ÚVOD DO SYSTÉMU MATLAB

MATLAB (**MAT**rix **LAB**oratory) je interaktívny softvérový systém pre numerické výpočty a grafiku. Ako názov napovedá, MATLAB bol vytvorený predovšetkým pre počítanie s maticami: riešenie systému lineárnych rovníc, násobenie matíc a pod. Navyše disponuje rôznorodými grafickými možnosťami a môže sa rozširovať o programy napísané v jeho vlastnom programovacom jazyku.

V MATLABe sa pracuje prostredníctvom MATLAB jazyka. Najjednoduchším spôsobom, ako zadať a vykonať nejaký príkaz, je napísať ho do príkazového okna MATLABu (Command Window) za znak ">>". Príkaz sa okamžite interpretuje a vykoná. Na editovanie postupnosti príkazov sa používa MATLAB editor. Postupnosti príkazov alebo aj celé programy sa ukladajú v textových súboroch ako skripty alebo ako funkcie, čím sa rozšíri knižnica dostupných príkazov.

V nasledujúcich častiach nájdete základný opis najpoužívanejších úkonov v MATLABe s množstvom užitočných príkladov. Najlepším spôsobom, ako sa naučiť používať softvér MATLAB, je naštartovať ho a podľa uvedených príkladov okamžite skúšať a experimentovať.

Zadávanie vektorov a matíc

Základným dátovým typom MATLABu je *n*-dimenzionálne pole čísel s dvojitou presnosťou (double precision). Nové dátové typy zahŕňajú štruktúry, triedy a "bunkové polia", ktoré môžu obsahovať aj rôzne dátové typy.

Nasledujúce príkazy ukazujú, ako sa zadávajú čísla, vektory a matice a ako sa priraďujú premenným.

```
>> a = 2  //scalar
```

Keď stlačíte enter, uvidíte

```
a =
     2
```

```
>> x = [1;2;3] //Vector
```

Stlačte enter.

```
x =
     1
     2
     3
```

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

Stlačte enter.

```
A =
```

```
     1     2     3
     4     5     6
     7     8     0
```

Všimnite si, že riadky matíc sú od seba oddelené bodkočiarkou, zatiaľ čo jednotlivé prvky v riadkoch sa oddeľujú medzerou alebo čiarkou.

Užitočným príkazom je príkaz "whos", ktorý zobrazí názvy všetkých definovaných premenných a ich dátový typ.

```
>> whos
  Name      Size          Bytes  Class

  A         3x3              72  double array
  a         1x1               8  double array
  x         3x1              24  double array

Grand total is 13 elements using 104 bytes
```

Všimnite si, že každá z týchto troch premenných je typu pole. Jeho rozmer určuje presný typ. Skalár "a" je pole typu 1x1, vektor "x" je pole typu 3x1 a matica "A" je pole typu 3x3. (pozri "size" pri každej premennej).

Jednou z možností ako zadať *n*-dimenzionálne pole (*n*>2) je použiť príkaz `cat` a napojiť tak za sebou dve alebo viac (*n*-1)-dimenzionálnych polí. Nasledujúci príklad spája dve polia typu 3x2 a vytvára nové pole ‘C’ typu 3x2x2.

```
>> C = cat(3,[1,2;3,4;5,6],[7,8;9,10;11,12])

C(:,:,1) =
     1     2
     3     4
     5     6

C(:,:,2) =
     7     8
     9    10
    11    12

>> whos

  Name      Size          Bytes  Class

  A         3x3              72  double array
  C         3x2x2            96  double array
  a         1x1               8  double array
  x         3x1              24  double array

Grand total is 25 elements using 200 bytes
```

Všimnite si, že argument "3" v príkaze `cat` indikuje, že napojenie sa má udiať v 3. rozmere. Ak *D* and *E* sú polia typu $kxmxn$, potom príkaz

```
>> cat(4,D,E)
```

vytvorí pole typu $kxmxnx2$ (vyskúšaj).

Prvky matice môžu byť aj komplexné čísla. Imaginárnu jednotku $i = \sqrt{-1}$ zastupuje jedna zo zabudovaných premenných `i` alebo `j`.

```
>> sqrt(-1)

ans =
        0 + 1.0000i
```

Na tomto príklade vidíme, ako MATLAB zobrazuje komplexné čísla a tiež, že druhá odmocnina je zabudovanou funkciou.
Výsledok posledného výpočtu, ktorý sme nepriradili žiadnej premennej, bol priradený automaticky do premennej `ans`. V ďalších výpočtoch môžeme s premennou `ans` pracovať ako s akoukoľvek inou premennou. Tu je príklad:

```
>> 100^2-4*2*3 //Slač enter

ans =
        9976

>> sqrt(ans) //Stlač enter

ans =
   99.8799

>> (-100+ans)/4 //Stlač enter

ans =
    -0.0300
```

Zabudovanou konštantou, ktorá sa často používa je konštanta π.

```
>> pi  //Press enter

ans =
    3.1416
```

