POČÍTAČOVÁ PODPORA V ELEKTROTECHNICE

Ing. Lenka Šroubová, Ph.D.
lsroubov@kte.zcu.cz

Ing. Petr Kropík, Ph.D.
pkropik@kte.zcu.cz

Katedra teoretické elektrotechniky
Fakulta elektrotechnická
Západočeská univerzita v Plzni

Místnost: EK602

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Základy práce s výpočetními systémy

Použití některých znaků a symbolů

`'` - transpozice (odlišuje se od překlopení pro komplexní čísla)

`.'` - překlopení matice pole podle hlavní diagonály,

např.:

```
A.'
ans =
        1    4
        2    5
        3   -6
```

`{}` - uzavírají (obklopují) struktury (složené proměnné)

- v případě tzv. „buňkových“ polí, vektorů – cell array
- u seznamů v příkazu **`switch-case`**

Základy práce s výpočetními systémy

Mějme proměnné: `PROM=1;` `PROm=1;` `PRom=1;` `Prom=1;`

`who` `%` výpis všech existujících proměnných

```
Your variables are:
PROM   PROm   PRom   Prom
```

`whos` `%` výpis všech existujících proměnných včetně rozměru, obsažené paměti a třídy

```
Name        Size                Bytes  Class     Attributes
  PROM      1x1                     8  double
  PROm      1x1                     8  double
  PRom      1x1                     8  double
  Prom      1x1                     8  double
```

`clear Prom` – maže proměnnou `Prom`

`clear` – maže všechny proměnné

`%`– značí, že zbytek řádku je komentář, (v Octave lze užít `#`)

Základy práce s výpočetními systémy

Způsob zobrazení čísel na obrazovce

format – nastavuje způsob zobrazení čísel na obrazovce

format short nebo **format** – krátké zobrazení desetinných čísel (4 místa)

format long – dlouhé zobrazení desetinných čísel (7 / 15 míst – float / double)

format rat – racionální zobrazení desetinných čísel - přibližné zlomky

format hex – zobrazení čísel v hexadecimální (šestnáctkové) soustavě

format + – tiskne **+** na místě kladných čísel, **-** na místě záporný a mezeru místo 0

format bank – „bankovní“ formát – dvě desetinná místa – peníze

format short e, **format long e** – inženýrský formát, pevný počet des. míst

format short g, **format long g** – „chytrý“ formát, vybere nejkratší verzi

Základy práce s výpočetními systémy

Jednoduchý tisk proměnné

```
nazev = 2+3;
nazev
nazev   =     5

disp(nazev)
   5
```

- to jak tento příkaz tiskne hodnoty proměnných je dáno nastavením pomocí příkazu **`format`** z předchozí ho snímku

- později budeme používat **`fprintf`**, který umožňuje jemnější a nezávislé nastavení způsobu tisku každé hodnoty a také tisk do textových souborů

Základy práce s výpočetními systémy

Vestavěné funkce

Některé příkazy pracují jako vestavěné funkce (built-in functions)
Např.:

```
sin(pi/2)
      ans = 1
```

- funkce jedním vstupem a jedním výstupem

```
v = size(A)
v =  2  3
```

- funkce vrátí počet řádků a sloupců matice **A**.
- funkce s jedním vstupem a jedním výstupem, výstupem je dvouprvkový řádkový vektor (počet řádků, počet sloupců matice).

- pokud existuje více výstupních argumentů (prvky vektoru v hranatých závorkách [ ]), počet řádků je přiřazen do prvního a počet sloupce do druhého, *např.*:

```
[r,s] = size(A)
      r = 2
      s = 3
```

help *něco* – textová nápověda, *např.* **help sin**, **help size**, apod.

Základy práce s výpočetními systémy

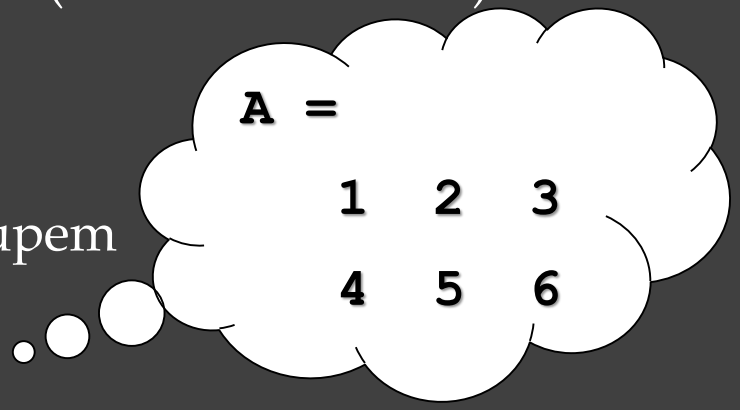
Vestavěné funkce

Některé příkazy pracují jako vestavěné funkce (built-in functions)

Např.:

```
sin(pi/2)
      ans = 1
```

- funkce jedním vstupem a jedním výstupem




```
A = [1,2,3;4,5,6]; v = size(A)
v =  2  3
```

- funkce vrátí počet řádků a sloupců matice **A**.
- funkce s jedním vstupem a jedním výstupem, výstupem je dvouprvkový řádkový vektor (počet řádků, počet sloupců matice).

- pokud existuje více výstupních argumentů (prvky vektoru v hranatých závorkách [ ]), počet řádků je přiřazen do prvního a počet sloupce do druhého, *např.*:

```
[r,s] = size(A)
      r = 2
      s = 3
```

help *něco* – textová nápověda, *např.* **help sin**, **help size**, apod.

