
  - returns cumulative products in vector of same size, i.e. [4 8 24]

Builtin functions applied to matrices

- Matrices (arrays) are stored in column major form
- When builtin functions for vectors are applied to a matrix function operates on columns and returns a row vector:

```
>> A = [1 2 3; 4 5 6]
>> sum(A)
>>   5  7  9
```

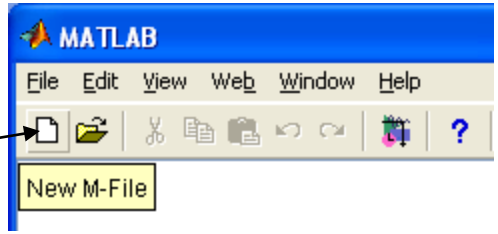
- Some special functions for matrices (arrays) are

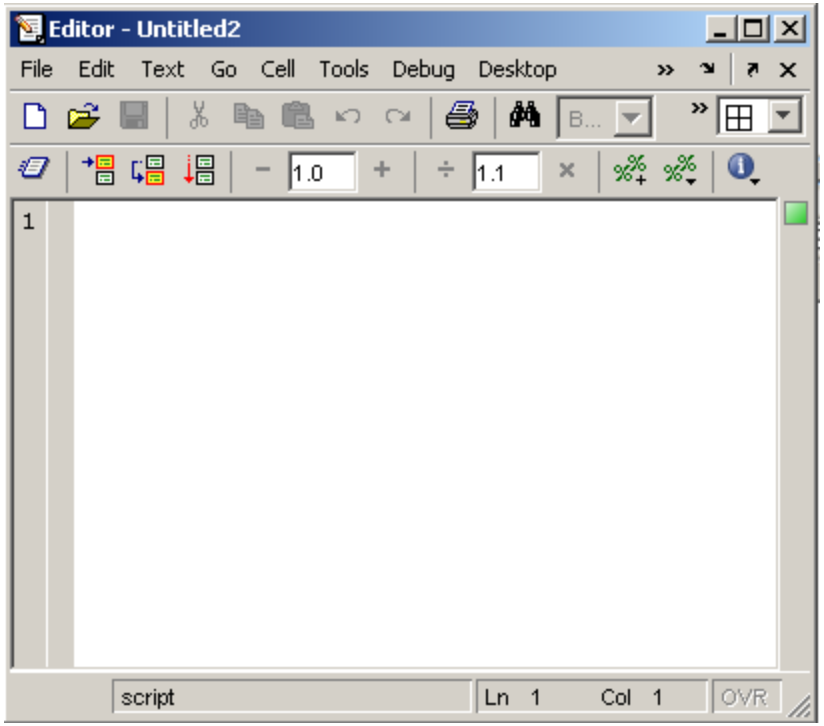
```
>> B = reshape(A,[1,3,2])      % reshape(A,[m n p ...]) returns old A but reshaped to have
                               % the new size m-by-n-by-p-by-....
>> A = squeeze(B)              % all singleton dimensions are removed

>> permute(A,[2 1])            % permute(A,order) rearranges the dimensions of A so that
                                  they are in the order specified by the vector order.
>> shiftdim(A,1)               % shiftdim(A,n) shifts the dimensions of the array A by N
```

