

A0B17MTB – Matlab

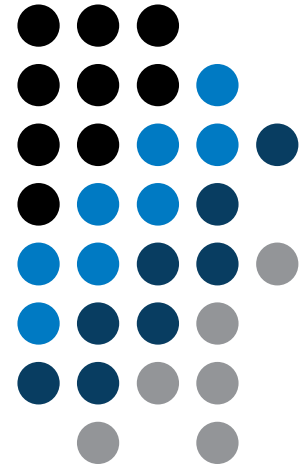
Part #2



Miloslav Čapek
miloslav.capek@fel.cvut.cz

Viktor Adler, Pavel Valtr

Department of Electromagnetic Field
B2-634, Prague



Learning how to ...

Complex numbers

Matrix creation

Operations with matrices

Vectorization

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

A

5	11	8
9	7	12
4	14	1

A([2 3 4], [1 2 4])

Complex numbers

- more entry options in Matlab
 - we want to avoid confusion
 - speed

```
>> C5 = sqrt(-1)
```

```
>> C1 = 1 + 1j
>> C2 = 1 + 5i % preferred
>> C3 = 1 + i
>> C4 = 1 + j5
```

- frequently used functions

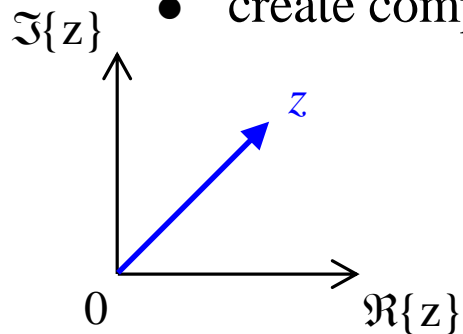
real, imag	real and imaginary part of a complex number
conj	complex conjugate
abs	absolute value of a complex number
angle	angle in complex plane (in [rad])
complex	constructs complex number from real and imaginary components
isreal	checks if input is a complex number (more on that later)
i, j	complex unit
cplxpair	sorts complex numbers into complex conjugate pairs

Complex numbers

300 s



- create complex number z and its complex conjugate



$$z = 1 + 1j$$

$$s = z^*$$

- switch between Cartesian and polar form (find $|z|, \varphi$)

$$z = \Re\{z\} + j\Im\{z\} = a + jb$$

$$z = |z|e^{j\varphi}, \quad |z| = \sqrt{a^2 + b^2}$$

$$z = |z|(\cos(\varphi) + j\sin(\varphi))$$

- verify Moivre's theorem

$$z^n = (|z|e^{j\varphi})^n$$

$$z^n = |z|^n (\cos(n\varphi) + j\sin(n\varphi))$$

$$Z = |z| = \sqrt{2} \approx 1.4142$$

$$\varphi = \arctan\left(\frac{\Im\{z\}}{\Re\{z\}}\right) = \arctan\left(\frac{1}{1}\right) \approx 0.7854 \text{ rad}$$

Complex numbers

300 s



- find out magnitude of a complex vector (avoid indexing)

- use `abs`, `sqrt`

$$\mathbf{Z} = (1+1j \quad \sqrt{2})$$

$$\|\mathbf{Z}\| = ?, \quad \mathbf{Z} \in \mathbb{C}^2$$

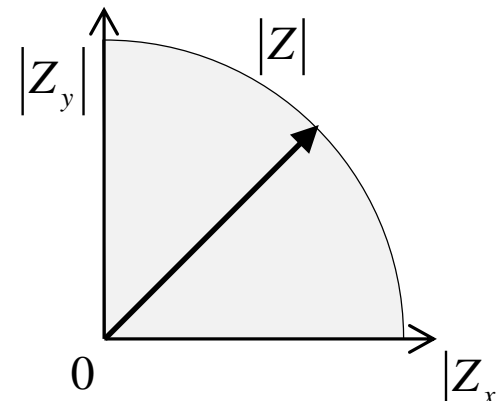
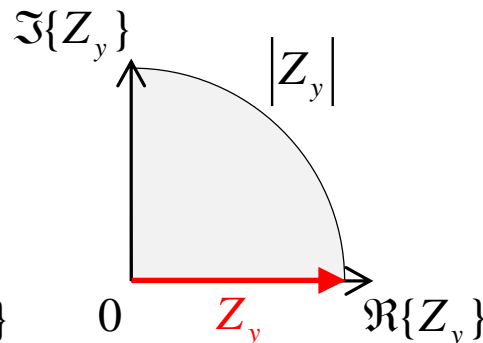
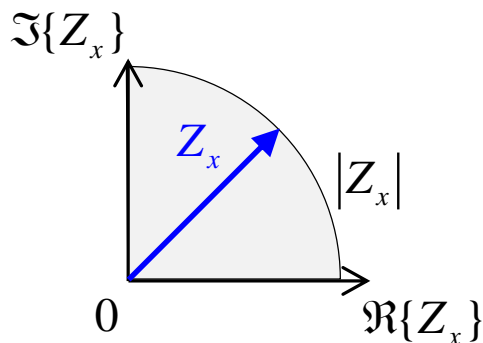
$$(1) \quad |Z_x|, |Z_y|$$

$$(2) \quad |\mathbf{Z}| = \sqrt{|Z_x|^2 + |Z_y|^2} = \sqrt{\mathbf{Z}_x \mathbf{Z}_x^* + \mathbf{Z}_y \mathbf{Z}_y^*}$$

$$= \sqrt{\mathbf{Z} \cdot \mathbf{Z}^*} = \sqrt{|\mathbf{Z}|^2}$$

- alternatively, use following functions:

- `norm`
- `dot` (*dot product*)
- `hypot` (*hypotenuse*)



Transpose and matrix conjugate

- Pay attention to situations where the matrix is complex, $\mathbf{A} \in \mathbb{C}^{M \times N}$
- two distinct operations:

transpose	$\mathbf{A}^T = [A_{ij}]^T = [A_{ji}]$	<code>transpose(A) % <- don't use</code>	<code>A.'</code>
transpose + conjugate	$\mathbf{A}^H = \mathbf{A}_{ij}^H = [\mathbf{A}^*]^T$	<code>ctranspose(A) % <- don't use</code>	<code>A'</code>

```
>> A = magic(2) + 1j*magic(2)'
```

A =

```
1.0000 + 1.0000i    3.0000 + 4.0000i
4.0000 + 3.0000i    2.0000 + 2.0000i
```

```
>> A.'
```

ans =

```
1.0000 + 1.0000i    4.0000 + 3.0000i
3.0000 + 4.0000i    2.0000 + 2.0000i
```

```
>> A'
```

ans =

```
1.0000 - 1.0000i    4.0000 - 3.0000i
3.0000 - 4.0000i    2.0000 - 2.0000i
```

Entering matrices – „:“

- large vectors and matrices with regularly increasing elements can be typed in using colon operator
 - a is the smallest element („from“), $incr$ is increment, b is the largest element („to“)

```
>> A = a:incr:b
```

```
>> A = 1:4:17
```

```
A =
```

```
1    5    9   13   17
```

- b doesn't have to be element of the series in question
 - last element $N \cdot incr$ then follows the inequality:

$$|a + N \cdot incr| \leq |b|$$

```
>> A = 1:4:7
```

```
A =
```

```
1    5
```

- if $incr$ is omitted, the increment is set equal to 1

```
>> A = a:b
```

```
>> A = 0:10
```

```
A =
```

```
0    1    2    3    4    5    6    7    8    9   10
```

Entering matrices

300 s ↑

- Using the colon operator „:“ create
 - following vectors

$$\mathbf{u} = (1 \quad 3 \quad \dots \quad 99)$$

$$\mathbf{v} = (25 \quad 20 \quad \dots \quad -5)^T$$

- matrix
 - caution, the third column cant be created using colon operator ":" only

$$\mathbf{T} = \begin{pmatrix} -4 & 1 & \frac{\pi}{2} \\ -5 & 2 & \frac{\pi}{4} \\ -6 & 3 & \frac{\pi}{6} \end{pmatrix}$$

Entering matrices – linspace, logspace

- colon operator defines vector with evenly spaced points
- In the case fixed number of elements of a vector is required, use linspace:

```
>> A = linspace(a, b, N);
```

```
>> A = linspace(0,2,5)
```

```
A =
```

```
0    0.5000    1.0000    1.5000    2.0000
```

- When the N parameter is left out, 100 elements of the vector are generated:

```
>> A = linspace(a, b);
```

- the function logspace works analogically, except that logarithmic scale is used

```
>> A = logspace(a, b, N);
```

Entering matrices

200 s ↑

- create a vector of 100 evenly spaced points in the interval $\langle -1.15, 75.4 \rangle$
- create a vector of 201 evenly spaced points in the interval $\langle 100, -100 \rangle$
- create a vector with spacing of -10 in the interval $\langle 100, -100 \rangle$
 - try both options using `linspace` and colon `:`

Entering matrices using functions

- special types of matrices of given size are needed quite often
 - Matlab offers number of functions to serve this purpose
- example: matrix filled with zeros
 - will be used quite often

```
zeros(m)                % matrix B of size m×m
zeros(m, n)             % matrix B of size m×n
zeros(m, n, p, ...)    % matrix B of size m×n×p×...
zeros([m n])           % matrix B of size m×n

B = zeros(m, 'single') % matrix B of size m×m, of type 'single'

% see Help for other options
```

Entering matrices using functions

- following useful functions analogical to the `zeros` function are available

<code>ones</code>	matrix filled with ones
<code>eye</code>	identity matrix
<code>NaN</code> , <code>Inf</code>	matrix filled with NaN, matrix filled with Inf
<code>magic</code>	matrix suitable for Matlab experiments, notice its interesting properties
<code>rand</code> , <code>randn</code> , <code>randi</code>	matrix filled with random numbers coming from uniform and normal distribution, matrix filled with uniformly distributed random integers
<code>randperm</code>	returns a vector containing a random permutation of numbers
<code>diag</code>	creates diagonal matrix or returns diagonal of a matrix
<code>blkdiag</code>	constructs block diagonal matrix from input arguments
<code>cat</code>	groups several matrices into one (depending on dimension)
<code>true</code> , <code>false</code>	creates a matrix of logical ones and zeros
<code>pascal</code> , <code>hankel</code>	Pascal matrix, Hankel matrix