Základy práce s výpočetními systémy

Tvorba vlastních funkcí

- zjednodušují a zpřehledňují program
- mohou být definovány přímo na příkazovém řádku během interaktivní relace nebo v externích souborech (m-file, přípona .m)
- lze je volat stejně jako zabudované funkce
- syntaxe:

```
function [vystup] = nazev(vstup)
        tělo funkce, příkazy
end
```

- vstup, výstup nejsou povinné, pokud je výstup jen jeden, nejsou třeba hranaté závorky

```
function vystup = nazev(vstup)
```

- vstupních parametrů může být více, oddělují se čárkami

```
function vystup = nazev(a,b,c)
```

- klíčové slovo `end` je možné vynechat v případě zadání funkce v samostatném souboru (m-file)
- hlavička funkce zajišťuje přenos dat z a do funkce
- proměnné ve funkci jsou lokální vzhledem k funkci (po skončení posledního příkazu funkce zaniknou), výstupní proměnné zůstanou zachovány
- volání funkce – jejím názvem

Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 1

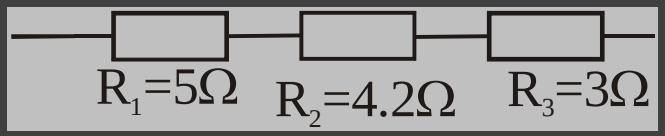
Vytváření vlastních funkcí v MATLABu , tzv. m-file

Menu: File –> New –> Function

(v novějších verzích MATLABu **New –> Function**)

- sériové zapojení rezistorů (výpočet výsledného odporu)

```
function vystup=seriove(vstup)
   vystup = sum(vstup);
end
```



Volání funkce pro 3 rezistory zapojené sériově

```
Ro = [5,4.2,3]; % vektor se zadanými hodnotami
                odporů
Rcelk = seriove(Ro) % volání funkce seriove se
                vstupním parametrem Ro, výstup R
```

Výsledek:

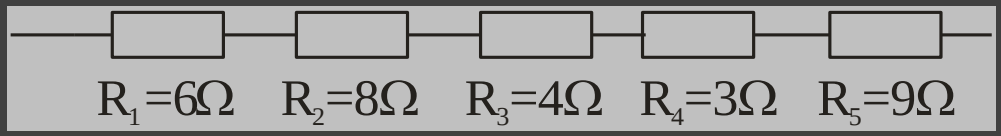
```
Rcelk  =  12.200
```

Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 1 - pokračování

- sériové zapojení rezistorů (výpočet výsledného odporu)

```
function vystup=seriove(vstup)
   vystup = sum(vstup);
end
```



Volání funkce pro 5 rezistorů zapojených sériově

```
Ro_jiny = [6,8,4,3,9]; % vektor se zadanými
                       hodnotami odporů
R = seriove(Ro_jiny) % volání funkce seriove se
                       vstupním parametrem Ro_jiny
```

Výsledek:

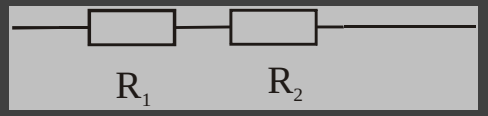
```
R = 30
```

Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 2

- sériové a paralelní zapojení 2 rezistorů (výpočet výsledného odporu)
- funkce se dvěma vstupy a výstupem dvouprvkovým vektorem

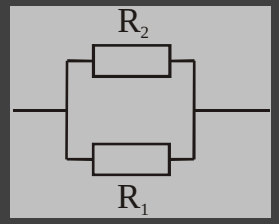
```
function [Rs,Rp] = seriove_paralelne(Rvst1,Rvst2)
 Rs = Rvst1 + Rvst2;
 Rp = (1/Rvst1+1/Rvst2)^(-1);
end
```



Volání funkce: pro R_1=11 Ω, R_2=6,5 Ω

```
R1=11;
R2=6.5;

[Rs,Rp] = seriove_paralelne(R1,R2)
 Rs =  17.500
 Rp =  4.0857
```



Základy práce s výpočetními systémy

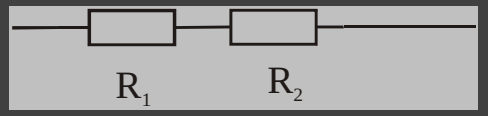
Tvorba vlastních funkcí – příklad 2

- sériové a paralelní zapojení 2 rezistorů (výpočet výsledného odporu)

```
function [Rs,Rp] = seriove_paralelne(Rvst1,Rvst2)
 Rs = Rvst1 + Rvst2;
 Rp = (1/Rvst1+1/Rvst2)^(-1);
end
```

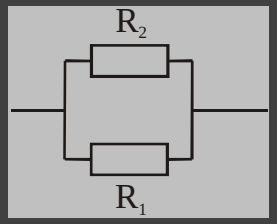
Volání funkce: pro R_1=5 Ω, R_2=10 Ω

```
[s,p]=seriove_paralelne(5,10)
 s =   15
 p =   3.3333
```



nebo pro R_1=4 Ω, R_2=4 Ω

```
[x,y]=seriove_paralelne(4,4)
 x =   8
 y =   2
```



Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 3

- výpočet obsahu a obvodu kruhu

Vytvořená funkce:

```
function [S,o] = obsah_obvod(r)
  S = pi.*(r.^2);
  o = 2.*pi.*r;
end
```

Soubor je nutné uložit.
Save as: obsah_obvod.m

Volání funkce v příkazovém okně Command Window

```
[obs,obv] = obsah_obvod(10)
  obs =  314.16
  obv =  62.832
```

Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 3 - pokračování

- výpočet obsahu a obvodu kruhu

`whos` % - vypíše proměnné držené v operační paměti

```
Name        Size                Bytes  Class     Attributes

  obs       1x1                     8  double
  obv       1x1                     8  double
```

- `r` je lokální proměnná (s ukončením funkce zaniká), výstupní proměnné zůstanou zachovány

Základy práce s výpočetními systémy

Tvorba vlastních funkcí – příklad 3 - pokračování

- výpočet obsahu a obvodu kruhu

Funkce bez výstupu:

```
function obsah_obvod2(r)
  S = pi.*(r.^2);
  o = 2.*pi.*r;
  disp('Obsah');
  disp(S);
  disp('Obvod');
  disp(o);
end
```

Volání funkce v příkazovém okně Command Window

```
obsah_obvod2(5)
Obsah
    78.5398
Obvod
    31.4159
```

`r, S, o` – lokální proměnné (s ukončením funkce zanikají), výstupní proměnná není

