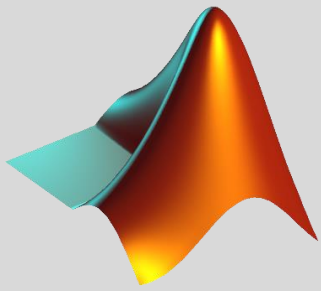
1. ČÁST



Matlab

1. Úvod
2. Režimy práce
3. Operace a použití čísel, vektorů a matic
4. Funkce
5. Pole
6. Cykly a větvení scriptu
7. 2D grafika a objekty
8. 3D grafika
9. Datové soubory
10. Toolbox symbolické matematiky
11. Numerické řešení nelineárních rovnic
12. Numerické řešení diferenciálních rovnic
13. Řešení soustav rovnic

1.1. Úvod

MATLAB (zkratka pro MAtrix LABoratory) je software pro numerickou analýzu a výpočty společnosti The Mathworks, Inc. Časem se vyvinul v programové prostředí, s jehož pomocí lze provádět nepřebernou řadu matematických operací, technických měření, statistických analýz, grafiky a podobně. Pomocí řady doplňujících knihoven, které se v Matlabu nazývají toolboxy, se jeho možnosti rozšiřují prakticky do všech možných technických aplikací. Uživatel zde může provádět získávání, analýzu, optimalizaci, vizualizaci dat a na jejich základě modelovat děje v různých, nejen technických oborech.

Tato část práce si nedává za cíl úplný popis všech funkcí Matlabu. Popíšeme zde pouze ty funkce, které jsou důležité pro další části této práce. Pro úplný popis možností Matlabu bych čtenáře odkázal na použitou literaturu.

Nejdůležitější věc pro úspěšné zvládnutí Matlabu je přijmout myšlenku, že vše je matice komplexních čísel, která se skládá z řádků a sloupců. Pokud má matice pouze jeden řádek, říkáme jí vektor. Skalár je také matice, která má právě jeden řádek a právě jeden sloupec.

1.2. Režimy práce v Matlabu

Matlab se skládá z velkého počtu funkcí, které jsou dostupné v knihovnách. To znamená, že společné funkce u různých úkolů (např. řešení soustavy lineárních rovnic), můžeme volat jednoduchou funkcí. Je zde rozsáhlá podpora grafiky, která umožní vizualizaci výsledků s použitím několika málo příkazů. S každým numerickým objektem Matlab zachází jako s polem s dvojnásobnou přesností. Orientace v programech napsaných v Matlabu, kterým se říká m-file, je velice jednoduchá a přehledná. Nejčastěji používanými režimy práce v Matlabu jsou dialogový, programový a grafický.

Příklady příkazů:

a=2:25;
5+6
*b=0:pi/180:2*pi;*
c=sin((pi/2)-0.2)
D=[1,2;3 4]
log(2)/(2^3)
E=D'
who
whos
clear
clear all
help
help elfun
clc
pi
eps
i; j;
sin(x)
abs(cos(x))
inv(D)
*h=inv(D')*3.14*
f=sqrt(tan(x))
ans

Dialogový režim

- je přístupný v okně *command window*
- zde zapsané příkazy se vykonávají ihned
- přiřazovací příkaz je realizován pomocí rovnítka =
- do proměnné *ans* se ukládají automaticky výsledky výpočtu, pokud nejsou nazvány jinak
- pokud chceme potlačit tisk výsledků (hlavně u vektorů, které mají velký počet prvků), napíšeme za příkaz středník
- v tomto režimu pracuje Matlab podobně jako inteligentní kalkulačka
- argumenty funkcí se zadávají do kulatých závorek
- lze generovat vektory a matice, řádky se oddělují středníkem, sloupce čárkou nebo mezerníkem, pokud pracujeme s prvky matice, používáme hranaté závorky
- pro elementární operace se používají znaky: *+,-,*,/,^* (umocňování)
- k transponování matic a vektorů používáme apostrof
- použité proměnné se zachovávají v paměti (vidíme je v okně *workspace*), lze je vypsat příkazem *who*, s jejich vlastnostmi příkazem *whos*
- proměnné lze mazat buď jednotlivě příkazem *clear název_proměnné* nebo všechny příkazem *clear all*

	• Matlab má velice propracovaný systém nápovědy zadáním příkazu *help* s mezerou a názvem tématu

	Programový režim
	• pracuje se zvláštním editorem, který se jmenuje *debugger*
	• při psaní skriptu je vhodné používat komentáře, které se označí znakem *%*, vše, co nalezneme za znakem *%* (procento) do konce řádku je považováno za komentář a je Matlabem ignorováno
	• jednotlivé příkazy lze oddělovat čárkou nebo středníkem
	• programový režim spustíme následujícím postupem:
	  ○ kliknutím na ikonu  v nabídce
	  ○ příkazem menu *New/ Script*
	  ○ *dvojklikem myši a otevřením již existujícího m-file*
	• soubor je nutné uložit na disk a pojmenovat dle obvyklých pravidel
	• m-file je možno spustit z okna *Command window* zadáním názvu souboru
	• je užitečné nastavit aktuální cestu v řádku Path Browser

Příklady příkazů: *figure(1)* *close* *close all* *plot(x,y)* *plot(a,b,'lz',a,c,'k')* *plot(x,x.^2)* *help plot* *grid on* *grid off* *hold on* *hold off* *axis* *axis equal* *axis square* *axis tight* *help axis* *xlabel* *ylabel* *title* *legend*	**Grafický režim** • používá samostatné grafické okno - *Figure* • příkazem *figure* lze otevřít nové okno • příkazem *close* lze zavřít poslední aktivní grafické okno • na začátku m-file je vhodné zavřít všechna dříve použitá grafická okna příkazem *close all* • pro základní zobrazení používáme příkaz *plot(a,b)*, kde *a* představuje vektor definičního oboru a *b* vektor oboru hodnot • příkaz *plot* lze použít také ve tvaru *plot(a,b,'lz'),* kde *l* představuje barvu čáry a *z* její typ • podrobnější informace o příkazu *plot* lze získat v nápovědě po zadání příkazu *help plot* • pro zobrazení pomocného měřítka můžeme použít příkaz *grid on* resp. *grid off* • zobrazení více křivek do jednoho grafu lze realizovat příkazem *hold on*, resp. zakázat příkazem *hold off* • pro změnu definičního oboru a oboru hodnot použijeme příkaz *axis ([xmin,xmax,ymin,ymax])* • popisky os lze doplnit příkazem *xlabel('text')*, *ylabel('text')* • titulek grafu lze doplnit příkazem *title('text')* • legendu grafu lze doplnit příkazem *legend('text')*

1.3. Datové typy, proměnné, operace a funkce čísel, vektorů a matic

Nejčastěji používané typy nebo třídy dat v Matlabu jsou *double*, *char* a *logic*. Všechny tyto typy jsou považovány v Matlabu za pole (*array*). Numerické objekty patří do třídy *double*, které reprezentuje pole s dvojnásobnou přesností. S čísly je zacházeno jako s polem typu 1x1. Prvky typu *char* jsou pole řetězců (posloupnost písmen). Logické typy obsahují pole, které obsahují buď *0* (*false*) nebo *1* (*true*).

Další důležitá třída, v Matlabu jedinečná (pokud porovnáváme i jiná programová prostředí), je *function_handle*. Obsahuje informace vyžadované pro nalezení a provedení dané funkce. Jméno *function_handle* se skládá ze znaku *@*, za kterým následuje závorka s uvedenými vstupními argumenty a vyjádření funkce; například *@(x) 2*cos(2*x+0.5)*. F*unction_handle* se používá jako vstupní argument pro volání funkce. Například předpokládejme, že máme v Matlabu nakreslit funkci 2cos(2x+0,5) na intervalu od 0 do 2π. Skript by mohl vypadat například takto:

```
F=@(x) 2*cos(2*x+0.5);   %function_handle
x=-2*pi:0.01:2*pi;   %deklarace definičního oboru
plot(x,F(x),'r').   %volání funkce
```

Další typy dat, se kterými můžeme v Matlabu pracovat, jsou *sparse* (řídké matice), *inline* (objekty) a *struct* (strukturované pole). Uživatel si také může definovat vlastní typy dat. Datový typ objektu lze vypsat pomocí příkazu *class*.

Název proměnné v Matlabu musí začínat písmenem (Matlab rozlišuje malá a velká písmena) a dále můžeme použít písmena i číslice. Délka názvu je neomezená, Matlab ale bere v úvahu pouze prvních 63 znaků. Lokální proměnné, které Matlab rozpozná automaticky, nejsou přístupné v jiných částech programu. Globální proměnné musí uživatel deklarovat pomocí příkazu *global* a může je sdílet se všemi částmi programu.

Matlab operuje s několika implementovanými konstantami a proměnnými. Nejdůležitější jsou: *ans* (automatické jméno výsledku), *eps* (nejmenší číslo pro které platí: $1 + eps > 1$), *inf* (nekonečno), *NaN* (nedefinovaná hodnota), *i* nebo *j* ($\sqrt{-1}$), *pi* (π), *realmin* (nejmenší použitelné kladné číslo) a *realmax* (největší použitelné kladné číslo).

Pole vytvoříme zadáním jeho prvků do hranatých závorek. Prvky oddělujeme mezerami a řádky oddělujeme středníkem nebo klávesou Enter. Matlab rozlišuje řádkový a sloupcový vektor. Například b=[1 2 3] je řádkový vektor, b=[1; 2; 3;] je sloupcový vektor a b=[1 2 3]' je transponovaný řádkový vektor. Apostrof je operátor pro transponování proměnných.

