FF505

Computational Science

# Lecture 4
# Programming: Scripts and Functions

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

# Resume

- Overview to MATLAB environment
- Overview of MATLAB programming and arrays
- Solving linear systems in MATLAB

- Large sparse matrices and performance comparison
- Arrays
- Mathematical functions

- Programming: Control Flow
- Graphics: 2D and 3D

Free alternatives to Matlab: python, Scilab, Octave

# Today

1. Functions
2. Data Types
3. Random Number Generators
4. Exercises
   - Monte Carlo Simulation
   - Function Arguments
   - Improving Performance
   - Stochastic Matrices

# Script and Function Files (M-Files)

Script file

```
x=(1:1000)';
for k=1:5
  y(:,k)=k*log(x);
end
plot(x,y)
```

command line `simple`
does not take arguments
operates on data in the workspace

Function file

```
function y=simpl(maxLoop)
       % (smart indent)
       x=(1:1000)';
       for k=1:maxLoop
       y(:,k)=k*log(x);
       end
       plot(x,y)
```

command line `g=simple(10)`
can take input arguments and return output arguments.
Internal variables are local to the function

Same name conventions for `.m` files as for variables.
Check if variables or functions are already defined.

```
exist("example1")
exist("example1.m","file")
exist("example1","builtin")
```

```
type fun
```

# Script and Function Files (M-files)

- Modularize
- Make interaction clear
  make functions interact via arguments (in case structures) rather than via global variables
- Partitioning
- Use existing functions
  (`http://www.mathworks.com/matlabcentral/fileexchange`)
- Any block of code appearing in more than one m-file should be considered for packaging as a function
- Subfunctions
  packaged in the same file as their functions
- Test scripts

# Outline

# User-Defined Functions

The first line in a function file distinguishes a function M-file from a script M-file. Its syntax is as follows:

```
function [output variables] = name(input variables)
```

The function name should be the same as the file name in which it is saved (with the .m extension).

Example

```
function z = fun(x,y)
% the first line of comments is accessed by lookfor
% comments immediately following the definition
% are shown in help
u = 3*x;
z = u + 6*y.^2;
```

```
q = fun(3,7)
q =
   303
```

$\rightsquigarrow$ variables have local scope

# Variable Scope

Local Variables: do not exist outside the function.

```
function z = fun(x,y)
u = 3*x;
z = u + 6*y.^2;
```

```
>> x = 3; y = 7;
>> q = fun(x,y);
>> x
x =
3
>> y
y =
7
>> u
??? Undefined function or variable 'u'.
```

The variables x, y and u are local to the function fun

# Local Variables

Local variables do not exist outside the function

```
>>x = 3;y = 7;
>>q = fun(x,y);
>>x
x =
3
>>y
y =
7
>>u
??? Undefined function or variable 'u'.
```

# Local Variables

Variable names used in the function definition may, but need not, be used when the function is called:

In fun.m

```
function z = fun(x,y)
x=x+1; %we increment x but x is local and
       will not change globally
z=x+y;
```

At prompt

```
>> x=3;
>> z=fun(x,4)
>> x
x =
     3
```

All variables inside a function are erased after the function finishes executing, except when the same variable names appear in the output variable list used in the function call.

# Global Variables

The `global` command declares certain variables global: they exist and have the same value in the basic workspace and in the functions that declare them global.

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

```
>> global GRAVITY
>> GRAVITY = 32;
>> y = falling((0:.1:5)');
```

Programming style guidelines recommend avoiding to use them.

# Parameters and Arguments

Arguments passed by position Only the order of the arguments is important, not the names of the arguments:

```
>> x = 7; y = 3;
>> z = fun(y, x)
z =
   303
```

The second line is equivalent to z = fun(3,7).

