Numerické metody

pro řešení diferenciálních rovnic
! PRACOVNÍ VERZE !
Průvodce softwarem a počítačová cvičení v prostředí MATLABu

R. Hlavička

Obsah

Úvod **3**

I. Obyčejné diferenciální rovnice 7

1. Počáteční problémy pro obyčejné diferenciální rovnice **9**
1.1. Základní pojmy . . . 9
1.1.1. Směrové pole . . . 9
1.1.2. Autonomní soustava diferenciálních rovnic . . . 12
1.2. Explicitní Eulerova metoda (EE) . . . 14
1.2.1. Řád a stabilita EE . . . 20
1.2.2. *Praktický odhad chyby a extrapolace . . . 24
1.3. Implicitní Eulerova metoda (IE) . . . 25
1.4. TR metoda . . . 27
1.5. *Taylorova metoda . . . 28
1.6. Explicitní Runge-Kuttovy metody . . . 34
1.6.1. Adaptivní integrace . . . 39
1.6.2. Hustý výstup . . . 41
1.7. Adamsovy metody . . . 43
1.8. Metody zpětného derivování . . . 46
1.9. Implementace v matlabu a příklady . . . 48
1.9.1. Keplerův problém a problém více těles . . . 48

2. Okrajové problémy pro obyčejné diferenciální rovnice **52**
2.1. Základní pojmy . . . 52
2.1.1. Metoda střelby . . . 52
2.1.2. Diferenční metoda . . . 54
2.1.3. Metoda konečných prvků . . . 57

II. Parciální diferenciální rovnice 60

3. Okrajové problémy pro parciální diferenciální rovnice **62**
3.1. Úloha eliptického typu . . . 62
3.1.1. Diferenční metoda . . . 62
3.1.2. Metoda konečných prvků . . . 67

4. Evoluční problémy **76**
4.1. Úloha parabolického typu . . . 76
4.2. Úloha hyperbolického typu . . . 81

Literatura **87**

Úvod

Candida candidis
(čistému vše čisté)

Na úvod si tak trochu připomeneme systém MATLAB. Pomocný matlabovský skript **cls**

cls.m

```
clear; clc; close;
```

volám na začátku každého samostatně spustitelného kódu (programu).

Funkce **struct2pairs** konvertuje matlabovskou strukturu do buněčného pole dvojic název, hodnota.

struct2pairs.m

```
function p = struct2pairs( s )
c = struct2cell(s);
f = fieldnames(s);
p = cell(1,length(c)*2);
p(1:2:end) = f;
p(2:2:end) = c;
end
```

V celém textu jsou matlabovské funkce formátovány jako struct2pairs.m a skripty tj. spustitelné programy (není u nich třeba zadávat parametry pro spuštění) jako cls.m. Dobrou knihou věnující se prostředí Matlabu je např. [3].

Příklad 1. Vytvoříme strukturu ze dvojic, odebereme položku se jménem *id* a přidáme položku se jménem *z*. Poté zkonvertujeme na dvojice a zapíšeme do souboru struct.txt

StructDemo.m

```
cls;
s = struct('id',5,'x',pi/2,'y',0);
s = rmfield(s,'id');
s.z = exp(1);
p =  struct2pairs(s);
fid=fopen('struct.txt','w');
fprintf(fid,'%6s | %6s | %6s\r\n',p{1:2:end});
fprintf(fid,'%6.4f | %6.4f | %6.4f',p{2:2:end});
fclose(fid);
```

struct.txt

```
     x |       y |       z
1.5708 | 0.0000 | 2.7183
```

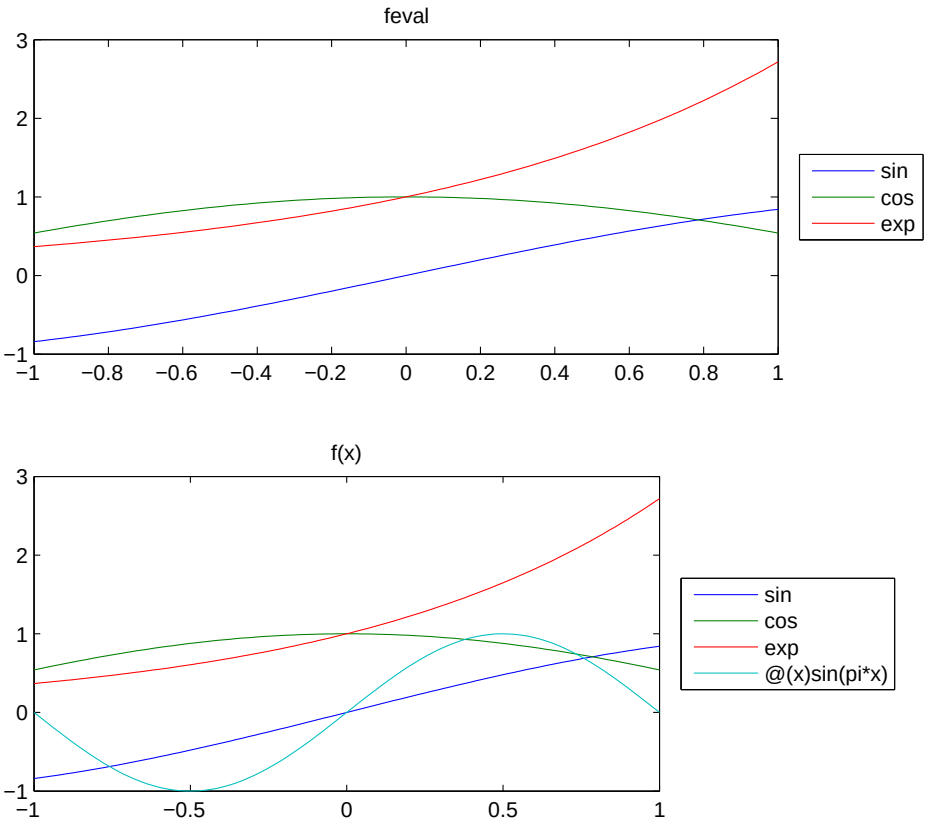
Často budeme používat anonymní funkce ukážememe si to pro funkce jedné a dvou proměnných.

Příklad 2. Nakreslete graf množiny funkcí pomocí příkazu feval a funkčních handlů.

uvod/s1.m

```
addpath ..
```

```
cls

fn = { 'sin', 'cos', 'exp'};
f = { @sin, @cos, @exp};
x = linspace(-1,1);
yn = zeros(length(fn),length(x));
f{end+1} = @(x) sin(pi*x);
yh = zeros(length(f),length(x));

for i=1:length(fn)
    yn(i,:) = feval(fn{i},x);
end

for i=1:length(f)
    yh(i,:) = f{i}(x);
end

subplot(2,1,1); plot(x,yn); title('feval');
legend(fn,'Location','EastOutside');
subplot(2,1,2); plot(x,yh); title('f(x)');
legend( cellfun( @char,f, 'UniformOutput', 0 ),'Location','EastOutside');
print -depsc o1
```

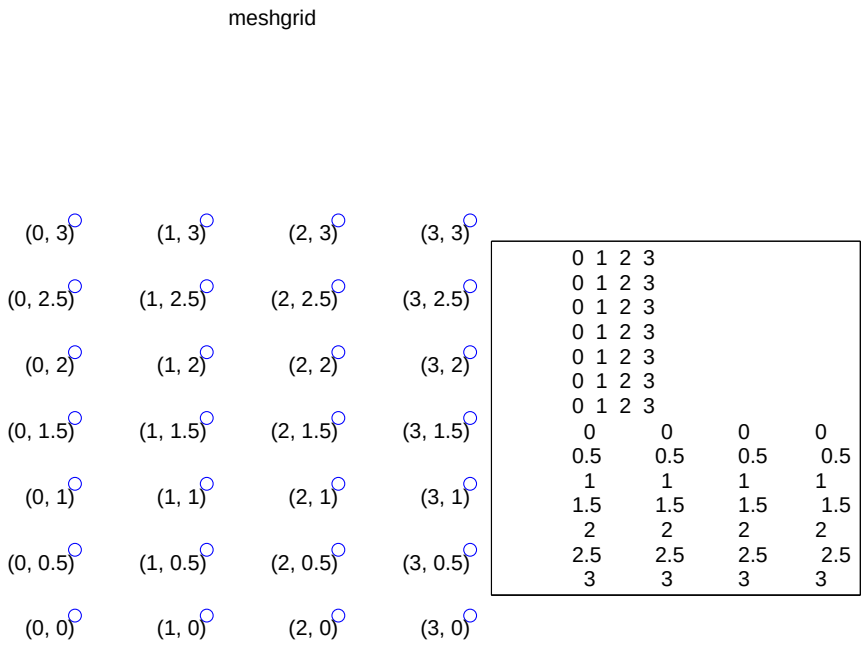


Příklad 3. Vysvětlete obrázkem matlabovslou funkci meshgrid.

uvod/s2.m

```
addpath ..
cls

x = linspace(0,3,4);
y = linspace(0,3,7);
[X,Y] = meshgrid(x,y);
plot(X(:),Y(:),'o');
txtx = arrayfun( @num2str, X, 'UniformOutput', 0 );
```

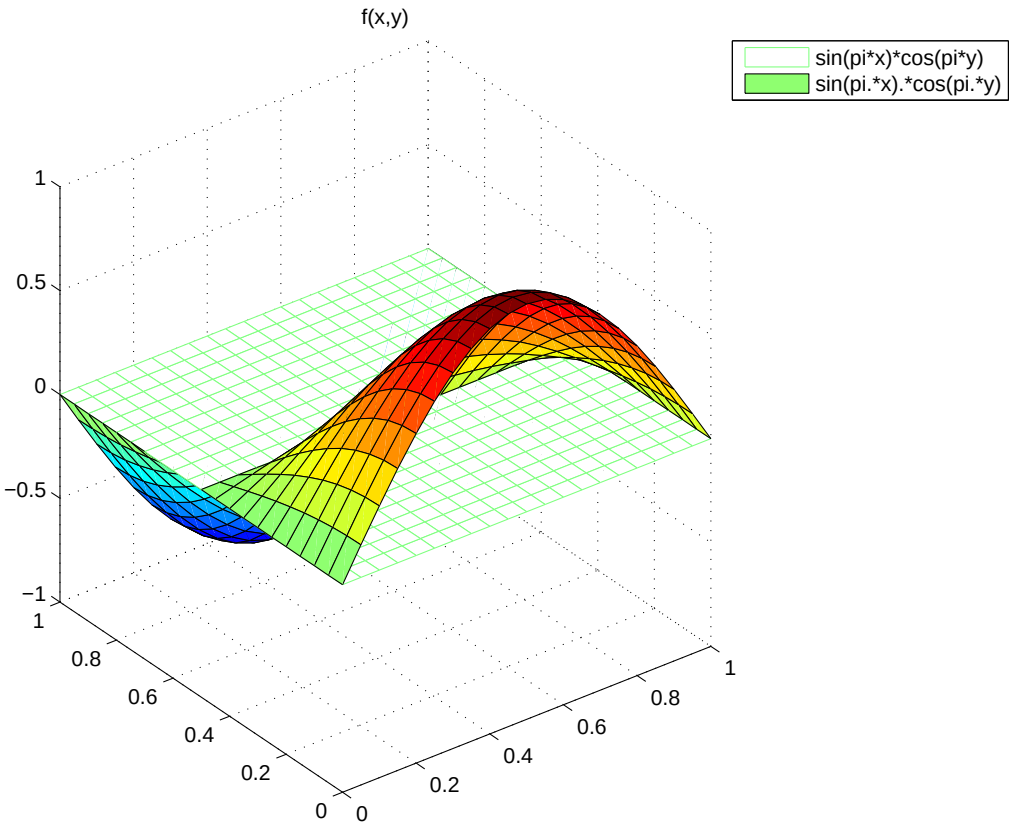
```
txty = arrayfun( @num2str, Y, 'UniformOutput', 0 );
tf = @(x,y) ['(' x ', ' y ')'];
txt = cellfun(tf,txtx,txty, 'UniformOutput', 0 );
h=text(X(:),Y(:), txt, ...
    'HorizontalAlignment', 'right', 'VerticalAlignment', 'top');
legend(h,num2str(X),num2str(Y),'Location','EastOutside');
title('meshgrid');
axis equal; axis off;
print -depsc o2
```



Příklad 4. Nakreslete graf funkce dvou proměnných.

uvod/s3.m

```
addpath ..
cls

x = linspace(0,1,21);
[X,Y] = meshgrid(x,x);

f = @(x,y) sin(pi*x)*cos(pi*y);
sf = 'sin(pi*x)*cos(pi*y)';
vf = vectorize(sf);
F = inline(vf);

figure;
mesh(X,Y,f(X,Y)); hold on;
surf(X,Y,F(X,Y));
legend({sf,vf});

title('f(x,y)');
print -depsc o3
```



Cvičení 5. V předchozím kódu volání funkce funkce f v příkazu surf nedává očekávaný graf (proč?). Ověřte, že funguje tato modifikace

```
surf(X,Y,arrayfun(f,X,Y));
```

Cvičení 6. Přečtěte si podrobně všechny kódy, neznámé příkazy najděte v dokumentaci.

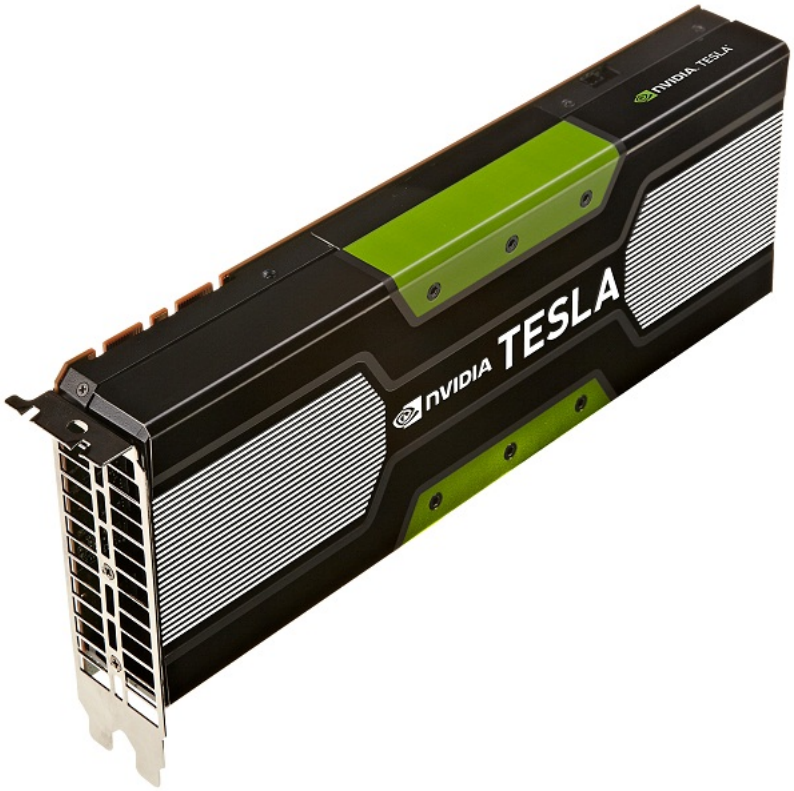
Část I.

Obyčejné diferenciální rovnice

Někerej revolver, paní Müllerová, vám nedá ránu, kdybyste se zbláznili. Takovejch systémů je moc.

Obyčejné diferenciální rovnice (ODR) jsou důležité v mnoha oborech. Klasická mechanika je posána soustavou ODR, další aplikace jsou v biologii (populační dynamika), statistické fyzice a molekulární dynamice. A především se numerické metody řešení ODR používají v numerických simulacích při řešení evolučních parciálních diferenciálních rovnic (PDR), se kterými budeme pracovat ve druhé části textu.

Existuje velké množství numerických knihoven pro ODR. Jsou nedílnou součástí obecných systémů numerické matematiky jako jsou IMSL, NAG. V souvislosti s revolučním nástupem paraleních technologií se software velmi mění. V této souvislosti zmíním otevřený software *Odeint* . Je moderní knihovna v jazyce C++, která je napsána obecným (generickým) způsobem technikou metaprogramování s použitím šablon. Tato technika přináší vyjímečnou flexibilitu a výkon, je možno použít paralelní akceleraci s pomocí procesorů počítače (CPU) i grafické karty (GPU). V blízké době se má stát součástí BOOSTu.



Numerické metody pro ODR jsou součástí systému MATLAB jako funkce z prefixem ode (ode45, ode15s, ode23, ode113, ode23t, ode23tb, ode23s) a jsou součástí i otevřených variant OCTAVE a SCILABu. Naším cílem bude i poodhalit co se za jejich implementací skrývá. Zde nám pomůže dokumentace k MATLABu a náš základní text [5] resp. [11, 12].

1. Počáteční problémy pro obyčejné diferenciální rovnice

1.1. Základní pojmy

1.1.1. Směrové pole

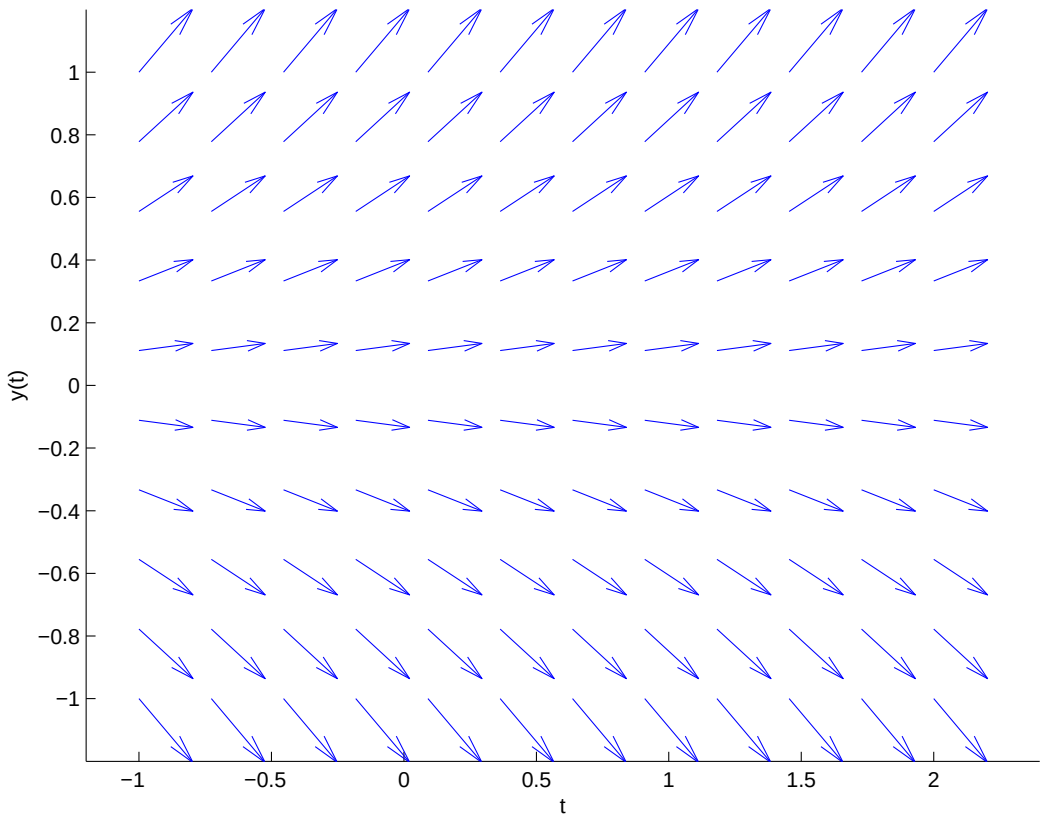
Budeme pracovat se skalární obyčejnou diferenciální rovnicí prvního řádu

$$y' = f(t, y).$$

Jak víme z předmětu M3, pravá strana této rovnice předepisuje v každém bodě $[t, y]$ vektor rychlosti. Směrové pole nakreslíme následujícím matlabovským skriptem. Šipky rychlostí v jsou generovány vestavěnou funkcí **quiver**, která zobrazuje vektory včetně jejich relativní velikosti. To je v tomto případě skalární funkce $y(t)$ poněkud nestandardní, velikost derivace určíme jako směrnici úsečky která vytváří šipku (stoupání) a tak bývá zvykem kreslit úsečky stejné délky.

s1.m

```
cls;
% y' = f(t,y) = y
f = @(t,y) y;
n = [10 12];
box = [-1 2 -1 1];
b = box*1.2;
axis(b); hold on;
t = linspace(box(1), box(2), n(2));
y = linspace(box(3), box(4), n(1));
% body
[T,Y] = meshgrid(t,y);
% derivace
F = f(T,Y);
% vektorove pole
quiver(T,Y,ones(size(T)),F);
xlabel('t');
ylabel('y(t)');
print -depsc o1
```



Jako funkci f jsme vybrali velmi jednoduchou funkci $f = \lambda y$, kde $\lambda = 1$. Rovnice je lineární, snadno spočítáme přesné (analytické) obecné řešení

$$\begin{aligned} y' &= \lambda y \\ &\Downarrow \\ y &= C\exp(\lambda t) = Ce^{\lambda t}. \end{aligned} \tag{1.1.1}$$

Z tohoto skriptu uděláme funkci ***dfield***, kde provedeme jisté modifikace

dfield.m

```
function [T,Y,F,h] = dfield( f, box, n, varargin )
opt = struct(varargin{:});
t = linspace(box(1), box(2), n(2));
y = linspace(box(3), box(4), n(1));
[T,Y] = meshgrid(t,y);                          % body
F1 = ones(size(T));
F2 = f(T,Y);                                    % derivace
F = F2;
if isfield(opt,'normalize')
    nF = sqrt(F1.^2+F2.^2);
    F1 = F1./nF;
    F2 = F2./nF;
    opt = rmfield(opt,'normalize');
end
if isfield(opt,'scale')
    scale = opt.scale;
    opt = rmfield(opt,'scale');
    pairs=struct2pairs(opt);
    % smerove pole, skalovani
    h(1)=quiver(T,Y,F1,F2,scale,pairs{:});
    h(2)=quiver(T,Y,-F1,-F2,scale,pairs{:});
else
    pairs=struct2pairs(opt);
```

```
    % smerove pole
    h(1)=quiver(T,Y,F1,F2,pairs{:});
    h(2)=quiver(T,Y,-F1,-F2,pairs{:});
end
xlabel('t'); ylabel('y(t)');
end
```

a hned si ji zavoláme v následujícím skriptu

s2.m

```
cls;
f = @(t,y) y;                                   % y' = f(t,y) = y
n = [10 12];
box = [-1 2 -1 1];
b = box*1.2;
axis(b); hold on;
[T, Y, F, h] = dfield(f,box,n,'normalize',true,'scale',0.5,...
    'ShowArrowHead','off','Color','k');
t = linspace(b(1),b(2));
[sz1 sz2] = size(T);
plot(T,Y,'+k');
for i1=1:sz1
    for i2=1:sz2
        plot(t,F(i1,i2)*exp(t-T(i1,i2)),'Color',[0.8 0.9 1]);
    end
end
uistack(h(:),'top');
print -depsc o2
```

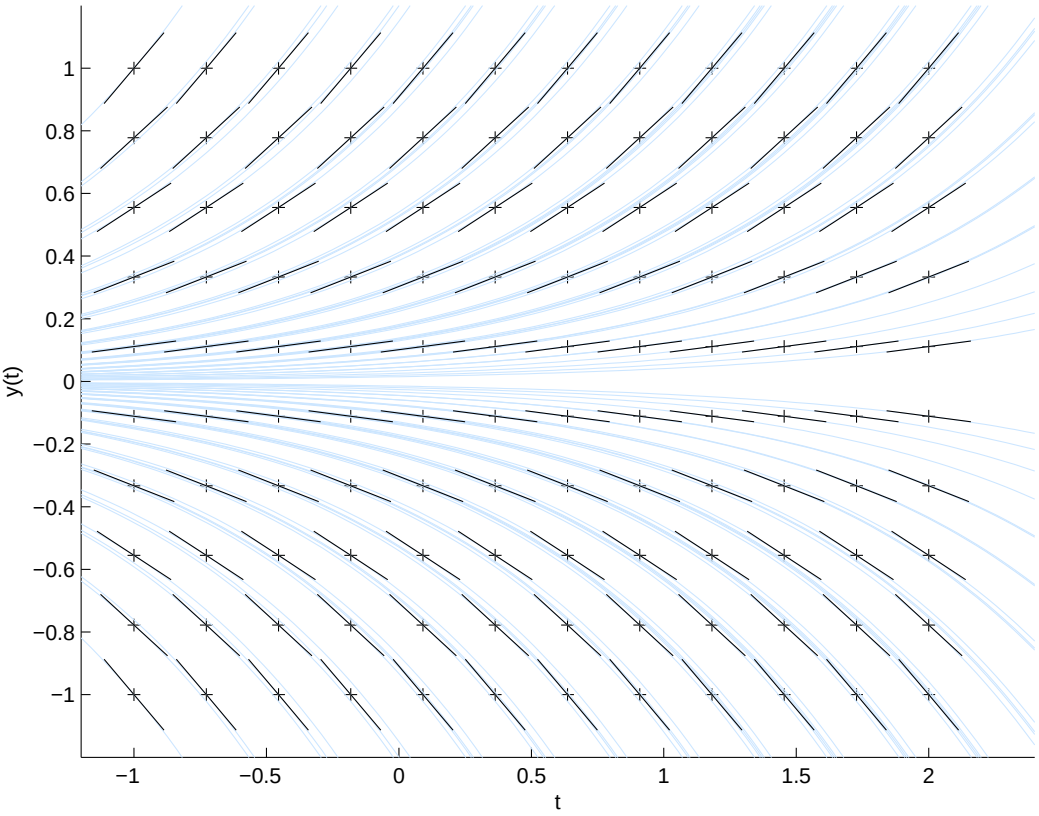
který kromě směrového pole, kreslí rovněž grafy přesných řešení pro počáteční úlohy

$$\begin{aligned} y' &= y, \\ y(a) &= \eta, \end{aligned} \tag{1.1.2}$$

kde dvojice $[a, \eta]$ jsou uzly sítě, kterou jsme dostaly použitím příkazu **meshgrid**. Tyto uzly zobrazíme na obrázku jako křížek +. Přesné řešení počáteční úlohy (1.1.2) je

$$y = \eta \exp(t - a).$$

Výsledný obrázek je zde:



Vidíme, že funkce **dfield** kreslí krátké úsečky, jejichž délku můžeme upravit pomocí vlastnosti *scale*. Volání funkce **uistack** nám přitom zajistilo, že úsečky se kreslí až po vykreslení řešení a jsou tedy dobře viditelné.

