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

Cyklus for

for
cyklus s udaným (pevně daným) počtem opakování
(víme kolikrát má proběhnout)

```
for cykl = start:krok:konec
     příkazy;
     příkazy;
end
% a další pokračování programu…
```

- **příkazy** v těle cyklu jsou opakovány tolikrát, kolik prvků obsahuje vektor **[start:krok:konec]**
- řídící proměnná cyklu **cykl** se postupně mění podle nastaveného **kroku** od **startu** až po **konec**, lze jí využívat v příkazech

*Pozn.: vyhýbejte proměnným **i** a **j** pro řídící proměnnou cyklu (v MATLABu vyhrazeno pro imaginární jednotky)*

Cyklus for

Lze použít:

```
for cykl = start:konec % automaticky krok 1
      příkazy;
end
```

Pozn.: zpětný chod – **záporný** krok (-1, -5 atd.) je běžný, samozřejmě v tomto případě musí být **start** **větší než** **konec**

```
for cykl = 10:-2:3
      disp(cykl)
end
```

```
      10
      8
      6
      4
```

Chybně napsaný cyklus se neprovede ani jednou:

```
for cykl = 10:2:3
      příkazy;
end
```

Nevypíše se vůbec nic.

Cyklus for

Pozn.: pozor cykly jsou pomalejší než zabudované funkce, resp. než funkce nad celými vektory.

Příklad: Vytvoření 5-ti prvkového vektoru *n*-tou odmocninou z čísla 1000 000, kde *n* je index prvku (pořadové číslo) ve vektoru:

$$x = \left[\sqrt[1]{10^6} \quad \sqrt[2]{10^6} \quad \sqrt[3]{10^6} \quad \sqrt[4]{10^6} \quad \sqrt[5]{10^6} \right]$$

```
for n=1:5           % krok 1 není uveden
 x(n)=1e6^(1/n);% naplnění n-tého prvku vektoru
                    % n-tou odmocninou z čísla 10^6
end                 % konec cyklu
format short g
disp(x)             % výpis vektoru x
```

```
    1e+006    1000     100    31.623    15.849
```

Cyklus for

Pokračování příkladu:

$$x = \left[\sqrt[1]{10^6} \quad \sqrt[2]{10^6} \quad \sqrt[3]{10^6} \quad \sqrt[4]{10^6} \quad \sqrt[5]{10^6} \right]$$

```
for n=1:5
 x(n)=1e6^(1/n);
end
```

Cyklus lze nahradit výpočtem pomocí vektorů (efektivnější - výpočet probíhá pomocí operací **prvek po prvku**) => cykly používat jen v nutných případech

```
n = 1:5;            % vektor od 1 do 5
x = 1e6.^(1./n)     % do vektoru x přiřadí první až
                    % pátou odmocninou z čísla 10^6
disp(x)             % výpis vektoru x
```

```
    1e+006    1000     100    31.623    15.849
```

Cyklus for

Příklad:
Součet prvků ve vektoru s 10 miliony prvky pomocí **sum()** a pomocí cyklu **for**.

Využijeme funkce:

rand (m,n) – matice *m* řádky a *n* sloupci obsahující „pseudonáhodné“ hodnoty z otevřeného intervalu (0,1)
Pomocí **rand(1,10000000)** vytvoříme vektor s **1** řádkem a **10000000** sloupci.

length – počet prvků ve vektoru, např.

```
a = [1,7,5,6,9]
length(a)
ans = 5
```

tic – spuštění stopek a ukládání aktuálního času
toc – zastavení stopek, změření a výpis času

tic a **toc** – funkce pracující společně – měří uplynulý čas

Cyklus for

Pokračování příkladu – vytvoření funkce:

```
function testsumace
vektor = rand(1,10000000);
tic          % spuštění stopek
prvni = sum(vektor);
toc          % zastavení stopek a výpis času
disp('Pomoci sum: ');
disp(prvni);
tic          % spuštění stopek
druhy = 0;  % druhy je 0, behem cyklu se zvetsi
for cykl=1:length(vektor)
    druhy = druhy + vektor(cykl);
end;         % konec cyklu
toc          % zastavení stopek a výpis času
disp('Pomoci cyklu for: ');
disp(druhy);
end    % konec cyklu
```

Cyklus for

Pokračování příkladu :

Volání funkce:

```
testsumace
```

Výsledky:

```
Elapsed time is 0.018375 seconds.
Pomoci sum:
       5e+006

Elapsed time is 0.101093 seconds.
Pomoci cyklu for:
       5e+006
```

Cyklus for

Pokračování příkladu :

Volání funkce:
`testsumace`

Výsledky:

```
Elapsed time is 0.018375 seconds.
Pomoci sum:
       5e+006

Elapsed time is 0.101093 seconds.
Pomoci cyklu for:
       5e+006
```

Cyklus for

Pokračování příkladu :

Volání funkce:
`testsumace`

Výsledky:

```
Elapsed time is 0.018375 seconds.
Pomoci sum:
       5e+006

Elapsed time is 0.101093 seconds.
Pomoci cyklu for:
       5e+006
```

Cyklus while

while

- cyklus s podmínkou na začátku
- cyklus bez udání počtu opakování

```
while podmínka  % dokud podmínka platí, běží příkazy
      příkazy;   % stale dokola
      příkazy;
      příkazy;
end;
% další pokračování programu…
```

Cyklus while

while

- cyklus s podmínkou na začátku
- cyklus bez udání počtu opakování

```
while podmínka
      příkazy;
      příkazy;
      příkazy;
end;
% další pokračování programu…
```

podmínka je libovolný logický výraz s logickou hodnotou 0 nebo 1 (většinou se používají relační nebo logické operátory)

příkazy v těle cyklu jsou jakékoli příkazy, které se mají provádět opakovaně, je-li **podmínka** **splněna** (hodnota 1)

Cyklus while

Speciální případy:

```
while(1)
 prikazy;
end
```

```
 while(0)
  prikazy;
 end
```

Cyklus while

Speciální případy:

```
while(1)
 příkazy;
end
```

Pozor: `podmínka` má logickou hodnotu `1`, je tedy stále **pravdivá**, `příkazy` probíhají stále. Jedná se o **nekonečnou** smyčku, lze ji opustit jen pomocí `break`.

```
while(0)
 příkazy;
end
```

Cyklus while

Speciální případy:

```
while(1)
 příkazy;
end
```

Pozor: `podmínka` má logickou hodnotu `1`, je tedy stále **pravdivá**, `příkazy` probíhají stále. Jedná se o **nekonečnou** smyčku, lze ji opustit jen pomocí `break`.

```
while(0)
 příkazy;
end
```

Pozor: `podmínka` má logickou hodnotu `0`, je tedy **nepravdivá**, `příkazy` neproběhnou.

Cyklus while

Příklad: Funkce pro výpočet součtu prvních *n* přirozených čísel
Přirozeným číslem se rozumí kladné celé číslo (1, 2, 3, …).

```
function soucet_n_prirozenych(n)
% n - vstup (pocet prirozenych cisel pro soucet)
s = 0; % soucet prvnich n prirozenych cisel
       % s je zatím 0, behem cyklu se zvetsi
a = 1; % 1. prirozene cislo, behem cyklu se zmeni
while (a <= n)
   s = s+a;     % posupne se pricte 1,2,3,...,n
   a = a+1;     % a se zvetsuje na 2,3,4,...,n+1
end             % konec while
disp(s)
end             % konec funkce
```

Jiný způsob řešení daného problému na
http://fyzmatik.pise.cz/706-gaussuv-soucet-aritmeticke-posloupnosti.html

Cyklus while

Příklad: Funkce pro výpočet součtu prvních *n* přirozených čísel
Přirozeným číslem se rozumí kladné celé číslo (1, 2, 3, …).

```
function soucet_n_prirozenych(n)
% n - vstup (pocet prirozenych cisel pro soucet)
s = 0; % soucet prvnich n prirozenych cisel
       % s je zatím 0, behem cyklu se zvetsi
a = 1; % 1. prirozene cislo, behem cyklu se zmeni
while (a <= n)
   s = s+a;     % posupne se pricte 1,2,3,...,n
   a = a+1;     % a se zvetsuje na 2,3,4,...,n+1
end             % konec while
disp(s)
end             % konec funkce
```

a nabude hodnoty n+1, ale tato hodnota už se nepřičítá

Jiný způsob řešení daného problému na
http://fyzmatik.pise.cz/706-gaussuv-soucet-aritmeticke-posloupnosti.html

Cykly

Pomocné příkazy pro cykly:

break - ukončí běh cyklu a vyskočí z něj za jeho koncový **end**

continue - vyvolá okamžitě další otočku cyklu (**for** nebo **while**)

- obvykle se použije **if** a **podmínka** => **break** nebo **continue**

Např.

```
% a = …; b = …;
while podmínka
      příkazy_1;
      if(a < 5)
            break;
      end;
      if (b > 30)
            continue;
      end;
      příkazy_2;
end;
% zbytek programu;
```

Cykly

Pomocné příkazy pro cykly:

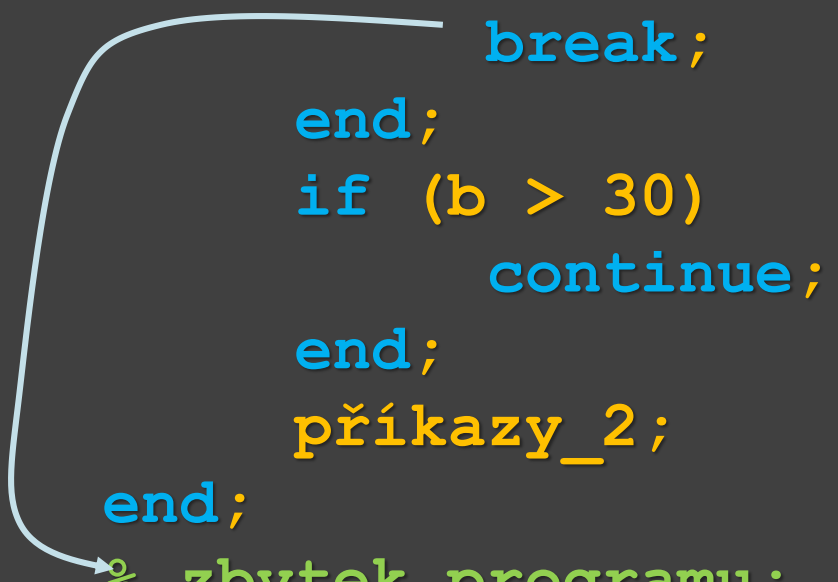
break - ukončí běh cyklu a vyskočí z něj za jeho koncový **end**

continue - vyvolá okamžitě další otočku cyklu (**for** nebo **while**)

- obvykle se použije **if** a **podmínka** => **break** nebo **continue**

Např.

```
% a = …; b = …;
while podmínka
      příkazy_1;
      if(a < 5)
            break;
      end;
      if (b > 30)
            continue;
      end;
      příkazy_2;
end;
% zbytek programu;
```



Cykly

Pomocné příkazy pro cykly:

break - ukončí běh cyklu a vyskočí z něj za jeho koncový **end**

continue - vyvolá okamžitě další otočku cyklu (**for** nebo **while**)

- obvykle se použije **if** a **podmínka** => **break** nebo **continue**

Např.

```
% a = …; b = …;
while podmínka
      příkazy_1;
      if(a < 5)
            break;
      end;
      if (b > 30)
            continue;
      end;
      příkazy_2;
end;
% zbytek programu;
```

Cykly

Pomocné příkazy pro cykly:

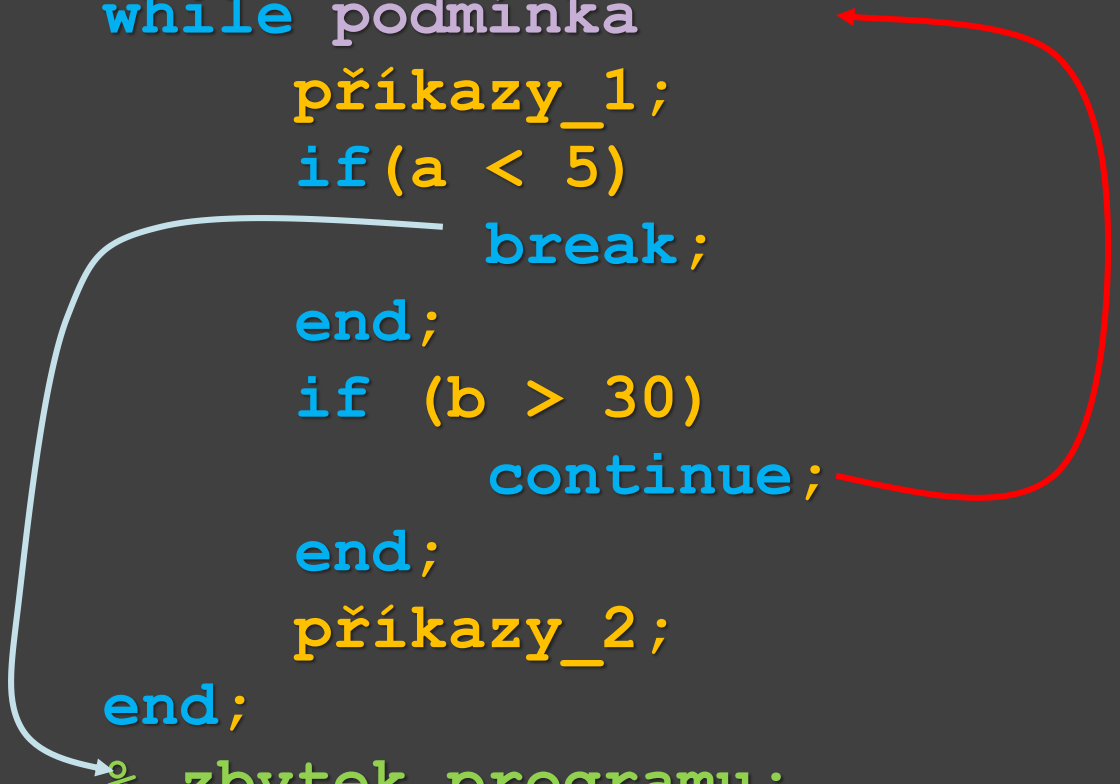
break - ukončí běh cyklu a vyskočí z něj za jeho koncový **end**

continue - vyvolá okamžitě další otočku cyklu (**for** nebo **while**)

- obvykle se použije **if** a **podmínka** => **break** nebo **continue**

Např.

```
% a = …; b = …;
while podmínka
      příkazy_1;
      if(a < 5)
            break;
      end;
      if (b > 30)
            continue;
      end;
      příkazy_2;
end;
% zbytek programu;
```



Řízení běhu výpočtu

Příklad: vstupní parametr *x* se zvětší nebo zmenší na číslo 100.

```
function sto_stop(x)
 while(1)              % nekonečný cyklus
    if (x < 100) % relační operátor „je menší"
          disp(x)
          x=x+1;
    elseif (x > 100)% relační operátor „je vetší"
          disp(x)
          x=x-1;
    else
          disp('Cislo je rovno 100 => konec !!!')
          break % konec běhu cyklu
    end           % konec podmíněného příkazu if
 end              % konec while
end               % konec zápisu funkce
```

Řízení běhu výpočtu

Pokračování příkladu
Volání funkce:

```
sto_stop(87)
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
Cislo je rovno 100 => konec !!!
```

nebo

```
sto_stop(102)
 102
 101
Cislo je rovno 100 => konec !!!
```

nebo

```
sto_stop(100)
Cislo je rovno 100 => konec !!!
```

Základy tvorby 2D grafů

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

`plot(y)` – vykreslí hodnoty vektoru **`y`** v závislosti na jejich indexu (pořadí ve vektoru)

`plot (x, y)` – vykreslí hodnoty vektoru **`y`** v závislosti na hodnotách vektoru **`x`**.

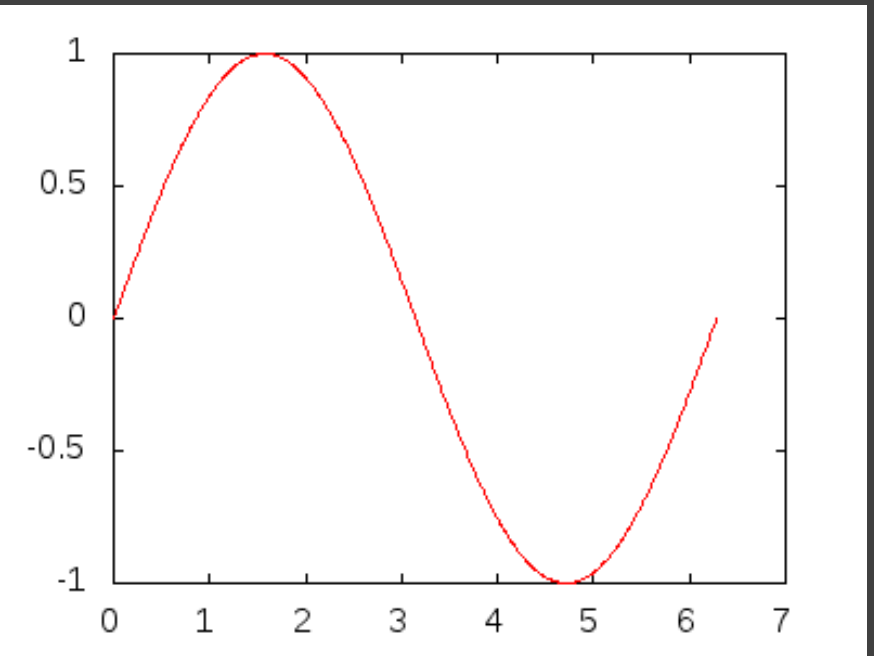
`plot (x, y,'řetězec')` – vykreslí hodnoty vektoru **`y`** v závislosti na hodnotách vektoru **`x`**, **`řetězec`** svým složením příslušných znaků nastaví barvu křivky, příp. značky, typ značky a styl čáry

Základy tvorby 2D grafů

Příklad: vykreslení grafu funkce $y=\sin(t)$
pro t z intervalu od 0 do 2π

Použití „:“, příp. `linspace()` , potom `sin()` a `plot()`

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



- lze nastavit barvu křivky:
`k` (black), `r` (red), `g` (green), `b` (blue), `m` (magenta), `c` (cyan), `w` (white), `y` (yellow)
– řetězec se píše do apostrofů
např.

```
plot(t,y,'m')
```

nebo

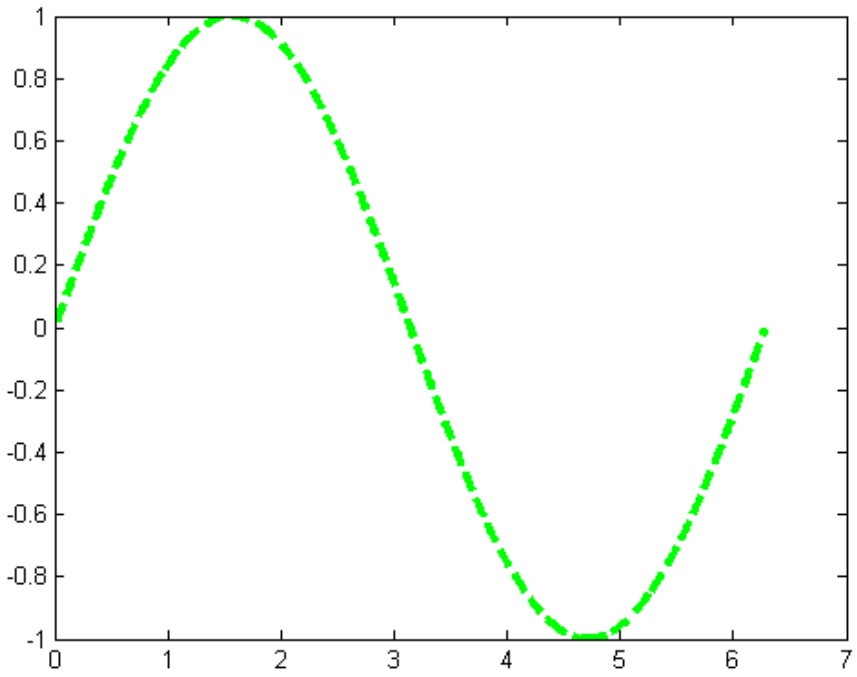
```
plot(t,y,'k')
```

nebo celým názvem

```
plot(t,y,'green')
```

Základy tvorby 2D grafů

```
t = [0:0.01:2*pi];
y = sin(t);
plot(t,y,'--g') % čárkovaná zelená křivka
```



– lze nastavit styl čáry:

`'-'`	plná čára
`'-.'`	čerchovaná čára
`'--'`	čárkovaná
`':'`	tečkovaná

viz **`help plot`**

Např.

