# Symbolická matematika (Symbolic Math Toolbox)

## K čemu to je?

Symbolický toolbox je volitelnou nadstavbou MATLABu (něco jako velká knihovna specializovaných funkcí), prodává se zvlášť. Podobných toolboxů existuje mnoho, každý se specializuje na určitou oblast (např. optimization toolbox nebo image processing toolbox). Pro zajímavost - ceník produktů v rámci ČR: http://www.humusoft.cz/produkty/matlab/cenik/matlab.htm.

Pokud máme k dispozici symbolický toolbox, můžeme MATLAB využívat pro symbolické výpočty (tj. analytické, přesné výpočty) především z oblasti matematické analýzy a lineární algebry:

| | |
|---|---|
| **matematická analýza (kalkul)** | derivování, integrace, limity, sumy,...<br>řešení diferenciálních rovnic a jejich soustav |
| **lineární algebra** | determinanty, inverzní matice, vlastní čísla a vektory,...<br>řešení algebraických rovnic a jejich soustav |
| a další | (např. metody pro zjednodušování algebraických výrazů) |

## Vývoj

Symbolický toolbox prochází podobně jako samotný MATLAB vývojem - až do verze 3.2.3 dodávané pro MATLAB R2008a bylo používáno jádro **MAPLE** (a existoval ještě rozšířený symbolický toolbox - extended symbolic math toolbox). Dne 28. září 2008 vstoupila v platnost nová smlouva, podle které je výpočetní jádro MAPLE v prodeji exkluzivně u firmy Maplesoft. Proto je symbolický toolbox od verze 4.9 (určený pro MATLAB R2007b+) postavený na jádru **MuPAD** (případně si zákazníci mohou koupit licenci MAPLE a používat jej jako alternativu symbolického jádra s využitím nového příkazu `symengine`). Změny byly poměrně velké (http://www.mathworks.com/help/toolbox/symbolic/rn/brqy13y-1.html) a zpočátku symbolický toolbox obsahoval hodně chyb.

V současnosti je k dispozici MATLAB R2010b a s ním Symbolic Math Toolbox ve verzi 5.5, kde je opět dost změn oproti předchozí verzi. Na školních počítačích je MATLAB R2008b a s ním máme k dispozici symbolický toolbox ve verzi 5.1.

## Jaké funkce jsou k dispozici?

Přehled všech symbolických funkcí získáte pomocí příkazu `>> help symbolic`. Na některé z nich se nyní podíváme podrobněji.

# Vytvoření symbolických proměnných a výrazů

Symbolický toolbox *pracuje jen se symbolickými proměnnými* (resp. výrazy).

Symbolickou proměnnou vytvoříme (všimněte si, jak se zobrazuje symbolická proměnná v okně Workspace):

- příkazem `syms`:

```
>> syms x % jedna symb. proměnná
>> syms a b c % více symbolických proměnných
>> syms t real % reálná symb. proměnná; tj. conj(t)=t
```

```
>> syms t clear % z 't' bude obyčejná symb. proměnná
>> syms u v positive % dvě kladné symb. proměnná
```

*Poznámka:* z důvodu zpětné kompatibility je zachována možnost `>> syms t unreal`, která odstraní vlastnost `real` u příslušné proměnné. V nových verzích je to nahrazeno možností `>> syms t clear`, která odstraňuje speciální vlastnosti vytvořené symbolické proměnné.
POZOR: v případě, že úplně smažeme symbolickou proměnnou (`>> clear t`), tak si MuPAD i nadále pamatuje její vlastnost. Takže když ji znovu vytvoříme, přiřadí se jí původní vlastnost automaticky. Příklad:

| krok | příkaz | workspace MATLABu | workspace jádra MuPADu |
|---|---|---|---|
| 1 | `syms x positive` | x | x je kladné |
| 2 | `clear x` | prázdné | x je kladné |
| 3 | `syms x` | x | x je kladné |
| 4 | `syms x clear` | x | prázdné |

- voláním funkce **`sym`**:
```
>> x = sym('x'); % jako syms x
>> t = sym('t', 'real'); % jako syms t real
>> t = sym('t', 'positive'); % jako syms t positive
>> z = sym('z', 'clear'); % jako syms z clear
>> b = sym('beta'); % 'b' se bude vypisovat jako 'beta'
```

Symbolický výraz vytvoříme:

- opět voláním funkce `sym`:
```
>> c = sym('1/10'); % 'c' obsahuje symbolický (přesný) výsledek výrazu
>> c = sym(1/10); % totéž
>> d1 = sym('1/2+4/3'); % 1/2+4/3
>> d2 = sym(1/2+4/3); % 11/6
>> M = sym([2 1; 3 -2]); % převod matice na symbolickou
>> M = sym([2 1; 3 -2],příznak); % převod na symb. proměnnou + JAK převádět
```
  Ve výše uvedeném může mít *příznak* následující hodnotu:
  `'r'`... rational = jako zlomky (implicitní)
  `'f'`... floating-point = tak, jak je číslo reprezentováno v binární soustavě (mantisa, exponent)
  `'e'`... estimate error = jako 'r', ale s uvedením chyby mezi 'r' a 'f' reprezentací
  `'d'`... decimal = číslo nejbližší reprezentaci 'f' při daném počtu číslic (např. na 10 číslic při nastaveném `digits(10)`)
- použitím symbolických proměnných, operací (`+`, `-`, `*`, `/`, `\`, `'`, `^` a `.*`, `./`, `.\`, `.'`, `.^`) a funkcí symbolického toolboxu (viz níže):
```
>> sym x; y = cos(x^2) % 'y' je symbolická funkce
>> v = sqrt(sym(2)) % srovnejte: v2 = sqrt(2)
>> w = sym(2)/sym(5)+sym(1)/sym(3) % srovnejte: w2 = 2/5+1/3
>> sym x; p = 3*x^2 - 2*x + 6 % 'p' je symbolická funkce - polynom
```

  *Poznámky:*

  - o některých operacích (především těch s tečkou) si více povíme příště - týkají se práce s jednotlivými prvky vektoru/matice.
  - porovnávání symbolických proměnných provádíme funkcemi `eq` (test rovnosti) a `ne` (test nerovnosti)

- v případě jádra MAPLE bylo možné vytvořit i faktoriál: `>> faktorial = sym('n!') % n!`
  V případě jádra MuPAD však musíme vytvořit rekurzivní funkci v jazyce MuPADu - více na http://www.mathworks.com/help/toolbox/symbolic/brs6v40.html#bsoxbw3-1.