`whos`

=> po příkazu whos se nevypíše nic

Základy práce s výpočetními systémy

Základy tvorby 2D grafů

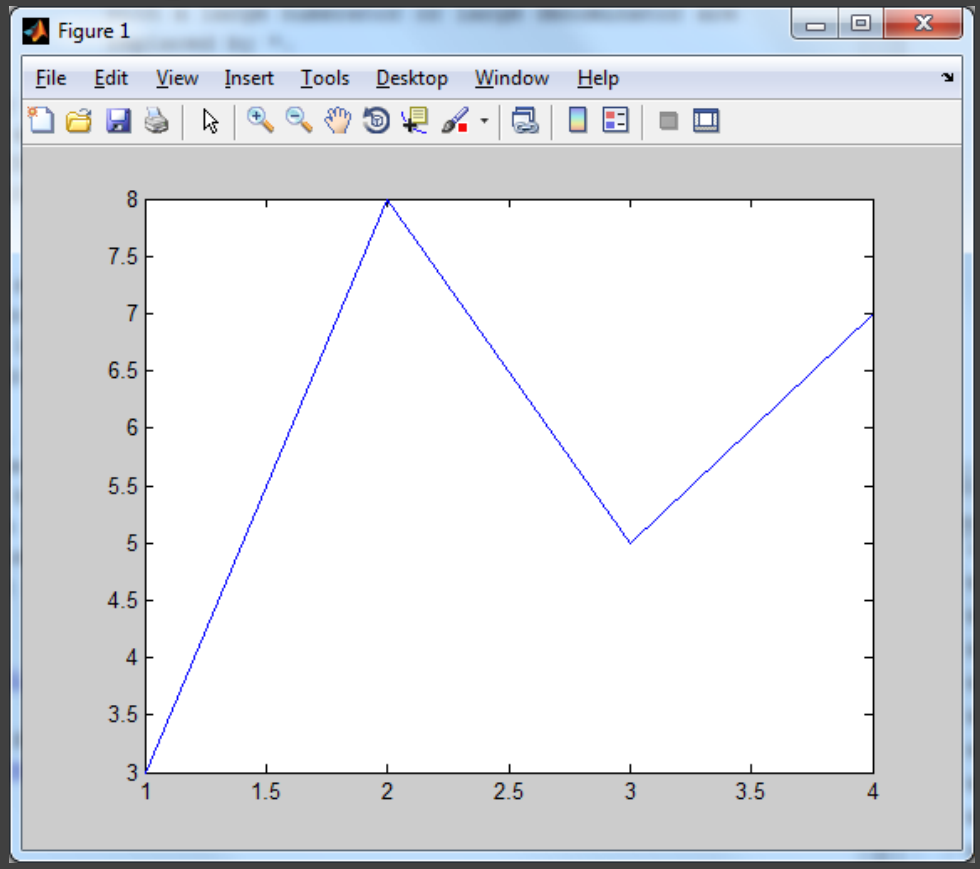
`plot()`

- vytváří dvou-dimenzionální grafy.
- mnoho různých kombinací vstupních argumentů.
- nejjednodušší formou je `plot(y)`, `plot(x,y)`