Ako sa dá očakávať, pre skaláry fungujú aritmetické operátory. Tiež sú preddefinované bežné funkcie ako sínus, kosínus, tangens, exponenciálna funkcia a logaritmus. Napríklad

```
>> cos(.5)^2+sin(.5)^2 //Stač enter

ans =
     1

>> exp(1)  // Stač enter

ans =
    2.7183

>> log(ans)  // Stač enter

ans =
     1
```

Ak potrebujete poradiť s niektorou funkciou alebo príkazom, pomocou príkazu `help <názov príkazu>` sa napojíte na rozsiahly on-line systém pomoci. Odpoveď dostanete v angličtine. Napríklad

```
>> help ans

 ANS     The most recent answer.
         ANS is the variable created automatically when expressions
         are not assigned to anything else. ANSwer.

>> help pi

 PI      3.1415926535897....

 PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897....
```

Ak zadáte `help help`, dostanete odpoveď, ktorá vysvetľuje ako pracuje systém pomoci a tiež niekoľko odpovedajúcich príkazov. Samotné `help` vráti zoznam tém, ktoré systém pomoci obsahuje. Keď sa pozrieme na výstup, medzi témami sa nachádza aj téma "elfun-elementary math functions". Po zadaní `help elfun` dostaneme zoznam dostupných matematických funkcií.

Aritmetické operácie s maticami

V MATLABe štandartné aritmetické operácie (súčet, rozdiel a násobenie) fungujú pre matice, vektory a skaláry. Navyše, MATLAB definuje pojem "maticové delenie" a "vektorové operácie". Všetky vektorové operácie (súčet, rozdiel a násobenie skalárom – pozri príklady) sa môžu aplikovať na akékoľvek *n*-rozmerné polia pre $n \geq 1$, zatiaľ čo násobenie a delenie je definované iba pre matice a vektory s $n \leq 2$.

Štandardné operácie

Ak `A` a `B` sú polia, potom MATLAB vypočíta `A+B` a `A-B` *ak tieto operácie sú pre dané matice definované*. Všimnite si nasledujúce príkazy:

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

>> whos

  Name      Size          Bytes  Class

  A         3x3              72  double array
  B         3x3              72  double array
  C         3x2              48  double array

Grand total is 24 elements using 192 bytes

>> A+B
ans =
     2     3     4
     6     7     8
    10    11    12
```

Ale, ak zadáte

```
>> A+C
```

MATLAB dá chybové hlásenie, kde vás upozorní, že rozmery matice musia byť rovnaké.

```
??? Error using ==> + because Matrix dimensions must agree.
```

Násobenie matíc:

```
>> A*C

ans =
    22    28
    49    64
    76   100

>> C*A

??? Error using ==> * because Matrix dimensions must agree.
```

Ak `A` je štvorcová matica a `m` je kladné celé číslo, potom `A^m` je súčinom `m` činiteľov `A`.

Pre násobenie viacrozmerných polí s viac ako dvomi dimenziami nie je definovaná žiadna operácia.

```
>> C = cat(3,[1 2;3 4],[5 6;7 8])

C(:,:,1) =
     1     2
     3     4

C(:,:,2) =
     5     6
     7     8

>> D = [1;2]

D =
     1
     2

>> whos
  Name      Size          Bytes  Class

  C         2x2x2            64  double array
  D         2x1              16  double array

Grand total is 10 elements using 80 bytes

>> C*D
??? Error using ==> *

No functional support for matrix inputs.
```

Z tohto dôvodu exponenciálny operátor `^` funguje iba pre štvorcové 2-rozmerné polia (matice).

Riešenie maticových rovníc pomocou "delenia matíc"

Ak A je štvorcová nesingulárna matica, potom riešenie rovnice $Ax = b$ je $x = A^{-1} b$. V MATLABe túto operáciu zapíšeme znakom \ (opačná lomka).

```
>> A = rand(3,3)

A =
    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    0.3835    0.0346

>> b = rand(3,1)

b =
    0.0535
    0.5297
    0.6711

>> x = A\b

x =
 -159.3380
  314.8625
 -344.5078
```

Poznámka: Zabudovaná funkcia `rand` vytvorí maticu z prvkov s normálnym rozdelením na intervale $\langle 0,1 \rangle$ (viac informácií sa dozviete po zadaní `help rand`).

Zápis `A\b` je ekvivalentný matematickému zápisu $A^{-1} b$ (násobenie b zľava inverznou maticou A^{-1}). Avšak MATLAB inverznú maticu nepočíta, lineárny systém rieši priamo. Ak znamienko `\` použijeme pre matice, ktoré nie sú štvorcové, systém sa rieši v zmysle metódy najmenších štvorcov. Viac pozri `help slash`.

Samozrejme, tak ako pri ostatných aritmetických operáciách, matice musia mať vhodný rozmer. Deliaci operátor nie je definovaný pre n-dimenzionálne polia s n>2.

"Vektorové" funkcie and operátory

MATLAB má na tvorbu špeciálnych matíc množstvo príkazov. Nasledujúci príkaz vytvorí riadkový vektor, ktorého členy tvoria rastúcu aritmetickú postupnosť.

```
>> t = 1:5
t =
     1     2     3     4     5
```

Krok nemusí byť celočíselný.

```
>> x = 0:.1:1

x =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

Záporný krok je tiež dovolený.