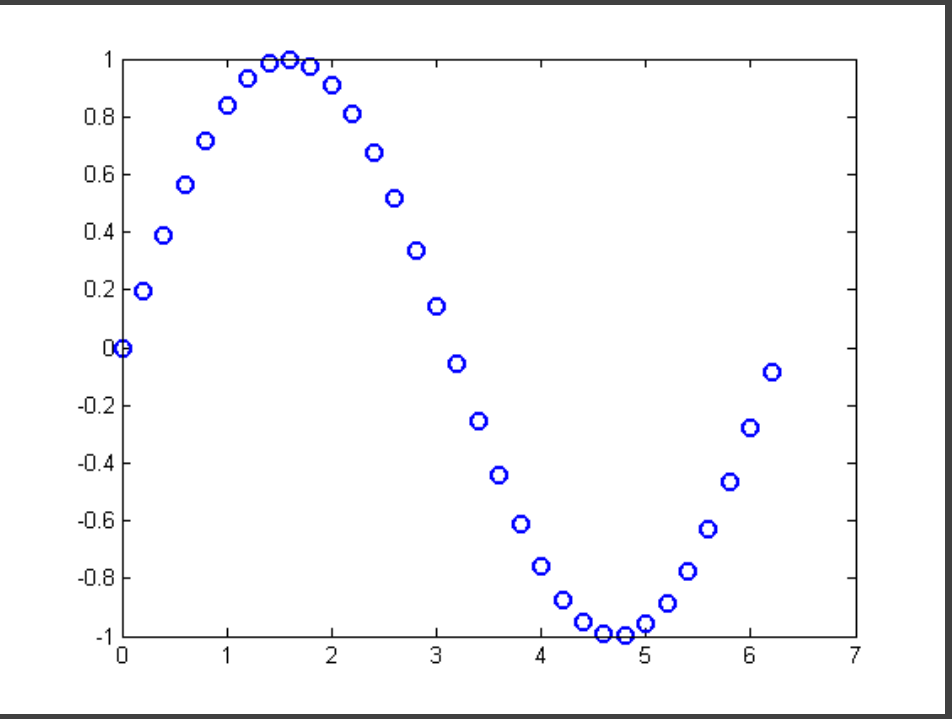
```
plot(t,y,'-.k')
```

(*Pozn.* platí pro MATLAB, v jiných výpočetních systémech jiné možnosti)

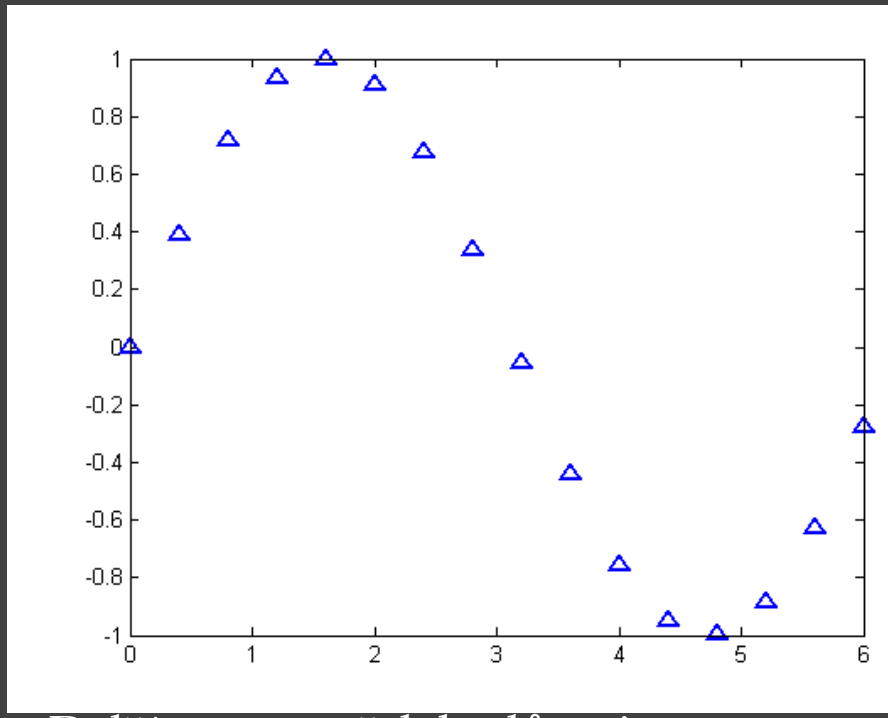
Základy tvorby 2D grafů

– lze nastavit typ bodu: **v** ,**^**, **<**, **>** trojúhelník (různě orientovaný), **o** kolečko, **.** bod, **+** plus, ***** hvězdička, **x** křížek, **s** čtverec, apod.

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




```
t = [0:0.4:2*pi];
y = sin(t);
plot(t,y,'^')
```



Další typy značek bodů - viz **help plot**

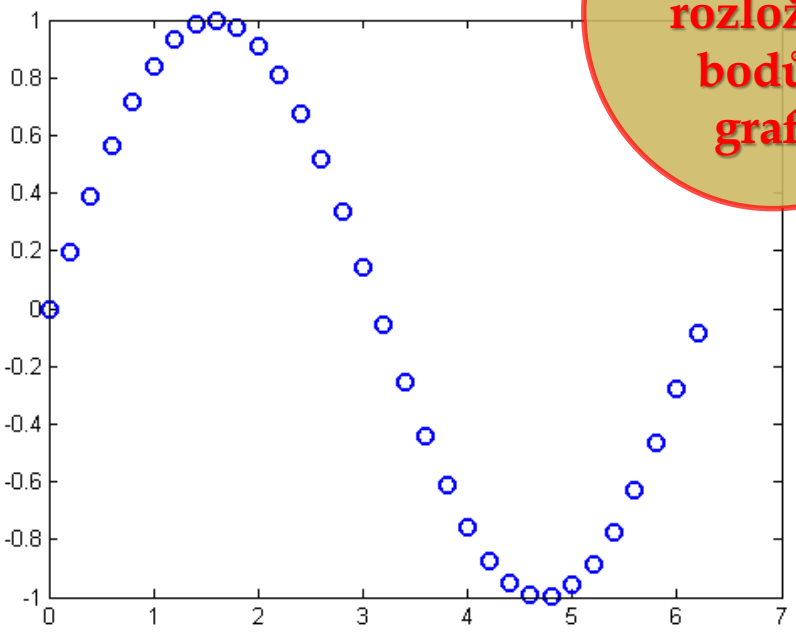
Základy tvorby 2D grafů

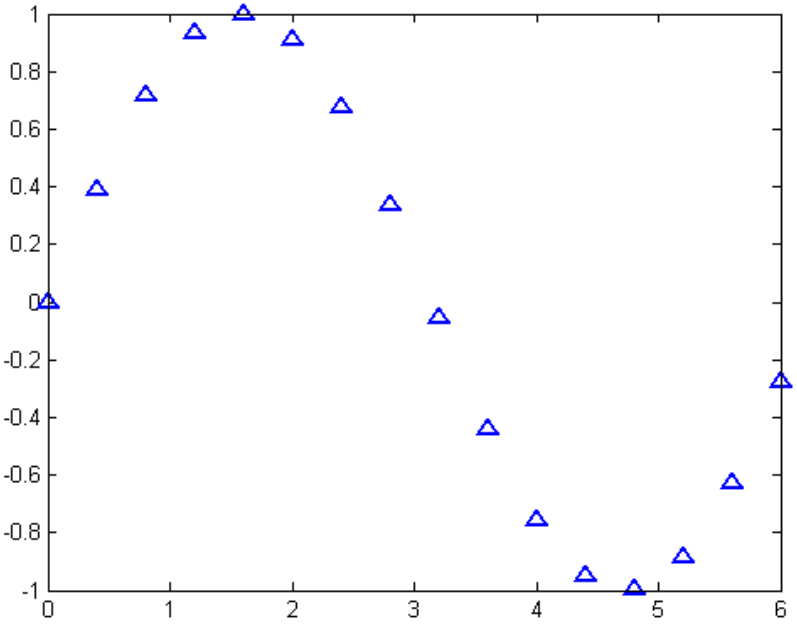
– lze nastavit typ bodu: **v** ,**^**, **<**, **>** trojúhelník (různě orientovaný), **o** kolečko, **.** bod, **+** plus, ***** hvězdička, **x** křížek, **s** čtverec, apod.

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

```
t = [0:0.4:2*pi];
y = sin(t);
plot(t,y,'^')
```

krok => rozložení bodů v grafu



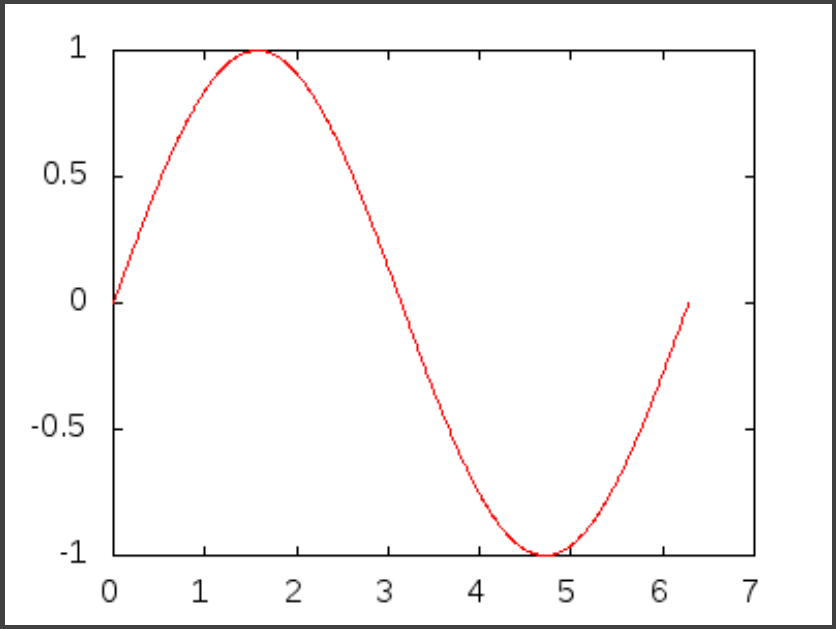


Další typy značek bodů - viz **help plot**

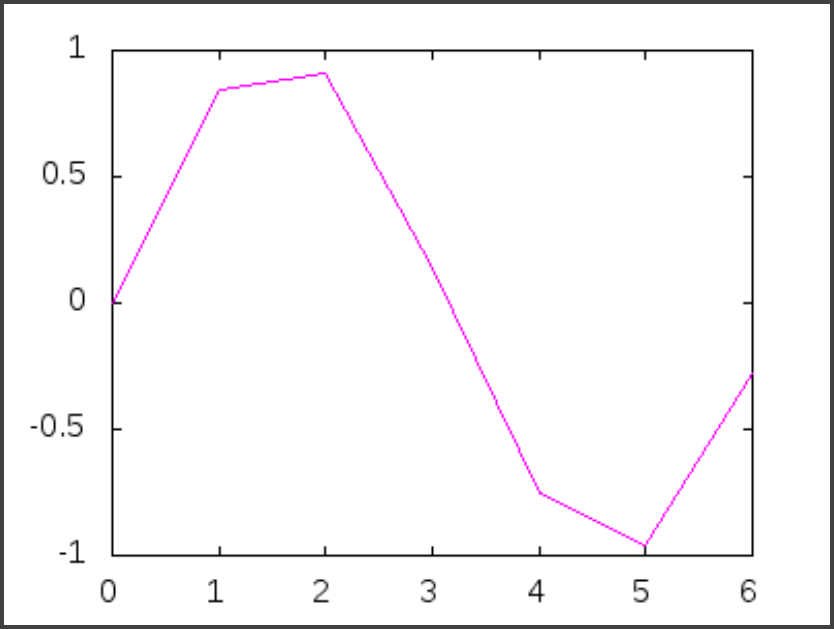
Základy tvorby 2D grafů

– při vykreslování křivky je důležitá velikost kroku, příp. počet prvků, ve vektoru *t* (na ose *x*).

```
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')
```



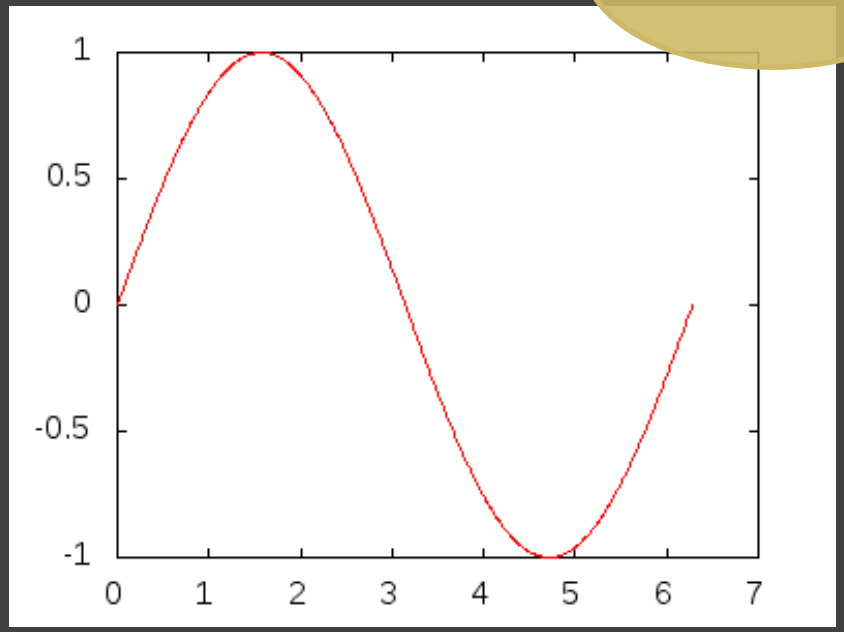
Základy tvorby 2D grafů

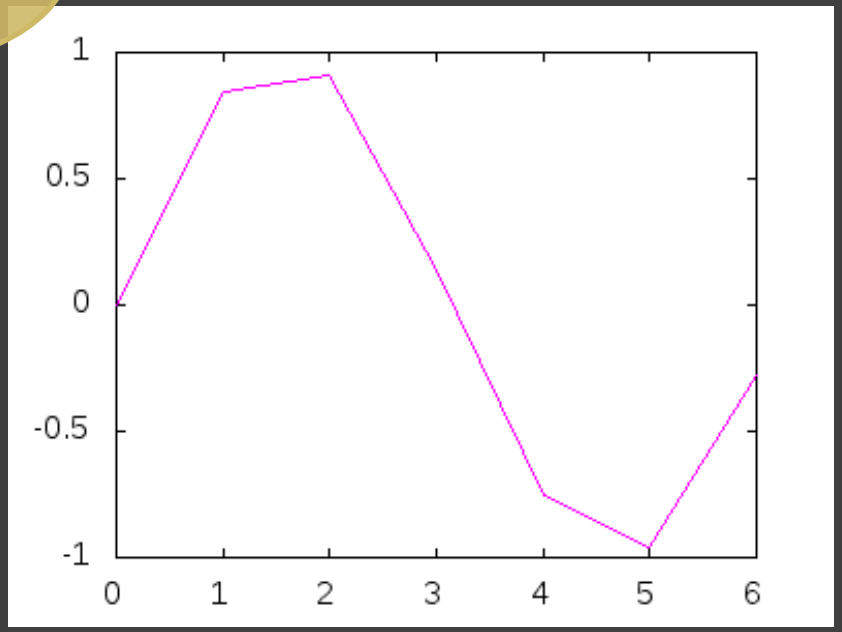
– při vykreslování křivky je důležitá velikost kroku, příp. počet prvků, ve vektoru ***t*** (na ose *x*).

```
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')
```

krok





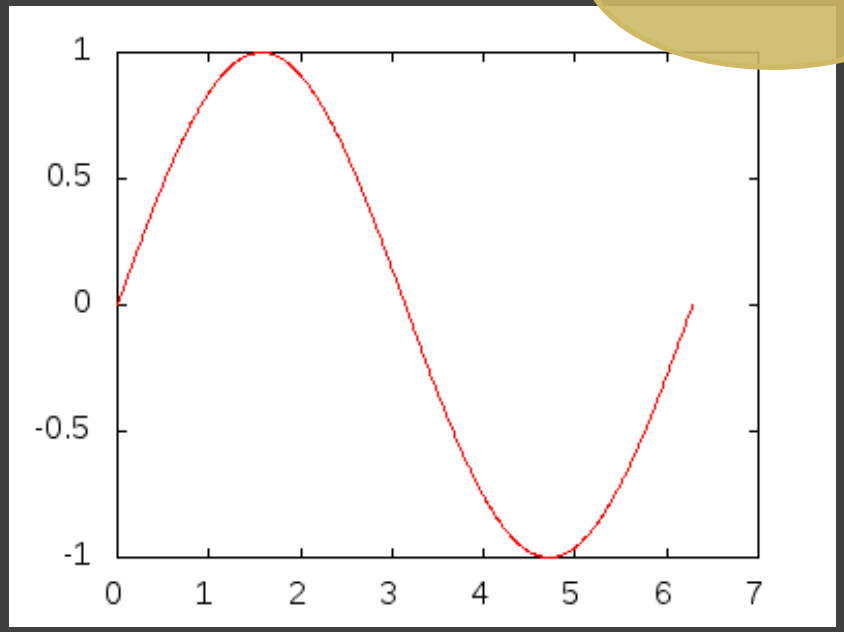
Základy tvorby 2D grafů

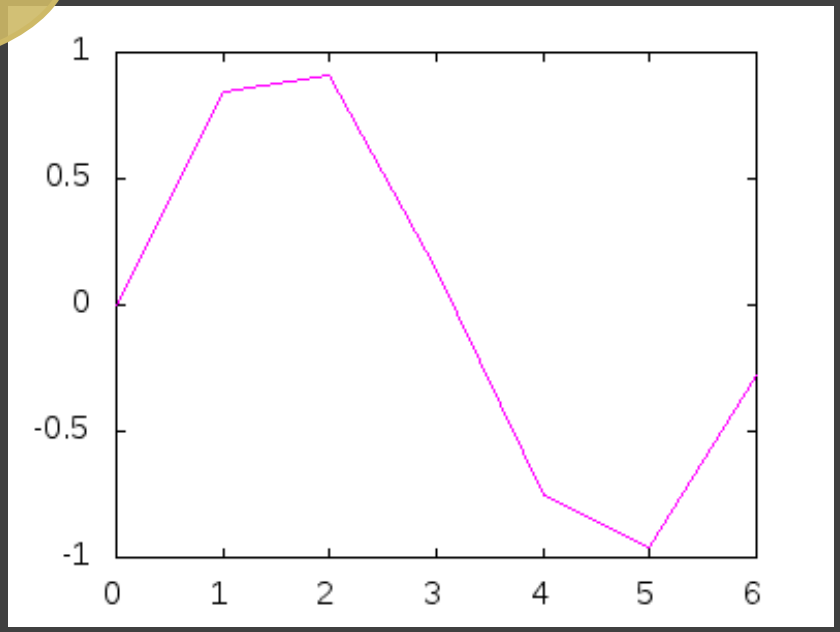
– při vykreslování křivky je důležitá velikost kroku, příp. počet prvků, ve vektoru *t* (na ose *x*).

```
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')
```

krok





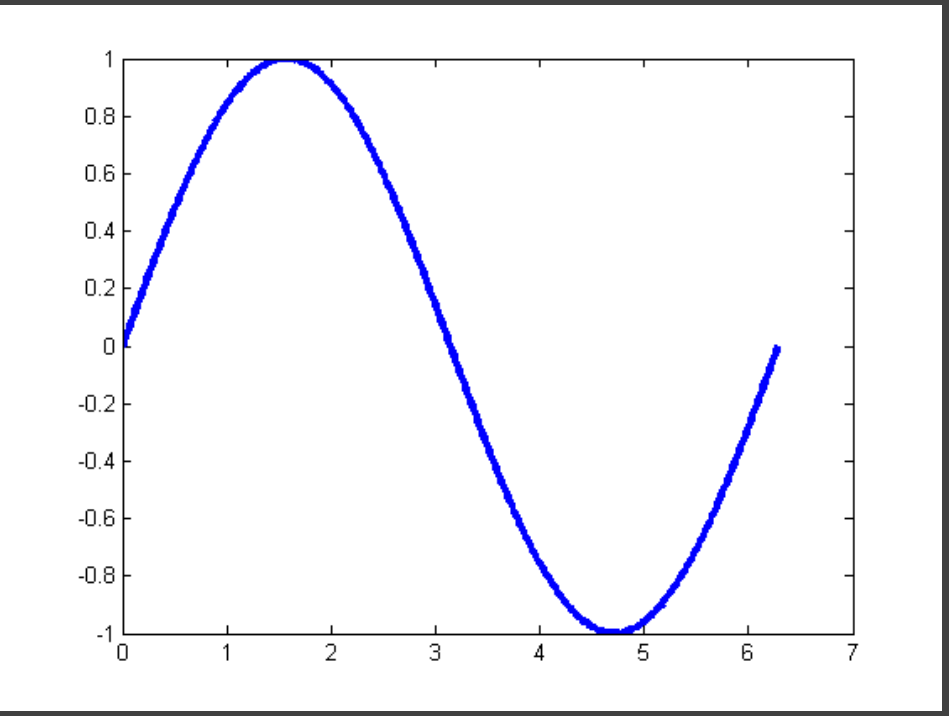

```
length(t)
ans = 629
```

```
length(t)
ans = 7
```

