Lecture 2: Vectors & Matrices

B0B17MTB, BE0B17MTB – MATLAB

Miloslav Čapek, Viktor Adler, *et al.*

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@fel.cvut.cz

Februrary 25
Summer semester 2025/26

Outline

1. MATLAB Editor
2. Matrix Creation
3. Operations with Matrices
4. Excercises

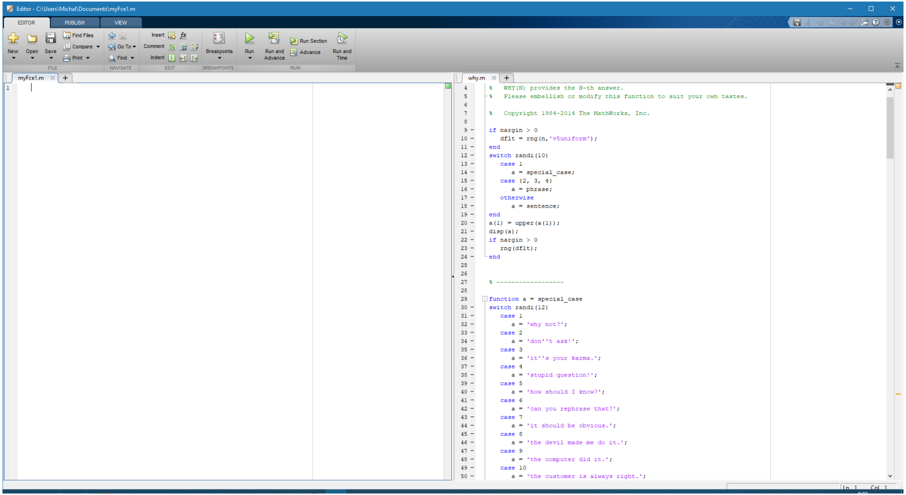
Matlab Editor

- It is often required to evaluate certain sequence of commands repeatedly ⇒ utilization of Matlab scripts (plain ASCII coding).
- The best option is to use Matlab Editor,
  - which can be opened using the following command:

```
>> edit
```

- A script is a sequence of statements what we have been up to now typing in the command line.
  - All the statements are executed one by one upon the launch of the script.
  - The script operates over Matlab base workspace data.
  - Scripts are suitable for quick analysis and solving problems involving multiple statements.
- There are specific naming conventions for scripts (and also for functions as we will see later).

Matlab Editor



Script Execution, m-files

- To execute a script:
  - F5 function key in Matlab Editor,
  - Current folder → select script → context menu → Run,
  - Current folder → select script → F9,
  - from the command line:

```
>> script_name
```

- Scripts are stored as so called m-files, .m
- Caution: If you have Mathematica installed, the .m files may be launched by Mathematica.

Data in Scripts

- Scripts can use data located in Workspace.
- Variables remain in the Workspace even after the calculation is finished.
- Operations on data in scripts are performed in the base Workspace.
- Matlab carries out commands sequentially.

Useful Functions for Script Generation I.

- Function `disp` displays value of a variable in Command Window.
  - Without displaying variable's name and the equation sign "=".
  - Can be combined with a text (more on that later).
  - Often it is advantageous to use more complicated but robust function `sprintf`.

```
a = 2^13 - 1;
b = [8*a 16*a];
b
```

```
a = 2^13 - 1;
b = [8*a 16*a];
disp(b);
```

Useful Functions for Script Generation II.

- Function `input` is used to enter variables.
  - If the function it terminated unexpectedly, the input request is repeated

```
A = input('Enter parameter A: ');
```

  - It is possible to enter strings as well:

```
str = input('Enter String str', 's');
```

Script Commenting

- MAKE COMMENTS!!
  - Important/complicated parts of code.
  - Description of functionality, ideas, change of implementation.

- Typical single-line comment:

```
% create matrix, sum all members
matX = [1, 2, 3, 4, 5];
sumX = sum(matX); % sum of matrix
```

- Multiple-line comment:

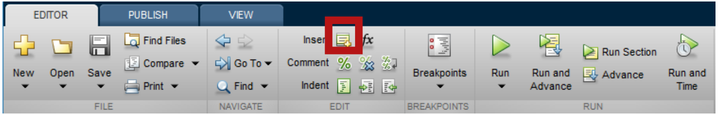
```
%{
This is a multiple-line comment.
Mostly, it is more appropriate to use
more single-line comments.
%}
```

- Cell mode enables to separate script into more blocks.

```
matX = [1, 2, 3, 4, 5];
%% CELL mode (must be enabled in Editor)
sumX = sum(matX);
```

Cell Mode in MATLAB Editor

- Cells enable to separate the code into smaller, logically compacted parts.
  - Separator `%%`.
  - The separation is visual only, but it is possible to execute a single cell – shortcuts CTRL+ENTER and CTRL+SHIFT+ENTER.