Príkaz `linspace` dá podobný výsledok. Vytvorí vektor, s lineárne usporiadanými členmi. Konkrétne príkaz `linspace(a,b,n)` vytvorí vektor s n členmi: $a, a+\frac{b-a}{n-1}, a+\frac{2(b-a)}{n-1}, ..., b$.

```
>> linspace(0,1,11)

ans =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

Podobný príkaz `logspace` vytvorí vektor hodnôt s logaritmickým delením.

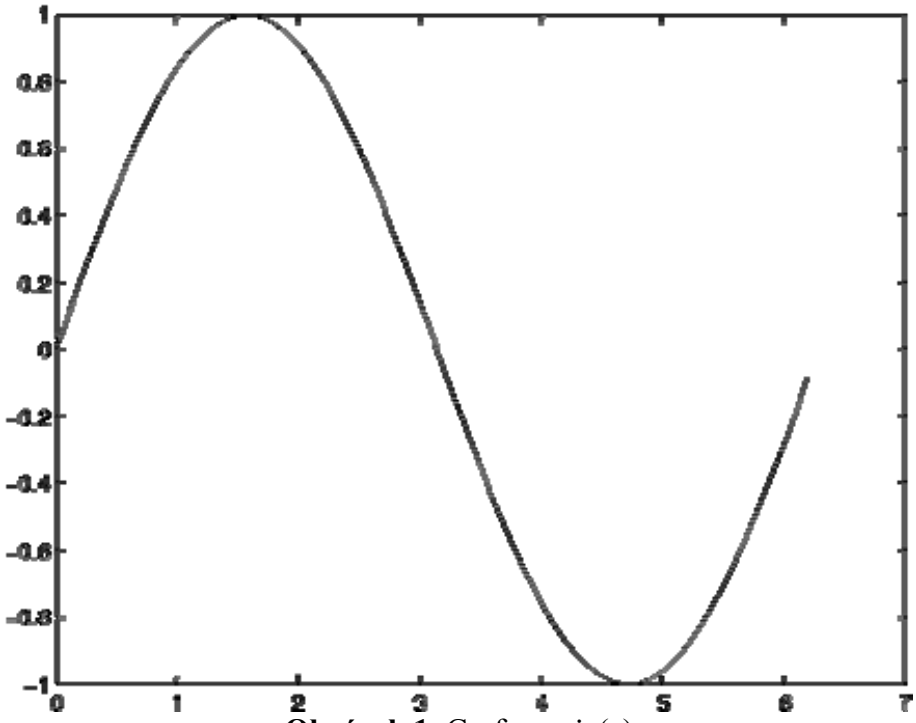
```
>> logspace(0,1,11)

ans =
  Columns 1 through 7
    1.0000    1.2589    1.5849    1.9953    2.5119    3.1623    3.9811
  Columns 8 through 11
    5.0119    6.3096    7.9433   10.0000
```

Viac informácií sa dozviete pomocou `help logspace`.

Vektor s lineárne usporiadanými členmi možno považovať za špecifikáciu jednorozmernej mriežky. To sa využíva pri kreslení grafov. Na nakreslenie grafu $y = f(x)$ vytvoríme mriežku v smere osi x, potom vektor odpovedajúcich hodnôt y, nakreslíme body a spojíme ich úsečkami.

Zabudované funkcie Matlabu sú *vektorizované*. To znamená, že ak zabudovanú funkciu, napríklad sínus, aplikujeme na jednorozmerné pole, výsledkom je nové pole rovnakého typu a veľkosti s hodnotami funkcie v bodoch zadaného poľa. Kreslenie grafov zabudovaných funkcií sa zadáva veľmi jednoducho. Napríklad (pozri obr. 1)

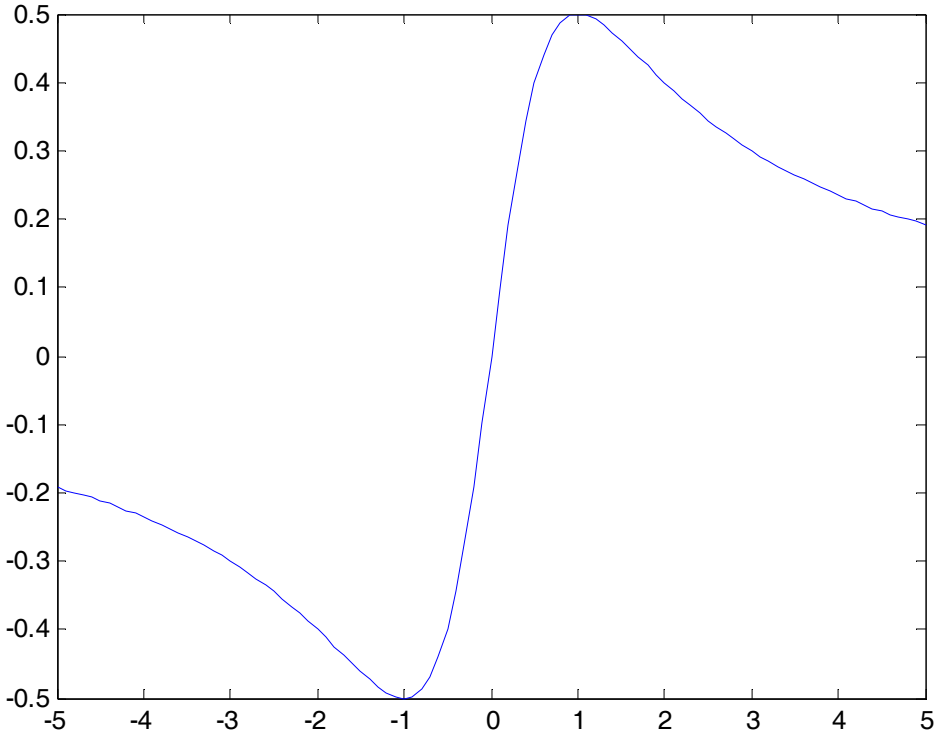
```
>> x = (0:.1:2*pi);
>> y = sin(x);
>> plot(x,y)
```



Obrázok 1. Graf $y = sin(x)$

Aritmetické operátory MATLABu sú tiež *vektorizované.* Označujú sa podobne ako bežné operátory, len sa pridáva "**.**". Napríklad, pri kreslení grafu $y = \frac{x}{1+x^2}$ zadáme