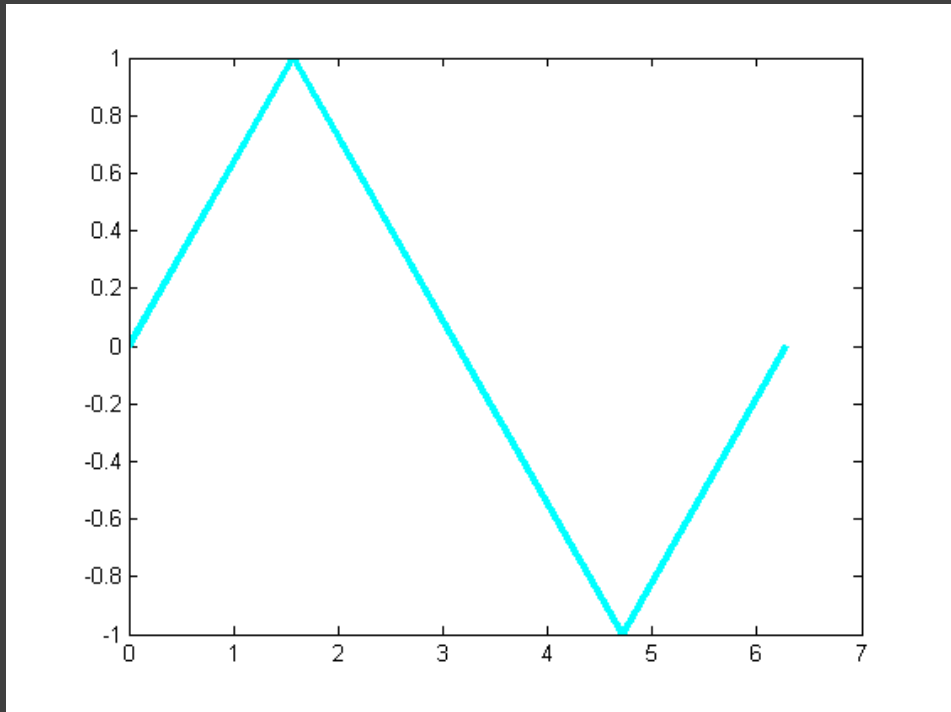
Základy tvorby 2D grafů

– při vykreslování křivky je důležitá velikost kroku, příp. počet prvků, ve vektoru ***t*** (na ose *x*).

```
t = linspace(0,2*pi,500);
y = sin(t);
plot(t,y,'b')
```




```
t = linspace(0,2*pi,5);
y = sin(t);
plot(t,y,'c')
```



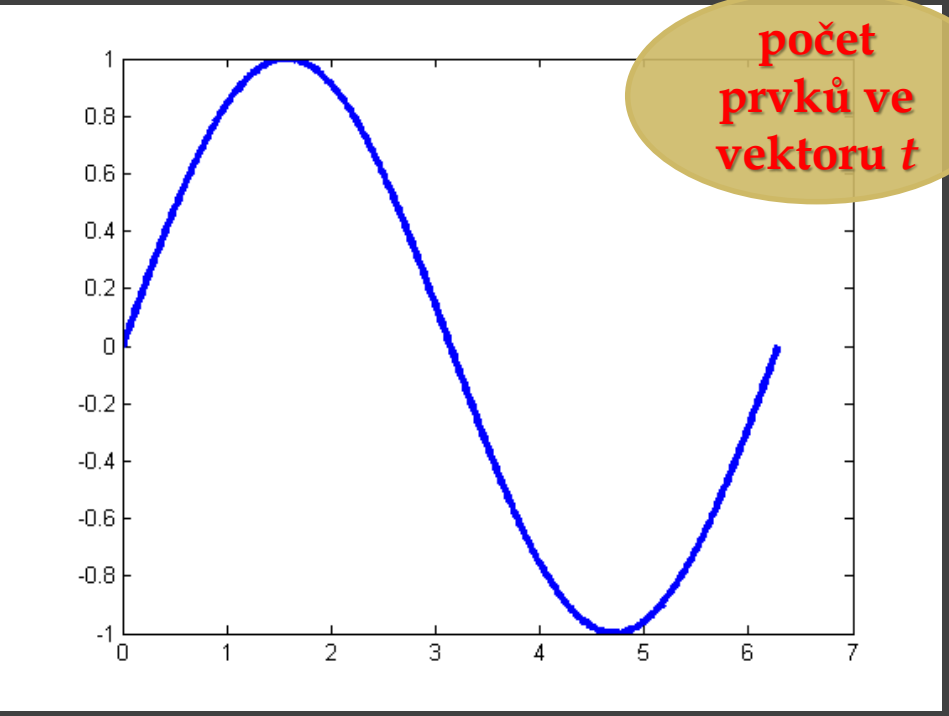
Základy tvorby 2D grafů

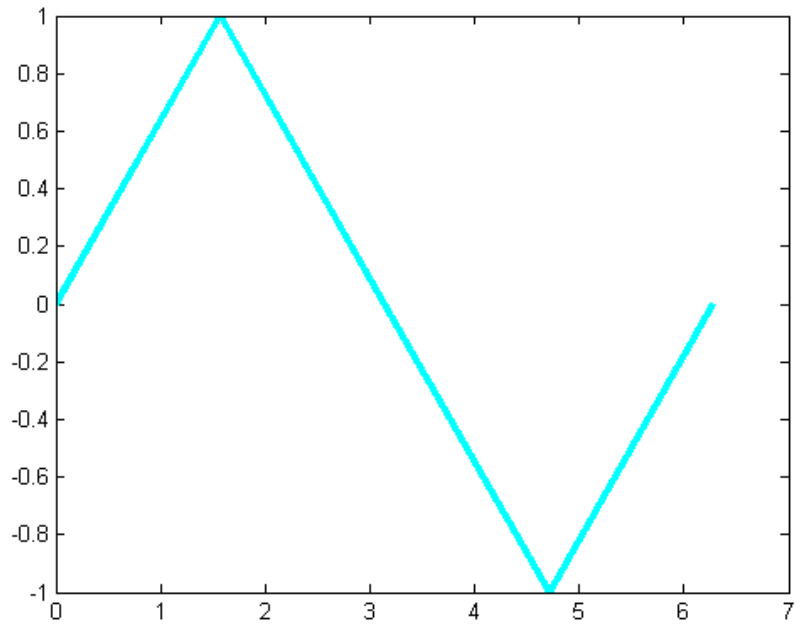
– při vykreslování křivky je důležitá velikost kroku, příp. počet prvků, ve vektoru *t* (na ose *x*).

```
t = linspace(0,2*pi,500);
y = sin(t);
plot(t,y,'b')
```

počet prvků ve vektoru *t*

```
t = linspace(0,2*pi,5);
y = sin(t);
plot(t,y,'c')
```





Základy tvorby 2D grafů

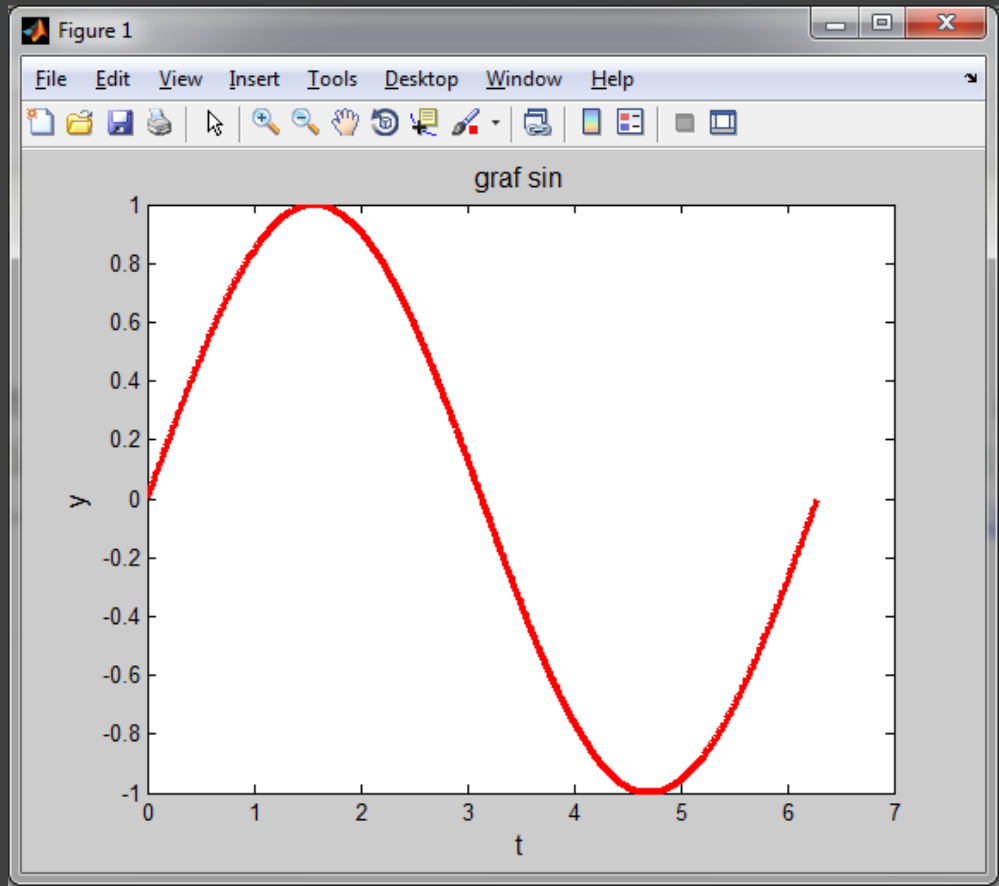
- název grafu – parametr – řetězec v apostrofech

```
title('graf sin')
```

– popis os – parametr
– řetězec v apostrofech

```
xlabel('t')
ylabel('y')
```

```
zlabel('z')
```
– pro 3D grafy



Základy tvorby 2D grafů

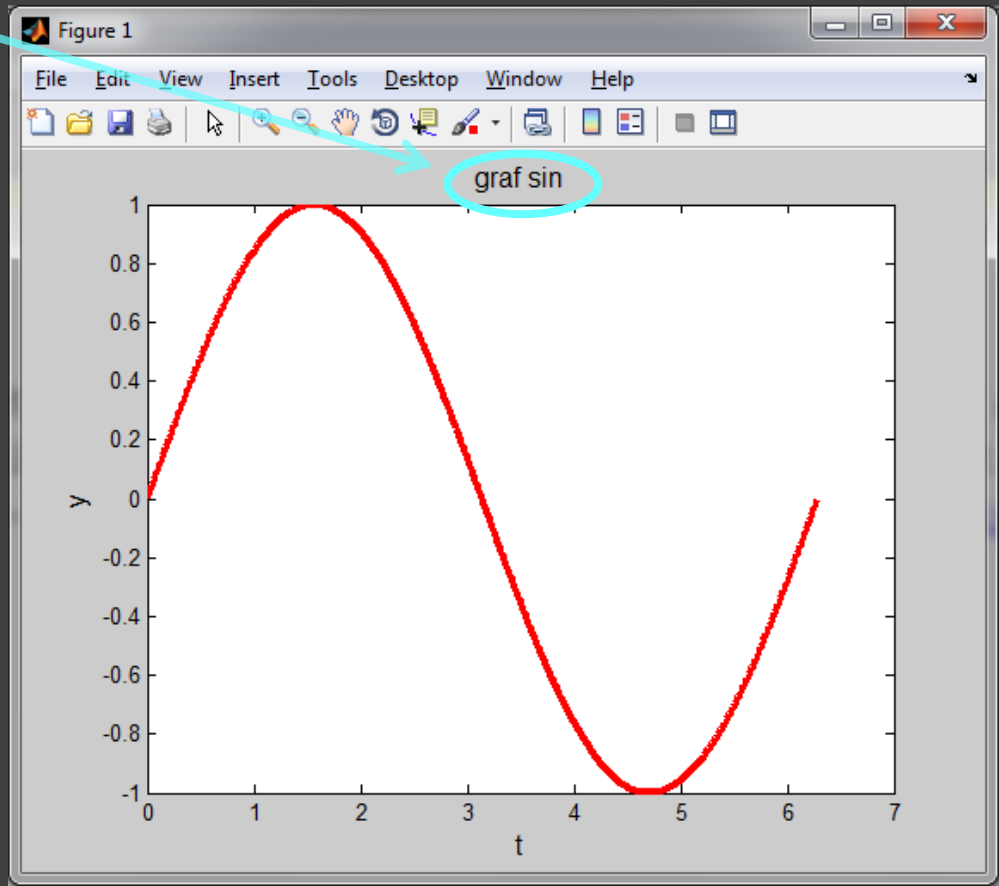
- název grafu – parametr – řetězec v apostrofech

```
title('graf sin')
```

– popis os – parametr
– řetězec v apostrofech

```
xlabel('t')
ylabel('y')
```

```
zlabel('z')
```
– pro 3D grafy



Základy tvorby 2D grafů

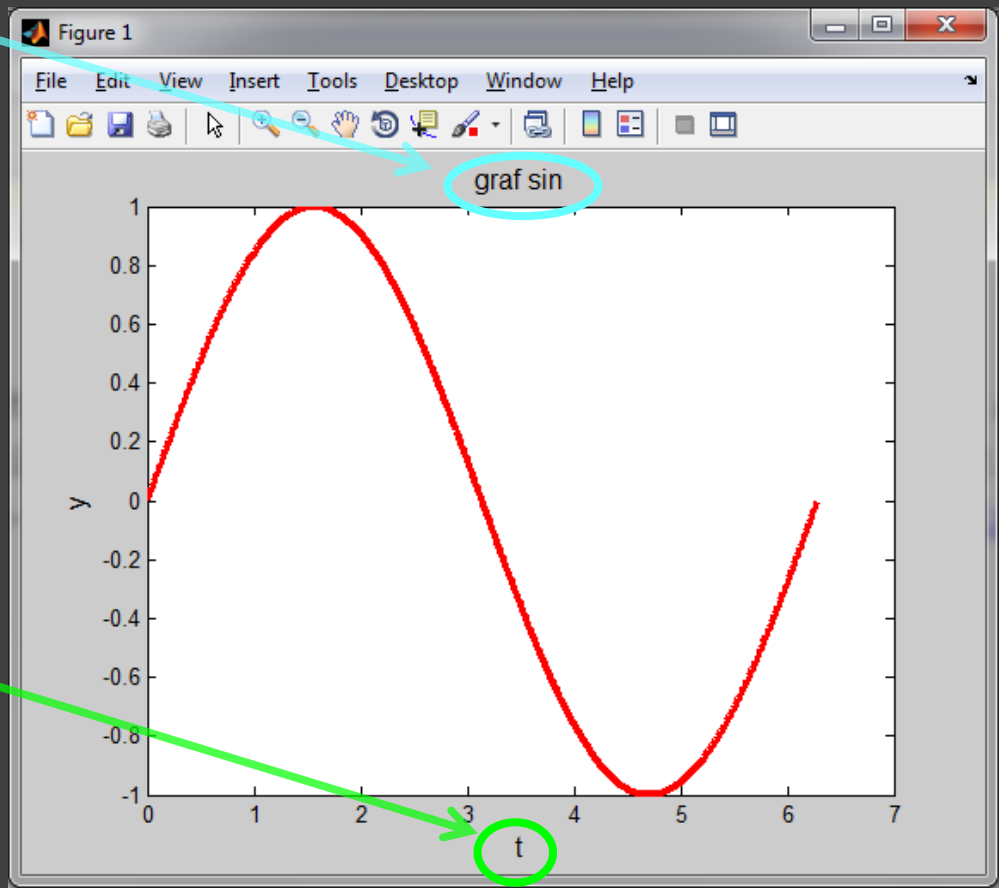
- název grafu – parametr – řetězec v apostrofech

```
title('graf sin')
```

– popis os – parametr
– řetězec v apostrofech

```
xlabel('t')
ylabel('y')
```

```
zlabel('z')
```
– pro 3D grafy



Základy tvorby 2D grafů

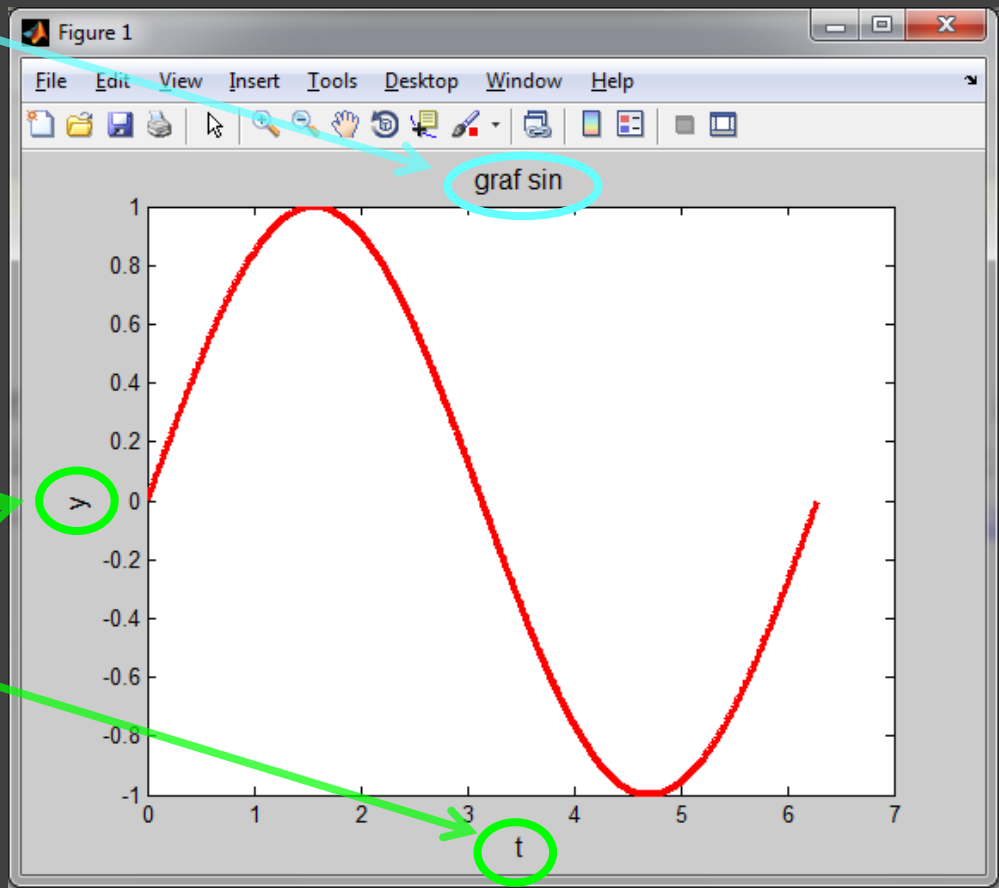
- název grafu – parametr – řetězec v apostrofech

```
title('graf sin')
```