Např.:

```
y=[3,8,5,7];
plot(y)
```

- pokud je vstupním argumentem vektor y, souřadnice x je vektor, který začíná 1

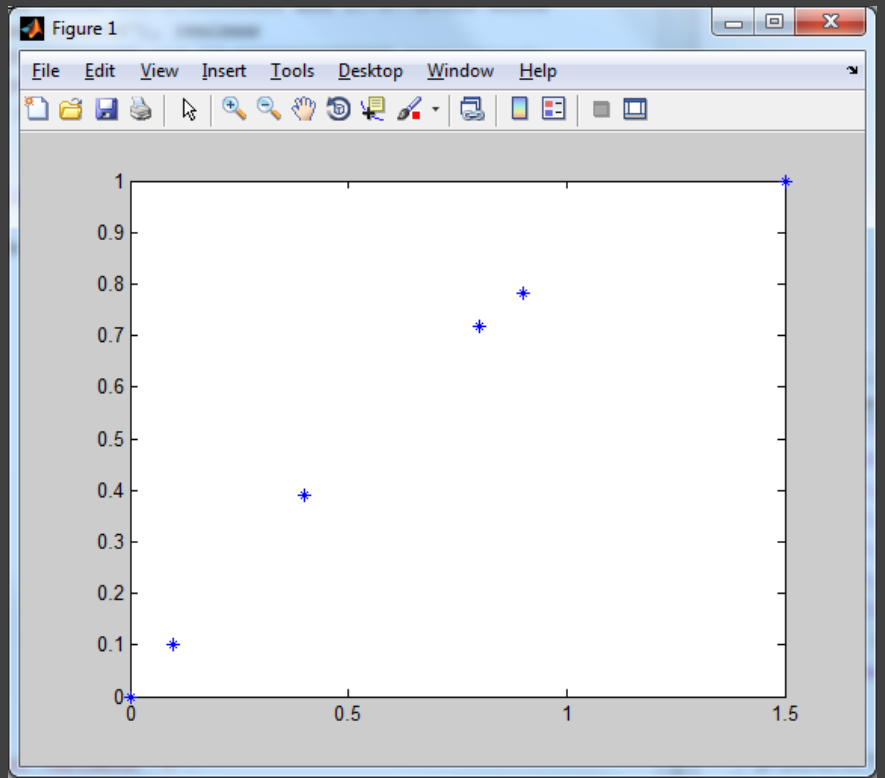


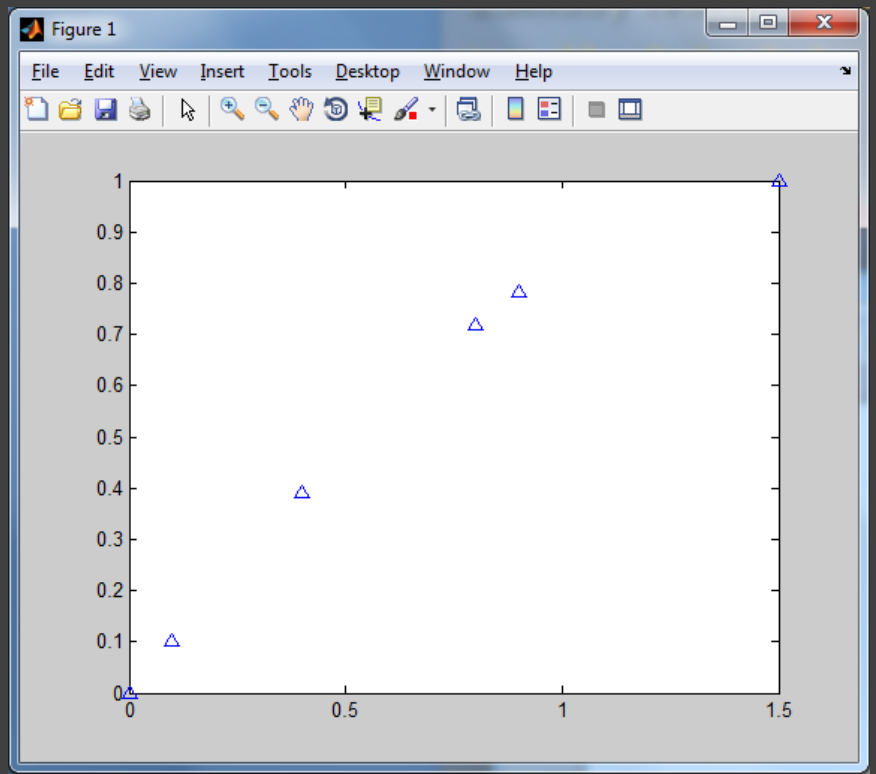
Základy práce s výpočetními systémy

Základy tvorby 2D grafů

```
t = [0, 0.1, 0.4, 0.8, 0.9, 1.5];
y = sin(x);
plot(t,y,'*')
```

- body zobrazeny jako hvězdičky, pro zobrazení bodu lze použít `*`, `o`, `x`, `^` atp.