## Některé základní funkce symbolického toolboxu

| | |
|---|---|
| zaokrouhlování | `round`, `ceil`, `floor`, `fix` |
| odmocnina | `sqrt` |
| zbytek po dělení | `mod` (jen zbytek), `quorem` (zbytek a celočíselný podíl) |
| komplexní čísla | `real` (reálná část), `imag` (imaginární část), `conj` (komplexně sdružené číslo) |
| logaritmy | `log` (přirozený), `log10` (dekadický), `log2` (dvojkový) |
| exponenciála | `exp` |
| goniometrické funkce | `sin`, `cos`, `tan`, `cot`, včetně inverzních a hyperbolických |

## Přesnost výpočtů, formátování výsledků (Variable-Precision Arithmetic)

Existují 3 možnosti: *numeric*, *rational* (implicitní nastavení) a *VPA*.

| možnost | význam | příklad | výsledek |
|---|---|---|---|
| Numeric | pohyblivá řádová čárka (MATLAB) | `format long`<br>`1/2 + 1/3` | `0.833333333333333` |
| Rational | přesný symbolický výpočet (MuPAD) | `sym(1/2) + 1/3` | `5/6` |
| VPA | pohyblivá řádová čárka (MuPAD) | `digits(25)`<br>`vpa('1/2 + 1/3')` | `0.83333333333333333333333333` |

Nejrychlejší, ale nejméně přesná je možnost *numeric*. Vyžaduje i nejméně paměti počítače. Výpis výsledku (datový typ *double*) závisí na nastavení příkazu `format` samotného MATLABu, nicméně interně má osmibajtovou reprezentaci (pohyblivá řádová čárka).

Možnost *rational* je nejpřesnější, ale zabere nejvíc počítačového času a paměti.

Implicitní počet desetinných míst pro možnost *VPA* je 32. Zvýšením této hodnoty (funkcí `digits`) vzrůstá spotřeba času a paměti, ale zvyšuje se přesnost.

## Výpis (nalezení) symbolických proměnných ve výrazu - *symvar*

Od verze 4.9 nabízí symbolický toolbox novou funkci pro hledání symbolických proměnných ve výrazech - **`symvar`**. Funkci lze volat dvěma způsoby:

- `>> symvar(`*`výraz`*`)` - vrací vektor názvů všech symb. proměnných v zadaném výrazu (řazeno abecedně, velká písmena před malými). Pokud výraz neobsahuje symb. proměnnou, je vrácen prázdný vektor. Názvy `pi`, `i` a `j` nejsou považovány za proměnné!
- `>> symvar(`*`výraz, n`*`)` - vrací vektor názvů *n* symb. proměnných, které jsou abecedně nejblíže písmenu *x*. Řazeno abecedně (malá písmena před velkými), tedy jinak než v předchozím případě.

Volání `symvar(`*`výraz`*`,1)` vrací proměnnou nejblíže písmenu *x*. Právě ona je použita při derivování, integrování, substitucích či řešení rovnic jako implicitní proměnná.

**Příklady:**

```
>> syms x y z a b
>> w = x^2/(sin(3*y - b));
>> symvar(w) % [ b, x, y]
```

```
>> syms v z
>> g = v + z;
>> symvar(g, 1) % z
```

```
>> syms X1 x2 xa xb
>> g = X1 + x2 + xa + xb;
>> symvar(g, 1) % x2
```

*Poznámka:* ve starších verzích toolboxu (před verzí 4.9) symbolické proměnné ve výrazu určí funkce **`findsym`**:

```
>> syms a b n t x
>> f1 = a*x^2 + b*x; f2 = t^n;
>> findsym(f1) % a, b, x
>> findsym(f2) % n, t
```

# Substituce v symbolických výrazech - *subs*

Náhradu symbolických proměnných za jiné symbolické proměnné nebo za konkrétní číselné hodnoty provedeme pomocí funkce **`subs`**.

Funkci lze volat především těmito způsoby:

- **`v = subs(výraz, new)`** - nahradí implicitní symbolickou proměnnou (viz funkce `symvar`) hodnotou/proměnnou *new*.
- **`v = subs(výraz, old, new)`** - nahradí proměnnou *old* za hodnotu/proměnnou *new*. Parametr *old* může být symbolická proměnná, řetězec s názvem symb. proměnné, vektor proměnných/názvů nebo pole buněk proměnných/názvů. Parametr *new* musí být stejné délky a typu jako *old*, obsahuje symb. proměnné nebo hodnoty, kterými budeme nahrazovat. Nahrazují se odpovídající prvky vektoru (pole buněk).
  V případě, že *výraz* i *old* jsou skaláry, tak lze nahrazovat vektorem hodnot nebo maticí - tehdy je *new* vektor/matice typu *double*.

Pokud nahradíme VŠECHNY symbolické proměnné čísly, je výsledkem číslo (typu *double*). V ostatních případech je výsledkem vznikne symbolický výraz.

