FF505/FY505

Computational Science

Lecture 4
Functions and Programming

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

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

- Graphics: 2D and 3D
- Random numbers generation
- Writing your own functions

Today

- More on Functions
- Programming scripts
- Questions, exercises
- Stochastic Matrices

Outline

1. Functions
2. Floating-Point Numbers
3. Programming
4. Stochastic Matrices

User-Defined Functions

function M-file (as opposed to script M-file) defined by syntax:

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

Example

In `fun.m`

```
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
```

⇝ variables have local scope

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.

```
global a x q
```

Programming style guides recommend avoiding to use them.

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).

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)
```

Finding zeros and minima

Example: Finding zeros and minima of a function

`X = FZERO(FUN,X0)` system function 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 first function of an M-file. Other are subroutines not callable.
- Subfunctions placed in the primary function
- Nested functions defined within another 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 restricted access.

Outline

1. Functions
2. Floating-Point Numbers
3. Programming
4. Stochastic Matrices

Floating-Point Numbers

A floating-point number in base b is a number of the form

$$\pm\left(\frac{d_1}{b}+\frac{d_2}{b^2}+\ldots+\frac{d_t}{b^t}\right)\times b^e$$

where $t, d_1, d_2, \ldots, d_t, b, e$ are all integers and

$$0 \leq d_i \leq b-1 \quad i=1,\ldots,t$$

- t refers to the number of digits and depends on the word length of the computer.
- e is restricted within bound $L \leq e \leq U$
- b is typically 2 or 10

Example: (5-digit, base 10)

0.53216×10^{-4} $\quad$ 0.00112×10^{8}

-0.81724×10^{21} $\quad$ 0.11200×10^{6}

Roundoff error

Most real numbers have to be rounded off to be represented in t-digit floating-point numbers.

Definition

If x is a real number and x' is its floating-point approximation, then the difference $x' - x$ is called the absolute error and the quotient $(x' - x)/x$ is called the relative error.

Real number x	4-digit decimal x'	Relative error
62.133	0.6213×10^5	$\frac{-3}{62.133} \approx -4.8 \times 10^{-5}$
0.12658	0.1266×10^0	$\frac{2\times 10^{-5}}{0.12658}$
47.213	0.4721×10^2	$\frac{-0.003}{47.213} \approx -6.4 \times 10^5$
π	0.3142×10^1	$\frac{3.142-\pi}{\pi} \approx 1.3 \times 10^{-4}$

With arithmetic operations additional roundoff errors occur:

Example: $a' = 0.263 \times 10^4$, $b' = 0.466 \times 10^1$:

$$a' + b' = 0.263446 \times 10^4$$

but in 3-digit floating point the sum is: 0.263×10^4.

Relative error:

$$\frac{fl(a'+b') - (a'+b')}{a'+b'} = \frac{-4.46}{0.263446 \times 10^4} \approx -0.17 \times 10^2$$

Machine Precision

Relative error:

$$\delta = \frac{(x' - x)}{x} \qquad \text{or} \qquad x' = x(1 + \delta)$$

$|\delta|$ can be bounded by a positive constant ϵ, called machine precision.

Machine precision is the smallest floating-point number ϵ for which

$$fl(1 + \epsilon) > 1$$

Example: (3-digit, decimal basis)

$$fl(1 + 0.499 \times 10^{-2}) = 1$$

while

$$fl(1 + 0.500 \times 10^{-2}) = 1.01$$

The machine ϵ would be 0.500×10^{2}

Outline

1. Functions
2. Floating-Point Numbers
3. Programming
4. Stochastic Matrices

Algorithms and Control Structures

Algorithm: an ordered sequence of instructions that perform some task in a finite amount of time.

Instructions can be numbered, but an algorithm has the ability to alter the order of its instructions using a control structure.

Three categories of algorithmic operations:

- Sequential operations
- Conditional operations: logical conditions that determine actions.
- Iterative operations (loops)

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.

Relational Operators

<	Less than.
<=	Less than or equal to.
>	Greater than.
>=	Greater than or equal to.
==	Equal to.
~=	Not equal to.

```
islogical(5~=8)
ans =
    1
islogical(logical(5+8))
ans =

    1
>> logical(5+8)
ans =
    1
>> double(6>8)
ans =
    0
>> isnumeric(double(6>8))
ans =
    1
```

Logical Operators

`~`	NOT	`~A` returns an array the same dimension as A; the new array has ones where A is zero and zeros where A is nonzero.
`&`	AND	`A & B` returns an array the same dimension as A and B; the new array has ones where both A and B have nonzero elements and zeros where either A or B is zero.
`|`	OR	`A | B` returns an array the same dimension as A and B; the new array has ones where at least one element in A or B is nonzero and zeros where A and B are both zero.
`&&`	Short-Circuit AND	Operator for scalar logical expressions. `A && B` returns true if both A and B evaluate to true, and false if they do not.
`||`	Short-Circuit OR	Operator for scalar logical expressions. `A || B` returns true if either A or B or both evaluate to true, and false if they do not.

Precedence

1. Parentheses; evaluated starting with the innermost pair.
2. Arithmetic operators and logical NOT (~); evaluated from left to right.
3. Relational operators; evaluated from left to right.
4. Logical AND.
5. Logical OR.

The if Statement

The if statement's basic form is

```
if logical expression
     statements
end
```

The else Statement

The basic structure for the use of the else statement is

```
if logical expression
     statement group 1
else
     statement group 2
end
```

```
if logical expression 1
   if logical expression 2
        statements
   end
end
```

can be replaced with the more concise program

```
if logical expression 1 & logical expression 2
      statements
end
```

The elseif Statement

The general form of the if statement is

```
if logical expression 1
    statement group 1
elseif logical expression 2
    statement group 2
else
    statement group 3
end
```

for Loops

A simple example of a for loop is

```
for k = 5:10:35
    x = k^2
end
```

while Loops

```
while logical expression
    statements
end
```

The while loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance. A simple example of a while loop is

```
x = 5;
while x < 25
    disp(x)
    x = 2*x - 1;
end
```

Switch

```
switch input expression (can be a scalar or string).
   case value1
       statement group 1
   case value2
       statement group 2
    .
    .
    .
   otherwise
       statement group n
end
```

Switch

```
switch angle
case 45
 disp('Northeast')
case 135
 disp('Southeast')
case 225
 disp('Southwest')
case 315
 disp('Northwest')
otherwise
 disp('Direction Unknown')
end
```

Outline

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 λ_1*