1.1.2. Autonomní soustava diferenciálních rovnic

Diferenciální rovnice (1.1.1) je přes svou jednoduchost velmi důležitá, její řešení se tudíž stalo tzv. elementární (transcendentální) funkcí. Popisuje např. model růstu populace při neomezeném dostatku potravy a za absence predátorů. Parametr λ je koeficient lineární závislosti (přírůstek populace za časovou jednotku závisí lineárně na počtu y jedinců v populaci). Slavný Lotka-Volterrův model přidává interakci s predátory.[1] Soustava Lotka-Volterrových rovnic vypadá takto

$$\mathbf{y}' = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} \alpha y_1 - \beta y_1 y_2 \\ -\gamma y_2 + \delta y_1 y_2 \end{bmatrix} = \mathbf{f}(t, \mathbf{y}) \tag{1.1.3}$$

kde

y_1	hustota populace kořisti
y_2	hustota populace predátorů
α	faktor množení kořisti
β	koeficient predace
γ	faktor úhynu predátorů
δ	reprodukční míra predátorů na jednu kořist.

[1] Model je pojmenován po svých autorech, kterými byli Alfred J. Lotka (1880-1949) a Vito Volterra (1860-1940). Tento model vytvořili nezávisle na sobě v letech 1925 a 1926.

Pravá strana $\mathbf{f}(t, \mathbf{y}) = \mathbf{f}(\mathbf{y})$ je opět nezávislá na čase, takové soustavě se říká autonomní soustava. Tento fakt nám umožňuje vizualizovat směrové pole v rovině. Program dfield.m trochu modifikujeme

dfielda.m

```
function h = dfielda( f1, f2, box, n, varargin )
opt = struct(varargin{:});
y1 = linspace(box(1), box(2), n(1));
y2 = linspace(box(3), box(4), n(2));
[Y1,Y2] = meshgrid(y1,y2);                          % uzly
F1 = f1(Y1,Y2);                                     % derivace
F2 = f2(Y1,Y2);
if isfield(opt,'normalize')
    if opt.normalize
        nF = sqrt(F1.^2+F2.^2);
        F1 = F1./nF;
        F2 = F2./nF;
    end
    opt = rmfield(opt,'normalize');
end
% smerove pole
if isfield(opt,'scale')
    scale = opt.scale;
    opt = rmfield(opt,'scale');
    pairs=struct2pairs(opt);
    h=quiver(Y1,Y2,F1,F2,scale,pairs{:});
else
    pairs=struct2pairs(opt);
    h=quiver(Y1,Y2,F1,F2,pairs{:});
end
xlabel('y_1');
ylabel('y_2');
end
```

a modifikovaný skript používá hodnoty

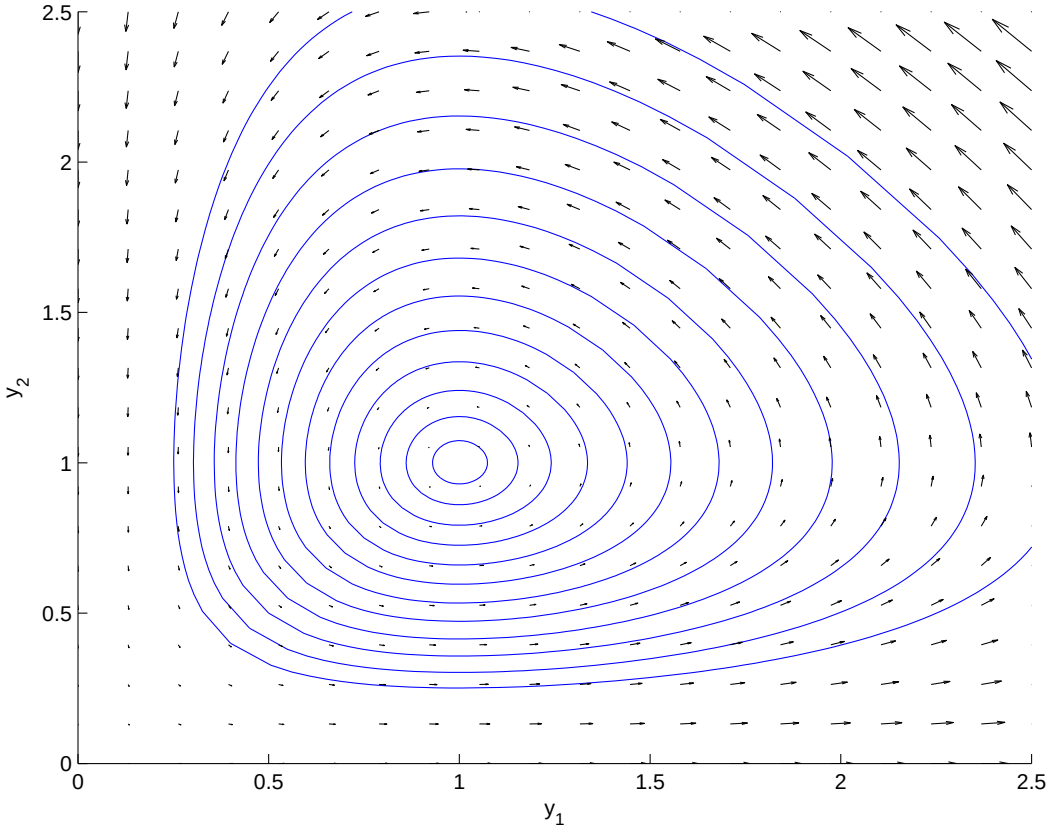
$$\alpha = \beta = \gamma = \delta = 1 \tag{1.1.4}$$

s2a.m

```
cls;
%parametry
a=1; b=1; c=1; d=1;
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f1 = @(y1,y2) a*y1-b*y1.*y2;
f2 = @(y1,y2) -c*y2+d*y1.*y2;
n(1) = 20; n(2) = 20;
box = [0,2.5,0,2.5];
% f = @(t,y) [f1(y(1),y(2)); f2(y(1),y(2))];
% [t,y] = ode23(f,[0 6.7], [0.5; 0.5]);
% plot(y(:,1),y(:,2));
for k=1:2
    clf;
    axis(box); hold on;
    h=dfielda(f1,f2,box,n,'normalize',k==1,'Color','k'); hold on;
    e = 0.4:0.05:0.95;
    for i=1:length(e)
        eta = [e(i); e(i)];
        C = eta(1)*eta(2)/exp(eta(1)+eta(2));
```

```
        R = @(t) t.^2-4*C*exp(t);
        z1 = fzero(R,1);
        z2 = fzero(R,3);
        tau = linspace(z1,z2,100);

        r = sqrt(abs(R(tau)));

        y1 = (tau+r)/2;
        y2 = (tau-r)/2;
        plot(y1,y2);
        plot(y2,y1);
    end
    uistack(h(:),'top');
    print('-depsc',['o2a' int2str(k)]);
end
```

dostaneme tzv. fázový portrét:



Soustava 1.1.3 obsahuje nelineární člen $y_1 y_2$, ale pro volbu 1.1.4existuje explicitní parametrizace trajektorie řešení viz [6], několik takových trajektorií jsme si nakreslili.

Cvičení 7. Přečtěte si podrobně všechny kódy, neznámé příkazy najděte v dokumentaci.

Cvičení 8. Zobrazte fázový portrét pro řešení soustavy, která vznikne z rovnice 2. řádu

$$y'' = -y$$

přepisem na soustavu dvou rovnic prvního řádu.

1.2. Explicitní Eulerova metoda (EE)

Počáteční úlohu pro jednu obyčejnou diferenciální rovnici prvního řádu

$$\begin{aligned} y' &= f(t,y), \\ y(a) &= \eta \end{aligned}$$

resp. pro soustavu rovnic d rovnic prvního řádu

$$\begin{aligned} y_1' &= f_1(t, y_1, \ldots, y_d), \\ &\vdots \\ y_d' &= f_d(t, y_1, \ldots, y_d), \\ y_1(a) &= \eta_1, \\ &\vdots \\ y_d(a) &= \eta_d \end{aligned}$$

budeme nejdříve aproximovat nejjednodušší explicitní Eulerovou metodou (stručně EE). Soustavu lze zapsat pomocí vektorů takto

$$\begin{aligned} \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}' &= \begin{bmatrix} f_1(t, y_1, \ldots, y_d) \\ \vdots \\ f_d(t, y_1, \ldots, y_d), \end{bmatrix}, \\ \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}(a) &= \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_d \end{bmatrix} \end{aligned}$$

a obvykle jen stručně (tučné písmo značí vektor) takto

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(t, \mathbf{y}), \\ \mathbf{y}(a) &= \boldsymbol{\eta} \quad . \end{aligned}$$

A tak jednoduché to je i s numerickými metodami řešení, místo skalárů budeme pracovat s vektory. Prostě přepíšeme všechna písmena na tučná s výjímkou kterou tvoří nezávisle proměnná t a její konkrétní hodnoty (zde a).

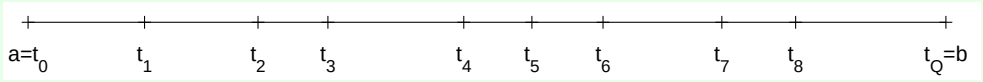
EE metodu dostaneme např. touto jednoduchou „kinematickou" úvahou. Nezávisle proměnná pro nás bude tedy čas, pravá strana předepisuje okamžitou rychlost pohybu hmotného bodu a vektor $\mathbf{y}$ je jeho polohový vektor (radiusvektor). Diferenciální rovnice je pohybový zákon, který je „infinitezimální", okamžitá rychlost se mění spojitě. V tom je právě problém, pokud si vyhodnotíme z počáteční podmínky počáteční rychlost

$$\mathbf{f}(t, \boldsymbol{\eta})$$

můžeme se podle klasických Newtonovských představ touto rzchlostí pohybovat jen po nekonečně malý tj. infinitezimální časový interval. Takhle se ale v konečném počtu kroků nikam nedostaneme! (viz Zenónovy aporie pohybu). Ale my to musíme spočítat a tak se po malý časový přírůstek τ_0 budeme touto rychlostí pohybovat. Dostaneme se do nového bodu

$$\mathbf{y}_1 = \mathbf{y}_0 + \underbrace{\mathbf{f}(t, \mathbf{y}_0)\tau_0}_{\text{rychlost*čas}}, \quad \mathbf{y}_0 = \mathbf{y}(a) = \boldsymbol{\eta}$$

a tak to budeme opakovat než se dostaneme v čase tam kam jsme chtěli (do času b). Tak to je právě ta jednoduchá EE metoda:



$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \tau_n \mathbf{f}(t_n, \mathbf{y}_n), \quad \mathbf{y}_0 = \mathbf{y}(a) = \boldsymbol{\eta}, \\ t_{n+1} &= t_n + \tau_n, \quad t_0 = a \end{aligned} \tag{1.2.1}$$

Příklad 9. *Spočítejte přibližná řešení* y_n

$$\begin{aligned} y' &= y^2, \\ y(0) &= \frac{1}{2} \end{aligned}$$

na intervalu $[a,b] = [0,1]$ *pro dělení* $t = [0, 0.5, 0.75, 1]$*, porovnejte s přesným řešením, spočtěte globální diskretizační chybu* $e_n = y(t_n) - y_n$ *a relativní globální diskretizační chybu* $re_n = \frac{y(t_n)-y_n}{y_n}$.

Přesné řešení je

$$y(t) = \frac{1}{2-t}.$$

Počítáme EE metodou

$$\begin{aligned} y_0 &= \frac{1}{2}, \\ y_1 &= y_0 + \tau_0 f(t_0, y_0) = \frac{1}{2} + 0.5\left(\frac{1}{2}\right)^2 = \frac{5}{8} = 0.625, \\ y_2 &= y_1 + \tau_1 f(t_1, y_1) = \frac{5}{8} + 0.25\left(\frac{5}{8}\right)^2 = \frac{185}{256} = 0.7227, \\ y_3 &= y_2 + \tau_2 f(t_2, y_2) = \frac{185}{256} + 0.25\left(\frac{185}{256}\right)^2 = 0.8532. \end{aligned}$$

Výsledky uspořádáme do tabulky:

n	t_n	y_n	$y(t_n)$	e_n	re_n
0	0	0.5	0.5	0	0
1	0.5	0.625	0.66667	0.041667	0.0625
2	0.75	0.72266	0.8	0.077344	0.09668
3	1	0.85321	1	0.14679	0.14679

Příklad 10. *Spočtěte lokální chyby v předchozím příkladě a ověřte, že globální chyba nevzniká jejich prostou akumulací, nakreslete.*

K tomu je potřeba spočítat přesná řešení $\{u_n\}_{n=0,1,2}$ počátečních úloh

$$\begin{aligned} u_n' &= u_n^2, \\ u_n(t_n) &= y_n, \end{aligned}$$

dostaneme

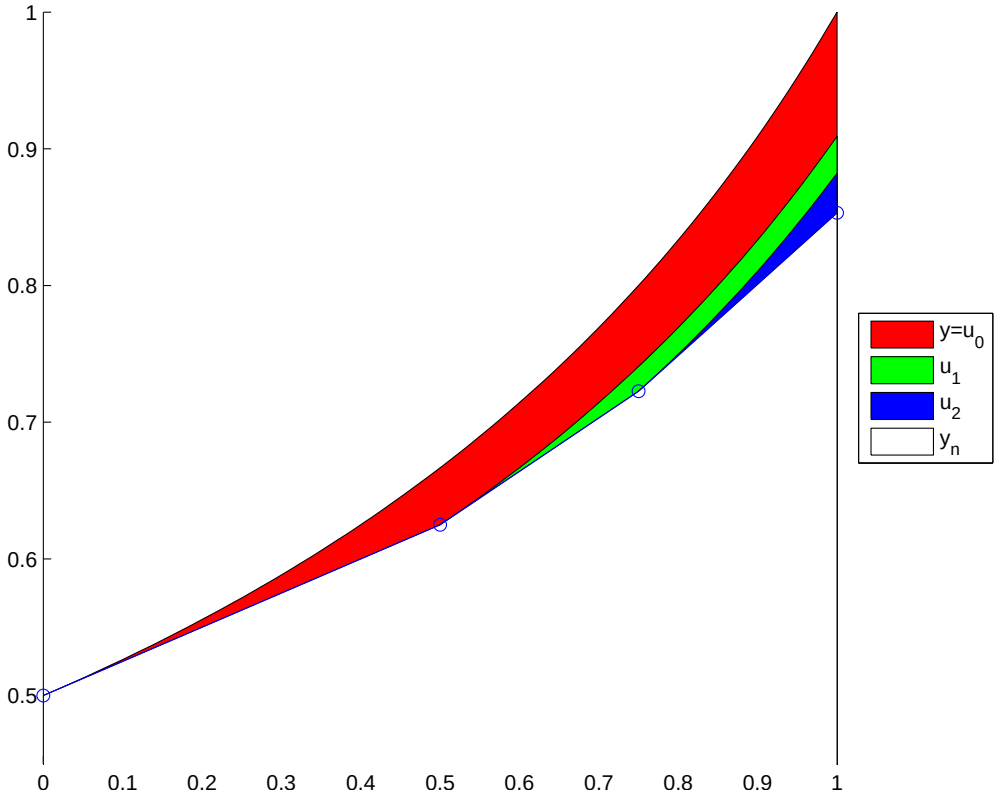
$$\begin{aligned} u_0(t) &= y(t) = \frac{1}{2-t}, \\ u_1(t) &= \frac{1}{2.1-t}, \\ u_2(t) &= \frac{1}{2.1338-t}, \end{aligned}$$

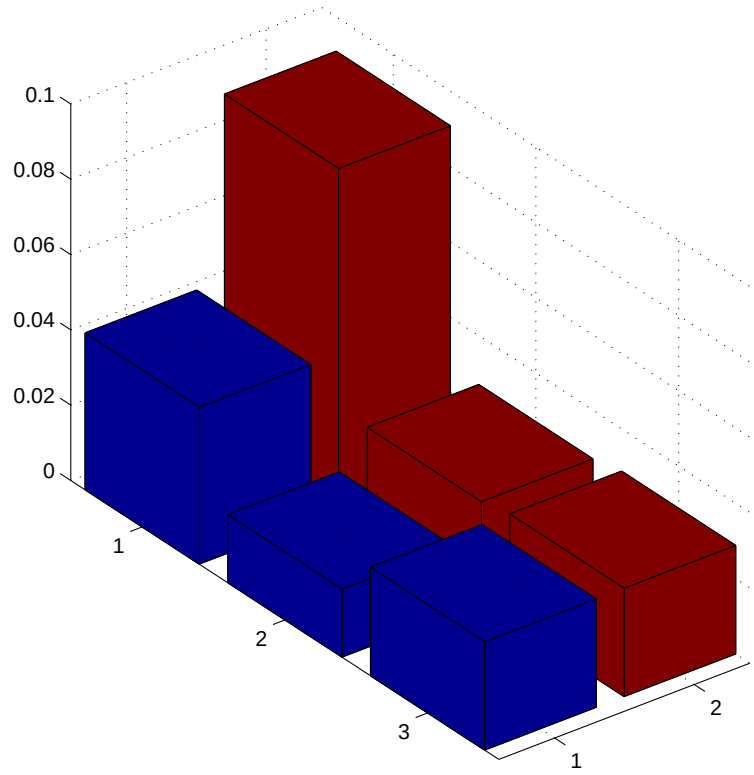
takže

$$\begin{aligned} le_1 &= e_1 = u_0(t_1) - y_1 = \frac{2}{3} - \frac{5}{8} = 0.041667, \\ le_2 &= u_1(t_2) - y_2 = 0.7407 - 0.72266 = 0.0181, \\ le_3 &= u_2(t_3) - y_3 = 0.8820 - 0.85321 = 0.0288. \end{aligned}$$

Nyní

$$\begin{aligned} le_1 &= e_1 \\ 0.0598 = le_1 + le_2 &\neq e_2 = 0.077344 \\ 0.0885 = le_1 + le_2 + le_3 &\neq e_3 = 0.14679 \end{aligned}$$





Příklad 11. Spočítejte přibližná řešení diferenciální rovnice

$$y' = \sin(ty)$$

pro počáteční podmínky

$$\begin{aligned} y(a_k) &= \eta_k \\ a_k &= 0 \\ \eta_k &= 1, 2, 3, 4, 5 \end{aligned}$$

EE metodou s krokem τ=0.1na intervalu $[a, b]$=[0,5]. Vykreslete do společného obrázku a přidejte směrové pole.

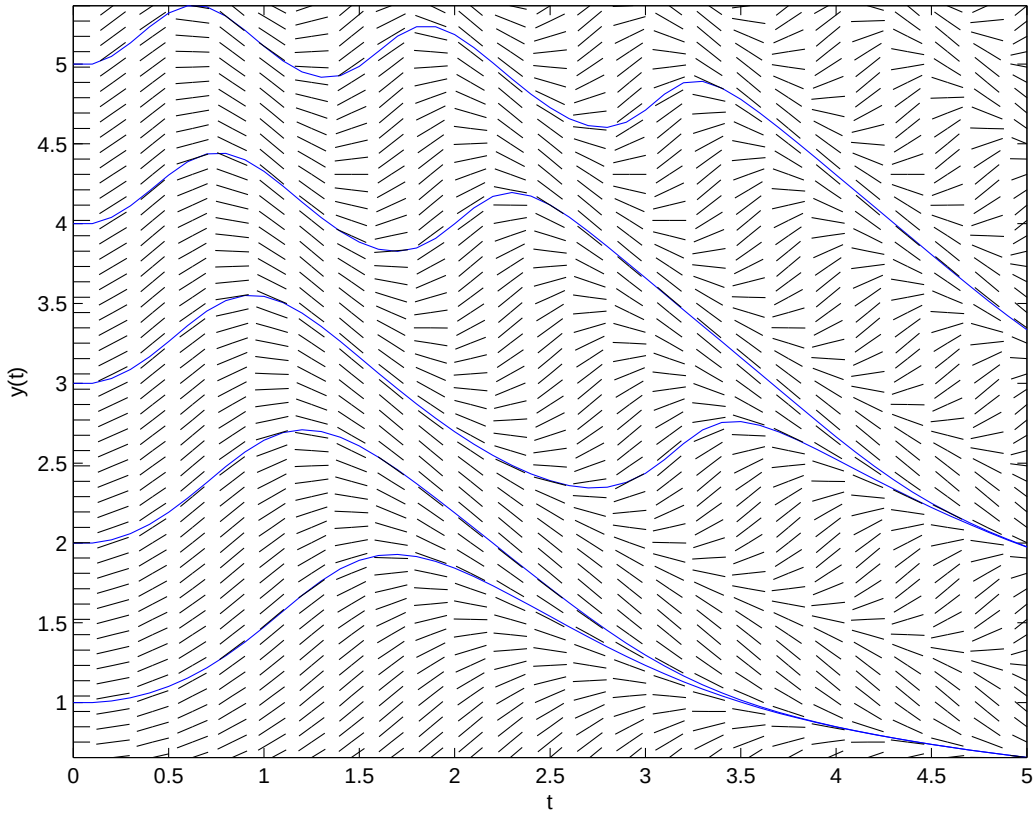
EE.m

```
function [t,y] = EE(f, t, eta)
% y'(t) = f(y,t), y(a)=eta
% f je skalarni nebo vektorova funkce
% kompatibilita s odexx
eta = eta(:);
% explicitni Eulerova metoda
Q = length(t);
y = zeros(length(eta),Q); y(:,1)=eta;
for n=1:Q-1
    tau = t(n+1)-t(n);
    y(:,n+1) = y(:,n) + tau*f(t(n),y(:,n));
end
% kompatibilita s odexx
y = y';
end
```

s3.m

```
cls;
```

```
f = @(t,y) sin(t.*y);
a=0; b=5;
t = a:0.1:b;
ymin = 1;
ymax = 1;
for eta = 1:5
    [t,y{eta}] = EE(f, t, eta);
    plot(t,y{eta}); hold on;
    ymin = min(ymin,min(y{eta}));
    ymax = max(ymax,max(y{eta}));
end
d = 0.0;
box = [a b ymin-d ymax+d];
axis(box);
n = [50 25];
[T, Y, F, h] = dfield(f,box,n,'normalize',true,'scale',0.4,...
    'ShowArrowHead','off','Color','k');
uistack(h(:),'top');
print -depsc o3
```



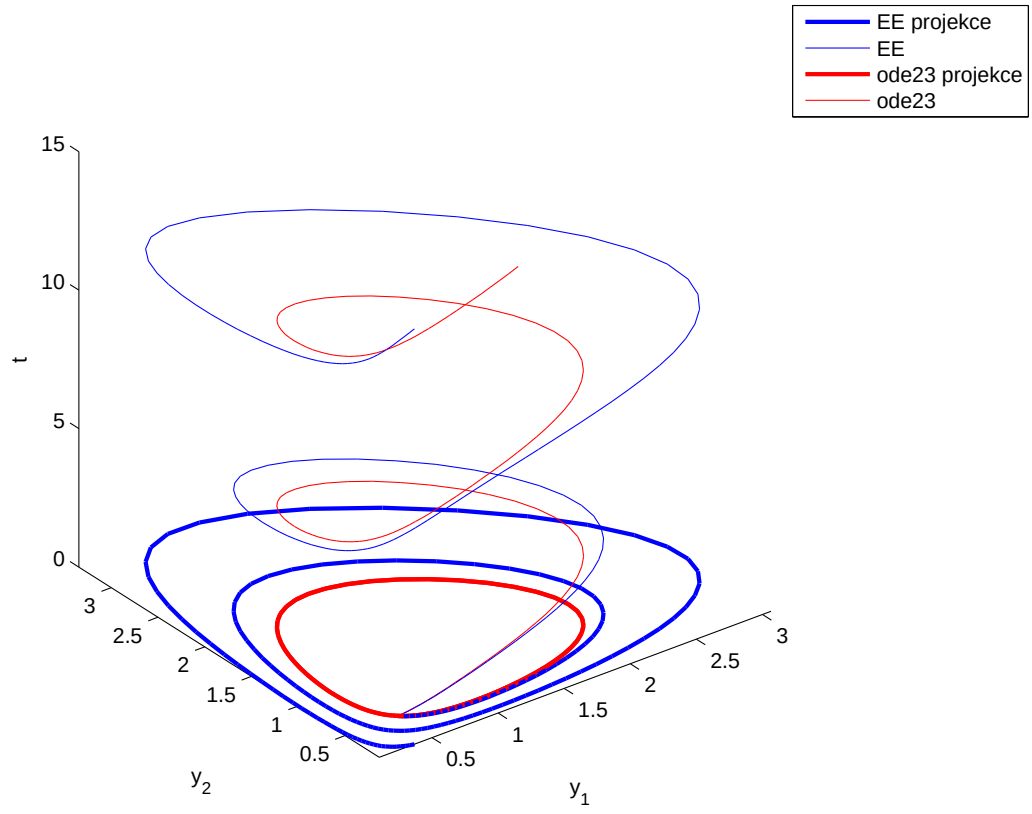
Nyní se zdá, že řešit ODR je velmi jednoduché, ale není tomu tak! Zkusíme jak se EE vyrovná s Lhotka-Volterovou soustavou.

Příklad 12. Spočítejte přibližné řešení soustavy 1.1.3 s počáteční podmínkou $\eta = [0.5,\ 0.5]$a nakreslete graf řešení v proměnných y_1, y_2a t a rovněž projekci (fázový portrét) v rovině y_1, y_2. Použijte časový interval $[0,\ 15]$. Výsledek porovnejte s výpočtem realizovaným pomocí v Matlabu vestavěné funkce **ode23**.

s4.m

```
cls;
%parametry
a=1; b=1; c=1; d=1;
% y1' = f1(y1,y2)
```

```
% y2' = f2(y1,y2)
f1 = @(y1,y2) a*y1-b*y1.*y2;
f2 = @(y1,y2) -c*y2+d*y1.*y2;
box = [0,2.5,0,2.5];
f = @(t,y) [f1(y(1),y(2)); f2(y(1),y(2))];
a=0; b=15;
t = a:0.1:b;
eta = [0.5  0.5];
[t,y] = EE(f, t, eta);
h=plot(y(:,1),y(:,2),'LineWidth',2); hold all;
plot3(y(:,1),y(:,2),t,'Color',get(h,'Color'));
[t,y] = ode23(f, t, eta);
h=plot(y(:,1),y(:,2),'LineWidth',2);
plot3(y(:,1),y(:,2),t,'Color',get(h,'Color'));
axis tight;
view(3)
set(gca,'Box','off');
legend('EE projekce','EE','ode23 projekce','ode23');
xlabel('y_1'); ylabel('y_2'); zlabel('t');
print -depsc o4
```



Cvičení 13. Kolik je kolik a kolik.