Přístup k prvkům pole se provádí pomocí kulatých závorek. Například definujeme matici *A=[8 1 6; 3 5 7; 4 9 2],* pak *A(2,3)* je 7, *A(:,2)* je druhý sloupeček, *A(2:3,2:3)* je submatice řádu 2x2 *[5 7; 9 2]* a *A(7)* je 6 (počítá se dolů po sloupcích).

Pole buněk je posloupnost libovolných objektů. Může být vytvořeno výčtem objektů ve složených závorkách. Například příkazem *C={[1 2 3],'text', 2-3i}*. Jeho výpis příkazem *celldisp(C).* Přístup k jednotlivým položkám realizujeme například takto: *C{1}* nebo *C{1}(2)*.

Řetězec je posloupnost znaků vytvořený jejich zápisem mezi apostrofy. Pracovat s nimi můžeme mimo jiné i podle následujících příkladů.

```
r1='dobry'; r2=' den';

r3=strcat(r1,r2)     (výsledek: r3=dobry den)

r4=r3(7:9)           (výsledek: r4=den)
```

Příklady příkazů:

```
A=[1 2 3;4 3 2];
B=[4 5 6;2 1 0];
A+B
ans =
   5   7   9
   6   4   2
A*B'
ans =
32   4
43   11
A.*B
ans =
4   10   18
8   3   0
A>B
ans =
0   0   0
1   1   1
(A>B)|(B>7)
ans =
0   0   0
1   1   1
```

Operátory

Operátory dělíme na aritmetické, relační a logické.

- Aritmetické
  - **Maticové**
    + (sčítání), - (odčítání), * (násobení), / (dělení zprava), \ (dělení zleva), ^ (umocňování), ‘ (transponování)
  - **Vektorové** - provádí se po jednotlivých prvcích
    *.** (násobení), *./* (dělení), *.^* (umocňování)
- **Relační** – výsledkem je buď 1 (true, pravda) nebo 0 (false, nepravda)
  < (menší), > (větší), <= (menší nebo rovno), >= (větší nebo rovno), == (je rovno), ~= (není rovno)
- **Logick**é
  & (a zároveň), | (nebo), ~ (negace)

MATLAB

1.4. Funkce

Příklady příkazů:

```
z1=exp(-2*t)
z2=log(t)
z3=log10(t)
z4=cosh(t)
z5=acos(t)
z6=acoth(t)
det(A)
inv(A)
sum(A)
min(min(A))
x=inv(A)*b
real(b)
round(c)
size(A)
length(B)
```

Základní skupiny funkcí v Matlabu jsou skalární, vektorové, maticové a uživatelské.

- **Skalární**
  tyto funkce se aplikují na každý prvek matice, patří sem běžné funkce dostupné z nápovědy help elfun, např. sin(x)
- **Vektorové**
  tyto převážně statistické funkce se aplikují na každý sloupec matice, patří sem například funkce sum(A)
- **Maticové**
  tyto funkce se aplikují na celou matici, patří sem běžné funkce dostupné z nápovědy help matfun, elmat, např. inv(A)
- **Vytvořené uživatelem**
  viz. následující řádek tabulky

Příklady příkazů:

Function
F=soucet(a,b,N)
h=(b-a)/N;
f=cos(a:h:b);
*F=sum(f(1:N))*h;*

a=0; b=10;N=100;
F=soucet(a,b,N)

Funkce vytvořené uživatelem

Funkce je vlastně pojmenovaná posloupnost příkazů. Její syntaxe je: *function[výstupní_proměnné]=jméno(vstupní_proměnné)*

- **jméno**
  jednoznačný název, pod tímto názvem musíme funkci uložit jako soubor do aktuálního adresáře s příponou *.m, potom je funkce přístupná i v ostatních částech programu
- **vstupní_proměnné**
  seznam vstupních parametrů v kulatých závorkách oddělené čárkou
- **výstupní_proměnné**
  seznam výstupních parametrů v hranatých závorkách oddělené čárkou

Vlastní tělo funkce může začínat komentářem. Komentář začíná znakem *%* a končí s koncem řádku, lze jej vyvolat příkazem *help jmeno_funkce.*

- **Inline funkce**
  další možnost, jak deklarovat funkci v Matlabu je využitím inline objektu z řetězce, který vyjadřuje funkci. Příkaz může vypadat takto: *function_name=inline('výraz', 'argument1', 'argument2', …)*

1.5. Pole, matice

Příklady příkazů:

P=[0 1.5 2.5 4];
Q=0:0.01:1;
R=linspace(0,1,9);
S=logspace(0,2,10);
T=zeros(m,n);
U=ones(m,n);
V=eye(m);
W=rand(2,3);

Pro základní práci s polem používáme tyto příkazy:

- pole zadané výčtem všech prvků do *hranatých závorek*
  např. *P=[0 1.5 2.5 4]*
- ekvidistantní pole prvků pomocí dvojtečky
  Q=první_prvek:přírůstek:poslední_prvek
  např. *Q=0:0.01:1;*
- ekvidistantní pole n prvků pomocí funkce linspace
  R=linspace(první_prvek,poslední_prvek,n)
  např. *R=linspace(0,1,9)*
- logaritmické pole n prvků pomocí funkce logspace
  S=logspace(první_exponent, poslední_exponent,n)
  např. *S=logspace(0,2,10)*
- nulové pole m řádků a n sloupečků
  např. *T=zeros(m,n)*
- pole samých jedniček m řádků a n sloupečků
  např. *U=ones(m,n)*
- jednotková matice m řádků
  např. *V=eye(m)*
- matice m řádků a n sloupečků náhodných čísel od 0 do 1
  např. *W=rand(2,3)*

Příklady příkazů:	**Příklady maticových funkcí** Matlab obsahuje velký počet funkcí pro práci s maticemi, uvedeme zde pouze některé. • počet prvků *n* vektoru *v* může být zjištěno funkcí *length* např. *n=length(v);* • řád matice M [m,n] může být zjištěn funkcí size např. *[m,n]=size(M);* • seřazení prvků vektoru v do matice M řádu [m,n] podle velikosti např. *M=reshape(v,m,n);* • skalární součin sk dvou vektorů u, v vypočítáme funkcí dot např. sk=*dot(u,v);* • součet s a součin ss prvků vektoru v vypočítáme funkcí sum resp. prod např. s=*sum(v); ss=prod(v);* • součet sm a součin ssm prvků matice M vypočítáme funkcí sum resp. prod výsledkem je vektor sloupcových součtů resp. součinů např. sm=*sum(M); ssm=prod(M);* • vektorový součin vs vektorů u, v vypočítáme funkcí cross např. vs=*cross(u,v);*
n=length(v); *[m,n]=size(M);* *M=reshape(v,m,n);* *sk=dot(u,v);* *s=sum(v);* *ss=prod(v);* *sm=sum(M);* *ssm=prod(M);* *vs=cross(u,v);*	

MATLAB	**1.6. Větvení, rozhodování a cykly**
Příklady příkazů: *if A>10* *B=A+1;* *else* *B=A+2;* *end*	**Větvení a rozhodování ve scriptu** Pokud nepoužijeme ve scriptu větvení, každý příkaz se provede právě jednou v uvedeném pořadí, pokud ano, každý příkaz uvnitř větvení se provede nejvýše jednou. Větvení vždy začíná příkazem *if*, za kterým následuje logická podmínka a končí příkazem *end*. Část scriptu mezi příkazy *if* a *end* nazýváme tělo, které může navíc obsahovat i příkazy *else*. Tělo scriptu nazýváme pozitivní, pokud mezi příkazy *if* a *end* chybí příkaz *else* a negativní, pokud se příkazy nacházejí mezi příkazy *else* a *end*. Příkazem *return* můžeme předčasně z větvení vyskočit. • Syntaxe úplné podmínky vypadá takto: *if podmínka* *% příkazy pozitivní části* *else* *%příkazy negativní části* *end* • Syntaxe neúplné podmínky vypadá takto: *if podmínka* *% příkazy pozitivní části* *end*