**Příklady:**

```
>> syms x y
>> f = x^2 -2*x + 1
>> subs(f,1) % číslo 0
>> subs(f,y) % y^2-2*y+1, tj. symb. výraz s jinou proměnnou
>> subs(f,sym('beta')) % beta^2-2*beta+1

>> g = x*y - 5*x + sqrt(y) % VÍCE PROMĚNNÝCH
>> subs(g,1) % y-5+y^(1/2) ... pro x=1 (defaultní proměnná)
>> subs(g,y,1) % -4*x+1 ... pro y=1
>> subs(g,{x,y},{1,4}) % číslo 1 ... pro x=1 a y=4
>> subs(g,{y,x},{4,1}) % totéž

>> f2 = x*y % náhrada vektorem
>> vysl = subs(f2,{x,y},{[1 2 3 4],[2 1 5 0]}) % vysl = [2 2 15 0], zpracovává se po prvcích

>> clear; syms a b c x
>> f = a*x^2 + b*x + c
>> kvadr = subs(f,{a,b,c},{1,-2,1}) % kvadr = x^2-2*x+1
```

# Převody symbolických výrazů (na číslo či řetězec)

Převod na číselné hodnoty - funkce **double**:

```
>> c1 = sym(1/5+3/4-1/6) % symb. proměnná 47/60
>> c2 = double(c1) % 0.7833 ... skalár typu double
```

Převod na řetězce - funkce **char**:

```
>> s1 = sym('a*x^2+b*x+c') % symb. výraz
>> s2 = char(s1) % řetězec
>> A = sym(hilb(3)); % Hilbertova matice řádu 3
>> char(A) % ans = matrix([[1,1/2,1/3],[1/2,1/3,1/4],[1/3,1/4,1/5]])
```

Převod na LaTeXový řetězec - funkce **latex**:

```
>> str = latex(sym('a*x^2+b*x+c')) % a{x}^{2}+bx+c
>> A = sym(hilb(2)); % Hilbertova matice řádu 2
>> latex(A)
% ans = \left[ \begin {array}{cc} 1&1/2\\\noalign{\medskip}1/2&1/3\end {array} \right]
```

Také lze převádět polynom na vektor a obráceně - viz funkce **sym2poly** a **poly2sym**.

# Vylepšení vzhledu výsledku - *pretty*

Výsledky symbolických výrazů v Command Window nevypadají zrovna "matematicky". Pro výpis výsledku "jako na papíře" můžeme použít funkci **pretty(symb_výraz)**.

**Příklady**:

```
>> syms x
>> f = x^3-6*x^2+11*x-6 % polynom v neupraveném výpisu
>> pretty(f)

       3     2
      x - 6 x  + 11 x - 6
```

# Zjednodušování výrazů

Funkce `pretty` neumí výraz zjednodušit, pouze ho vzhledově "zpřístupní" lidem. Zjednodušování provádějí následující funkce (ve všech příkladech předpokládáme `syms a b c x y`):

- **simplify(f)** ... základní funkce pro zjednodušování výrazů (využívá zjednodušovací pravidla MuPADu)

| f | simplify(f) |
|---|---|
| sin(x)^2 + cos(x)^2 | 1 |
| exp(c*log(sqrt(a+b))) | (a + b)^(c/2) |
| (x^2 + 5*x + 6)/(x + 2) | x + 3 |
| sqrt(16) | 4 |

- **`collect(f)`** ... nahlíží na `f` jako na polynom a sloučí stejné mocniny
  **`collect(f,proměnná)`** ... sloučí části obsahující zadanou proměnnou

| f | collect(f) |
|---|---|
| (exp(x)+x)*(x+2) | x^2 + (exp(x) + 2)*x + 2*exp(x) |
| (x+1)*(y+1) | y + x*(y + 1) + 1 |
| collect((x+y)*(x^2+y^2+1), y) | y^3 + x*y^2 + (x^2 + 1)*y + x*(x^2 + 1) |

- **`expand(f)`** ... expanze: roznásobí se závorky a aplikují součtové vzorce

| f | expand(f) |
|---|---|
| (x-2)*(x-4) | x^2 - 6*x + 8 |
| cos(x+y) | cos(x)*cos(y) - sin(x)*sin(y) |
| exp((a+b)^2) | exp(2*a*b)*exp(a^2)*exp(b^2) |

- **`factor(f)`** ... je-li `f` polynom s racionálními koeficienty, pokusí se jej vypsat jako součin kořenových činitelů

| f | factor(f) |
|---|---|
| a^2 - b^2 | (a - b)*(a + b) |
| a^3 + b^3 | (a + b)*(a^2 - a*b + b^2) |
| x^3-y^3 | (x - y)*(x^2 + x*y + y^2) |

*Poznámka:* jestliže vstupem funkce `factor` je přirozené číslo (symbolické), pak výsledkem je jeho **prvočíselný rozklad**:

```
>> factor(sym('12345678901234567890'))
ans =
2*3^2*5*101*3541*3607*3803*27961
```

- **`horner(f)`** ... vytvoří Hornerovo schéma symbolického polynomu `f` (vstupem musí být polynom)

| f | horner(f) |
|---|---|
| x^3-6*x^2+11*x-6 | x*(x*(x - 6) + 11) - 6 |
| x^2+x | x*(x + 1) |
| y^3-2*y | y*(y^2 - 2) |

- **`vysledek = simple(f)`** ... zkouší aplikovat různá algebraická pravidla tak, aby výsledek měl co nejméně znaků (občas je vhodné ji použít např. dvakrát). Vypisuje všechny pokusy o zjednodušení (pokud není volána druhým způsobem)!
  **`[vysledek, metoda] = simple(f)`** ... vrátí nejen výsledek zjednodušení, ale i metodu použitou pro zjednodušení. Nevypisuje pokusy o zjednodušení.

| volání | konečný výsledek |
|---|---|
| `simple(cos(3*acos(x)))` | `4*x^3 - 3*x` |
| `simple(cos(x) + i*sin(x))` | `exp(x*i)` |
| `f = (x + 1)*x*(x - 1);`<br>`[f, jak] = simple(f)` | `f =`<br>`x^3 - x`<br>`jak =`<br>`simplify(100)` |

# Matematická analýza

## Kreslení grafů symbolických funkcí

Symbolický toolbox nabízí několik funkcí pro kreslení grafů symbolických funkcí:

- graf symbolické **funkce jedné proměnné** (zadané symbolickým výrazem: explicitně, implicitně nebo parametricky) vytvoří funkce **`ezplot`**:
  - **`ezplot(f)`** ... graf funkce `f` pro nezávisle proměnnou z intervalu [-2π;2π]
    Pro **implicitně zadanou funkci** f(*x*,*y*)=0 jsou *x* i *y* z intervalu [-2π;2π].
  - **`ezplot(f, [a b])`** ... graf funkce `f` pro nezávisle proměnnou z intervalu [*a*;*b*]
    V případě implicitně zadané funkce je *x* i *y* z intervalu [*a*;*b*].
  - **`ezplot(x,y)`** ... graf **parametricky zadané křivky** `x=x(t), y=y(t)` pro *t* z intervalu [0;2π];
  - **`ezplot(x,y,[a b])`** ... graf parametricky zadané křivky `x=x(t), y=y(t)` pro *t* z intervalu [*a*;*b*].