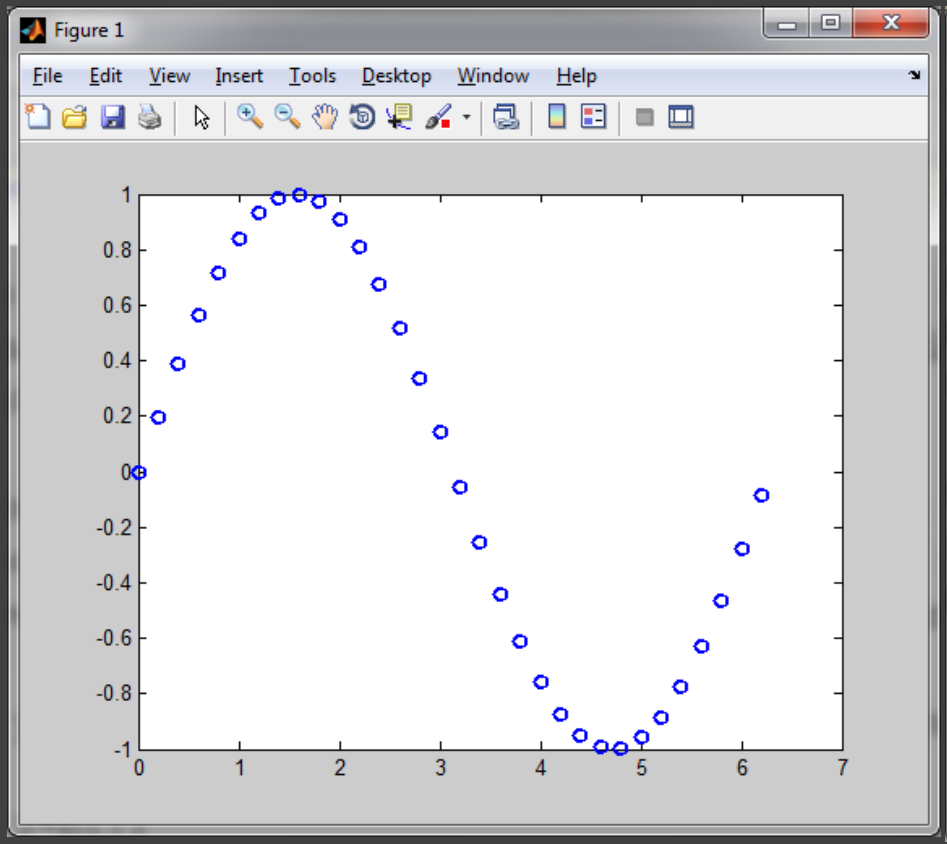




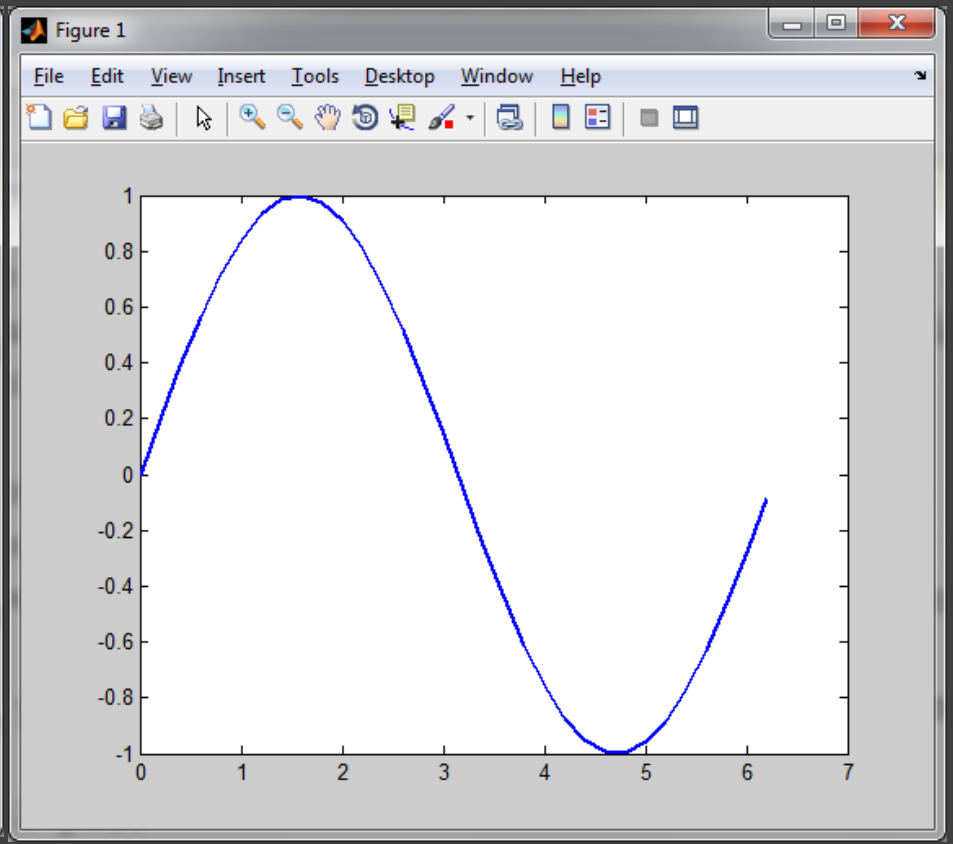
Základy práce s výpočetními systémy

Základy tvorby 2D grafů

```
t = [0:0.2:2*pi];
y = sin(t);
plot(t,y,'o')
```



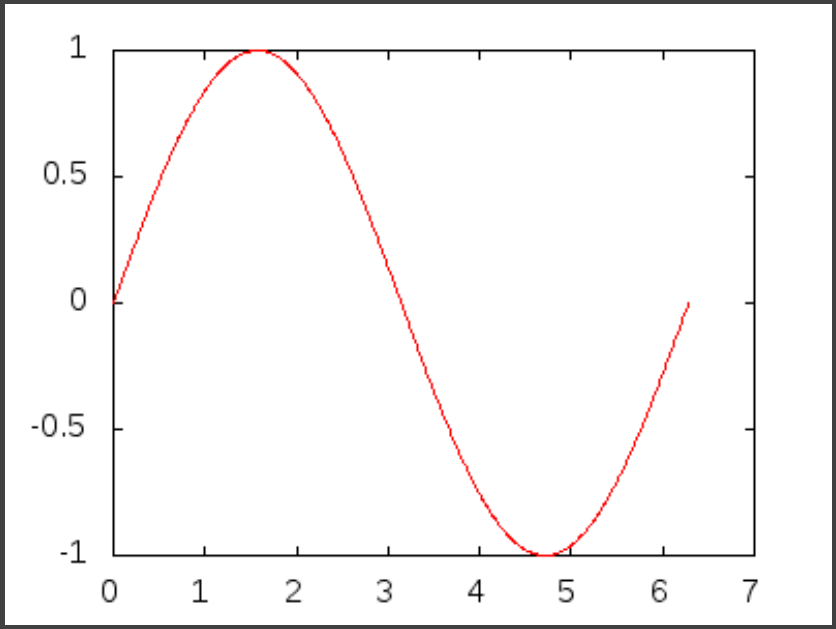

```
t = [0:0.2:2*pi];
y = sin(t);
plot(t,y)
```



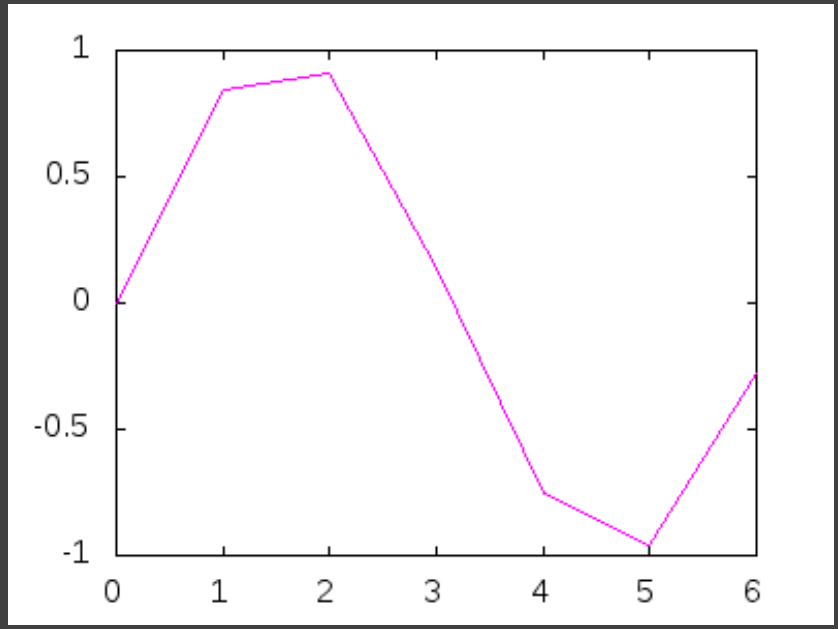
Základy práce s výpočetními systémy

Základy tvorby 2D grafů

```
t = [0:0.01:2*pi];
y = sin(t);
plot(t,y,'r')
```




```
t = [0:1:2*pi];
y = sin(t);
plot(t,y,'m')
```



Operátory

`=` přiřazení

Operátory pro operace prvek po prvku
(operace s maticemi **prvek po prvku**)

`+` sčítání
`-` odečítání
`.*` násobení prvek po prvku
`./` dělení prvek po prvku
`.\` dělení zleva prvek po prvku
`.^` mocnina prvek po prvku
`.'` překlopení matice (pole) podle hlavní diagonály (např. `A.'`)

Operátory pro maticové operace (operace **s celými maticemi**)

`*` násobení maticové
`/` dělení maticové
`\` dělení zleva maticové
`^` mocnina maticová
`'` transpozice matice (např. `A'`) – pozor z komplexních čísel v matici budou čísla komplexně sdružená (z `2+3i` bude `2-3i`)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`+` sčítání

