(Velice) Stručný úvod do MATLABu

pro informatiky

Martin Trnečka

Katedra informatiky, Přírodovědecká fakulta
Univerzita Palackého v Olomouci

Co je to **MAT**rix **LAB**oratory

- „The Language of Technical Computing.“
- „MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications. The language, tools, and builtin math functions enable you to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java.“
- „You can use MATLAB for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.“

K čemu je MATLAB dobrý

- Numerické výpočty.
- Analýza a vizualizace dat.
- Programování a vývoj algoritmů.
- Vývoj aplikací a jejich distribuce.
- Návrh modelů.

Kdo používá MATLAB

- NASA
- Airbus
- U.S. Air Force
- Los Alamos National Laboratory
- Xerox
- Geoscience Australia
- Vodafone
- Ford
- RWE
- Samsung
- Swiss Re
- a další `http://www.mathworks.com/company/user_stories/`

Nevýhody MATLABu

- Real-time aplikace.
- Aplikace kde je důležitý čas.
- Placený software (alternativa GNU Octave).

Další aspekty MATLABu

- Verze pro všechny běžné operační systémy a pro smartphony.
- Masivní dokumentace a široká komunita.
- Simunlink.
- Různé hardwarové platformy.

Prostředí MATLABu

Vektory a Matice

Příklad

```
>> v = [1 2 3 4 5]
v =
    1 2 3 4 5
```

Příklad

```
>> v = [1 2 3 4 5];
```

Příklad

```
>> v = [1:10]
v =
    1 2 3 4 5 6 7 8 9 10
```

Příklad

```
>> v = [1:2:10]
v =
     1 3 5 7 9
```

Příklad

```
>> v = [1 2 3 4 5]'
v =
    1
    2
    3
    4
    5
```

Přístup k prvkům vektoru

Příklad

```
>> v = [1:10]
v =
    1 2 3 4 5 6 7 8 9 10

>> v(1)
ans =
    1

>> v(1:2:5)
ans =
    1 3 5

>> v(end)
ans =
    10
```

Přístup k prvkům vektoru

MATLAB indexuje veškerá pole od 1!.

Příklad

```
>> v = [1:10];
>> v(0)
??? Subscript indices must either be real positive integers or logicals.
```

Matice

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9
```

Příklad

```
>> B = [ [1 2 3]' [4 5 6]' [7 8 9]']
B =
     1 4 7
     2 5 8
     3 6 9
```

Přístup k prvkům Matice

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9

>> A(3)
ans =
    7

>> A(4)
ans =
    2

>> A(2,2)
ans =
    5
```

Přístup k prvkům Matice

Příklad

```
>> A(1:2,1:2)
ans =
     1 2
     4 5
```

Základní operace

`plus, +`	Plus
`uplus, +`	Unary plus
`minus, -`	Minus
`uminus, -`	Unary minus
`mtimes, *`	Matrix multiply
`times, .*`	Array multiply
`mpower, ^`	Matrix power
`power, .^`	Array power
`mldivide, \`	Backslash or left matrix divide
`mrdivide, /`	Slash or right matrix divide
`ldivide, .\`	Left array divide
`rdivide, ./`	Right array divide
`idivide`	Integer division with rounding option
`kron`	Kronecker tensor product

Základní operace

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9];
>> B = [ 1 2 3; 4 5 6; 7 8 9];
>> A * 10
ans =
     10 20 30
     40 50 60
     70 80 90

>> A * B
ans =
      30  36  42
      66  81  96
     102 126 150
```

Základní operace

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9];
>> B = [ 1 2 3; 4 5 6; 7 8 9];
>> A .* B
ans =
      1  4  9
     16 25 36
     49 64 81

>> A.*B(1:2,1:2)
??? Error using ==> times
Matrix dimensions must agree.
```

Další příkazy pro vytváření matic

Příklad

```
>> zeros(3)
ans =
     0 0 0
     0 0 0
     0 0 0

>> zeros(2,3)
ans =
     0 0 0
     0 0 0

>> ones(2,3)
ans =
     1 1 1
     1 1 1
```

Další příkazy pro vytváření matic

Příklad

```
>> rand(3)
ans =
     0.9502 0.3816 0.1869
     0.0344 0.7655 0.4898
     0.4387 0.7952 0.4456

>> randi([0 1],3)
ans =
     1 0 0
     1 1 0
     0 0 1

>> []
ans =
     []
```

Užitečné příkazy MATLABu

`clear`	vymaže všechny proměnné v prostředí
`clc`	vymaže příkazové okno
`who`	seznam proměnných
`whos`	detailní seznam všech proměnných

Ukládání a načítání proměnných

Příkazy `save` a `load` slouží pro uložení a načtení všech aktuální proměnných.