  **Příklady:**

```
>> syms u;    ezplot(cos(u)) % kosinus

>> syms x y;  ezplot(x^2-y^4) % implicitně zadaná fce

>> syms x;    ezplot(cos(x),sin(x),[0 2*pi]) % parametricky zadaná kružnice
```

- graf symbolické **funkce dvou proměnných** vytvářejí následující funkce:
  **`ezmesh`** ... "síťový" 3D graf
  **`ezmeshc`** ... "síťový" 3D graf s vrstevnicemi
  **`ezsurf`** ... "ploškový" 3D graf
  **`ezsurfc`** ... "ploškový" 3D graf s vrstevnicemi
  **`ezplot3`** ... 3D graf parametricky zadané křivky (1 parametr)
  **`ezcontour`** ... 2D graf vrstevnic
  **`ezcontourf`** ... vyplněný 2D graf vrstevnic
  Podrobnosti naleznete v nápovědě.
- graf v polárních souřadnicích - funkce **`ezpolar`**. Podrobnosti naleznete v nápovědě.

### Grafické handly

V grafickém režimu MATLABu lze každou část okna Figure chápat jako objekt, který má svůj *grafický handle* (grafický "ukazatel" neboli "ovladač", angl. handle graphics). Grafický handle lze nejsnáze získat jako výstup funkce, která něco kreslí (např. `ezplot`).

Pomocí grafického handlu lze ovlivňovat vlastnosti každého objektu (grafické objekty mají svou hierarchiii). Podrobněji si o grafických handlech povíme příště. Nyní stačí vědět, že:

- ke zjištění dostupných vlastností grafického objektu používáme funkci `get`:
  **`get(handle)`** ... vypíše VŠECHNY vlastnosti objektu s jejich aktuálními hodnotami
  **`get(handle,'vlastnost')`** ... vypíše aktuální hodnotu zadané vlastnosti
- ke změně jedné nebo více vlastností grafického objektu používáme funkci `set`:
  **`set(objekt, 'vlastnost', 'hodnota')`** ... nastaví novou hodnotu dané vlastnosti
  **`set(objekt, 'vlastnost', 'hodnota', 'vlastnost2', 'hodnota2')`** ... lze upravovat více vlastností najednou (někdy je to dokonce nutné)

**Příklad** - změna barvy grafu vykresleného pomocí `ezplot`:

```
>> c = ezplot(f); % odebereme si grafický handle
>> set(c,'color',[0 .3 0]) % změna barvy čáry grafu na tmavozelenou
```

Pozn.: každá z barev je definována kombinací 3 základních barev RGB (červená - zelená - modrá) v rozsahu [0;1].

# Symbolická integrace - *int*

Funkce `int` vrací jeden výstup (symbolický), který můžeme uložit do nějaké proměnné, a pak třeba převést na *double* (po vhodné substituci - viz výše: `subs`). Funkci lze použít pro dva účely:

1. **výpočet určitého integrálu**
   **`int(výraz,a,b)`**, kde *výraz* musí obsahovat některou existující symbolickou proměnnou, *a* je dolní mez a *b* je horní mez integrálu. Při výpočtu je použita implicitní symbolická proměnná (viz `symvar`).
   **`int(výraz,v,a,b)`**, kde platí výše uvedené a *v* určuje symbolickou proměnnou, pro kterou chceme určitý integrál spočítat.

   **Příklad** - určitý integrál polynomu:

   ```
   >> syms x
   >> int(x^2+3*x-5,0,2)
   ans =
   -4/3 % výsledkem je zase symbolická proměnná
   ```

2. **nalezení primitivní funkce**
   **`int(výraz)`** ... primitivní funkce pro implicitní symb. proměnnou.
   **`int(výraz,v)`** ... primitivní funkce pro symb. proměnnou *v*.

   Pro vylepšení vzhledu výsledku můžeme použít funkci `pretty`.

   **Příklad** - primitivní funkce:

   ```
   >> syms x; f3 = int(x^2+3*x-5) % primitivní funkce k polynomu
   f3 =
   1/3*x^3+3/2*x^2-5*x
   >> pretty(f3)

                3        2
         1/3 x  + 3/2 x  - 5 x

   >> clear; syms x z;
   >> f4 = int(x/(1 + z^2), z) % primitivní funkce vzhledem k proměnné 'z'
   ```

```
f4 =
x*atan(z)
```

V případě, že integrál existuje, můžeme symbolické výsledky použít při ověřování výsledků získaných numericky.

# Symbolická derivace funkce - *diff*

Výstupem funkce `diff` je symbolický výraz = hledaná derivace (můžeme ji uložit do proměnné).

1. **první derivace**
   **`diff(výraz)`**, kde *výraz* musí obsahovat symbolickou proměnnou. Derivuje se podle implicitní symb. proměnné (viz `symvar`).
   **`diff(výraz,proměnná)`** nebo **`diff(výraz,sym('proměnná'))`** ... první derivace podle zadané *proměnné* (pokud výraz obsahuje více symb. proměnných).