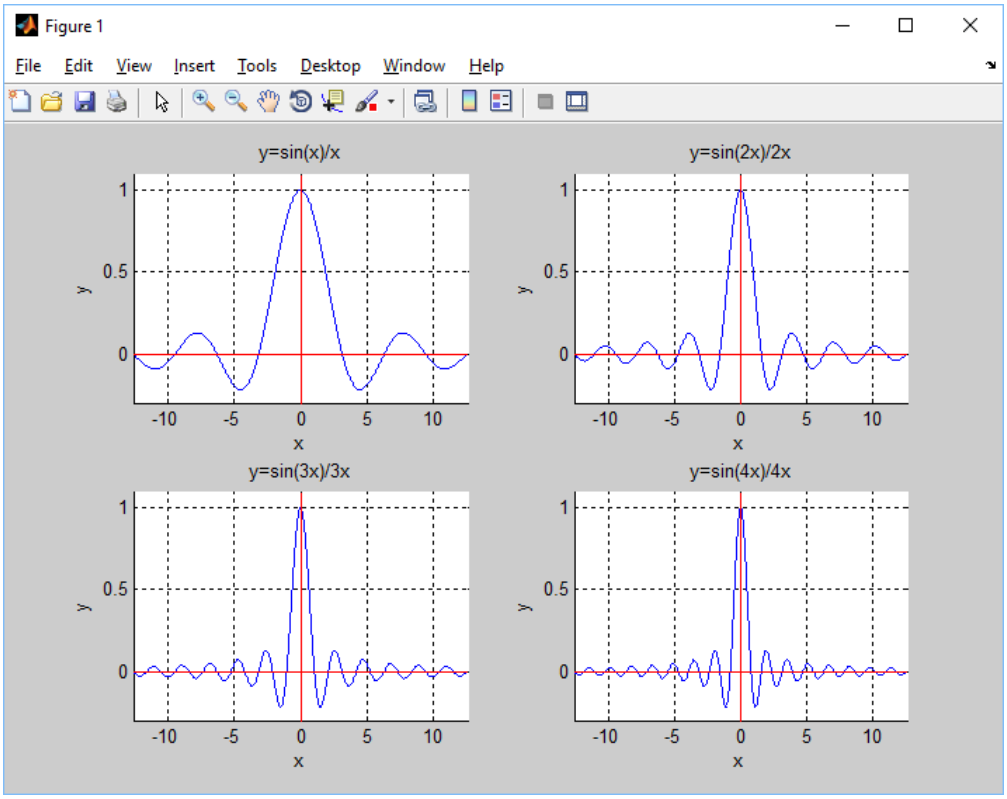
Příklady příkazů:	**Cyklus while**
while a~=b *if a >b* *a= a+b;* *else* *b=a-b;* *end* *end*	Cyklus *while* je cyklus s podmínkou na začátku. Pokud je podmínka splněna, provedou se příkazy uvnitř cyklu. Začátek cyklu je realizován příkazem *while*, za kterým následuje logická podmínka. Potom následuje tělo cyklu a končí příkazem *end*. Příkazem *return* můžeme předčasně z cyklu vyskočit. • Syntaxe cyklu while vypadá takto: *while logická podmínka* *% tělo cyklu* *end*
Příklady příkazů:	**Cyklus for**
for n=1:10 *a= a+5;* *end*	Cyklus *for* provede dané příkazy pro všechny prvky řádkového vektoru *v*, tedy pro hodnoty od 1 do length(v). Bohužel v Matlabu nelze indexovat od 0, vždy musíme začínat 1. Začátek cyklu je realizován příkazem for se jménem proměnné typu vektor, pak následuje tělo cyklu, jehož příkazy se vykonají pro všechny prvky vektoru a nakonec je příkaz end. Příkazem *return* můžeme předčasně z cyklu vyskočit. • Syntaxe cyklu for vypadá takto: *for proměnná=1:počet opakování cyklu* *% tělo cyklu* *end*

MATLAB	**1.7. 2D grafika a objekty**
Příklady příkazů: *figure;* set (gcf,'Units', 'Normal','Position', [0.1,0.2,0.3,0.4]); *s = axes('Position',* *[*0.1,0.2,0.3,0.4*])* set (s,'Color', [1 0 1]) *subplot (4,2,2);* *fplot(@fce,[1 2])*	Obecný příkaz v Matlabu pro práci s daným objektem je *název = příkaz (vlastnost,hodnota).* Nastavení vlastnosti objektu lze příkazem *set (objekt, vlastnost, hodnota)* a nastavovat lze najednou i více vlastností. Příkaz pro otevření grafického okna je *figure*. Používané jednotky jsou v režimu *Normal*, což je interval $\langle 0;1\rangle x\langle 0;1\rangle$, kde pozice $\langle 0;0\rangle$ je levý dolní roh obrazovky a $\langle 1;1\rangle$ její pravý horní roh. Pozici a velikost aktuálního grafického okna nastavujeme příkazem: set (gcf,'Units','Normal','Position',[x_1, y_1, x_2, y_2]), kde x_1, y_1 je pozice levého dolního rohu grafického okna od levého dolního rohu obrazovky a x_2, y_2 šířka a výška grafického okna ve stejných jednotkách. Příkazem *souradnice = axes('Position', [x_1, y_1, x_2, y_2])* lze umístit do grafického okna souřadný systém. Existující vlastnosti objektu s jejich aktuálními hodnotami vypíše Matlab po zadání příkazu *get(objekt).* Barvy nastavujeme jako kombinaci RGB v rozsahu $\langle 0;1\rangle$. Nejčastěji používané příkazy pro 2D grafiku byly uvedeny už v části 1.2. Režimy práce v Matlabu v části grafický režim. Přesto ještě některé doplníme, hlavně příkaz *subplot (parametr1, parametr2, parametr3)*, který dokáže rozdělit grafické okno na počet částí daných číslem parametr3, do řádků, jejichž počet je dám číslem parametr1 a do sloupců, jejichž počet je dán číslem parametr2.

Jako příklady pro práci s 2D grafikou uvedeme následující dva skripty s vysvětlujícím komentářem a výslednými grafy viz Obr.1 a Obr.2.

Příklad 1:

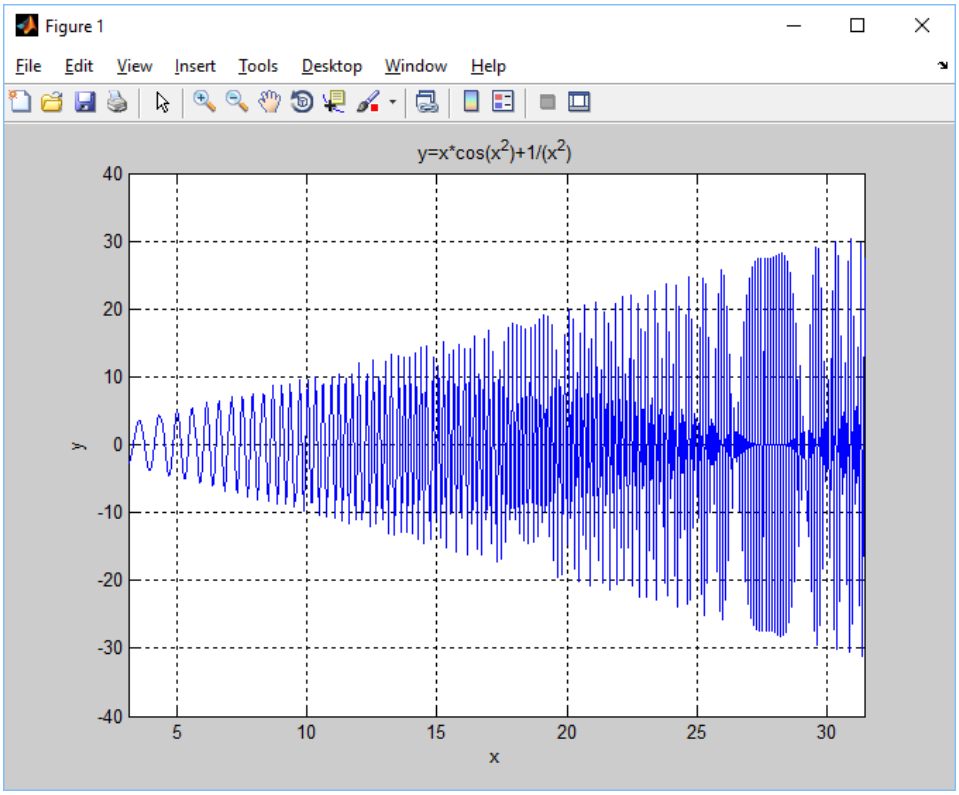
```
clc;close;clear;          % vymazání Command Window, všech grafických oken a všech proměnných
x=-4*pi:0.001:4*pi;       % definiční obor funkcí
y1=sin(x)./(x);           % obor hodnot první funkce
y2=sin(2*x)./(2*x);       % obor hodnot druhé funkce
y3=sin(3*x)./(3*x);       % obor hodnot třetí funkce
y4=sin(4*x)./(4*x);       % obor hodnot čtvrté funkce

subplot(2,2,1)            % obrázek 1
grid on; hold on;         % zapnutí mřížky a přikreslování
   plot(x,y1)             % graf první funkce
   xlabel('x');ylabel('y') % popisky os
   title('y=sin(x)/x')    % titulek grafu 1
   osy=[-4*pi,4*pi,-0.3,1.1]; % rozsah na osách
   axis(osy)
   plot([-4*pi,4*pi],[0,0],'r') % osa x jako úsečka
   plot([0,0],[-0.5,1.1],'r') % osa y jako úsečka

 subplot(2,2,2)
 grid on;hold on;
    plot(x,y2)
    xlabel('x');ylabel('y')
    title('y=sin(2x)/2x')
    osy=[-4*pi,4*pi,-0.3,1.1];
    axis(osy)
    plot([-4*pi,4*pi],[0,0],'r')
    plot([0,0],[-0.5,1.1],'r')

 subplot(2,2,3)
 grid on;hold on;
    plot(x,y3)
    xlabel('x');ylabel('y')
    title('y=sin(3x)/3x')
    osy=[-4*pi,4*pi,-0.3,1.1];
    axis(osy)
    plot([-4*pi,4*pi],[0,0],'r')
    plot([0,0],[-0.5,1.1],'r')

 subplot(2,2,4)
 grid on;hold on;
    plot(x,y4)
    xlabel('x');ylabel('y')
    title('y=sin(4x)/4x')
    osy=[-4*pi,4*pi,-0.3,1.1];
    axis(osy)
    plot([-4*pi,4*pi],[0,0],'r')
    plot([0,0],[-0.5,1.1],'r')
```



Obr. 1

Příklad 2:

```
function y=fce(x)                    % uložená funkce
y=(x).*cos(x.^2)+1./(x.^2);

fplot(@fce,[pi 10*pi])               %graf uložené funkce na definičním oboru od π do 10 π
grid on
xlabel('x');ylabel('y')
title('y=x*cos(x^2)+1/(x^2)')
```