– popis os – parametr
– řetězec v apostrofech

```
xlabel('t')
ylabel('y')
```

```
zlabel('z')
```
– pro 3D grafy



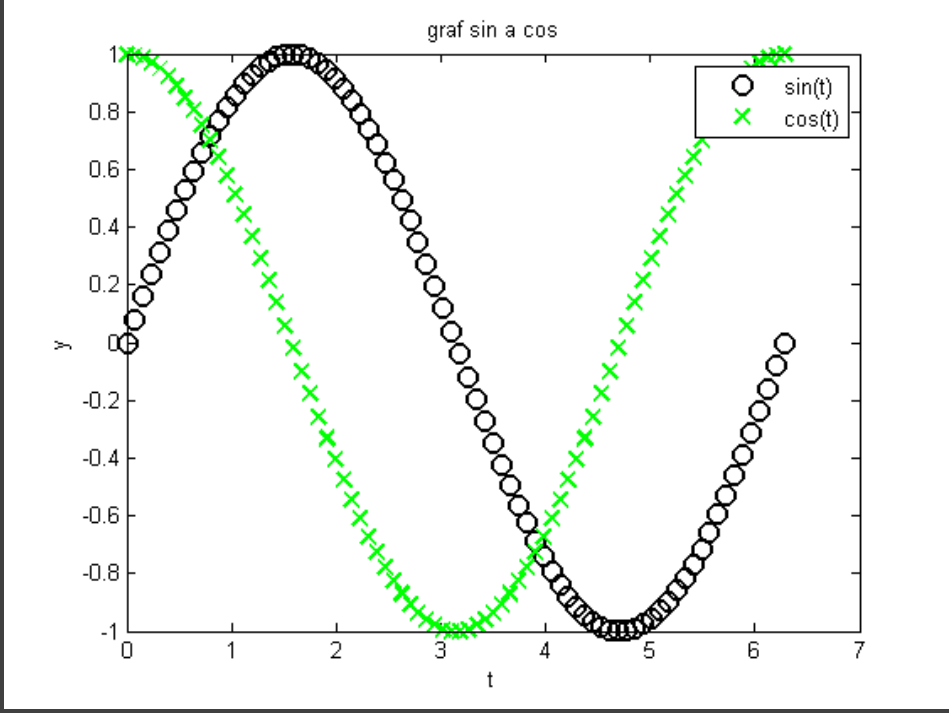
Základy tvorby 2D grafů

– více křivek v jednom grafu

`plot(x1,y1,x2,y2,…,xn,yn)`

Např. $y_1=\sin(t)$, $y_2=\cos(t)$

```
t = linspace(0,2*pi,80);
y1 = sin(t);
y2 = cos(t);
plot(t,y1,'ok',t,y2,'xg')
xlabel('t')
ylabel('y')
title('graf sin a cos')
legend('sin(t)','cos(t)')
```



Základy tvorby 2D grafů

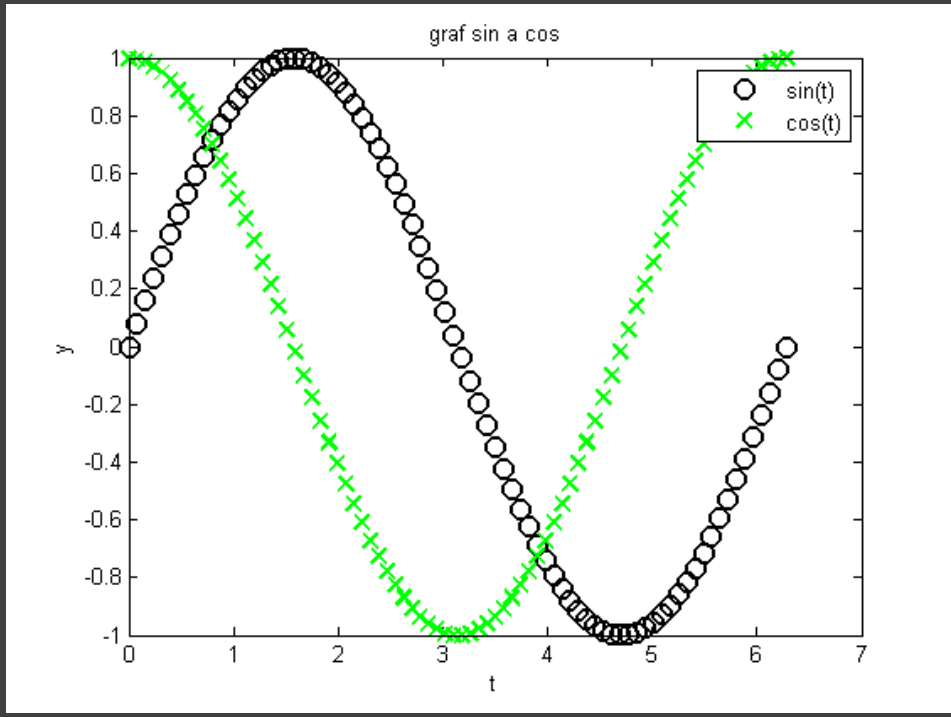
– více křivek v jednom grafu

`plot(x1,y1,x2,y2,…,xn,yn)`

Např. $y_1=\sin(t)$, $y_2=\cos(t)$

```
t = linspace(0,2*pi,80);
y1 = sin(t);
y2 = cos(t);
plot(t,y1,'ok',t,y2,'xg')
xlabel('t')
ylabel('y')
title('graf sin a cos')
legend('sin(t)','cos(t)')
```

– křivka `sin(t)` zobrazena černými kolečky (`'ok'`), křivka `cos(t)` zelenými křížky (`'xg'`)



Základy tvorby 2D grafů

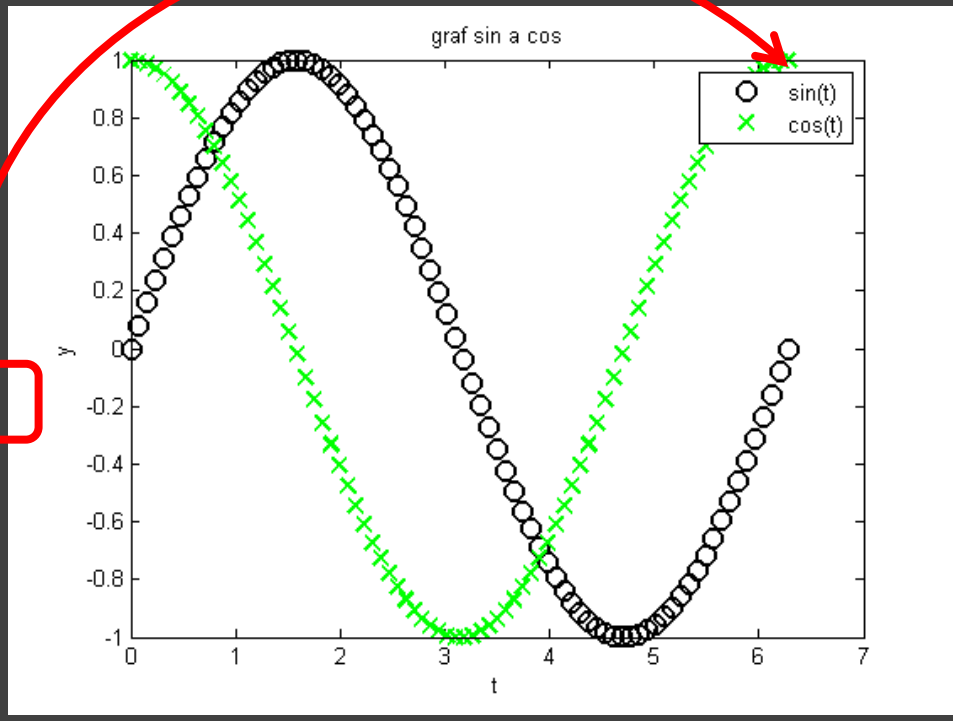
– více křivek v jednom grafu

`plot(x1,y1,x2,y2,…,xn,yn)`

Např. $y_1=\sin(t)$, $y_2=\cos(t)$

```
t = linspace(0,2*pi,80);
y1 = sin(t);
y2 = cos(t);
plot(t,y1,'ok',t,y2,'xg')
xlabel('t')
ylabel('y')
title('graf sin a cos')
legend('sin(t)','cos(t)')
```

– křivka `sin(t)` zobrazena černými kolečky (`'ok'`), křivka `cos(t)` zelenými křížky (`'xg'`)



Základy tvorby 2D grafů

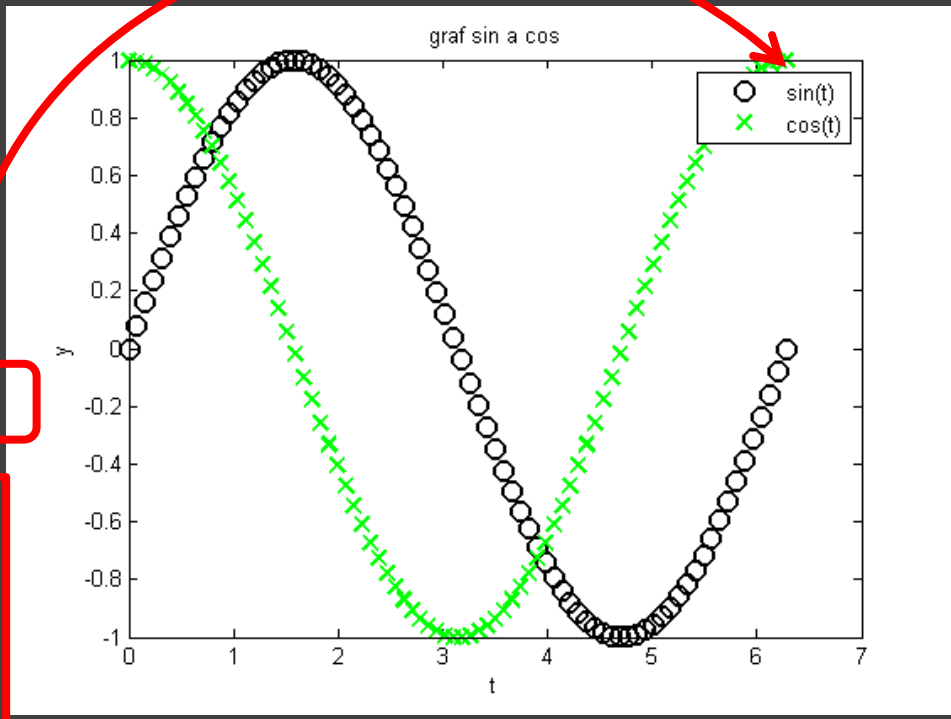
– více křivek v jednom grafu

`plot(x1,y1,x2,y2,…,xn,yn)`

Např. $y_1=\sin(t)$, $y_2=\cos(t)$

```
t = linspace(0,2*pi,80);
y1 = sin(t);
y2 = cos(t);
plot(t,y1,'ok',t,y2,'xg')
xlabel('t')
ylabel('y')
title('graf sin a cos')
legend('sin(t)','cos(t)')
```

– křivka `sin(t)` zobrazena černými kolečky (`'ok'`), křivka `cos(t)` zelenými křížky (`'xg'`)



– `legend ('řetězec1', …, 'řetězecn')` umístí legendu podle pořadí uvedených řetězců (nutno dodržet pořadí jednotlivých křivek, jak byly kresleny, např. příkazem `plot`)

Základy tvorby 2D grafů

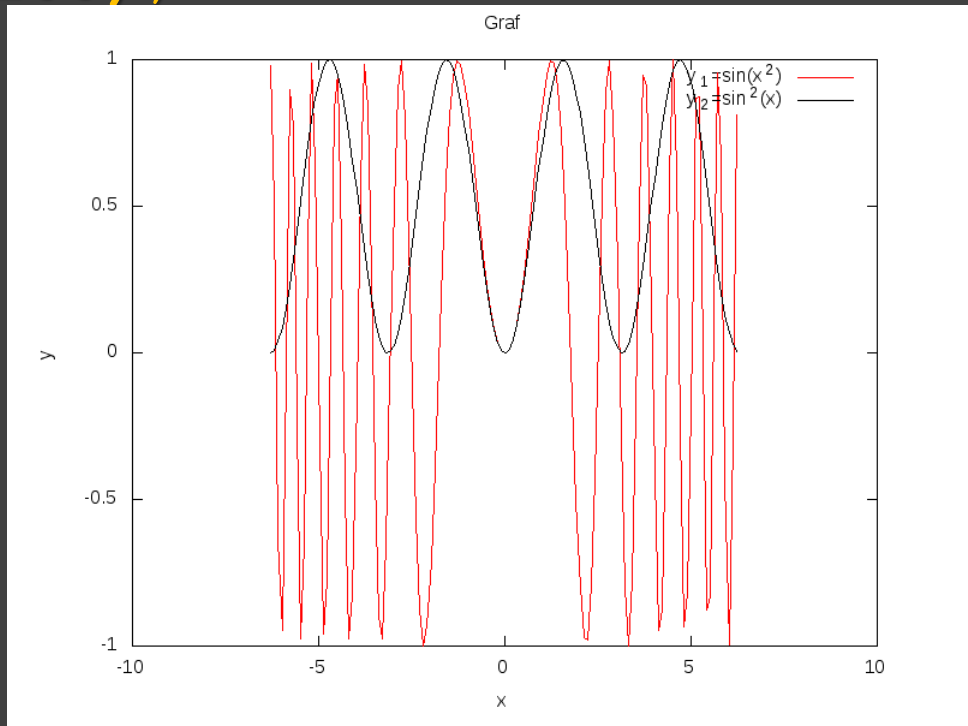
– více křivek v jednom grafu

hold on – přidrží aktuální graf v grafickém okně, lze nakreslit více grafů do jednoho grafického okna postupně

hold off – vypnutí, konec možnosti kreslit více grafů do jednoho grafického okna

Např. $y_1=\sin(x^2)$, $y_2=\sin^2(x)$ pro x od -2π do 2π.

```
x = linspace(-2*pi,2*pi,100);
y1 = sin(x.^2);
plot(x,y1,'r')
xlabel('x')
ylabel('y')
title('Graf')
hold on
y2 = (sin(x)).^2;
plot(x,y2,'k')
hold off
```



Základy tvorby 2D grafů

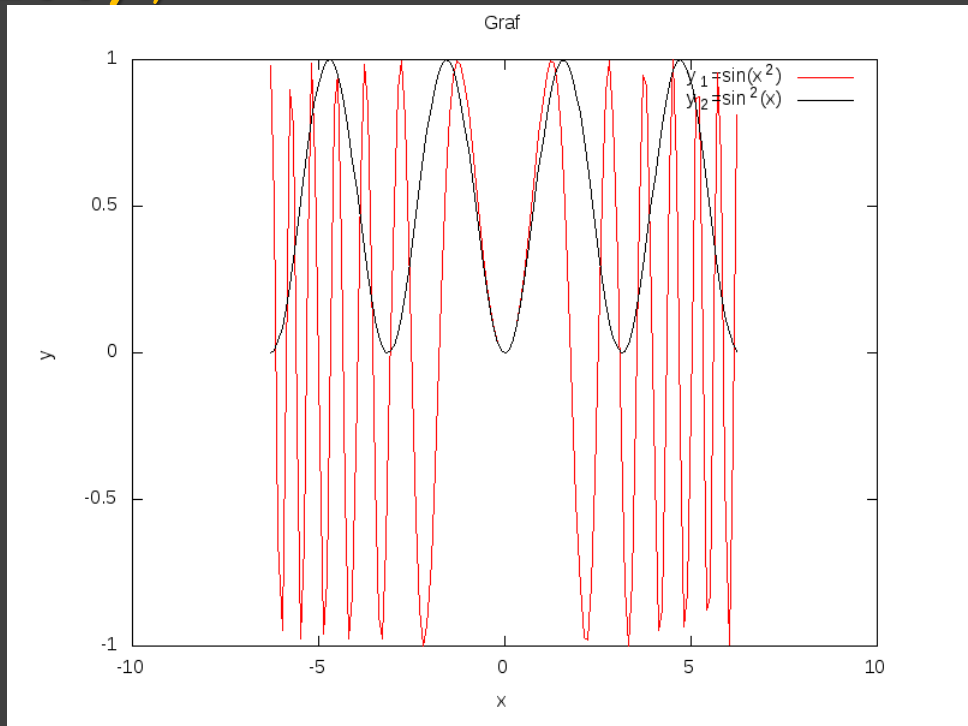
– více křivek v jednom grafu

hold on – přidrží aktuální graf v grafickém okně, lze nakreslit více grafů do jednoho grafického okna postupně

hold off – vypnutí, konec možnosti kreslit více grafů do jednoho grafického okna

Např. $y_1=\sin(x^2)$, $y_2=\sin^2(x)$ pro x od -2π do 2π.

```
x = linspace(-2*pi,2*pi,100);
y1 = sin(x.^2);
plot(x,y1,'r')
xlabel('x')
ylabel('y')
title('Graf')
hold on
y2 = (sin(x)).^2;
plot(x,y2,'k')
hold off
```



Základy tvorby 2D grafů

– více křivek v jednom grafu

hold on – přidrží aktuální graf v grafickém okně, lze nakreslit více grafů do jednoho grafického okna postupně

hold off – vypnutí, konec možnosti kreslit více grafů do jednoho grafického okna

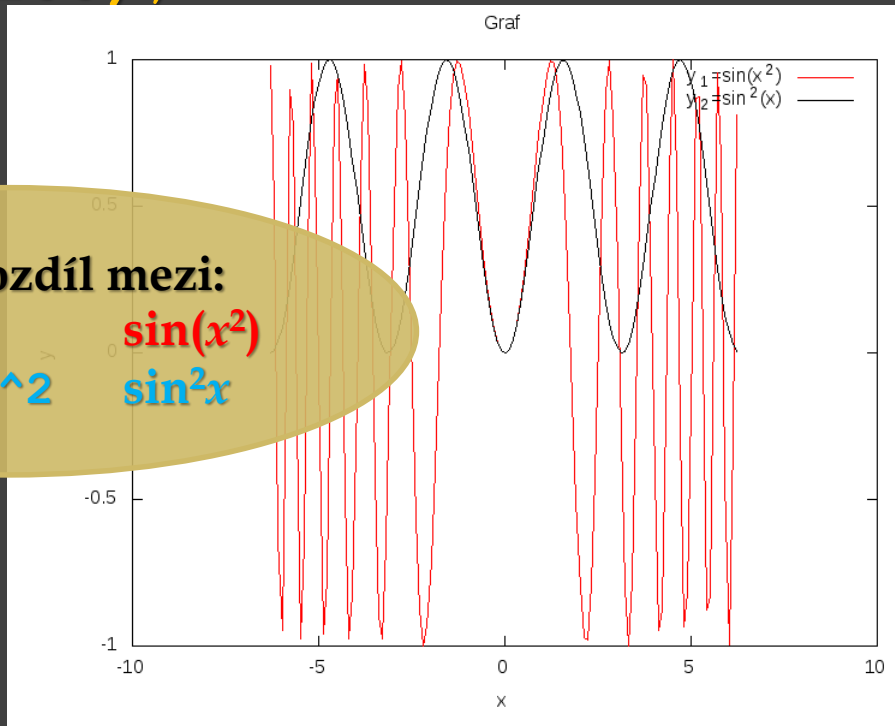
Např. $y_1=\sin(x^2)$, $y_2=\sin^2(x)$ pro x od -2π do 2π.

```
x = linspace(-2*pi,2*pi,100);
y1 = sin(x.^2);
plot(x,y1,'r')
xlabel('x')
ylabel('y')
title('Graf')
hold on
y2 = (sin(x)).^2;
plot(x,y2,'k')
hold off
```