1.2.1. Řád a stabilita EE

Z teorie víme, že Eulerova metoda pro 1.1.2 konverguje. Ve skutečnosti je EE metoda základem důkazu existenční věty. V literature věnované numerickým metodám řešení ODR, např. [8, 10], najdeme zajímavý odhad chyby. Nejdříve si definujme po částech lineární interpolant (tzv. Eulerovy polygony)

$$y_\tau(t) = y_n \frac{t_{n+1}-t}{\tau_n} + y_{n+1}\frac{t-t_n}{\tau_n} \quad \text{pro} \quad t \in [t_n, t_{n+1}],$$

takže y_τ je spojitá, po částech lineární funkce procházející body $[t_n, y_n]$. Pro dostatečně jemné dělení τ platí

$$|y(t) - y_\tau(t)| < C\tau_{\max}, \tag{1.2.2}$$

kde $\tau_{\max} = \max \tau_n$ a konstanta C nezávisí na τ , ale závisí na odhadech pravé strany v okolí řešení

$$|f| \le A, \quad \left|\frac{\partial f}{\partial t}\right| \le M, \quad \left|\frac{\partial f}{\partial y}\right| \le L,$$

takto

$$C = \frac{M + AL}{L}\left(\mathrm{e}^{L(b-a)} - 1\right).$$

Protože τ je v tomto odhadu v první mocnině říkáme, že EE metoda je prvního řádu přesnosti. Nerovnost 1.2.2 je podrobnější formou věty o konvergenci EE, pro libovolný bod $t \in [a, b]$ platí: při zjemňování dělení τ konverguje hodnota Eulerova polygonu k řešení počáteční úlohy. To je to co vidíme na obrázcích pokud zadáváme kreslení čarou, vidíme Eulerovy polygony.

Zabývejme se jednoduchým ale důležitým příkladem.

Příklad 14. Zkonstruujte posloupnost aproximující Eulerovo číslo e.
Takovouto posloupnost psokytuje metoda EE, stačí zvolit počáteční úlohu1.1.2 a položit $a = 0$ a $\eta = 1$, přesné řešení bude $y = \mathrm{e}^{\mathrm{x}}$ a spočítat přibližné řešení v bodě $x = 1$. Volíme tedy $b = 1$ a podle 1.2.2 bude

$$\lim_{Q\to\infty} y_Q = \mathrm{e}.$$

Člen y_Q lze ale vyjádřit vzorcem

$$\begin{aligned} y_Q &= y_{Q-1} + \tau y_{Q-1} = (1+\tau)\, y_{Q-1} \\ &= (1+\tau)^2\, y_{Q-2} = \cdots = (1+\tau)^Q y_0 \\ &= \left(1 + \frac{1}{Q}\right)^Q \end{aligned}$$

Pišme n místo Q a vidíme slavnou posloupnost

$$\lim_{n\to\infty}\left(1 + \frac{1}{n}\right)^n = \mathrm{e},$$

kterou se snad poprvé zabýval Jacob Bernoulli.

Cvičení 15. Zkonstruujte posloupnost aproximující číslo e^{-1}, a obecně e^{λ}.

Příklad 16. Použijte odhad 1.2.2 na úlohu $y' = -100y$, $y(0) = 1$, $b = 1$ a porovnejte se skutečnou chybou.
Zde je $y = \mathrm{e}^{-100\mathrm{t}}$, takže okolím řešení může být čtverec $[0, 1] \times [0, 1]$ a odhady jsou

$$|-100y| \le A = 100, \quad \left|\frac{\partial f}{\partial t}\right| \le M = 0, \quad \left|\frac{\partial f}{\partial y}\right| \le L = 100$$

a konstanta

$$C = \frac{0 + 100^2}{100}\left(\mathrm{e}^{100(1-0)} - 1\right) = 100\left(\mathrm{e}^{100} - 1\right) = 2.6881 \times 10^{45}$$

je astronomicky velká. Zvolme $Q = 1000$ a rovnoměrné dělení, potom odhad chyby 1.2.2 je 2.6881×10^{42} avšak skutečná chyba je prakticky nulová

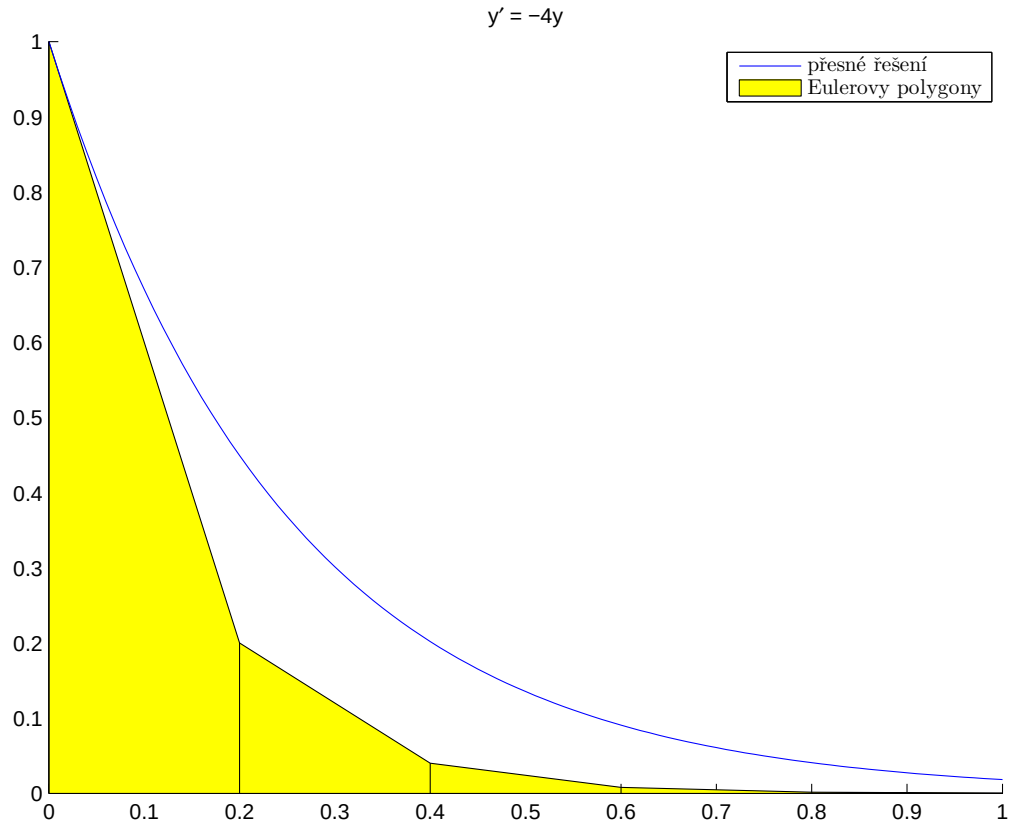
$$\left|e^{-100} - \left(1 - \frac{100}{1000}\right)^{1000}\right| = 3.7026 \times 10^{-44}$$

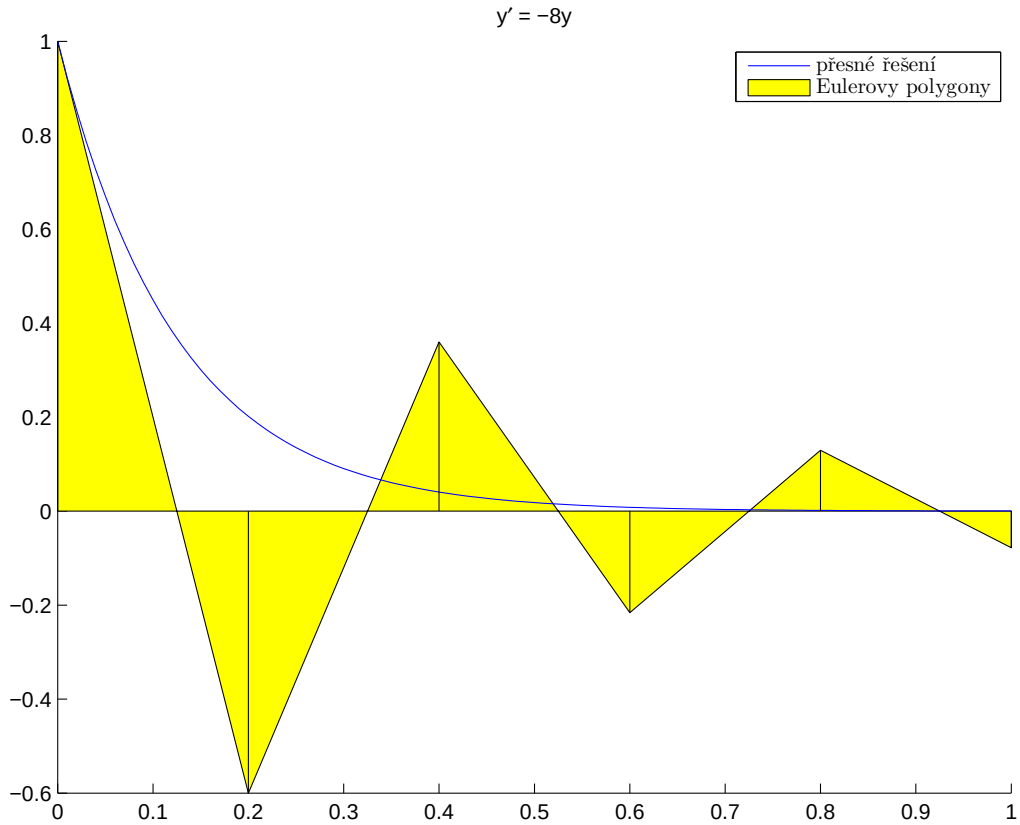
K řízení chyby není tedy odhad 1.2.2 použitelný.

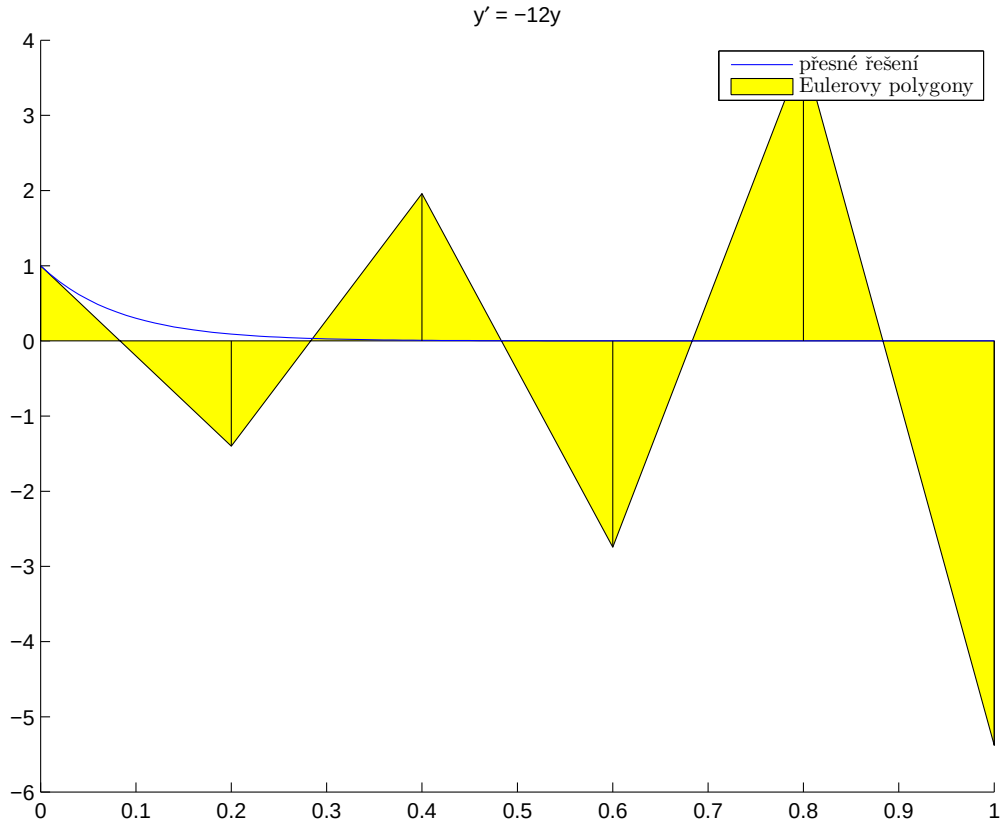
Příklad 17. Nakreslete Eulerovy polygony tak aby byl zjevný jejich po částech lineární charakter. Zvolte interval $[a, b] = [0, 1]$, rovnoměrné dělení $\tau = \frac{1}{5}$a v rovnici 1.1.1 $\lambda = -4$, $\lambda = -8$ a $\lambda = -12$, počáteční podmínka budiž $y(0) = 1$.

s5.m

```
cls;
% y' = la*y
fla = @(t,y,la) la*y;
yla = @(t,la) exp(la*t);
a=0; b=1;
t = a:0.2:b;
tt = a:0.01:b;
eta = 1;
la = [-4,-8,-12];
modif = ['a' 'b' 'c'];
for i = 1:length(la);
    f = @(t,y) fla(t,y,la(i));
    y = @(t) yla(t,la(i));
    [t,yn] = EE(f, t, eta);
    clf; hold on;
    h1=fill([a t b],[0; yn; 0],'y','edgecolor','k');
    h2=plot(tt,y(tt));
    z = zeros(size(t));
    plot([t; t], [z; yn'], 'k');
    h=legend([h2,h1],'p\v{r}esn\''{e}␣\v{r}e\v{s}en\''{i}','Eulerovy␣polygony');
    set(h,'Interpreter','LaTeX');
    title(['y\prime␣=␣' int2str(la(i)) 'y']);
    print('-depsc', ['o5' modif(i)]);
end
```







Obrázky jsou docela poučné. Pro $\lambda = -4$ vidíme, že přibližné řešení odpovídá našemu očekávání, EE je jen prvního řádu. Řešení je dosti nepřesné, chyby jednotlivých kroků se však kupodivu neakumulují. Naproti tomu pro hodnotu $\lambda = -8$ vidíme, že se objevily oscilace kolem přesného řešení, důvodem je záporná hodnota y_n pro lichá n. Poslední případ $\lambda = -12$ je ovšem alarmující, kromě ocilací vidíme prudký nárůst chyby. A především zatímco pro přesné řešení platí $\lim_{t\to\infty} y(t) = 0$ pro přibližné je zřejmě $\lim_{n\to\infty} |y_n| = \infty$! Víme, že v případě této

jednoduché lineární rovnice má krok metody tvar

$$y_{n+1} = (1 + \lambda\tau)\, y_n.$$

Je potřeba pohlídat aby faktor $1 + \lambda\tau$ nenabýval v modulu hodnot větších než 1! Právě jsme objevili oblast stability pro EE je dána podmínkou

$$\tau < \frac{2}{|\lambda|}.$$

1.2.2. *Praktický odhad chyby a extrapolace

Když je pravá strana diferenciání rovnice a řešení dostatečně hladké lze dokázat asymptotický odhad (viz [2])

$$y(t) - y_\tau(t) = C\,(t)\,\tau + \mathcal{O}\left(\tau^2\right), \tag{1.2.3}$$

kde funkce chyby $C\,(t)$ splňuje lineární diferenciální úlohu

$$\begin{aligned} C' &= f'_y(t, y(t))C + \frac{1}{2}y''(t), \\ C\,(a) &= 0. \end{aligned} \tag{1.2.4}$$

Příklad 18. Spočtěme funkci $C\,(t)$ pro úlohu

$$\begin{aligned} y' &= -y, \\ y\,(0) &= 1 \end{aligned}$$

a porovnejme s výpočtem. Počáteční úloha 1.2.4

$$\begin{aligned} C' &= -C + \frac{1}{2}e^{-t}, \\ C\,(0) &= 0 \end{aligned}$$

má řešení

$$C\,(t) = \frac{1}{2}te^{-t},$$

takže

$$y(t_n) - y_\tau(t_n) \approx \frac{1}{2}t_n e^{-t_n}\tau.$$

Počítejme EE metodou dvě přibližná řešení na dvou ekvidistatních sítích se vzdáleností uzlů τ a 2τ

$$\begin{aligned} \mathbf{y}_\tau\,(t_{n+1}) &= \mathbf{y}_\tau\,(t_n) + \tau\mathbf{f}\,(t_n, \mathbf{y}_\tau\,(t_n))\,, \\ \mathbf{y}_{2\tau}\left(t_{2(n+1)}\right) &= \mathbf{y}_{2\tau}\,(t_{2n}) + 2\tau\mathbf{f}\,(t_{2n}, \mathbf{y}_{2\tau}\,(t_{2n}))\,, \\ \mathbf{y}_\tau\,(t_0) = \mathbf{y}_{2\tau}\,(t_0) &= \eta. \end{aligned}$$

Podle 1.2.3 máme

$$\begin{aligned} y(t) - y_\tau(t) &= C\,(t)\,\tau + \mathcal{O}\left(\tau^2\right), \\ y(t) - y_{2\tau}(t) &= 2C\,(t)\,\tau + \mathcal{O}\left(\tau^2\right). \end{aligned}$$

Odečteme od dvojnásobku první rovnice druhou a dostaneme pozoruhodný odhad

$$y(t) - [2y_\tau(t) - y_{2\tau}(t)] = \mathcal{O}\left(\tau^2\right). \tag{1.2.5}$$

Veličina

$$y_\tau^*(t)=2y_\tau(t) - y_{2\tau}(t)$$

je řádově lepší aproximací řešení. Říkáme, že jsme ji získali extrapolací. Zanedbáním pravé strany ve 1.2.5 dostáváme praktický odhad chyby původní metody

$$y(t) - y_\tau(t) \approx \mathrm{est}_\tau(t) := y_\tau(t) - y_{2\tau}(t),$$

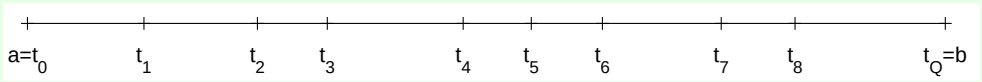
takže

$$y_\tau^*(t)=y_\tau(t) + \mathrm{est}_\tau(t).$$

$\tau = 0.1$

t	$y(t) - y_\tau(t)$	$\frac{1}{2}te^{-t}\tau$	$y_\tau(t) - y_{2\tau}(t)$	$y_\tau^*(t)$	$y(t) - y_\tau^*(t)$
0	0	0	0	1	0
1	0.019201	0.018394	0.020998	0.36968	-0.0017974
2	0.013759	0.013534	0.014202	0.13578	-0.00044384
3	0.0073959	0.0074681	0.0072068	0.049598	0.00018912
4	0.0035348	0.0036631	0.0032517	0.018033	0.00028309
5	0.0015842	0.0016845	0.0013759	0.0065297	0.00020829
6	0.00068174	0.00074363	0.00055907	0.0023561	0.00012267

1.3. Implicitní Eulerova metoda (IE)



$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \tau_n \mathbf{f}(t, \mathbf{y}_{n+1}), \quad \mathbf{y}_0 = \mathbf{y}\,(a) = \eta, \\ t_{n+1} &= t_n + \tau_n, \quad t_0 = a \end{aligned} \tag{1.3.1}$$

Tato metoda představuje řešení soustavy obecně nelineárních rovnic v každém kroku. Ovšem v případě rovnice 1.1.1 je to snadné

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \lambda\tau_n \mathbf{y}_{n+1} \\ &\Downarrow \\ \mathbf{y}_{n+1} &= \frac{1}{(1 - \lambda\tau_n)}\mathbf{y}_n \end{aligned}$$

Situace se stabiltou je tu zcela jiná, pro$\lambda < 0$ je fakor $\frac{1}{(1-\lambda\tau_n)} < 1$ pro libovolné τ_n!

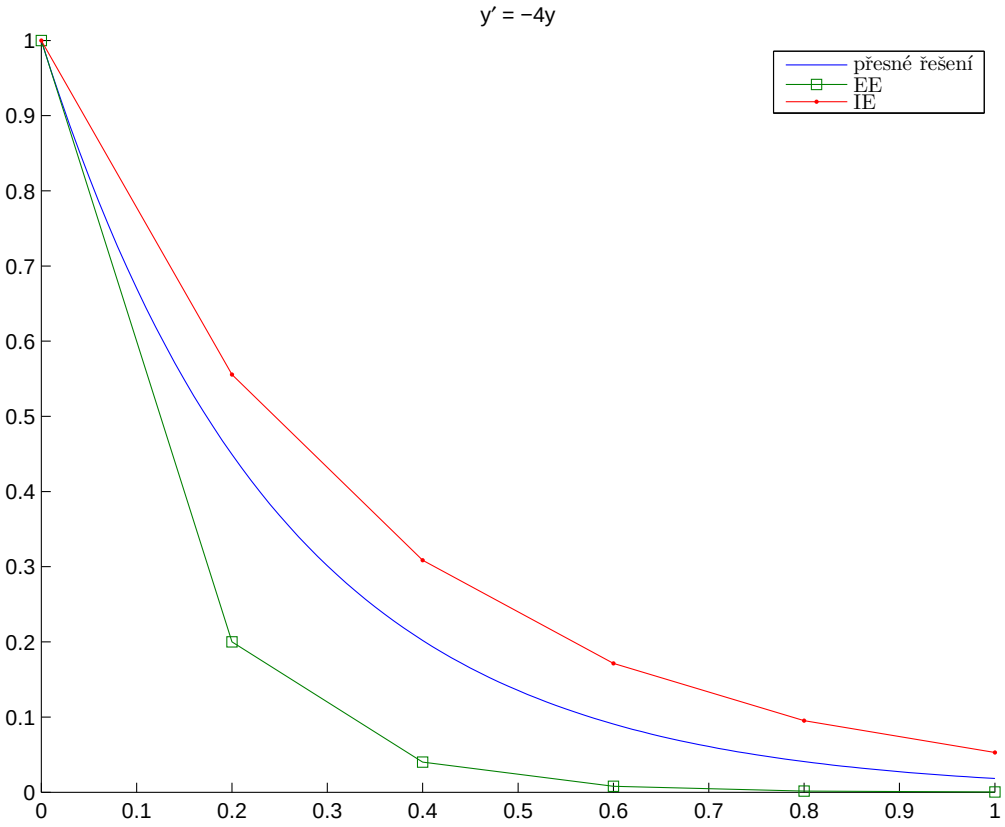
ie.m

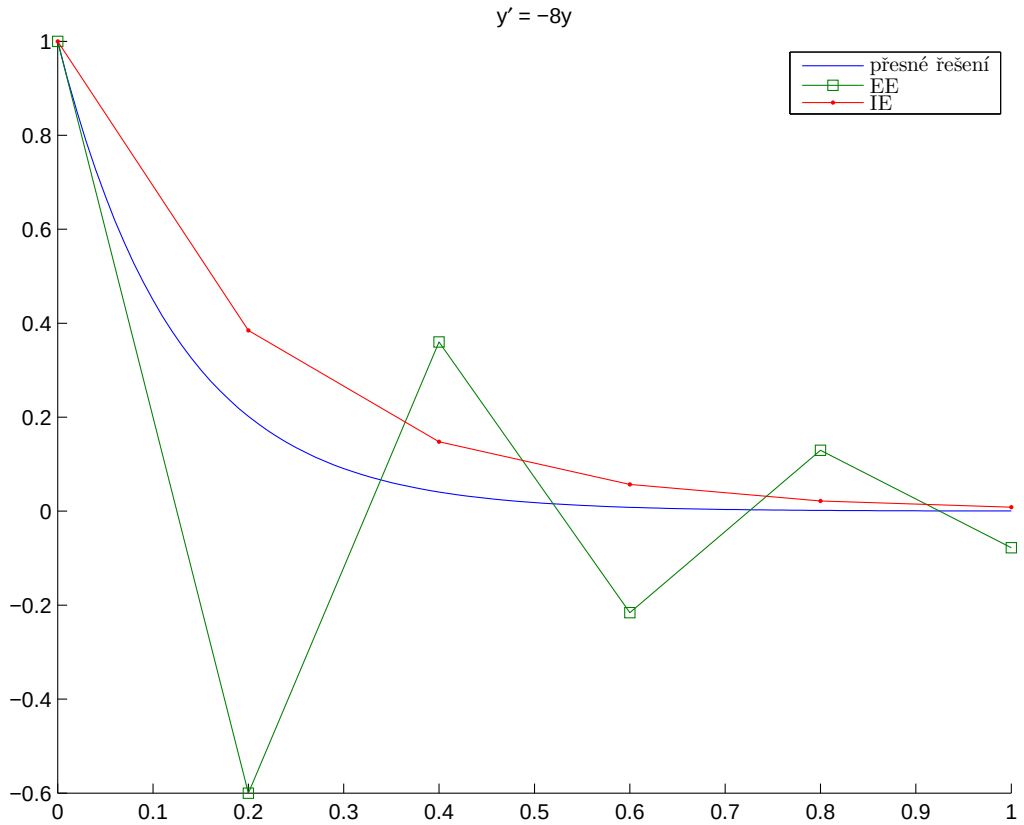
```
function [t,y] = IE(f, t, eta)
% y'(t) = f(y,t), y(a)=eta
% f je skalarni funkce
% implicitni Eulerova metoda
Q = length(t);
y = zeros(Q,1); y(1)=eta;
for n=1:Q-1
    tau = t(n+1)-t(n);
    y(n+1) =fzero(@(z) z - y(n) - tau*f(t(n),z),y(n));
end
end
```

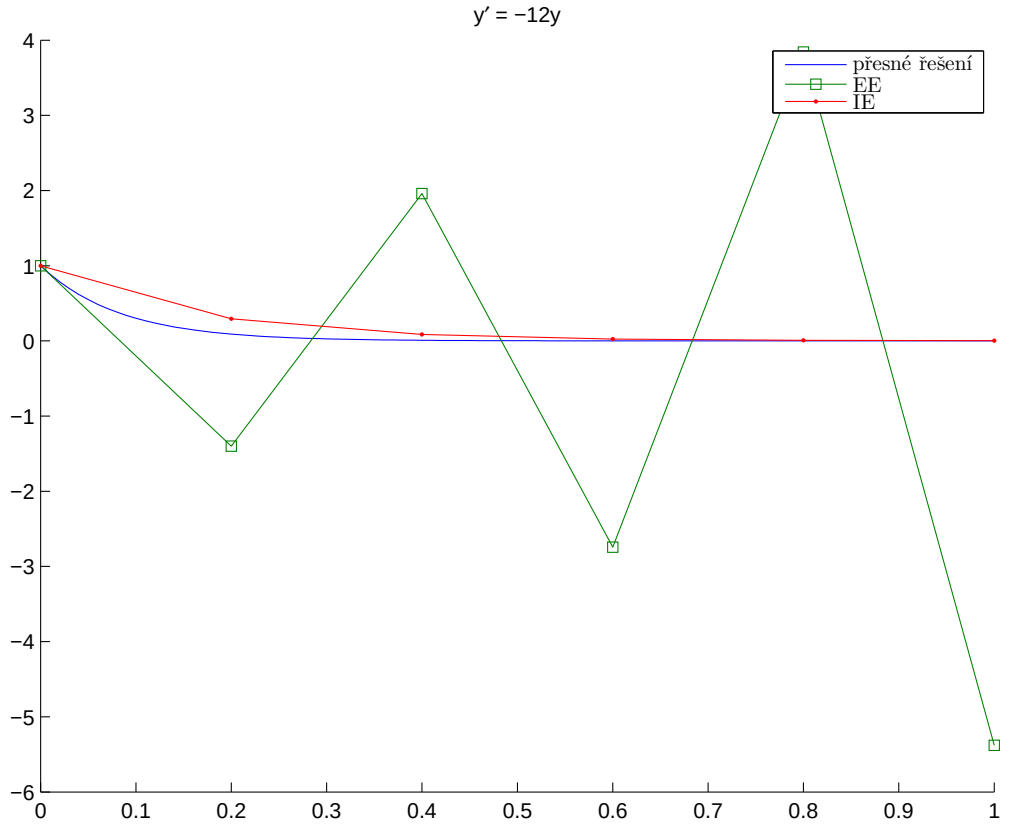
Příklad 19. Porovnejte metody ve stejném grafu

s6.m

```
cls;
% y' = la*y
fla = @(t,y,la) la*y;
yla = @(t,la) exp(la*t);
a=0; b=1;
t = a:0.2:b;
tt = a:0.01:b;
eta = 1;
la = [-4 -8 -12];
modif = ['a' 'b' 'c'];

for i = 1:length(la);
    clf;
    hold all;
    f = @(t,y) fla(t,y,la(i));
    y = @(t) yla(t,la(i));
    [t,yn] = EE(f, t, eta);
    [t,iyn] = IE(f, t, eta);
    plot(tt,y(tt),t,yn,'sq-',t,iyn,'.-');
    h=legend('p\v{r}esn\''{e}␣\v{r}e\v{s}en\''{i}','EE','IE');
    set(h,'Interpreter','LaTeX');
    title(['y\prime␣=␣' int2str(la(i)) 'y']);
    print('-depsc', ['o6' modif(i)]);
end
```