Entering Matrices Using " : " I.

- Large vectors and matrices with regularly increasing elements can be typed in using colon operator.
  - `a` is the smallest element ("from"), `x` is increment, `b` is the largest element ("to")

```
A = a:x:b
```

```
>> A = 1:4:13
A =
    1   5   9   13
```

- `b` doesn't have to be an element of the series.
  - Last element $N \cdot x$ then follows the inequality:

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

```
>> A = 1:4:10
A =
    1   5   9
```

- If `x` is omitted, the increment is set equal to 1.

```
A = a:b
```

```
>> A = 3:8
A =
    3   4   5   6   7   8
```

Entering Matrices Using " : " II.

- Using the colon operator ":" create:
  - Following vectors

$$\begin{aligned} \mathbf{u} &= [1\ \ 3\ \ \ldots\ \ 99] \\ \mathbf{v} &= [25\ \ 20\ \ \ldots\ \ -5]^{\mathrm{T}} \end{aligned}$$

  - Matrix
    - Caution, the third column can't be created using colon operator ":" only,

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

but can be created using ":" and dot operator "." (*we will see later*).

300

Entering Matrices Using " : " II.

- Using the colon operator ":" create:
  - Following vectors

$$\mathbf{u} = [1\ \ 3\ \ \ldots\ \ 99]$$
$$\mathbf{v} = [25\ \ 20\ \ \ldots\ \ -5]^{\mathrm{T}}$$

```
u = 1:2:99
v = (25:-5:-5).'
```

  - Matrix
    - Caution, the third column can't be created using colon operator ":" only,

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

```
T = [-4:-1:-6; 1:3; pi/2 pi/4 pi/6].'
```

but can be created using ":" and dot operator "." (*we will see later*).

```
T = [-4:-1:-6; 1:3; pi./(2:2:6)].'
T = [(-4:-1:-6).' (1:3).' (pi./(2:2:6)).']
```

Entering Matrices Using linspace, logspace I.

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

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

```
>> A = linspace(0, 2, 5)
A =
   0   0.5000  1.0000  1.5000  2.000
```

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

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

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

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

Entering Matrices Using `linspace`, `logspace` II.

- Create a vector of 100 evenly spaced points in the interval $[-1.15, 75.4]$.

- Create a vector of 201 evenly spaced points in the interval $[-100, 100]$ sorted in descending order.

- Create a vector with spacing of -10 in the interval $[100, -100]$ sorted in descending order.
  - Try both options using `linspace` and colon ":".

200

Entering Matrices Using `linspace`, `logspace` II.

- Create a vector of 100 evenly spaced points in the interval $[-1.15, 75.4]$.

```
u = linspace(-1.15, 75.4)
```

- Create a vector of 201 evenly spaced points in the interval $[-100, 100]$ sorted in descending order.

```
v = linspace(100, -100, 201)
```

- Create a vector with spacing of -10 in the interval $[100, -100]$ sorted in descending order.
  - Try both options using `linspace` and colon ":".

```
v = 100:-10:-100
```

```
v = linspace(100, -100, 200/10+1)
```

Entering Matrices Using Functions I.

- Special types of matrices of given sizes are needed quite often.
  - MATLAB offers a number of functions to serve the purpose.
- Example: matrix filled with zeros
  - Will be used frequently.

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

B = zeros(m, 'single') % matrix of size [m x n], of type 'single'

% see documentation for other options
```

Matrix Operations

Entering Matrices Using Functions II.

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

`ones`	matrix filled with ones
`eye`	identity matrix
`nan`, `inf`	matrix filled with NaN, matrix filled with Inf
`magic`	matrix suitable for MATLAB experiments, notice its properties
`rand`, `randn`, `randi`	matrix filled with random numbers coming from uniform and normal distribution, matrix filled with uniformly distributed random integers
`randperm`	returns vector containing random permutation of numbers
`diag`	creates diagonal matrix or returns diagonal of a matrix
`blkdiag`	construct block diagonal matrix from input arguments
`cat`	groups several matrices into one
`true`, `false`	creates a matrix of logical ones and zeros

- For further functions see MATLAB → Mathematics → Elementary Mathematics → Constants and Test Matrices.

Entering Matrices Using Functions III.

- Create following matrices
  - use MATLAB functions
  - begin with matrices you find easy to cope with.

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

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

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

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

360

Entering Matrices Using Functions III.

- Create following matrices
  - use MATLAB functions
  - begin with matrices you find easy to cope with.

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