Pozor – rozdíl mezi:

`sin(x.^2)` $\sin(x^2)$

`(sin(x)).^2` $\sin^2 x$



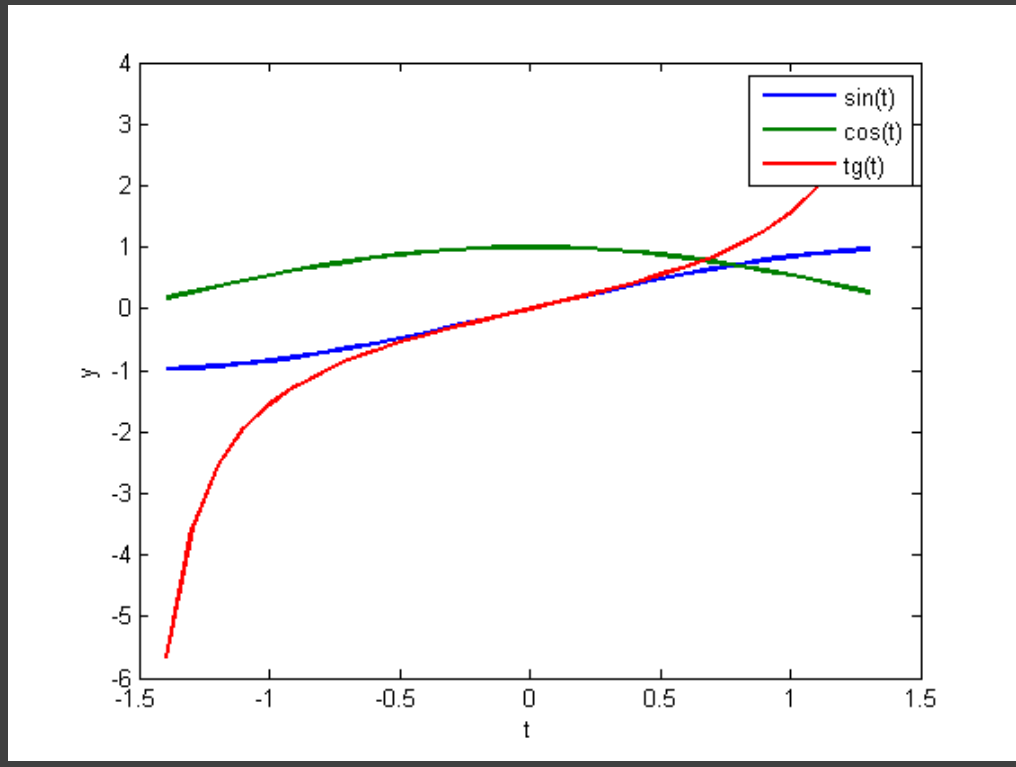
Základy tvorby 2D grafů

– **grid** – mřížka v grafu
grid on – přidává mřížku ke osám aktuálního grafu,
grid off – odstraňuje mřížku z aktuálního grafu,
grid – sám o sobě přidává / vypíná mřížku u aktuálního grafu.

Např. goniometrické funkce
y_1=sin(t), y_2=cos(t), y_3=tg(t)

```
t = -4*pi/9:0.1:4*pi/9;
y1 = sin(t);
y2 = cos(t);
y3 = tan(t);
plot(t,y1,t,y2,t,y3)
xlabel('t')
ylabel('y')

legend('sin(t)','cos(t)','tg(t)')
```

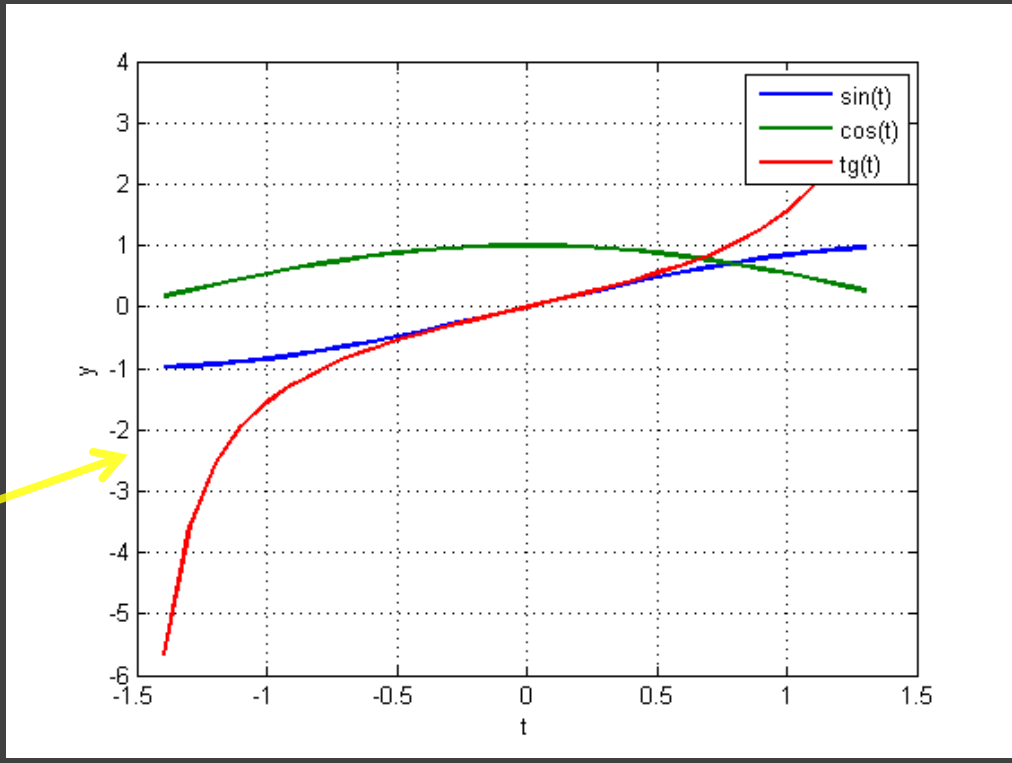


Základy tvorby 2D grafů

– **grid** – mřížka v grafu
grid on – přidává mřížku ke osám aktuálního grafu,
grid off – odstraňuje mřížku z aktuálního grafu,
grid – sám o sobě přidává / vypíná mřížku u aktuálního grafu.

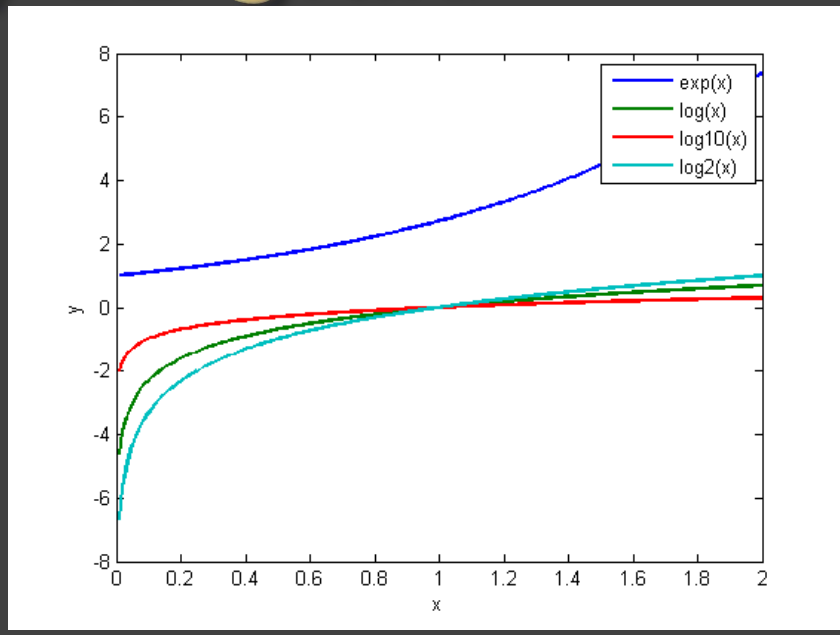
Např. goniometrické funkce
y_1=sin(t), y_2=cos(t), y_3=tg(t)

```
t = -4*pi/9:0.1:4*pi/9;
y1 = sin(t);
y2 = cos(t);
y3 = tan(t);
plot(t,y1,t,y2,t,y3)
xlabel('t')
ylabel('y')
grid
legend('sin(t)','cos(t)','tg(t)')
```



Základy tvorby 2D grafů

Příklad: vykreslení funkcí
$y_1 = e^x$, $y_2 = \ln(x)$,
$y_3 = \log_{10}(x)$, $y_4 = \log_2(x)$
pro x od **0.01** do **2**




```
x = 0.01:0.01:2;
plot(x,exp(x),x,log(x),x,log10(x),x,log2(x))
xlabel('x'); ylabel('y');
legend('exp(x)','log(x)','log10(x)','log2(x)')
```

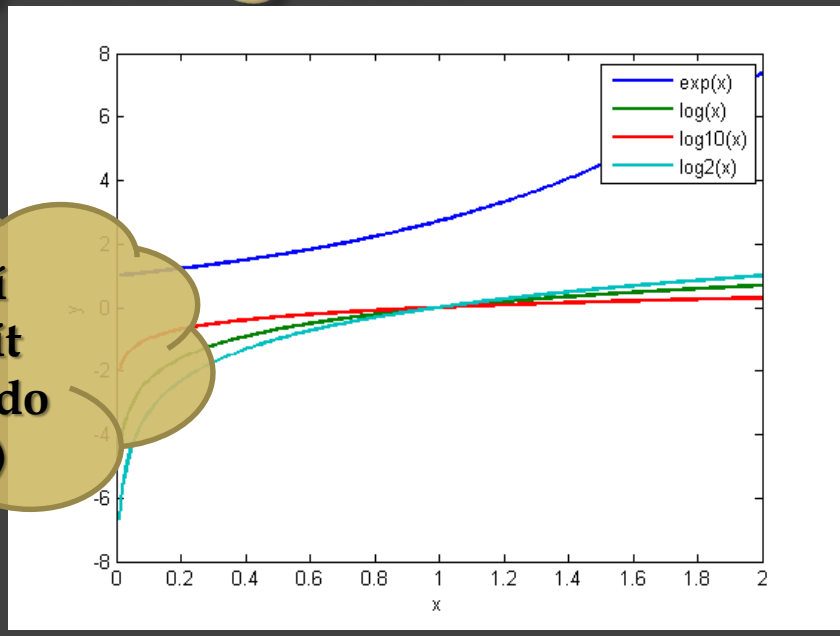
Použité funkce:

`exp(x)` – exponenciální funkce e^x
`log(x)` – logaritmus při základu e – přirozený (v matematice **ln** x)
`log10(x)` – logaritmus při základu **10** – dekadický (**log** x)
`log2(x)` – logaritmus při základu **2** – binární (dvojkový)

Pozn. `logm(A)` – maticový logaritmus (jen pro čtvercové matice)

Základy tvorby 2D grafů

Příklad: vykreslení funkcí
$y_1 = e^x$, $y_2 = \ln(x)$,
$y_3 = \log_{10}(x)$, $y_4 = \log_2(x)$
pro x od **0.01** do **2**



Pro vykreslení grafu lze použít vnoření funkce do funkce `plot()`

```
x = 0.01:0.01:2;
plot(x,exp(x),x,log(x),x,log10(x),x,log2(x))
xlabel('x'); ylabel('y');
legend('exp(x)','log(x)','log10(x)','log2(x)')
```

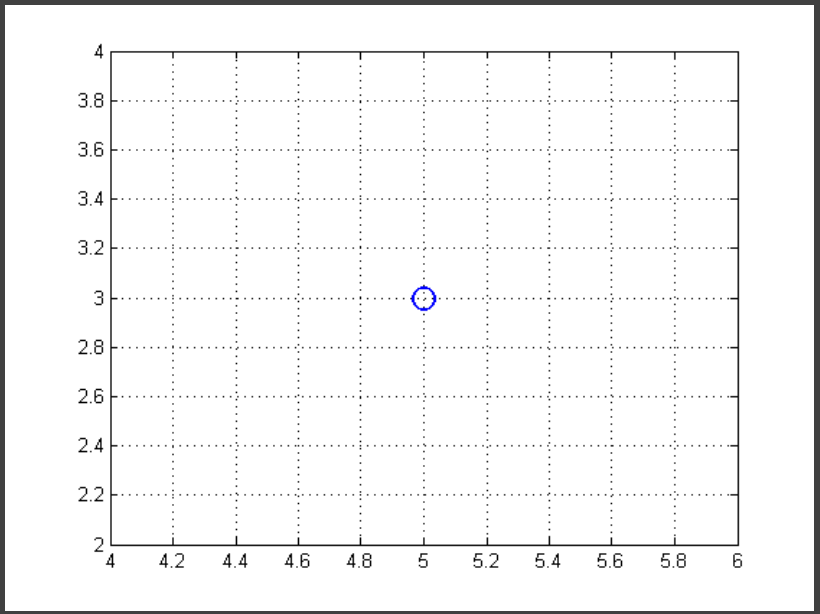
Použité funkce:

`exp(x)` – exponenciální funkce e^x
`log(x)` – logaritmus při základu e – přirozený (v matematice **ln** x)
`log10(x)` – logaritmus při základu **10** – dekadický (**log** x)
`log2(x)` – logaritmus při základu **2** – binární (dvojkový)
`logm(A)` – maticový logaritmus (jen pro čtvercové matice)

Základy tvorby 2D grafů

Příklad: bod o souřadnicích [5,3]

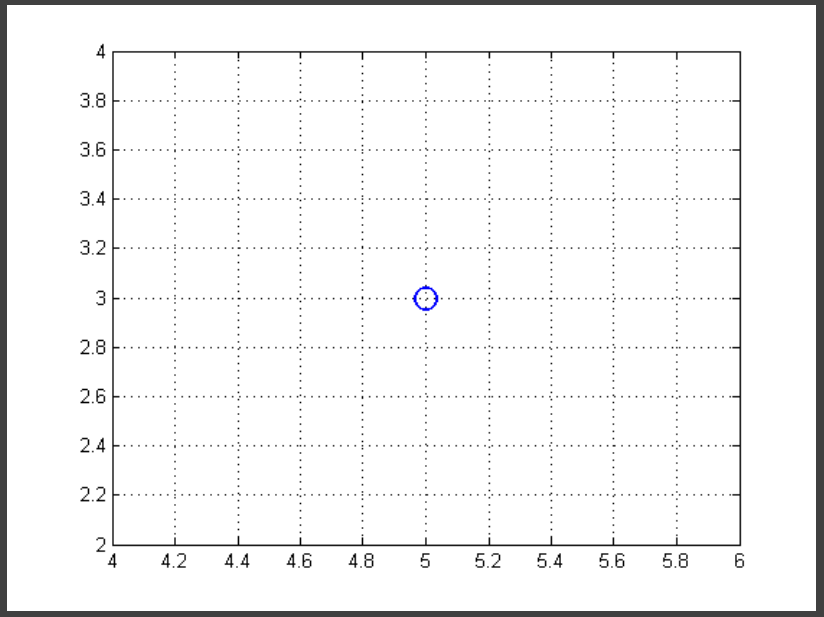
```
plot(5,3,'o');
```



Základy tvorby 2D grafů

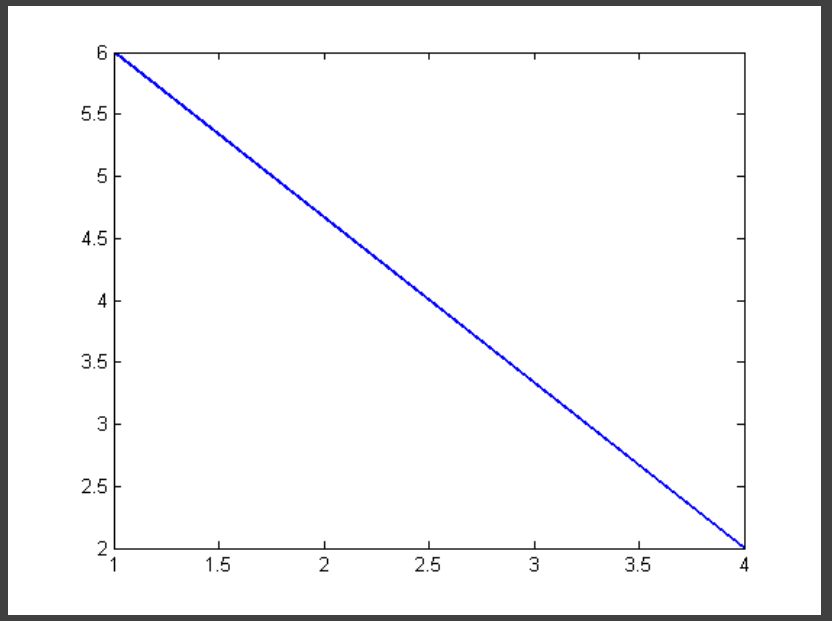
Příklad: bod o souřadnicích [5,3]

```
plot(5,3,'o');
```



Příklad: úsečka z bodu [1,6]
do bodu [4,2]

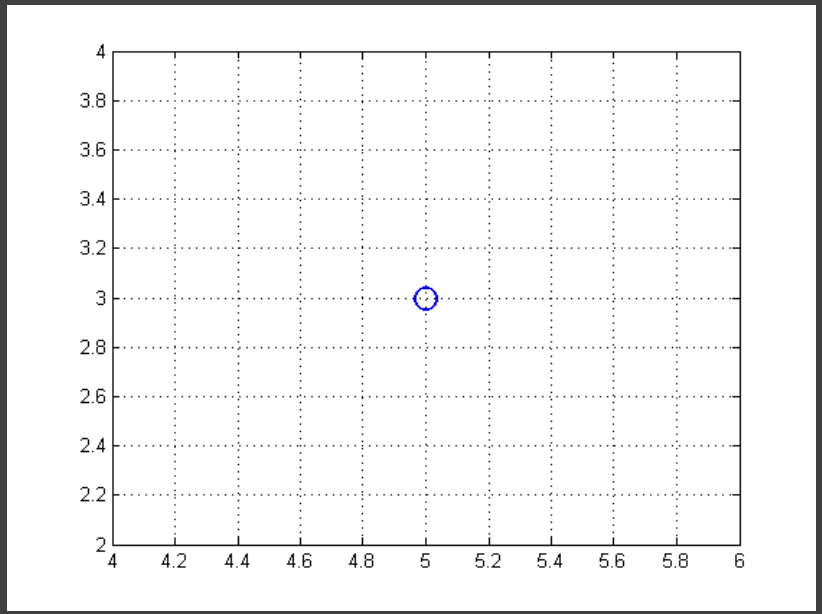
```
plot([1,4],[6,2])
```



Základy tvorby 2D grafů

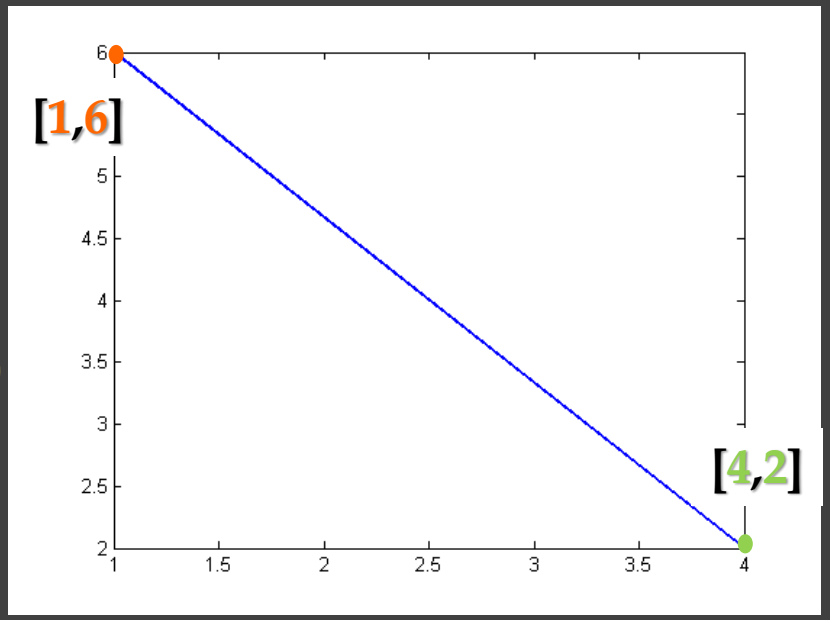
Příklad: bod o souřadnicích [5,3]

```
plot(5,3,'o');
```