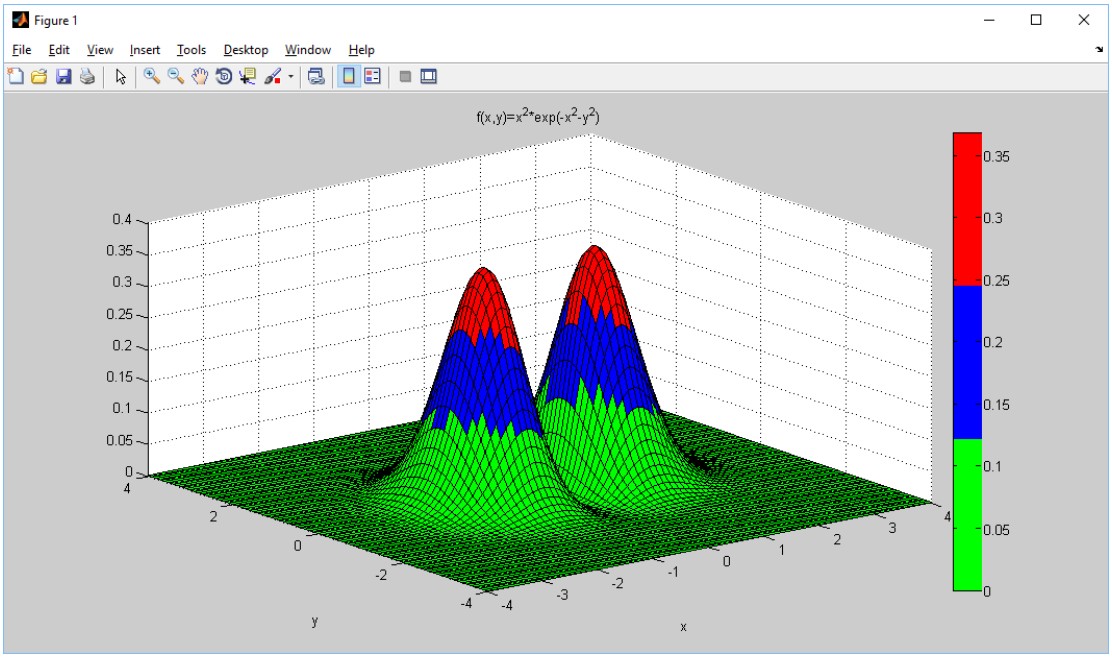


Obr. 2

MATLAB	**1.8. 3D grafika**
Příklady příkazů: *[x,y]=meshgrid (-4:0.1:4);* *surf(x,y,z);* *mesh(x,y,z);* *contour(x,y,z)* *colormap(M)* *colorbar*	Režim pro 3D grafiku se používá pro zobrazení funkcí dvou proměnných $z = f(x, z)$. Definiční obor jednotlivých bodů se vygeneruje jako mřížka pomocí příkazu *meshgrid*. K zobrazení funkce lze využít více příkazů – například *plot3, mesh, meshc, surf, surfc, surfl, surface*. Právě příkaz *surface* vytvoří plochu jako objekt, se kterým pak můžeme pracovat dále, jako s jinými objekty, například měnit jeho vlastnosti. Příkazem *contour* můžeme vytvořit vrstevnice dané plochy. Dále můžeme modifikovat barvy, používáme model RGB v rozsahu $\langle 0; 1 \rangle$, pomocí příkazu *colormap* a zobrazit měřítko pomocí příkazu *colorbar*.

Jako příklady pro práci s 3D grafikou uvedeme následující skript s vysvětlujícím komentářem a výsledným grafem viz Obr.3.

```
[x,y]=meshgrid(-4:0.1:4);                          % definice mřížky pro definiční obor
z=x.^2.*exp(-x.^2-y.^2);                           % obor hodnot funkce
surf(x,y,z);                                       % graf funkce
title('f(x,y)=x^2*exp(-x^2-y^2)');xlabel('x');ylabel('y');   % titulek, popis os
M=[0 1 0;0 0 1;1 0 0];
colormap(M)                                        % definice barev
colorbar                                           % měřítko v daných barvách
```



Obr. 3

MATLAB	**1.9. Datové soubory**
Příklady příkazů: *figure;* *A=[0 1 2]; b=6;* *save data1;* *save data2 A;* *clear all;* *load data1;* *who*	V Matlabu můžeme pracovat i s daty z jiných programů. Vždy máme na výběr z několika možností, nejlepší metoda importu a exportu vždy záleží na objemu zpracovávaných dat, jejich formátu atd. Nejčastěji používané činnosti v této oblasti jsou ukládání do souboru, načtení ze souboru a import dat z Exelu. • **Ukládání do souboru** K uložení dat do souboru použijeme příkaz *save*. Možná syntaxe příkazu je například: *save jméno_souboru* – uloží celý pracovní prostor do souboru, který se bude jmenovat *jméno_souboru.mat* *save jméno_souboru prom1 prom2 … promn* – do souboru s názvem *jméno_souboru.mat* se uloží proměnné s názvem *prom1, prom2, …, promn* • **Načtení ze souboru** Načíst proměnné ze souborů s příponou *.mat realizujeme pomocí příkazu *load*

	• **Import dat z Excelu** Soubor s importovanými daty musí být v aktuálním adresáři. Soubor si otevřeme, vybereme příslušnou oblast, nastavíme názvy proměnných a pomocí tlačítka import data importujeme. Matlab nám také může vygenerovat skript na import příslušných dat.

MATLAB	**1.10. Toolbox symbolické matematiky**
Příklady příkazů: *help symbolic* *syms a b t x y;* *V=sin(x)^2+cos(x)^2* *simple(V)* *pretty(V)* *limit(sin(x)/x)* *limit(1/x,x,0,'left')* *limit(1/x,x,0,'right')* *limit(1/x+5,x,inf)* *ezplot(y,[-2 2])* *d=(a+b)^2;* *subs(d,{a,b},{1 2})* *symsum* *(1/n)^2,1,inf)* *DE='t^2*Dy+t*y=1';* *y=dsolve* *(DE,'y(1)=2')* *ezplot(y,0.01,4)* *f1=double* *(subs(y,t,2.5))*	Toolbox symbolické matematiky slouží jako nástroj pro obecné výpočty jako jsou derivace, primitivní funkce, řešení diferenciálních rovnic apod. Detailní popis všech možností a demo programů je popsán v nápovědě, po zadání příkazu *help symbolic*. Pokud chceme pracovat s toolboxem symbolické matematiky musíme vždy zavést symbolické proměnné, buď příkazem *sym('prom')* nebo *syms prom1 prom2 …* . Výrazy je možno zjednodušovat příkazem *simple(prom)*, či upravit příkazem *pretty(prom)*. Výsledek symbolického výpočtu lze převést do proměnných výpočetního prostředí Matlabu pomocí příkazu *double(symbolická_proměnná).* Pro symbolický výpočet limit použijeme příkaz *limit* v několika alternativách. Příkazy *limit(výraz)* vypočítáme limitu, kdy se symbolická proměnná blíží k nule, *limit(výraz,x,a)* vypočítáme limitu, kdy se symbolická proměnná se blíží k hodnotě a, *limit(výraz,x,a,'left')* vypočítáme limitu, kdy se symbolická proměnná se blíží k hodnotě a zleva a *limit(výraz,x,a,'right')* vypočítáme limitu, kdy se symbolická proměnná blíží k hodnotě a zprava. Grafy funkcí zobrazujeme příkazem *ezplot(prom, interval)*. Dosazení do výrazu za danou proměnnou realizujeme příkazem *subs(výraz, proměnná, hodnota)*. Pomocí příkazu *symsum* můžeme sčítat nekonečné konvergentní řady. Dále můžeme jednoduše derivovat funkce příkazem *diff(f1)*, integrovat funkce příkazem *int(f2)* a řešit diferenciální rovnice příkazem *dsolve('DifR', 'proměnná')*. Pro zápisy diferenciálních výrazů $y'\left(\frac{dy}{dx}\right)$ a $y''\left(\frac{d^2y}{dx^2}\right)$ používáme značení Dy a $D2y$. Pro symbolické řešení diferenciální rovnice použijeme příkaz *dsolve*, který má parametry danou diferenciální rovnici a počáteční podmínky. Výpočet hodnoty výsledné funkce pro daný parametr realizujeme příkazem *double*.

Symbolické řešení diferenciálních rovnic v Matlabu je velice důležité téma pro tuto práci. Proto zde ukážeme několik typických příkladů z různých technických oborů. Každý příklad je věnován jednomu základnímu typu obyčejných diferenciálních rovnic prvního a druhého řádu.

Příklad 1.

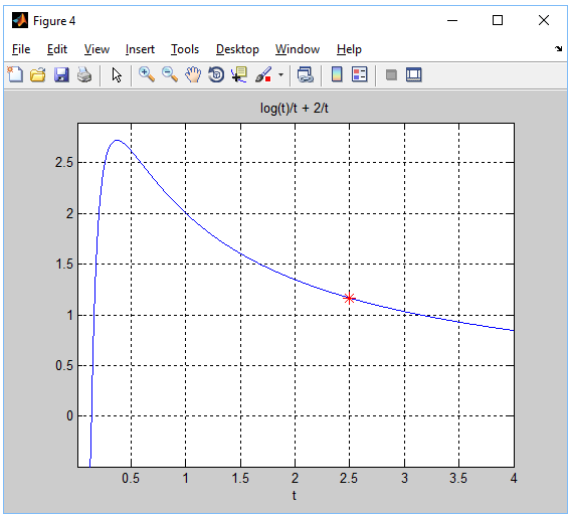
Řešte symbolicky počáteční problém pro diferenciální rovnici prvního řádu: $t^2y' + ty = 1$, s počáteční podmínkou $y(1) = 2$. Graf partikulárního řešení nakreslete v intervalu od 0,01 do 4. Určete $y(2{,}5)$.