Matlab: use of M-File

Click to create a new M-File

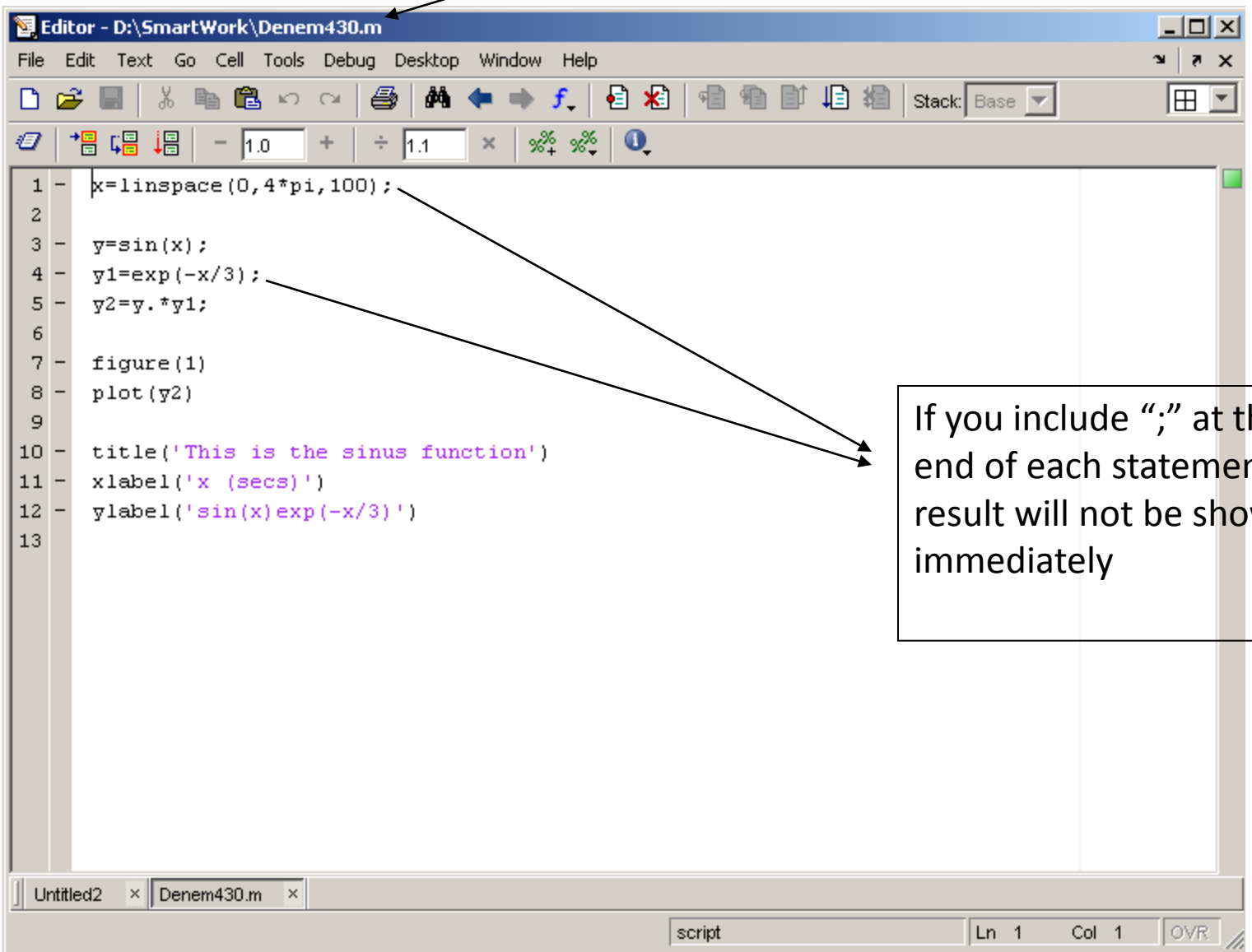




- Extension ".m"
- A text file containing script or function or program to run

Matlab: use of M-File

Saved file as *Denem430*.m



If you include ";" at the end of each statement, result will not be shown immediately