```
M1 = nan(2)
```

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

```
M2 = ones(1, 4)
```

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

```
M3 = diag([2 3 -5])
```

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

```
M4 = diag([1 1 1], 1)
M4 = [zeros(4, 1) [eye(3); zeros(1, 3)]]
M4 = diag(ones(3, 1), 1)
```

Entering Matrices Using Functions IV.

- Try to create an empty three-dimensional array of type `double`.

```
A1 = zeros(0, 0, 0) % or ones(...) etc.
```

- Can you find another option?

```
A2 = double.empty(0, 0, 0)
```

  - `empty` is hidden (but public) method of all non-abstract classes in MATLAB.

360

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 a sparse matrix `S` out of 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.

Entering Matrices

- Quite often, there are several options how to create a given matrix.
  - It is possible to use an output of one function as an input of another function in MATLAB:

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

- Consider:
  - clarity,
  - simplicity,
  - speed,
  - convention.
- E.g. band matrix with "1" on main diagonal and with "2" and "3" on secondary diagonals.

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

  - Can be done using `for` cycle as well (see later in the semester).
  - Some other idea?

Transpose and Matrix Conjugate

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

transpose	$\mathbf{A}^{\mathrm{T}} = [A_{ij}]^{\mathrm{T}} = [A_{ji}]$	`transpose(A)` ← don't use	`A.'`
transpose + conjugate	$\mathbf{A}^{\mathrm{H}} = [A_{ij}]^{\mathrm{H}} = [\mathbf{A}^*]^{\mathrm{T}}$	`ctranspose(A)` ← don't use	`A'`

```
>> 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
```

Matrix Operations I.

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

```
P = magic(4)
```

- upper triangular matrix,

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

- lower triangular matrix,

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

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

```
V = triu(P, -1)
```

$$\mathbf{P} = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 0 & 11 & 10 & 8 \\ 0 & 0 & 6 & 12 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} 16 & 0 & 0 & 0 \\ 5 & 11 & 0 & 0 \\ 9 & 7 & 6 & 0 \\ 4 & 14 & 15 & 1 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 0 & 7 & 6 & 12 \\ 0 & 0 & 15 & 1 \end{bmatrix}$$

Matrix Operations

Matrix Operations II.

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

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

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

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

$$\mathbf{B} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{11} & A_{12} & A_{13} \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{11} & A_{12} & A_{13} \end{bmatrix}$$

- `repmat` is a very fast function.
  - Comparison of execution time of creation a $10^4 \times 10^4$ matrix filled with pi (HW, SW and MATLAB version dependent):

```
X = ones(1e4) % computed in 0.71s
Y = repmat(1, 1e4, 1e4)  % computed in 0.4s, BUT... don't use it
```

  - It is for you to consider the way of matrix creation…

Matrix Operations

Matrix Operations III.

- Function `reshape` is used to rearrange a matrix

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

- e.g.

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

```
C = reshape(A, [4, 1])
D = reshape(A, 1, 4)
D = reshape(A, [], 4)
```

$$\mathbf{C} = \begin{bmatrix} A_{11} \\ A_{21} \\ A_{12} \\ A_{22} \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} A_{11} & A_{21} & A_{12} & A_{22} \end{bmatrix}$$

Matrix Operations IV.

- Following functions are used to swap the order of
  - columns: `fliplr`,

```
B = fliplr(A)
```

  - rows: `flipud`,

```
C = flipud(A)
```

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

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

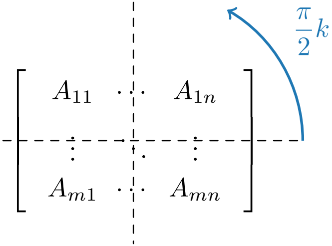
  - Indexing gives the same results (*see later*).
- The following function is used to rotate an array

```
D = rot90(A)
E = rot90(A, 2) = fliplr(flipud(A))
```

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

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

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



Matrix Operations V.

- Circular shift is also available.
  - Can be carried out along an arbitrary dimension (row-wise/column-wise).
  - Can be carried out in both directions (back/forth).
- Consider the difference between `flip` and `circshift`.

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

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

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

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

Matrix Operations VI.

- Convert matrix $\mathbf{A}$ into the form of matrices $\mathbf{A}_1$ to $\mathbf{A}_4$.

