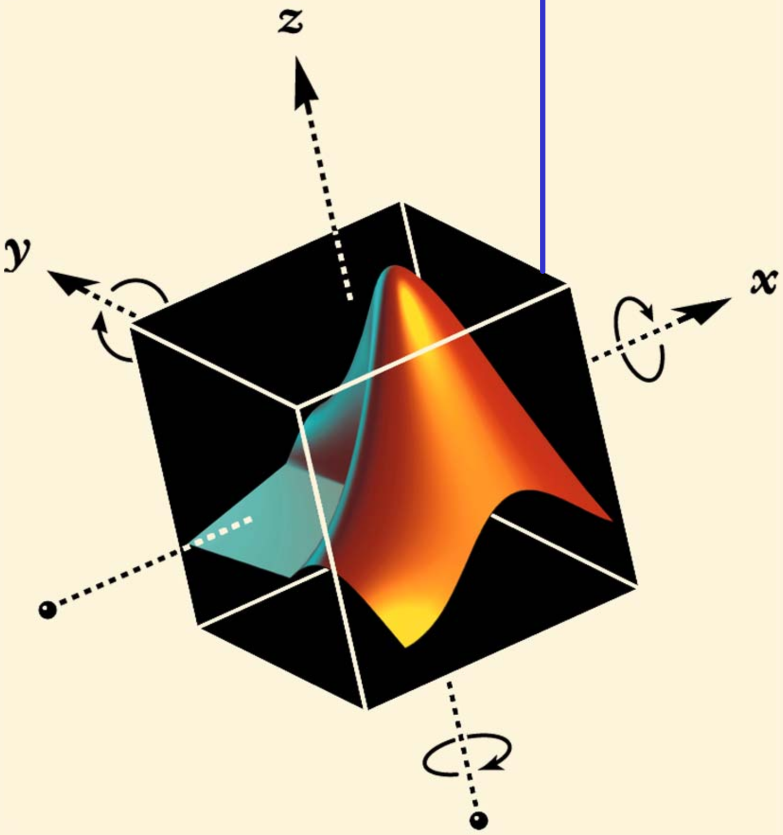
2E1215, Lecture 2 – Matlab Programming

http://www.s3.kth.se/control/kurser/2E1215/



Mikael Johansson and Frank Lingelbach
Department of Signals, Sensors, and Systems

Today’s Lecture

Matlab programming

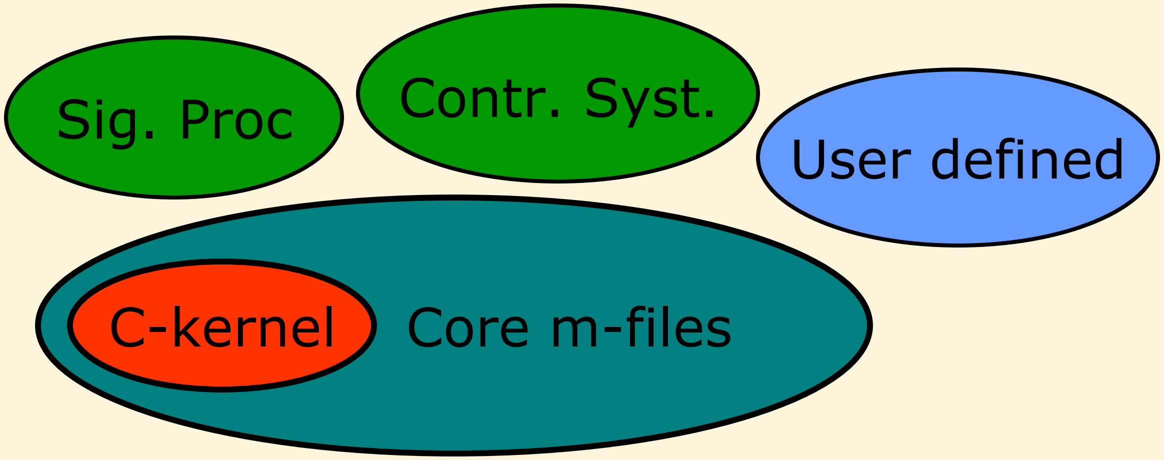
- Programming environment and search path
- M-file scripts and functions
- Flow control statements
- Function functions
- Programming tricks and tips