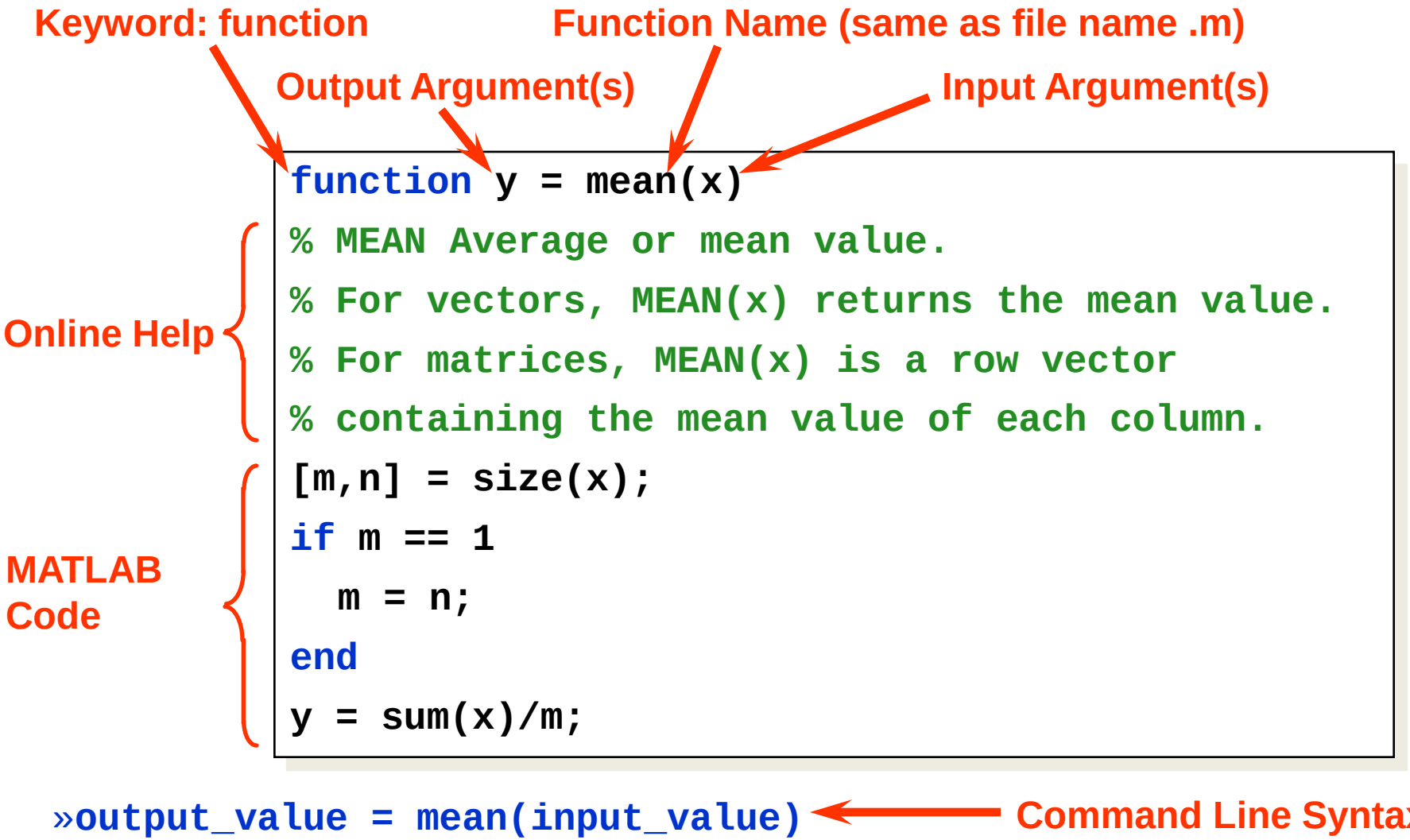
Matlab: User Defined Functions

- Functions are m-files which can be executed by specifying some inputs and supply some desired outputs.
- The code telling the Matlab that an m-file is actually a function is

```
function out1=functionname(in1)
function out1=functionname(in1,in2,in3)
function [out1,out2]=functionname(in1,in2)
```

- You should write this function declaration command at the *beginning of the m-file* and you should save the m-file with a *file name same as the function name.*

Structure of a Function M-file



Multiple Input & Output Arguments

```
function r = ourrank(X,tol)
% OURRANK Rank of a matrix
s = svd(X);
if nargin==1
   tol = max(size(X))*s(1)*eps;
end
r = sum(s>tol);
```

Multiple Input Arguments (,)

Multiple Output Arguments [,]

```
function [mean,stdev] = ourstat(x)
% OURSTAT Mean & std. deviation
[m,n] = size(x);
if m==1
   m = n;
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m – mean.^2);
```

```
» RANK = ourrank(rand(5),0.1);
» [MEAN,STDEV] = ourstat(1:99);
```

nargin, nargout, nargchk

- ***nargin*** – number of input arguments

  - Many of Matlab functions can be run with different number of input variables.

```
if nargin==0
    disp('you have to send input arguments!');
end
```