```
A = [1 pi; exp(1) -1i]
```

$$\mathbf{A} = \begin{bmatrix} 1 & \pi \\ \mathrm{e} & -\mathrm{i} \end{bmatrix}$$

- Use `repmat`, `reshape`, `triu`, `tril` and `conj`.

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

$$\mathbf{A}_2 = \begin{bmatrix} 1 & \pi & \mathrm{e} & -\mathrm{i} \end{bmatrix}$$

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

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

450

Matrix Operations VI.

- Convert matrix $\mathbf{A}$ into the form of matrices $\mathbf{A}_1$ to $\mathbf{A}_4$.

```
A = [1 pi; exp(1) -1i]
```

$$\mathbf{A} = \begin{bmatrix} 1 & \pi \\ \mathrm{e} & -\mathrm{i} \end{bmatrix}$$

- Use `repmat`, `reshape`, `triu`, `tril` and `conj`.

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

$$\mathbf{A}_2 = \begin{bmatrix} 1 & \pi & \mathrm{e} & -\mathrm{i} \end{bmatrix}$$

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

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

```
A1 = repmat(A, [1 3])
```

```
A3 = repmat(conj(A), [3 1])
```

```
A2 = reshape(A.', [1 4])
```

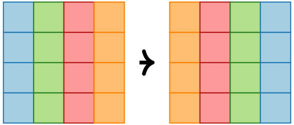
```
A4 = triu(tril(repmat(A, 3), 1), -1)
```

Matrix Operations VII.

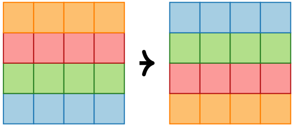
- Create the following matrix (use advanced techniques)

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

- Create matrix **B** by swapping columns in matrix **A**.



- Create matrix **C** by swapping rows in matrix **B**.



Matrix Operations VII.

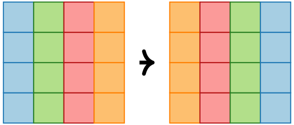
- Create the following matrix (use advanced techniques)

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

```
A = [1:3; 0:2:4; 0 0 5];
A = repmat(A, [1 2])
```

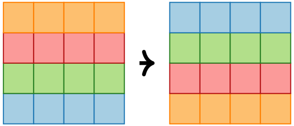
- Create matrix **B** by swapping columns in matrix **A**.

```
B = fliplr(A)
```



- Create matrix **C** by swapping rows in matrix **B**.

```
C = flipud(B)
```



Matrix Operations VIII. – Tensor Products

Kronecker tensor product

```
K = kron(A, B)
```

- Convolution kernel `A` is applied to a mask `B`.

Example:

$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\otimes\frac{1}{2}\begin{bmatrix}1 & -1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}0 & 0 & 1 & -1\\1 & -1 & 0 & 0\end{bmatrix}$$

```
kron([0 1; 1 0], [1/2, -1/2])
```

Tensor product

```
C = tensorprod(A, B, dimA, dimB)
```

- Inner product

$$\sum_n \cdots \sum_k \sum_j A_{jk\cdots n} B_{jk\cdots n} = c$$

- Outer product

$$[A_{jk\cdots n}][B_{pq\cdots t}] = [C_{jk\cdots npq\cdots t}]$$

- Tensor product

$$\sum_j \cdots \sum_p \sum_j A_{jk\cdots n} B_{pq\cdots t} = [C_{k\cdots nq\cdots t}]$$

Matrix Operations IX.

- Compare and interpret following commands.

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

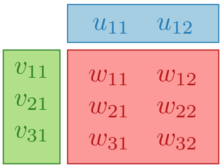
- `repmat` is powerful, but whenever possible, replace it with implicit expansion.

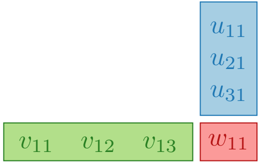
Vector and Matrix Operations

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

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

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



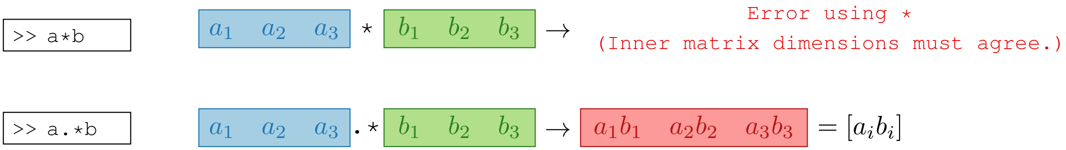


…pay attention to the dimensions of matrices!

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:

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$



Matrix Operations

Element-by-element Matrix Product

- If element-by-element multiplication of two matrices of the same size is needed, use the `.*` operator.
  - It is so called *Hadamard product/element-wise product/Schur product*: $\mathbf{A} \circ \mathbf{B}$.
  - These two cases of multiplication are distinguished:

```
>> A*B
```

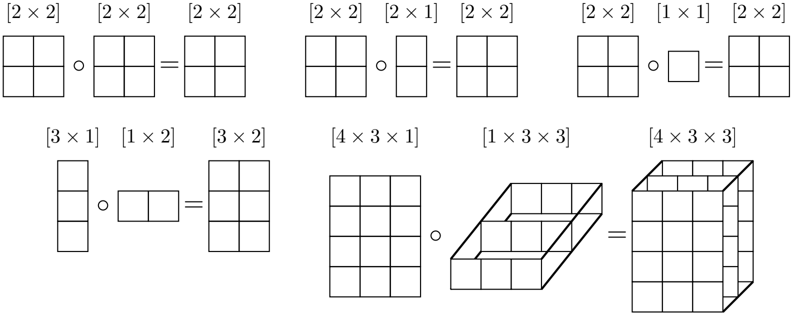
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

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

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} .* \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} \\ A_{21}B_{21} & A_{22}B_{22} \end{bmatrix}$$

Compatible Array Size

- Since MATLAB version R2016b most two-input (binary) operators support arrays that have *compatible sizes*.
  - Variables have compatible sizes if their sizes are either the same or one of them is 1 (for all dimensions).
- Examples:
  - ∘ represents arbitrary two-input element-wise operator (+, −, .*, ./, &, <, ==, ...).

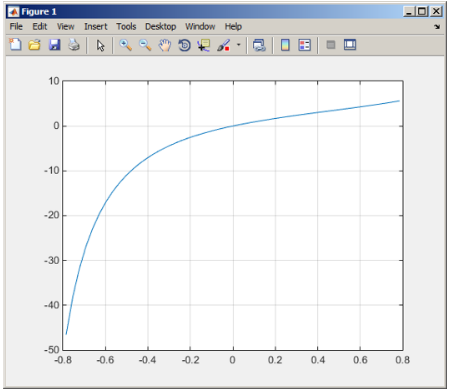


Element-wise Operations I.

- Elements-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 use of so called vectorization (*see later*).

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

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



Element-wise Operations II.

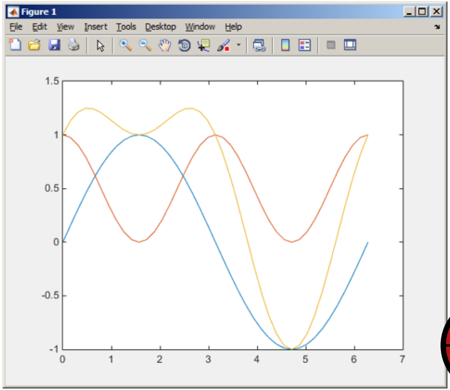
- Evaluate functions of the variable $x \in [0, 2\pi]$:

$$f_1(x) = \sin(x)$$
$$f_2(x) = \cos^2(x)$$
$$f_3(x) = f_1(x) + f_2(x)$$

- Evaluate the functions in evenly spaced points of the interval, the spacing is $\Delta x = \pi/20$.

- For verification use:

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



Element-wise Operations II.

- Evaluate functions of the variable $x \in [0, 2\pi]$:

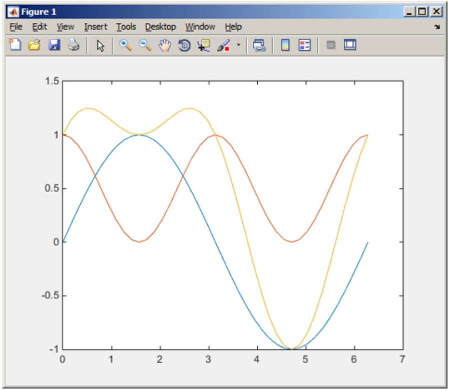
$$f_1(x) = \sin(x)$$
$$f_2(x) = \cos^2(x)$$
$$f_3(x) = f_1(x) + f_2(x)$$

- Evaluate the functions in evenly spaced points of
the interval, the spacing is $\Delta x = \pi/20$.