   **Příklady**:
   ```
   >> syms a x
   >> f = sin(a^2*x);
   >> diff(f) % 1. deriv. podle 'x' => a^2*cos(a^2*x)
   >> diff(f,a) % 1. deriv. podle 'a' => 2*a*x*cos(a^2*x)

   >> syms t
   >> diff(t^6,6) % 6. derivace podle 't' => 720
   ```

2. **derivace vyšších řádů**
   **`diff(výraz,n)`**, kde `n` musí být kladné celé číslo (značí řád derivace). Derivuje se podle implicitní symb. proměnné.
   **`diff(výraz,prom,n)`** ... *n*-tá derivace *výrazu* podle symb. proměnné *prom*.

   **Příklady**:
   ```
   >> syms a x
   >> diff(x^3+5*x^2-5*x,2); % 6*x+10 ... 2. derivace
   >> diff(x^3+a*x^2-2*a*x,2); % 6*x+2*a ... 2. derivace dle 'x'
   >> diff(x^3+a*x^2-2*a*x,a,2); % 0 ... 2. derivace dle 'a'
   ```

**Příklad** - derivace polynomu a vykreslení dvou grafů do jednoho obrázku:
```
>> syms x
>> y = x^2+3*x-5;
>> dy = diff(y)
>> ezplot(y) % graf fce, modrá čára
>> hold on % zachovat staré grafy
>> h = ezplot(dy); set(h,'color',[1 0 0]) % graf derivace, červená čára
>> hold off
```
*Poznámka:* graf by se měl také správně popsat pomocí funkcí `xlabel`, `title`, `legend` - podrobnosti si uvedeme příště.

**Příklad** - první, druhá a třetí derivace funkce $\sin(x^2)$:
```
>> syms x
>> diff(sin(x^2)) % 1. derivace
ans =
2*cos(x^2)*x
```

```
>> diff(sin(x^2),2) % 2. derivace
ans =
-4*sin(x^2)*x^2+2*cos(x^2)
>> diff(sin(x^2),3) % 3. derivace
ans =
-8*cos(x^2)*x^3-12*sin(x^2)*x
>> pretty(diff(sin(x^2),3)) % hezky vypis

                  2   3           2
          -8 cos(x ) x  - 12 sin(x ) x
```

# Jakobián - *jacobian*

Pomocí **`jacobian(f, v)`** spočítáme jakobián (funkcionální determinant, Jacobiho determinant) funkce `f` vzhledem k proměnné `v`. V případě, že `f` je skalár, tak funkce vrátí gradient, tedy stejný výsledek jako volání `diff(f,v)`. V případě, že `f` je vektor, tak výsledek obsahuje na pozici (*i*,*j*) hodnotu parciální derivace $\delta f_i/\delta v_j$.

**Příklad**:

```
>> syms x y z
>> f = [x*y*z; y; x + z]; % vektorová funkce 3 proměnných
>> v = [x, y, z]; % proměnné
>> R = jacobian(f, v)
R =
[ y*z, x*z, x*y]
[   0,   1,   0]
[   1,   0,   1]

>> b = jacobian(x+z, v) % skalární funkce 3 proměnných
b =
[ 1, 0, 1]
>> diff(x+z, x), diff(x+z, y), diff(x+z, z) % pro srovnání
```

# Symbolická kalkulačka - *funtool*

Příkazem **`funtool`** se spustí symbolická kalkulačka - je obsažena ve třech grafických oknech a umožňuje pracovat se dvěma symbolickými funkcemi (a současně ukazuje jejich grafy). Každou funkci lze derivovat, integrovat, invertovat,... stisknutím správného tlačítka.

# Symbolická limita funkce - *limit*

**`limit(výraz)`** ... limita (oboustranná) *výrazu*, kdy implicitní symbolická proměnná (nalezená pomocí `symvar`) se blíží nule
**`limit(výraz,a)`** ... limita *výrazu*, kdy se implicitní symbolická proměnná blíží k hodnotě *a*
**`limit(výraz,p,a)`** ... limita *výrazu*, kdy symbolická proměnná *p* se blíží k hodnotě *a*
**`limit(výraz,p,a,'left')`** ... limita *výrazu*, kdy symbolická proměnná *p* se blíží k hodnotě *a* ZLEVA
**`limit(výraz,p,a,'right')`** ... limita *výrazu*, kdy symbolická proměnná *p* se blíží k hodnotě *a* ZPRAVA

**Příklady:**

```
>> syms x
```

```
>> limit(sin(x)/x) % 1
>> limit(1/x) % NaN
>> limit(1/x,x,0,'left') % -inf
>> limit(1/x,x,0,'right') % inf
>> limit(1/x+5,x,inf) % 5
>> syms h;
>> limit((sin(x + h) - sin(x))/h, 0) % sin(h)/h, derivace sin(x) podle definice
>> limit((sin(x + h) - sin(x))/h, h, 0) % cos(x)
```

# Symbolický součet řady - *symsum*

**`v = symsum(`*`výraz`*`)`** ... vztah pro součet od 0 do x-1, kde x je symbolická proměnná použitá ve *výrazu* (defaultní proměnná - viz výše)
**`v = symsum(`*`výraz,k`*`)`** ... vztah pro součet od 0 do *k*-1, kde *k* je určená symbolická proměnná
**`v = symsum(`*`výraz,a,b`*`)`** ... vztah pro součet od a do b pro symbolickou proměnnou nalezenou automaticky
**`v = symsum(`*`výraz,k,a,b`*`)`** ... vztah pro součet od a do b pro symbolickou proměnnou k

**Příklady:**

```
>> syms s
>> s1 = symsum(1/s^2,1,inf) % nekonečná řada; s1 = 1/6*pi^2
>> s_obec=symsum(s^2) % s_obec = 1/3*s^3-1/2*s^2+1/6*s
>> s2=symsum(s^2,0,10) % s2 = 385, tj. 0+1+4+9+...+100
>> subs(s_obec,11) % ověření 's2' ... 's_obec' pro prvních 11 členů (0-10)
ans =
    385.0000 % zde už výsledek není symbolická proměnná
```