Script:

```
syms y t;                          % zavedení symbolických proměnných y, t
DE='t^2*Dy+t*y=1';                 % určení diferenciální rovnice
y=dsolve(DE,'y(1)=2')              % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0.01,4)                   % vykreslení partikulárního řešení
hold on; grid on;                  % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,2.5))           % výpočet hodnoty partikulárního řešení pro t=2,5
plot(2.5,f1,'*r','MarkerSize',10)  % zakreslení hodnoty partikulárního řešení pro t=2,5 do grafu
```

Výsledek z Matlabu:

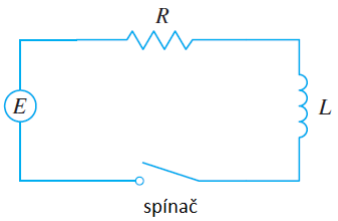
```
y =
log(t)/t + 2/t
f1 =
   1.1665
```



Obr. 4
Partikulární řešení příkladu 1

Příklad 2.

Uvažujme elektrický obvod se spínačem, rezistencí 12 Ω a induktancí 4 H. Baterie dává konstantní napětí 60 V. Spínač zapneme v čase 0 s. Proud v čase 0 s je 0 A.
a) nalezněte funkci, která vyjadřuje závislost proudu na čase I(t),
b) hodnotu proudu v čase 1 s,
c) z grafu odhadněte limitní hodnotu ustáleného proudu (pro t jdoucí k nekonečnu).



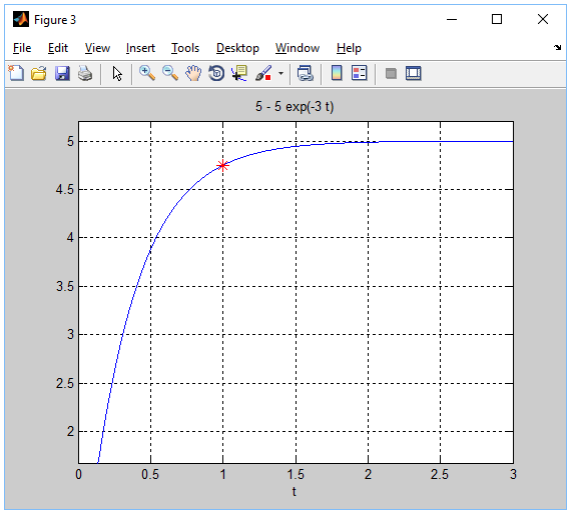
Tuto situaci popíšeme obyčejnou diferenciální rovnicí prvního řádu ve tvaru: $LI' + RI = R(t)$. Po dosazení a úpravě dostaneme počáteční problém pro diferenciální rovnici $y' + 3y = 15, y(0) = 0$, který budeme řešit pomocí následujícího scriptu:

Script:

```
syms y t;                       % zavedení symbolických proměnných y, t
DE='Dy+3*y=15';                 % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0')           % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0,3)                   % vykreslení partikulárního řešení
hold on; grid on;               % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,1))          % výpočet hodnoty partikulárního řešení pro t=2,5
plot(1,f1,'*r','MarkerSize',10) % zakreslení hodnoty partikulárního řešení pro t=2,5 do grafu
```

Výsledek z Matlabu:

```
y =

 5 - 5*exp(-3*t)


f1 =

    4.7511
```



Obr. 5
Partikulární řešení příkladu 2, ustálená hodnota proudu bude 5A

Příklad 3.

Uvažujme elektrický obvod z příkladu 2. Změníme ovšem zdroj proudu na generátor, který generuje proud dle funkce $E(t) = 60\sin(30t)$. Proud v čase 0 s je 0 A.
a) nalezněte závislost proudu na čase I(t),
b) proud v čase 2 s,
c) nakreslete průběh proudu v intervalu od 0 do 3 sekund.

Upravíme počáteční problém pro diferenciální rovnici z příkladu 2 na $4y' + 12y = 15\sin(30t)$, $y(0) = 0$, který budeme řešit pomocí následujícího scriptu:

Script:

```
syms y t;                       % zavedení symbolických proměnných y, t
DE='4*Dy+12*y=60*sin(30*t)';    % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0')           % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0,3)                   % vykreslení partikulárního řešení
hold on; grid on;               % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,2))          % výpočet hodnoty partikulárního řešení pro t=2
plot(2,f1,'*r','MarkerSize',10) % zakreslení hodnoty partikulárního řešení pro t=2 do grafu
```

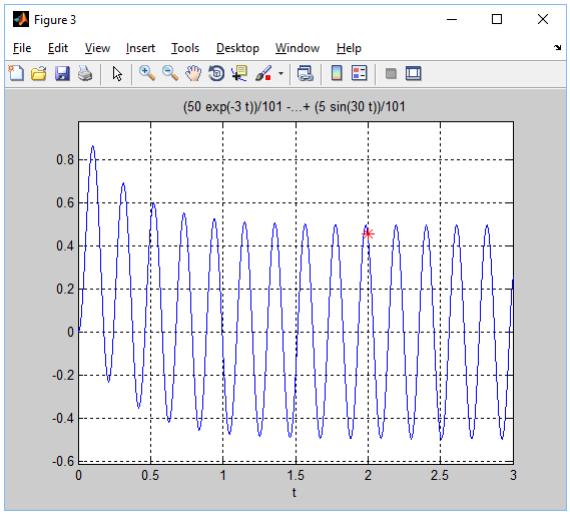
Výsledek z Matlabu:

y =

*(50*exp(-3*t))/101 -*
*(50*cos(30*t))/101+ (5*sin(30*t))/101*

f1 =

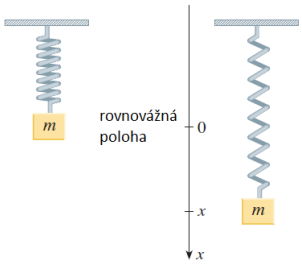
0.4576



Obr. 6
Partikulární řešení příkladu 3

Příklad 4.

Na pružině je zavěšeno závaží o hmotnosti 2 kg. Délka pružiny v klidu je 0,5 m. Pro natažení pružiny na délku 0,7 m potřebujeme sílu 25,6 N. Po natažení do uvedené délky pružinu uvolníme, počáteční rychlost je 0 m/s. Nalezněte polohu závaží v čase t = 4 s. Nakreslete graf závislosti výchylky na čase v intervalu od 0 do 5 s.



Tuto situaci popíšeme obyčejnou diferenciální rovnicí druhého řádu ve tvaru $mx'' + cx' + kx = F(t)$. Po dosazení a úpravě dostaneme počáteční problém pro diferenciální rovnici $y'' + 64y = 0, y(0) = 0{,}2,\ y'(0) = 0$, který budeme řešit pomocí následujícího scriptu:

Script:

```
syms y t;                          % zavedení symbolických proměnných y, t
DE='D2y+64*y=0';                   % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0.2', 'Dy(0)=0') % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0,5)                      % vykreslení partikulárního řešení
hold on; grid on;                  % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,4))             % výpočet hodnoty partikulárního řešení pro t=2,5
plot(1,f1,'*r','MarkerSize',10)    % zakreslení hodnoty partikulárního řešení pro t=2,5 do grafu
```

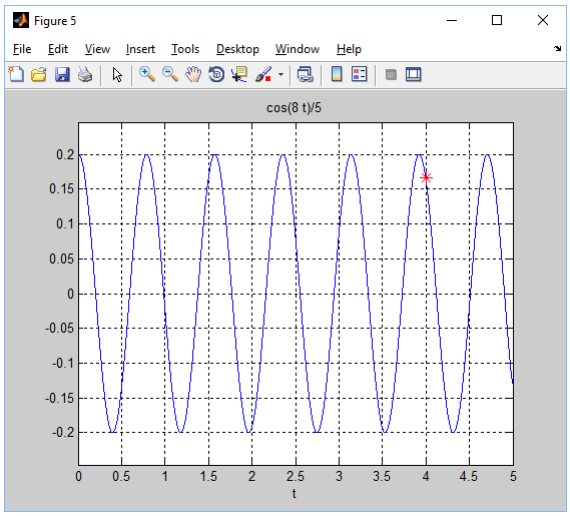
Výsledek z Matlabu:

y =

*cos(8*t)/5*

f1 =

0.1668



Obr. 7
Partikulární řešení příkladu 4

Příklad 5.

Pružinu z příkladu 4 umístíme do kapaliny s konstantou tlumení c = 40. Nalezněte pozici závaží v čase t, jestliže startuje s počáteční rychlostí 0,6 m/s. Nakreslete graf závislosti výchylky na čase v intervalu od 0 do 2 s. Nalezněte polohu závaží v čase t = 0,5 s.

Tuto situaci popíšeme obyčejnou diferenciální rovnicí druhého řádu ve tvaru $mx'' + cx' + kx = F(t)$. Po dosazení a úpravě dostaneme počáteční problém pro diferenciální rovnici $y'' + 20y' + 64y = 0,\ y(0) = 0,\ y'(0) = 0{,}6,$ který budeme modelovat pomocí následujícího scriptu:

Script:

```
syms y t;                          % zavedení symbolických proměnných y, t
DE='D2y+20*Dy+64*y=0';             % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0', 'Dy(0)=0.6') % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0,5)                      % vykreslení partikulárního řešení
hold on; grid on;                  % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,0,5))           % výpočet hodnoty partikulárního řešení pro t=2,5
plot(0,5,f1,'*r','MarkerSize',10)  % zakreslení hodnoty partikulárního řešení pro t=2,5 do grafu
```

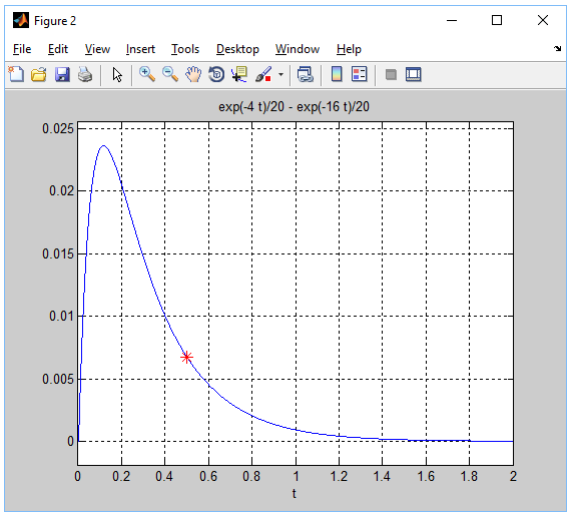
Výsledek z Matlabu:

y =

*exp(-4*t)/20 - exp(-16*t)/20*

f1 =

0.0067



Obr. 8
Partikulární řešení příkladu 5

Příklad 6.