Inside the function variables `nargin` and `nargout` tell the number of input and output arguments involved in each particular use of the function

One can use arrays as input arguments:

```
>> r = fun(2:4,7:9)
r =
   300 393 498
```

A function may have no input arguments and no output list.

```
function show_date
clear
clc
today = date
```

# Function Handles

- A function handle is an address to reference a function.
- It is declared via the @ sign before the function name.
- Mostly used to pass the function as an argument to another function.

```
function y = f1(x)
y = x + 2*exp(-x) - 3;
```

```
>> plot(0:0.01:6, @f1)
```

# Example: Finding zeros and minima

`X = FZERO(FUN,X0)` system function with syntax:

```
fzero(@function, x0) % zero close to x0
fminbnd(@function, x1, x2) % min between x1 and x2
```

```
fzero(@cos,2)
ans =
   1.5708
>> fminbnd(@cos,0,4)
ans =
   3.1416
```

Ex: plot and find the zeros and minima of $y = x + 2e^x - 3$

To find the minimum of a function of more than one variable

```
fminsearch(@function, x0)
```

where `@function` is a the handler to a function taking a vector and $x_0$ is a guess vector

# Other Ways

```
>> fun1 = 'x.^2-4';
>> fun_inline = inline(fun1);
>> [x, value] = fzero(fun_inline,[0, 3])
```

```
>> fun1 = 'x.^2-4';
>> [x, value] = fzero(fun1,[0, 3])
```

```
>>[x, value] = fzero('x.^2-4',[0, 3])
```

# Types of User-Defined Functions

- The primary function is the first function of an M-file. Other are subroutines not callable.
- Subfunctions placed in the file of the primary function, not visible outside the file
- Nested functions defined within another function. Have access to variables of the primary function.
- Anonymous functions at the MATLAB command line or within another function or script

```
% fhandle = @(arglist) expr
>> sq = @(x) (x.^2)
>> poly1 = @(x) 4*x.^2 - 50*x + 5;
>> fminbnd(poly1, -10, 10)
>> fminbnd(@(x) 4*x.^2 - 50*x + 5, -10, 10)
```

- Overloaded functions are functions that respond differently to different types of input arguments.
- Private functions placed in a `private` folder and visible only to parent folder

# Debugging

Two types of errors: (i) syntax errors (ii) runtime errors

- test on small examples whose result can be verified by hand
- display intermediate calculations
- use the debugger (not needed in this course)
- modus tollens/pones (remove/add pieces of code)

# Programming Style

- Document your scripts:
  - author and date of creation
  - what the script is doing
  - which input data is required
  - the function that the user has to call
  - definitions of variables used in the calculations and units of measurement for all input and all output variables!
- Organize your script as follows:
  1. input section (input data and/or input functions)
     Eg: `x=input("give me a number")`, `input("enter a key",'s')`
  2. calculation section
  3. output section (functions for displaying the output on the screen or files)
     Eg: `display(A)`, `display("text")`

# Example

```
% Program M3eP32.m
% Program Falling_Speed.m: plots speed of a falling object.
% Created on March 1, 2009 by W. Palm III
%
% Input Variable:
% tfinal = final time (in seconds)
%
% Output Variables:
% t = array of times at which speed is computed (seconds)
% v = array of speeds (meters/second)
%
% Parameter Value:
g = 9.81; % Acceleration in SI units
%
% Input section:
tfinal = input('Enter the final time in seconds:');
%
% Calculation section:
dt = tfinal/500;
t = 0:dt:tfinal; % Creates an array of 501 time values.
v = g*t;
%
% Output section:
plot(t,v),xlabel('Time (seconds)'),ylabel('Speed (meters/second)')
```

# Recall: Getting Help

- `help funcname`: Displays in the Command window a description of the specified function `funcname`.
- `lookfor topic`: Looks for the string topic in the first comment line (the H1 line) of the HELP text of all M-files found on MATLABPATH (including private directories), and displays the H1 line for all files in which a match occurs.
  Try: `lookfor imaginary`
- `doc funcname`: Opens the Help Browser to the reference page for the specified function `funcname`, providing a description, additional remarks, and examples.

# Documentation

Effective documentation can be accomplished with the use of

- Proper selection of variable names to reflect the quantities they represent.
- Use of comments within the program.
- Use of structure charts.
- Use of flowcharts.
- A verbal description of the program, often in pseudocode.

