Matlab – základy

Úvod do programovacích jazyků MATLAB a Octave (a knihovny NumPy v Pythonu)

1 Úvod

1.1 Proč MATLAB

- Skriptovací jazyk
- Rychlý vývoj
- Vhodný pro matematické operace
- Java například na předzpracování hrozně pomalá, ale vyvíjet všechno v C/C++ trvá dlouho
- Dnešní trend C/C++ na výpočetně náročné operace, Skriptovací jazyk(MATLAB, Python, Perl) na různé předzpracování a na prototypování. ⇒ mnohem kratší vývojový čas při zachování efektivity

1.2 Základní vlastnosti

MATLAB = MATrix LABoratory – Základní datový typ je matice. V podstatě všechno je matice:

- číslo je matice 1x1
- pole (vektor) matice n x 1 resp. 1 x n
- text je matice jeden řádek textu = jeden řádek matice, řádek je vektor znaků
- černobílý obrázek je matice intenzit pixelů o rozměrech šířka x výška
- barevný obrázek je trojrozměrná matice šířka x výška x 3 (RGB)
- atd...

Naproti tomu NumPy má zvláštní typy pro skalár, vektor a matici.
Octave je volně dostupný ořezaný MATLAB.
MATLAB má mnohem více high-level funkcí, Octave je syntakticky bližší C-like jazykům.

1.3 Práce s prostředím

```
clc # clear console
clear all # delete all variables
close all # close all windows
```

Každý skript by měl začínat těmito třemi příkazy, aby se různé po sobě spuštěné skripty vzájemně nežádoucím způsobem neovlivňovaly.

```
addpath # načte skripty v zadané cestě
pwd
cd
disp # vypíše obsah proměnné do konzole
who # vypíše všechny proměnné v aktuálním rozsahu platnosti
```

V implicitní proměnné *ans* se nachází vždy výsledek poslední operace.

Copyright © 2017-2022 Ondřej Pražák.

1.4 Konvence

```
A, B, C, X, Y ... # matice velkými písmeny
a, b, c, x, y ... # vektory malými písmeny
```

2 Vytvoření matice

2.1 Výčtem prvků

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

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Prvky na řádcích oddělené čárkou, řádky středníkem.
Čárky můžeme vynechat:

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

Výčtem prvků v NumPy:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

2.2 Speciální matice

```
zeros(10) # matice samých nul (10x10)

ones(3, 3) # matice samých jedniček (3x3)

rand(4, 3, 2) # trojrozměrná matice (tensor) náhodných reálných čísel
              # z roznoměrného rozdělení (0, 1)

randn(3, 3) # náhodná čísla z normovaného normálního rozdělení

diag(x) # čtvercová diagonální matice s prvky vektoru x na diagonále

eye(n) # jednotková matice řádu n
```

NumPy:

```
import numpy as np
np.zeros(10) # nulový vektor a deseti prvcích
np.ones([3, 3]) # matice samých jedniček (3x3)

np.random.uniform(size=[4, 3, 2]) # trojrozměrná matice (tensor)
        # náhodných reálných čísel z roznoměrného rozdělení (0, 1)

np.random.standard_normal([3, 3])
        # náhodná čísla z normovaného normálního rozdělení

np.diag(x) # čtvercová diagonální matice s prvky vektoru x na diagonále

np.eye(n) # jednotková matice řádu n
```

2.3 Velikost matic

V MATLABu je všechno matice o minimálně dvou rozměrech (číslo je matice 1x1, vektor n x 1 nebo 1 x n)

```
size(X) # vrací řádkový vektor jednotlivých rozměrů matice (min. 2 prvky)
length(x) # vrací počet pruků vektoru (interně vrací max(size(x)))
                # POZOR na length(X)
reshape(X) # Změní rozměry matice na ty specifikované
```

Funkce length není v MATLABu.
NumPy:
Vektor v Numpy má jen jednu dimenzi a je tedy vždy řádkový.

```
np.shape(X) | x.shape # vrací ntici velikostí jednotlivých dimenzí
                      # (pro vektor skalár)
np.reshape(X) # Změní rozměry matice na ty specifikované
np.expand_dims(x, 0) # řádkový vektor s první dimenzí velikosti 1
np.expand_dims(x, 1) # sloupcový vektor
```