- for further functions see Matlab → Mathematics → Elementary Math → Constants and Test Matrices

Entering matrices using functions

360 s ↑

- create following matrices
 - use Matlab functions
 - begin with matrices you find easy to cope with

$$\mathbf{M}_1 = \begin{pmatrix} \text{NaN} & \text{NaN} \\ \text{NaN} & \text{NaN} \end{pmatrix}$$

$$\mathbf{M}_2 = (1 \ 1 \ 1 \ 1)$$

$$\mathbf{M}_3 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -5 \end{pmatrix}$$

$$\mathbf{M}_4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Entering matrices using functions

360 s ↑

- try to create empty 3-dimensional array of type `double`

- can you find another option?

Entering matrices

- quite often there are several options how to create given matrix
 - it is possible to use output of one function as an input of another function in Matlab:
- consider
 - clarity
 - simplicity
 - speed
 - convention
- e.g. band matrix with '1' on main diagonal and with '2' and '3' on secondary diagonals

```
>> plot(diag(randn(10, 1), 1))
```

```
>> N = 10;  
>> diag(ones(N,1)) + diag(2*ones(N-1,1),1) + diag(3*ones(N-1,1),-1)
```

- can be sorted out using `for` cycle as well (see next slides), might be faster ...
- some other idea?

Dealing with sparse matrices

- Matlab provides support for working with sparse matrices
 - most of the elements of sparse matrices are zeros and it pays off to store them in a more efficient manner

- to create sparse matrix S out of a matrix A :

```
S = sparse(A) ,
```

- conversion of a sparse matrix to a full matrix :

```
B = full(S) ,
```

- in the case of need see Help for other functions

Matrix operations #1

- there are other useful functions apart from transpose (`transpose`) and matrix diagonal (`diag`):

P =

```
0.3404  0.2551  0.9593  0.2575
0.5853  0.5060  0.5472  0.8407
0.2238  0.6991  0.1386  0.2543
0.7513  0.8909  0.1493  0.8143
```

- upper triangular matrix

```
>> U = triu(P),
```

```
>> U = triu(P)
```

U =

```
0.3404  0.2551  0.9593  0.2575
0        0.5060  0.5472  0.8407
0        0        0.1386  0.2543
0        0        0        0.8143
```

- lower triangular matrix

```
>> L = tril(P),
```

```
>> L = tril(P)
```

L =

```
0.3404  0        0        0
0.5853  0.5060  0        0
0.2238  0.6991  0.1386  0
0.7513  0.8909  0.1493  0.8143
```

- a matrix can be modified taking into account secondary diagonals as well

```
>> L = triu(P, -1),
```

```
>> U2 = triu(P, -1)
```

U2 =

```
0.3404  0.2551  0.9593  0.2575
0.5853  0.5060  0.5472  0.8407
0        0.6991  0.1386  0.2543
0        0        0.1493  0.8143
```

Matrix operations #2

- function `repmat` is used to copy (part of) a matrix

```
>> B = repmat(A, m, n),
```

- e.g.

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \end{pmatrix}$$

```
>> B = repmat(A, 1, 2),
>> B = repmat(A, [2 1]),
```

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{11} & A_{12} & A_{13} \end{pmatrix}$$

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{11} & A_{12} & A_{13} \end{pmatrix}$$

- `repmat` is a very fast function
 - comparison of execution time of creating a $1e4 \times 1e4$ matrix filled with zeros (HW, SW and Matlab version dependent):

```
>> X = zeros(1e4, 1e4);      % computed in 0.18s
>> Y = repmat(0, 1e4, 1e4); % computed in 0.0004s, BUT... don't use it
```

- it is for you to consider the way of matrix allocation ...

Matrix operations #3

- function reshape is used to reshuffle a matrix

```
>> B = reshape(A, m, n),
```

- e.g.

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

```
>> B = reshape(A, [4 1]),
>> B = reshape(A, 1, 4),
>> B = reshape(A, [], 4),
```

$$\begin{matrix} A_{11} \\ A_{21} \\ A_{12} \\ A_{22} \end{matrix}$$

$$A_{11} \quad A_{21} \quad A_{12} \quad A_{22}$$

Matrix operations #4

- following functions are used to swap the order of

- columns: `fliplr`

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{pmatrix}$$

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

$$\mathbf{A} = \begin{pmatrix} A_{13} & A_{12} & A_{11} \\ A_{23} & A_{22} & A_{21} \end{pmatrix}$$

- rows: `flipud`

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

$$\mathbf{A} = \begin{pmatrix} A_{21} & A_{22} & A_{23} \\ A_{11} & A_{12} & A_{13} \end{pmatrix}$$

- row-wise or column-wise: `flip`

```
>> B = flip(A, 1),
>> B = flip(A, 2),
```

- the same result is obtained using indexing (see next slides)

Matrix operations #5

- circular shift is also available
 - can be carried out in chosen dimension (row-wise/ column-wise)
 - can be carried out in both directions (back / forth)

