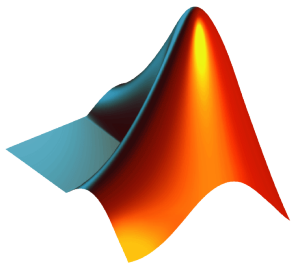
MatLab Programming Fundamentals

guarantor: Maroš Tunák
tel.: 3465
e-mail: *maros.tunak@tul.cz*



Course objectives

The aim of the course is to acquire basics knowledge and skills of students the MatLab program. At the end of the course students will be able to use MatLab for their own work and will be ready to deepen their programming skills in MatLab.

MatLab Programming Fundamentals

time requirements:	0p+2c
credits:	4
exercises:	Monday 10:40-12:15; 12:30-14:05 (B-PC2, Tunák M.)
	Tuesday 08:50-10:25; 10:40-12:15 (B-PC2, Tunák M.)
consultation:	Wednesday 10:40-12:15 (E-KHT)

Requirements on student/graded credit

1. participation in exercises (max. 3 absences)
2. elaboration of semester work (after approval of the semester work, you can attend a practical demonstration)
3. practical demonstration of acquired skills (there will be 1-2 examples to solve; elaboration time 1 hour; you can use any materials ...)

Content

IS/STAG Syllabus

1. Getting started with Matlab. Working environment, windows, paths, basic commands, variables. Loading, saving and information about variables. Help.
2. Mathematics with vectors and matrices. Creating vectors and matrices. Indexing. Special matrices. Matrix operations. Element by element operations. Relational operations, logical operations, examples and tricks.
3. Control flow. Loops, conditional statements, examples.
4. Script m-files, Function m-files.
5. Visualisation. Two-dimensional graphics. Three-dimensional graphics.
6. Graphical user interface.
7.-10. Statistics and Machine Learning Toolbox. Basics of statistical data processing, exploratory data analysis, descriptive statistics, data visualisation, hypothesis testing, confidence intervals, regression analysis, control charts.
11.-13. Solution of practical problems in textile and industrial engineering.

Literature

Recommended

MathWorks. *Getting Started with MATLAB*. [Online]. Dostupné z:
https://www.mathworks.com/help/matlab/getting-started-with-matlab.html

Study materials

http://elearning.tul.cz

Installation

http://liane.tul.cz/cz/software/MATLAB

Mathematics with vectors and matrices. Creating vectors and matrices. Indexing. Special matrices. Matrix operations. Element by element operations. Relational operations, logical operations, examples and tricks.

Vectors and matrices

Matrix Laboratory - the basic element of MatLab is a matrix (or and array), special cases

- 0×0 matrix: empty array
- 1×1 matrix: scalar or single number
- $1 \times n$ matrix: row vector
- $m \times 1$ matrix: column vector
- $m \times n$ matrix ($m = n$ square matrix)
- $m \times n \times o$ matrix (eg. $m \times n \times 3$ (RGB image matrix))

Command	Operations
[ ]	matrix or array (elements must be enclosed by square brackets)
, or space	column separators (comma or space)
;	row separator (semicolon)

Vectors and matrices - examples

- row vector a with numbers from 1 to 3 (separated by , or space):

```
>> a=[1 2,3]

a =
     1     2     3
```

- column vector b with numbers from 4 to 6 (separated by semicolon ;):

```
>> b=[4;5;6]

b =

     4
     5
     6
```

Vectors and matrices - examples

- matrix A with numbers from 1 to 6 organized in two rows and three columns:

```
>> A=[1 2 3; 4 5 6]

A =

     1     2     3
     4     5     6
```

- or with the using of existing variables a and b:

```
>> A=[a;b']

A =

     1     2     3
     4     5     6
```

Vectors and matrices - examples

- or with the using of existing variables a and b (pay attention to dimension of concatenated arrays):

```
>> A=[a;b]
Error using vertcat
Dimensions of arrays being concatenated are not consistent.

>> A=[a,b']

A =

     1     2     3     4     5     6
```