- ***nargout*** – number of output arguments

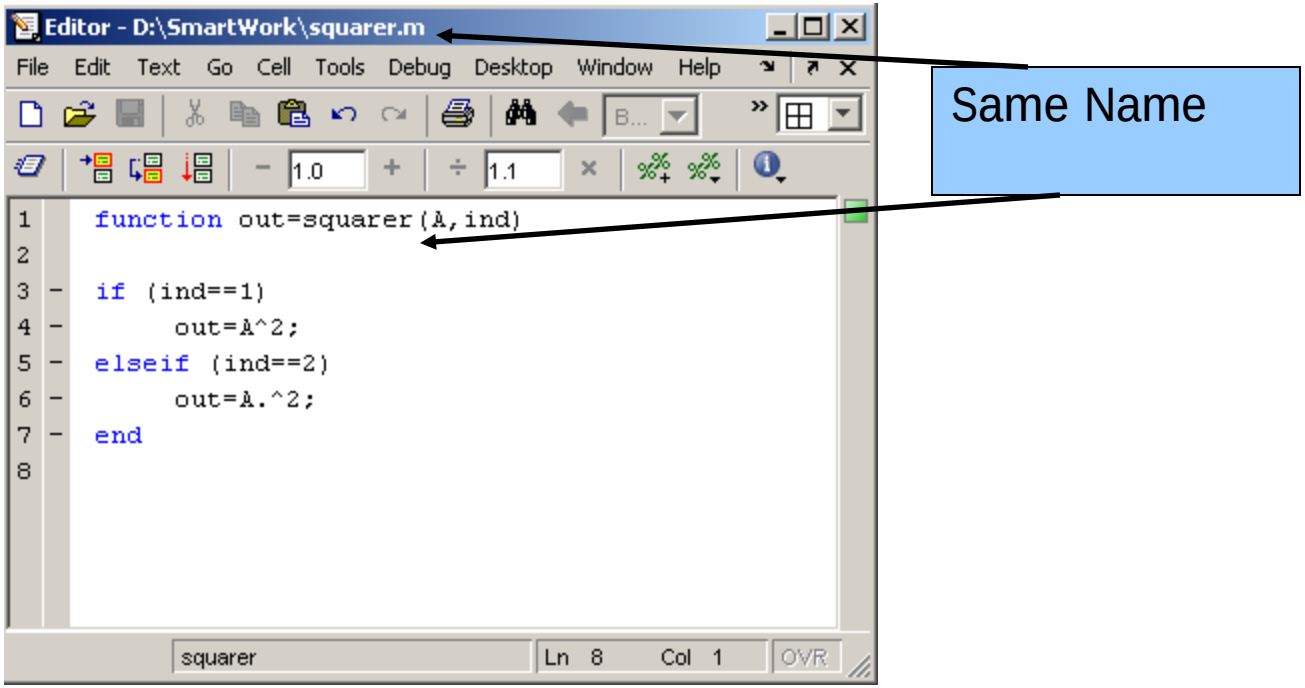
  - efficiency

- ***nargchk*** – check if number of input arguments is between some ‘low’ and ‘high’ values