1.4. TR metoda

Eulerovy metody lze odvodit z velmi užitečné integrální formulace počáteční úlohy

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(x, y(x))dx.$$

V EE nahradíme určitý integrál na parvé straně numerickou kvadraturou

$$\int_{t_n}^{t_{n+1}} f(x, y(x))dx \approx f(t_n, y(t_n))\tau_n \approx f(t_n, y_n)\tau_n$$

a v IE zase

$$\int_{t_n}^{t_{n+1}} f(x, y(x))dx \approx f(t_n, y(t_{n+1}))\tau_n \approx f(t_n, y_{n+1})\tau_n.$$

Použitím lichoběžníkového pravidla vznikne zase TR-metoda

$$\int_{t_n}^{t_{n+1}} f(x, y(x))dx \approx \frac{f(t_n, y_n) + f(t_n, y_{n+1})}{2}\tau_n$$

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \tau_n \frac{\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_n, \mathbf{y}_{n+1})}{2}, \quad \mathbf{y}_0 = \mathbf{y}(a) = \eta, \\ t_{n+1} &= t_n + \tau_n, \quad t_0 = a \end{aligned}$$

Tato metoda je opět implicitní. Její přínos je, že je to naše první metoda druhého řádu. Co se tím myslí? Napišme Taylorův rozvoj přesného řešení se středem v bodě t_n a pišme τ místo τ_n

$$y(t_{n+1}) = y(t_n) + y'(t_n)\tau + y''(t_n)\frac{\tau^2}{2} + \mathcal{O}(\tau^3) \tag{1.4.1}$$

a také Taylorův rozvoj první derivace řešení

$$\begin{aligned} y'(t_{n+1}) &= y'(t_n) + y''(t_n)\tau + \mathcal{O}(\tau^2) \\ &\Downarrow \\ y''(\tau_n)\tau &= y'(t_{n+1}) - y'(t_n) + \mathcal{O}(\tau^2) \end{aligned} \tag{1.4.2}$$

a dosadíme 1.4.2 do 1.4.1

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + y'(t_n)\tau + \left[y'(t_{n+1}) - y'(t_n) + \mathcal{O}(\tau^2)\right]\frac{\tau}{2} + \mathcal{O}(\tau^3) \\ &= y(t_n) + \frac{y'(t_n) + y'(t_{n+1})}{2}\tau + \mathcal{O}(\tau^3) \end{aligned} \tag{1.4.3}$$

Právě jsme dokázali, že lokální diskretizační chyba

$$\mathrm{lte}_n = y(t_{n+1}) - y(t_n) - \frac{y'(t_n) + y'(t_{n+1})}{2}\tau = \mathcal{O}(\tau^3),$$

takže je to opravdu metoda 2. řádu přesnosti. Oblast absolutní stability obsahuje celou zápornou reálnou poloosu, pro libovolně velké τ zůstaneme v oblasti stability.

1.5. *Taylorova metoda

Explicitní Eulerova metoda je rovněž speciálním případem tzv. metody Taylorových řad, TS metody prvního řádu, zkráceně TS(1) metody. Obecná TS(r) metoda je tvaru

$$\begin{aligned}
\mathbf{y}_{n+1} &= \mathbf{y}_n + \mathbf{y}'_n\tau_n + \mathbf{y}''_n\frac{\tau_n^2}{2!} + \cdots + \mathbf{y}_n^{(r)}\frac{\tau_n^r}{r!}, \quad \mathbf{y}_0 = \mathbf{y}(a) = \eta, \\
\mathbf{y}'_n &= \mathbf{f}(t_n, \mathbf{y}_n) \\
\mathbf{y}''_n &= \mathbf{f}'_t(t_n, \mathbf{y}_n) + \mathbf{f}'_{\mathbf{y}}(t_n, \mathbf{y}_n)\mathbf{f}(t_n, \mathbf{y}_n) \\
&\cdots \\
t_{n+1} &= t_n + \tau_n, \quad t_0 = a
\end{aligned} \tag{1.5.1}$$

Metoda je to explicitní r-tého řádu. Aplikace je velmi sympatická pro rovnici 1.1.1, kde snadno získáme další derivace (místo teček) postupným derivováním diferenciální rovnice

$$\begin{aligned}
y' &= \lambda y \\
y'' &= \lambda y' = \lambda^2 y \\
&\cdots \\
y^{(r)} &= \lambda y^{(r-1)} = \lambda^r y
\end{aligned}$$

takže TS(r) metoda má tvar

$$\begin{aligned}
y_{n+1} &= \left(1 + \lambda\tau + \frac{(\lambda\tau)^2}{2!} + \cdots + \frac{(\lambda\tau)^r}{r!}\right) y_n, \quad \mathbf{y}_0 = \mathbf{y}(a) = \eta, \\
y_n &= [P(\lambda\tau)]^n \eta
\end{aligned}$$

kde

$$P(z) = \sum_{i=0}^{r} \frac{z^i}{i!}.$$

Metoda bude absolutně stabilní, když modul $|P(z)| < 1$.

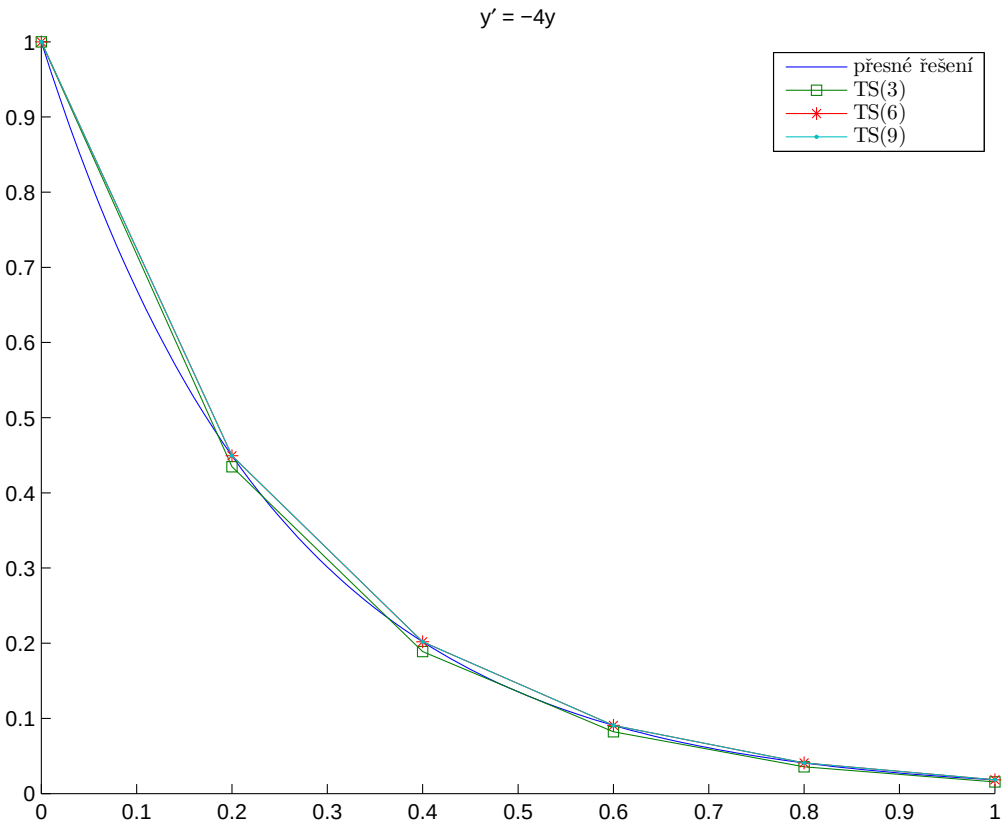
texp.m

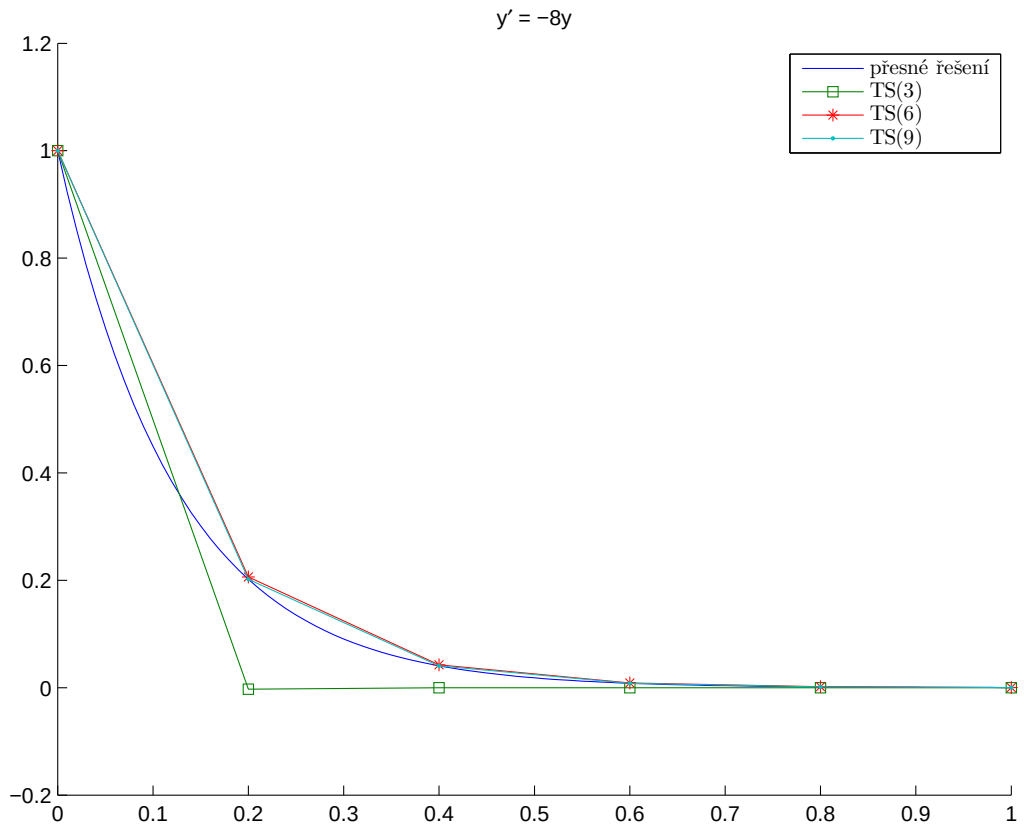
```
function [y, p] = texp( r, z, n)
if nargin==2
    n=1;
end
p = zeros(1,r+1);
p(end) = 1;
for i=1:r
    p(end-i) = p(end-i+1)/i;
end
if nargin==1
    y = [];
    return
end
y = zeros(1,n+1);
y(1) =1;
pz = polyval(p,z);
for i=1:n
    y(i+1)=pz*y(i);
end
end
```

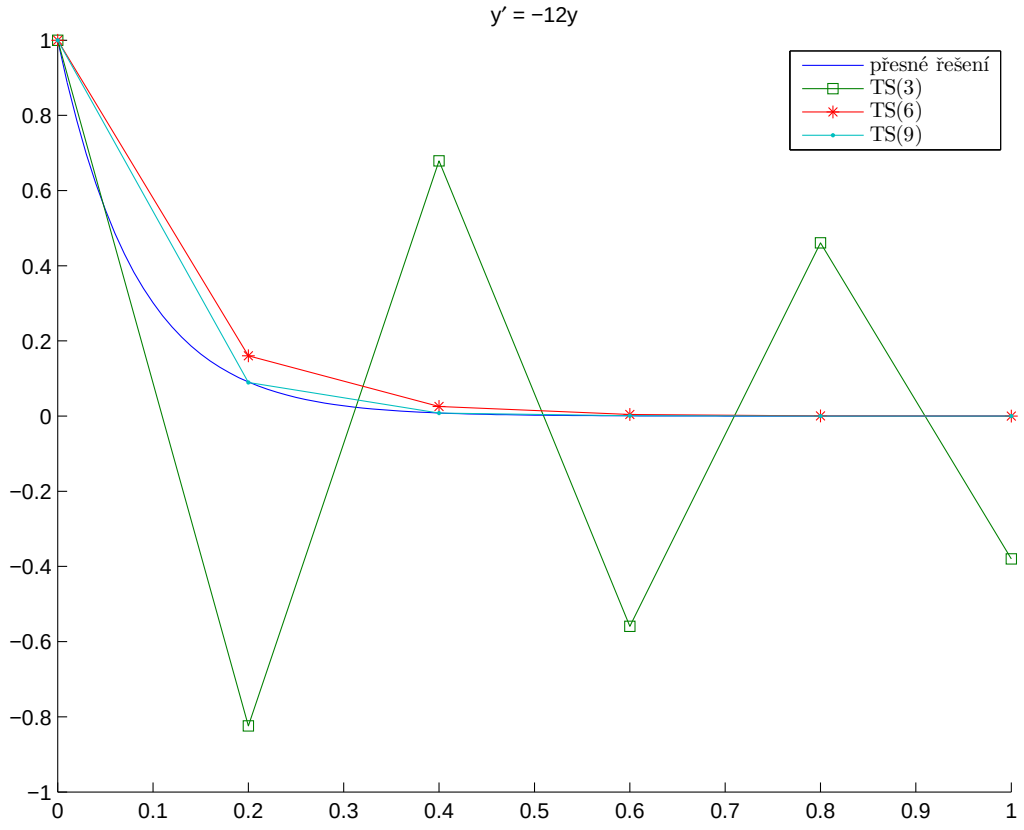
Příklad 20. Porovnejte metody ve stejném grafu

s7.m

```
cls;
% y' = la*y
fla = @(t,y,la) la*y;
yla = @(t,la) exp(la*t);
a=0; b=1;
tau=0.2;
t = a:tau:b;
tt = a:0.01:b;
eta = 1;
la = [-4 -8 -12];
modif = ['a' 'b' 'c'];

for i = 1:length(la);
    clf;
    hold all;
    f = @(t,y) fla(t,y,la(i));
    y = @(t) yla(t,la(i));
    yn1 = texp(3,tau*la(i),5);
    yn2 = texp(6,tau*la(i),5);
    yn3 = texp(9,tau*la(i),5);
    plot(tt,y(tt),t,yn1,'sq-',t,yn2,'*-',t,yn3,'.-');
    h=legend('p\v{r}esn\''{e}␣\v{r}e\v{s}en\''{i}','TS(3)','TS(6)','TS(9)');
    set(h,'Interpreter','LaTeX');
    title(['y\prime␣=␣' int2str(la(i)) 'y']);
    print('-depsc', ['o7' modif(i)]);
end
```



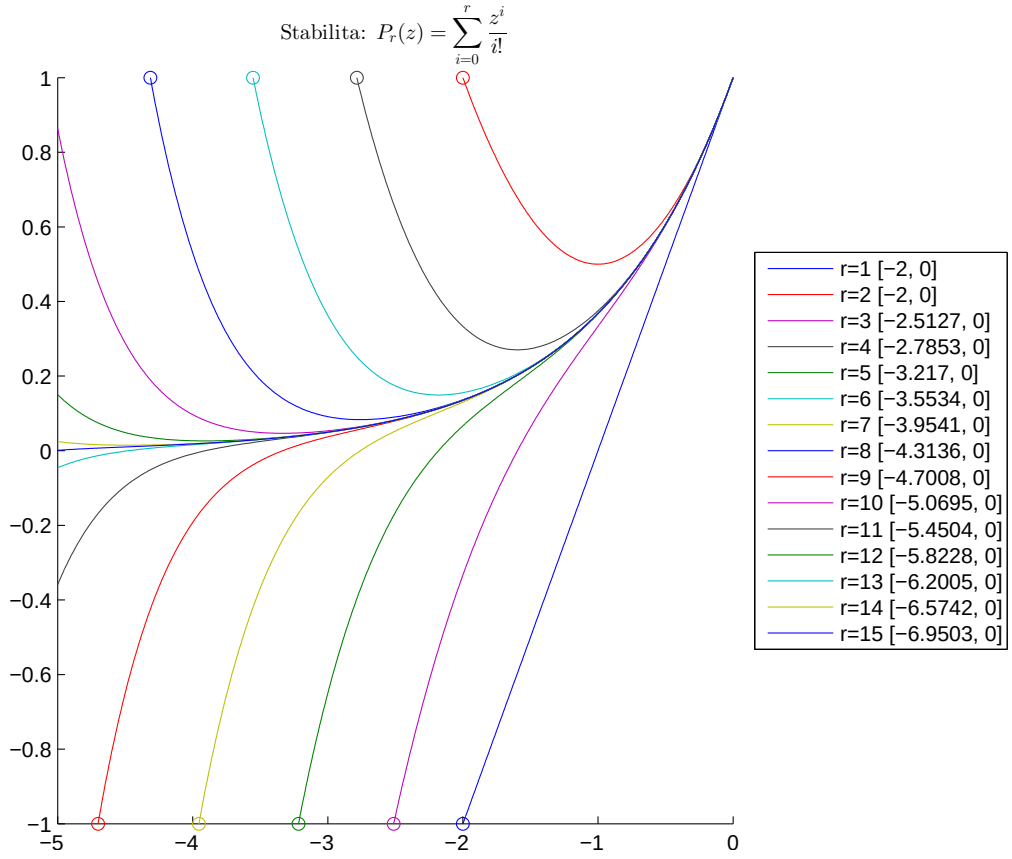




s8.m

```
cls;
a=-5; b=0;
t = a:0.01:b;
hold all;
z = -1; z0 = -2;
n = 15;
```

```
r = 1:n;
ar = cell(n,1);
ah = zeros(n,1);
for i=r
    [~,p] = texp(i);
    ah(i) = plot(t,polyval(p,t));
    c = get(ah(i),'Color');
    pp = p;
    pp(end) = pp(end)-z;
    z0 = fzero(@(x) polyval(pp,x),z0);
    plot(z0,z,'o','Color',c);
    ar{i} = ['r=' num2str(i) ' [' num2str(z0) ', 0]'];
    z = -z;
end
lh=legend(ah,ar,'Location','EastOutside');
title('Stabilita: $$P_r(z)=\sum_{i=0}^r \frac{z^i}{i!}$$','Interpreter','LaTeX')
    ;
axis([a b -1 1]);
print('-depsc', 'o8');
```



Ukažme si TS(r) metodu na soustavě 1.1.3 a objasníme tak význam symbolů v 1.5.1

$$
\begin{aligned}
\mathbf{f}(t,\mathbf{y}) &= \begin{bmatrix} f_1(t,y_1,y_2) \\ f_2(t,y_1,y_2) \end{bmatrix} = \begin{bmatrix} \alpha y_1 - \beta y_1 y_2 \\ -\gamma y_2 + \delta y_1 y_2 \end{bmatrix} \\
\mathbf{f}'_t(t_n,\mathbf{y}_n) &= \begin{bmatrix} f'_{1,t} \\ f'_{2,t} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\mathbf{f}'_{\mathbf{y}}(t,\mathbf{y}) &= \begin{bmatrix} f'_{1,y_1} & f'_{1,y_2} \\ f'_{2,y_1} & f'_{2,y_2} \end{bmatrix} = \begin{bmatrix} \alpha - \beta y_2 & -\beta y_1 \\ \delta y_2 & -\gamma + \delta y_1 \end{bmatrix} \\
\mathbf{y_n}'' = \mathbf{f}'_{\mathbf{y}}(t_n,\mathbf{y}_n)\mathbf{f}(t_n,\mathbf{y}_n) &= \begin{bmatrix} \alpha - \beta y_{2,n} & -\beta y_{1,n} \\ \delta y_{2,n} & -\gamma + \delta y_{1,n} \end{bmatrix} \begin{bmatrix} \alpha y_{1,n} - \beta y_{1,n} y_{2,n} \\ -\gamma y_{2,n} + \delta y_1 y_{2,n} \end{bmatrix} \\
\mathbf{y}_{n+1} &= \mathbf{y}_n + \mathbf{y}'_n \tau_n + \mathbf{y}''_n \frac{\tau_n^2}{2!}, \quad \mathbf{y}_0 = \mathbf{y}(a) = \eta,
\end{aligned}
$$

Dostali jsme metodu TS(2), získat metodu řádu r je ale dosti pracné a málo obecné (generické).

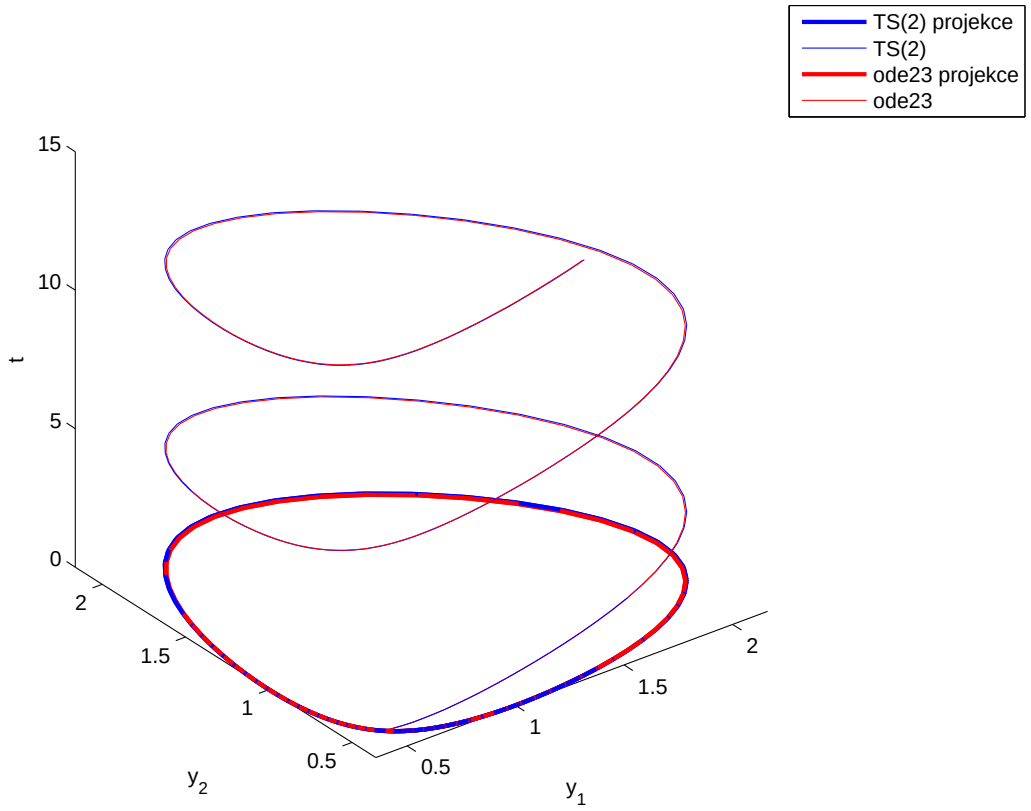
TS2.m

```
function [t, y] = TS2( T, t, eta)
% y'(t) = f(y,t), y(a)=eta
% T jsou Taylorovy koeficienty
% kompatibilita s odexx
eta = eta(:);
% TS(2) metoda
Q = length(t);
y = zeros(length(eta),Q); y(:,1)=eta;
for n=1:Q-1
    tau = t(n+1)-t(n);
    dy=T{1}; dy2=T{2};
    y(:,n+1) = y(:,n) + dy(t(n),y(:,n))*tau+dy2(t(n),y(:,n))*(tau^2/2);
end
% kompatibilita s odexx
y = y';
end
```

s9.m

```
cls;
%parametry
a=1; b=1; c=1; d=1;
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f1 = @(y1,y2) a*y1-b*y1.*y2;
f2 = @(y1,y2) -c*y2+d*y1.*y2;
f1y1 = @(y1,y2) a-b*y2;
f1y2 = @(y1,y2) -b*y1;
f2y1 = @(y1,y2) d*y2;
f2y2 = @(y1,y2) -c+d*y1;
box = [0,2.5,0,2.5];
dy{1} = @(t,y) [f1(y(1),y(2)); f2(y(1),y(2))];
f=dy{1};
dfy = @(t,y) [f1y1(y(1),y(2)) f1y2(y(1),y(2))
              f2y1(y(1),y(2)) f2y2(y(1),y(2))];
dy{2} = @(t,y) dfy(t,y)*f(t,y);
a=0; b=15;
t = a:0.1:b;
eta = [0.5 0.5];
[t,y] = TS2(dy, t, eta);
h=plot(y(:,1),y(:,2),'LineWidth',2); hold all;
```

```
plot3(y(:,1),y(:,2),t,'Color',get(h,'Color'));
[t,y] = ode23(f, t, eta);
h=plot(y(:,1),y(:,2),'LineWidth',2);
plot3(y(:,1),y(:,2),t,'Color',get(h,'Color'));
axis tight;
view(3)
set(gca,'Box','off');
legend('TS(2) projekce','TS(2)','ode23 projekce','ode23');
xlabel('y_1'); ylabel('y_2'); zlabel('t');
print -depsc o9
```



1.6. Explicitní Runge-Kuttovy metody

Explicitní Runge-Kuttovi metody, krátce RK metody, jsou praktickou alternativou Taylorových metod. Patří mezi ně EE metoda, která je současně Taylorovou metodou prvního řádu. Uveďme si další jednoduchou RK metodu, tzv. první modifikaci EE metody, zkráceně EM1 metodu

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n) \\ \mathbf{k}_2 &= \mathbf{f}\left(t_n + \frac{\tau_n}{2}, \mathbf{y}_n + \frac{\tau_n}{2}\mathbf{k}_1\right) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \tau_n \mathbf{k}_2, \quad \mathbf{y}_0 = \mathbf{y}(a) = \eta, \\ t_{n+1} &= t_n + \tau_n, \quad t_0 = a \end{aligned}$$

Rozviňme přesné řešení úlohy

$$\begin{aligned} y' &= f(t,y), \\ y(t_n) &= y_n \end{aligned}$$

do Taylorovy řady

$$y(t_n+\tau_n) = y_n + f(t_n,y_n)\tau_n + \left[f'_t(t_n,y_n) + f'_y(t_n,y_n)f(t_n,y_n)\right]\frac{\tau_n^2}{2} + \mathcal{O}\left(\tau_n^3\right).$$

Veličina y_{n+1} je ovšem také funkcí τ_n a má zase tento Taylorův rozvoj

$$y(t_n+\tau_n) = y_n + \tau_n\left[f(t_n,y_n) + f'_t(t_n,y_n)\frac{\tau_n}{2} + f'_y(t_n,y_n)\frac{\tau_n}{2}k_1\right] + \mathcal{O}\left(\tau_n^3\right),$$

Vzhledem k tomu, že $k_1 = f(t_n,y_n)$ jsou tyto rozvoje až do druhého řádu totožné! EM1 metoda spočte $\mathbf{y}_{n+1}$ se stejným řádem přesnosti jako TS(2) metoda, avšak bez potřeby derivovat soustavu diferenciálních rovnic! Místo toho vyhodnocujeme pravou stranu kromě bodu $[t_n,\mathbf{y}_n]$ ještě v bodu $\left[t_n+\frac{\tau_n}{2},\mathbf{y}_n+\frac{\tau_n}{2}\mathbf{k}_1\right]$.