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

C = A + B
C =
     4    14     4     9
    10    10     3    17
```

(Stejný výsledek dostaneme i pomocí příkazu: `C = B + A`
Sčítání matic je komutativní.)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`-` odečítání

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

D = A - B
D =
    0    4    2    3
    4   -8    1    1
```

(Stejný výsledek dostaneme i pomocí příkazu `D = -B + A`)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`.*` násobení prvek po prvku

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

E = A .* B
E =
     4    45     3    18
    21     9     2    72
```

(Stejný výsledek dostaneme i pomocí příkazu: `E = B .* A`
Násobení matic prvek po prvku je komutativní.)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`./` dělení prvek po prvku

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

F = A ./ B
F =
    1.00000    1.80000    3.00000    2.00000
    2.33333    0.11111    2.00000    1.12500
```

Výsledek ve formátu zlomku

```
format rat
A ./ B
ans =
    1      9/5    3      2
    7/3    1/9    2      9/8
```

(Stejný výsledek dostaneme i pomocí příkazu `B.\A`)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`.\` dělení zleva prvek po prvku

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

G = A .\ B
G =
    1.00000    0.55556    0.33333    0.50000
    0.42857    9.00000    0.50000    0.88889
```

Výsledek ve formátu zlomku

```
format rat
A .\ B
ans =
 1      5/9    1/3    1/2
 3/7    9      1/2    8/9
```

(Stejný výsledek dostaneme i pomocí příkazu `B./A`)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`.^` mocnina prvek po prvku

```
A =
    2    9    3    6
    7    1    2    9

H = A.^2
H =
    4   81    9   36
   49    1    4   81
```

(Stejný výsledek dostaneme i pomocí příkazu: `A.*A`)

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`.'` překlopení dvourozměrného pole úhlopříčně podle hlavní diagonály

```
A =
    2    9    3    6
    7    1    2    9

K = A.'
K =
    2    7
    9    1
    3    2
    6    9
```

Operátory

Operátory pro operace prvek po prvku

(operace s maticemi prvek po prvku)

`.'` překlopení dvourozměrného pole úhlopříčně podle hlavní diagonály

```
A =
    2    9    3    6
    7    1    2    9

K = A.'
K =
    2    7
    9    1
    3    2
    6    9
```

Operátory

Operátory pro operace prvek po prvku
(operace s maticemi prvek po prvku)

Rozměry matic, se kterými probíhají operace prvek po prvku, musí být stejné, výsledek má také stejnou velikost, např.

```
a = [1,2];          1 řádek, 2 sloupce
b = [3,4];          1 řádek, 2 sloupce

a .* b        (.*   násobení prvek po prvku)
ans =                                   ans(1)=1*3=6
     3    8         1 řádek, 2 sloupce  ans(2)=2*4=8
```

Pozn.

```
a * b         (*    násobení maticové)
```

Nelze: rozměry matic nesouhlasí.

Operace s vektory a maticemi

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

```
a + 5
ans =
    6     7     8     9    10
```

– sčítání prvek po prvku

Stejný výsledek dostaneme i pomocí : `5 + a`

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
a.'+ 5
ans =
     6
     7
     8
     9
    10
```

– sčítání prvek po prvku

Stejný výsledek dostaneme i pomocí : `5 + a.'`

Rozměry vektoru, se kterým probíhají operace prvek po prvku, jsou stejné jako výsledek.

Operace s vektory a maticemi

```
a = [1,2,3,4,5];
b = [6,7,8,9,10];
```

```
a + b
ans =
    7     9    11    13    15
```

– sčítání prvek po prvku

Stejný výsledek dostaneme i pomocí : `b + a`

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

```
a.'+ b.'
ans =
     5
     6
     7
     8
     9
```

– sčítání prvek po prvku

Stejný výsledek dostaneme i pomocí : `b.' + a.'`

Rozměry vektorů, se kterými probíhají operace prvek po prvku, jsou stejné, výsledek má také stejnou velikost

Operace s vektory a maticemi

```
a = [1,2,3,4,5]; b = [6,7,8,9,10];
```

```
a + b.'     – sčítání prvek po prvku
% verze R2015a
Error using  +
Matrix dimensions must agree.
```

```
a + b.'   – sčítání prvek po prvku
% verze R2017b
ans =
     7     8     9    10    11
     8     9    10    11    12
     9    10    11    12    13
    10    11    12    13    14
    11    12    13    14    15
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

Stejný výsledek dostaneme i pomocí :

```
b.' + a
```

```
% všechny verze
[b(1)+a(1,:);b(2)+a(1,:);b(3)+a(1,:);b(4)+a(1,:);b(5)+a(1,:)]
```

Operace s vektory a maticemi

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

```
b.'
ans =
     6
     7
     8
     9
    10
```

`a + b.'` – sčítání **prvek po prvku**

verze R2017b

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

Stejný výsledek dostaneme i pomocí :

`b.' + a`

```
% všechny verze
[b(1)+a(1,:);b(2)+a(1,:);b(3)+a(1,:);b(4)+a(1,:);b(5)+a(1,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[a(1,:)+b(1);a(1,:)+b(2);a(1,:)+b(3);a(1,:)+b(4);a(1,:)+b(5)]
```

Operace s vektory a maticemi

```
a = [1,2,3,4,5]; b = [6,7,8,9,10];
```

```
a.' + b    – sčítání prvek po prvku
% verze R2015a
Error using  +
Matrix dimensions must agree.
```

```
a.' + b   – sčítání prvek po prvku
% verze R2017b
ans =
     7     8     9    10    11
     8     9    10    11    12
     9    10    11    12    13
    10    11    12    13    14
    11    12    13    14    15
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

Stejný výsledek dostaneme i pomocí :

```
b + a.'
```

```
% všechny verze
[a(1)+b(1,:);a(2)+b(1,:);a(3)+b(1,:);a(4)+b(1,:);a(5)+b(1,:)]
```