Příklad

```
>> save nazev_souboru;
>> load naze_souboru;
```

Popřípadě pouze konkrétní proměnné

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9];
>> save nazev_souboru A;
```

Uložení je provedeno do souboru s příponou `.mat`.

Příkaz sum

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9];
>> sum(A)
ans =
     12 15 18

>> sum(A, 2)
ans =
      6
     15
     18

>> sum(sum(A))
ans=
    45;
```

Logická indexace matic a vektorů

Pro přístup ke konkrétnímu prvku, či prvkům matice nebo vektoru je možné použít logickou indexaci (prvek má být či nemá být vybrán). `pole(logicka podminka)`

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
>> A>5
ans=
     0 0 0
     0 0 0
     1 1 1

>> A(A>6)
ans=
     8
     6
     9
```

Logická indexace matic a vektorů

Někdy můžeme požadovat zjištění pozitivních indexů `pole(logicka podminka)`

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
>> find(A>6)
ans=
     3
     6
     9

>> find(A==10)
ans=
    Empty matrix: 0-by-1
```

Zástupný znak :

Při indexaci je možné skupinu indexů zastoupit jedním znakem : s významem „pro všechny“.

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
>> A(1,:)
ans=
     1 2 3
>> A(:,1)
ans=
    1
    4
    7
```

Velikost pole

Příklad

```
>> A = [ 1 2 3; 4 5 6];
>> size(A)
ans=
    2 3
>> size(A,1)
ans=
    2
>> size(A,2)
ans=
    3
>> [m, n] = size(A);
```

Velikost pole

Příklad

```
>> length(A)
ans =
     3
>> numel(A)
ans =
     9
```

Základní tisk proměnné

Příklad

```
>> A = [ 1 2 3; 4 5 6];
>> display(A)
>> A =
     1 2 3
     4 5 6
     7 8 9

>> disp(A)
     1 2 3
     4 5 6
     7 8 9
```

Mazání řádků a sloupců matice

Příklad

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
>> A(2,:) = []

A =

     1 2 3
     7 8 9

>> A(:,2) = []

A =

     1 3
     7 9
```

Nápověda v MATLABu

- Klasická (vyčerpávající) nápověda dostupná z menu.
- Příkaz `help`, konzolová verze nápovědy, velice rychlá a užitečná.

Příklad

```
>> help log
LOG Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also log1p, log2, log10, exp, logm, reallog.

    Overloaded methods:
        gf/log
        codistributed/log
        fints/log

    Reference page in Help browser
        doc log
```

Cvičení

1. Vytvořte následující vektor

```
v =
    10  9  8  7  6  5  4  3  2  1
```

2. Vytvořte následující matici

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

3. Vytvořte následující matici

```
A =
     1  1  0  0
     1  1  0  0
     0  0  1  1
     0  0  1  1
```

Klíčové aspekty programování v MATLABu

- MATLAB provides a high-level language and development tools that let you quickly develop and analyze algorithms and applications.
- The MATLAB language provides native support for the vector and matrix operations that are fundamental to solving engineering and scientific problems, enabling fast development and execution.
- With the MATLAB language, you can write programs and develop algorithms faster than with traditional languages because you do not need to perform lowlevel administrative tasks such as declaring variables, specifying data types, and allocating memory. In many cases, the support for vector and matrix operations eliminates the need for for-loops. As a result,one line of MATLAB code can often replace several lines of C or C++ code.
- MATLAB provides features of traditional programming languages, including flow control, error handling, and object-oriented programming (OOP). You can use fundamental data types or advanced data structures, or you can define custom data types.

Program v MATLABu

Program v MATLABu se zapisuje do M-file souborů (`*.m`) a může být zapsán jako:

1. Skript
   sekvence příkazů MATLABu.
2. Funkce
   sekvence příkazů MATLABu, definovaná názvem, vstupními a výstupními parametry. Na rozdíl od skriptu, je možné ji volat jako podprogram.

Skript Hello World!

1. Vytvoříme nový skript `hello_world.m`.
2. Do zdrojového kódu zapíšeme:

Příklad

```
display('Hello world!');
```

3. Spustíme program.

Užitečné zkratky:

`CTRL + N`	Vytvoří nový skript.
`F5`	Vykoná aktuální skript.

Dělení skriptu na buňky

Skripty je možné dělit na samostatné buňky pomocí příkazu %%, které je možno spouštět nezávisle na celém skriptu.