# Symbolický Taylorův rozvoj funkce (Taylorova řada) - *taylor*

**`taylor(`*`funkce`*`)`** ... Taylor v bodě nula => Maclaurinův polynom 5. řádu (Maclaurinova řada pro 0,1,...,5) aproximující zadanou funkci
**`taylor(`*`funkce,n`*`)`** ... pro kladné celé *n* vrátí Maclaurinův polynom (*n*-1)-ního řádu (Maclaurinova řada pro 0,1,...,n-1) aproximující zadanou funkci
**`taylor(`*`funkce,a`*`)`** ... pro reálné *a* vrátí Taylorovu řadu v bodě *a* (5. řádu)
**`taylor(`*`funkce,n,a`*`)`** ... pro kladné celé *n* a reálné *a* vrátí Taylorovu řadu v bodě *a* (řádu *n*-1)
**`taylor(`*`funkce,n,v`*`)`** ... pro kladné celé *n* a symb. proměnnou *v* vrátí Maclaurinův polynom (řádu *n*-1)
**`taylor(`*`funkce,n,v,a`*`)`** ... pro kladné celé *n* vrátí Taylorovu řadu (řád *n*-1) v bodě *a* pro symb. proměnnou *v*

Podrobnosti (resp. přehledné matematické zápisy řad) naleznete v nápovědě: `>> doc taylor`.

**Příklady:**

```
>> syms x t
>> taylor(exp(-x)) % 1-x+1/2*x^2-1/6*x^3+1/24*x^4-1/120*x^5
>> taylor(log(x),6,1) % x-1-1/2*(x-1)^2+1/3*(x-1)^3-1/4*(x-1)^4+1/5*(x-1)^5
>> taylor(sin(x),pi/2,6) % 1-1/2*(x-1/2*pi)^2+1/24*(x-1/2*pi)^4
>> taylor(x^t,3,t) % 1+log(x)*t+1/2*log(x)^2*t^2

>> taylor(exp(-x)) % neupravený výpis Maclaurinovy řady pro funkci e^(-x)
 ans =
 1-x+1/2*x^2-1/6*x^3+1/24*x^4-1/120*x^5
>> pretty(taylor(exp(-x))) % hezčí výpis, "jako na papíře"
```

```
                  2         3         4          5
    1 - x + 1/2 x  - 1/6 x  + 1/24 x  - 1/120 x
```

*Poznámka:* zkuste `>> latex(taylor(exp(-x)))`.
Dále můžete zkusit spustit grafický nástroj **`taylortool`**, který ukazuje i graf funkce a její aproximace pomocí Taylorova polynomu.

# Symbolické řešení diferenciálních rovnic (i soustav) - *dsolve*

**`r = dsolve('eq1','eq2',...,'c1','c2',...,'proměnná')`**
nebo "zkrácená" verze
**`r = dsolve('eq1,eq2,...', 'c1,c2,...', 'proměnná')`**

`dsolve` řeší obyčejnou diferenciální rovnici `eq1` (resp. soustavu rovnic) s počáteční podmínkou `c1` (resp. počátečními podmínkami) pro nezávisle proměnnou `proměnná` (implicitní je `t`!!!).

Zadávání vstupů:

- diferenciální rovnice:
  Derivaci označíme písmenem D. Pokud za ním následuje číslo *n*, tak jde o *n*-tou derivaci (např. D2 značí 2. derivaci). Jakýkoli znak následující poté označuje závisle proměnnou, např. D3y značí 3. derivaci funkce y(*x*), resp. y(*t*).
- počáteční podmínky:
  Každou podmínku zadáváme ve tvaru rovnosti `y(a) = b` nebo `Dy(a) = b`, kde `y` je závisle proměnná a `a` i `b` jsou konstanty. Pokud je počátečních podmínek méně než diferenciálních rovnic, pak výsledek bude obsahovat konstanty `C1`, `C2`,....

Funkce může vrátit následující tři typy výstupů:

- V případě jedné dif. rovnice a jednoho výstupu funkce vrátí výsledné řešení; u nelineární rovnice vrátí nalezená řešení v symbolickém vektoru.
- Pro *m* diferenciálních rovnic a *m* výstupů funkce seřadí výsledky abecedně a přiřadí je jeden po druhém do odebíraných výstupů.
- Pro *m* diferenciálních rovnic a jeden výstup funkce vrátí strukturu obsahující jednotlivá řešení.

Pokud `dsolve` nemůže najít explicitní řešení, pokusí se najít řešení implicitní (v tomto případě funkce generuje varování). Pokud `dsolve` nemůže najít explicitní ani implicitní řešení, pak vygeneruje varování a vrátí prázdnou symbolickou proměnnou. V takovém případě můžete nalézt *numerické řešení* pomocí funkcí MATLABu (např. `ode23` nebo `ode45`).

POZOR:

- Pro některé případy nelineárních rovnic je výstupem ekvivalentní diferenciální rovnice nižšího řádu nebo integrál.
- Funkce nezaručuje korektnost a úplnost nalezeného řešení, proto je vhodné výsledky ověřit (např. derivováním a funkcí `eq`, neboť tvar derivace se může od zadání lišit).