3 Matematicko-logické operace

1. **Matice-skalár**

   ```
   A + 2;
   2 * A;
   ```

   Aplikuje se postupně na všechny prvky matice

2. **Matice-matice**

```
A + B

A * B # maticové násobení

A .* B # násobení po prvcích

A ^ 2 # umocnění na druhou (maticově A * A)
A .^ 2 # umocnění jednotlivých prvků

# násobení matic není komutativní ->
A / B    # klasické dělení zprava (ekvivalentní A * B^(-1))
A \ B    # dělení zleva :-) (ekvivalentní A^(-1) * B)

&& # logický součin
|| # logický součet
== # porovnání matic
~= # nerovnost
!= # možné pouze v Octave (v MATLABu ne)
```

Zkrácené zápisy jako A++ A += 5 atd. jsou opět možné pouze v Octave ⇒ nepoužívat

3. **Matice-vektor (broadcast)**

Při použití operací "po prvcích"se řádkový vektor se aplikuje na všechny řádky matice a sloupcový vektor na všechny sloupce. Obecně pro matice vyšších řádů: Všechny dimenze velikosti 1 se broadcastují. To znamená že výsledek operace je stejný, jako by se matice kopírováním rozšířila na stejný rozměr, který má v dané dimenzi druhá matice.

Příklad:

```
A = [1 2 3; 4 5 6; 7 8 9];
v = [1 2 3];
A .* v
```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 8 & 10 & 12 \\ 21 & 24 & 27 \end{pmatrix}$$

```
A .* v'
```

$$A = \begin{pmatrix} 1 & 4 & 9 \\ 4 & 10 & 18 \\ 7 & 16 & 27 \end{pmatrix}$$

Numpy:

```
A + B

A * B # násobení po prvcích

np.matmul(A, B) # maticové násobení

A ** 2 # umocnění jednotlivých prvků
np.linalg.matrix_power(A, 2) # umocnění na druhou (maticově A * A)

# maticové dělení pomocí násobení inverzní maticí


np.logical_and() # logický součin
np.logical_or() # logický součet
np.logical_not()
== # porovnání matic
!= # nerovnost
```

```
inv(A) # inverze matice A
pinv(A) # pseudo-inverze matice A
transpose(A)
A'        # transpozice
det(A) # výpočet determinantu
```

pseudo-inverze Je zobecněním inverze matice pro singulární a obdélníkové matice. Pseudo-inverzní matice má následující vlastnosti:

1. $AA^+A = A$, (AA^+ nemusí být jednotkovou maticí, ale mapuje sloupce A sami na sebe;
2. $A^+AA^+ = A^+$
3. $AA^+)^* = AA^+$, (AA^+ je Hermitovská);
4. $(A^+A)^* = A^+A$, (A^+A je Hermitovská).

Pseudo-inverzní matici je možné vypočítat například pomocí singulárního rozkladu, kde:

$$A = U\Sigma V^*$$

$$A^+ = V\Sigma^+ U^*$$

Numpy:

```
np.linalg.inv(A) # inverze matice A
np.linalg.pinv(A) # pseudo-inverze matice A
np.transpose(A)
np.linalg.det(A) # výpočet determinantu
```

3.1 Redukční operace

Mnoho aritmetických operací nad maticemi má redukční charakter (redukuje počet dimenzí). Příkladem může být součet nebo průměr prvků. V Matlabu mají redukční operace druhý nepovinný parametr, dimenzi, která se má redukovat, výchozí hodnota je typicky 1 (první dimenze). Příklad:

```
A = [1 2 3; 4 5 6; 7 8 9]
sum(A), sum(A, 1) # součet řádků
sum(A, 2) # součet sloupců

A ./ sum(A .^ 2, 1) # normalizace sloupců
A ./ sum(A .^ 2, 2) # normalizace řádků
```

NumPy:

```
import numpy as np

A = [1 2 3; 4 5 6; 7 8 9]
sum(A) # součet všech prvků
sum(A, 0) # součet řádků
sum(A, 1) # součet sloupců

A / np.sum(A ** 2, 0) # normalizace sloupců
A / np.sum(A ** 2, 1) # POZOR není normalizace řádků
A / np.transpose(np.sum(A ** 2, 1)) # POZOR také špatně,
                        # transpose() nevytvoří sloupcový vektor (překvapivě)
A / np.expand_dims(np.sum(A ** 2, 1), 1) # správná normalizace řádků
                        # (je poteba vytvořit sloupcový vektor)
```