Příklad

```
% prvni bunka
a = 10;
%%
% druha bunka
b = 10;
```

Užitečné zkratky:

F5	Vykoná aktuální skript bez ohledu na rozdělení do buněk.
F9	Vykoná vybraný kus kodu.
CTRL + ENTER	Vykoná aktualni bunku kodu.

Komentáře a štábní kultura

Příklad

```
% komentar, radek 1
% komentar, radek 2
toto uz neni komentar
```

Užitečné zkratky:

`CTRL + R` Zakomentuje blok kodu.
`CTRL + T` Odkomentuje blok kodu.

Dlouhé příkazy je možné rozdělit na více řádků pomocí `...`

Příklad

```
a = 1;
a = a + ...
    1;
```

Užitečné zkratky:

`CTRL + I` Automaticky zarovná blok kodu.

Závorky

Funkce

Vytvoření funkce: `New File` → `Function`.
Název funkce by se měl shodovat s názvem `.m` souboru.
Vstupní proměné jsou volané hodnotou.

Příklad

```
function [ output_args ] = funkce( input_args )
%FUNKCE Summary of this function goes here
%   Detailed explanation goes here


end
```

Typy funkcí

- Hlavní funkce (`*.m` soubory) a podfunkce (platne) v rámci `*.m` souboru.
- Vnořené funkce. Přístup k proměnným nadřazené funkce.
- Ukazatele na funkce `@nazev_funkce`.
- Anonymní funkce `@(argumenty)telo_funkce`.
- Inline funkce `inline(telo_funkce, argumenty)`.

Funkce s proměnným počtem argumentů

nargin	vrací počet vstupních parametrů.
nargout	vrací počet výstupních parametrů.

Příklad

Lokální a globální proměnné

Podmínky

Příklad

```
if i < 10
    i = i + 1;
elseif i == 10
    display('preteceni');
else
    i = 0;
end
```

Relační operátory

`eq`, `==`	Equal
`ne`, `~=`	Not equal
`lt`, `<`	Less than
`gt`, `>`	Greater than
`le`, `<=`	Less than or equal
`ge`, `>=`	Greater than or equal

Logické operátory

`relop, &&`	Short-circuit logical AND
`relop, ||`	Short-circuit logical OR
`and, &`	Element-wise logical AND
`or, |`	Element-wise logical OR
`not, ~`	Logical NOT
`punct, ~`	Ignore function argument or output
`xor`	Logical EXCLUSIVE OR
`any`	True if any element of vector is nonzero
`all`	True if all elements of vector are nonzero

Priorita operátorů

1. Parentheses `()`
2. Transpose `.'`, power `.^`, complex conjugate transpose `'`, matrix power `^`
3. Unary plus `+`, unary minus `-`, logical negation `~`
4. Multiplication `.*`, right division `.\` , left division `./` , matrix multiplication `*`, matrix right division `\`, matrix left division `/`
5. Addition `+`, subtraction `-`
6. Colon operator `:`
7. Less than `<`, less than or equal to `<=`, greater than `>`, greater than or equal to `>=`, equal to `==`, not equal to `~=`
8. Element-wise `AND`, `&`
9. Element-wise `OR`, `|`
10. Short-circuit `AND`, `&&`
11. Short-circuit `OR`, `||`

Cykly

Příklad

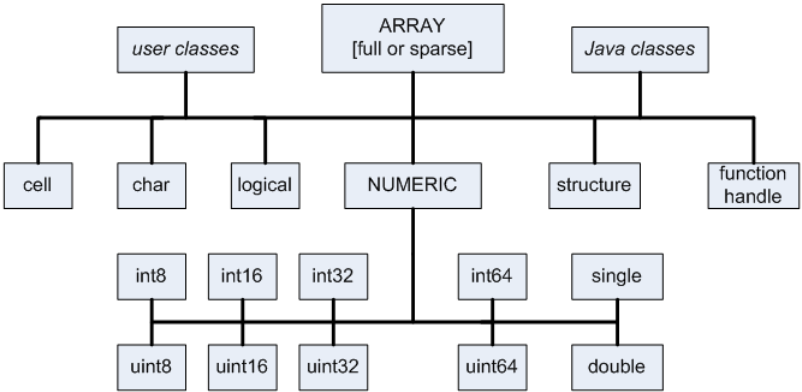
```
for i=1:10
    i
end
```

Příklad

```
while i~=10
    i
    i=i+1;
end
```

Rekurze

Datové typy


ARRAY
[full or sparse]
user classes
Java classes
cell
char
logical
NUMERIC
structure
function handle
int8
int16
int32
int64
single
uint8
uint16
uint32
uint64
double

Práce s datovými typy

Příklad

```
>> v = uint16([1: 100]);
>> v = uint8(v);
>> true
ans =
     1

>> false
ans =
     0
```

Další operátory

`bitand`	Bit-wise AND.
`bitcmp`	Complement bits.
`bitor`	Bit-wise OR.
`bitmax`	Maximum floating point integer.
`bitxor`	Bit-wise XOR.
`bitset`	Set bit.
`bitget`	Get bit.
`bitshift`	Bit-wise shift.
`union`	Set union.
`unique`	Set unique.
`intersect`	Set intersection.
`setdiff`	Set difference.
`setxor`	Set exclusive-or.
`ismember`	True for set member.