Cvičení 21. Nakreslete si v matlabu směrnice k_1a k_2 pro úlohu 1.1.1 .

Butcher.m

```
function tab = Butcher( name )
names = {'EE','EM1','EM2','R2','R3','cRK4','BS32','DP54','F87'};
if nargin==0
    if nargout==1
        tab = names;
        return;
    end
    disp('Zadejte jednu z metod:')
    for i=1:length(names)
        disp(names{i});
    end
    return;
end
switch name
    case 'EE'
        tab.a = {};
        tab.c = {};
        tab.b = 1;
        tab.s = length(tab.b);
    case 'EM1'
        tab.a = {[];1/2};
        tab.c = [0, 1/2];
        tab.b = [0, 1];
        tab.s = length(tab.b);
    case 'EM2'
        tab.a = {[];1};
        tab.c = [0, 1];
        tab.b = [1/2, 1/2];
        tab.s = length(tab.b);
    case 'R2'
        tab.a = {[];2/3};
        tab.c = [0, 2/3];
        tab.b = [1/4, 3/4];
        tab.s = length(tab.b);
    case 'R3'
```

```
        tab.a = {[];1/2;[0  3/4]};
        tab.c = [0, 1/2, 3/4];
        tab.b = [2/9, 1/3, 4/9];
        tab.s = length(tab.b);
    case 'cRK4'
        tab.a = {[];1/2;[0  1/2];[0  0  1]};
        tab.c = [0, 1/2, 1/2, 1];
        tab.b = [1/6, 1/3, 1/3, 1/6];
        tab.s = length(tab.b);
    case 'BS32'
        tab.a = {[];1/2;[0  3/4];[2/9, 1/3, 4/9]};
        tab.c = [0, 1/2, 3/4, 1];
        tab.b1 = [7/24, 1/4, 1/3, 1/8];
        tab.b2 = [2/9, 1/3, 4/9, 0];
        tab.E = tab.b2-tab.b1;
        tab.s = length(tab.E);
        tab.pow = 1/3;
    case 'DP54'
        tab.a = {[];1/5;[3/40, 9/40];[44/45, -56/15, 32/9];...
            [19372/6561, -25360/2187, 64448/6561, -212/729];...
            [9017/3168, -355/33, 46732/5247, 49/176, -5103/18656];...
            [35/384, 0, 500/1113, 125/192, -2187/6784, 11/84]};
        tab.c = [0, 1/5, 3/10, 4/5, 8/9, 1, 1];
        tab.b1 = [5179/57600, 0, 7571/16695, 393/640, -92097/339200, 187/2100,
            1/40];
        tab.b2 = [35/384, 0, 500/1113, 125/192, -2187/6784, 11/84, 0];
        tab.E = tab.b2-tab.b1;
        tab.s = length(tab.E);
        tab.pow = 1/5;
        tab.B = ...
            [1,   -183/64,    37/12,      -145/128
                  0,   0,             0,             0
                  0,   1500/371,   -1000/159,  1000/371
                  0,   -125/32,    125/12,     -375/64
                  0,   9477/3392,  -729/106,   25515/6784
                  0,   -11/7,      11/3,       -55/28
                  0,   3/2,        -4,         5/2];

    case 'F87'
        F87
    otherwise
        disp('Neznama metoda')
end
end
```

F87.m

```
tab.c = [ 0, 1/18, 1/12, 1/8, 5/16, 3/8, 59/400, 93/200, 5490023248/9719169821,
   13/20, 1201146811/1299019798, 1, 1]';
tab.a = { []; 1/18;
          [1/48, 1/16];
          [1/32, 0, 3/32];
          [5/16, 0, -75/64, 75/64];
          [3/80, 0, 0, 3/16, 3/20];
          [29443841/614563906, 0, 0, 77736538/692538347, -28693883/1125000000,
              23124283/1800000000];
          [16016141/946692911, 0, 0, 61564180/158732637, 22789713/633445777,
              545815736/2771057229, -180193667/1043307555];
          [39632708/573591083, 0, 0, -433636366/683701615,
              -421739975/2616292301, 100302831/723423059, 790204164/839813087,
              800635310/3783071287];
```

```
          [246121993/1340847787, 0, 0, -37695042795/15268766246,
              -309121744/1061227803, -12992083/490766935, 6005943493/2108947869,
              393006217/1396673457, 123872331/1001029789];
          [-1028468189/846180014, 0, 0, 8478235783/508512852,
              1311729495/1432422823, -10304129995/1701304382,
              -48777925059/3047939560, 15336726248/1032824649,
              -45442868181/3398467696, 3065993473/597172653];
          [185892177/718116043, 0, 0, -3185094517/667107341,
              -477755414/1098053517, -703635378/230739211, 5731566787/1027545527,
               5232866602/850066563, -4093664535/808688257,
              3962137247/1805957418, 65686358/487910083];
          [403863854/491063109, 0, 0, -5068492393/434740067,
              -411421997/543043805, 652783627/914296604, 11173962825/925320556,
              -13158990841/6184727034, 3936647629/1978049680,
              -160528059/685178525, 248638103/1413531060, 0];
          [ 13451932/455176623, 0, 0, 0, 0, -808719846/976000145,
              1757004468/5645159321, 656045339/265891186,
              -3867574721/1518517206,   465885868/322736535,  53011238/667516719,
               2/45, 0]
        };
tab.b1 = [ 14005451/335480064, 0, 0, 0, 0, -59238493/1068277825,
    181606767/758867731,   561292985/797845732,   -1041891430/1371343529,
    760417239/1151165299, 118820643/751138087, -528747749/2220607170,  1/4];
tab.b2 = [ 13451932/455176623, 0, 0, 0, 0, -808719846/976000145,
    1757004468/5645159321, 656045339/265891186,   -3867574721/1518517206,
    465885868/322736535,  53011238/667516719, 2/45,    0];
tab.E = tab.b2-tab.b1;
tab.s = length(tab.E);
tab.pow = 1/8;
```

RKStep.m

```
function yn1  = RKStep( yn, f, t, tau, but )
if isfield(but,'b2')
    yn1 = RKStep2( yn, f, t, tau, but );
    return;
end
s = length(but.b);
k = zeros(length(yn),s);
k(:,1) = f(t,yn);
for i=2:s
    y = k(:,1:i-1)*but.a{i}';
    k(:,i) = f(t+tau*but.c(i),yn+tau*y);
end
yn1 = yn + tau*k*but.b';
end
```

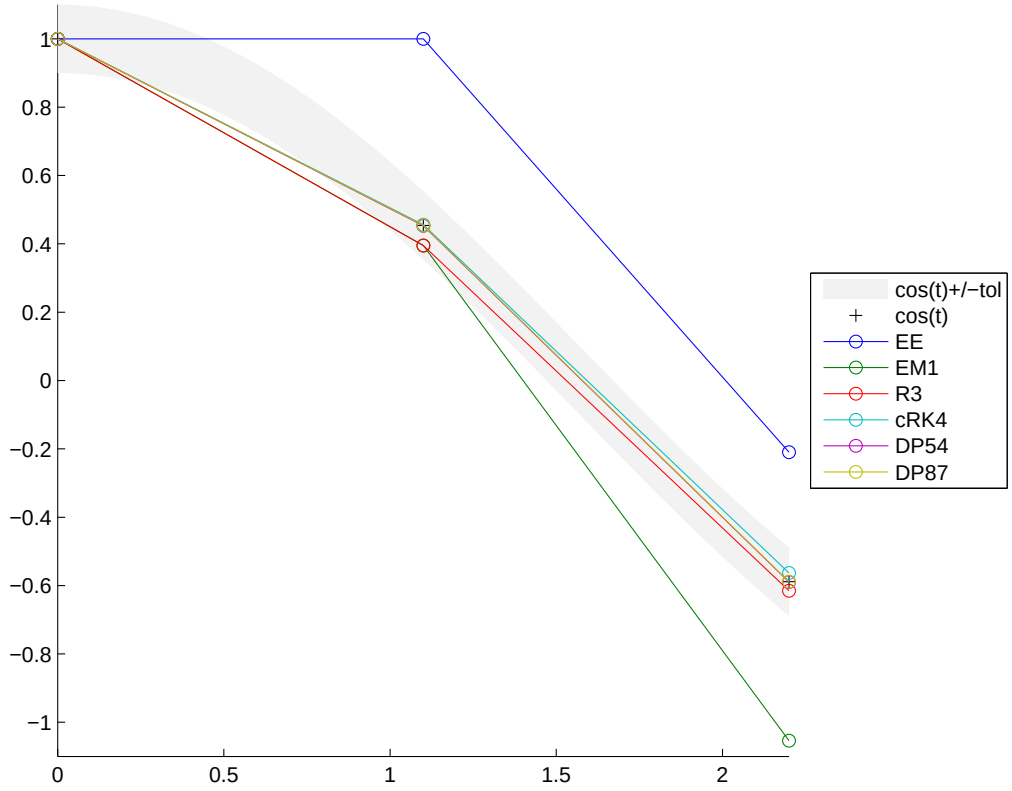
RKStep2.m

```
function [yn1, estn, k]  = RKStep2( yn, f, t, tau, but, k1 )
s = length(but.b1);
k = zeros(length(yn),s);
if ~exist('k1','var') || isempty(k1)
    k(:,1) = f(t,yn);
else
    k(:,1) = k1;
end
for i=2:s
    y = k(:,1:i-1)*but.a{i}';
    k(:,i) = f(t+tau*but.c(i),yn+tau*y);
end
```

```
yn1 = yn + tau*y;
if nargout>1
    estn = tau*k*but.E';
end
```

s10.m

```
cls;
omega = 1;
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f = @(t,y) [y(2); -omega^2*y(1)];
a=0; b=2.2; tol = 0.1;
box = [a,b,-1.1,1.1];
t = linspace(a,b,3);
eta = [ 1; 0];
Q = length(t);
y = zeros(2,Q);
y(:,1) = eta;
names = Butcher;
hold all;
tt = linspace(a,b);
rt = tt(end:-1:1);
fill([tt rt], [cos(tt)-tol cos(rt)+tol],[0.95 0.95 0.95],'EdgeColor','none');
plot(t,cos(t),'+k');
s = 0;
leg = {'cos(t)+/-tol' 'cos(t)'};
for iname=1:length(names)
    but = Butcher(names{iname});
    if but.s<=s
        continue;
    else
        s = but.s;
    end

    leg = {leg{:} names{iname}};
    for i=2:Q
        tau = t(i)-t(i-1);
        y(:,i) = RKStep(y(:,i-1),f,t(i),tau,but);
    end
    plot(t,y(1,:),'o-');
end
axis(box);
legend(leg{:},'Location','EastOutside');
print -depsc o10
```



1.6.1. Adaptivní integrace

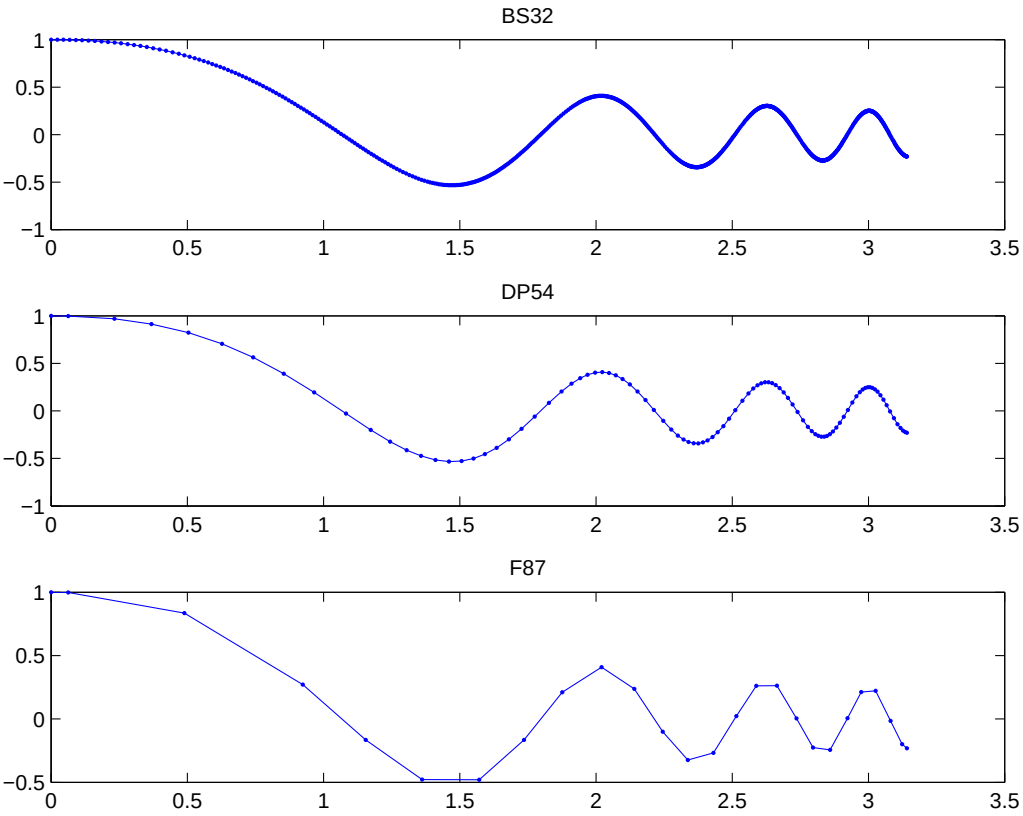
intadapt.m

```
function [tout,y] = intadapt(but,f,t,y0,opt)
if nargin == 4
    opt = odeset();
end
pow = but.pow;
% maximalni delka kroku
taumax=odeget(opt,'MaxStep');
if isempty(taumax)
    taumax = (t(2) - t(1))/2.5;
end;
% pocatecni delka kroku
tau=odeget(opt,'InitialStep');
if isempty(tau)
    tau = (t(2) - t(1))/50;
    if tau>0.1
        tau=0.1;
    end;
    if tau>taumax
        tau = taumax;
    end;
end;
%  relativni tolerance
tol=odeget(opt,'RelTol');
if isempty(tol)
    tol = 1.e-6;
end;
t0 = t(1);
tfinal = t(2);
t = t0;
% minimalni delka kroku
```

```
taumin = 16*eps;
x = y0(:);
tout = t;
y = x.';
k1 = [];
% hlavni smycka
while (t < tfinal) & (tau >= taumin)
    if (t + tau) > tfinal
        tau = tfinal - t;
    end;
    [y1, est, k] = RKStep2(x,f,t,tau,but,k1);
    % odhad chyby
    Error_step = norm(est,'inf');
    Error_tol = tol*max(norm(y1,'inf'),1.0);
    % akcepujeme reseni
    if Error_step <= Error_tol
        t = t + tau;
        x = y1;
        tout = [tout; t];
        y = [y; x.'];
        k1 = k(:,end);
    end;
    tau = min(taumax, 0.9*tau*(Error_tol/Error_step)^pow);
end;
if (t < tfinal)
    disp('Selhani ...')
end;
```

s14.m

```
cls;
omega = @(t) exp(t);
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f = @(t,y) [y(2); -omega(t)^2*y(1)];
a=0; b=pi;
box = [a,b,-1.1,1.1];
eta = [ 1; 0];
i=1;
for name={'BS32','DP54','F87'}
    but = Butcher(name{1});
    [t,y] = intadapt(but,f,[a b],eta);
    subplot(3,1,i);
    plot(t,y(:,1),'.-');
    title(name{1});
    i = i+1;
end
print -depsc o14
```



1.6.2. Hustý výstup

RKDenseStep.m

```
function ydense  = RKDenseStep( yn, t, tau, timespan, B, k )
po = size(B,2);
tt = timespan(t<timespan & timespan<=t+tau);
q = (tt-t)/tau;
lq = length(q);
Q = zeros(po,lq);
Q(1,:) = q;
for i=2:po
    Q(i,:) = q.*Q(i-1,:);
end
y = repmat(yn,1,lq);
ydense = y+tau*k*B*Q;
end
```

rk45.m

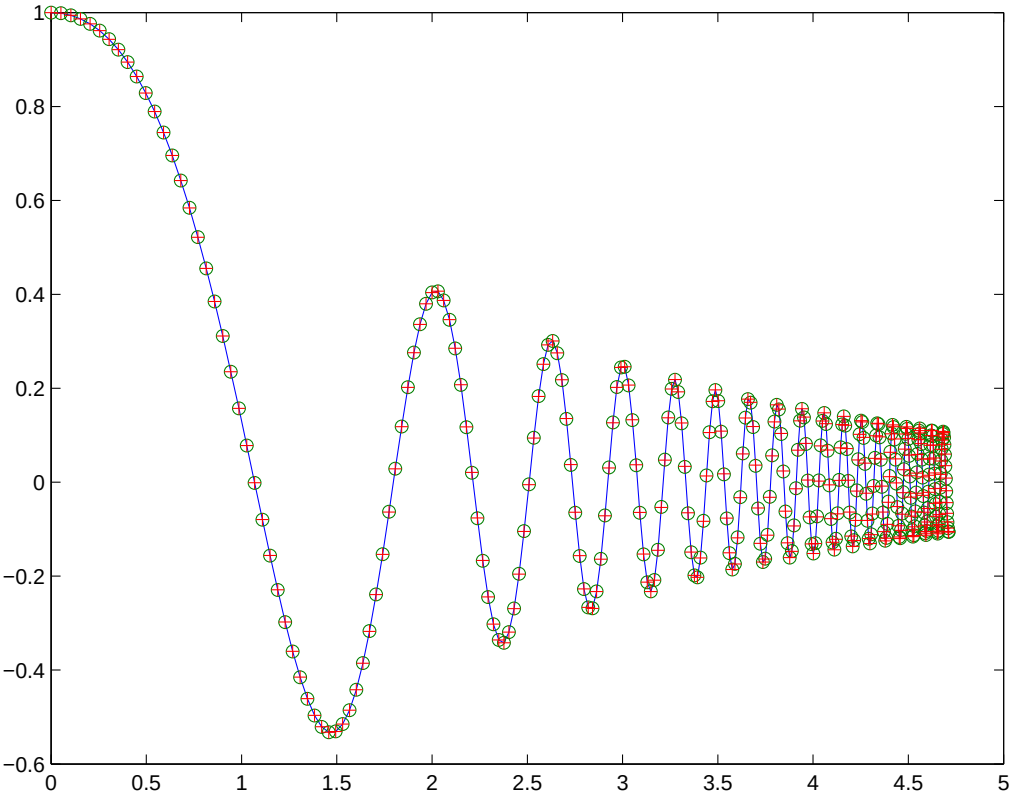
```
function [tout,y] = rk45(f,tspan,y0,opt)
dense = length(tspan)>2;
but = Butcher('DP54');
t = [tspan(1) tspan(end)];
if nargin == 3
    opt = odeset();
end
pow = but.pow;
% maximalni delka kroku
taumax=odeget(opt,'MaxStep');
if isempty(taumax)
    taumax = (t(2) - t(1))/2.5;
end;
```

```
% pocatecni delka kroku
tau=odeget(opt,'InitialStep');
if isempty(tau)
    tau = (t(2) - t(1))/50;
    if tau>0.1
        tau=0.1;
    end;
    if tau>taumax
        tau = taumax;
    end;
end;
%  relativni tolerance
tol=odeget(opt,'RelTol');
if isempty(tol)
    tol = 1.e-6;
end;
t0 = t(1);
tfinal = t(2);
t = t0;
% minimalni delka kroku
taumin = 16*eps;
x = y0(:);
tout = t;
y = x.';
k1 = [];
% hlavni smycka
while (t < tfinal) & (tau >= taumin)
    if (t + tau) > tfinal
        tau = tfinal - t;
    end;
    [y1, est, k] = RKStep2(x,f,t,tau,but,k1);
    % odhad chyby
    Error_step = norm(est,'inf');
    Error_tol = tol*max(norm(y1,'inf'),1.0);
    % akcepujeme reseni
    if Error_step <= Error_tol
        if dense
            y  = [y; RKDenseStep( x, t, tau, tspan, but.B, k )'];
        else
            y = [y; y1.'];
        end
        t = t + tau;
        tout = [tout; t];
        x = y1;
        k1 = k(:,end);
    end;
    tau = min(taumax, 0.9*tau*(Error_tol/Error_step)^pow);
end;
if dense
   tout = tspan;
end
if (t < tfinal)
    disp('Selhani ...')
end;
```

s15.m

```
cls;
omega = @(t) exp(t);
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
```

```
f = @(t,y) [y(2); -omega(t)^2*y(1)];

S=dsolve('D2y+exp(t)^2*y','y(0)=1,Dy(0)=0','t');
sol=matlabFunction(S);

a=0; b=3*pi/2;
box = [a,b,-1.1,1.1];
eta = [ 1; 0];

tspan = b*(1-exp(-linspace(0,3.2,301)))/(1-exp(-3.2));

[t,y] = rk45(f, tspan, eta);
[t1,y1] = ode45(f, tspan, eta);

yexact = sol(tspan);

plot(tspan,y(:,1),tspan,y1(:,1),'o',tspan,yexact,'+');

print -depsc o15
```



1.7. Adamsovy metody

Adams.m

```
function [B, M, c] = Adams( k )
g = zeros(1,k+1); g(1)=1;
cg = ones(k+1,1);
bd = [1 -1];
B = cell(k+1,1);
B{1} = 1;
for i=1:k
    cg(i) = 1/(i+1);
    g(i+1) = 1 - g(i:-1:1)*cg(1:i);
```

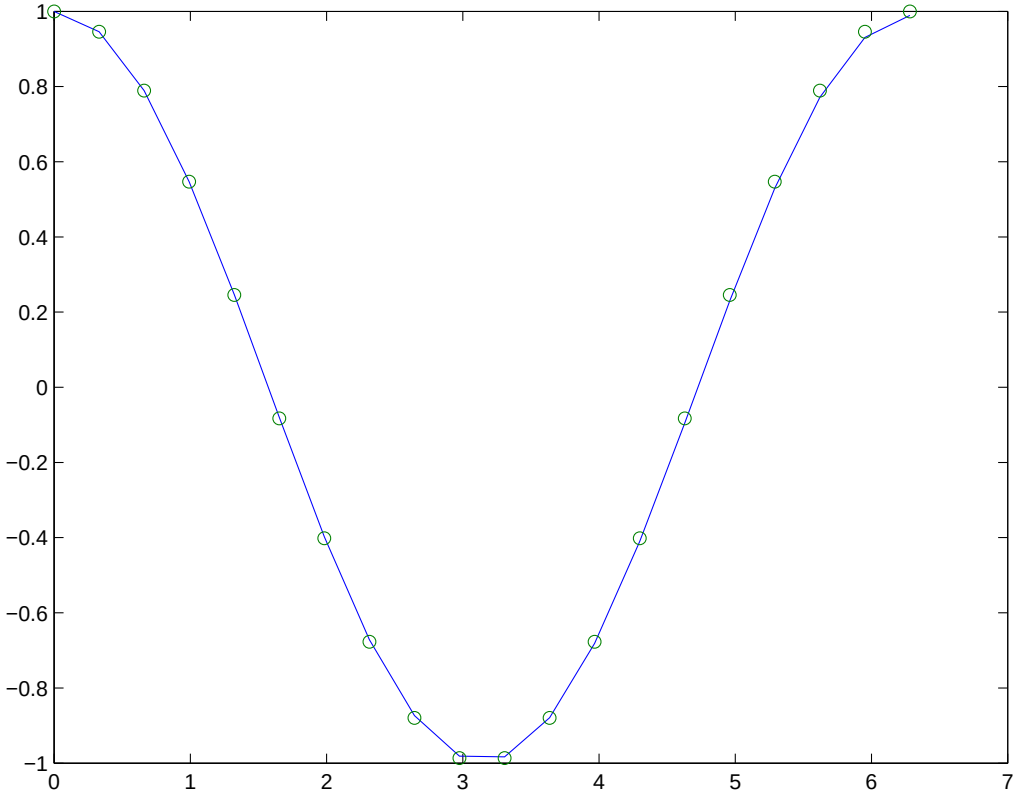
```
    B{i+1}=[B{i} 0] + g(i+1)*bd;
    bd = [bd 0] - [0 bd];
end
M = cell(k,1);
bd = [1 -1];
M{1} = 1;
gs = g;
c = zeros(1,k);
for i=1:k
    g(i+1) = gs(i+1) - gs(i);
    M{i+1}=[M{i} 0] + g(i+1)*bd;
    bd = [bd 0] - [0 bd];
    tmp = -abs(M{i+1}(end));
    c(i) = tmp/(abs(B{i+1}(end))-tmp);
end
B(end) = [];
M(end) = [];
end
```

ABStep.m

```
function [yn1, fi] = ABStep( yn, f, fi, t, tau, b )
yn1 = yn + tau*fi*b';
fi(:,2:end) = fi(:,1:end-1);
fi(:,1) = f(t+tau,yn1);
end
```

s11.m

```
cls;
omega = 1;
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f = @(t,y) [y(2); -omega^2*y(1)];
a=0; b=2*pi;
box = [a,b,-1.1,1.1];
t = linspace(a,b,20);
eta = [ 1; 0];
Q = length(t);
y = zeros(2,Q);
y(:,1) = eta;
but = Butcher('cRK4');
fi = zeros(length(eta),4);
fi(:,end) = f(a,eta);
for i=2:4
    tau = t(i)-t(i-1);
    y(:,i) = RKStep(y(:,i-1),f,t(i),tau,but);
    fi(:,end-i+1) = f(t(i+1),y(:,i));
end
[B,~] = Adams(4);
beta = B{end};
for i=5:Q
    [y(:,i), fi] = ABStep(y(:,i-1),f,fi,t(i),tau,beta);
end
plot(t,y(1,:),t,cos(t),'o');
print -depsc o11
```