Special matrices

Some special built-in matrices of size specified by user

Command	Operation
» [ ]	empty matrix of size 0×0
» eye(r,c)	identity matrix of size $r \times c$
» zeros(r,c)	matrix of zeros of size $r \times c$
» ones(r,c)	matrix of ones of size $r \times c$
»rand(r,c)	matrix of elements from $U(0,1)$ of size $r \times c$
»randn(r,c)	matrix of elements from $N(0,1)$ of size $r \times c$

Special matrices - examples

```
>> s1=[]
s1 =
     []

>> size(s1)
ans =
     0     0

>> s2=eye(3)
s2 =
     1     0     0
     0     1     0
     0     0     1

>> s3=zeros(3,8)
s3 =
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Special matrices - examples

```
>> s4=ones(4)
s4 =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> s5=rand(1)
s5 =
    0.0975

>> s6=randn(4,3)
s6 =
    0.7254   -0.1241    0.6715
   -0.0631    1.4897   -1.2075
    0.7147    1.4090    0.7172
   -0.2050    1.4172    1.6302
```

Vectors and matrices - examples

Sometimes it is useful to create vectors and matrices that consist of a entries linearly spaced. Is it possible to created this using the `first:step:last` syntax, where `first` is starting value, `last` is the last value and `step` is a division step.

- an example of a vector of values from 1 to 7 with the default of step 1:

```
>> c=1:7

c =

     1     2     3     4     5     6     7
```

Vectors and matrices - examples

- a column vector of values from 0 to 0.9 with a step of 0.2:

```
>> d=(0:0.2:0.9)'

d =

         0
    0.2000
    0.4000
    0.6000
    0.8000
```

- the dividing step can also be negative:

```
>> e=[6:-1:1]

e =

     6     5     4     3     2     1
```

Vectors and matrices - examples

Lineary spaced vectors can also be created using `linspace(first,last,n)` function, where `n` represents the number of elements.

- example of a column vector of values from 1 to 5 with seven elements:

```
>> f=linspace(1,5,7)'

f =

    1.0000
    1.6667
    2.3333
    3.0000
    3.6667
    4.3333
    5.0000
```

Vectors and matrices - examples

- the matrix A can also be created:

```
>> A=[1:3;linspace(4,6,3)]

A =

     1     2     3
     4     5     6
```

Vectors and matrices - examples

The matrix size information can be obtained using the `size` command,command returns the number of rows and columns. The `length` command returns the greater value of the `size` response.

```
>> size(A)

ans =

     2     3

>> length(A)

ans =

     3
```

Vectors and Matrices

Command	Operations
» first:step:last	vector of entities with the value first, incrementing by the step, until it reaches last
» first:last	step $= 1$
» first:-step:last	negative step is also possible
» linspace(first,last,n)	generates linearly spaced vectors similar to the colon operator, but gives direct control over the number of points *n*
» length(variable)	length of vector size of longest dimension of matrix
» size(variable)	sizes of each dimension of matrix or array

Matrix indexing

On individual elements in a vector or matrix of size $m \times n$ we can refer by their indexes in closed parentheses, taking each index is separated by a comma ($M(i,j)$), where M is a matrix, i is a row index, and j column index.

- example of element in second row and third column of A matrix:

```
>> A(2,3)

ans =

     6
```

- the index can also be a vector where end in this case denotes to the end

```
>> A(2,1:2:end)

ans =

     4     6
```

Matrix indexing

- colon operator : represents all elements in a row or column

```
>> A(1,:)

ans =

     1     2     3
```

- using indexing, vectors or matrices can be expanded; adding new values, further observation or measurement

```
>> A(3,:)=7:9

A =

     1     2     3
     4     5     6
     7     8     9
```

Matrix indexing

- empty matrix [] is used to remove vector or matrix elements

```
>> A(3,:)=[]

A =

     1     2     3
     4     5     6
```

Matrix indexing

Indexing into a matrix is a means of selecting a subset of elements from the matrix

Command	Operations
A(i)	*i*th element of matrix
A(i,j)	element in *i*th row and *j*th column
A(1:2,2:end)	elements in first two rows and in columns from second to the last
:	for all elements in specified dimension
end	to the last element

Manipulation with matrices

Command	Operation
» rot90	90 degree rotation counterclockwise
» fliplr	flip matrix left to right
» flipud	flip matrix up to down
» tril	lower triangular part of matrix
» triu	upper triangular part of matrix
» diag(vector)	diagonal matrix with *vector* on diagonal
» diag(matrix)	extract diagonal element of *matrix*
» reshape	reshape array

Matrix indexing