3.2 Příklad

```
x = [2 5 1 6]
      a = x + 16
      c = sqrt(x) or c = x.^(0.5)
      d = x.^2 or d = x.*x
```

4 Indexování a operátor :

V MATLABu a Octave se indexuje od 1 a index se píše do kulatých závorek, např. $A(1,2)$ vrátí prvek v prvním řádku a druhém sloupci. matici je možné indexovat i jedním indexem, prvky jsou pak číslované takto: $\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$

Matici je možné indexovat opět maticí. Výsledkem je potom matice o stejných rozměrech jako indexační matice, kde je každý prvek nahrazen prvkem z indexované matice na odpovídajícím indexu. Příklad:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$i = \begin{pmatrix} 1 & 4 & 6 \end{pmatrix}$$

$$A(i) = \begin{pmatrix} 1 & 2 & 8 \end{pmatrix}$$

Numpy:

Jednorozměrné indexování v NumPy vrací celý sloupec, pro výběr několika prvků je třeba indexovat nticí polí, nebo použít *reshape*(). **POZOR** V NumPy jsou prvky indexované obráceně než v Matlabu (na první pohled intuitivnější), tedy takto:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

4.1 Hledání prvků

funkce *find* vrací pole indexů nenulových prvků Pokud tedy chceme zjistit, na kterém indexu vektoru v se nachází číslo 3, uděláme to následovně:

$$find(v == 3)$$

výsledkem operace $v == 3$ je vektor o stejné délce jako v, který obsahuje jedničky na indexech, kde v obsahuje hodnotu 3 a nuly na ostatních.

výraz $find('logick_v raz')$ tedy vrátí indexy prvků, které zadaný logický výraz splňují.

V **NumPy** je stejná funkce *np.where*()

4.1.1 Logické indexování

Když indexujeme maticí s logickými hodnotami (0 a 1) Matlab vrátí vektor prvků indexované matice na indexech nenulových prvků indexační matice. **NumPy** se chová úplně stejně.

```
x = [1: 10];
i = zeros(1, 10);
i(1:3:end) = 1;
x(logical(i))
```

NumPy:

```
x = np.arange(10)
i = np.zeros(10)
i[1::3] = 1
x[i == 1]
```

4.2 Operátor :

V Matlabu obecně slouží k vytvoření aritmetické posloupnosti

```
A = 1: 10 # vytvoří vektor s prvky 1 2 3 4 5 6 7 8 9 10

# v případě třech parametrů je druhý parametr krok (default 1)
A = 1: 2 : 10 # A = [1 3 5 7 9]

# krok nemusí být celočíselný
A = 1 : 0.1 : 2

# ani kladný
A = 10 : -1 : 1
```

jelikož dvojtečkový operátor vytvoří vektor, je jím možné indexovat (za předpokladu, že jsou meze celočíselné a krok také).

Navíc při indexování lze využít další věci

klíčové slovo *end* značí konec indexované oblasti (pole)

```
a = [1 2 3 4 5]

a(3 : end)
3 4 5
a(0 : end)
1 2 3 4 5
```

předchozí příklad je pro vektor sice nesmyslný, má ale velké využití u dvou a více rozměrných matic kde vybíráme například celý sloupec. Používá se tak často že pro něj byla vytvořena zkratka a můžeme psát pouze:

```
A(:, 2) # vybere druhý sloupec matice A
```

4.2.1 Příklad

```
x = [2 5 1 6]
      b = x(1:2:end) + 3

A = [ 2 4 1 ; 6 7 2 ; 3 5 9]
      x1 = A(1,:)
      y = A(end-1:end,:)
      c = sum(A)
      d = sum(A,2) or d = sum(A')'
      N = size(A,1), e = std(A)/sqrt(N)
```

NumPy:

V Pythonu slouží operátor : pouze pro řezy. Pro vytvoření aritmetické posloupnosti se používá funkce *arange*(). Dalším rozdílem je pořadí operandů. V Pythonu má operátor : následující syntax: $A[start : stop : step]$

Úlohy (Matlab):

1. Vyberte z vektoru pouze prvky na sudém indexu
2. Vytvořte jednotkovou matici bez použití funkcí eye() a diag()
3. Vytvořte matici 5x10 s prvky od jedné do padesáti rostoucími po řádcích
4. Vytvořte následující matici (o libovolných rozměrech typu Nx2N-1):

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

Úlohy (NumPy):