```
>> x = (-5:.1:5);
>> y = x./(1+x.^2);
>> plot(x,y)
```



Obrázok 2. Graf $\mathbf{y = x / (1 + x^2)}$

Zápis `x.^2` priradí každému členu `x` jeho druhú mocninu a zápis `x./z` predelí každý člen poľa `x` odpovedajúcim členom poľa `z`. Súčet a rozdiel majú z definície vlastnosť dedenia na členy poľa, a tak operátory ".+" or ".-" nemajú opodstatnenie a sa nepoužívajú.

Všimnite si rozdiel medzi `A^2` a `A.^2`. Prvá operácia sa vzťahuje na štvorcovú maticu `A`, druhá operácia sa vzťahuje na akékoľvek *n*-dimenzionálne pole `A`.

Niektoré ďalšie príkazy

Dôležitým operátorom v MATLABe sú jednoduché úvodzovky " ' ", ktorými sa označuje transpozícia a konjugácia.

```
>> A = [1 2;3 4]

A =
     1     2
     3     4

>> A'

ans =
     1     3
     2     4

>> B = A + i*.5*A
```

```
B =
   1.0000 + 0.5000i   2.0000 + 1.0000i
   3.0000 + 1.5000i   4.0000 + 2.0000i

>> B'
ans =
   1.0000 - 0.5000i   3.0000 - 1.5000i
   2.0000 - 1.0000i   4.0000 - 2.0000i
```

V špeciálnych prípadoch, keď požadujeme iba transpozíciu, použijeme operátor "**.'**" .

```
>> B.'

ans =
   1.0000 + 0.5000i   3.0000 + 1.5000i
   2.0000 + 1.0000i   4.0000 + 2.0000i
```

(Všimnite si, že operátory "`'`" a "`.'`" sú pre matice s reálnymi prvkami ekvivalentné.)

Nasledujúce príkazy sa používajú veľmi často. Viac informácií sa dozviete zo systému pomoci online.

Vytváranie matíc

- `zeros(m,n)` vytvorí nulovú maticu typu *mxn*
- `ones(m,n)` vytvorí maticu jednotiek typu *mxn*
- `eye(n)` vytvorí jednotkovú maticu typu *nxn*
- `diag(v)` (`v` je *n*-rozmerný vektor) vytvorí diagonálnu maticu typu *nxn* s `v` na diagonále

Napríklad

```
>> ones(3,4)

ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> v=[-1 2 3.5]

v =
   -1.0000    2.0000    3.5000

>> diag (v)

ans =
   -1.0000         0         0
         0    2.0000         0
         0         0    3.5000
```

V príkazoch `zeros` a `ones` môžeme zadať akýkoľvek počet argumentov. *k* argumentov vytvorí *k*-rozmerné pole žiadaného typu.

Formátovanie písomného a grafického výstupu

Nasledujúce príkazy formátujú písomný vzhľad výstupu.

- `format` definuje formát výstupu