```
>> A
A =
     1     2     3
     4     5     6

>> rot90(A)
ans =
     3     6
     2     5
     1     4

>> fliplr(A)
ans =
     3     2     1
     6     5     4

>> flipud(A)
ans =
     4     5     6
     1     2     3
```

Matrix indexing

```
>> t=1:16

t =
     1     2     3     4     5     6     7     8     9    10    ...
    11    12    13    14    15    16

>> T=reshape(t,4,4)

T =
     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16

>> tril(T)
ans =
     1     0     0     0
     2     6     0     0
     3     7    11     0
     4     8    12    16
```

Matrix indexing

```
>> triu(T)

ans =
     1     5     9    13
     0     6    10    14
     0     0    11    15
     0     0     0    16

>> diag(T)
ans =
     1
     6
    11
    16
```

Matrix indexing

```
>> diag(1:6)

ans =

     1     0     0     0     0     0
     0     2     0     0     0     0
     0     0     3     0     0     0
     0     0     0     4     0     0
     0     0     0     0     5     0
     0     0     0     0     0     6
```

Matrix indexing

An array that has more than two dimensions is called a multidimensional array, an extension of a two-dimensional matrix, and uses an additional index for the next dimension. For example, a three-dimensional array uses three indexes: row, column, and third dimension (eg, an RGB image matrix).

- a multidimensional matrix can be created by extending a two-dimensional matrix

```
>> B=fliplr(flipud(A))

B =

     6     5     4
     3     2     1
```

Matrix indexing

```
>> C=A;
>> C(:,:,2)=B

C(:,:,1) =

     1     2     3
     4     5     6


C(:,:,2) =

     6     5     4
     3     2     1
```

Matrix indexing

```
>> C(:,:,3)=pi*ones(size(C(:,:,1)))

C(:,:,1) =

     1     2     3
     4     5     6


C(:,:,2) =

     6     5     4
     3     2     1


C(:,:,3) =

    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
```

Matrix indexing

- individual elements can be referred to by individual indexes, elements of the matrix C on the second row in the third column and on all layers

```
>> C(2,3,:)

ans(:,:,1) =

     6


ans(:,:,2) =

     1


ans(:,:,3) =

    3.1416
```

Concatenation

- concatenation of multiple matrices into a larger matrix. [] is a concatenation operator

```
>> E=[A, B]

E =

     1     2     3     6     5     4
     4     5     6     3     2     1

>> F=[A; B]

F =

     1     2     3
     4     5     6
     6     5     4
     3     2     1
```

Concatenation

- or concatenation arrays along specified dimension, command `cat`

```
>> cat(1,A,B)

ans =

     1     2     3
     4     5     6
     6     5     4
     3     2     1

>> cat(2,A,B)

ans =

     1     2     3     6     5     4
     4     5     6     3     2     1
```

Concatenation

```
>> cat(3,A,B)

ans(:,:,1) =

     1     2     3
     4     5     6


ans(:,:,2) =

     6     5     4
     3     2     1
```

Examples for practice

Examples for practice

1. Create a 1×5 vector b with all elements equal to 0 and 3×1 vector c with all elements equal to 1
2. Create a 1×5 vector d with elements equal to 1, 2, 3, 4, π respectively
3. Define the three vectors: e = [2, 4, 6, ..., 20]; f = [-21, 20, , ..., -12]; g = randn(1,10), and create a matrix A whose rows are e, f and g, in that order.
   - read out the first five elements of rows one and two
   - replace the element in the second row, third column, with $-\infty$
4. Find a **short** MATLAB expression to build the matrix

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 7 & 5 & 3 & 1 & -1 & -3 \\ 4 & 8 & 16 & 32 & 64 & 128 & 256 \end{pmatrix}$$

- from B select submatrix $C1$, which contains 3^{rd} and 5^{th} column
- from B select submatrix $C2$, which contains 3×4^{th} column
- from B select submatrix $C3$ from 2^{nd} to 3^{rd} row and 4^{th} to last column
- from B create a column vector
- swap 2^{nd} and 5^{th} columns in the matrix in the B

Solution