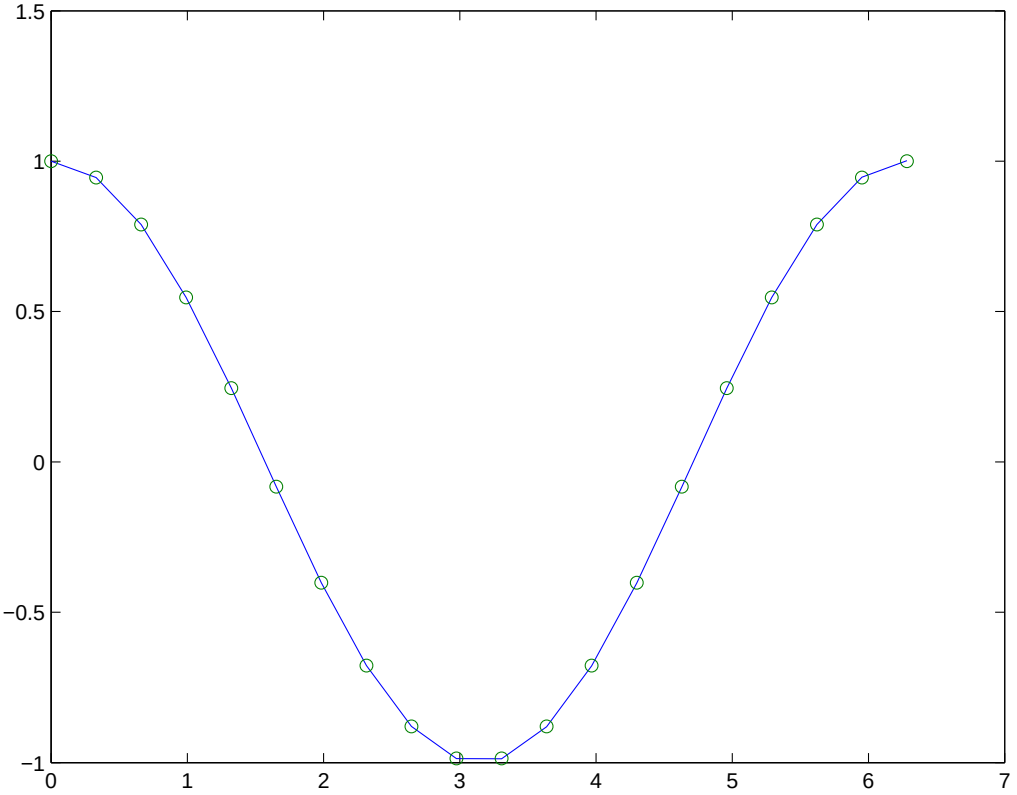
PCStep.m

```
function [yn1, fi, estn] = PCStep( yn, f, fi, t, tau, b, bm, c )
yn1 = yn + tau*fi*b';
fi(:,2:end) = fi(:,1:end-1);
fi(:,1) = f(t+tau,yn1);
yn1c = yn + tau*fi*bm';
estn = c*(yn1c-yn1);
yn1 = yn1c+estn;
fi(:,1) = f(t+tau,yn1);
end
```

s12.m

```
cls;
omega = 1;
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f = @(t,y) [y(2); -omega^2*y(1)];
a=0; b=2*pi;
box = [a,b,-1.1,1.1];
t = linspace(a,b,20);
eta = [ 1; 0];
Q = length(t);
y = zeros(2,Q);
y(:,1) = eta;
but = Butcher('cRK4');
fi = zeros(length(eta),4);
fi(:,end) = f(a,eta);
for i=2:4
    tau = t(i)-t(i-1);
    y(:,i) = RKStep(y(:,i-1),f,t(i),tau,but);
    fi(:,end-i+1) = f(t(i+1),y(:,i));
end
[B,M,c] = Adams(4);
b = B{end};
```

```
bm = M{end};
c = c(end);
for i=5:Q
    [y(:,i), fi] = PCStep(y(:,i-1),f,fi,t(i),tau,b,bm,c);
end
plot(t,y(1,:),t,cos(t),'o');
print -depsc o12
```



1.8. Metody zpětného derivování

BDF.m

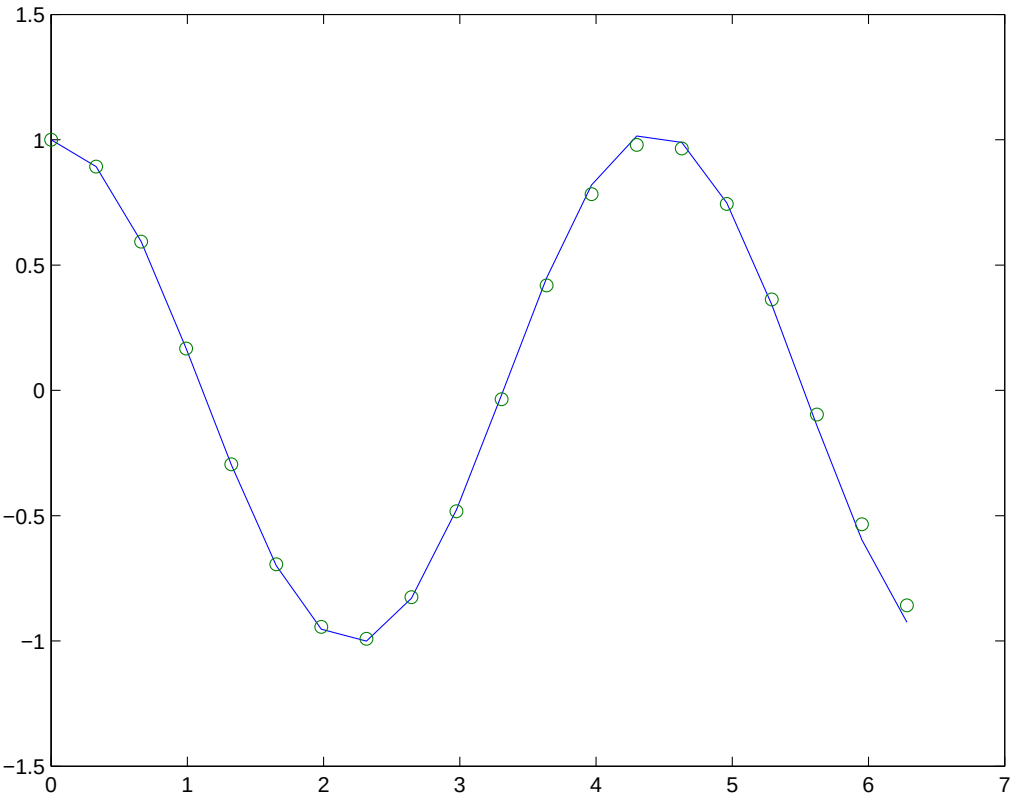
```
function a = BDF( k )
a = cell(1,k);
d = [1 -1];
a{1} = d;
for i=2:k
    d = [d 0]-[0 d];
    a{i} = [a{i-1} 0] + d/i;
end
end
```

BDFStep.m

```
function [y1] = BDFStep( y, f, i, t, tau, a)
k = length(a)-1;
n = size(y,1);
tol = ones(n,1)*1e-6;
f = @(z) [z y(:,i-1:-1:i-k)]*a'-tau*f(t,z);
y1 = nsoli(y(:,i-1),f,tol);
end
```

s13.m

```
cls;
omega = sqrt(2);
% y1' = f1(y1,y2)
% y2' = f2(y1,y2)
f = @(t,y) [y(2); -omega^2*y(1)];
a=0; b=2*pi;
box = [a,b,-1.1,1.1];
t = linspace(a,b,20);
eta = [ 1; 0];
Q = length(t);
y = zeros(2,Q);
y(:,1) = eta;
but = Butcher('cRK4');
fi = zeros(length(eta),4);
fi(:,end) = f(a,eta);
for i=2:4
    tau = t(i)-t(i-1);
    y(:,i) = RKStep(y(:,i-1),f,t(i),tau,but);
end
a = BDF(4); a = a{end};
for i=5:Q
    y(:,i) = BDFStep(y,f,i,t(i),tau,a);
end
plot(t,y(1,:),t,cos(omega*t),'o');
print -depsc o13
```



1.9. Implementace v matlabu a příklady

1.9.1. Keplerův problém a problém více těles

Uvažujme systém dvou těles, jejichž polohové vektory jsou $\mathbf{r}_1$ a $\mathbf{r}_2$, hmotnosti m_1 a m_2. Potenciální energie U je dána předpisem

$$\begin{aligned} U &= -\gamma \frac{m_1 m_2}{r_{12}}, \\ r_{12} &= |\mathbf{r}_1 - \mathbf{r}_2| . \end{aligned}$$

Pohybové rovnice jsou tedy

$$\begin{aligned} m_1 \ddot{\mathbf{r}}_1 &= -\nabla_{\mathbf{r}_1} U, \\ m_2 \ddot{\mathbf{r}}_2 &= -\nabla_{\mathbf{r}_2} U. \end{aligned}$$

Označíme si

$$\begin{aligned} \mathbf{r}_1 &= [x_{1,1}, x_{1,2}, x_{1,3}] \\ \mathbf{r}_2 &= [x_{2,1}, x_{2,2}, x_{2,3}] \end{aligned}$$

takže

$$r_{12} = \sqrt{\sum_{i=1}^{3} (x_{1,i} - x_{2,i})^2}$$

a

$$-\nabla_{\mathbf{r}_i} U = \gamma \frac{m_1 m_2}{r_{12}^3} (\mathbf{r}_j - \mathbf{r}_i) , \; i, j = 1, 2 \; i \neq j$$

a pohybové rovnice jsou druhého řádu

$$\begin{aligned} \ddot{\mathbf{r}}_1 &= \gamma \frac{m_2}{r_{12}^3} (\mathbf{r}_2 - \mathbf{r}_1) , \\ \ddot{\mathbf{r}}_2 &= \gamma \frac{m_1}{r_{12}^3} (\mathbf{r}_1 - \mathbf{r}_2) , \end{aligned}$$

převedeme na soustavu 1. řádu

$$\begin{aligned} \dot{\mathbf{r}}_1 &= \mathbf{v}_1, \\ \dot{\mathbf{v}}_1 &= \gamma \frac{m_2}{r_{12}^3} (\mathbf{r}_2 - \mathbf{r}_1) , \\ \dot{\mathbf{r}}_2 &= \mathbf{v}_2, \\ \dot{\mathbf{v}}_2 &= \gamma \frac{m_1}{r_{12}^3} (\mathbf{r}_1 - \mathbf{r}_2) , \end{aligned}$$

což je soustava pro čtyři vektorové funkce, čili pro 12 skalárů celkem, které v matlabovském kódu budeme reprezentovat vektorem $y = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{v}_1, \mathbf{v}_2]^T$.

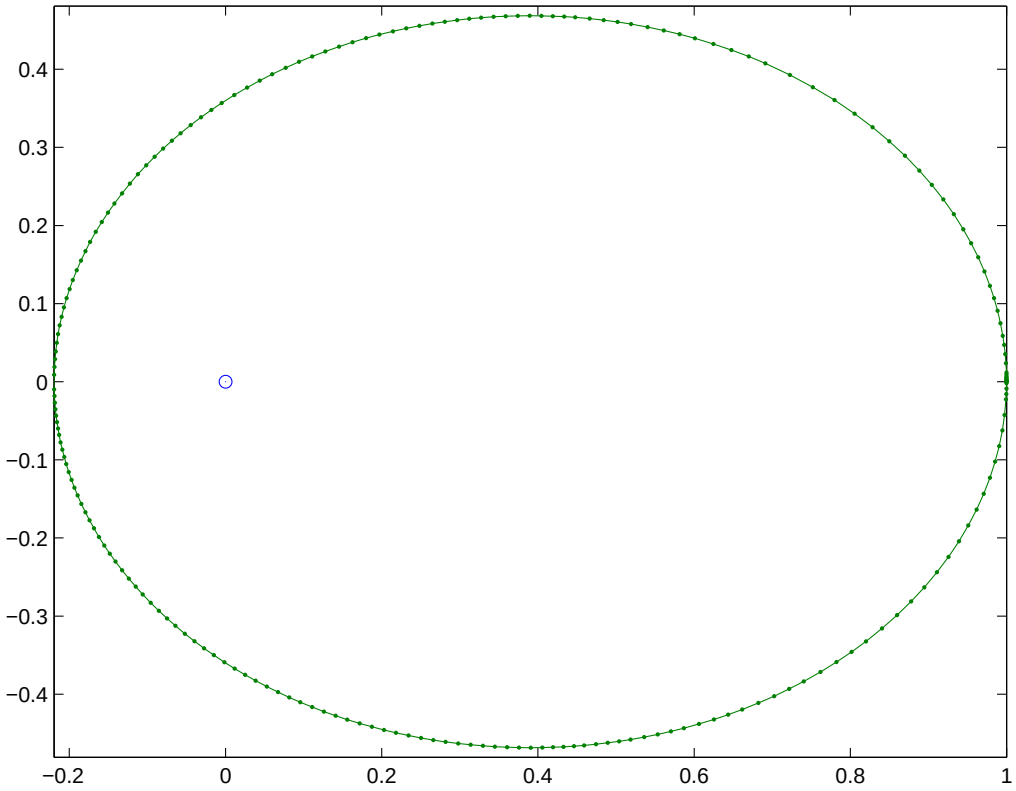
Kepler.m

```
function dy = Kepler( gamma, m1, m2, y )
r1 = y(1:3);
r2 = y(4:6);
v1 = y(7:9);
v2 = y(10:12);
r3 = 1/norm(r1-r2)^3;
dy = [ v1
```

```
        v2
        gamma*m2*r3*(r2-r1)
        gamma*m1*r3*(r1-r2)
      ];
end
```

s16.m

```
cls;
m1 = 1;
m2 = 0.000001;
gamma = 1;
r1 = [0; 0; 0];
r2 = [1; 0; 0];
v1 = [0; 0; 0];
v2 = [0; 0.6; 0];
tspan = [0 3];
eta = [r1; r2; v1; v2];
f = @(t,y) Kepler(gamma,m1,m2,y);
opt = odeset('AbsTol',1e-10,'RelTol',1e-6);
[t,y] = ode45(f,tspan,eta,opt);
%[t,y] = ode113(f,tspan,eta,opt);
x = { y(:,1) y(:,2)
      y(:,1+3) y(:,2+3) };
plot(x{1,1},x{1,2},'o-',x{2,1},x{2,2},'.-');
%plot(y(:,[1 4])+y(:,[2 5])*1i);
axis equal
print -depsc o16
```



Počáteční úlohu můžeme snadno zobecnit na iterakci N těles. Sestavíme výraz pro potenciální energii

$$\begin{aligned} U &= -\gamma \sum_{\substack{i,j=1 \\ i<j}}^{N} \frac{m_i m_j}{r_{ij}}, \\ r_{ij} &= |\mathbf{r}_i - \mathbf{r}_j|, \end{aligned}$$

a pohybové rovnice

$$\ddot{\mathbf{r}}_i = \gamma \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{m_j}{r_{ij}^3} (\mathbf{r}_j - \mathbf{r}_i).$$

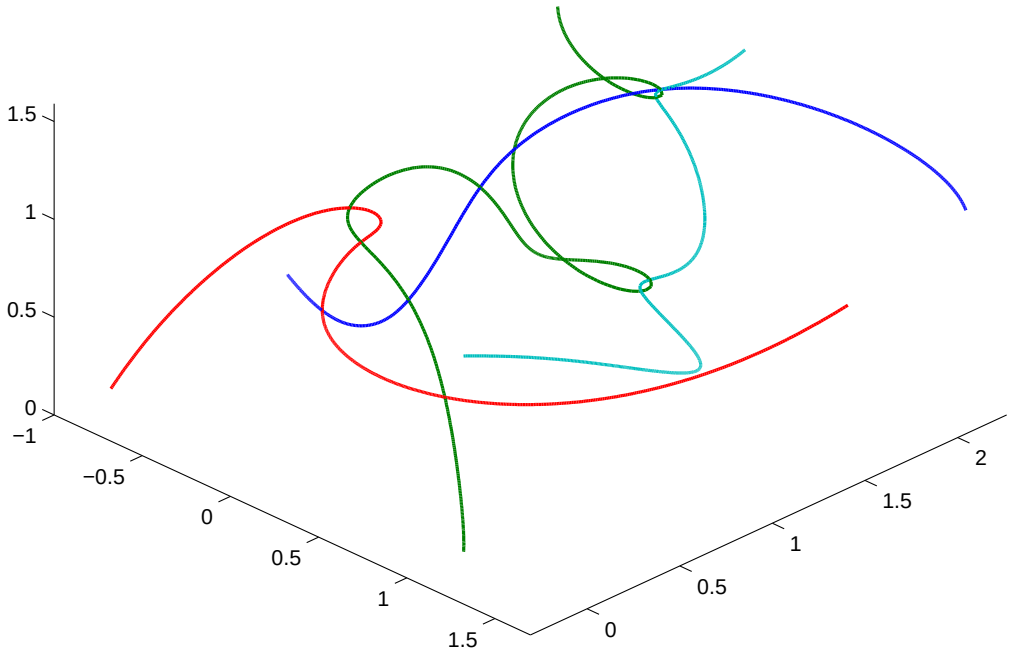
NBody.m

```
function dy = NBody( gamma, m, y )
N = length(m);
r = zeros(3,N);
v = zeros(3,N);
for i=0:N-1
    r(:,i+1) = y(3*i+1:3*i+3);
    v(:,i+1) = y(3*(N+i)+1:3*(N+i)+3);
end
r3 = zeros(N);
d = cell(1,3);
for i=1:3
    [X, Y] = meshgrid(r(i,:),r(i,:));
    d = X-Y; d2 = d.*d;
    r3 = r3 + d2;
end
r3 = sqrt(r3).^(-3);
r3(r3==Inf)=0;
dy = zeros(6*N,1);
dy(1:3*N,:) = v(:);
ix = 3*N+1;
for i=1:N
    dy(ix:ix+2) = gamma*m(i)*(r-repmat(r(:,i),1,N))*r3(:,i);
    ix = ix+3;
end
end
```

s17.m

```
cls;
m = [1,2,2,1];
gamma = 1;
r{1} = [0; 0; 1];
r{2} = [1; 0; 0];
r{3} = [-1; 0; 0];
r{4} = [1; 0; 1];
v{1} = [0; 0; 0];
v{2} = [-1; 1; 0];
v{3} = [0; 1; 1];
v{4} = [1; 1; 0];
N = length(m);
tspan = linspace(0,3,600);
eta = [];
```

```
for i=1:N
    eta = [eta; r{i}];
end
for i=1:N
    eta = [eta; v{i}];
end
f = @(t,y) NBody(gamma,m,y);
opt = odeset('AbsTol',1e-10,'RelTol',1e-6);
[t,y] = ode45(f,tspan,eta,opt);
x = cell(1,3);
hold all;
for i=0:N-1
    for j=1:3
        x{j} = y(:,3*i+j);
    end
    plot3(x{1},x{2},x{3},'LineWidth',1.5);
end
axis tight
view(45,45);

print -depsc o17
```



2. Okrajové problémy pro obyčejné diferenciální rovnice

2.1. Základní pojmy

Budeme se zabývat lineární diferenciální rovnicí 2. řádu

$$-\left[p(x)u' - r(x)\right]' + q(x)u = f(x), \qquad x \in (0, l)$$

s okrajovými podmínkami Dirichletova typu

$$\begin{aligned} u\left(0\right) &= g_0, \\ u\left(l\right) &= g_l, \end{aligned}$$

rsp. s kombinací Dirichletovy a Robinovy podmínky

$$\begin{aligned} u\left(0\right) &= g_0, \\ -p\left(l\right)u'\left(l\right) &= \alpha u\left(l\right) - \beta_l \end{aligned}$$

nebo Robinovy a Dirichletovy podmínky

$$\begin{aligned} p\left(0\right)u'\left(0\right) &= \alpha u\left(0\right) - \beta_0, \\ u\left(l\right) &= g_l. \end{aligned}$$

2.1.1. Metoda střelby

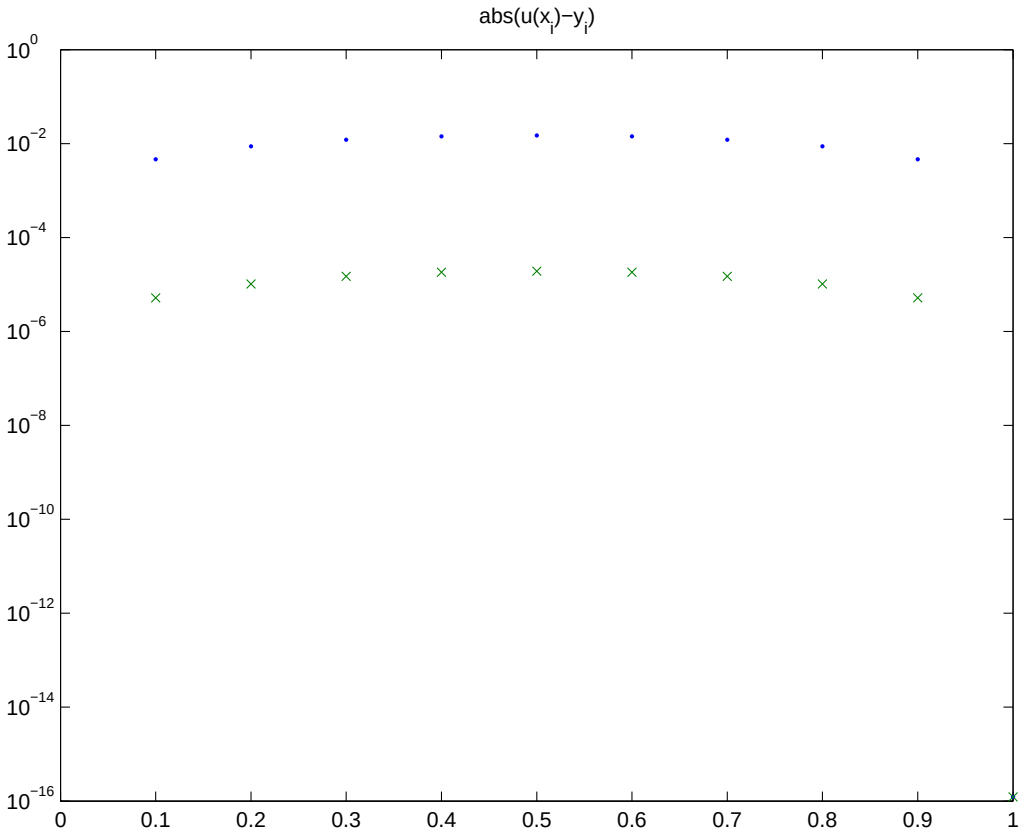
Lichoběžníková metoda a její porovnání s matlabovskou funkcí bvp4c.

exbvp4c.m

```
function y = exbvp4c(n)
solinit = bvpinit(linspace(0,1,n),[0 0]);
sol = bvp4c(@myode,@mybc,solinit);
y = sol.y(1,:);
end

function dydx = myode(x,y)
  A = [0 1; 1 0];
  q = [0; -f(x)];
  dydx = A*y+q;
end

function res = mybc(ya,yb)
  res = [ ya(1)
          yb(1)];
end

function y = f(x)
   y = sin(pi*x) + pi^2*sin(pi*x);
end
```

s18.m

```
addpath ..
cls


u = @(x) sin(pi*x);

syms x
f = -diff(u(x),x,2)+u(x);
sf = char(f)
clear x

f = @(x) eval(sf);

A = [0 1; 1 0];
q = @(x) [ 0 -f(x)];
n = 11; N = 2*n;
x = linspace(0,1,n);
h = x(2)-x(1);
spA = sparse(N,N);
v = zeros(N,1);
I = eye(2);
R = -I-h/2*A;
S = I-h/2*A;
Ba = [1 0; 0 0];
Bb = [0 0; 1 0];
for i=1:n-1
    r = 2*i-1:2*i;
    spA(r,r)=R;
    spA(r,r+2)=S;
    v(r) = h/2*(q(x(i))+q(x(i+1)));
end
spA(N-1:N,1:2)=Ba;
spA(N-1:N,N-1:N)=Bb;

y = spA\v; y = y';
y1 = exbvp4c(n);
semilogy(x,abs(u(x)-y(1:2:end)),'.',x,abs(u(x)-y1),'x');
title('abs(u(x_i)-y_i)');

print -depsc o18
```



2.1.2. Diferenční metoda

defbvp.m

```
function bvp = defbvp
bvp.p = @(x) x.^0;
bvp.q = @(x) x.^0;
bvp.r = @(x) 0*x;
bvp.f = @(x) x.^0;
bvp.domain = [0 1];
bvp.bc = {[0 0] 0};
end
```

exsol1d.m

```
function eq = exsol1d( eq, u)
syms x;
sdu = diff(u(x),x);
du = matlabFunction(sdu,'vars',x);
f = expand(-diff(eq.p(x)*sdu-eq.r(x)*u(x),x) + eq.q(x)*u(x));
eq.f = matlabFunction(f);
for i=1:2
    vx = eq.domain(i);
    vu = u(vx);
    if length(eq.bc{i})==1
        eq.bc{i} = vu;
    else
        a = eq.bc{i}(1);
        eq.bc{i}(2)=a*vu+(-1)^i*eq.p(vx)*du(vx);
    end
end
end
```

diff1d.m

```
function u = diff1d( x, eq)
if ~exist('eq','var')
    eq = defbvp;
end
h = x(2)-x(1);
xmid = (x(2:end)+x(1:end-1))/2;
p = eq.p(xmid);
r = eq.r(xmid);
zero = 0*xmid;
rplus = max(r,zero);
rminus = min(r,zero);
q = eq.q(x);
b = eq.f(x');
dc = [length(eq.bc{1}) length(eq.bc{2})]==1;
n = length(x);
A = sparse(n,n);
for i=2:n-1
    r = i-1:i+1;
    A(i,r) = [-p(i-1) p(i-1)+p(i)+q(i)*h^2 -p(i)] + ... % (2.15)
        [-rplus(i-1) rplus(i)-rminus(i-1) rminus(i)]*h; % (2.29)
end
for i=1:2
    if i==1
        k=1; r=1:2;
    else
        k=size(b,1); r = [k k-1];
    end
    if dc(i)
        A(k,:)=[]; b(k)=[];
        b = b - A(:,k)*eq.bc{i}/h^2;
        A(:,k)=[];
    else
        A(k,r) = [p(k)+h*eq.bc{i}(1)+q(k)*h^2/2 -p(k)] + ... % (2.19a)
            [rplus(k) rminus(k)]*h ; % (2.29)
        b(k) = eq.bc{i}(2)/h + b(k)/2;
    end
end
A = A/h^2;
u = A\b;
csvwrite('d1_4r.csv',[full(A) b u]);
u = u';
if dc(1)
    u = [ eq.bc{1} u];
end
if dc(2)
    u = [ u eq.bc{2}];
end
csvwrite('d1_4ru.csv',[x; u]);

end
```

s19.m

```
addpath ..
cls

eq = defbvp;

%eq.bc{1} = [2  1];
```

```
eq.bc{1} = 0;
eq.bc{2} = 0;

eq.p = @(x) 1+x;
eq.q = @(x) 2-x;

%presne reseni
u = @(x) x-x.^3;
%u = @(x) sin(x);
%u = @(x) x;

eq = exsol1d(eq,u);

x = linspace(0,1,5);
U = diff1d(x,eq);

plot(x,abs(U-u(x)));
ylabel('abs(U-u(x))');
xlabel('x');

print -depsc o19
```