Operace s vektory a maticemi

```
b = [6,7,8,9,10];
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
a.' + b     – sčítání prvek po prvku
% verze R2017b
ans =
     6     7     8     9    10
 1   7     8     9    10    11
 2   8     9    10    11    12
 3   9    10    11    12    13
 4  10    11    12    13    14
 5  11    12    13    14    15
```

Stejný výsledek dostaneme i pomocí :

```
b.' + a
```

```
% všechny verze
[a(1)+b(1,:);a(2)+b(1,:);a(3)+b(1,:);a(4)+b(1,:);a(5)+b(1,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[b(1,:)+a(1);b(1,:)+a(2);b(1,:)+a(3);b(1,:)+a(4);b(1,:)+a(5)]
```

Operace s vektory a maticemi

```
M = [1,2,3;4,5,6]; n = [7;8];
```

```
M + n      – sčítání prvek po prvku
% verze R2015a
Error using  +
Matrix dimensions must agree.
```

```
M + n    – sčítání prvek po prvku
% verze R2017b
ans =
     8     9    10
    12    13    14
```

```
M =
     1     2     3
     4     5     6

n =
     7
     8
```

Stejný výsledek dostaneme i pomocí :

```
n + M
```

```
% všechny verze
[n(1)+M(1,:);n(2)+M(2,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[M(1,:)+n(1);M(2,:)+n(2)]
```

Operace s vektory a maticemi

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

```
a .* 5
ans =
     5    10    15    20    25
```

– násobení prvek po prvku

Stejný výsledek dostaneme i pomocí : `5 + a`

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
a.'.* 5
ans =
     5
    10
    15
    20
    25
```

– násobení prvek po prvku

Stejný výsledek dostaneme i pomocí : `5 + a.'`

Rozměry vektoru, se kterým probíhají operace prvek po prvku, jsou stejné jako výsledek.

Operace s vektory a maticemi

```
a = [1,2,3,4,5];
b = [6,7,8,9,10];
```

```
a .* b
ans =
    6    14    24    36    50
```

– násobení prvek po prvku

Stejný výsledek dostaneme i pomocí : `b .* a`

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

```
a.'.* b.'
ans =
     6
    14
    24
    36
    50
```

– násobení prvek po prvku

Stejný výsledek dostaneme i pomocí : `b.' .* a.'`

Rozměry matic, se kterými probíhají operace prvek po prvku, jsou stejné, výsledek má také stejnou velikost

Operace s vektory a maticemi

```
a = [1,2,3,4,5];  b = [6,7,8,9,10];
```

```
a .* b.'   – násobení prvek po prvku
% verze R2015a
Error using  .*
Matrix dimensions must agree.
```

```
a .* b.'   – násobení prvek po prvku
% verze R2017b
ans =
     6    12    18    24    30
     7    14    21    28    35
     8    16    24    32    40
     9    18    27    36    45
    10    20    30    40    50
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

Stejný výsledek dostaneme i pomocí :

```
b.' .* a
```

```
% všechny verze
[a(1).*b(1,:);a(2).*b(1,:);a(3).*b(1,:);a(4).*b(1,:);a(5).*b(1,:)]
```

Operace s vektory a maticemi

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

```
b.'
ans =
     6
     7
     8
     9
    10
```

```
a  .* b.'   – násobení prvek po prvku
verze R2017b
ans =
     1     2     3     4     5
 6   6    12    18    24    30
 7   7    14    21    28    35
 8   8    16    24    32    40
 9   9    18    27    36    45
10  10    20    30    40    50
```

Stejný výsledek dostaneme i pomocí :

```
b.'  .* a
```

```
% všechny verze
[b(1).*a(1,:);b(2).*a(1,:);b(3).*a(1,:);b(4).*a(1,:);b(5).*a(1,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[a(1,:).*b(1);a(1,:).*b(2);a(1,:).*b(3);a(1,:).*b(4);a(1,:).*b(5)]
```

Operace s vektory a maticemi

```
a = [1,2,3,4,5];  b = [6,7,8,9,10];
```

```
a.' .* b   – násobení prvek po prvku
% verze R2015a
Error using  .*
Matrix dimensions must agree.
```

```
a.' .* b   – násobení prvek po prvku
% verze R2017b
ans =
     6     7     8     9    10
    12    14    16    18    20
    18    21    24    27    30
    24    28    32    36    40
    30    35    40    45    50
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
b.'
ans =
     6
     7
     8
     9
    10
```

Stejný výsledek dostaneme i pomocí :

```
b .* a.'
```

```
% všechny verze
[a(1).*b(1,:);a(2).*b(1,:);a(3).*b(1,:);a(4).*b(1,:);a(5).*b(1,:)]
```

Operace s vektory a maticemi

```
b = [6,7,8,9,10];
```

```
a.'
ans =
     1
     2
     3
     4
     5
```

```
a.' .* b   – násobení prvek po prvku
% verze R2017b
ans =
     6     7     8     9    10
 1   6     7     8     9    10
 2  12    14    16    18    20
 3  18    21    24    27    30
 4  24    28    32    36    40
 5  30    35    40    45    50
```

Stejný výsledek dostaneme i pomocí :

```
b.' .* a
```

```
% všechny verze
[a(1).*b(1,:);a(2).*b(1,:);a(3).*b(1,:);a(4).*b(1,:);a(5).*b(1,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[b(1,:).*a(1);b(1,:).*a(2);b(1,:).*a(3);b(1,:).*a(4);b(1,:).*a(5)]
```

Operace s vektory a maticemi

```
M = [1,2,3;4,5,6]; n = [7;8];
```

```
M .* n      – násobení prvek po prvku
% verze R2015a
Error using  .*
Matrix dimensions must agree.
```

```
M .* n    – násobení prvek po prvku
% verze R2017b
ans =
     7    14    21
    32    40    48
```

```
M =
     1     2     3
     4     5     6

n =
     7
     8
```

Stejný výsledek dostaneme i pomocí :

```
n .* M
```

```
% všechny verze
[n(1).*M(1,:);n(2).*M(2,:)]
```