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

```
>> B = circshift(A, -2),
```

$$\mathbf{A} = \begin{pmatrix} A_{31} & A_{32} & A_{33} \\ A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{pmatrix}$$

```
>> B = circshift(A, [-2 1]),
```

$$\mathbf{A} = \begin{pmatrix} A_{33} & A_{31} & A_{32} \\ A_{13} & A_{11} & A_{12} \\ A_{23} & A_{21} & A_{22} \end{pmatrix}$$

- Consider the difference between flip a circshift

Matrix operations #1

450 s ↑

- convert the matrix $\mathbf{A} = \begin{pmatrix} 1 & \pi \\ e & -i \end{pmatrix}$ to have the form of matrices \mathbf{A}_1 to \mathbf{A}_4

- use `repmat`, `reshape`, `triu`, `tril` and `conj`

$$\mathbf{A}_1 = \begin{pmatrix} 1 & \pi & 1 & \pi & 1 & \pi \\ e & -i & e & -i & e & -i \end{pmatrix}$$

$$\mathbf{A}_2 = (1 \ \pi \ e \ -i)$$

$$\mathbf{A}_3 = \begin{pmatrix} 1 & \pi \\ e & +i \\ 1 & \pi \\ e & +i \\ 1 & \pi \\ e & +i \end{pmatrix}$$

$$\mathbf{A}_4 = \begin{pmatrix} 1 & \pi & 0 & 0 & 0 & 0 \\ e & -i & e & 0 & 0 & 0 \\ 0 & \pi & 1 & \pi & 0 & 0 \\ 0 & 0 & e & -i & e & 0 \\ 0 & 0 & 0 & \pi & 1 & \pi \\ 0 & 0 & 0 & 0 & e & -i \end{pmatrix}$$

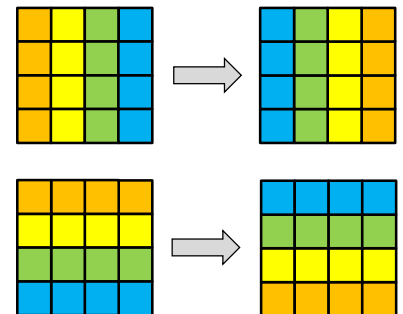
Matrix operations #2

450 s ↑

- create following matrix (use advanced techniques)

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 0 & 2 & 4 & 0 & 2 & 4 \\ 0 & 0 & 5 & 0 & 0 & 5 \end{pmatrix}$$

- save the matrix in file named `'matrix.mat'`
- create matrix **B** by swapping columns in matrix **A**
- create matrix **C** by swapping rows in matrix **B**
- add matrices **B** and **C** in the file `'matrix.mat'`



Matrix operations #3

150 s ↑

- compare and interpret following commands:

```
>> x = (1:5) .';           % entering vector
>> X = repmat(x, [1 10]), % 1. option
>> X = x(:, ones(10, 1)), % 2. option
```

```
>> x = (1:5)';
x =
     1
     2
     3
     4
     5
```

- repmat is powerful, but not always the most time-efficient function

```
>> X = repmat(x, [1 10])
X =

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

>> X = x(:, ones(10, 1))
X =

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


Vector and matrix operations

- remember that matrix multiplication is not commutative, i.e. $\mathbf{AB} \neq \mathbf{BA}$
- remember that vector \times vector product results in

$$\mathbf{v}_{M,1} \mathbf{u}_{1,N} = \mathbf{w}_{M,N}$$

	u_{11}	u_{12}
v_{11}	w_{11}	w_{12}
v_{21}	w_{21}	w_{22}
v_{31}	w_{31}	w_{32}

$$\mathbf{v}_{1,M} \mathbf{u}_{M,1} = w_{1,1}$$

			u_{11}
			u_{21}
			u_{31}
v_{11}	v_{12}	v_{13}	w_{11}

- ... pay attention to the dimensions of matrices!

Element-by-element vector product

- it is possible to multiply arrays of the same size in the element-by-element manner in Matlab
 - result of the operation is an array
 - size of all arrays are the same, e.g. in the case of 1×3 vectors:

$$\mathbf{a} = (a_1 \quad a_2 \quad a_3) \quad \mathbf{b} = (b_1 \quad b_2 \quad b_3)$$

```
>> a*b
```

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}, \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \rightarrow$$

Error using *
(Inner matrix dimensions must agree.)