Diferenční metodou budeme řešit úlohu

$$-\left[(1+x)u'\right]' + (2-x)u = x^4 - 2x^3 + 8x^2 + 8x - 1, \qquad x \in (0,l)$$

s okrajovými podmínkami

$$\begin{aligned} u'(0) &= 2u(0) - 1, \\ u(l) &= 0. \end{aligned}$$

A				b	u
27	-18	0	0	-4.5	-0.022365
-18	41.75	-22	0	1.4727	0.21645
0	-22	49.5	-26	4.8125	0.36213
0	0	-26	57.25	8.9727	0.32119

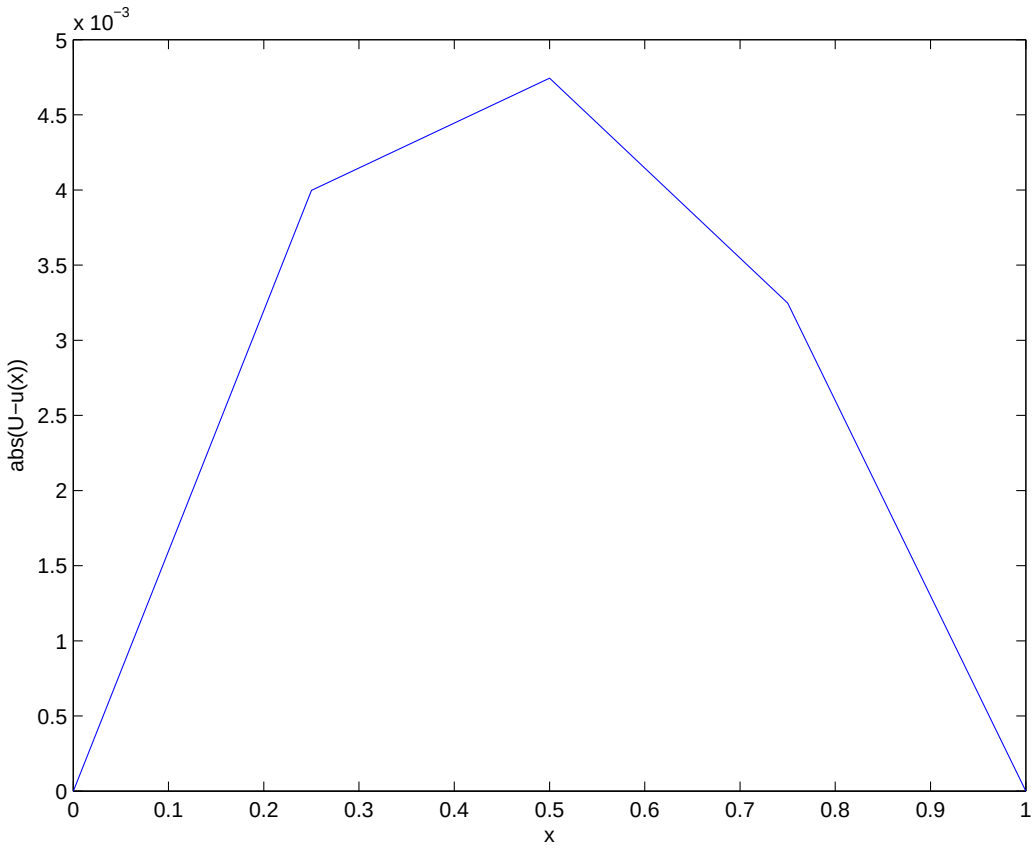
x_i	0	0.25	0.5	0.75	1
U_i	-0.022365	0.21645	0.36213	0.32119	0

a s Dirichletovými okrajovými podmínkami

$$\begin{aligned} u(0) &= 0, \\ u(l) &= 0. \end{aligned}$$

A			b	u
41.75	-22	0	1.4727	0.23038
-22	49.5	-26	4.8125	0.37026
0	-26	57.25	8.9727	0.32488

x_i	0	0.25	0.5	0.75	1
U_i	0	0.23038	0.37026	0.32488	0



2.1.3. Metoda konečných prvků

defbvpm.m

```
function eq = defbvpm
eq.p = {@(x) x.^0, @(x) 2*x.^0};
eq.q = {@(x) x.^0, @(x) x.^0};
eq.f = {@(x) 0*x, @(x) x.^0};
eq.domain = [-1 0 2];
eq.bc = {[0 0] 0};
end
```

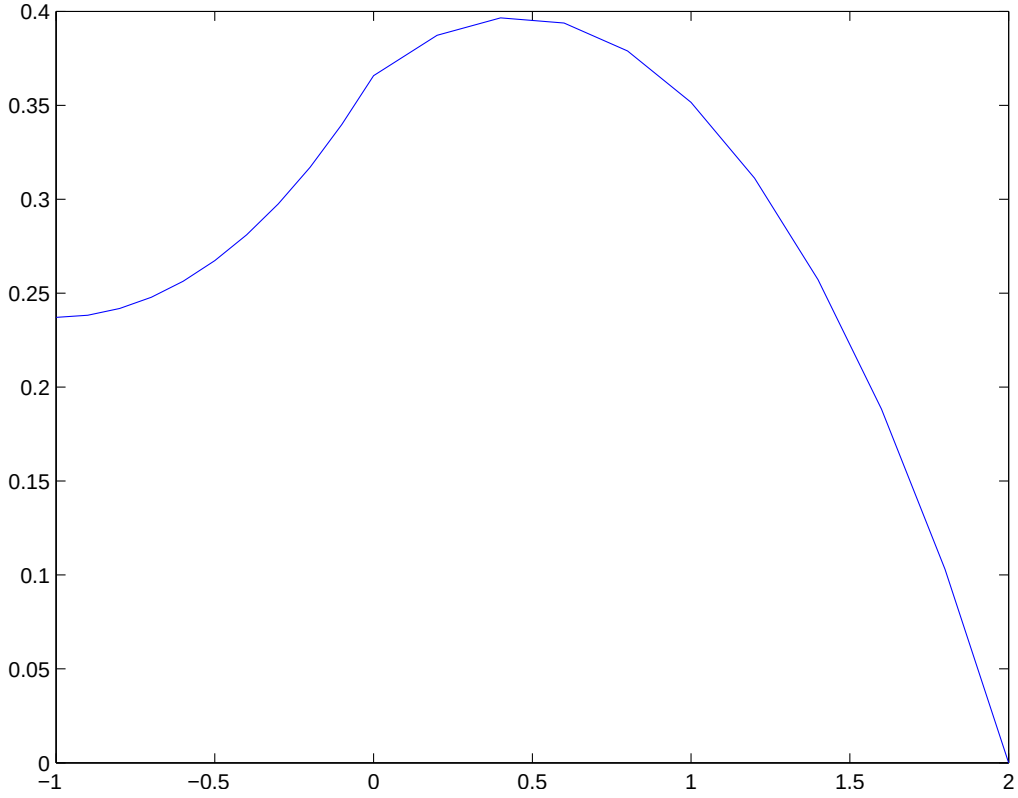
MKPd1.m

```
function [xall,y] = MKPd1( X, eq )
ndom = length(X);
dc = [length(eq.bc{1}) length(eq.bc{2})]==1;
K = []; I = []; J = []; S = []; V = [];
ie = 1;
xall = X{1}(1);
for i=1:ndom
    x = X{i};
    xall = [xall x(2:end)];
    p = eq.p{i};
    q = eq.q{i};
    f = eq.f{i};
    n = length(x);
    for k=2:n
        h = x(k)-x(k-1);
        xm = (x(k)+x(k-1))/2;
        I = [I [ie ie ie+1 ie+1]];
        J = [J [ie ie+1 ie ie+1]];
```

```
        S = [S p(xm)/h*[1 -1 -1 1]+h/2*[q(x(k-1)) 0 0 q(x(k))]];
        V = [V h/2*[f(x(k-1)); f(x(k))]];
        ie = ie+1;
    end
end
nele = size(V,2);
K = sparse(I,J,S,nele+1,nele+1);
F = zeros(nele+1,1);
for i=1:nele
    F([i i+1]) = F([i i+1]) + V(:,i);
end
for i=1:2
    if i==2
        ix = size(K,1);
    else
        ix=1;
    end
    if dc(i)
        F = F - eq.bc{i}*K(:,ix);
        F(ix)=[];
        K(ix,:)=[];
        K(:,ix)=[];
    else
        K(ix,ix) = K(ix,ix) + eq.bc{i}(1);
        F(ix) = F(ix) + eq.bc{i}(2);
    end
end
y = K\F; y = y';
if dc(1)
    y = [ eq.bc{1} y];
end
if dc(2)
    y = [ y eq.bc{2}];
end
end
```

s20.m

```
cls

eq = defbvpm;

ndom = length(eq.domain)-1;
X = cell(1,ndom);
n=10;
for i=1:ndom
    X{i} = linspace(eq.domain(i),eq.domain(i+1),n+1);
end

[x,y] = MKPd1(X,eq);
plot(x,y);

print -depsc o20
```

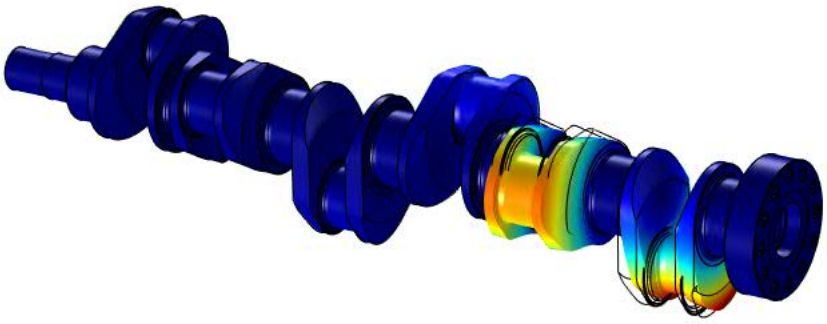

0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
−1
−0.5
0
0.5
1
1.5
2

Část II.

Parciální diferenciální rovnice

Obyčejné diferenciální rovnice modelují jednorozměrné dynamické systémy. Parciální diferenciální rovnice modelují vícerozměrné systémy, vyznačující se často komplikovanou geometrií. Umožňují tudíž modelování a simulaci složitých technických systémů. Tři nejrozšířenější numerické metody pro řešení parciálních rovnic jsou metoda konečných prvků (MKP), metoda konečných objemů a metoda konečných diferencí. MKP má významné postavení mezi těmito metodami pro svou geometrickou sílu. V jednoduché formě jsme se s nimi již seznámili v jednorozměrném případě.

V současnosti existují mnoho vyspělých komerčních programů jako jsou ANSYS, ABAQUS, COMSOL, FLUENT a mnoho dalších. Z otevřeného softwaru lze jmenovat např. FreeFEM++ `http://www.freefem.org/ff++/`, který jsem zde použil pro generování jednoduchých triangulací, dále bych doporučil ELMER `http://www.csc.fi/english/pages/elmer` a FEniCS `http://fenicsproject.org/`.Velmi úspěšný OpenFOAM `http://www.openfoam.com/` založený na metodě konečných objemů. Použití těchto systémů je však dosti složité a překračuje rámec našeho předmětu.



3. Okrajové problémy pro parciální diferenciální rovnice

3.1. Úloha eliptického typu

Předpokládejme, že Ω je omezená oblast v rovině. Hranice $\partial\Omega$ sestává z jedné nebo více disjunktních částí na nichž se předepisují podmínky Dirichletova resp. Robinova typu. Lineární eliptická rovnice 2. řádu má tvar

$$-\frac{\partial}{\partial x}\left(p\left(x,y\right)\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial y}\left(p\left(x,y\right)\frac{\partial u}{\partial y}\right)+q\left(x,y\right)=f\left(x,y\right)\qquad \text{v } \Omega.$$

Na části hranice $\emptyset \neq \Gamma_1 \subset \partial\Omega$ předepíšeme Dirichletovu podmínku

$$u=g\left(x,y\right)\qquad \text{na } \Gamma_1$$

a na části hranice $\Gamma_2 \subset \partial\Omega$ předepíšeme Robinovu podmínku

$$-p\left(x,y\right)\left[n_1\frac{\partial u}{\partial x}+n_2\frac{\partial u}{\partial y}\right]=\alpha\left(x,y\right)u-\beta\left(x,y\right)\qquad \text{na } \Gamma_2,$$

kde $[n_1, n_2]$ je jednotkový vektor vnější normály.

3.1.1. Diferenční metoda

bvprect.m

```
function eq = bvprect
eq.p = @(x,y) 1;
eq.q = @(x,y) 0;
eq.r = {@(x,y) 0, @(x,y) 0};
eq.f = @(x,y) 1;
eq.bc = {@(x,y) 0, @(x,y) 0, @(x,y) 0, @(x,y) 0};
end
```

exsol2d.m

```
function eq = exsol2d( eq, sol)
syms x y;
u = sol(x,y);
ux = diff(u,x);
uy = diff(u,y);
p = eq.p(x,y);
q = eq.q(x,y);
r1 = eq.r{1}(x,y);
r2 = eq.r{2}(x,y);
f = - diff(p*ux-r1*u,x) - diff(p*uy-r2*u,y) + eq.q*u;
eq.f = mf(f,x,y);
for i=1:4
    if length(eq.bc{i})==1
```

```
        eq.bc{i} = sol;
    else
        switch i
            case 1
                s = -1; du = uy;
            case 2
                s = 1; du = ux;
            case 3
                s = 1; du = uy;
            case 4
                s = -1; du = ux;
        end
        eq.bc{i} = { eq.bc{i}{1}, mf(eq.bc{i}{1}*u+s*p*du,x,y)};
    end
end
end

function f = mf(sf,varargin)
f = matlabFunction(simplify(sf),'vars',varargin);
end
```

diff2d.m

```
function [X,Y,Z] = diff2d( x, y, eq)
% pro rovnice s konvekcnim clenem nejsou implementovany Robinovy podminky!!
if ~exist('eq','var')
    eq = bvprect;
end
nx = length(x);
ny = length(y);
hx = x(2)-x(1);
hy = y(2)-y(1);

xmid = (x(2:end)+x(1:end-1))/2;
ymid = (y(2:end)+y(1:end-1))/2;
[Xmx,Ymx] = meshgrid(xmid,y);
Xmx = Xmx'; Ymx= Ymx';
[Xmy,Ymy] = meshgrid(x,ymid);
Xmy = Xmy'; Ymy= Ymy';
[X,Y] = meshgrid(x,y);
X = X'; Y= Y';

pmx = arrayfun(eq.p,Xmx,Ymx);
pmy = arrayfun(eq.p,Xmy,Ymy);
r1mx = arrayfun(eq.r{1},Xmx,Ymx);
r2my = arrayfun(eq.r{2},Xmy,Ymy);

zerox = 0*Xmx;
zeroy = 0*Xmy;

r1plus = max(r1mx,zerox);
r1minus = min(r1mx,zerox);
r2plus = max(r2my,zeroy);
r2minus = min(r2my,zeroy);

q = arrayfun(eq.q,X,Y);
f = arrayfun(eq.f,X,Y);

dc = cellfun(@(x)length(x)==1,eq.bc);
dc = [dc dc(1)];
```

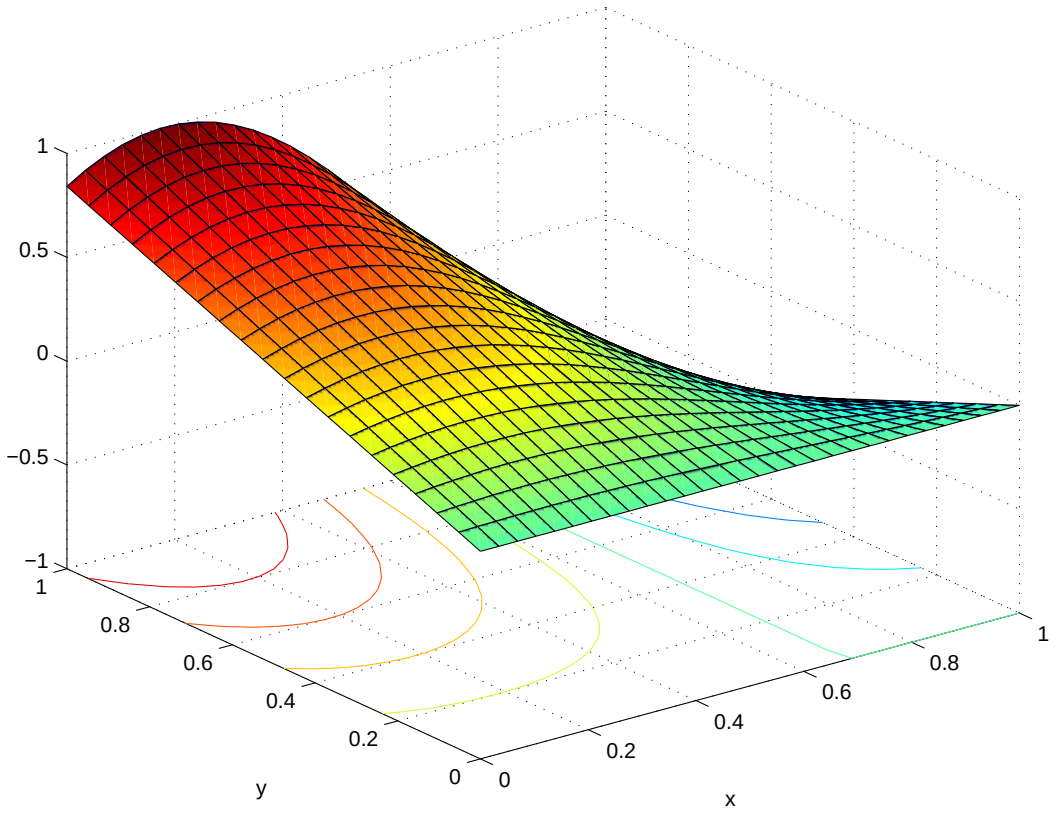
```
N = nx*ny;
ixy = reshape(1:N,nx,ny);
ix = 1;

%pocet stran s Robinovou podminkou
nrc = 4-sum(dc);
%pocet vrcholu s dvojici Robinovych podminek
nrcc = sum(~dc(1:4) & ~dc(2:5));

I = zeros(1,5*(nx-2)*(ny-2)+4*nrc+3*nrcc);
J = I; S = I;
% cyklus pres vsechny vnitrni uzly site
for i=2:nx-1
    for j=2:ny-1
        %index radku
        I(ix:ix+4) = ixy(i,j)*[1 1 1 1 1];
        %indexy sloupcu
        J(ix:ix+4) = [ixy(i-1,j) ixy(i+1,j) ixy(i,j-1) ixy(i,j+1) ixy(i,j)];
        %koeficienty
        tmp = [ [pmx(i-1,j) pmx(i,j)]/hx^2 [pmy(i,j-1) pmy(i,j)]/hy^2 ];
        rtmp = [ [-r1plus(i-1,j) r1minus(i,j)]/hx ...
                 [-r2plus(i,j-1) r2minus(i,j)]/hy ];
        S(ix:ix+3) = -tmp + rtmp;
        S(ix+4) = sum(tmp) + q(i,j) + ...
            (r1plus(i,j)-r1minus(i-1,j))/hx + ...
            (r2plus(i,j)-r2minus(i,j-1))/hy;
        ix = ix+5;
    end
end

%strany
rect = {    1:nx    nx      nx:-1:1     1
    1       1:ny    ny          ny:-1:1 };

irect = cell(1,4);
idc = []; ag = [];
g = cell(1,4);
rij = 1;
for ie=1:4
    irect{ie} = ixy(rect{1,ie},rect{2,ie});
    ex = X(rect{1,ie},rect{2,ie});
    ey = Y(rect{1,ie},rect{2,ie});
    last = length(ex);
    if dc(ie)
        if dc(ie+1)
            r = 1:last-1;
        else
            r = 1:last;
        end
        tmp = irect{ie}(r);
        idc = [idc tmp(:)'];
        g{ie} = arrayfun(eq.bc{ie},ex,ey);
        tmp = g{ie}(r);
        ag = [ag tmp(:)'];
    else %robinova podminka
        alfa = arrayfun(eq.bc{ie}{1},ex,ey);
        beta = arrayfun(eq.bc{ie}{2},ex,ey);
        ir = inner(rect{1,ie});
        jr = inner(rect{2,ie});
        for i=ir
```

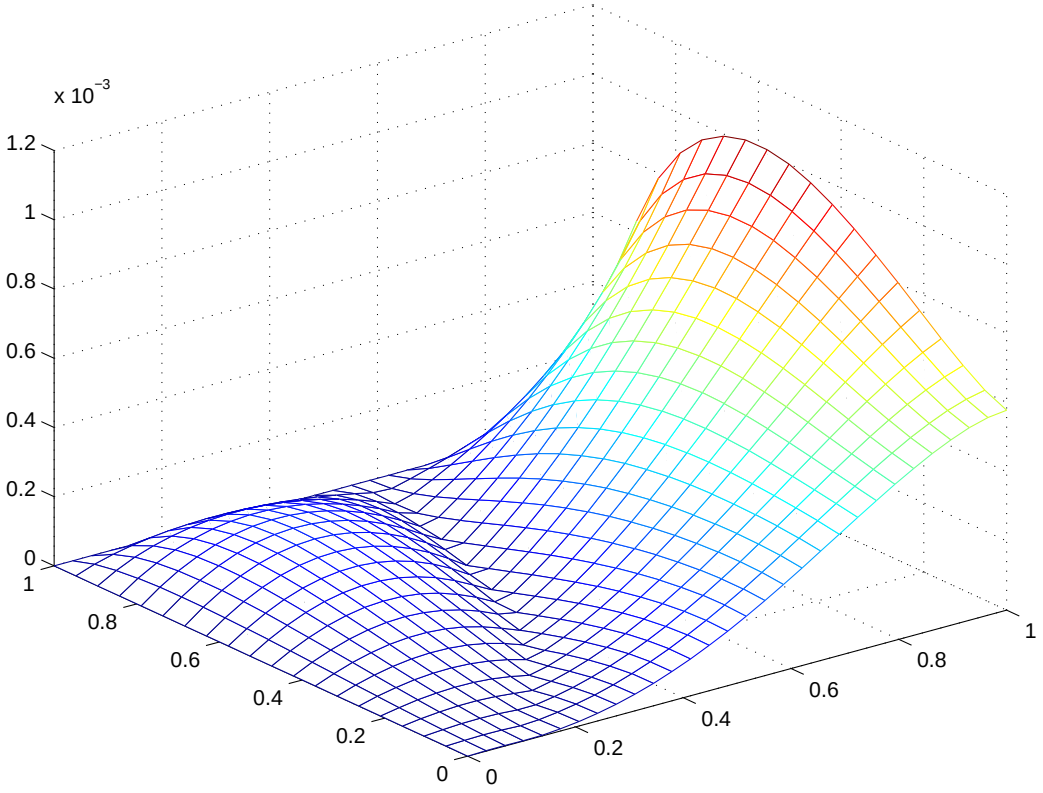
```
            for j=jr
                ij = [ i j];
                I(ix:ix+3) = ixy(i,j)*[1 1 1 1];
                [si, psi] = stencil(i,nx);
                [sj, psj] = stencil(j,ny);
                J(ix:ix+3) = [ixy(si,j)' ixy(i,sj) ixy(i,j)];
                if length(si)==2
                    tmp = pmx(psi,j)'/(2*hx^2);
                else
                    tmp = pmx(psi,j)'/(hx^2);
                    h = hx;
                end
                if length(sj)==2
                    tmp = [tmp pmy(i,psj)/(2*hy^2)];
                else
                    tmp = [tmp pmy(i,psj)/(hy^2)];
                    h = hy;
                end
                S(ix:ix+2) = -tmp;
                S(ix+3) = sum(tmp) + alfa(ij(rij))/h + q(i,j)/2;
                ix = ix+4;
                f(i,j) = f(i,j)/2 + beta(ij(rij))/h;
            end
        end
        %roh
        h = [hy hx hy hx];
        if ~dc(ie+1)
            ij(rij) = ij(rij)+1;
            i = ij(1); j=ij(2);
            I(ix:ix+2) = ixy(i,j)*[1 1 1];
            [si, psi] = stencil(i,nx);
            [sj, psj] = stencil(j,ny);
            J(ix:ix+2) = [ixy(si,j)' ixy(i,sj) ixy(i,j)];
            tmp = [pmx(psi,j)'/(2*hx^2) pmy(i,psj)/(2*hy^2)];
            S(ix:ix+1) = -tmp;
            S(ix+2) = sum(tmp)+ ...
                  eq.bc{ie}{1}(X(i,j),Y(i,j)) / (2*h(ie)) + ...
                  eq.bc{ie+1}{1}(X(i,j),Y(i,j)) / (2*h(ie+1)) + q(i,j)/4;
            ix = ix+3;
            f(i,j) = f(i,j)/4 + ...
                  eq.bc{ie}{2}(X(i,j),Y(i,j)) / (2*h(ie)) + ...
                  eq.bc{ie+1}{2}(X(i,j),Y(i,j)) / (2*h(ie+1));
        end
    end
    if rij==1
        rij = 2;
    else
        rij = 1;
    end
end

K = sparse(I,J,S,N,N);
K(idc,:)=[];
F = reshape(f,1,N)';
F(idc) =[];
F = F - K(:,idc)*ag';
K(:,idc)=[];
u = K\F;
Z = zeros(size(X));
```

```
dofx = 1+dc(4):nx-dc(2);
dofy = 1+dc(1):ny-dc(3);
Z(dofx,dofy) = reshape(u,length(dofx),length(dofy));
for ie=1:4
    if dc(ie)
        Z(rect{1,ie},rect{2,ie}) = g{ie};
    end
end

end

function ii = inner(i)
ii = i(2:end-1);
if isempty(ii)
    ii = i;
end
end

function [s, ps] = stencil(i,n)
s = [i-1 i+1];
ps = [i-1 i];
fi = s>0 & s<=n;
s = s(fi);
ps = ps(fi);
end
```

s21.m

```
cls

n = 30; m = 20;
x = linspace(0,1,n);
y = linspace(0,1,m);

eq = bvprect;
eq.p = @(x,y) 1+x+2*y;
eq.bc{1} = {@(x,y) 1 @(x,y) 0};
eq.bc{2} = {@(x,y) 0 @(x,y) 0};
u = @(x,y) y.*sin(pi*x+1);
eq = exsol2d(eq,u);

[X,Y,Z] = diff2d(x,y,eq);
U = u(X,Y);

surfc(X,Y,U,'FaceColor','interp');
xlabel('x');
ylabel('y');
print -depsc o21

figure;
mesh(X,Y,abs(U-Z));
print -depsc o21a
```