```
>> format short, pi
ans =
    3.1416
>> format short e, pi
ans =
      3.1416e+000
>> format long, pi
ans =
      3.14159265358979
>> format long e, pi
ans =
    3.141592653589793e+000
```

Príkaz `format compact` vynechá vo výstupe prázdne riadky (všetky výstupy sú tu vo formáte compact format).

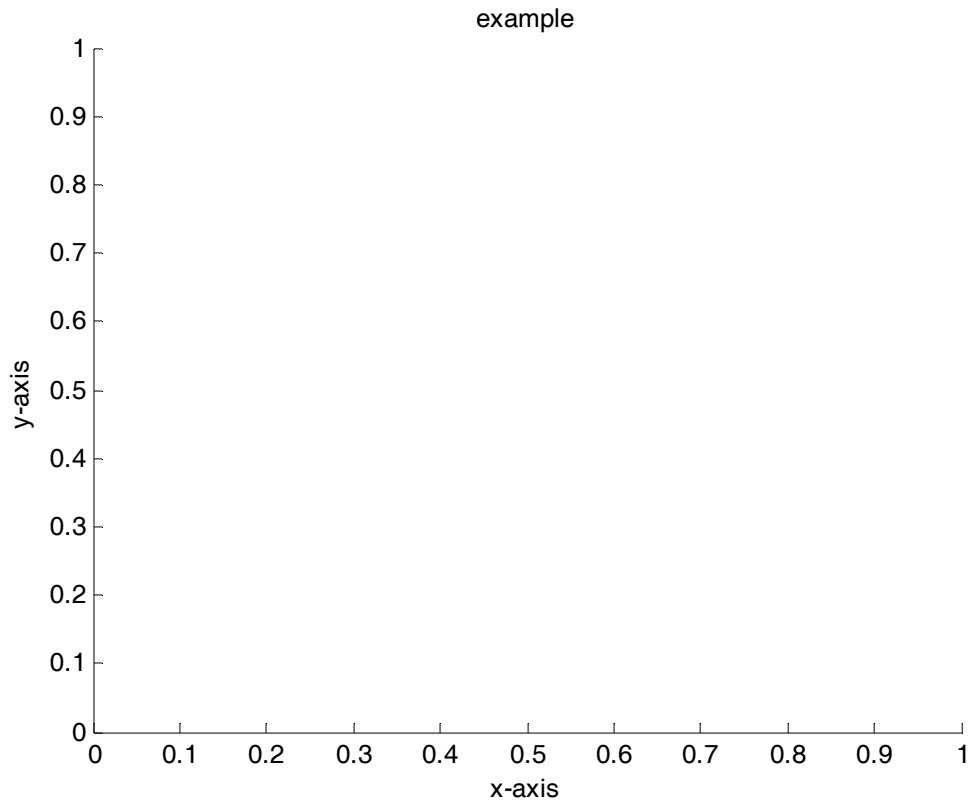
Príkaz `format loose` vráti prázdne riadky späť.

```
>> format loose, pi

ans =
    3.1416
```

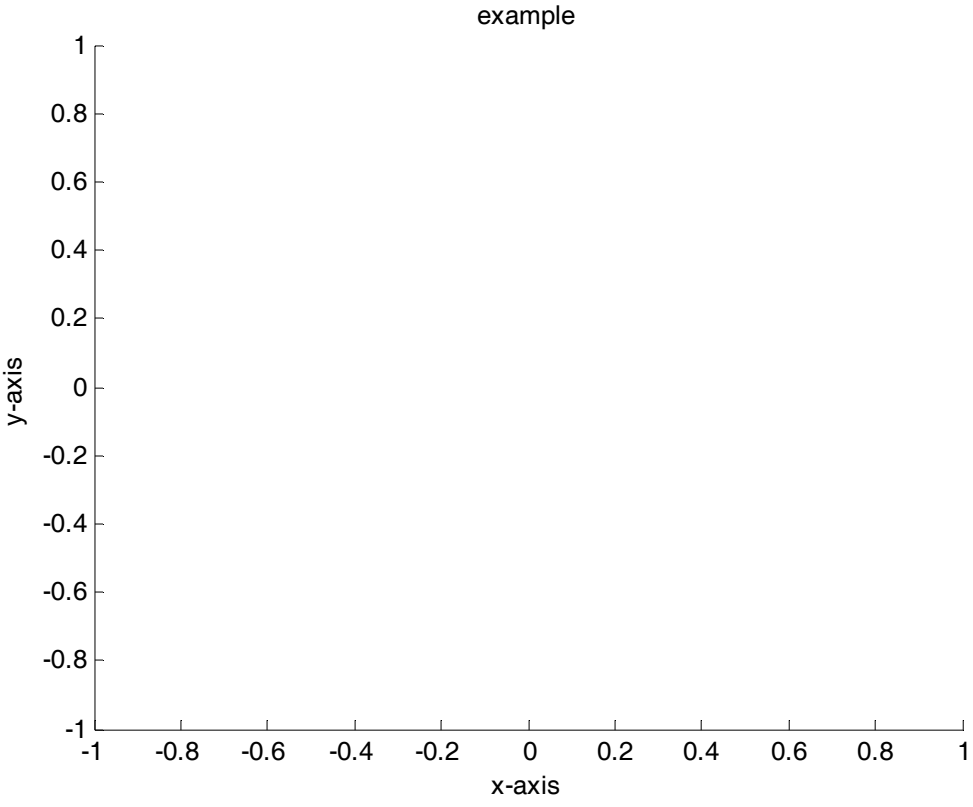
- `xlabel('string'), ylabel('string')` pomenovávajú v príslušnom grafe horizontálnu, resp. vertikálnu os
- `title('string')` pridá do grafu názov
- `axis([a b c d])` zmení okno grafu na $a \le x \le b, c \le y \le d$;
- `grid` pridá do grafu pravouhlú mriežku
- `hold on` zachová na obrazovke aktuálny graf. Nasledujúci graf sa zobrazí spolu s ním v jednom spoločnom obrázku.
- `hold off` zruší zachovanie aktuálneho grafu. Pred zobrazením ďalšieho grafu sa aktuálny graf vymaže.
- `subplot` zobrazí viacero grafov v jednom grafickom okne.

Napríklad

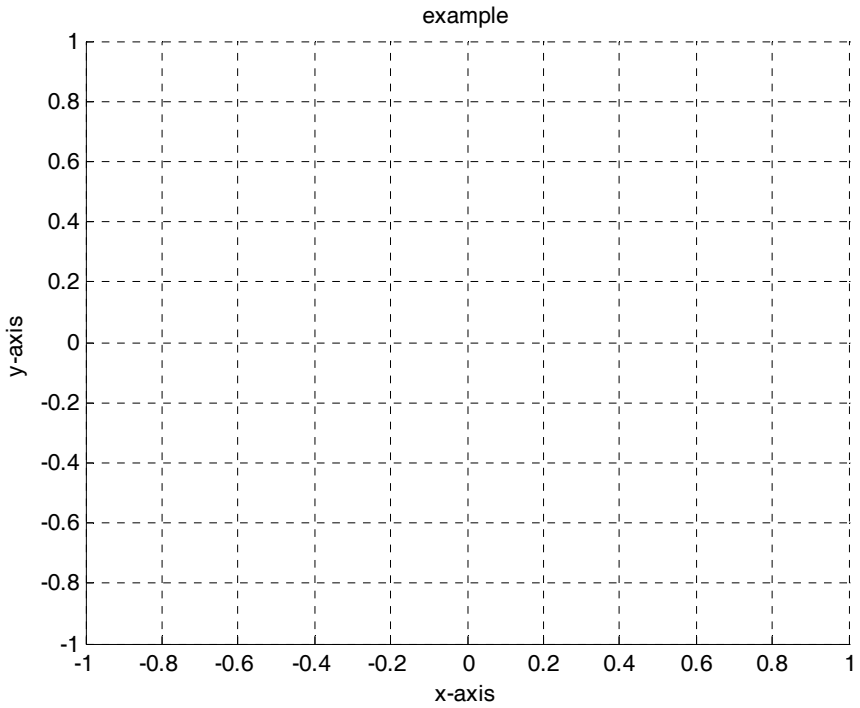
```
>> xlabel('x-axis');
>> ylabel('y-axis');
>> title('example');
```


example
y-axis
x-axis
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