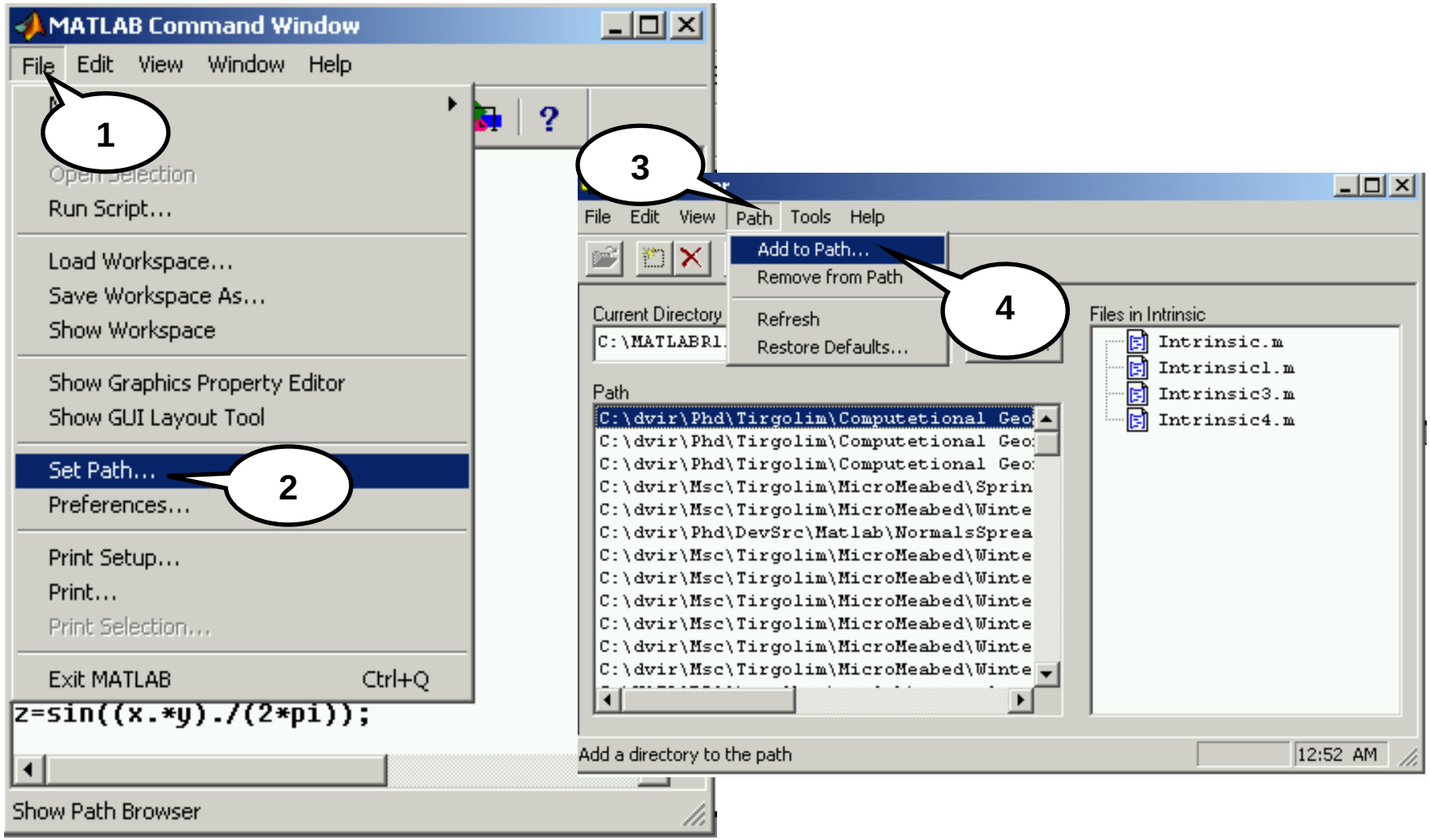
```
IsOk=nargchk(1,5,nargin)
```

Matlab: User Defined Functions

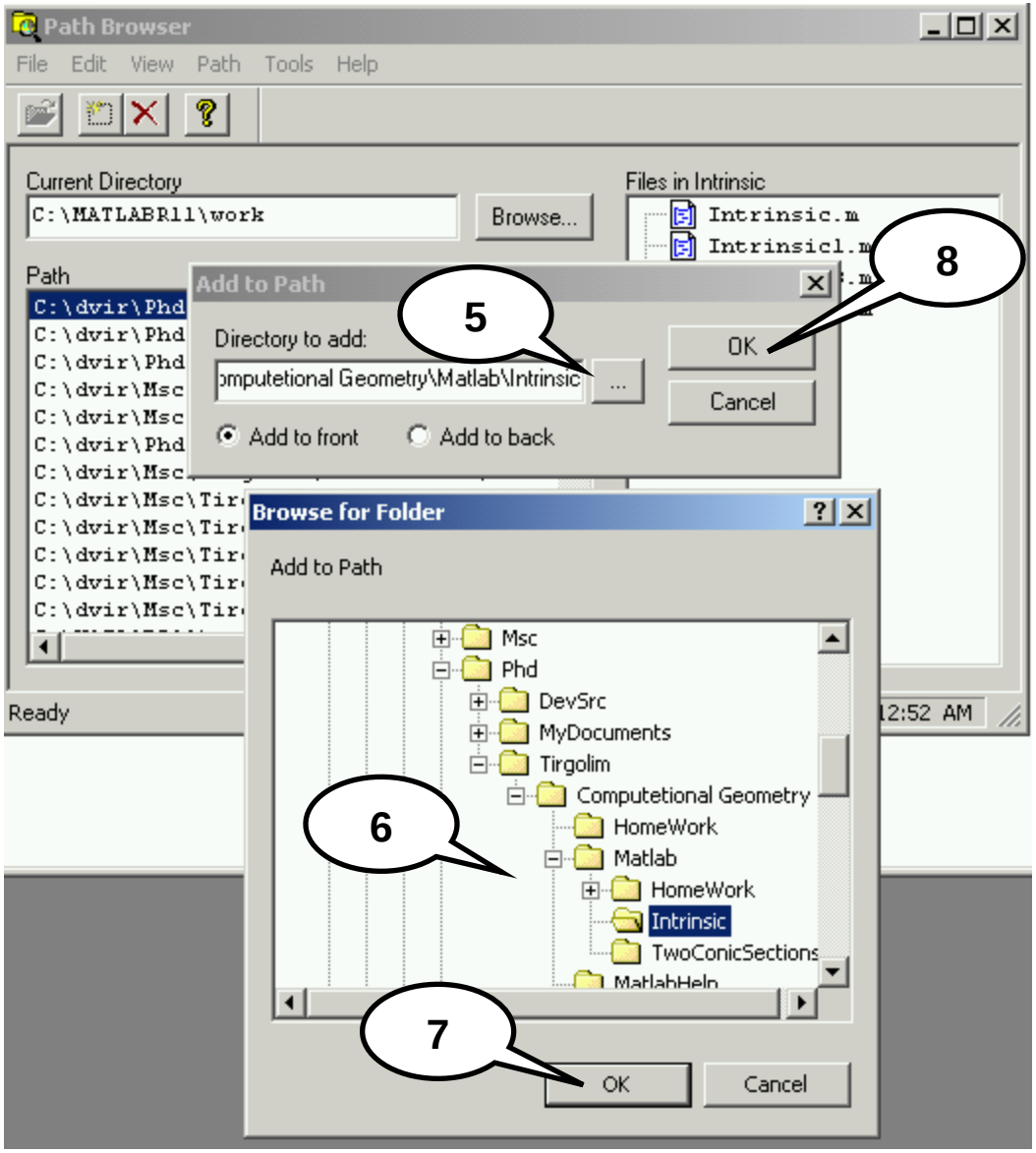
- Examples
  - Write a function : out=squarer (A, ind)
    - Which takes the square of the input matrix if the input indicator is equal to 1
    - And takes the element by element square of the input matrix if the input indicator is equal to 2