```
x = 0:pi/20:2*pi;
f1 = sin(x);
f2 = cos(x).^2;
f3 = f1 + f2;
```

- For verification use:

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

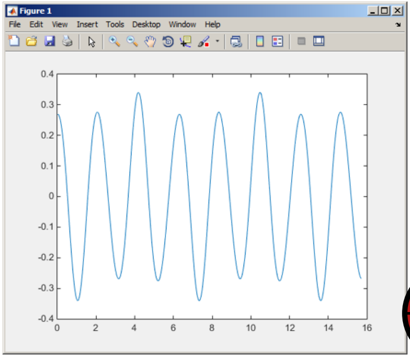


Element-wise Operations III.

- Depict graphically following functional dependency in the interval $x \in [0, 5\pi]$.
- Plot the result using the 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 multiplication of matrices of the same size.

```
>> A*B
```

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

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

240

Element-wise Operations III.

- Depict graphically following functional dependency in the interval $x \in [0, 5\pi]$.
- Plot the result using the following function:

```
clear; clc;
x = linspace(0, 5*pi, 1e5);
f4 = (-cos(3*x)) ./ ...
(cos(x) .* sin(x - pi / 5) - pi);
plot(x, f4)
```

- Explain the difference in the way of multiplication of matrices of the same size.

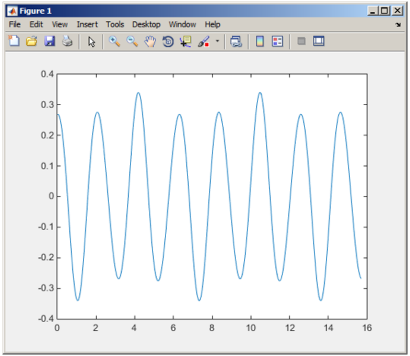
```
>> A*B
```

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

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

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

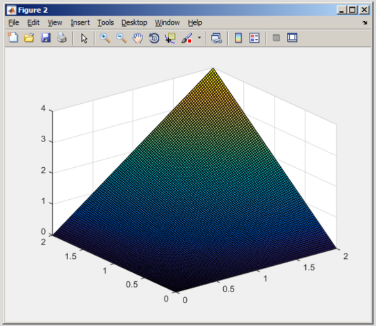
```
plot(x, f4)
```



Element-wise Operation IV.

- Evaluate the function $f(x,y) = xy, \quad x,y \in [0,2]$, 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, also try $f(x,y) = x^{0.5}y^2$ on the same interval.

```
x = linspace(0, 2, 101);
y = x.';
f = x.*y;
figure;
surf(x, y, f)
```

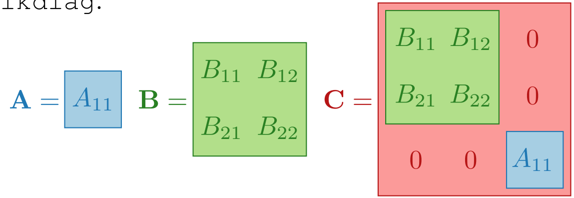


360

Matrix Operations

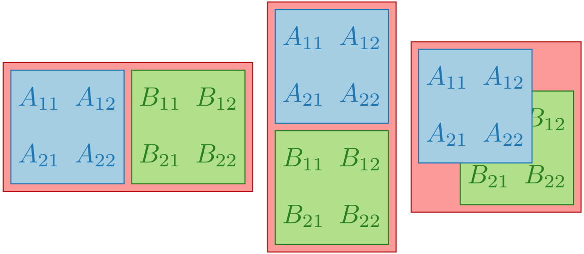
- Contruct block diagonal matrix: `blkdiag`.

```
A = 1;
B = [2 3; -4 -5];
C = blkdiag(B, A);
```



- Arranging two matrices of the same size: `cat`.

```
A = eye(2); B =
ones(2);
C = cat(2, A, B)
C = cat(1, A, B)
C = cat(3, A, B)
```



Size of Matrices and Other Structures I.

- It is often needed to know sizes of matrices and arrays.
- Function `size` returns vector giving the size of a matrix/array.

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

- Function `length` returns largest dimension of an array.