```
>> axis([-1 1 -1 1]);
```

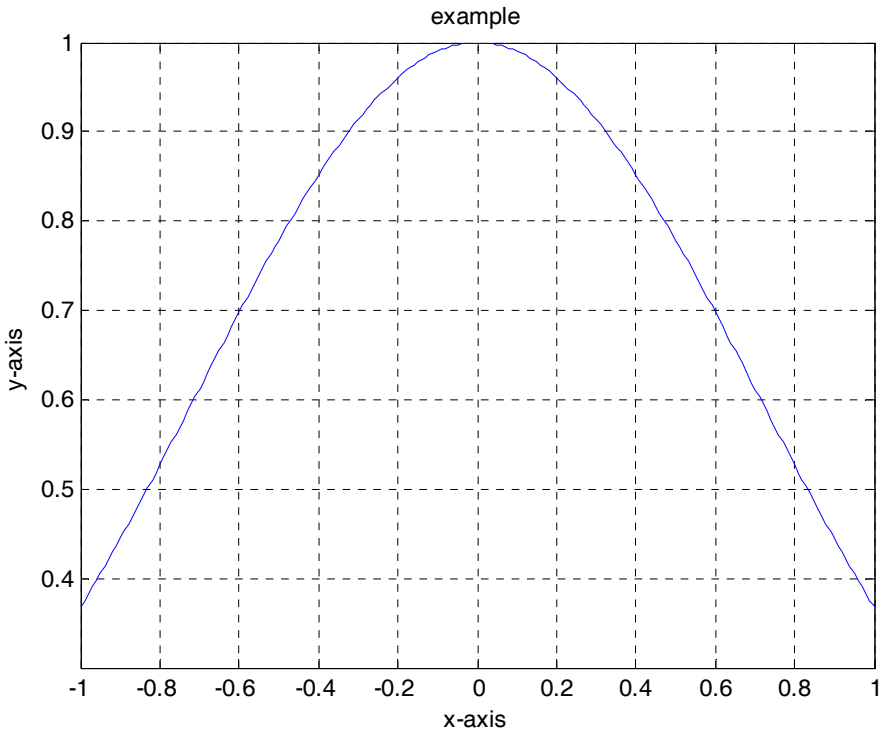

example
y-axis
x-axis
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