3D pole

Příklad

```
>> A = zeros(3,3,3)
A(:,:,1) =
     0     0     0
     0     0     0
     0     0     0

A(:,:,2) =
     0     0     0
     0     0     0
     0     0     0

A(:,:,3) =
     0     0     0
     0     0     0
     0     0     0
```

Indexace 3D pole

`A(1)`	První prvek v první dimenzi.
`A(10)`	První prvek v druhé dimenzi.
`A(1,2)`	Prvek na souřadnicích 1, 2 v první dimenzi.
`A(4,3)`	Změní velikost pole!
`A(1,2,3)`	Prvek na souřadnicích 1, 2 ve třetí dimenzi.
`A(:,:,3)`	Třetí dimenze.

Cell pole

Příklad

```
>> v = ['Prostejov', 'Olomouc', 'Praha']
v =
    ProstejovOlomoucPraha

>> v(1)
ans =
    P

>> v = ['Prostejov', 'Olomouc', 'Praha']
v =
     'Prostejov' 'Olomouc' 'Praha'

>> v(1)
ans =
    'Prostejov'
```

Práce s řetězci

Příklad

```
>> s = 'textovy retezec';
```

Vstup a výstup do souboru

```
dlmwrite(nazev_souboru, data, 'parametry');
```

Příklad

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> dlmwrite('output.txt', []);
>> dlmwrite('output.txt', A, '-append');
>> B = load('output.txt');
>> B =
     1 2 3
     4 5 6
     7 8 9
```

Snadné použití, ale velice pomalé pokud zapisujeme často!

Vstup a výstup do souboru (low level)

`fclose`	Close one or all open files.
`feof`	Test for end-of-file.
`ferror`	Information about file I/O errors.
`fgetl`	Read line from file, removing newline characters.
`fgets`	Read line from file, keeping newline characters.
`fopen`	Open file, or obtain information about open files.
`fprintf`	Write data to text file.
`fread`	Read data from binary file.
`frewind`	Move file position indicator to beginning of open file.
`fscanf`	Read data from text file.
`fseek`	Move to specified position in file.
`ftell`	Position in open file.
`fwrite`	Write data to binary file.

Vstup a výstup do souboru

```
file_id = fopen(filename, permission)
```

`'r'`	read
`'w'`	write (create if necessary)
`'a'`	append (create if necessary)
`'r+'`	read and write (do not create)
`'w+'`	truncate or create for read and write
`'a+'`	read and append (create if necessary)
`'W'`	write without automatic flushing
`'A'`	append without automatic flushing

Příklad

Příklad

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> fid = fopen('input.txt', 'a+');
>> fwrite(fid, A);
>> fclose(fid);

>> fid = fopen('input.txt', 'r');
>> fread(fid, 3);
ans =
    1
    4
    7
>> fclose(fid);
```

Strukturované datové typy

OOP v MATLABu

OOP v MATLABu umožňuje:

- Zapouzdření dat a metod
- Přetěžování funkcí a operátorů
- Dědičnost a agregace

naopak, neumožňuje:

- Operator `::` z C++
- Abstraktní třídy
- Šablony

OOP v MATLABu

Třída reprezentující uzel orientovaného grafu

Příklad

```
classdef node
    %NODE represent the node of a graph

    properties
        id
        value
        neighbors = [];
    end

    methods
        % constructor
        function obj = node(id, value)
            obj.id = id;
            obj.value = value;
        end
    end
end
```

Třída reprezentující orientovaný graf

Příklad

```
classdef graph
    %GRAPH represent graph

    properties
        ids = 0;
        list_of_nodes = [];
    end

    methods
        % add new node, O(1)
        function [obj] = add_node(obj, value)
            obj.ids = obj.ids + 1;
            obj.list_of_nodes = [obj.list_of_nodes, ...
            node(obj.ids, value)];
        end
```