Matlab environment

Matlab construction

- Core functionality as compiled C-code, m-files
- Additional functionality in toolboxes (m-files)

Today: Matlab programming (construct own m-files)



The programming environment

The working directory is controlled by

```
>> dir
>> cd catalogue
>> pwd
```

The path variable defines where matlab searches for m-files

```
>> path
>> addpath
>> pathtool
>> which function
```

The programming environment

Matlab can’t tell if identifier is variable or function

```
>> z=theta;
```

Matlab searches for identifier in the following order

1. variable in current workspace
2. built-in variable
3. built-in m-file
4. m-file in current directory
5. m-file on search path

Note: m-files can be located in current directory, or in path

Script files

Script-files contain a sequence of Matlab commands

factscript.m

```
%FACTSCRIPT – Compute n-factorial, n!=1*2*...*n
y = prod(1:n);
```

- Executed by typing its name

  `>> factscript`

- Operates on variables in global workspace
  - Variable `n` must exist in workspace
  - Variable `y` is created (or over-written)
- Use comment lines (starting with %) to document file!

Displaying code and getting help

To list code, use `type` command

```
>> type factscript
```

The `help` command displays first consecutive comment lines

```
>> help factscript
```

Functions

Functions describe subprograms

- Take inputs, generate outputs
- Have local variables (invisible in global workspace)

```
[output_arguments]= function_name(input_arguments)
% Comment lines
<function body>
```

factfun.m

```
function [z]=factfun(n)
% FACTFUN - Compute factorial
% Z=FACTFUN(N)

z = prod(1:n);
```

```
>> y=factfun(10);
```

Scripts or function: when use what?

Functions

- Take inputs, generate outputs, have internal variables
- Solve general problem for arbitrary parameters

Scripts

- Operate on global workspace
- Document work, design experiment or test
- Solve a very specific problem once

Exam: all problems will require you to write functions

facttest.m

```
% FACTTEST - Test factfun

N=50;
y=factfun(N);
```

Flow control - selection

The if-elseif-else construction

```
if <logical expression>
   <commands>
elseif <logical expression>
   <commands>
else
   <commands>
end
```

```
if height>170
   disp('tall')
elseif height<150
   disp('small')
else
   disp('average')
end
```

Logical expressions

Relational operators (compare arrays of same sizes)

==	(equal to)	~=	(not equal)
<	(less than)	<=	(less than or equal to)
>	(greater than)	>=	(greater than or equal to)

Logical operators (combinations of relational operators)

&	(and)
|	(or)
~	(not)

Logical functions

- xor
- isempty
- any
- all

```
if (x>=0) & (x<=10)
  disp(‘x is in range [0,10]’)
else
  disp(‘x is out of range’)
end
```

Flow control - repetition

Repeats a code segment a fixed number of times

```
for index=<vector>
   <statements>
end
```

The <statements> are executed repeatedly.
At each iteration, the variable `index` is assigned
a new value from `<vector>`.

```
for k=1:12
   kfac=prod(1:k);
   disp([num2str(k),’ ‘,num2str(kfac)])
end
```

Example – selection and repetition

fact.m

```
function y=fact(n)
% FACT – Display factorials of integers 1..n
if nargin < 1
   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

for k=1:n
   kfac=prod(1:k);
   disp([num2str(k),’ ’,num2str(kfac)])
   y(k)=kfac;
end;
```