```
>> grid on
```



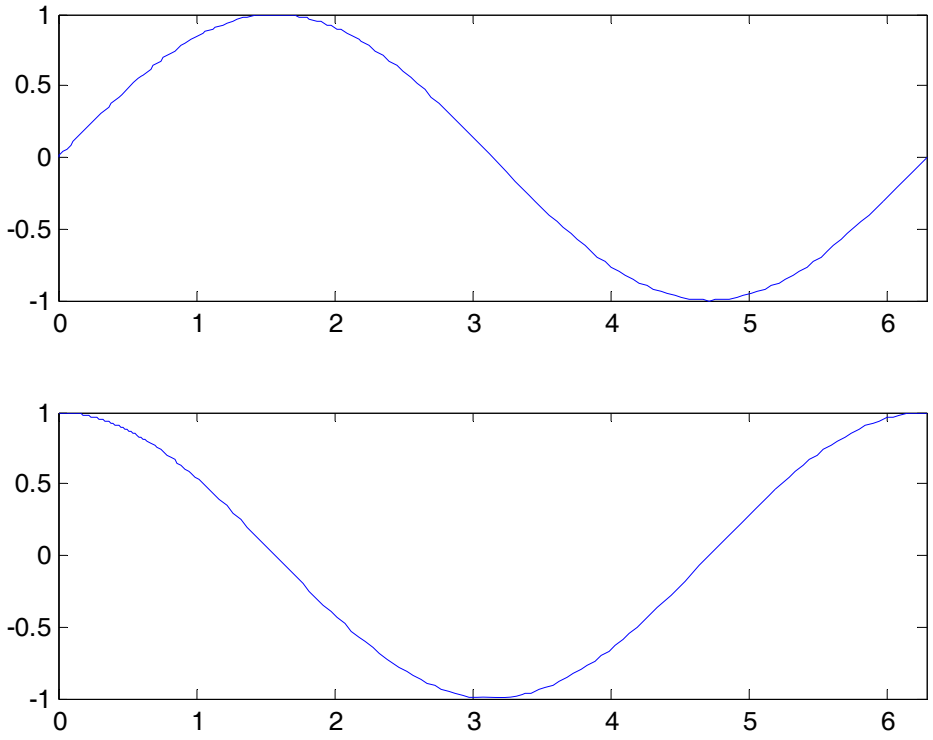
Zobrazíme funkciu $y = e^{-x^2}$, pomenujeme osi a pridáme mriežku.

```
>> u=x.^2;
>> y= exp(-u);
>> plot (x,y);
```



Na vizualizáciu rôznych funkcií naraz použijeme príkaz `subplot`.

```
>> subplot(2,1,1);
>> fplot('sin(x)',[0 2*pi])
>> subplot(2,1,2);
>> fplot('cos(x)',[0 2*pi])
```



Rôzne

- `max(x)` vráti najväčšiu hodnotu *x*, pričom x je vektor. V prípade, že `x` je *k*-rozmerné pole, pozri `help max`.
- `min(x)` je analogický príkaz ku príkazu `max`.
- `abs(x)` vráti pole rovnakého typu ako `x` s absolútnymi hodnotami členov poľa `x`.
- `size(A)` zobrazí vektor typu 1*xk* s počtom riadkov, stĺpcov, atď *k*-rozmerného poľa `A`.
- `length(x)` vráti "dĺžku" poľa, t.j. `max(size(A))`.
- `save fname` uloží aktuálne premenné do súboru s názvom fname.mat.
- `load fname` načíta premenné zo súboru s názvom fname.mat.
- `quit` zatvorí MATLAB

Napríklad

```
>> x=[ 3 -4 60 -71 -13 12];
>> max(x)
```

```
ans =
    60
>> min(x)
ans =
   -71
>> abs(x)
ans =
     3     4    60    71    13    12
```

Jeden príklad na záver. Numerické integrovanie

Numerické integrovanie predstavuje približný výpočet integrálu za použitia numerických techník. Na numerické integrovanie existuje veľa metód. V tomto príklade použijeme tri: obdĺžnikovú metódu, lichobežníkovú metódu a Simpsonovu metodu.

$$\int_0^1 (1-x)^{1/2}\,dx$$

1. Obdĺžniková metóda

Prvá metóda je veľmi jednoduchá. Interval rozdelíme na subintervaly. Na každom subintervale $\langle x_i, x_{i+1}\rangle$ nahradíme funkciu konštantou a vypočítame obsah obdĺžnika so stranou $h = x_{i+1} - x_i$ a výškou $f(x_i)$. Obsahy všetkých obdĺžnikov sčítame.
Interval budeme postupne deliť na 2, 4, 8, 16, 32, 64 a 128 subintervalov.

Pre dva subintervaly na rozdelenie intervalu $\langle 0,1\rangle$ potrebujeme 3 body.

```
>> X=linspace(0,1,3)

X =
         0    0.5000    1.0000
```

Hodnoty funkcie $\sqrt{1-x}$ uložíme do premennej *Y*.

```
>> Y=sqrt(1-X)

Y =
    1.0000    0.7071         0
```

Prvý obdĺžnik má výšku *f(0),* jej hodnota je uložená v prvej zložke vektora *Y*.

```
>> q=Y(1)

q =
     1
```

Druhý obdĺžnik má výšku *f(0.5),* jej hodnota je uložená v druhej zložke vektora *Y*. Výšky obdĺžnikov sčítame.

```
>> q=q+Y(2)

q =
    1.7071
```

Dĺžka subintervalu je 0.5. Súčet výšok vynásobíme touto hodnotou a dostaneme obsah plochy.

```
>> q=q*1/2

q =
    0.8536
```

Keď chceme počítať s väčším počtom subintervalov, na prácu s vektormi použijeme cyklus.
Na rozdelenie intervalu $\langle 0,1 \rangle$ na 128 subintervalov potrebujeme 129 bodov.