1. Vyberte z vektoru pouze prvky na sudém indexu
2. Vytvořte jednotkovou matici bez použití funkcí eye() a diag()
3. Vytvořte matici 5x10 s prvky od jedné do padesáti rostoucími po sloupcích
4. Vytvořte následující matici (o libovolných rozměrech typu Nx2N-1):

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

5. Od každého prvku matice odečtěte nejmenší prvek v odpovídajícím řádku.
6. Všechny prvky v matici, které jsou menší, než průměr odpovídajícího sloupce, nahraďte nulou.
7. Najděte v matici dva nejpodobnější řádkové vektory (jako metriku ppodobnosti použijte kosínus úhlu mezi vektory).
8. Zadanou matici "vymaskujte"na dolní trojúhelníkovou (prvky nad diagonálou nahraďte nulou).

5 Podmínky a cykly

```
if podmínka
    příkazy
elseif podmínka
    příkazy
else
    příkazy
end
```

```
for i = 1:10 #prochází přes všechny prvky pole přiřazeného do i
    příkazy
end
```

```
while podmínka
    příkazy
end
```

5.1 Příklad

1. Naprogramujte následující hru. Program na začátku vygeneruje náhodné číslo 1-128 a uživatel se ho snaží uhodnout. Odpovědi programu jsou $<$, $>$, $=$. Uživatel má sedm pokusů. Užitečné funkce: *input()* a *disp()*.

6 Vizualizace

```
plot(x, y)  # napojí se na poslední vizualizační okno
# (v případě, že žádné neexistuje, vytvoří ho) a vykreslí do něj xy graf.
linspace(min, max, granularity)

hist(x) # opět do posledního okna vykreslí histogram.
figure # vytvoří nové vizualizační okno.
hold on # pokud je vypnuto, vykreslením nových hodnot smažeme hodnoty
hold off #  předchozí. Když je zapnuto, vykreslíme další křivku,
    # bez odstranění té předchozí.

title()
xlabel()
ylabel()
legend()
axis(xMin, xMax, yMin, yMax)
```

NumPy:

```
import numpy as np
import matplotlib.pyplot as plt

plt.plot(x, y)  # napojí se na poslední vizualizační okno
# (v případě, že žádné neexistuje, vytvoří ho) a vykreslí do něj xy graf.
np.linspace(min, max, granularity)

plt.hist(x) # opět do posledního okna vykreslí histogram.
plt.figure() # vytvoří nové vizualizační okno.
plt.show()

plt.title()
plt.xlabel()
plt.ylabel()
plt.legend()
plt.axis(xMin, xMax, yMin, yMax)
```

7 Funkce

```
function [výstupní parametry] = nazev(vstupní parametry)
příkazy
end
```

návratová hodnota se vrací inicializací výstupních parametrů. výstupních parametrů může být více v takovém případě se funkce volá takto:

```
[a, b] = funkce(x);
```

7.1 Příklad

```
function [ y ] = fact( x )
if(x == 0)
        y = 1;
else
        y = x * factorial(x - 1);
end
end
```

1. Napište maticovou verzi funkce pro výpočet faktoriálu (funkci která bude mít na vstupu matici libovolných rozměrů)

8 Buňky (cell)

Nehomogenní pole, které může obsahovat několik polí různých rozměrů a typů

```
A = [1 2 3; 1 2 3; 1 2 3];  #pole
C = {A sum(A) det(A) };     #buňka
```

Indexuje se složenými závorkami:

```
C{1}                        A
C{2}                        3 6 9
C{3}                        0
```

9 Struktury

Hierarchické datové typy.

```
a = "field2";
x.a = 1;    ve struktuře x vytvoří člen a a přiřadí mu hodnotu 1
x.(a) = 2;
x
      x =
        {
          a =  1
          field2 =  2
        }
```

10 Řetězce

- Jednořádkový text je v matlabu řádkový vektor znaků.
- Víceřádkový text je matice znaků, nevýhodou je, že jednotlivé řádky musí být stejně dlouhé.
- Funkce *char*() spojí řetězce a doplní je na konci bílými znaky tak, aby byly řádky stejně dlouhé. Zároveň čísla převede na znaky.
- Funkce *double*() převede chary zpět na čísla

10.1 Příklad

1. Naprogramujte dvě funkce en*crypt()* a *decrypt()*. Postup šifry je následující: Znaky zprávy převedete na čísla, ke každému číslu přičtete konstantu (klíč) a na každé sudé místo vložíte náhodné číslo. Funkce decrypt provede inverzní operaci (dešifruje zprávu).