Příklad: úsečka z bodu [1,6]
do bodu [4,2]

```
plot([1,4],[6,2])
```



první parametr příkazu `plot`
***x*-ové souřadnice bodů,**
druhý parametr příkazu `plot`
***y*-ové souřadnice bodů**

Funkce pro práci s vektory a maticemi

`size(matice)` – vrátí počet **řádků** a **sloupců** matice
`size(matice, 1)` – vrátí počet **řádků** matice
`size(matice, 2)` – vrátí počet **sloupců** matice

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

```
size(A,1)
ans =
     2
```

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

```
size(A,2)
ans =
     3
```

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

Funkce pro práci s vektory a maticemi

`length(v)` – vrací jedno číslo – délku vektoru **`v`**, tj. počet prvků ve vektoru **`v`** (u matic vrátí větší rozměr z počtu řádek a sloupců)

```
a = [9,1,8,2,7,3]
a =
     9   1   8   2   7   3
length(a)
ans =
      6
```

```
A = [1,2,3;4,5,6]
A =
     1     2     3
     4     5     6
length(A)
ans =
      3
```

```
b = [29;-1;78;6]
b =
    29
    -1
    78
     6
length(b)
ans =
      4
```

Funkce pro práci s vektory a maticemi

Pozn.: Pro velké řídké matice převážně plné nul, lze užít "komprese" příkazem **`sparse()`**, který matici převede na „řídkou formu“ vypuštěním nul.

Příklad:

```
V =
     0     7     0     9     0     0     0     0     0
     0     0     6     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     5     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

```
K = sparse(V)
K =
   (1,2)        7
   (2,3)        6
   (1,4)        9
   (5,5)        4
   (7,8)        5
```

```
whos
  Name          Size                    Bytes  Class     Attributes

  K             9x9                       100  double    sparse
  V             9x9                       648  double
```

Funkce pro práci s vektory a maticemi

Pokračování příkladu:

```
clear V      % odstranění matice V
```

```
whos
  Name        Size                Bytes  Class     Attributes
  K           9x9                   100  double    sparse
```

```
K
K =
    (1,2)        7
    (2,3)        6
    (1,4)        9
    (5,5)        4
    (7,8)        5
```

```
D = K.^2    % operace prvek po prvku
D =
    (1,2)       49
    (2,3)       36
    (1,4)       81
    (5,5)       16
    (7,8)       25
```

```
whos
  Name        Size                Bytes  Class     Attributes
  K           9x9                   100  double    sparse
  D           9x9                   100  double    sparse
```

Funkce pro práci s vektory a maticemi

Pozn.: **`full()`** – převede řídkou matici na plný tvar, tj. opačná funkce k **`sparse`** („nafoukne“ matici zpět)

Pokračování příkladu:

```
D
D =
    (1,2)       49
    (2,3)       36
    (1,4)       81
    (5,5)       16
    (7,8)       25
```

```
Q = full(D)
Q =
     0    49     0    81     0     0     0     0     0
     0     0    36     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0    16     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0    25     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

```
clear K
clear D
```

```
whos
  Name        Size                Bytes  Class     Attributes
  Q           9x9                   648  double
```

Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,3];
```

sum(a) – součet hodnot všech prvků ve vektoru **a**

```
ans =
     30
```

cumsum(a) – kumulativní součet – součet prvků se všemi předchůdci

```
ans =
     9    10    18    20    27    30
```

prod(a) – součin hodnot všech prvků ve vektoru **a**

```
ans =
     3024
```

cumprod(a) – kumulativní součin – součin prvků se všemi předchůdci

```
ans =
     9     9    72   144   1008   3024
```

Funkce pro práci s vektory a maticemi

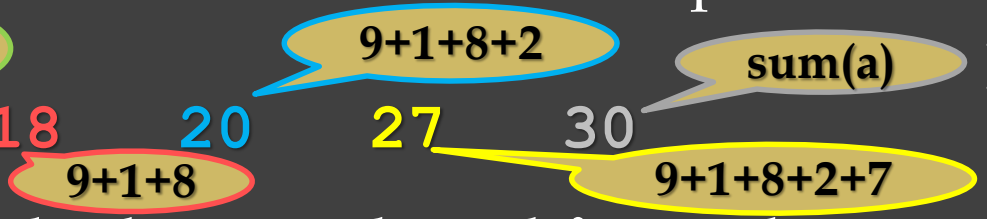
```
a = [9,1,8,2,7,3];
```

sum(a) – součet hodnot všech prvků ve vektoru **a**

```
ans =
     30
```

cumsum(a) – kumulativní součet – součet prvků se všemi předchůdci

```
ans =
     9    10    18    20    27    30
```



prod(a) – součin hodnot všech prvků ve vektoru **a**

```
ans =
     3024
```

cumprod(a) – kumulativní součin – součin prvků se všemi předchůdci

```
ans =
     9     9    72   144  1008  3024
```

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

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

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

```
sum(A)
ans =
     5     7     9
```
– součty prvků ve **sloupcích** matice

```
sum(A,1)
ans =
     5     7     9
```
– totéž (součty ve **sloupcích**)

```
sum(A,2)
ans =
     6
    15
```
– součty prvků v **řádcích** matice

```
sum(A.')
ans =
     6    15
```
– součty čísel v **řádcích** matice (tj. v sloupcích matice transponované)

Funkce pro práci s vektory a maticemi

Součet prvků v matici, např.

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

```
sum(A)
ans =
     5     7     9
```

– součty prvků ve sloupcích matice

```
sum(sum(A))
ans =
    21
```

– sečte všechny prvky v matici

Funkce pro práci s vektory a maticemi

Součet prvků v matici, např.

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

```
sum(A)
ans =
     5     7     9
```

– součty prvků ve sloupcích matice

```
sum(A.')
ans =
     6    15
```

– součty čísel v řádcích matice (tj. v sloupcích matice transponované)

```
sum(sum(A))
ans =
    21
```

– sečte všechny prvky v matici

```
sum(sum(A.'))
ans =
    21
```

– sečte rovněž všechny prvky

Funkce pro práci s vektory a maticemi

Součet prvků v matici, např.

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

```
sum(A)
ans =
     5     7     9
```
– součty prvků ve sloupcích matice

```
sum(sum(A))
ans =
    21
```
– sečte všechny prvky v matici

sum(sum(A)), tj. **sum([5 7 9])**
5 + 7 + 9 = 21
sum(sum(A)), tj. **sum([6 15])**
6 + 15 = 21

```
sum(A.')
ans =
     6    15
```
– součty čísel v řádcích matice (tj. v sloupcích matice transponované)

```
sum(sum(A.'))
ans =
    21
```
– sečte rovněž všechny prvky

Funkce pro práci s vektory a maticemi

`max(x)` – nalezne prvek s **maximální** velikostí ve vektoru *x*

```
a = [9,1,8,2,7,6];
```

```
max(a)
ans =
     9
```

```
b = [1,2,8,3,7];
```

```
max(b)
ans =
     8
```

Funkce pro práci s vektory a maticemi

max(x) – nalezne prvek s **maximální** velikostí ve vektoru *x*

```
a = [9,1,8,2,7,6];
```

```
max(a)
ans =
     9
```

```
b = [1,2,8,3,7];
```

```
max(b)
ans =
     8
```

[m,p] = max(x) – nalezne **maximum** a zobrazí **polohu** (**index**) zjišťovaného prvku ve vektoru *x*.

```
[ma,pa] = max(a)
     ma =  9
     pa =  1
```

```
[mb,pb] = max(b)
     mb =  8
     pb =  3
```

Funkce pro práci s vektory a maticemi

max(x) – nalezne prvek s **maximální** velikostí ve vektoru *x*

```
a = [9,1,8,2,7,6];
```

```
max(a)
ans =
     9
```

```
b = [1,2,8,3,7];
```

```
max(b)
ans =
     8
```

[m,p] = max(x) – nalezne **maximum** a zobrazí **polohu** (**index**) zjišťovaného prvku ve vektoru *x*.

```
[ma,pa] = max(a)
     ma =  9
     pa =  1
```

```
[mb,pb] = max(b)
     mb =  8
     pb =  3
```

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
mc = max(c)
mc =
     5
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Funkce pro práci s vektory a maticemi

max(x) – nalezne prvek s **maximální** velikostí ve vektoru *x*

```
a = [9,1,8,2,7,6];
```

```
max(a)
ans =
     9
```

```
b = [1,2,8,3,7];
```

```
max(b)
ans =
     8
```

[m,p] = max(x) – nalezne **maximum** a zobrazí **polohu** (**index**) zjišťovaného prvku ve vektoru *x*.

```
[ma,pa] = max(a)
     ma =  9
     pa =  1
```

```
[mb,pb] = max(b)
     mb =  8
     pb =  3
```

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
mc = max(c)
mc =
     5
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Pokud je maxim více, je vrácen index toho prvního.

Funkce pro práci s vektory a maticemi

`min(x)` – nalezne prvek s **minimální** velikostí ve vektoru *x*

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

```
min(a)
ans =
     1
```

```
min(b)
ans =
     1
```

`[mi,p] = min(x)` – nalezne **minimum** a zobrazí **polohu** (**index**) zjišťovaného prvku ve vektoru *x*.

```
[mia,ina] = min(a)
     mia = 1
     ina = 2
```

```
[mib,inb] = min(b)
     mib = 1
     inb = 1
```

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
mic = min(c)
mic =
    -4
```

```
[mic,inc] = min(c)
    mic = -4
    inc = 2
```

Funkce pro práci s vektory a maticemi

`min(x)` – nalezne prvek s **minimální** velikostí ve vektoru *x*

```
a = [9,1,8,2,7,6];
```

```
min(a)
ans =
     1
```

```
b = [1,2,8,3,7];
```

```
min(b)
ans =
     1
```

`[mi,p] = min(x)` – nalezne **minimum** a zobrazí **polohu** (**index**) zjišťovaného prvku ve vektoru *x*.

```
[mia,ina] = min(a)
     mia = 1
     ina = 2
```

```
[mib,inb] = min(b)
     mib = 1
     inb = 1
```

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
mic = min(c)
mic =
    -4
```

```
[mic,inc] = min(c)
    mic = -4
    inc = 2
```

Pokud je minim více, je vrácen index toho prvního.

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
max(B)
ans =
     3     5     4     2     5
```

– maxima prvků ve sloupcích matice

```
[m,p] = max(B)
  m =
     3     5     4     2     5
  p =
     1     2     1     3     2
```

```
B. '
ans =
     3    -4    -3
     4     5     0
     4     1    -3
    -3    -1     2
     3     5    -2
```

```
[mt,pt] = max(B.')
    mt =     4     5     2
    pt =     2     2     4
```

– maxima z prvků v řádcích matice (tj. v sloupcích matice transponované) s indexy

`max(max(B))` nebo `max(max(B.'))`

```
ans =
     5
```

– maximum z celé matice

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
max(B)
ans =
     3     5     4     2     5
```

– maxima prvků ve sloupcích matice

```
[m,p] = max(B)
  m =
     3     5     4     2     5
  p =
     1     2     1     3     2
```

poloha (index) maxima

```
B. '
ans =
     3    -4    -3
     4     5     0
     4     1    -3
    -3    -1     2
     3     5    -2
```

```
[mt,pt] = max(B.')
     mt =     4     5     2
     pt =     2     2     4
```

– maxima z prvků v řádcích matice (tj. v sloupcích matice transponované) s indexy

`max(max(B))` nebo `max(max(B.'))`

```
ans =
     5
```

– maximum z celé matice

Funkce pro práci s vektory a maticemi

Tytéž výsledky lze získat i pomocí parametru 1 nebo 2 (u `max()` na třetí pozici), např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
[m,p] = max(B,[],1)
  m =
     3     5     4     2     5
  p =
     1     2     1     3     2
```

```
[m,p] = max(B,[],2)
  m =
     4
     5
     2
  p =
     2
     2
     4
```

poloha (index) maxima

viz `help max`

Funkce pro práci s vektory a maticemi

`sort(x)` – **setřídění** prvků ve vektoru *x* vzestupně dle velikosti

```
a = [9,1,8,2,7,6];
```

```
sort(a)
ans =
     1   2   6   7   8   9
```

```
b = [1,2,8,3,7];
```

```
sort(b)
ans =
     1   2   3   7   8
```

Funkce pro práci s vektory a maticemi

`sort(x)` – **setřídění** prvků ve vektoru *x* vzestupně dle velikosti

```
a = [9,1,8,2,7,6];
```

```
sort(a)
ans =
     1   2   6   7   8   9
```

```
b = [1,2,8,3,7];
```

```
sort(b)
ans =
     1   2   3   7   8
```

`[s,p] = sort(x)` – **setřídí** prvky vzestupně a zobrazí jejich **polohu** (**index**) ve vektoru *x*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[sc,pc] = sort(c)
sc =
  -4   -4   -3   -2   -1    0    1    2    3    4    5    5
pc =
   2    8    4    1   10    7    3   11   12    5    6    9
```

Funkce pro práci s vektory a maticemi

`sort(x)` – setřídění prvků ve vektoru *x* vzestupně dle velikosti

```
a = [9,1,8,2,7,6];
```

```
sort(a)
ans =
     1   2   6   7   8   9
```

```
b = [1,2,8,3,7];
```

```
sort(b)
ans =
     1   2   3   7   8
```

`[s,p] = sort(x)` – setřídí prvky vzestupně a zobrazí jejich polohu (index) ve vektoru *x*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[sc,pc] = sort(c)
sc =
  -4  -4  -3  -2  -1   0   1   2   3   4   5   5
pc =
   2   8   4   1  10   7   3  11  12   5   6   9
```

Funkce pro práci s vektory a maticemi

`sort(x,'způsob')` – setřídění prvků ve vektoru *x* dle velikosti podle zvoleného způsobu: `'ascend'` – vzestupně (defaultní nastavení), `'descend'` – sestupně, např.

```
a = [9,1,8,2,7,6];
```

```
b = [1,2,8,3,7];
```

```
sort(a)
```
– totéž jako
```
sort(a,'ascend')
ans =
     1   2   6   7   8   9
```

```
sort(b,'descend')
ans =
     8   7   3   2   1
```

Funkce pro práci s vektory a maticemi

`sort(x,'způsob')` – setřídění prvků ve vektoru *x* dle velikosti podle zvoleného způsobu: `'ascend'` – vzestupně (defaultní nastavení), `'descend'` – sestupně, např.

```
a = [9,1,8,2,7,6];
```

```
b = [1,2,8,3,7];
```

```
sort(a)
```
– totéž jako
```
sort(a,'ascend')
ans =
     1   2   6   7   8   9
```

```
sort(b,'descend')
ans =
     8   7   3   2   1
```

```
q = sort(b)
q =
     1   2   3   7   8
q(end:-1:1)
ans =
     8   7   3   2   1
```

sestupné setřídění - zobrazení vzestupně seřazeného vektoru od konce (`end`) k `1`. prvku

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

`[s,p] = sort(B)`, příp. `sort(B,1)`

```
s =
    -4     0    -3    -3    -2
    -3     4     1    -1     3
     3     5     4     2     5
p =
     2     3     3     1     3
     3     1     2     2     1
     1     2     1     3     2
```

– seřadí prvky ve **sloupcích** ,
– zobrazí indexy prvků ve **sloupci**

`[s2,p2] = sort(B,2)`

```
s2 =
    -3     3     3     4     4
    -4    -1     1     5     5
    -3    -3    -2     0     2
p2 =
     4     1     5     2     3
     1     4     3     2     5
     1     3     5     2     4
```

– seřadí prvky v **řádcích** ,
– zobrazí jejich index v **řádku**

Funkce pro práci s vektory a maticemi

Podobně – volba způsobu setřídění pro matice, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
[sest,psest] = sort(B,2, 'descend')
sest =
     4     4     3     3    -3
     5     5     1    -1    -4
     2     0    -2    -3    -3
psest =
     2     3     1     5     4
     2     5     3     4     1
     4     2     5     1     3
```

– seřadí prvky v **řádcích sestupně**

– zobrazí indexy (polohu) prvků v **řádku**

Funkce pro práci s vektory a maticemi

`sortrows(X)` – setřídění řádků vzestupně podle velikosti 1. prvku
`sortrows(X,sloupec)` – setřídění řádků vzestupně na základě prvků vybraného sloupce, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
[r1,p1] = sortrows(B), příp.
r1 =                          sortrows(B,1)
    -4     5     1    -1     5
    -3     0    -3     2    -2
     3     4     4    -3     3
p1 =
     2
     3
     1
```

– zobrazení indexů jednotlivých řádků v matici (setříděno podle **1. sloupce**)

Funkce pro práci s vektory a maticemi

`sortrows(X)` – setřídění řádků vzestupně podle velikosti 1. prvku
`sortrows(X,sloupec)` – setřídění řádků vzestupně na základě prvků vybraného sloupce, např.

```
B =
     3     4     4    -3     3
    -4     5     1    -1     5
    -3     0    -3     2    -2
```

```
[r1,p1] = sortrows(B), příp.
r1 =                          sort(B,1)
    -4     5     1    -1     5
    -3     0    -3     2    -2
     3     4     4    -3     3
p1 =
     2
     3
     1
```

– zobrazení indexů jednotlivých řádků v matici (setříděno podle **1. sloupce**)

```
[r5,p5] = sortrows(B,5)
r5 =
    -3     0    -3     2    -2
     3     4     4    -3     3
    -4     5     1    -1     5
p5 =
     3
     1
     2
```

– zobrazení indexů jednotlivých řádků v matici (setříděno podle **5. sloupce**)

```
[s6,p6] = sortrows(B,6)
??? Error
```

– **nelze**: **6. sloupec** matice **B** neexistuje

Funkce pro práci s vektory a maticemi

`mean(x)` – průměr ze všech prvků ve vektoru: $\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$

```
a = [9,1,8,2,7,6];
```

```
mean(a)
ans = 5.50
```

```
sum(a)/length(a)
ans = 5.50
```

```
b = [1,2,8,3,7];
```

```
mean(b)
ans = 4.20
```

```
sum(b)/length(b)
ans = 4.20
```

Funkce pro práci s vektory a maticemi

mean(x) – průměr ze všech prvků ve vektoru: $\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$