**Příklady**:

```
% dif. rovnice 1. řádu pro funkci x(t)
>> dsolve('Dx = -a*x')
```

```
ans =
C2/exp(a*t) % řešení

% dif. rovnice 1. řádu pro funkci f(t)
>> dsolve('Df = f + sin(t)')
ans =
C4*exp(t) - sin(t)/2 - cos(t)/2 % řešení

% nelineární dif. rovnice 1. řádu, pro funkci y(s)
>> dsolve('(Dy)^2 + y^2 = 1','s')
ans =
             1
            -1
 cosh(C7 + s*i)
 cosh(C11 - s*i)

% dif. rce 1. řádu s poč. podmínkou; funkce y(t)
>> dsolve('Dy = a*y', 'y(0) = b')
ans =
b*exp(a*t)

% dif. rovnice 2. řádu s poč. podmínkami
>> dsolve('D2y = -a^2*y', 'y(0) = 1', 'Dy(pi/a) = 0')
ans =
(1/exp(a*t*i))/2 + exp(a*t*i)/2

% SOUSTAVA DIF. ROVNIC:
>> z = dsolve('Dx = y', 'Dy = -x')
z =
    x: [1x1 sym]
    y: [1x1 sym]

>> z.x % první hledaná funkce
ans =
C20*cos(t) + C19*sin(t)

>> z.y % druhá hledaná funkce
ans =
C19*cos(t) - C20*sin(t)
```

# Lineární algebra

## Matice

- ### operace s maticemi

  Matice lze sčítat (operátor +), odčítat (operátor -), násobit skalárem či jinou maticí (operátor *) a transponovat (operátor ') podle pravidel lineární algebry. Dále je lze umocňovat a dělit. Více si povíme za 2 týdny.

- ### hodnost matice - *rank*

  Hodnost matice vrací funkce `rank` - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice!

  **Příklad**:
```
>> syms a b c d
>> A = [a b;c d];
>> rank(A) % 2
>> rank(subs(A,{b d},{0 0})) % 1
```

- ### jádro zobrazení - ortogonální doplněk - řešení homogenní soustavy - *null*

  Symbolickou funkci `null` lze využít pro:

  - vrácení báze jádra zobrazení příslušného k dané matici,
  - řešení homogenní soustavy (báze podprostoru všech řešení) se zadanou maticí soustavy,
  - vrácení báze ortogonálního doplňku podprostoru, jehož generátory jsou ŘÁDKY dané matice.

  Ve všech případech je vrácena matice, jejíž *sloupce* představují vektory hledané báze. V případě, že řešením je triviální podprostor, tak je vrácena prázdná symbolická proměnná.
  POZOR: symbolická funkce je opět volána jen v případě, že vstupem funkce je symbolická matice!

  **Příklad** - nalezněte bázi ortogonálního doplňku k podprostoru $P$=[(1,1,1,0), (2,3,1,-1), (1,1,2,1)] v $\mathbf{R}^4$:

```
>> P = sym([1 1 1 0; 2 3 1 -1; 1 1 2 1]); % podprostor P
>> B = null(P) % báze ortog. doplňku
B =
  1
  0
 -1
  1
>> P*B % kontrola - musí vyjít nulový vektor
```

- ### obraz zobrazení - báze sloupcového prostoru matice - báze podprostoru - *colspace*

  Symbolickou funkci `colspace` lze využít pro:

  - vrácení báze obrazu zobrazení příslušného k dané matici,
  - vrácení báze sloupcového podprostoru matice,
  - vrácení báze libovolného podprostoru, jehož generátory jsou SLOUPCE dané matice.

Ve všech případech je vrácena matice, jejíž *sloupce* představují "hezké" vektory hledané báze (přesněji: výsledek je matice v Hermiteově tvaru).
POZOR: symbolická funkce je opět volána jen v případě, že vstupem funkce je symbolická matice!

**Příklad**:

```
>> A = sym([2,0,2;3,4,7;0,5,5])
A =
[ 2, 0, 2]
[ 3, 4, 7]
[ 0, 5, 5]
>> B = colspace(A) % báze obrazu zobrazení
B =
[     1,   0]
[     0,   1]
[ -15/8, 5/4]
```

- ## Hermiteův tvar matice - *rref*

Převod matice na Hermiteův tvar ("hezká" horní stupňovitá matice) provede funkce **`rref`** (z anglického "reduced row echelon form") - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice!

**Příklad**:

```
>> syms a b c
>> A = [a b c; b c a; a + b, b + c, c + a];
>> rref(A) % Hermiteův tvar

ans =
[ 1, 0, -(a*b - c^2)/(a*c - b^2)]
[ 0, 1, -(b*c - a^2)/(a*c - b^2)]
[ 0, 0,                        0]
```

- ## determinant - *det*

Determinat se vypočte pomocí funkce **`det`** - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice!

**Příklady**:

```
>> syms a b c d;
>> det([a, b; c, d]) % a*d - b*c

>> A = sym([2/3 1/3; 1 1]);
>> r = det(A) % r = 1/3
```

- ## inverzní matice - *inv*

Inverzní matici vrací funkce **`inv`** - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice! Matice musí mít nenulový determinant.

**Příklady**:

```
>> A = sym([2,-1,0;-1,2,-1;0,-1,2]);
>> inv(A)
```

```
ans =
[ 3/4, 1/2, 1/4]
[ 1/2, 1, 1/2]
[ 1/4, 1/2, 3/4]

>> syms a b c d;
>> A = [a b; c d];
>> inv(A)
ans =
[ d/(a*d - b*c), -b/(a*d - b*c)]
[ -c/(a*d - b*c), a/(a*d - b*c)]
```

- ## charakteristický polynom matice - *poly*

Charakteristický polynom vrací symbolická funkce **`poly`** - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice! Jako druhý vstup lze zadat jméno symbolické proměnné použité v charakteristickém polynomu.

**Příklad**:

```
>> A = sym([-149 -50 -154; 537 180 546; -27 -9 -25]);
>> p1 = poly(A); % p1 = x^3 - 6*x^2 + 11*x - 6
>> p2 = poly(A,sym('lambda')); % p2 = lambda^3 - 6*lambda^2 + 11*lambda - 6
```

- ## vlastní čísla a vlastní vektory matice - *eig*

Vlastní čísla (angl. eigenvalues) a vlastní vektory (angl. eigenvectors) matice vrací funkce **`eig`** - symbolická funkce je volána jen v případě, že vstupem funkce je symbolická matice!
**`lambda = eig(A)`** ... vrací všechna vlastní čísla matice `A`,
**`[X,D] = eig(A)`** ... vrací všechny vlastní vektory (sloupce matice `X`) náležející k odpovídajícím vlastním číslům na diagonále matice `D`. Lze použít při diaonalizaci matice `A`.