Repetition: Animation demo

The function movie replays a sequence of captured frames

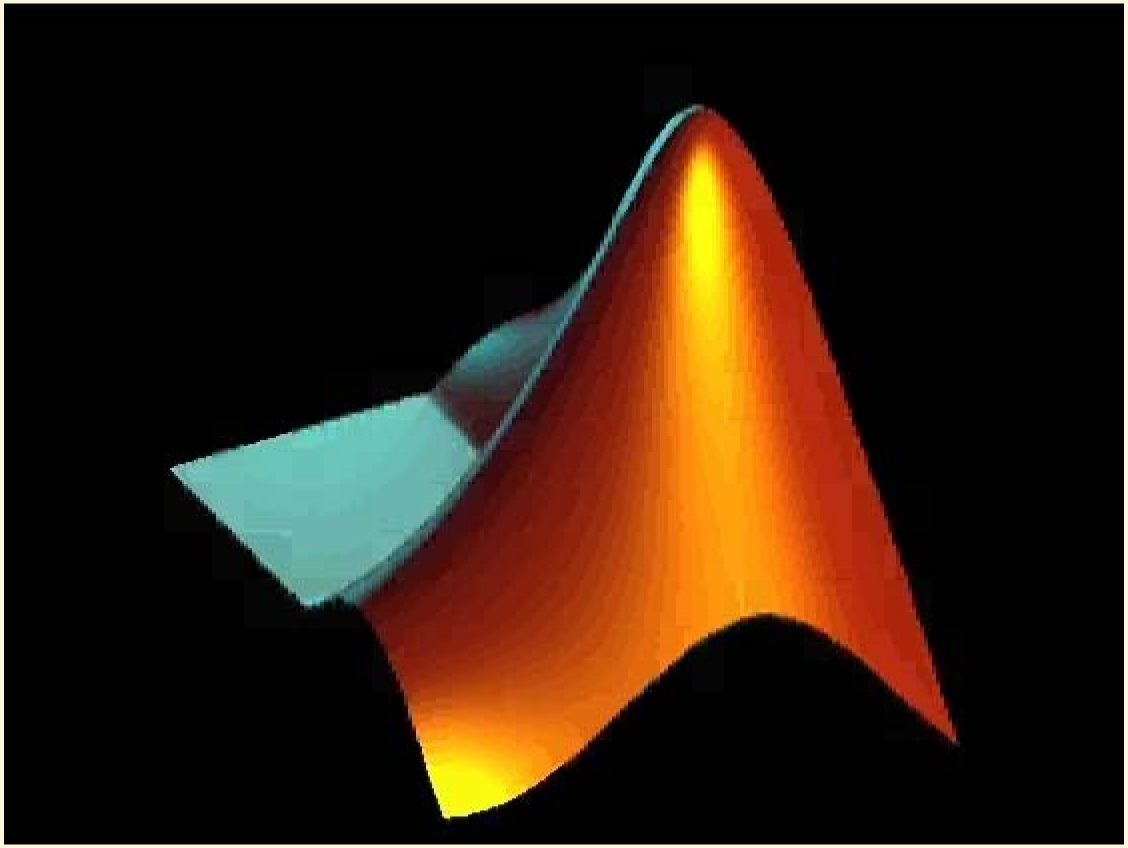
Construct a movie of a 360° tour around the Matlab logo

logomovie.m

```
% logomovie - make movie of 360 degree logo tour
logo;
no_frames=40;
dtheta=360/no_frames;
for frame = 1:no_frames,
  camorbit(dtheta,0)
  M(frame) = getframe(gcf);
end
```

Animation demo

```
>> movie(gcf,M)
```



Flow control – conditional repetition

while-loops

```
while <logical expression>
   <statements>
end
```

`<statements>` are executed repeatedly as long as the `<logical expression>` evaluates to true

```
k=1;
while prod(1:k)~=Inf,
    k=k+1;
end
disp([‘Largest factorial in Matlab:’,num2str(k)]);
```

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.

```
>> 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]);
```

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!

Summary

User-defined functionality in m-files

- Stored in current directory, or on search path

Script-files vs. functions

- Functions have local variables,
- Scripts operate on global workspace

Writing m-files

- Header (function definition), comments, program body
- Have inputs, generate outputs, use internal variables
- Flow control: ”if...elseif...if”, ”for”, ”while”
- General-purpose functions: use functions as inputs

Programming style and speed

- Vectorization, memory allocation, profiler