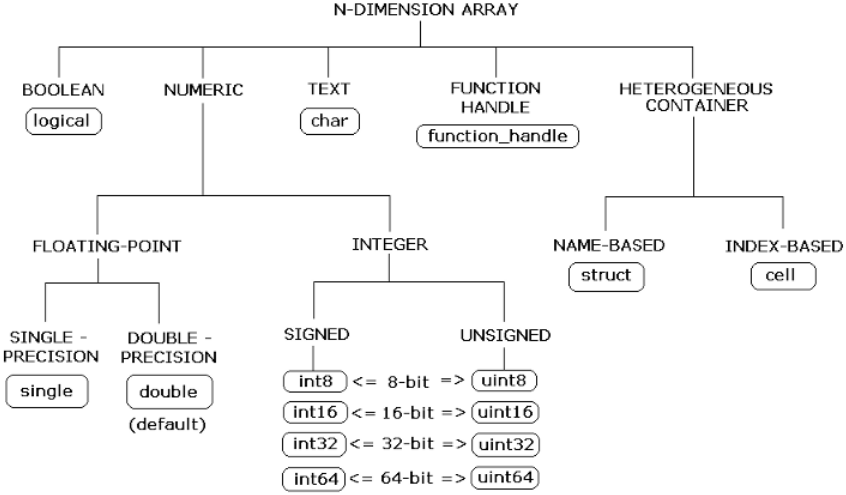
# More Guidelines on Style


More https://sites.google.com/site/matlabstyleguidelines

# Outline

1. Functions
2. Data Types
3. Random Number Generators
4. Exercises
   - Monte Carlo Simulation
   - Function Arguments
   - Improving Performance
   - Stochastic Matrices

# Data Types


N-DIMENSION ARRAY
BOOLEAN
logical
NUMERIC
TEXT
char
FUNCTION HANDLE
function_handle
HETEROGENEOUS CONTAINER
FLOATING-POINT
INTEGER
NAME-BASED
struct
INDEX-BASED
cell
SINGLE - PRECISION
single
DOUBLE - PRECISION
double
(default)
SIGNED
UNSIGNED
int8 <= 8-bit => uint8
int16 <= 16-bit => uint16
int32 <= 32-bit => uint32
int64 <= 64-bit => uint64

# Further Functions

- Numerical integration: `trapz` (trapezoidal integration), `polyint`
- Numerical Differentiation: `diff(y)./diff(x)`, `polyder`, `gradient`
- Ordinary Differential Equations: `ode45`, `ode15s`
- Fourier Transform: `fft`. `fftgui`

# Outline

# Random Generators

How can we generate random numbers in computers?
We cannot, we generate pseudo-random numbers.

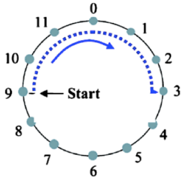
**Modular Arithmetic**

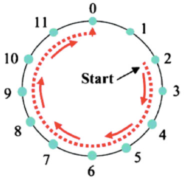
Modulo function: returns the reminder of division between two natural numbers: $x = a \cdot m + r$

Examples:

$$9 \bmod 8 = 1$$
$$16 \bmod 8 = 0$$
$$(9+6) \bmod 12 = 15 \bmod 12 = 3$$
$$(6 \cdot 2 + 15) \bmod 12 = 27 \bmod 12 = 3$$
$$1143 \bmod 1000 = 143$$

```
>> mod(9,8)
>> mod(16,8)
>> mod(9+6,12)
>> mod(6*2+15,12)
>> mod(27,12)
>> mod(1143,1000)
```



Given $a$, $c$, $m$ and $x_0$:

$$
\begin{aligned}
x_1 &:= (a \cdot x_0 + c) \bmod m,\\
x_2 &:= (a \cdot x_1 + c) \bmod m,\\
x_3 &:= (a \cdot x_2 + c) \bmod m,\\
x_4 &:= (a \cdot x_3 + c) \bmod m,\\
&\vdots\\
x_{i+1} &:= (a \cdot x_i + c) \bmod m
\end{aligned}
$$

Setting $a = 5, c = 1, m = 16$ and the seed $x_0 = 1$

$$
\begin{aligned}
x1 &:= (5 \cdot 1 + 1) \bmod 16 = 6,\\
x2 &:= (5 \cdot 6 + 1) \bmod 16 = 15,\\
x3 &:= (5 \cdot 15 + 1) \bmod 16 = 12,\\
x4 &:= (5 \cdot 12 + 1) \bmod 16 = 13,\\
x5 &:= (5 \cdot 13 + 1) \bmod 16 = 2,\\
x6 &:= (5 \cdot 2 + 1) \bmod 16 = 11,\\
&\vdots
\end{aligned}
$$

- Sequence of numbers that approximates the properties of random numbers
- Deterministic and periodic behaviour
- sequence not truly random, but completely determined by a relatively small set of initial values, called the PRNG's state, which includes a truly random seed
- By varying $a, c, m$ we can reach a full period of length $m$.