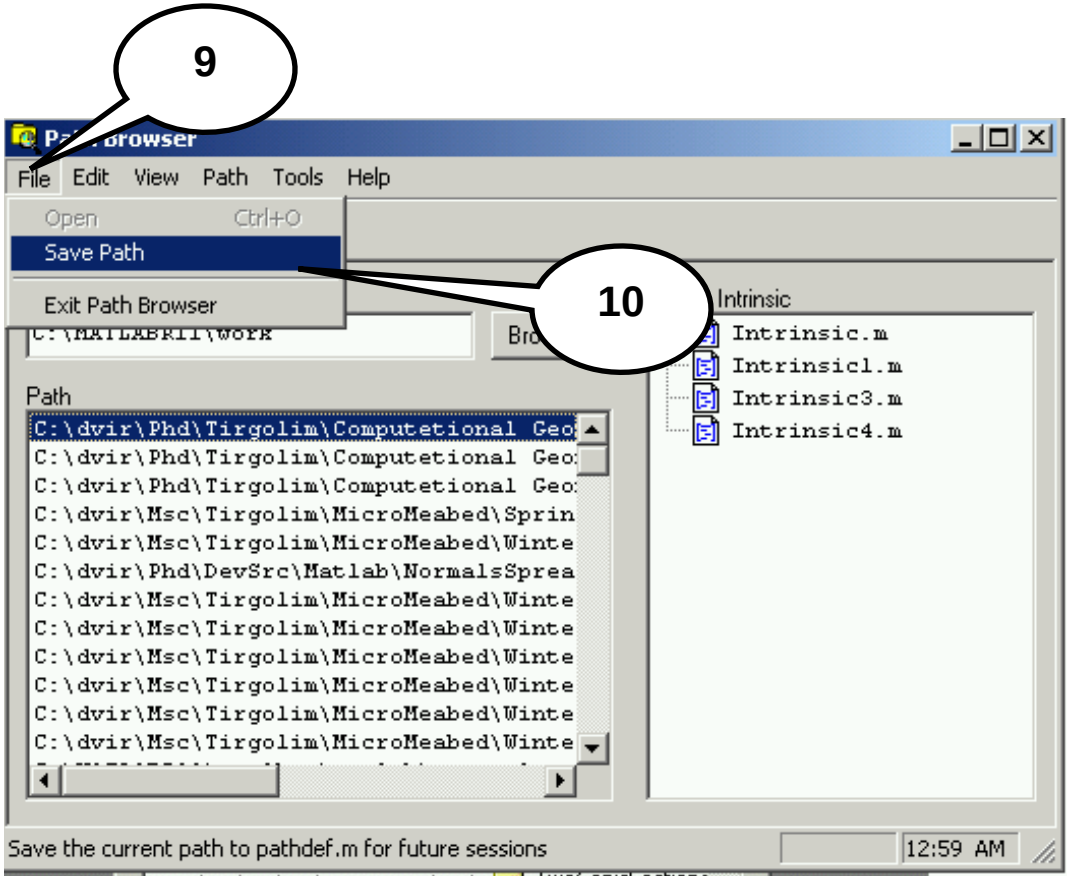
Adding a path to a library



Adding a path to a library



Adding a path to a library



Exercise

- Write a function file to compute the factorial of a number.
- Input: N
- Output: NF
- Function name: factorial

Script file to compute factorial

```
% program to calculate the factorial of a number
% input  N : an integer
% if N is not an integer the program obtains the
% factorial of the integer part of N
% output FC : the factorial of N
%
FC=1;                          % initial value of FC
   for  i=1:N
      FC=FC*i;                 % n! =(n-1)!*n
   end
```

Comments are used to explain MATLAB statements

A solution for a function

output | Function name | input

First statement **must** start with 'function'

```
function  [FC] = factorial(N)
FC = 1;
   for  i=1:N
      FC = FC*i;
   end
```

Save the program using 'factorial' as a name

- Save it in directory recognized by MATLAB
- If the directory is not recognized by MATLAB add it to the MATLAB path

A Better one

These comments will be displayed when

'help factorial'

is typed

```
function  [FC]=factorial(N)
% [FC]=factorial(N)
% program to calculate the factorial of a number
% input  N : an integer
% if N is not an integer the program obtains the
% factorial of the integer part of N
% output FC : the factorial of N

FC = 1;                    % initial value of FC
for  i=1:N
  FC = FC*i;               % n! =(n-1)!*n
end
```

Comments are used to explain MATLAB statements

even better …

factorial.m

```
function  [FC]=factorial(N)
% [FC]=factorial(N)
% program to calculate the factorial of a number
% input  N : an integer
% if N is not an integer the program obtains the
% factorial of the integer part of N
% output FC : the factorial of N

if  nargin<1                % Check for correct input
  error('No input argument assigned')
elseif  N<0
  error('Input must be non-negative')
elseif  abs(N-round(N))>eps
  error('Input must be an integer')
end

FC = 1;                     % initial value of FC
for  i=1:N
  FC = FC*i;              % n! =(n-1)!*n
end
```

Or quite different…

factorial.m

```
function  [FC]=factorial(N)
% [FC]=factorial(N)
% program to calculate the factorial of a number
% input  N : an integer
% output FC : the factorial of N

if  nargin<1                          % Input ok?
  error('No input argument assigned')
elseif  N<0
  error('Input must be non-negative')
elseif  abs(N-round(N))>eps
  error('Input must be an integer')
end

if N==0, FC = 1;                      % 0!=1
else     FC = N*factorial(N-1);       % n!=n*(n-1)!
end
```

MATLAB Calling Priority

High

low

- variable
- built-in function
- subfunction
- private function
- MEX-file
- P-file
- M-file

```
» cos='This string.';
» cos(8)
ans =
   r

» clear cos
» cos(8)
ans =
   -0.1455
```

Matlab Notes:

- "%" is the neglect sign for Matlab (equaivalent of "//" in C). Anything after it on the same line is neglected by Matlab compiler. Used for comments.
- Sometimes slowing down the execution is done deliberately for observation purposes. You can use the command "pause" or "pause(seconds)" for this purpose.

```
>> pause          % wait until any key is pressed
>> pause(3.4)     % wait 3.4 seconds
```

Flow control – conditional repetition

- Solutions to nonlinear equations

$$f(x) = 0$$

- can be found using Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- **Task**: write a function that finds a solution to

$$f(x) = e^{-x} - \sin(x)$$

- Given x_0, iterate `maxit` times or until $|x_n - x_{n-1}| \leq \text{tol}$

Flow control – conditional repetition

newton.m

```
function [x,n] = newton(x0,tol,maxit)
% NEWTON – Newton's method for solving equations
% [x,n] = NEWTON(x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
  n = n + 1;
  x_new = x - (exp(-x)-sin(x))/(-exp(-x)-cos(x));
  done = (n>=maxit) | ( abs(x_new-x)<tol );
  x=x_new;
end
```

- >> [x,n] = newton(0,1e-3,10)

Function functions

- Do we need to re-write `newton.m` for every new function?