```
>> a.*b
```

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}, \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 b_1 & a_2 b_2 & a_3 b_3 \end{bmatrix} = [a_i b_i]$$

Element-by-element matrix product

- if element-by-element multiplication of two matrices of the same size is needed, use the ‘`.*`’ operator
 - i.e. two cases of multiplication are distinguished

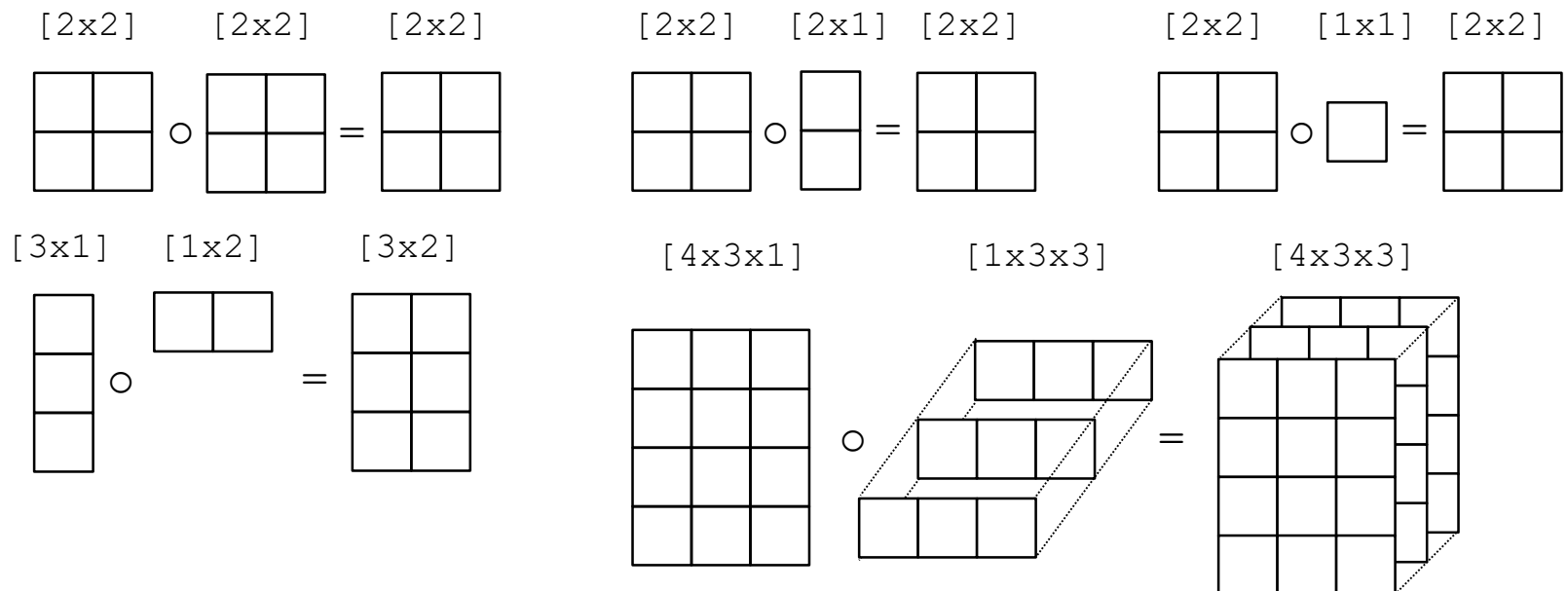
$$\gg A * B \quad \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

$$\gg A .* B \quad \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} \\ A_{21}B_{21} & A_{22}B_{22} \end{bmatrix}$$

- It is so called *Hadamard product* / *element-wise product* / *Schur product*: $\mathbf{A} \circ \mathbf{B}$

Compatible Array Sizes

- From R2016b most two-input (binary) operators support arrays that have *compatible sizes*
 - variables has compatible sizes if its sizes are either the same or one of them is 1 (for all dimensions)
- Examples:
 - \circ is arbitrary two-input element-wise operator (+, -, .*, ./, &, <, ==, ...)

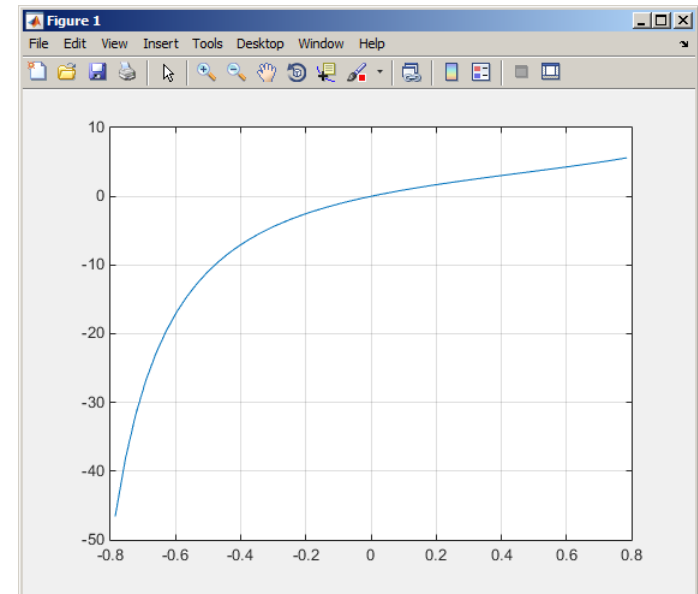


Element-wise operations #1

- element-wise operations can be applied to vectors as well in Matlab. Element-wise operations can be usefully combined with vector functions
- it is possible, quite often, to eliminate 1 or even 2 for-loops!!!
- these operations are exceptionally efficient
→ allow the use of so called vectorization (see later)

- e.g.: $f(x) = \frac{10}{(x+1)} \tan(x),$
 $x \in \left[-\frac{\pi}{4}, \frac{\pi}{4} \right]$