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

```
mean(a)
ans = 5.50
```

```
mean(b)
ans = 4.20
```

```
sum(a)/length(a)
ans = 5.50
```

```
sum(b)/length(b)
ans = 4.20
```

median(a) – střední hodnota, pro lichý počet prvků ve vektoru při seřazení je to prostřední prvek, pro sudý počet je to průměr z velikosti dvou členů nejblíže středu seřazeného vektoru

```
sort(a)
ans = 1   2   6   7   8   9
```

```
sort(b)
ans = 1   2   3   7   8
```

```
median(a)
ans = 6.50
```

```
median(b)
ans = 3
```

Funkce pro práci s vektory a maticemi

mean(x) – průměr ze všech prvků ve vektoru: $\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$

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

```
mean(a)
ans = 5.50
```

```
mean(b)
ans = 4.20
```

```
sum(a)/length(a)
ans = 5.50
```

```
sum(b)/length(b)
ans = 4.20
```

median(a) – střední hodnota, pro lichý počet prvků ve vektoru při seřazení je to prostřední prvek, pro sudý počet je to průměr z velikosti dvou členů nejblíže středu seřazeného vektoru

```
sort(a)
ans = 1   2   6   7   8   9
```

```
median(a)
ans = 6.50
```

(6+7)/2

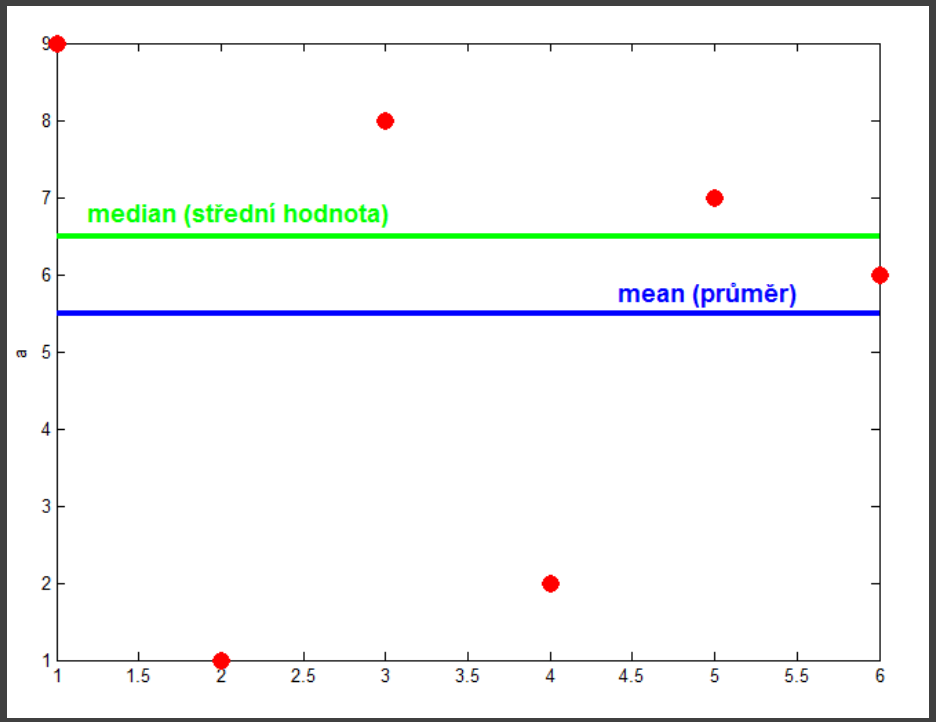
```
sort(b)
ans = 1   2   3   7   8
```

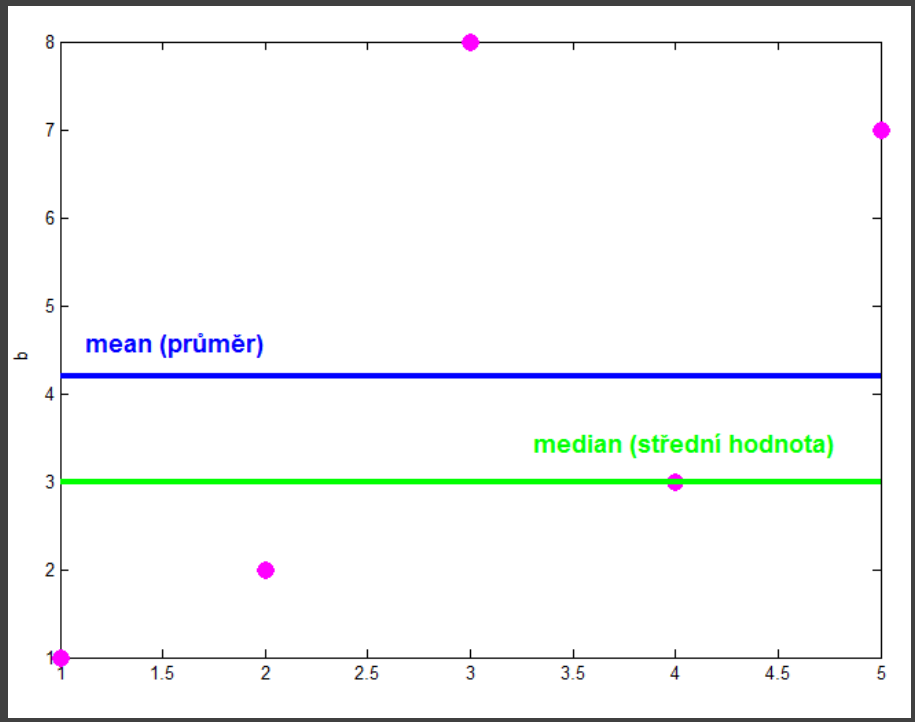
```
median(b)
ans = 3
```

Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,6];
mean(a); median(a);
```

```
b = [1,2,8,3,7];
mean(b); median(b);
```

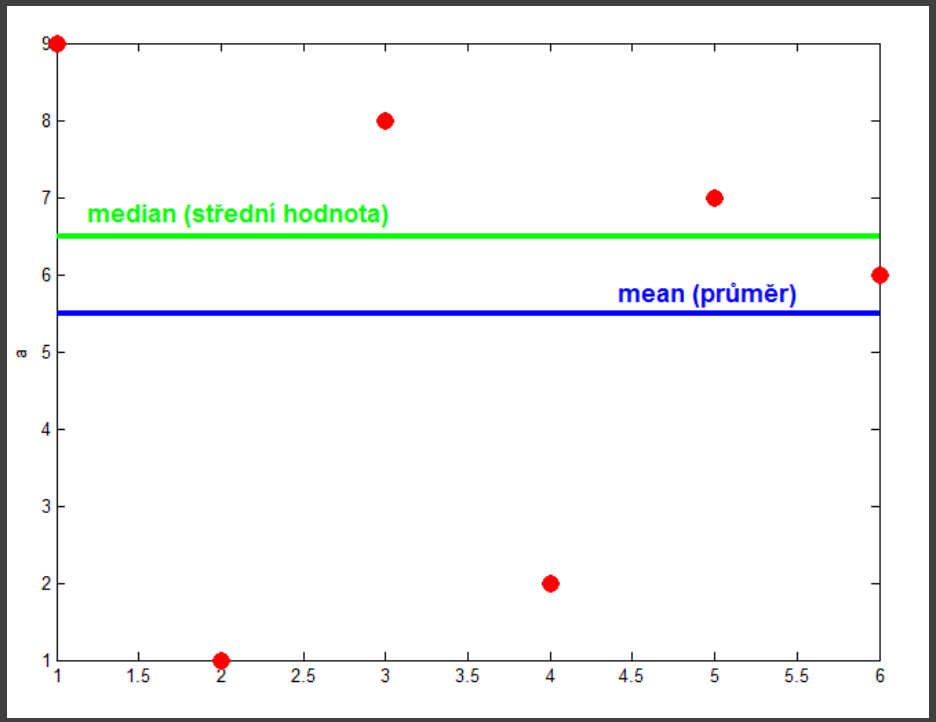


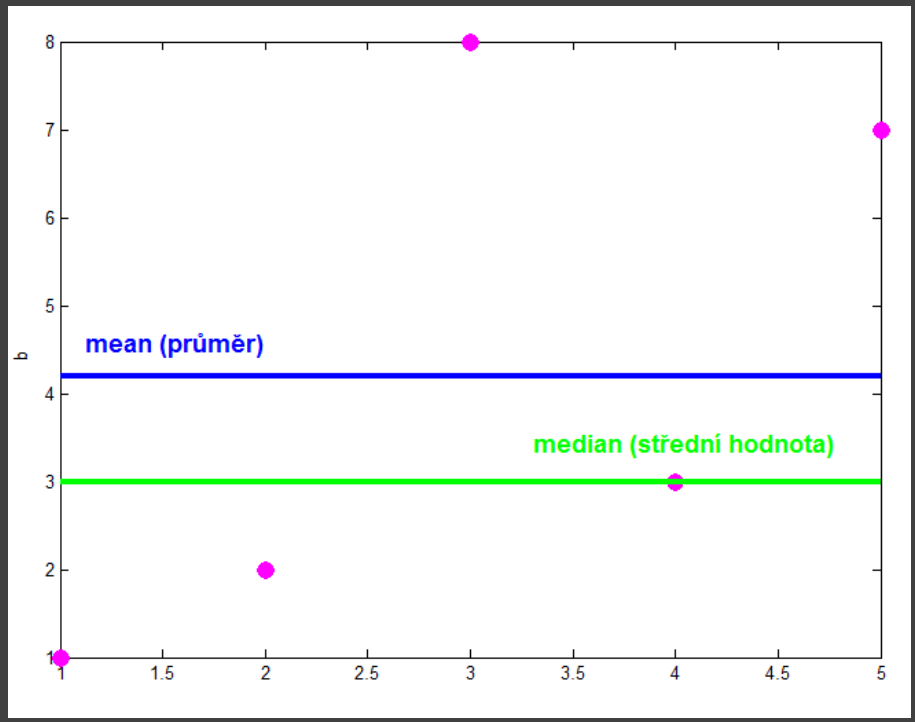


Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,6];
mean(a); median(a);
```

```
b = [1,2,8,3,7];
mean(b); median(b);
```






```
sort(a)
ans =
     1   2   6   7   8   9
```

```
sort(b)
ans =
     1   2   3   7   8
```

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
A =
     1     2     3
     4     5     6
```

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

```
mean(A)
ans =
    2.50    3.50    4.50
```
– průměr z prvků v **sloupcích** matice

```
mean(A,1)
ans =
    2.50    3.50    4.50
```
– totéž (průměr ze **sloupců**)

```
mean(A,2)
ans =
     2
     5
```
– průměr z prvků v **řádcích** matice

```
mean(A.')
ans =
     2     5
```
– průměr z prvků v **řádcích** matice (tj. v sloupcích matice transponované)

```
mean(mean(A))
ans =
     3.50
```
– průměr z celé matice

Funkce pro práci s vektory a maticemi

std(x) – směrodatná odchylka – kvadratický průměr odchylek hodnot od jejich aritmetického průměru

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
a = [9,1,8,2,7,6];
```

```
std(a)
ans = 3.2711
```

```
sqrt(sum((a-mean(a)).^2)./(length(a)-1))
ans = 3.2711
```

Funkce pro práci s vektory a maticemi

std(x) – směrodatná odchylka – kvadratický průměr odchylek hodnot od jejich aritmetického průměru

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
a = [9,1,8,2,7,6];
```

```
std(a)
ans = 3.2711
```

```
sqrt(sum((a-mean(a)).^2)./(length(a)-1))
ans = 3.2711
```

Směrodatná odchylka - ukazuje, jak se od sebe navzájem liší typické případy v souboru zkoumaných čísel. Je-li směrodatná odchylka malá , jsou si prvky většinou navzájem podobné, jestliže je velká, znamená to velké vzájemné odlišnosti prvků.

Funkce pro práci s vektory a maticemi

`g = unique(f)` – vrací stejné hodnoty jako jsou prvky ve vektoru *f*, ale **bez opakování**. *g* je vektor jedinečných hodnot z vektoru *f*. Výsledný vektor *g* je seřazený **vzestupně**.
Např.

```
f = [9,1,3,1,2,7,2,9,1,4,1,0,9,6];
```

```
sort(f)
ans =  0  1  1  1  1  2  2  3  4  6  7  9  9  9
```

```
g = unique(f)
g =
    0  1  2  3  4  6  7  9
```

```
length(f)
ans =  14
```

```
length(g)
ans =  8
```

Funkce pro práci s vektory a maticemi

`g = unique(f)` – vrací stejné hodnoty jako jsou prvky ve vektoru *f*, ale **bez opakování**. *g* je vektor jedinečných hodnot z vektoru *f*. Výsledný vektor *g* je seřazený **vzestupně**.
Např.

```
f = [9,1,3,1,2,7,2,9,1,4,1,0,9,6];
```

```
sort(f)
ans =  0  1  1  1  1  2  2  3  4  6  7  9  9  9
```

```
g = unique(f)
g =
    0  1  2  3  4  6  7  9
```

```
length(f)
ans =  14
```

```
length(g)
ans =  8
```

Funkce pro práci s vektory a maticemi

`g = unique(f)` – vrací stejné hodnoty jako jsou prvky ve vektoru *f*, ale **bez opakování**. *g* je vektor jedinečných hodnot z vektoru *f*. Výsledný vektor *g* je seřazený **vzestupně**.
Např.

```
f = [9,1,3,1,2,7,2,9,1,4,1,0,9,6];
```

```
sort(f)
ans =  0  1  1  1  1  2  2  3  4  6  7  9  9  9
```

```
g = unique(f)
g =
    0  1  2  3  4  6  7  9
```

```
length(f)
ans =  14
```

```
length(g)
ans =  8
```

```
t = 'opakujici_se_znak_ven'          – zadán řetězec
u = unique(t)
u = _aceijknopsuvz                   – řazeno podle abecedy
sort(t)
ans = ___aaceeiijkknnopsuvz          – mezery na začátku
```

Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,3];
```

diff(a) – diference – rozdíly mezi sousedními prvky vektoru **a**

```
ans =
    -8    7   -6    5   -4
```

```
length(a)
ans =
     6
```

```
length(diff(a))
ans =
     5
```

Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,3];
```

diff(a) – diference – rozdíly mezi sousedními prvky vektoru **a**

```
ans =
    -8    7   -6    5   -4
```

```
length(a)
ans =
     6
```

```
length(diff(a))
ans =
     5
```

= 6-1

Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,3];
```

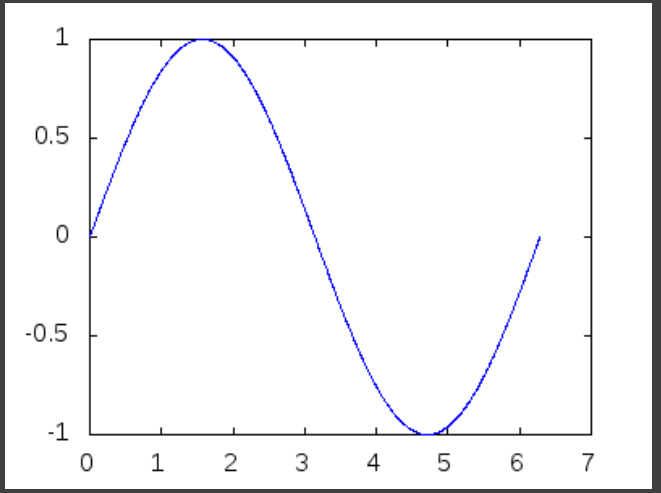
diff(a) – diference – rozdíly mezi sousedními prvky vektoru **a**

```
ans =
    -8    7   -6    5   -4
```

```
length(a)
ans =
     6
```

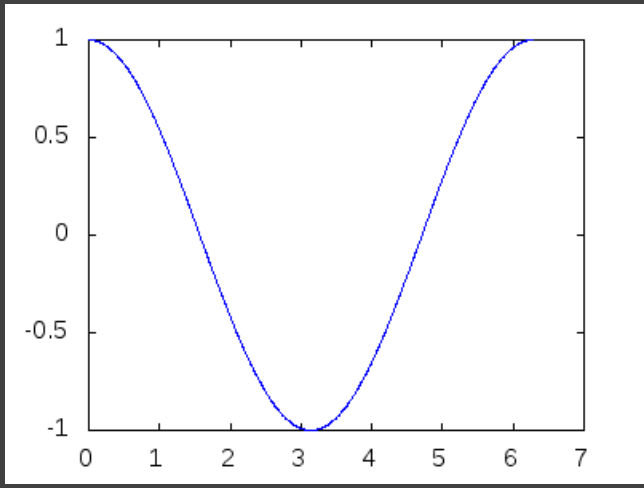
```
length(diff(a))
ans =
     5
```

= 6-1



Lze využít pro numerické derivování:

```
t = linspace(0,2*pi,1000);
y = sin(t);
dydt = diff(y) ./ diff(t);
plot(t,y)
plot(t(1:end-1), dydt)
```



Funkce pro práci s vektory a maticemi

```
a = [9,1,8,2,7,3];
```

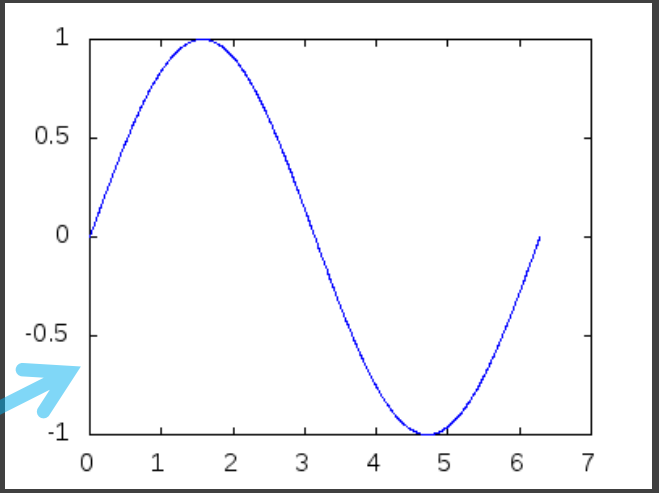
diff(a) – diference – rozdíly mezi sousedními prvky vektoru **a**

```
ans =
    -8    7   -6    5   -4
```

```
length(a)
ans =
     6
```

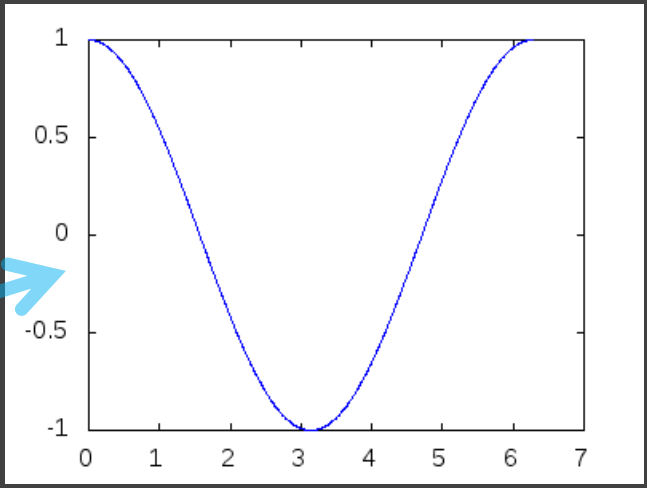
```
length(diff(a))
ans =
     5
```

= 6-1



Lze využít pro numerické derivování:

```
t = linspace(0,2*pi,1000);
y = sin(t);
dydt = diff(y) ./ diff(t);
plot(t,y)
plot(t(1:end-1), dydt)
```



Funkce pro práci s vektory a maticemi

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

```
A.'  – prostá transpozice
ans =
     1     4
     2     5
     3     6
```

```
rot90(A)
ans =
     3     6
     2     5
     1     4
```

– otočení matice o 90° (proti směru hodinových ručiček)

```
rot90(rot90(A))
ans =
     6     5     4
     3     2     1
```