Pružinu z příkladu 5 budeme budit silou F(t)=1000cos(2πt). Nalezněte pozici závaží v čase t, jestliže startuje s počáteční rychlostí 0,6 m/s. Nakreslete graf závislosti výchylky na čase v intervalu od 0 do 5 s. Nalezněte polohu závaží v čase t = 2,5 s.

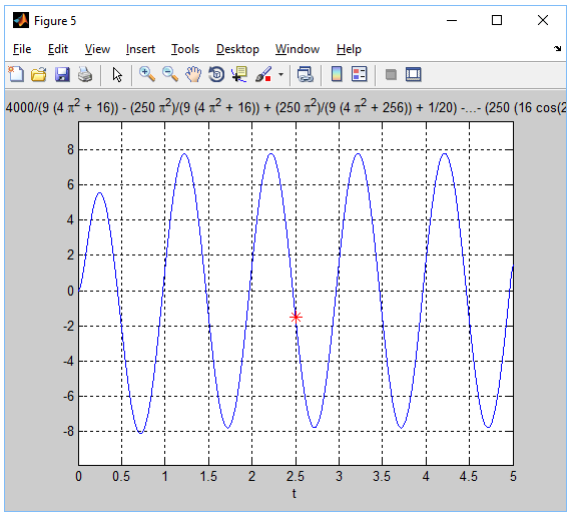
Tuto situaci popíšeme obyčejnou diferenciální rovnicí druhého řádu ve tvaru $mx'' + cx' + kx = F(t)$. Po dosazení konstant a úpravě dostaneme počáteční problém pro diferenciální rovnici $y'' + 20y' + 64y = 1000\cos(2\pi t), y(0) = 0,\ y'(0) = 0{,}6$, který budeme řešit pomocí následujícího scriptu:

Script:

```
syms y t;                                   % zavedení symbolických proměnných y, t
DE='D2y+20*Dy+64*y=1000*cos(2*pi*t)';       % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0', 'Dy(0)=0.6')          % řešení počátečního problému pro diferenciální rovnici DE
ezplot(y,0,5)                               % vykreslení partikulárního řešení
hold on; grid on;                           % přikreslování do grafu, zapnutí mřížky
f1=double(subs(y,t,2,5))                    % výpočet hodnoty partikulárního řešení pro t=2,5
plot(2,5,f1,'*r','MarkerSize',10)           % zakreslení hodnoty partikulárního řešení pro t=2,5 do grafu
```

Výsledek z Matlabu:

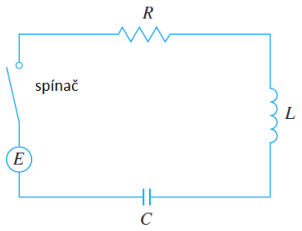
```
y =

exp(-4*t)*(16000/(9*(4*pi^2 + 256)) -
4000/(9*(4*pi^2 + 16)) -
(250*pi^2)/(9*(4*pi^2 + 16)) +
(250*pi^2)/(9*(4*pi^2 + 256)) + 1/20) -
exp(-16*t)*(4000/(9*(4*pi^2 + 256)) -
1000/(9*(4*pi^2 + 16)) -
(250*pi^2)/(9*(4*pi^2 + 16)) +
(250*pi^2)/(9*(4*pi^2 + 256)) + 1/20) +
(250*(4*cos(2*pi*t) +
2*pi*sin(2*pi*t)))/(3*(4*pi^2 + 16)) -
(250*(16*cos(2*pi*t) +
2*pi*sin(2*pi*t)))/(3*(4*pi^2 + 256))


f1 =

   -1.4962
```



Obr. 9
Partikulární řešení příkladu 6

Příklad 7.

Nalezněte náboj a proud v čase t v elektrickém obvodu jestliže R = 40 Ω, L = 1 H, C = 0,0016 F, E(t)=100cos(10t) a počáteční náboj a proud jsou oba 0. Určete číselnou hodnotu náboje i proudu v čase t = 2,5 s. Průběh proudu i náboje zakreslete v intervalu od 0 do 3 s.



Tuto situaci popíšeme obyčejnou diferenciální rovnicí druhého řádu $LQ'' + RQ' + \left(\frac{1}{C}\right)Q = E(t)$. Po dosazení odpovídajících konstant a úpravě dostaneme počáteční problém pro diferenciální rovnici $y'' + 40y' + 625y = 100\cos(10t), y(0) = 0,\ y'(0) = 0$, který budeme řešit pomocí následujícího scriptu:

```
syms y t                                % zavedení symbolických proměnných y, t
DE='D2y+40*Dy+625*y=100*cos(10*t)';     % určení diferenciální rovnice
y=dsolve(DE,'y(0)=0','Dy(0)=0')         % řešení počátečního problému pro diferenciální rovnici DE
I=diff(y)                               % derivace y (derivace náboje je proud)
f1=double(subs(y,t,2.5))                % náboj v čase 2,5s
f2=double(subs(I,t,2.5))                % proud v čase 2,5s
subplot(2,1,1)                          % vykreslení partikulárního řešení
   ezplot(y,[0,3])                      % vykreslení partikulárního řešení náboj
   hold on; grid on;                    % přikreslování do grafu, zapnutí mřížky
   plot(2.5,f1,'*r','MarkerSize',10)    % zakreslení hodnoty náboje v čase t=2,5 do grafu
subplot(2,1,2)                          % vykreslení partikulárního řešení proud
   ezplot(I,[0,3])                      % vykreslení partikulárního řešení proud
   hold on; grid on;                    % přikreslování do grafu, zapnutí mřížky
   plot(2.5,f2,'*r','MarkerSize',10)    % zakreslení hodnoty proudu v čase t=2,5 do grafu
```

Výsledek z Matlabu:

y =

*cos(15*t)*((2*cos(5*t))/51 + (10*cos(25*t))/123 - (8*sin(5*t))/51 - (8*sin(25*t))/123) + sin(15*t)*((8*cos(5*t))/51 + (8*cos(25*t))/123 + (2*sin(5*t))/51 + (10*sin(25*t))/123) - (84*cos(15*t)*exp(-20*t))/697 - (464*sin(15*t)*exp(-20*t))/2091*

I =

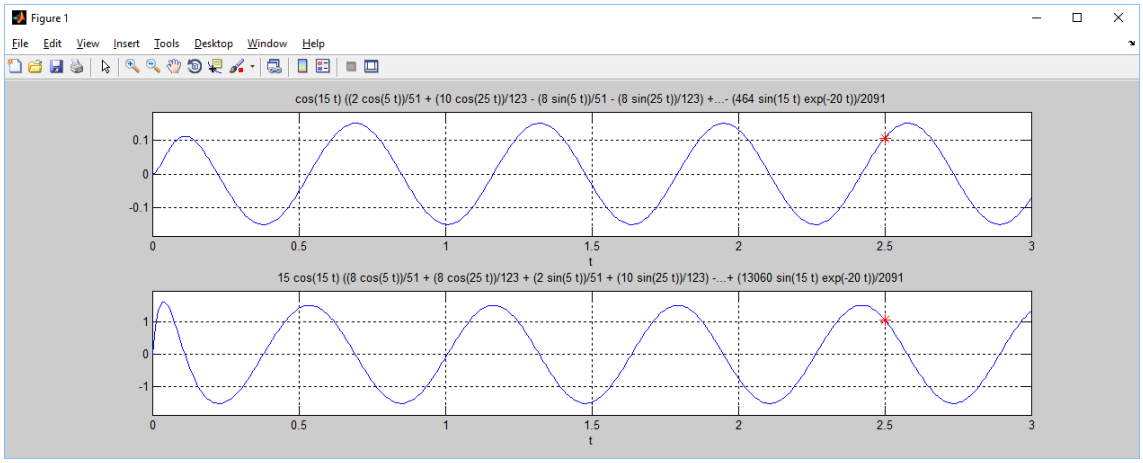
*15*cos(15*t)*((8*cos(5*t))/51 + (8*cos(25*t))/123 + (2*sin(5*t))/51 + (10*sin(25*t))/123) - cos(15*t)*((40*cos(5*t))/51 + (200*cos(25*t))/123 + (10*sin(5*t))/51 + (250*sin(25*t))/123) - 15*sin(15*t)*((2*cos(5*t))/51 + (10*cos(25*t))/123 - (8*sin(5*t))/51 - (8*sin(25*t))/123) + sin(15*t)*((10*cos(5*t))/51 + (250*cos(25*t))/123 - (40*sin(5*t))/51 - (200*sin(25*t))/123) - (640*cos(15*t)*exp(-20*t))/697 + (13060*sin(15*t)*exp(-20*t))/2091*