```
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.

```
ndims(A) = length(size(A))
```

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

- Function `numel` returns number of elements of a matrix/array.

```
numel(A) = prod(size(A))
```

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

- Functions `height` and `width` return number of rows and columns, respectively.

Size of Matrices and Other Structures II.

- 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, rot90(A,2))
```

- And now:
  - Find out the size of `A`.
  - Find 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`.

250

Size of Matrices and Other Structures II.

- 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, rot90(A,2))
```

- And now:
  - Find out the size of `A`.

```
size(A)
```

  - Find the number of elements of `A`.

```
numel(A)
```

  - Find out the number of elements of `A` in the “longest” dimension.

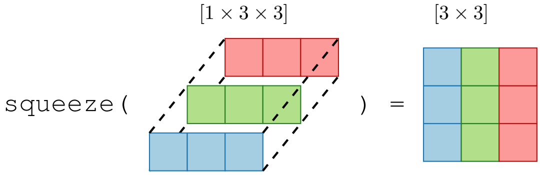
```
length(A)
```

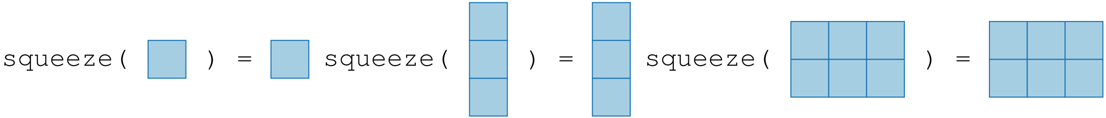
  - Find out the number of dimensions of `A`.

```
ndims(A)
```

Squeeze

- Function `squeeze` removes dimension of an array with length 1.
  - If the input is scalar, vector or array without any dimension of the length 1, the output is identical to the input.





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: `doc gallery`
- Try for instance:

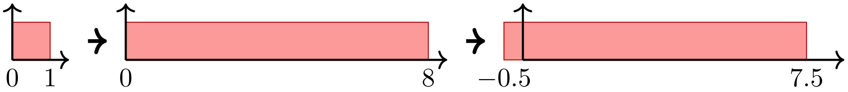
```
gallery('pei', 5, 4)
gallery('leslie', 10)
gallery('clement', 8)
```

Exercises

Exercise I.

- Create matrix **M** of size `size(M) = [3 4 2]` containing random numbers coming from uniform distribution on the interval $[-0.5, 7.5]$.

$$I(x) = (I_{\text{max}} - I_{\text{min}})\,\text{rand}(\dots) + I_{\text{min}}$$



360

Exercise I.

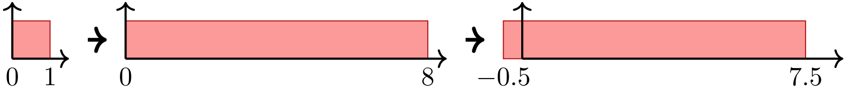
- Create matrix **M** of size `size(M) = [3 4 2]` containing random numbers coming from uniform distribution on the interval $[-0.5, 7.5]$.

$$I(x) = (I_{\max} - I_{\min})\,\mathrm{rand}(\dots) + I_{\min}$$

```
rand(   )
```

```
8*rand(   )
```

```
8*rand(   ) - 0.5
```




```
M = 8*rand(3, 4, 2) - 0.5
```

Exercise II.

- Consider the operation `a1^a2`. Is this operation applicable to the 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.^a2`

You can always create the matrices `a1`, `a2` and make a test ...

200

Exercise II.

- Consider the operation `a1^a2`. Is this operation applicable to the following cases?

`a1` – matrix	`a2` – scalar	YES
`a1` – matrix	`a2` – matrix	NO
`a1` – matrix	`a2` – vector	NO
`a1` – scalar	`a2` – scalar	YES
`a1` – scalar	`a2` – matrix	YES
`a1`, `a2` – matrix	`a1.^a2`	YES

You can always create the matrices `a1`, `a2` and make a test ...

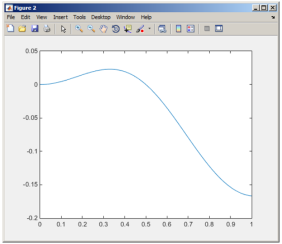
Exercise III.

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

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

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

- 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 $f(x)$.



420

Exercise III.