– otočení matice o 180°

```
rot90(rot90(rot90(A)))
ans =
     4     1
     5     2
     6     3
```

– otočení matice o 270°

Funkce pro práci s vektory a maticemi

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

```
A.'  – prostá transpozice
ans =
     1     4
     2     5
     3     6
```

rot90(*matice*,*k*)
– otočení *matice* o *k**90°

```
rot90(A)
ans =
     3     6
     2     5
     1     4
```

– otočení matice o 90° (proti směru hodinových ručiček)

```
rot90(rot90(A))
ans =
     6     5     4
     3     2     1
```

– otočení matice o 180°

```
rot90(rot90(rot90(A)))
ans =
     4     1
     5     2
     6     3
```

– otočení matice o 270°

```
rot90(A,1)
ans =
     3     6
     2     5
     1     4
```

1* 90°

– otočení matice o 90°

```
rot90(A,2)
ans =
     6     5     4
     3     2     1
```

2* 90°

– otočení matice o 180°

```
rot90(A,3)
ans =
     4     1
     5     2
     6     3
```

3* 90°

– otočení matice o 270°

Funkce pro práci s vektory a maticemi

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

```
hlDiagonala = diag(D)
hlDiagonala =
     1
     5
    -9
```

`D.'` – prostá transpozice

```
ans =
     1     4     7
     2     5     8
     3     6    -9
```

```
diag(D.')
ans =
     1
     5
    -9
```

`rot90(D)` – otočení matice o 90°

```
ans =
     3     6    -9
     2     5     8
     1     4     7
```

```
diag(rot90(D))
ans =
     3
     5
     7
```

Funkce pro práci s vektory a maticemi

Testovací funkce vrací na příslušných pozicích **1** (pravda) nebo **0** (nepravda) , např.

```
D =
    1    2    3    4    5
    1    0  Inf   -1  1-1i
  NaN  Inf    0    6  7+2i
    9    8  Inf  Inf  NaN
```

`isfinite(D)` – jsou prvky matice **D** **konečné**?

```
ans =
     1     1     1     1     1
     1     1     0     1     1
     0     0     1     1     1
     1     1     0     0     0
```

`isinf(D)` – jsou prvky matice **D** **nekonečné**?

```
ans =
     0     0     0     0     0
     0     0     1     0     0
     0     1     0     0     0
     0     0     1     1     0
```

`isnan(D)` – jsou prvky **NaN** (neplatná numerická hodnota)?

```
ans =
     0     0     0     0     0
     0     0     0     0     0
     1     0     0     0     0
     0     0     0     0     1
```

Funkce pro práci s vektory a maticemi

Testovací funkce vrací na příslušných pozicích **1** (pravda) nebo **0** (nepravda) , např.

```
F =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
```

isprime(F) – jsou prvky matice **F** **prvočísla**?

```
ans =
     1     0     0     0     0
     1     1     1     0     0
     0     0     1     0     0
     0     0     1     0     1
```

```
G =
   6+8i    -1          4-9i
   0+1i     3 - 2i     5
   2        7 + 3i     0
```

isreal(G) – jsou prvky matice **G** reálné, **nemají imaginární část**?

```
ans =   0
```

~isreal(G) – nejsou prvky matice **G** reálné, **mají** prvky matice **G** **imaginární část**?

```
ans =   1
```

```
isreal(F)
ans =   1
```

Funkce pro práci s vektory a maticemi

`find(H)` – nalezne polohu nenulových hodnot v matici **H**

`[r,s] = find(H)` – nalezne indexy nenulových prvků v **H**

```
H =
     9     0     2     0     7
     8     0     0     0     0
     0     5     1     0     2
     4     0     0     6     0
```

```
find(H)
ans =
     1
     2
     4
     7
     9
    11
    16
    17
    19
```

```
     1     5     9    13    17
     2     6    10    14    18
     3     7    11    15    19
     4     8    12    16    20
```

– indexy, jsou-li prvky indexovány lineárně **po sloupcích**

```
[r,s,h] = find(H)
r =       s =       h =
     1         1         9
     2         1         8
     4         1         4
     3         2         5
     1         3         2
     3         3         1
     4         4         6
     1         5         7
     3         5         2
```

– indexy (**řádek** a **sloupec**) v matici **H** a **hodnoty** prvků

Funkce pro práci s vektory a maticemi

Příklad: prvky matice větší než 20

```
F =
    17   24    1    8   15
    23    5    7   14   16
     4    6   13   20   22
    10   12   19   21    3
```

```
F>20
ans =
     0     1     0     0     0
     1     0     0     0     0
     0     0     0     0     1
     0     0     0     1     0
```

`find(F>20)` – nalezne pozice prvků větších než 20 v matici **F**, protože na těchto pozicích je v pravdivostní matici hodnota **1**

```
find(F>20)
ans =
     2
     5
    16
    19
```

```
     1     5     9    13    17
     2     6    10    14    18
     3     7    11    15    19
     4     8    12    16    20
```

– indexy, jsou-li prvky indexovány **po sloupcích**

```
[r,s,h] = find(F>20)
r =         s =         h =
     2           1           1
     1           2           1
     4           4           1
     3           5           1
```

– indexy prvků (**řádek** a **sloupec**)

Funkce pro práci s vektory a maticemi

Příklad: Víme, že pomocí **`max()`** lze nalézt maximum a polohu (index) maxima ve vektoru *c*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Pokud je maxim více, je vrácen index toho prvního.

Nalezení polohy (indexu) všech maximálních prvků ve vektoru *c* pomocí funkce **`find()`**:

```
c == max(c)
ans =
     0   0   0   0   0   1   0   0   1   0   0   0
```

– nalezne pozice všech prvků ve vektoru *c*, které se rovnají maximu, protože na těchto pozicích je v pravdivostní matici hodnota 1

Funkce pro práci s vektory a maticemi

Příklad: Víme, že pomocí **`max()`** lze nalézt maximum a polohu (index) maxima ve vektoru *c*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Pokud je maxim více, je vrácen index toho prvního.

Nalezení polohy (indexu) všech maximálních prvků ve vektoru *c* pomocí funkce **`find()`**:

```
c == max(c)
ans =
     0   0   0   0   0   1   0   0   1   0   0   0
```

– nalezne pozice všech prvků ve vektoru *c*, které se rovnají maximu, protože na těchto pozicích je v pravdivostní matici hodnota 1

```
find(c == max(c))
ans =
     6     9
```

Funkce pro práci s vektory a maticemi

Příklad: Víme, že pomocí **max()** lze nalézt maximum a polohu (index) maxima ve vektoru *c*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Pokud je maxim více, je vrácen index toho prvního.

Nalezení polohy (indexu) všech maximálních prvků ve vektoru *c* pomocí funkce **find()**:

```
c == max(c)
ans =
     0   0   0   0   0   1   0   0   1   0   0   0
```

```
[r,s,h]= find(c == max(c))
r =
     1     1      - číslo řádku
s =
     6     9      - číslo sloupce
h =
     1     1      - hodnota 1 v pravdivostní matici
```

– nalezne pozice všech prvků ve vektoru *c*, které se rovnají maximu, protože na těchto pozicích je v pravdivostní matici hodnota 1

```
find(c == max(c))
ans =
     6     9
```

Funkce pro práci s vektory a maticemi

Příklad: Víme, že pomocí **max()** lze nalézt maximum a polohu (index) maxima ve vektoru *c*.

```
c = [-2,-4,1,-3,4,5,0,-4,5,-1,2,3];
```

```
[mc,pc] = max(c)
    mc =  5
    pc =  6
```

Pokud je maxim více, je vrácen index toho prvního.

Nalezení polohy (indexu) všech maximálních prvků ve vektoru *c* pomocí funkce **find()**:

```
c == max(c)
ans =
→ 0  0  0  0  0  1  0  0  1  0  0  0
```

```
[r,s,h]= find(c == max(c))
r =
     1     1      - číslo řádku
s =
     6     9      - číslo sloupce
h =
     1     1      - hodnota 1 v pravdivostní matici
```

– nalezne pozice všech prvků ve vektoru *c*, které se rovnají maximu, protože na těchto pozicích je v pravdivostní matici hodnota 1

```
find(c == max(c))
ans =
     6     9
```

Funkce pro práci s vektory a maticemi

`any(x)` – je-li některý prvek vektoru *x* nenulový, vrací hodnotu 1, jinak vrací 0.
`all(x)` – jsou-li všechny prvky vektoru *x* nenulové, vrací hodnotu 1, jinak vrací 0.

```
k = [9,1,8,2];   l = [3,0,4,0,6];
m = [0,0,0,0];   n = [5,NaN,7,Inf];
```

```
any(k)
ans =
     1
```

```
any(l)
ans =
     1
```

```
all(k)
ans =
     1
```

```
all(l)
ans =
     0
```

```
any(m)
ans =
     0
```

```
any(n)
ans =
     1
```

```
all(m)
ans =
     0
```

```
all(n)
ans =
     1
```

`any()`, **`all()`** ignorují položky **NaN** (neplatné numerické hodnoty).

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
K =   9    0    2    0    7
      8    0    0    0    0
      0    5    1    0    2
      4    0    0    0    0
      0    0    0    0    0
```

```
any(K)
ans =
     1     1     1     0     1
```

– je-li **ve sloupci** matice **K** **nějaký** (alespoň jeden) **prvek nenulový**, pak vrátí 1 (pravda), jinak vrátí 0 (nepravda) – pro každý **sloupec**

```
any(K,1)
ans =
     1     1     1     0     1
```

– totéž

```
any(K,2)
ans =
     1
     1
     1
     1
     0
```

– je-li **v řádku** matice **K** **nějaký** (alespoň jeden) **prvek nenulový**, vrátí 1 (pravda), jinak vrátí 0 (nepravda)

```
any(K, 3)
ans =
     1     0     1     0     1
     1     0     0     0     0
     0     1     1     0     1
     1     0     0     0     0
     0     0     0     0     0
```

– je-li některý **prvek** matice **K** **nenulový**, vrátí 1, jinak vrátí 0.

Funkce pro práci s vektory a maticemi

Podobně pro matice, např.

```
L =   4    7    6    5    0
      0    5   -8    2    1
      1   -2    0    9    3
      6    5    2    1   -3
      0    4    0    6    0
```

```
all(L)
ans =
     0     1     0     1     0
```

– jsou-li **ve sloupci** matice **L** **všechny prvky nenulové**, pak vrátí 1 (pravda), jinak vrátí 0 (nepravda) – pro každý **sloupec**

```
all(L,1)  – totéž
ans =
     0     1     0     1     0
```

```
all(L,2)
ans =
     0
     0
     0
     1
     0
```

– jsou-li **v řádku** matice **L** **všechny prvky nenulové**, vrátí 1 (pravda), jinak vrátí 0 (nepravda)

```
all(L,3)
ans =
     1     1     1     1     0
     0     1     1     1     1
     1     1     0     1     1
     1     1     1     1     1
     0     1     0     1     0
```

– jsou-li **prvky** matice **K** **nenulové**, vrátí 1, jinak vrátí 0.

Funkce pro práci s vektory a maticemi

Pro uvedené matice:

```
K =
     9     0     2     0     7
     8     0     0     0     0
     0     5     1     0     2
     4     0     0     0     0
     0     0     0     0     0
```

```
L =
     4     7     6     5     0
     0     5    -8     2     1
     1    -2     0     9     3
     6     5     2     1    -3
     0     4     0     6     0
```

```
any(K)
ans =
     1     1     1     0     1
```

```
any(L)
ans =
     1     1     1     1     1
```

```
all(K)
ans =
     0     0     0     0     0
```

```
all(L)
ans =
     0     1     0     1     0
```

Funkce pro práci s vektory a maticemi

`randperm(n)` – náhodná permutace čísel od **1** do ***n*** (tj. čísla **1** až ***n*** v náhodném pořadí)

```
randperm(6)
ans =   6    3    5    1    2    4
```

```
randperm(3)
ans = 3    1    2
```

Funkce pro práci s vektory a maticemi

`randperm(n)` – náhodná permutace čísel od **1** do ***n*** (tj. čísla **1** až ***n*** v náhodném pořadí)

```
randperm(6)
ans =   6    3    5    1    2    4
```

```
randperm(3)
ans = 3    1    2
```

Příklad užití – náhodné zpřeházení položek ve vektoru:

```
x = 0:2:14
x = 0    2    4    6    8   10   12   14
```

```
length(x)
ans = 8
```

```
z = randperm(length(x))
z = 4   5   2   7   3   6   1   8
```

```
x(z)
ans =
     6    8    2   12    4   10    0   14
```

Funkce pro práci s vektory a maticemi

`randperm(n)` – náhodná permutace čísel od **1** do ***n*** (tj. čísla **1** až ***n*** v náhodném pořadí)

```
randperm(6)
ans =   6    3    5    1    2    4
```

```
randperm(3)
ans = 3    1    2
```

Příklad užití – náhodné zpřeházení položek ve vektoru:

```
x = 0:2:14
x = 0    2    4    6    8   10   12   14
```

```
length(x)
ans = 8
```

```
z = randperm(length(x))
z = 4   5   2   7   3   6   1   8
```

```
x(z)
ans =
     6    8    2   12    4   10    0   14
```

výběr prvků se zadanými indexy
`x([4,5,2,7,3,6,1,8])`

Funkce pro práci s vektory a maticemi

`randperm(n)` – náhodná permutace čísel od **1** do ***n*** (tj. čísla **1** až ***n*** v náhodném pořadí)

```
randperm(6)
ans =   6    3    5    1    2    4
```

```
randperm(3)
ans = 3    1    2
```

Příklad užití – náhodné zpřeházení položek ve vektoru:

```
x = 0:2:14
x = 0    2    4    6    8   10   12   14
```

```
length(x)
ans = 8
```

```
z = randperm(length(x))
z = 4    5    2    7    3    6    1    8
```

```
x(z)
ans =
     6    8    2   12    4   10    0   14
```

výběr prvků se zadanými indexy

`x([4,5,2,7,3,6,1,8])`

Funkce pro práci s vektory a maticemi

`randperm(n)` – náhodná permutace čísel od **1** do ***n*** (tj. čísla **1** až ***n*** v náhodném pořadí)

```
randperm(6)
ans =   6    3    5    1    2    4
```

```
randperm(3)
ans = 3    1    2
```

Příklad užití – náhodné zpřeházení položek ve vektoru:

```
x = 0:2:14
x = 0    2    4    6    8   10   12   14
```

```
length(x)
ans = 8
```

```
z = randperm(length(x))
z = 4    5    2    7    3    6    1    8
```

```
x(z)
ans =
     6    8    2   12    4   10    0   14
```

výběr prvků se zadanými indexy
`x([4,5,2,7,3,6,1,8])`

Lze psát i takto pomocí vnořených funkcí: `x(randperm(length(x)))`

Funkce pro práci s vektory a maticemi

Práce s časem

clock – aktuální datum a čas jako data vektoru:

```
                [Rok Měsíc Den Hodina Minuta Sekundy]
ans =
  1.0e+003 *
     2.0170    0.0100    0.0310    0.0140    0.0480    0.0253
```

```
format short g
cas = clock
cas =
      2017          10          31          14          48      25.346
```

date – aktuální datum jako řetězec

```
ans =
      31-Oct-2017
```

```
whos
  Name        Size                    Bytes  Class     Attributes
  ans         1x11                       22  char
  cas         1x6                        48  double
```

Práce s komplexními čísly

Např. komplexní číslo `c = 6+3i` (lze zapsat i jako `c = 6+3j`)

`isreal(c)` – test, je-li *c* reálné číslo (nepravda => vyjde 0)

```
ans  =  0
```

`real(c)` – vypíše reálnou část komplexního čísla *c*

```
ans  =  6
```

`imag(c)` – vypíše imaginární část komplexního čísla *c*

```
ans  =  3
```

`conj(c)` – vypíše číslo komplexně sdružené k číslu *c* (u imaginární části s opačným znaménkem)

```
ans  =  6.00 - 3.00i
```

`complex(2,-7)` – vytvoření komplexního čísla, výsledek je složen z reálné části 2 a imaginární části -7

```
ans  =  2.00 - 7.00i
```

Práce s komplexními čísly

```
c = 6+3i;
```

abs(c) – vypíše **absolutní hodnotu** (velikost, modul) komplexního čísla $\sqrt{(\mathrm{Re}(c))^2 + (\mathrm{Im}(c))^2}$

```
ans  =  6.7082
```

Pozn. stejný výsledek získáme také:

```
sqrt(real(c).^2+imag(c).^2)
ans  =  6.7082
```

angle(c) – vypíše **úhel** (fázi, argument) komplexního čísla **v radiánech** $\varphi = \arctan \frac{\mathrm{Im}(c)}{\mathrm{Re}(c)}$

```
ans  =  0.4636
```

angle(c)*180/pi – převod úhlu na stupně

```
ans  =  26.5651
```

Práce s komplexními čísly

Zobrazení komplexních čísel

`plot(c)` – pokud *c* je komplexní číslo, `plot(c)` je ekvivalentní k zobrazení grafu `plot(real(c),imag(c))`, tj. na *x*-ové ose je reálná část a na *y*-ové ose je imaginární část

`compass(c)` – pokud *c* je komplexní číslo, je zobrazen vektor se složkami `real(c)`, `imag(c)` ve formě šipky vycházející z počátku

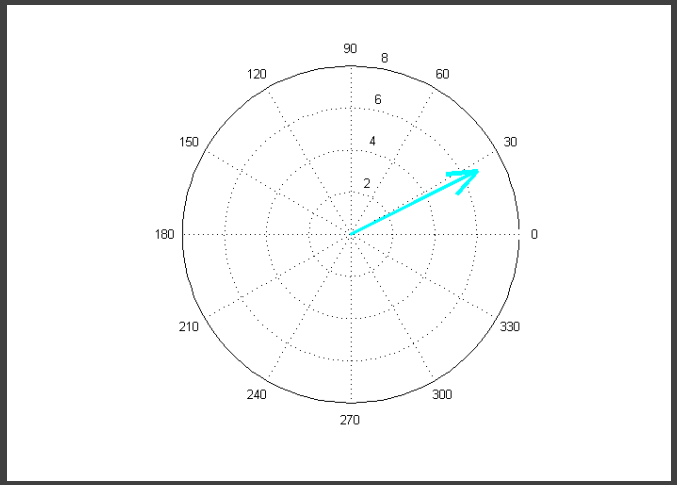
`polar(uhel, velikost)` – graf v polárních souřadnicích, zadáváme úhel v radiánech `angle(c)` a velikost (modul) komplexního čísla `abs(c)`, tedy `polar(angle(c), abs(c))`

Práce s komplexními čísly

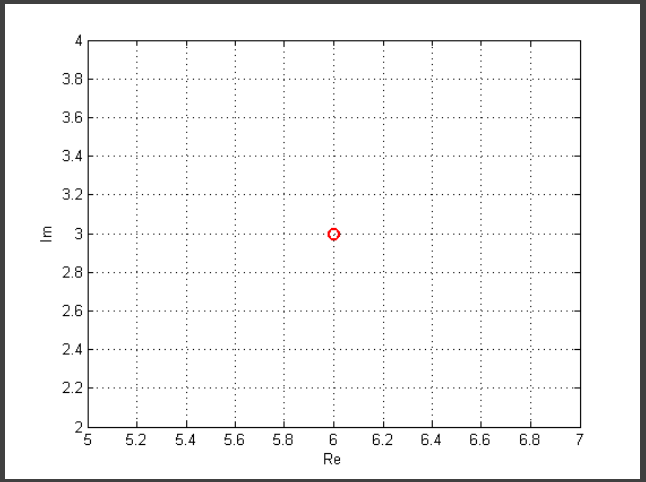
Příklad:
zobrazení čísla *c*

```
c = 6+3i;
```

```
compass(c,'c')
```




```
plot(c,'ro')
grid
xlabel('Re')
ylabel('Im')
```




```
u = angle(c);
v = abs(c);
polar(u,v)
```