Příklad

```
% remove node, O(n)
function obj = remove_node(obj, id)
    for i=size(obj.list_of_nodes, 2):-1:1
        if obj.list_of_nodes(i).id == id;
            obj.list_of_nodes(i) = [];
        else
            obj.list_of_nodes(i).neighbors(...
            obj.list_of_nodes(i).neighbors==id) = [];
        end
    end
end
```

Příklad

```
        % add neighbors to node, O(n)
        function obj = set_neighbors(obj, id, neighbors)
            for i=1:size(obj.list_of_nodes, 2)
                if obj.list_of_nodes(i).id == id;
                    obj.list_of_nodes(i).neighbors = neighbors;
                    return;
                end
            end
        end

        % show all edges, O(n)
        function edges(obj)
            for i=1:size(obj.list_of_nodes,2)
                disp([obj.list_of_nodes(i).id, obj.list_of_nodes(i).neighbors]);
            end
        end
    end
end
```

Použití třídy

Příklad

```
G = graph();
G = G.add_node(10); % node id 1
G = G.add_node(20); % node id 2
G = G.add_node(30); % node id 3
G = G.add_node(40); % node id 4
G = G.set_neighbors(1, [2 3]);
G = G.set_neighbors(3, [4]);
G = G.set_neighbors(4, [1]);
G.edges();
G = G.remove_node(2);
G.edges();
G = G.add_node(50); % node id 5
G.edges();
```

Základní optimalizace programů

- Prealokace paměti.
- Vektorizace (nahrazování cyklů vektorovými, či maticovými operecemi).
-

Cvičení

1. Vytvorte funci, ktera podle vstupniho parametru vypise „kladne“, „zaporne“ nebo „nula“.
2. Vytvořte funkci `matrix_min`, která bere jako vstupní argument libovolnou matici, nalezne v ní minimum a vrátí hodnotu tohoto minima a počet výskytů této hodnoty.
3. Napište funkci `setrid_vektor`, ktera setridi vektor hodnot od nejmensi po nejvetsi hodnotu v tomto vektoru.

Základní funkce pro práci s obrázky

`imread`	načtení obrázku
`imwrite`	uložení obrázku
`imshow`	zobrazení obrázku

Základní framework pro práci s obrázky

Příklad

```
obrazek = zeros(m,n);

for i = 1 : m
    for j = 1 : n
        obrazek(i,j) = 1;
    end
end

imshow(obrazek, []);
```

Převod do odstínů šedi I.

Příklad

```
obrazek_original = imread('lenna.jpg');
[m, n, o] = size(obrazek_original);
obrazek = zeros(m,n);

for i = 1 : m
    for j = 1 : n
        obrazek(i,j) = 0.229 * obrazek_original(i,j,1) ...
                     + 0.857 * obrazek_original(i,j,2) ...
                     + 0.111 * obrazek_original(i,j,3);
    end
end

imshow(obrazek, []);
```

Převod do odstínů šedi II.

Příklad

```
obrazek_original = imread('lenna.jpg');

obrazek = 0.229 .* obrazek_original(:,:,1) ...
          + 0.857 .* obrazek_original(:,:,2) ...
          + 0.111 .* obrazek_original(:,:,3);

imshow(obrazek, []);
```

Převod do odstínů šedi III.

Příklad

```
obrazek_original = imread('lenna.jpg');

obrazek = rgb2gray(obrazek_original);

imshow(obrazek, []);
```

Otočení obrázku

Příklad

```
function obrazek_rotate = my_rotate( input )
%ROTATE rotate image input to left

obrazek = imread(input);
obrazek = rgb2gray(obrazek);

[m, n] = size(obrazek);
obrazek_rotate = zeros(n, m);

for i = 1 : m
    for j = 1 : n
        obrazek_rotate(j, i) = obrazek(i, j);
    end
end
```

Cvičení - na body

1. viz. `www.rdgsw.cz/marketa/`

Základní prostředky pro ladění programu

- Automatické určování chyb
- Manuální ladění programu pomocí Brakepointů a krokování
- Vlastní výpisy
- Profiler

JIT akcelerace

- MATLAB obsahuje Just In Time akcelerátor (JIT).
- Snižuje rozdíl výkonu mezi MATLABem a tradičními programovacími jazyky jako je C/C++ a Fortran.
- Největší výhodu při použití JIT budou mít kódy, které používají následující typy dat, typy polí a programové konstrukce:
  - double, init a char;
  - prázdná pole, skaláry, vektory;
  - smyčky a podmíněné příkazy s podmínkami, které hodnotí skalární hodnoty.
- JIT-akcelerátor snižuje potřebu uživatelů psát MEX soubory, aby tak zvyšovali výkon svých aplikací.

Toolboxy v MATLABu

Vlastní toolbox