```
>> x = -pi/4:pi/100:pi/4;
>> fx = 10./(1+x).*tan(x);
>> plot(x, fx);
>> grid on;
```



Element-wise operations #1

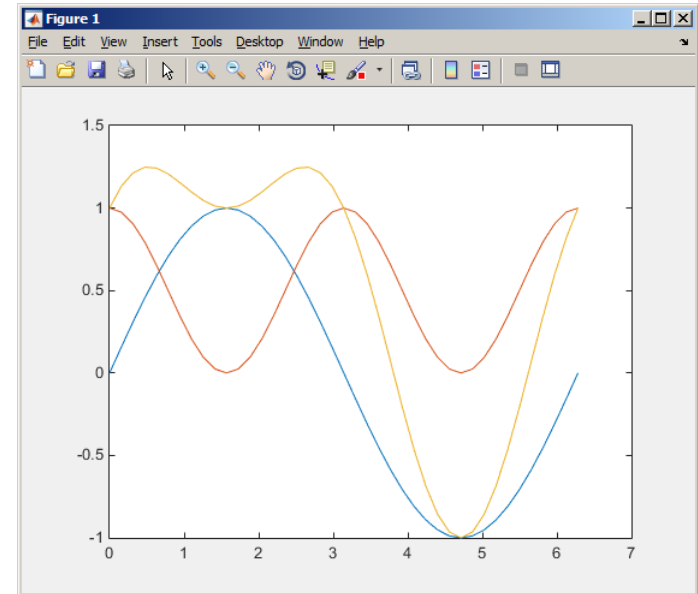
300 s ↑

- evaluate functions $f_1(x) = \sin(x)$ of the variable $x \in [0, 2\pi]$
 $f_2(x) = \cos^2(x)$
 $f_3(x) = f_1(x) + f_2(x)$
- evaluate the functions in evenly spaced points of the interval, the spacing is $\Delta x = \pi/20$

- for verification:

```
>> plot(x, f1, x, f2, x, f3),
```

- Matlab also enables symbolic solution (see later)



Element-wise operations #2

240 s



- depict graphically following functional dependence in the interval

$$x \in [0, 5\pi]$$

- plot the result using following function

$$f_4(x) = \frac{-\cos(3x)}{\cos(x)\sin\left(x - \frac{\pi}{5}\right) - \pi}$$

```
>> plot(x, f4);
```

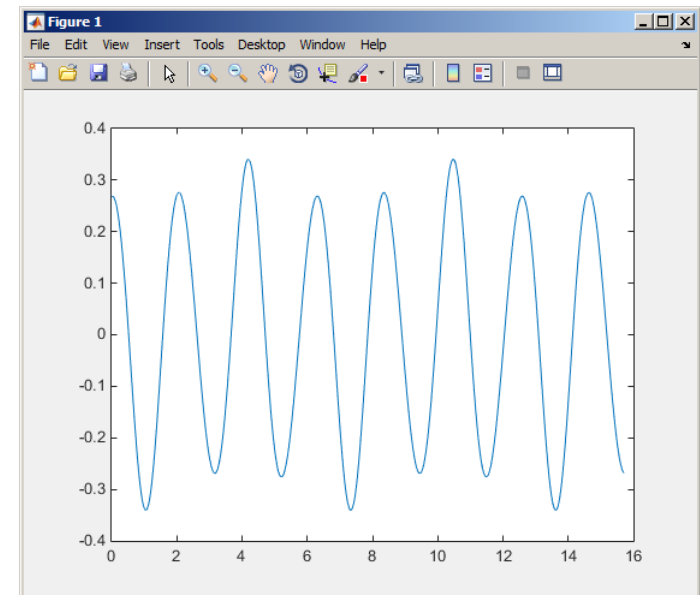
- explain the difference in the way of

```
>> A*B,
```

```
>> A.*B,
```

```
>> A'.*B,
```

multiplication of matrices of the same size

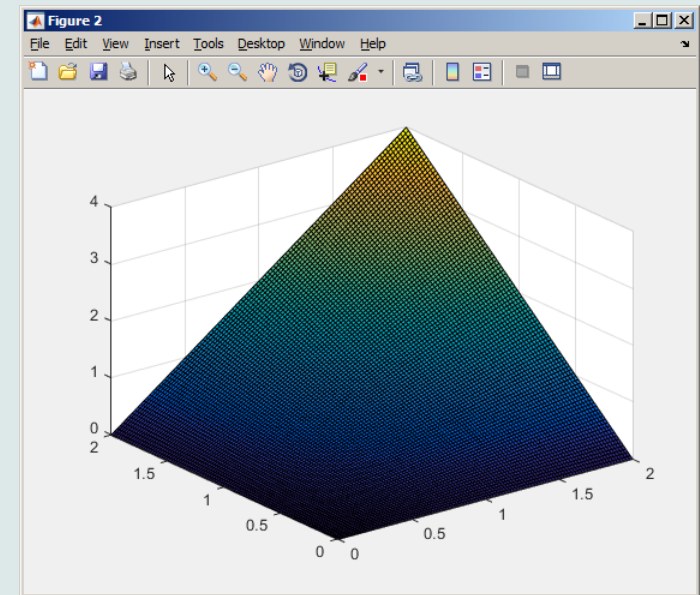


Element-wise operations #3

360 s



- evaluate the function $f(x, y) = xy$, $x, y \in [0, 2]$, use 101 evenly spaced points in both x and y
- the evaluation can be carried out either using vectors, matrix element-wise vectorization or using two for loops
 - plot the result using `surf(x, y, f)`
 - when ready, try also $f(x, y) = x^{0.5}y^2$ on the same interval



Size of matrices and other structures

- it is often needed to know size of matrices and arrays in Matlab
- function `size` returns a vector giving the size of the matrix / array