Stejný výsledek dostaneme i pomocí :

```
[M(1,:).*n(1);M(2,:).*n(2)]
```

Operace s vektory a maticemi

```
M = [1,2,3;4,5,6]; n = [7;8];
```

```
M - n      – odečítání prvek po prvku
% verze R2015a
Error using  -
Matrix dimensions must agree.
```

```
M - n    – odečítání prvek po prvku
% verze R2017b
ans =
    -6    -5    -4
    -4    -3    -2
```

```
% všechny verze
[M(1,:)- n(1);M(2,:)- n(2)]
```

Stejný výsledek dostaneme i pomocí :

```
[-n(1) + M(1,:);-n(2) + M(2,:)]
```

```
M =
     1     2     3
     4     5     6

n =
     7
     8
```

Stejný výsledek dostaneme i pomocí :

```
-n + M
```

Podobné změny nastaly u operací `./` a `.\`

Operátory

Operátory pro maticové operace (operace s celými maticemi)

***** násobení maticové

- Aby matice bylo možno násobit, musí být počet sloupců první matice stejný jako počet řádků druhé matice.
- Výsledek má počet řádek jako první matice a počet sloupců jako druhá matice.

Např. máme-li:

C = A * B

(*m*,*p*) = (*m*,*n*) * (*n*,*p*)

m,*n*,*p* – počty řádků a sloupců příslušných matic **A**, **B**, **C**

- musí mít matice **A** tolik sloupců (*n* sloupců), jako má matice **B** řádků (*n* řádků), jinak nelze násobit maticově,
- výsledná matice **C** má tolik řádků, jako měla **A** (*m* řádků) a má tolik sloupců jako měla **B** (*p* sloupců).

Operátory

Operátory pro maticové operace (operace s celými maticemi)

`*` násobení maticové

`C = A * B`
(*m*,*p*) = (*m*,*n*) * (*n*,*p*)
m,*n*,*p* – počty řádků a sloupců příslušných matic `A`, `B`, `C`

`D = B * A`
(???) = (*n*,*p*) * (*m*,*n*)

- nelze násobit maticově, $m \neq p$

⇒ `A * B ≠ B * A`
(Maticové násobení není komutativní !!!)

Operátory

Operátory pro maticové operace (operace s celými maticemi)

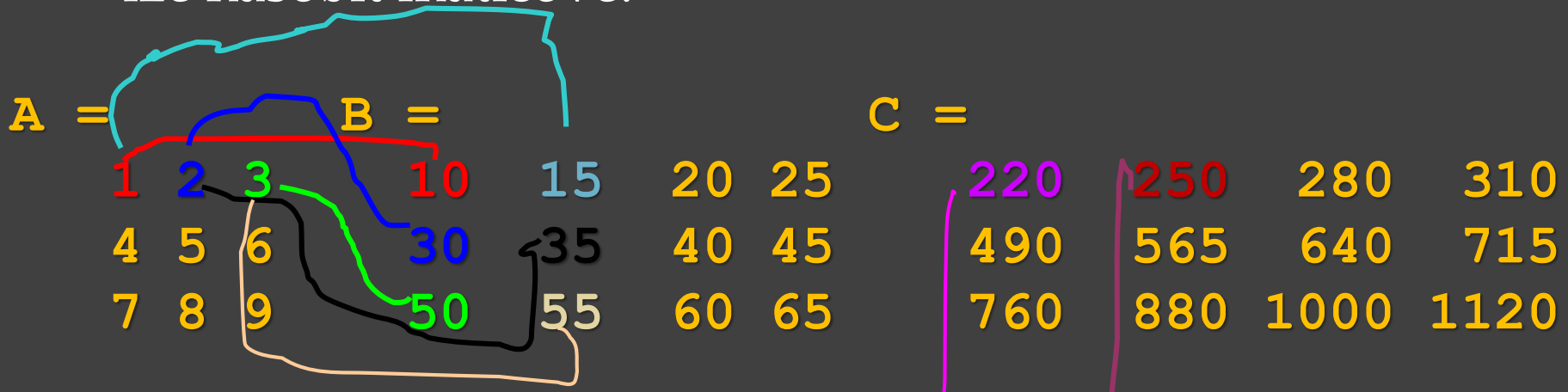
***** násobení maticové

Příklad:

C = A * B

(3,4) = (3,3) * (3,4)

- matice **A** má 3 sloupce, tj. stejný počet jako má matice **B** řádků
  => lze násobit maticově:



Operátory

Operátory pro maticové operace (operace s celými maticemi)

atd. - první řádek z A se všemi sloupci B.

```
1 * 20 + 2 * 40 + 3 * 60 = 280
1 * 25 + 2 * 45 + 3 * 65 = 310
```

A potom totéž pro druhý řádek A, opět se všemi sloupci B.

```
4 * 10 + 5 * 30 + 6 * 50 = 490
4 * 15 + 5 * 35 + 6 * 55 = 565
4 * 20 + 5 * 40 + 6 * 60 = 640
4 * 25 + 5 * 45 + 6 * 65 = 715
```

A dále totéž i pro třetí řádek A, opět se všemi sloupci B

```
7 * 10 + 8 * 30 + 9 * 50 = 760
7 * 15 + 8 * 35 + 9 * 55 = 880
7 * 20 + 8 * 40 + 9 * 60 = 1000
7 * 25 + 8 * 45 + 9 * 65 = 1120
```

… až dostaneme výslednou matici C

Operátory

Operátory pro maticové operace (operace s celými maticemi)

`*` násobení maticové

```
K =                    L =
     1   2                  5   6
     3   4                  7   8
```

Příklad:

`M = K * L` (2,2) = (2,2) * (2,2)

`N = L * K` (2,2) = (2,2) * (2,2) – první matice má 2 sloupce, tj. stejný počet jako má druhá matice řádků => v obou případech lze násobit maticově:

```
M =                    N =
    19  22                 23  34
    43  50                 31  46
```

Pro čtvercové matice `K`, `L` platí rovněž `K * L ≠ L * K`, pro `K * L` bude jiný výsledek než pro `L * K`.

Maticové násobení není komutativní!!!