Characteristics of good generators:

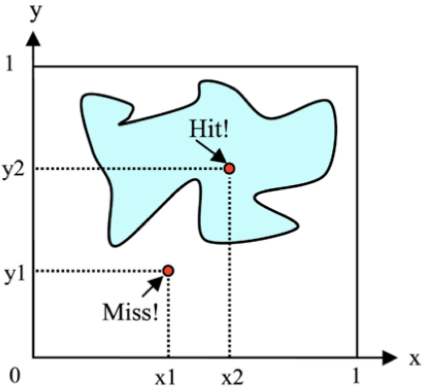
- long period
- uniform unbiased distribution
- uncorrelated (time series analysis)
- efficient

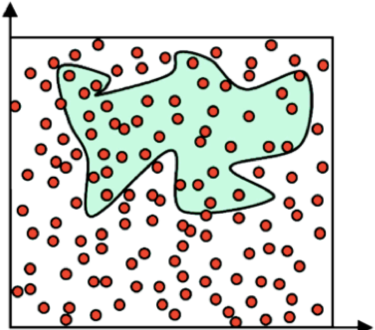
Mersenne Twister is the default algorithm

search “seed” in the Help. Changing random number generator syntax

# Monte Carlo Simulation

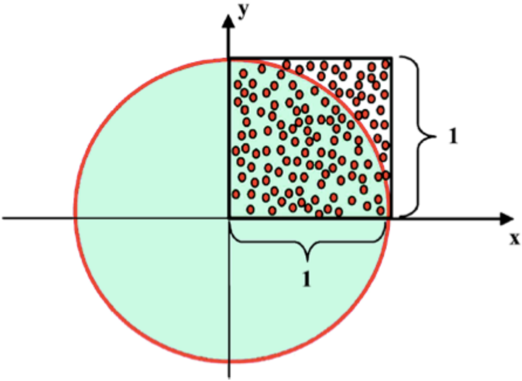
Calculate area by random rain:



# Outline

# Calculate $\pi$



# Solution

Let $A_s$ be the simulated area:

$$\frac{\pi}{4} = A_s$$

```
S=1000;
XY=rand(S,2);
P=sum(XY.^2,2);
hits=sum(P<1);
As=hits/S;
pi=4*As;
```

# Function Arguments

Create a new function in a file named addme.m that accepts one or two inputs and computes the sum of the number with itself or the sum of the two numbers. The function must be then able to return one or two outputs (a result and its absolute value).

# Solution

```
function [result,absResult] = addme2(a,b)

switch nargin
    case 2
        result = a + b;
    case 1
        result = a + a;
    otherwise
        result = 0;
end

if nargout > 1
   absResult = abs(result);
end
```

# Techniques for Improving Performance

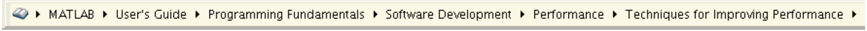
Can you improve performance and use memory more efficiently for this code?

```
A=rand(1000,400)>0.7
s=[]
M=0
for j=1:400
    tmp_s=0
    for i=1:1000
        if A(i,j)>M
            M=A(i,j)
        end
        if A(i,j)>0
            tmp_s=tmp_s+A(i,j)
    end
    s=[s, tmp_s]
end
```

Use `tic ... toc` and `whos` to analyse your code.
`tic; bad; toc`

For inspiration look at User's Guide:

MATLAB > User's Guide > Programming Fundamentals > Software Development > Performance > Techniques for Improving Performance



# Solution

```
A=rand(10000,400)>0.7;

M=max(A);
s=sum(A,1);
```

# Stochastic Matrices

| | current car ($j$) | | | | |
|---|---|---|---|---|---|
| **new car ($i$)** | Volkswagen | Fiat | Ford | Peugeot | Toyota |
| Volkswagen | 335 | 717 | 586 | 340 | 104 |
| Fiat | 375 | 257 | 409 | 551 | 626 |
| Ford | 491 | 43 | 614 | 292 | 445 |
| Peugeot | 246 | 383 | 373 | 567 | 649 |
| Toyota | 554 | 600 | 18 | 250 | 177 |