f1 =

0.1073

f2 =

1.0696



Obr. 10
Partikulární řešení příkladu 7

MATLAB	**1.11. Řešení nelineárních rovnic v Matlabu**
Příklady příkazů: *syms a b c x* *rce='x^2-2*x-4';* *k=solve(rce);* *kd=double(k)* *kvrce=* *'a*x^2+b*x+c=0'* *solve(kvrce,x)* *kp=roots([1 -2 -4])*	V této kapitole uvedeme vybrané metody řešení nelineárních rovnic v Matlabu. Rovnice můžeme řešit v Matlabu pomocí implementovaných funkcí. V symbolickém toolboxu využíváme příkaz *solve*. Tato funkce hledá řešení rovnice i soustav rovnic. Pokud není schopna nalézt symbolické řešení, vrátí řešení numerické. Příkaz *solve(rovnice)* vypíše řešení rovnice, která je zadaná buď řetězcem, nebo symbolickým výrazem, který se automaticky rovná nule. Pokud je v symbolickém výrazu více symbolických proměnných, musíme zadat, pro kterou z nich hledáme řešení příkazem *solve(rovnice,x).* Z dalších možností jak řešit rovnice v Matlabu uveďme například funkci *roots*, která nalezne všechny kořeny polynomu, který je dán

	vektorem koeficientů jednotlivých mocnin neznámé uvedené v sestupném pořadí. Například vektorem *v = [3 -4 0 2]* je dán polynom $3x^3 - 4x^2 + 2$. Pro numerické metody řešení rovnic si musíme vytvořit svoje vlastní funkce. Tyto metody rozdělujeme na uzavřené, u kterých musíme přesně lokalizovat kořeny (separace kořenů) a otevřené, kdy nám stačí pouze kvalitní počáteční odhad kořene. Do skupiny uzavřených metod patří například metoda půlení intervalu (bisekce), metoda regula falsi atd. Do skupiny otevřených metod patří Newtonova metoda, metoda prosté iterace, metoda sečen a další. Tyto metody jsou v technické praxi velice důležité, proto si uvedeme několik příkladů scriptů s vysvětlujícím komentářem.

Příklad 1.

Metodou půlení intervalu nalezněte kořen rovnice $10\cos(x-1) - x^2 + 2x - 1 = 0$, který leží v intervalu $\langle 2{,}3;\ 2{,}4\rangle$ s přesností $0{,}01\%$.

Script funkce:

```
function [fv,k,re,i]=bisection(fce,xd,xh,me,mi)
% bisection: Metoda hledani korene s 2 pocatecnimi odhady; uzavrena metoda
% Vstup:
% fce: Funkce jejiz koren hledame
% xd: pocatecni dolni odhad
% xh: pocatecni horni odhad
% me: maximalni relativni chyba (prednastaveno = 0.0001%)
% mi: maximalni pocet iteraci (prednastanoveno = 100)
% Vystup:
% fv: hodnota funkce v korenu
% k: koren funkce
% re: relativni chyba aproximace (%)
% i: pocet iteraci

   if nargin<3,error('minimum 3 input arguments'),end
   if nargin<4, maxm_err=0.005;end
   if nargin<5, max_iter=100;end
   if fce(xd)*fce(xh)>0,error('no sign change'),end
% prvni krok
   i = 0;
   while(1)
      xn =(xd + xh)/2;
      re=abs((xn-xh)/xn);
      if((re<me) || mi<i),break;end
      i = i + 1;
      test = fce(xd)*fce(xn);
      if test<0
         xh = xn;
      elseif test>0
         xd = xn;
      else
         re = 0;
      end
   end
      k=xn;
      fv =fce(k);
```

```
end
```

Script řešení přikladu 1:

```
clc; close; clear;                                  % vymazání Command window, grafických oken a proměnných
fce=@(x) 10*cos(x-1)-x^2+2*x-1;                     % definovaná funkce dle zadání
format long                                             % zobrazení výsledků na 15 desetinných čísel
[fv,k,re,i]=bisection(fce,2.3,2.4,0.0001,100)           % volání funkce bisection s danými parametry
```

Výsledek Matlabu:

```
fv =  -0.001604652167318        % hodnota zadané funkce v aproximaci
k =   2.379492187500000         % aproximace kořene s danou přesností
re =     8.208158909949030e-05  % relativní chyba aproximace
i =    8                        % počet aproximaci k dosažení dané přesnosti
```

Příklad 2.

Metodou Regula-Falsi nalezněte kořen rovnice $10\cos(x-1)-x^2+2x-1=0$, který leží v intervalu $\langle 2{,}3;\ 2{,}4\rangle$ s přesností $0{,}01\%$.

Script funkce:

```
function [fv,k,re,i]=regulafalsi(fce,xd,xh,me,mi)
% regulafalsi: hledani korenu metodou s 2 pocatecnimi odhady (linearni interpolace); uzavrena metoda
% Vstup
% fce: funkce jejiz koren hledame
% xd: pocatecni dolni odhad
% xh: pocatecni horni odhad
% me: maximalni relativni chyba (prednastaveno = 0.0001%)
% mi: maximalni pocet iteraci (prednastanoveno = 100)
% Vystup:
% fv: hodnota funkce v korenu
% k: koren funkce
% re: relativni chyba aproximace (%)
% i: pocet iteraci

   if nargin<3,error('minimum 3 vstupni parametry'),end
   if nargin<4, me=0.005;mi=10000;end
   if nargin<5, mi=10000;end
   if fce(xd)*fce(xh)>0,error('není znamenkova zmena '),end
   % první krok
   i = 0;
   while(1)
      t1=fce(xd);
      t2=fce(xh);
      if t1==t2,error('vyber jiny interval, zde metoda nekonverguje');end
      xn =(xd*t2-xh*t1)/(t2-t1);
      re=abs((xn-xh)/xn);
      if((re<me) || mi<i),break;end
      i = i + 1;
      test=t1*t2;
      if test<0,
         xh = xn;
      elseif test>0
```

```
        xd = xn;
      else
        re = 0;
      end
   end
      k=xn;
      fv =fce(k);
end
```

Script řešení přikladu 2:

```
clc; close; clear;                                  % vymazání Command window, grafických oken a proměnných
fce=@(x) 10*cos(x-1)-x^2+2*x-1;                     % definovaná funkce dle zadání
format long                                             % zobrazení výsledků na 15 desetinných čísel
[fv,k,re,i]=regulafalsi(fce,2.3,2.4,0.0001,100)         % volání funkce regulafalsi s danými parametry
```

Výsledek Matlabu:

```
fv =    3.552713678800501e-15       % hodnota zadané funkce v aproximaci
k =   2.379364594222031             % aproximace kořene s danou přesností
re =    6.290953031559411e-11       % relativní chyba aproximace
i =    3                            % počet aproximaci k dosažení dané přesnosti
```

Příklad 3.

Prostou iterační metodou nalezněte kořen rovnice $10\cos(x-1)-x^2+2x-1=0$, který leží v intervalu $\langle 2{,}3;\ 2{,}4\rangle$ s přesností $0{,}01\%$.

U této metody musíme poznamenat, že ze zadání musíme nejdříve odvodit kontraktivní zobrazení (Banachova věta) $x=\varphi(x)$, které generuje posloupnost, která konverguje ke kořenu. Vždy máme více možností. V tomto případě například:
fce1=@(x) sqrt(10*cos(x-1)+2*x-1);
fce2=@(x) -1/2*(10*cos(x-1)-x^2-1);
fce3=@(x) 1+acos(1/10*(x^2-2*x+1));
Samozřejmě lze dokázat, které zobrazení je kontraktivní, ale v Matlabu je rychlejší konvergenci vyzkoušet. Dle výsledků vidíme, že kontraktivní zobrazení je fce3=@(x) 1+acos(1/10*(x^2-2*x+1)).

Script funkce:

```
function [k,re,i]=mpi(fce,x1,me,mi)
% metoda proste iterace: hledani korene s pocatecnim odhadem
% Vstup
% fce: Funkce jejiz koren hledame
% x1: pocatecni odhad
% me: maximalni relativni chyba (prednastaveno = 0.0001%)
% mi: maximalni pocet iteraci (prednastanoveno = 100)
% Vystup:
% k: koren funkce
% re: relativni chyba aproximace (%)
% i: pocet iteraci
   if nargin<2,error('minimum 2 vstupni parametry'),end
   if nargin<3, me=0.005;mi=100;end
   if nargin<4, mi=100;end
% prvni krok
   i = 0;
```

```
   xn=x1;
   while(1)
      xb=xn;
      xn=fce(x1);
      re=abs((xn-xb)/xn);
      if((re<me) || mi<=i),break;end
      x1=xn;
      i=i + 1;
   end
      k=xn;
end
```

Script řešení přikladu 2:

```
clc; close; clear;                      %  vymazání Command window, grafických oken a proměnných
%fce=@(x) 10*cos(x-1)-x^2+2*x-1;        %  definovaná funkce dle zadání
fce1=@(x) sqrt(10*cos(x-1)+2*x-1);      %  1. zobrazení
fce2=@(x) -1/2*(10*cos(x-1)-x^2-1);     %  2. zobrazení
fce3=@(x) 1+acos(1/10*(x^2-2*x+1));     %  3. zobrazení
format long                             %  zobrazení výsledků na 15 desetinných čísel
[k,re,i]=mpi(fce3,2.3,0.0001)           %  volání funkce mpi s danými parametry
```

Výsledek Matlabu pro zobrazení fce1 a fce2:

```
k =   NaN                               %  aproximace kořene   %  zde vidíme, že zobrazení nejsou kontraktivní
re =   NaN                              %  relativní chyba kořenu
i =   100                               %  počet iteraci
```

Výsledek Matlabu pro zobrazení fce3:

```
k =   2.379326468875754                 %  aproximace kořene   %  zde vidíme, že zobrazení je kontraktivní
re =    7.304277791300346e-05           %  relativní chyba aproximace
i =    5                                %  počet iterací
```

Příklad 4.