- **No!** General purpose functions take other m-files as input.

Function handle

- Useful as parameter to other functions
- Can be considered as an alternate name for a function – but with more capabilities
- Example:

```
>> sin_handle = @sin            % “@” converts function to handle
>> sin_handle(x)                % same values as sin(x) for all x
>> integral(sin_handle,0,pi);   % or integral(@sin,0,pi);

>> f = input('Functionsname: ','s')
>> fc = [‘@’ f]                 % add “@” to function name
>> fct = eval(fc)               % evaluate the string

>> fct = str2func(f)            % replaces the 2 former commands

>> integral(fct,0,pi)           % integrate the function
```

Using anonymous functions

- Another use is anonymous functions
- Assume the user needs to work temporarily with the function
  $x^3+3*x-1$
- Instead of writing the function
  - function y = mypoly(x) ;
  - y = x.^3+3*x-1
- and storing it as mypoly.m in subdirectory work we can use an anonymous function with the function handle mypoly
  - **mypoly = @(x) x.^3+3*x-1**

Using anonymous functions

- With a function handle an anonymous function can be used like any other
  - **integral(mypoly, 0, pi)**
- Without the function handle the anonymous function can be inserted directly as the parameter
  - **integral(@(x) x.^3+3*x-1, 0, pi)**

Function functions

- Back to our question: do we need to re-write `newton.m` for every new function?
- No! General purpose functions take other m-files as input.
- `>> help feval`
- `>> [f,f_prime]=feval('myfun',0);`

myfun.m

```
function [f,f_prime] = myfun(x)
% MYFUN- Evaluate f(x) = exp(x)-sin(x)
% and its first derivative
% [f,f_prime] = myfun(x)

f=exp(-x)-sin(x);
f_prime=-exp(-x)-cos(x);
```

Function functions

- Can update `newton.m`

newtonf.m

```
function [x,n] = newtonf(fname,x0,tol,maxit)
% NEWTON – Newton's method for solving equations
% [x,n] = NEWTON(fname,x0,tol,maxit)
x = x0; n = 0; done=0;
while ~done,
   n = n + 1;
   [f,f_prime] = feval(fname,x);
   x_new = x – f/f_prime;
   done = n>maxit  |  abs(x_new-x)<tol ;
   x = x_new;
end
```

- `>> [x,n]=newtonf('myfun',0,1e-3,10)`

Function functions in Matlab

- Heavily used: integration, differentiation, optimization, …

  >> help ode45

- Find the solution to the ordinary differential equation

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_1 + 0.1(1 - x_1^2)x_2$$

myodefun.m

- ```
  function x_dot = myodefun(t,x)
  % MYODEFUN – Define RHS of ODE
  x_dot(1,1) = x(2);
  x_dot(2,1) = -x(1)+0.1*(1-x(1)^2)*x(2);
  ```

>> ode45('myodefun',[0 10],[1;-10]);
  ```

Some more special functions

- **feval(fhandle, x1,..., xn)** evaluates the function handle, fhandle, using arguments x1 through xn.
- **eval(expression)** evaluates (transforms) the string expression in MATLAB code
- **str2func('str')** returns a function handle for the function named in string 'str'.
- **func2str(fhandle)** constructs a string that holds the name of the function to which the function handle fhandle belongs (O-Ton MATLAB).
- x=2; **exist('x', 'var')** Here: does the variable x exist? (1)
- is* Help of Matlab lists all is* functions
- **Isa(Variable,Class)** gehört die Varibale zur Klasse? isa(pi,'integer')

Programming tips and tricks

- Programming style has huge influence on program speed!

slow.m

```
tic;
X=-250:0.1:250;
for ii=1:length(x)
  if x(ii)>=0,
    s(ii)=sqrt(x(ii));
  else
    s(ii)=0;
  end;
end;
toc
```

fast.m

```
tic
x=-250:0.1:250;
s=sqrt(x);
s(x<0)=0;
toc;
```

Loops are slow: **Replace loops by vector operations**!

Memory allocation takes a lot of time: **Pre-allocate memory**!

Use profile to find code bottlenecks!