At time $0$:

| Volkswagen | Fiat | Ford | Peugeot | Toyota |
|---|---|---|---|---|
| 426 | 436 | 364 | 437 | 336 |

All current cars will buy a new car ⇝ we can normalize over the columns:

$$\mathbf{W} = \begin{bmatrix} 0.1673 & 0.3587 & 0.2930 & 0.1699 & 0.0520 \\ 0.1873 & 0.1283 & 0.2046 & 0.2756 & 0.3128 \\ 0.2457 & 0.0216 & 0.3068 & 0.1458 & 0.2224 \\ 0.1229 & 0.1915 & 0.1866 & 0.2837 & 0.3245 \\ 0.2769 & 0.2998 & 0.0091 & 0.1251 & 0.0883 \end{bmatrix} \qquad \mathbf{p}(0) = \begin{bmatrix} 0.2131 \\ 0.2180 \\ 0.1822 \\ 0.2185 \\ 0.1682 \end{bmatrix}$$

# Stochastic process

A stochastic process is any sequence of experiments for which the outcome at any stage depends on chance.

A Markov process is a stochastic process with the following properties:

- the set of possible states is finite
- the probability of the next outcome depends only on the previous outcome
- The probabilities are constant over time

**Theorem**

*If a Markov chain with an $n \times n$ transition matrix $\mathbf{A}$ converges to a steady-state vector $\mathbf{x}$, then:*

- $\mathbf{x}$ *is a probability vector*
- $\lambda_1 = 1$ *is an eigenvalue of $A$ and $\mathbf{x}$ is an eigenvector belonging to $\lambda_1$*

# Data Input and Export

Matlab has functions to handle several types of file formats.

- Importing/Exporting data space-separated to matrices: `load filename`

```
>> load cars.txt
>> cars
...
```

- Importing/Exporting from Spreadsheet Files

```
>> A=csvhread('cars.xls')
>> A=xlsread('cars.xls')
>> [A,B]=xlsread('cars.xls')
>> [A,B,D]=xlsread('cars.xls')
>> xlswrite(filename,A)
```

- Import wizard

```
uiimport
```

# Data Input and Export

- Low-level file I/O

```
Read selected rows or columns from a file:
% Create a file with values from 1 to 9
fid = fopen('nine.bin', 'w');
alldata = reshape([1:9],3,3);
fwrite(fid, alldata);
fclose(fid);
```

```
% Read the first six values into two
    columns
fid = fopen('nine.bin');
two_cols = fread(fid, [3, 2]);

% Return to the beginning of the file
frewind(fid);

% Read two values at a time, skip one
% Returns six values into two rows
% (first two rows of 'alldata')
two_rows = fread(fid, [2, 3], '2*uint8',
    1);

% Close the file
fclose(fid);
```

# Solution

Recalling the definition of multiplication between matrices we can write the probability distribution of cars at period 1 as:

$$Wp_0 = p_1$$

After two periods:

$$Wp_1 p_0 = W^2 p_0 = p_2$$

We search the steady state at which:

$$Wp_k = p_{k+1}$$

Remembering the definition of eigenvalues and eigenvectors:

$$Wp = \lambda_i p_i$$

# Solution

hence the steady state probability distribution of cars is given by the eigenvector corresponding to the eigenvector equal to 1. In Matlab:

```
A=xlsread('FF505/Slides/cars.xls');
s=sum(A,1);
W=A./repmat(s,[5,1]);
[V,l]=eig(W);
p_1=V(:,1)

p_1 =

    0.4740
    0.4849
    0.4053
    0.4858
    0.3742

p_k=p1./sum(p1)
W*p_k
p_0=[426 436 364 437 336];
p_k./sum(p_k)*sum(p_0)
```

# Summary

Topics common to main computational environments for Math, Physics and Engineering:

- Matrix calculations
- Graphics to visualize data
- Programming main structures (control flow, functions)

Take away messages:

- A computational environment like Matlab may be very helpful. Keep it alive, use it, experiment, try things.
- Develop skills to search things by your own. No way to remember everything. But helpful to know about existence. Look at User's Guide. Remember how things are called.