**Příklad**:

```
>> M = sym([1 1 1; 3 1 0; 4 2 1]);
>> l = eig(M) % jenom vlastní čísla
l =
  0
  4
 -1
>> [X,D] = eig(M) % vl. vektory a vl. čísla
X =
[ -2,  1,  1]
[  3,  1, -3]
[  1,  2,  2]

D =
[ -1,  0,  0]
[  0,  4,  0]
[  0,  0,  0]
```

- ## Jordanův kanonický tvar matice - *jordan*

Funkce **`jordan`** - viz nápověda.

- **funkce *diag***

  `M = diag(vektor)` ... vytvoří diagonální matici ze zadaného vektoru,
  `M = diag(vektor,k)` ... vytvoří matici ze zadaného vektoru která je pro `k=0` diagonální, pro `k<0` jsou prvky pod diagonálou na `k`-té rovnoběžce a pro `k>0` jsou prvky nad diagonálou na `k`-té rovnoběžce,
  `d = diag(matice)` ... extrahuje diagonální prvky zadané matice (jako vektor),
  `d = diag(matice,k)` ... vrací všechny prvky pod (`k<0`)/nad (`k>0`) hlavní diagonálou zadané matice (jako vektor).

  **Příklady**:

```
>> syms a b c;
>> v = [a b c];
>> diag(v) % tvorba diagonální matice
ans =
[ a, 0, 0]
[ 0, b, 0]
[ 0, 0, c]
>> diag(v, -2) % tvorba matice s "druhou poddiagonálou"
ans =
[ 0, 0, 0, 0, 0]
[ 0, 0, 0, 0, 0]
[ a, 0, 0, 0, 0]
[ 0, b, 0, 0, 0]
[ 0, 0, c, 0, 0]

>> M = sym([1 1 1; 3 1 0; 4 2 1]);
>> diag(M) % extrakce hl. diagonály
ans =
1
1
1
```

# Symbolické řešení rovnic (i soustav rovnic) - *solve*

Symbolická funkce `solve` hledá řešení rovnice (nebo soustavy rovnic). Možnosti volání:

- **řešení (ne)lineární rovnice f(*x*)=0**:
  `r = solve(eq)` ... řeší rovnici `eq` vzhledem k implicitní symbolické proměnné (viz `symvar`),
  `r = solve(eq, var)` ... řeší rovnici `eq` vzhledem k symbolické proměnné `var`.
  Výstup bude obsahovat nalezená řešení (v případě nelineární rovnice může existovat více řešení).
- **řešení soustavy *n* (ne)lineárních rovnic o *n* proměnných**:
  `solve(eq1,eq2,...,eqn)` ... řeší soustavu *n* rovnic `eq1=0` až `eqn=0` vzhledem k implicitním *n* symbolickým proměnným (viz`symvar`),
  `solve(eq1,eq2,...,eqn, var1,var2,...,varn)` ... řeší soustavu *n* rovnic `eq1=0` až `eqn=0` vzhledem k proměnným `var1` až `varn`.
  Pokud odebíráme 1 výstup, bude to struktura obsahující jednotlivá řešení (*n*položek pojmenovaných podle proměnných). Pokud odebíráme *n* výstupů, jsou výsledky nejprve abecedně seřazeny a přiřazeny do těchto výstupů. V případě více řešení jsou položky struktury nebo výstupy vektorem.

Ve všech případech se každá rovnice `eq` zadává jako symbolický výraz (tehdy se řeší `eq=0`) nebo jako řetězec (je-li bez rovnítka, tak se řeší `'eq=0'`; je-li s rovnítkem, tak se řeší dle zadáné pravé strany).

*Poznámka:* pokud se nepodaří nalézt symbolické řešení rovnice (resp. soustavy rovnic), tak funkce `solve` vrátí řešení numerické (a měla by generovat varování).

V případě, kdy hledáme kořeny polynomů, tak by funkce měla vrátit VŠECHNY kořeny. U transcendentních rovnic nalezne jen reálné kořeny - zde totiž většinou musí hledat numericky.

**Příklady**:

```
>> syms a b c x;
>> solve('a*x^2 + b*x + c') % chceme obecné řešení kvadratické rovnice
ans =
-(b + (b^2 - 4*a*c)^(1/2))/(2*a)
-(b - (b^2 - 4*a*c)^(1/2))/(2*a)
% zkuste pretty(ans)

>> solve('a*x^2 + b*x + c','b') % řešení lineární rovnice v proměnné 'b'
ans =
-(a*x^2 + c)/x

% SOUSTAVA 2 lineárních rovnic (nehomogenní):
>> S = solve('x + y  = 1', 'x - 11*y = 5');
>> S.x, S.y % výpis řešení (struktura S)
S.x =
4/3

S.y =
-1/3
% Poznamka: kdyby soustava byla homogenni, mohli bychom pouzit funkci 'null'.

% SOUSTAVA 3 nelineárních rovnic:
>> syms a u v;
>> A = solve('a*u^2 + v^2', 'u - v = 1', 'a^2 - 5*a + 6')
A =
    a: [4x1 sym]
    u: [4x1 sym]
    v: [4x1 sym]
>> Aa = A.a, Au = A.u, Av = A.v % položky struktury do proměnných
Aa =
 2
 3
 3
 2
Au =
 1/3 + (2^(1/2)*i)/3
 1/4 + (3^(1/2)*i)/4
 1/4 - (3^(1/2)*i)/4
 1/3 - (2^(1/2)*i)/3
Av =
 - 2/3 + (2^(1/2)*i)/3
 - 3/4 + (3^(1/2)*i)/4
 - 3/4 - (3^(1/2)*i)/4
 - 2/3 - (2^(1/2)*i)/3
% soustava má tedy 4 možná řešení - kontrola 3. možnosti:
>> double(subs([a*u^2+v^2, u-v, a^2-5*a+6], {a,u,v}, {Aa(3),Au(3),Av(3)}))
ans =
0.0000   1.0000   0 % OK, přesně mělo vyjít 0,1,0
```