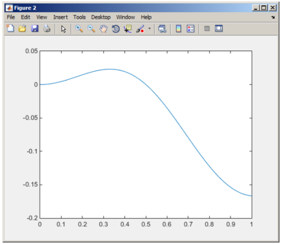
- Make corrections to the following piece of code to get values of the function $f(x)$ for 200 points on the interval $[0, 1]$:

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

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

```
% correct code
clear;
x = linspace(0, 1, 200);
g = x.^3+1; h = x+2;
y = cos(x*pi); z = x.^2;
f = y.*z./(g.*h)
plot(x, f);
```

- 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 $f(x)$.



Exercise IV.

- Create a random matrix **M** of size $N \times N$ containing only 0 and 1 elements.
- Compute the percentage of 0 elements in matrix.
- Compute number of 1 elements on the matrix main diagonal.

300

Exercise IV.

- Create a random matrix $\mathbf{M}$ of size $N \times N$ containing only 0 and 1 elements.

```
M = randi([0 1], 5, 5);
```

- Compute the percentage of 0 elements in matrix.

```
percentage = 100 - 100 * sum(sum(M)) / numel(M);
```

- Compute number of 1 elements on the matrix main diagonal.

```
mainDiagonal = diag(M);
n = sum(mainDiagonal);
```

Exercise V.a

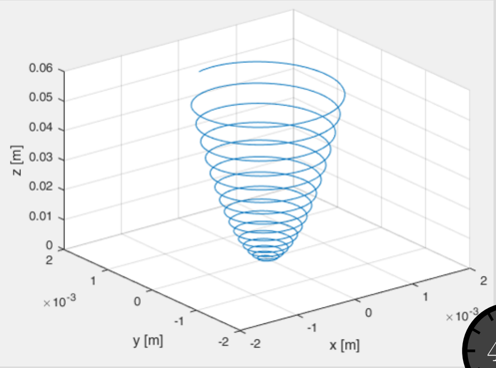
- A proton, carrying a charge of $Q = 1.602 \cdot 10^{-19}\,\mathrm{C}$ with a mass of $m = 1.673 \cdot 10^{-31}\,\mathrm{kg}$ enters a homogeneous magnetic and electric field in the direction of the z axis in the way that the proton follows a helical path; the initial velocity of the proton is $v_0 = 1 \cdot 10^{7}\,\mathrm{ms}^{-1}$. The intensity of the magnetic field is $B = 0.1\,\mathrm{T}$, the intensity of the electric field is $E = 1 \cdot 10^{5}\,\mathrm{Vm}^{-1}$
  - Velocity of the proton among the z axis is $v = \dfrac{QE}{m}t + v_0$,
  - where t is time, traveled distance along the z axis is $z = \dfrac{1}{2}\dfrac{QE}{m}t^2 + v_0 t$,
  - radius of the helix is $r = \dfrac{vm}{BQ}$,
  - frequency of orbiting the helix is $f = \dfrac{v}{2\pi r}$,
  - the x and y coordinates of the proton are $x = r\cos(2\pi f t)$, $y = r\sin(2\pi f t)$.

Exercise V.b

- 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; clc; close all;
```

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



420

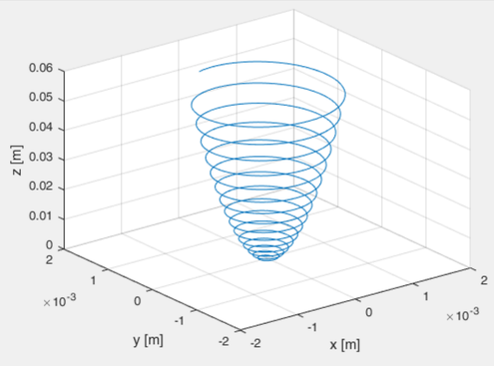
Exercise V.b

- 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; clc; close all;
```

```
m = 1.673e-31;  Q = 1.609e-19;
v0 = 1e7;  E = 1e5;  B = 0.1;
t = linspace(0, 1e-9, 1001);
v = Q * E * t / m + v0;
z = 0.5 * Q * E / m * t .^ 2 + v0 * t;
r = v * m / (B * Q);
f = v / (2 * pi * r);
x = r .* sin(2 * pi * f * t);
y = r .* cos(2 * pi * f * t);
```

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



Questions?

B0B17MTB, BE0B17MTB – MATLAB

matlab@fel.cvut.cz

Februrary 25
Summer semester 2025/26

Acknowledgement: Filip Kozák, Pavel Valtr, Michal Mašek, Vít Losenický, and Jakub Liška.