Newtonovou metodou nalezněte kořen rovnice $10\cos(x-1)-x^2+2x-1=0$, který leží v intervalu $\langle 2{,}3;\,2{,}4\rangle$ s přesností $0{,}01\%$.

Script funkce:

```
function [fv,k,re,i]=newton(fce,fceder,x1,me,mi)
% newton: hledani korene s jednim pocatecnim odhadem
% otevrena metoda
% Vstup
% fce: funkce u ktere hledame koren
% fceder: derivace funkce u ktere hledame koren
% x1: pocatecni odhad
% me: maximalni relativni chyba (prednastaveno = 0.0001%)
% mi: maximalni pocet iteraci (prednastanoveno = 100)
% Vystup
% k: koren
% fv: Hodnota funkce v aproximaci korene
% re: relativni chyba aproximace (%)
% i: pocet iteraci

    if nargin<3,error('minimum 3 vstupni parametry'),end
   if nargin<4, me=0.005;end
```

```
   if nargin<5, mi=100;end
% prvni krok
   i = 0;
   xn=x1;
   if fceder(xn)==0,error('Spatny odhad, metoda nekonverguje'),end
   while(1)
      xb=xn;
      xn=xn-fce(xn)/fceder(xn);
      i = i + 1;
      if xn==0,xn=0.001;warning('Newtonova metoda muze divergovat, zkus jinou metodu');end
      re=abs(xn-xb)/xn;
      if((re<=me) || mi<i),break;end
   end
      k=xn;
      fv =fce(k);
end
```

Script řešení přikladu 4:

```
clc; close; clear;                          % vymazání Command window, grafických oken a proměnných
fce=@(x) 10*cos(x-1)-x^2+2*x-1;             % definovaná funkce dle zadání
fceder=@(x) -10*sin(x-1)-2*x+2;             % derivace definované funkce dle zadání
format long                                 % zobrazení výsledků na 15 desetinných čísel
[fv,k,re,i]=newton(fce,fceder,2.3,0.0001,50) % volání funkce newton s danými parametry
```

Výsledek Matlabu:

```
fv =    -1.015200368215119e-09              % hodnota zadané funkce v aproximaci

k =   2.379364594302756                     % aproximace kořene s danou přesností

re =    9.586114182151763e-06               % relativní chyba aproximace

i =    3                                    % počet iterací
```

Příklad 5.

Metodou sečen nalezněte kořen rovnice $10\cos(x-1)-x^2+2x-1=0$, který leží v intervalu $\langle 2{,}3;\ 2{,}4\rangle$ s přesností $0{,}01\%$.

Script funkce:

```
function [k,re,i]=secna(fce,x1,x2,me,mi)
% metoda secen: hledani korenu se 2 pocatecnimi odhady
% Otevrena metoda
% vstup
% fce: Funkce jejiz koren hledame
% x1: Pocatecni odhad 1
% x2: Pocatecni odhad 2
% me: Maximalni relativni chyba (prednastaveno = 0.0001%)
% mi: maximalni pocet iteraci (prednastanoveno = 100)
% Vystup:
% k: koren
% re: relativni chyba aproximace (%)
% i: pocet iteraci

   if nargin<3,error('minimum 3 vstupni parametry'),end
   if nargin<4, me=0.005;mi=100;end
   if nargin<5, mi=100;end
```

```
% Prvni krok
   i = 0;
   while(1)
      t1=fce(x1);
      t2=fce(x2);
      if t1==t2,error('Spatny odhad, metoda nekonverguje');end
      x3=(x1*t2-x2*t1)/(t2-t1);
      i=i + 1;
      re=abs((x3-x2)/x3);
      if((re<me) || mi<i),break;end
      x1=x2;
      x2=x3;
   end
      k=x3;
end
```

Script řešení přikladu 5:

```
clc; close; clear;                          % vymazání Command window, grafických oken a proměnných
fce=@(x) 10*cos(x-1)-x^2+2*x-1;             % definovaná funkce dle zadání
format long                                 % zápis výsledků na 15 desetinných čísel
[k,re,i]=secna(fce,2.3,2.4,0.0001,100)      % volání funkce secna s danými parametry
```

Výsledek Matlabu:

```
k =   2.379364594257317                     % aproximace kořene s danou přesností

re =    3.549720839167417e-07               % relativní chyba aproximace
i =    3                                    % počet iterací
```

Příklad 6.

Newtonovou metodou řešte soustavu nelineárních rovnic $\begin{pmatrix} 1-4x+2x^2-2y^3 \\ -4+x^4+4y+4y^4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ s počáteční aproximací $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0{,}1 \\ 0{,}7 \end{pmatrix}$ s přesností $0{,}0000001$.

Řešení užitím Newtonovy iterační metody popíšeme celkem ve čtyřech krocích. V prvním kroku načteme parametry zadání a definujeme přesnost řešení: maximální počet iterací (mi), přesnost (eps) a počet rovnic soustavy. Ve druhém načteme všechny rovnice soustavy, které definujeme jako anonymní funkce a vektor počátečních odhadů. Ve třetím kroku definujeme iterační proces, který se bude opakovat, dokud nebude splněna požadovaná přesnost. Zde musíme vypočítat hodnotu daných funkcí v odhadech, seskupit matici numerických derivací funkcí do Jacobiánu, vyřešit soustavu rovnic, vypočítat následující odhad a otestovat, zda je splněno konvergenční kritérium pro danou přesnost. Nakonec vypíšeme odhady, které splňují požadovanou přesnost a počet iterací.

Script řešeni příkladu 6:

```
clc; close; clear;
%  Resic soustav nelinearnich rovnic:
%  tento script resi soustavy nelinearnich rovnic uzitim Newtonovy iteracni metody
%  kroky:
%  1) nacteni parametru zadani a reseni:
%  -maximalni pocet iteraci: mi
%  -presnost konvergence: eps
%  -pocet rovnic: n
%  2) nacteni rovnic
```

```
%   -nacteni vsech rovnic soustavy
%   -nacteni vektoru pocatecnich odhadu
%   3) Iterace (opakovani dokud neni splnena pozadovana presnost):
%      3.1) seskupeni matice derivaci funkci
%      3.2) vypocet nove hodnoty promennych
%      3.3) porovnani rozdilu mezi novym a starym odhadem a konvergencnim kriteriem (eps)
%   4) Prezentace a ulozeni vypoctenych hodnot
% 1 nacteni parametru zadani a reseni:
mi=1000;
eps=0.0000001;
n=2;
% 2 nacteni rovnic
%F1 = @(x(1),x(2)) 1-4*x(1)+2*x(1)^2-2*x(2)^3;
%   rovnice jsou definovany jako anonymni funkce;
%   musi byt cislovane postupne: F1,F2,... ,Fn;
%   musi byt seskupeny do pole funkci: F={F1,F2,... Fn}
%   promenne musi byt cislovane: x(1),x(2),... ,x(n);
%1 rovnice: (F1)
F1 = @(x) 1-4*x(1)+2*x(1)^2-2*x(2)^3;
%2 rovnice: (F2)
F2 = @(x) -4+x(1)^4+4*x(2)+4*x(2)^4;
% pole funkci
F={F1,F2};
%  pocatecni odhady:
x(1)=0.1;
x(2)=0.7;
% ... x(n)=a;
% 3) Iterace
ni=0;
nc=0;
while ni<mi
   % seskupeni matice derivaci
   dFidxj=zeros(n,n);
   for i=1:1:n
      for j=1:1:n
         derivace;
      end
   end
   % vypocet vektoru hodnot funkci v bodech
   for i=1:1:n
     j=F{i};
     D(i,1)=j(x);
   end
   % vypocet chyby
   A=inv(dFidxj);
   error=A*D;
   dx=error';
   % pridanim chyby k x dostaneme novy odhad
   x=x-dx;
   % Parametry pro ukonceni cyklu while
   ni=ni+1;
   nc=nc+1;
% kriterium konvergence (error < eps)
   maxdx=max(abs(error));
      if maxdx<eps
         ni=mi;
      end
```

```
end
% 4) Prezentace a ulozeni vypoctenych hodnot
x
nc
```

Script pro výpočet numerických derivací v souboru derivace.m:

```
% vypocet derivaci soustavy
fx=F{i};
xj=x;
%funkce za bodem
xj(j)=x(j)+eps/2;
y1=fx(xj);
%funkce pred bodem
xj(j)=x(j)-eps/2;
y0=fx(xj);
%Derivace v bode
dFidxj(i,j)=(y1-y0)/eps;
```

Výsledek Matlabu:

```
x =    0.0618    0.7245
nc =     4
```

MATLAB	**1.12. Numerické řešení diferenciálních rovnic v Matlabu**
Příklady příkazů: *help symbolic* *syms a b t x y;* *V=sin(x)^2+cos(x)^2* *simple(V)* *pretty(V)*	Diferenciální rovnice patří mezi nejdůležitější matematické nástroje, které používáme pro modelování fyzikálních, technických, biologických, sociologických a jiných problémů. V této části budeme uvažovat numerické řešení obyčejných diferenciálních rovnic, které mají pouze jednu nezávislou proměnnou, nejčastěji čas. Numerické řešení parciálních diferenciálních rovnic uvedeme v následující kapitole. Řád diferenciální rovnice (resp. soustavy) je řád nejvyšší derivace, která je v ní obsažená. Podle řádu dělíme diferenciální rovnice na diferenciální rovnice prvního, druhého a vyšších řádů. Lineární diferenciální rovnice obsahuje hledanou funkci v první mocnině a nevyskytují se nikde její součiny s derivacemi ani součiny derivací. Pokud není diferenciální lineární je nelineární.