```
>> A = randn(3, 5);
>> d = size(A) % d = [3 5]
```

- function `length` returns largest dimension of an array
 - i.e. `length(A) = max(size(A))`

```
>> A = randn(3, 5, 8);
>> e = length(A) % e = 8
```

- function `ndims` returns number of dimensions of a matrix / array
 - i.e. `ndims(A) = length(size(A))`

```
>> m = ndims(A) % m = 3
```

- function `numel` returns number of elements of a matrix / array
 - i.e. `numel(A) = prod(size(A))`

```
>> n = numel(A) % n = 120
```

Size of matrices and other structures

250 s ↑

- create an arbitrary 3D array
 - you can make use of the following commands :

```
>> A = rand(2+randi(10), 3+randi(5));  
>> A = cat(3, A, flipud(fliplr(A)))
```

- and now:
 - find out the size of A
 - find out the number of elements of A
 - find out the number of elements of A in the 'longest' dimension
 - find out the number of dimensions of A

Data types in Matlab

- can be postponed for later ...

```
>> whos
```

Name	Size	Bytes	Class	Attributes
D	50x1	400	double	
DD	1x20	160	double	
DWx	20x20	3200	double	
DWy	20x20	3200	double	
Eps	1x1	8	double	
KA	20x20	3200	double	
L	1x1	8	double	
Lcheck	20x20	3200	double	
N	1x1	8	double	
Nth	1x1	8	double	
OK	1x1	1	logical	
PR	20x20	3200	double	
Pr	1x1	8	double	
SOL	20x20	400	logical	
Tcross	1x1	4	single	
lam	1x1	8	double	
omWA	20x20	3200	double	
psi	1x1	8	double	
type_of_connection	1x6	12	char	

```
>> class(type_of_connection)

ans =

char
```

Bonus: function gallery

- function enabling to create a vast set of matrices that can be used for Matlab code testing
- most of the matrices are special-purpose
 - function gallery offers significant coding time reduction for advanced Matlab users
- see `help gallery` / `doc gallery`
- **try** for instance:

```
>> gallery('pei', 5, 4)
>> gallery('leslie', 10)
>> gallery('clement', 8)
```

Function why

- it is a must to try that one! :)
- try help `why`
- try to find out how many answers exist

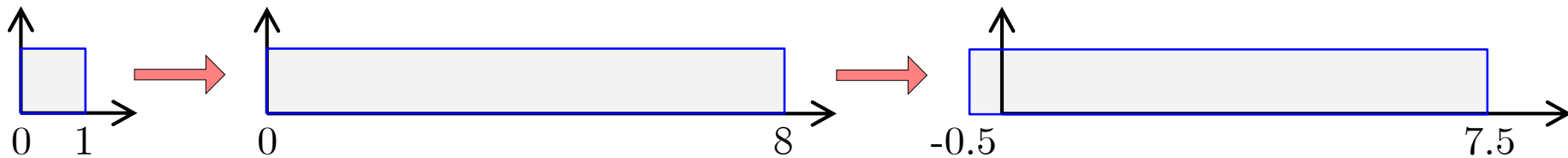
Discussed functions

real, imag, cong, angle, complex	complex numbers related functions	
norm, cumsum	norm (of a matrix / vector), cummulative sum	•
hypot	square root of sum of squares (real / complex numbers)	
linspace, logspace	vector generation - evenly spaced, linear / logarithmic scale	
zeros, ones, eye, NaN, magic	create matrix	
rand, randn, randi	matrix of random numbers with uniform or normal distribution, matrix of random integers	
randperm	vector containing a random permutation of numbers	
true, false	create matrix (logical)	
pascal, hankel, gallery	special purpose matrices	•
blkdiag, cat	block diagonal matrix, groups several matrices into one	
diag, triu, tril,	diagonal matrix, upper and lower triangular matrix	•
fliplr, flipud, circshift	element swapping, circular shift	
repmat, reshape	matrix operation (replication, reshaping)	•
length, size, ndims, numel	length of a vector, size of a matrix, number of dim. and elements	
sparse, full	sparse and full matrix operations	
grid on, grid off	Turns grid of a graph on / off	
figure, surf	opens new figure, 3D graph surf	•

Exercise #1

360 s ↑

- create matrix \mathbf{M} of size $\text{size}(\mathbf{M}) = [3 \ 4 \ 2]$ containing random numbers coming from uniform distribution on the interval $[-0.5, 7.5]$



Exercise #2

200 s



- Consider the operation $a1 \wedge a2$, is this operation applicable to following cases?
 - $a1$ – matrix, $a2$ – scalar
 - $a1$ – matrix, $a2$ – matrix
 - $a1$ – matrix, $a2$ – vector
 - $a1$ – scalar, $a2$ – scalar
 - $a1$ – scalar, $a2$ – matrix
 - $a1, a2$ – matrix, $a1 \cdot a2$

you can always create the matrices $a1, a2$ and make a test ...

Exercise #3

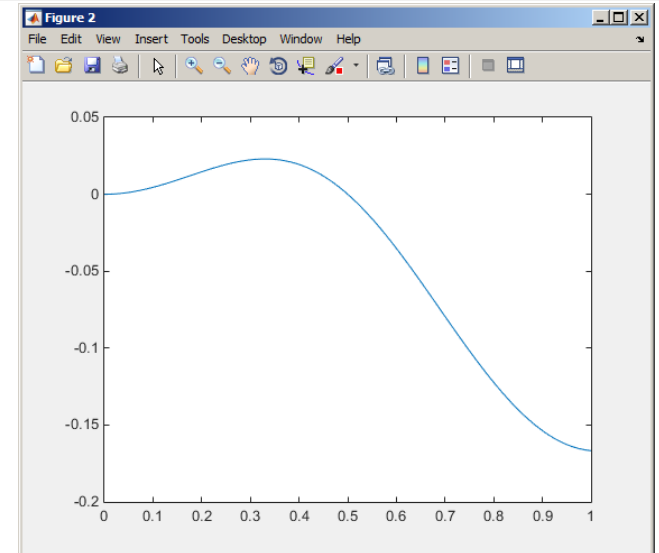
420 s ↑

- make corrections to the following piece of code to get values of the function $f(x)$ for 200 points on the interval $[0, 1]$:

$$f(x) = \frac{x^2 \cos(\pi x)}{(x^3 + 1)(x + 2)}$$

- find out the value of the function for $x = 1$ by direct accessing the vector
- what is the value of the function for $x = 2$?
- to check, plot the graph of the function $f(x)$