```
>> X=linspace(0,1,129);
```

Hodnoty funkcie $\sqrt{1-x}$ uložíme opäť do vektora *Y*.

```
>> Y=sqrt(1-X);
>> q=Y(1);
```

V tomto prípade budeme počítať súčet výšok od *f(0)* po *f(127)*, t.j. od prvého člena vektora *Y(1)* až po predposledný člen *Y(128)*.

```
>> for j=1:127
q=q+Y(j+1);
end
>> q=q*1/128;
```

Nakoniec, na určenie približnej hodnoty integrálu použijeme nasledovný program, ktorý vypočíta približný obsah plochy pri rozdelení intervalu na 2, 4, 8, 16, 32, 64 a 128 subintervalov.

```
function rectangleRule=rectangleRule(Y)
q=zeros(7,1);
N=2; %Pocet subintervalov
for i=1:7
    h=1/N;
    q(i)=Y(1);
    for j=1:N-1
        q(i)=q(i)+Y(j*(256/N)+1);
    end
    q(i)=q(i)*h;
    N=N*2;
end
disp('Aproximacie obdlznikovou metodou:')
      q
```

Vstupom pre výpočet bude 256 členov vektora a funkcia, z ktorej počítame integrál na intervale $\langle 0,1 \rangle$. Vo výstupe dostaneme postupnosť aproximácií integrálu obdĺžnikovou metódou pri rozdelení intervalu na 2, 4, 8, 16, 32, 64 a 128 subintervalov.

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> rectangleRule(Y)
Aproximacie obdlznikovou metodou:
```

```
q =
   0.85355339059327
   0.76828304624275
   0.72063022162445
   0.69483119687723
   0.68118393627894
   0.67408331137851
   0.67043190729683
```

2. Lichobežníková metóda.

V tomto príklade modifikujeme predošlý program na výpočet lichobežníkovou metódou. Dostaneme nasledujúci program:

```
function trapezoidalRule=trapezoidalRule(Y)
q=zeros(7,1);
N=2; % Pocet subintervalov
for i=1:7
   h=1/N;
   q(i)=0.5*Y(1);
   q(i)=q(i)+0.5*Y(257);
   for j=1:N-1
      q(i)=q(i)+Y(j*(256/N)+1);
   end
   q(i)=q(i)*h;
   N=N*2;
end
disp('Aproximacie lichobeznikovou metodou:')
q
```

Vstupom pre výpočet je vektor s 256 členmi a funkcia, z ktorej počítame integrál na intervale $\langle 0,1 \rangle$. Vo výstupe dostaneme postupnosť aproximácií integrálu lichobežníkovou metódou pri rozdelení intervalu na 2, 4, 8, 16, 32, 64 a 128 subintervalov.

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> trapezoidalRule(Y)
Aproximacie lichobeznikovou metodou:
q =
   0.60355339059327
   0.64328304624275
   0.65813022162445
   0.66358119687723
   0.66555893627894
   0.66627081137851
   0.66652565729683
```

3. Simpsonova metóda

Nakoniec použijeme Simpsonovu metódu. Modifikovaný program bude vyzerať nasledovne

```
function simpsonRule=simpsonRule(Y)
q=zeros(7,1);
N=2; % Pocet subintervalov
for i=1:7
   h=1/N;
```

```
    q(i)=Y(1);
    q(i)=q(i)+Y(257);
    k=0;
       for j=1:N-1
          k=k+1;
          q(i)=q(i)+4*Y(k/2*(256/N)+1);
          k=k+1;
          q(i)=q(i)+2*Y(k/2*(256/N)+1);
       end
       k=k+1;
       q(i)=q(i)+4*Y(k/2*(256/N)+1); % bod v strede posledneho interva-
lu.
       q(i)=q(i)*h/6;
       N=N*2;
end
disp(' Aproximacie Simpsonovou metodou:')
q
```

Vstupom pre výpočet bude vektor s 256 členmi a funkcia, z ktorej počítame integrál na intervale $\langle 0,1 \rangle$. Vo výstupe dostaneme postupnosť aproximácií integrálu Simpsonovou metódou pri rozdelení intervalu na 2, 4, 8, 16, 32, 64 a 128 subintervalov.

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> simpsonRule(Y)
Aproximacie Simpsonovou metodou:

q =

   0.65652626479257
   0.66307928008502
   0.66539818862815
   0.66621818274618
   0.66650810307836
   0.66661060593627
   0.66664684620310
```

Výsledky všetkých troch metód uvádzame v tabuľke.

Počet subintervalov	Obdĺžniková metóda	Lichobežníková metóda	Simpsonova metóda
2	0.85355339059327	0.60355339059327	0.65652626479257
4	0.76828304624275	0.64328304624275	0.66307928008502
8	0.72063022162445	0.65813022162445	0.66539818862815
16	0.69483119687723	0.66358119687723	0.66621818274618
32	0.68118393627894	0.66555893627894	0.66650810307836
64	0.67408331137851	0.66627081137851	0.66661060593627
128	0.67043190729683	0.66652565729683	0.66664684620310

Preložila Daniela Richtáriková, SjF STU v Bratislave.