```
>> % erroneous code
>> x = linspace(0, 1);
>> clear;
>> g = x^3+1; H = x+2;
>> y = cos xpi; z = x.^2;
>> f = y*z/gh
```



Exercise #4

200 s ↑

- think over how many ways there are to calculate the length of the hypotenuse when two legs of a triangle are given
- make use of various Matlab operators and functions
- consider also the case where the legs are complex numbers

Exercise #5

- A proton, carrying a charge of $Q = 1.602 \cdot 10^{-19}$ C and of a mass of $m = 1.673 \cdot 10^{-31}$ kg enters a homogeneous magnetic and electric field in the direction of the z axis in the way that the proton follows a helical path; the initial velocity of the proton is $v_0 = 1 \cdot 10^7$ m/s. The intensity of the magnetic field is $B = 0.1$ T, the intensity of the electric field is $E = 1 \cdot 10^5$ V/m

- velocity of the proton along the z axis is
$$v = \frac{QE}{m}t + v_0$$

- where t is time, travelled distance along the z axis is
$$z = \frac{1}{2} \frac{QE}{m}t^2 + v_0t$$

- radius of the helix is
$$r = \frac{vm}{BQ}$$

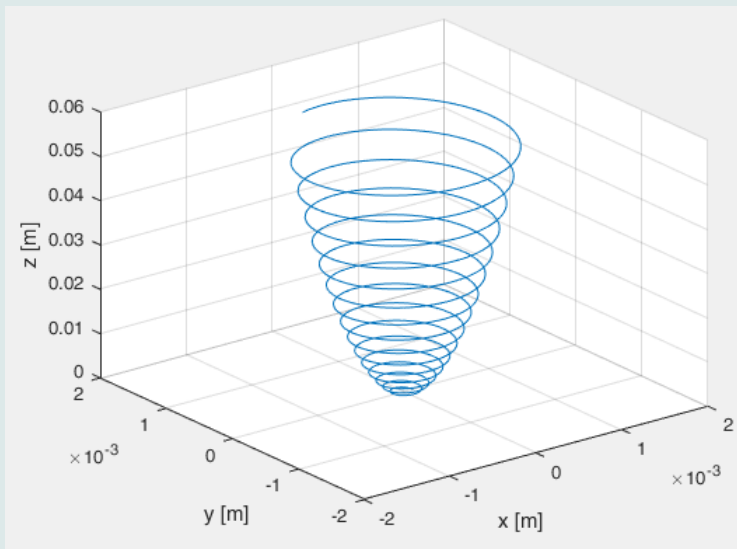
- frequency of orbiting the helix is
$$f = \frac{v}{2\pi r}$$

- the x and y coordinates of the proton are
$$x = r \cos(2\pi f t), \quad y = r \sin(2\pi f t)$$

Exercise #6

500 s ↑

- plot the path of the proton in space in the time interval from 0 ns to 1 ns in 1001 points using function `comet3(x, y, z)`



```
>> clear; close all; clc;
```

```
>> % put your code here
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> % ...
```

```
>> comet3(x, y, z)
```

Thank you!



ver. 10.1 (08/10/2018)

Miloslav Čapek, Pavel Valtr

miloslav.capek@fel.cvut.cz

pavel.valtr@fel.cvut.cz

Apart from educational purposes at CTU, this document may be reproduced,
stored or transmitted only with the prior permission of the authors.

Document created as part of A0B17MTB course.