3.1.2. Metoda konečných prvků

meshrect.m

```
function mesh = meshrect( x, y)
n = length(x);
m = length(y);
ix = 1:n;
```

```
iy = 1:m;
[iX iY] = meshgrid(ix,iy);
mesh.ELEM = delaunay(iX(:),iY(:));
[X Y] = meshgrid(x,y);
mesh.x = X(:);
mesh.y = Y(:);
mesh.REG = ones(size(mesh.ELEM,1),1);
iy = (1:m-1)'; oy = ones(size(iy));
ix = (1:m:(n-1)*m)'; ox = ones(size(ix));
c1 = [iy; (n-1)*m+iy];
c2 = c1 +1;
r1 = [c1 c2];
c1 = [ix; ix+m-1 ];
c2 = c1 + m;
r2 = [c1 c2];
mesh.SIDE = [r1; r2];
mesh.LAB = [4*oy; 2*oy; ox; 3*ox];
end
```

plotmesh.m

```
function plotmesh( m, opt )
if isfield(m,'u')
    if ~exist ( 'opt','var' )
        opt = meshopt;
    end
    pairs = struct2pairs(opt);
    trisurf(m.ELEM,m.x,m.y,m.u,pairs{:});
else
    triplot(m.ELEM,m.x,m.y);
    text(mean(m.x(m.SIDE),2),mean(m.y(m.SIDE),2),int2str(m.LAB),'BackgroundColor
        ',[.7 .9 .7]);
    text(mean(m.x(m.ELEM),2),mean(m.y(m.ELEM),2),int2str(m.REG));
    axis equal;
end
end
```

bvptri.m

```
function eq = bvptri(eq)
if ~exist('eq','var')
    eq.p = {@(x,y) x.^0, @(x,y) x.^0};
    eq.q = {@(x,y) x.^0, @(x,y) x.^0};
    eq.f = {@(x,y) x.^0, @(x,y) x.^0};
    eq.bc = {@(x,y) 0*x};
    %eq.dc = [];
else
    eq.p = {eq.p};
    eq.q = {eq.q};
    eq.f = {eq.f};
end
end
```

s22r.m

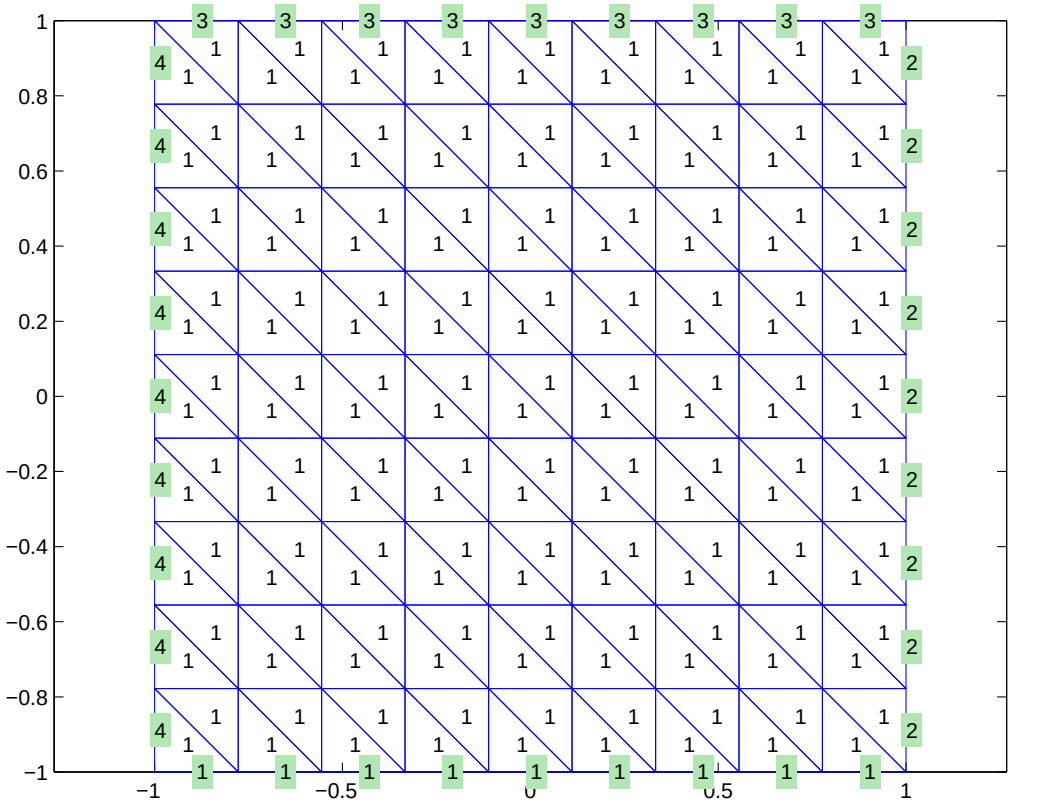
```
cls

x = linspace(-1,1,10);
y = linspace(-1,1,10);
mesh = meshrect(x,y);
plotmesh(mesh);
```

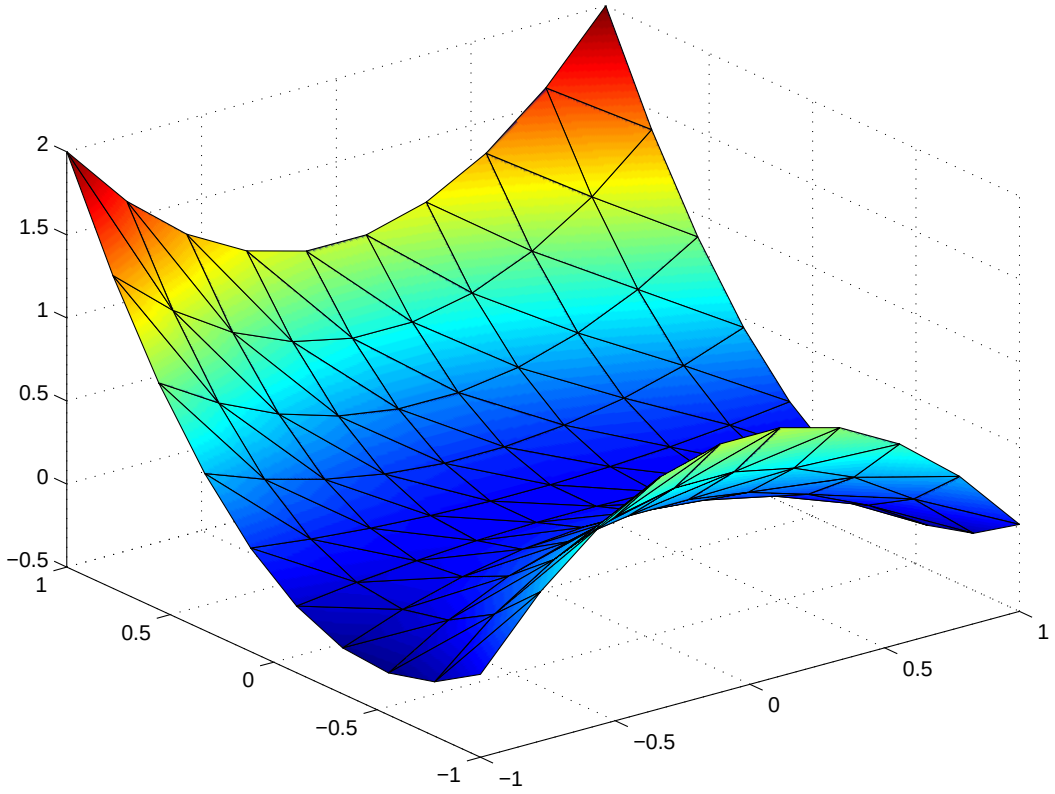
```
print -depsc o22rm

u = @(x,y) x^2*y+y^2;
eq = bvprect;
eq.p = @(x,y) x+y+3;
eq.bc{1} = {@(x,y) 1, @(x,y) 0};
eq.bc{2} = {@(x,y) 0, @(x,y) 0};
eq = exsol2d(eq,u);
eq = bvptri(eq);

mesh.u = MKPd2(mesh,eq);

figure;
plotmesh(mesh);
print -depsc o22r

mesh.u = abs(arrayfun(u,mesh.x,mesh.y)-mesh.u);
figure;
plotmesh(mesh);
```

MKPd2.m

```
function u = MKPd2( mesh, bvp )
nlab = length(bvp.bc);
%Dirichlet/Robin condition (podminka)
dc = []; rc = [];
plab = zeros(size(mesh.x));
for i = 1:nlab
    if length(bvp.bc{i}) == 1
        ii = mesh.LAB == i;
        plab(mesh.SIDE(ii,:))=i;
        dc = [dc i];
    else
        rc = [rc i];
    end
end

nt = size(mesh.ELEM,1);
np = size(mesh.x,1);
F = zeros(np,1);
I = zeros(1,3*nt); J=I; S=I;
o3 = ones(3,1);
O3 = eye(3);
ix=1;

for i=1:nt
    t = mesh.ELEM(i,:);
    r = mesh.REG(i);
    T = [mesh.x(t) mesh.y(t) o3];
    xc = mean(mesh.x(t));
    yc = mean(mesh.y(t));
    d = det(T);
    B = T\O3;
    B = B(1:2,:);
    K1 = 1/2*abs(d)*bvp.p{r}(xc,yc)*B'*B;
    K2 = 1/6*abs(d)*diag(bvp.q{r}(mesh.x(t),mesh.y(t)));
    K = K1+K2;
    I(ix:ix+8) = t([1 1 1 2 2 2 3 3 3]);
    J(ix:ix+8) = t([1 2 3 1 2 3 1 2 3]);
    S(ix:ix+8) = K(:);
    ix = ix+9;
```

```
    F(t) = F(t) + 1/6*abs(d)*bvp.f{r}(mesh.x(t),mesh.y(t));
end
for i = rc
    ii = mesh.LAB == i;
    side = mesh.SIDE(ii,:);
    ns = size(side,1);
    Is = zeros(1,2*ns); Ss=Is;
    dx = mesh.x(side(:,2))-mesh.x(side(:,1));
    dy = mesh.y(side(:,2))-mesh.y(side(:,1));
    d = sqrt(dx.^2+dy.^2);
    alfa = bvp.bc{i}{1};
    beta = bvp.bc{i}{2};
    ix = 1;
    for j = 1:ns
        Is([ix ix+1]) = side(j,:);
        P1.x = mesh.x(side(j,1)); P1.y = mesh.y(side(j,1));
        P2.x = mesh.x(side(j,2)); P2.y = mesh.y(side(j,2));
        Ss([ix ix+1]) = d(j)/2*[alfa(P1.x,P1.y) alfa(P2.x,P2.y)];
        ix = ix + 2;
        F(side(j,:)) = F(side(j,:)) + d(j)/2*[beta(P1.x,P1.y); beta(P2.x,P2.y)];
    end
    I = [I Is]; J = [J Is]; S = [S Ss];
end

K = sparse(I,J,S,np,np);

u = zeros(size(mesh.x));
for i = dc
    ii = plab==i;
    u(ii) = arrayfun(bvp.bc{i},mesh.x(ii),mesh.y(ii));
end

idc = plab>0;
F = F-K(:,idc)*u(idc);
K(idc,:)=[]; K(:,idc)=[]; F(idc)=[];
y = K\F;
u(~idc)=y;
end
```





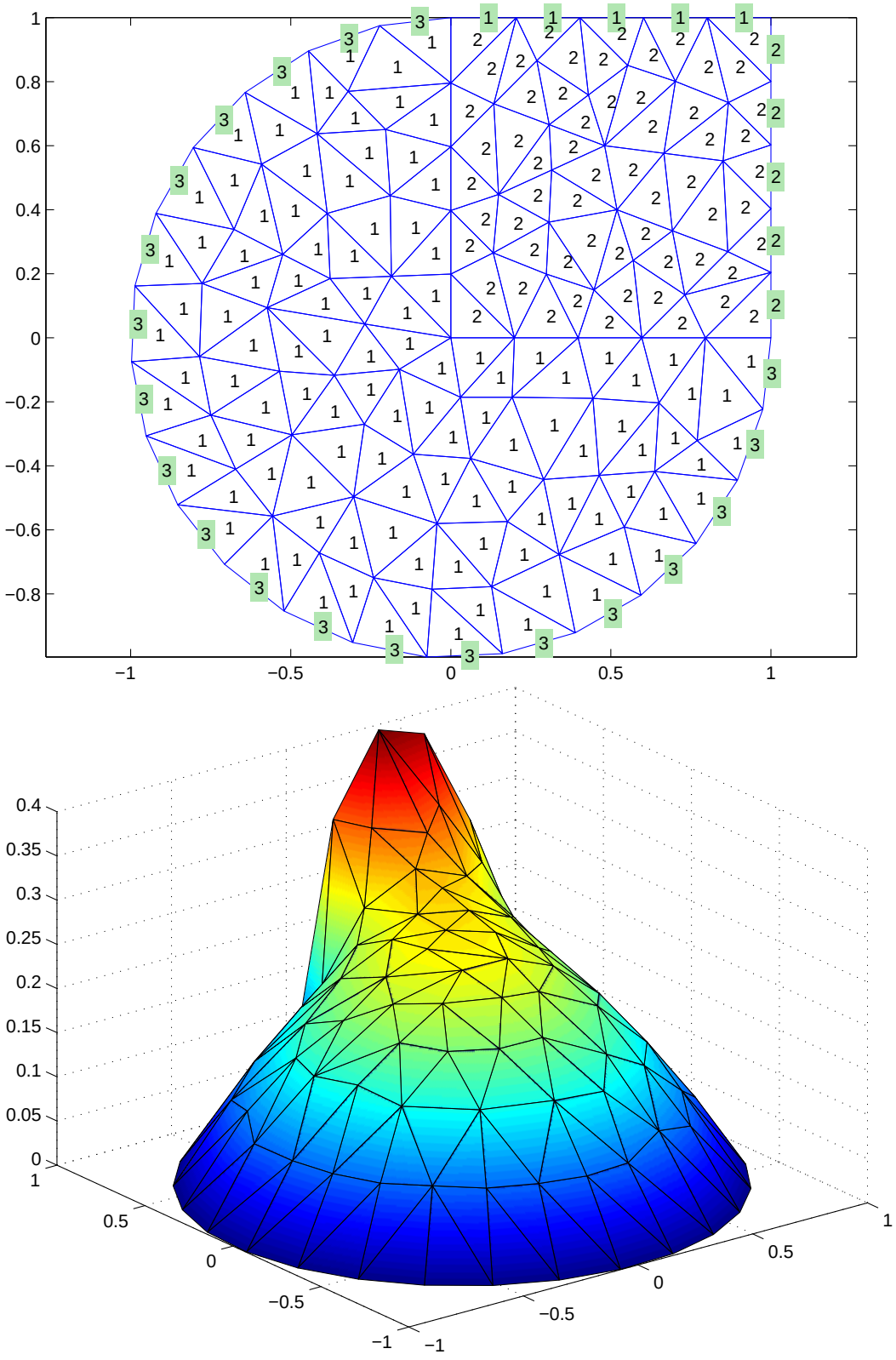
importmesh.m

```
function mesh = importmesh( subdir )
data = importdata([subdir 'nodes.dat']);
x = data(:,1); y = data(:,2);
data = importdata([subdir 'tri.dat']);
tri = data(:,1:3)+1; region = data(:,4)+1;
data = importdata([subdir 'be.dat']);
be = data(:,1:2)+1; belab = data(:,3);
clear data;
```

```
del = belab<0;
be(del,:) = [];
belab(del) = [];
mesh.ELEM = tri;
mesh.REG = region;
mesh.SIDE = be;
mesh.LAB = belab;
mesh.x = x;
mesh.y = y;
end
```

s22.m

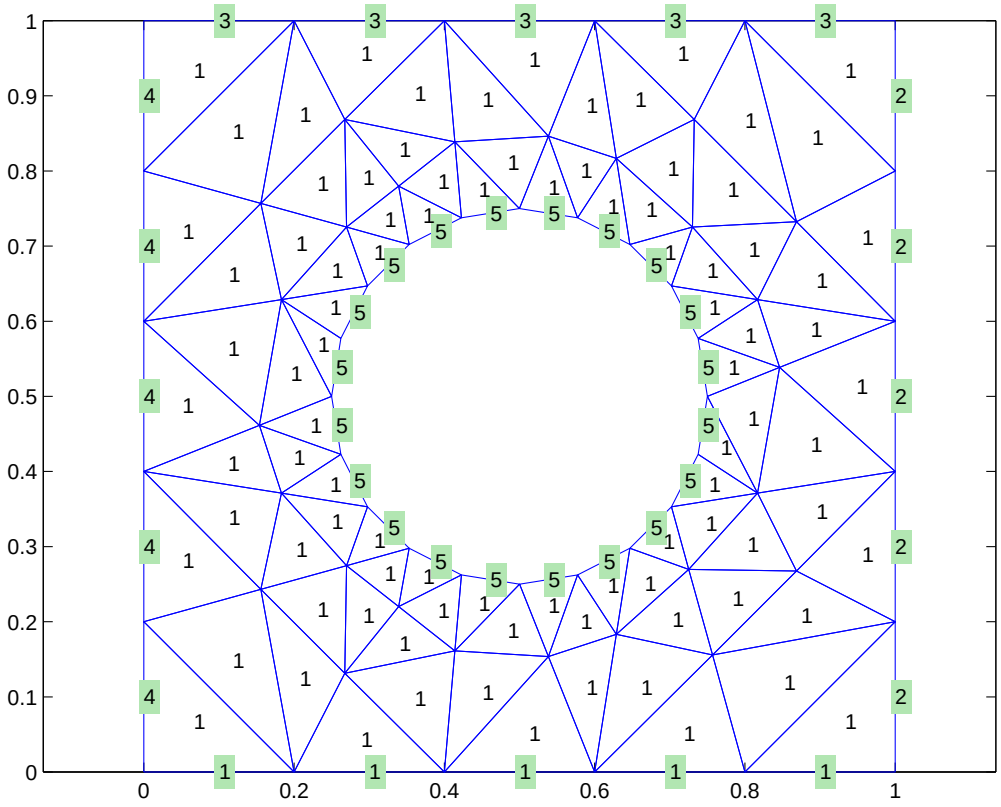
```
cls

mesh = importmesh('data/mesh1/size2/');
plotmesh(mesh);
print -depsc o22m

bvp = bvptri;
bvp.bc = { {@(x,y) x.^0 @(x,y) x.^0} @(x,y) 0*x.^0 @(x,y) 0*x.^0 };
mesh.u = MKPd2(mesh,bvp);

figure;
plotmesh(mesh);
print -depsc o22
```

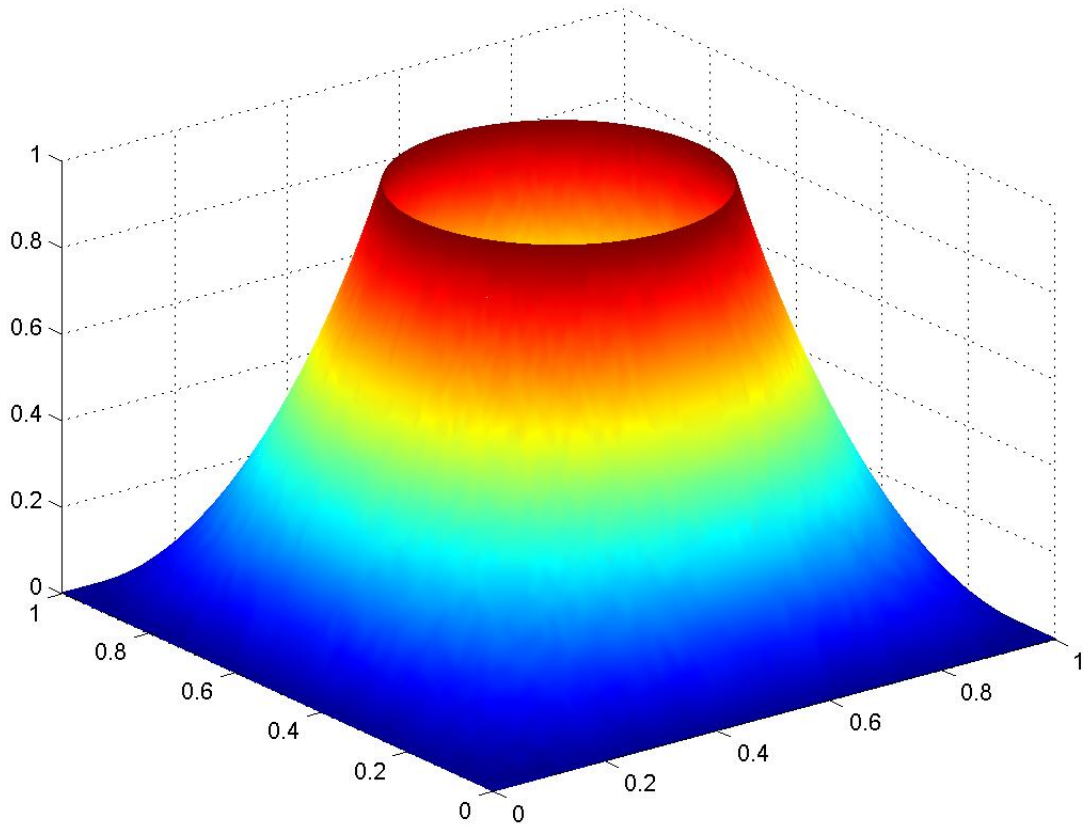


s22a.m

```
cls

mesh = importmesh('data/mesh2/size2/');
plotmesh(mesh);
print -depsc o22ma
mesh = importmesh('data/mesh2/size5/');

bvp = bvptri;
```

```
bvp.bc = { @(x,y) 0*x, @(x,y) 0*x, @(x,y) 0*x, @(x,y) 0*x, @(x,y) x.^0 };
mesh.u = MKPd2(mesh,bvp);

figure;
opt = meshopt;
opt.EdgeColor = 'interp';
plotmesh(mesh,opt);

print -djpeg o22a
```


0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8


0
0.2
0.4
0.6
0.8
1

4. Evoluční problémy

4.1. Úloha parabolického typu

Lineární parabolická rovnice je tvaru

$$c(x)\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x}\left(p(x)\frac{\partial u}{\partial x}\right) + q(x)u = f(x,t), \qquad x \in (0,l)\,,\ t \in (0,T)$$

s okrajovými podmínkami Dirichletova typu

$$\begin{aligned} u(0,t) &= g_0(t)\,, \\ u(l,t) &= g_l(t)\,, \end{aligned}$$

rsp. s kombinací Dirichletovy a Robinovy podmínky

$$\begin{aligned} u(0) &= g_0(t)\,, \\ -p(l)\,u'(l,t) &= \alpha u(l,t) - \beta_l(t) \end{aligned}$$

nebo Robinovy a Dirichletovy podmínky

$$\begin{aligned} p(0)\,u'(0) &= \alpha u(0,t) - \beta_0(t)\,, \\ u(l) &= g_l(t) \end{aligned}$$

a početční podmínkou

$$u(x,0) = \varphi(x) \qquad x \in (0,l)\,.$$

defEvol.m

```
function eq = defEvol
eq.p = @(x) 1;
eq.q = @(x) 0;
eq.r = @(x) 0;
eq.f = @(x,t) 0;
eq.c = @(x) 1;
eq.rho = @(x) 1;
eq.domain = [0 1];
eq.bc = {@(t) 0 @(t) 0};
eq.phi = @(x) 0;
eq.psi = @(x) 0;
end
```

evolK.m

```
function [K,elim] = evolK(x,eq)
elim = cell(1,2);
h = x(2)-x(1);
xmid = (x(2:end)+x(1:end-1))/2;
p = arrayfun(eq.p,xmid);
r = arrayfun(eq.r,xmid);
zero = 0*xmid;
```

```
rplus = max(r,zero);
rminus = min(r,zero);
q = arrayfun(eq.q,x);
dc = [length(eq.bc{1}) length(eq.bc{2})]==1;
n = length(x);
K = sparse(n,n);
for i=2:n-1
    r = i-1:i+1;
    K(i,r) = [-p(i-1) p(i-1)+p(i)+q(i)*h^2 -p(i)] + ... % (2.15)
        [-rplus(i-1) rplus(i)-rminus(i-1) rminus(i)]*h; % (2.29)
end
K1 = cell(2,3);
for i=1:2
    if i==1
        k=1; r=1:2;
    else
        k=size(K,1); r = [k k-1];
    end
    if dc(i)
        K(k,:)=[];
        [I,J,S] = find(K(:,k));
        elim{i} = S;
        K(:,k)=[];
    else
        K(k,r) = [p(k)+h*eq.bc{i}{1}+q(k)*h^2/2 -p(k)] + ... % (2.19a)
            [rplus(k) rminus(k)]*h ; % (2.29)
    end
end
K = K/h^2;
end
```

evolF.m

```
function F = evolF(elim,x,t,eq)
[X,T] = meshgrid(x,t);
F = arrayfun(eq.f,X,T)';
dc = ~cellfun(@isempty,elim);
h = x(2)-x(1);
ix = {@(f) 1,@(f) size(f,1)};
for i=1:2
    if dc(i)
        F(ix{i}(F),:)=[];
        F(ix{i}(F),:) = F(ix{i}(F),:) - elim{i}*eq.bc{i}(t)/h^2;
    else
        F(ix{i}(F),:) = eq.bc{i}{2}(t)/h + F(k,:)/2;
    end
end
end
```

theta.m

```
function U = theta( x, t, eq, th )
if ~exist('eq','var')
    eq = defEvol;
end
if ~exist('th','var')
    th = 1/2;
end

[K,elim] = evolK(x,eq);
lt = length(t);
```

```
lx = length(x);

F = evolF(elim,x,t,eq);

U = zeros(length(x),lt);
U(:,1) = eq.phi(x);
dc = [length(eq.bc{1}) length(eq.bc{2})]==1;
r = 1+dc(1):lx-dc(2);
C = diag(arrayfun(eq.c,x(r)));
for n=1:lt-1
    tau = t(n+1)-t(n);
    U(r,n+1) = (C+tau*th*K)\((C-tau*(1-th)*K)*U(r,n)+ ...
        tau*((1-th)*F(:,n)+th*F(:,n+1)));
end
if dc(1)
    U(1,:) = eq.bc{1}(t);
end
if dc(2)
    U(end,:) = eq.bc{2}(t);
end
U=U';
end
```

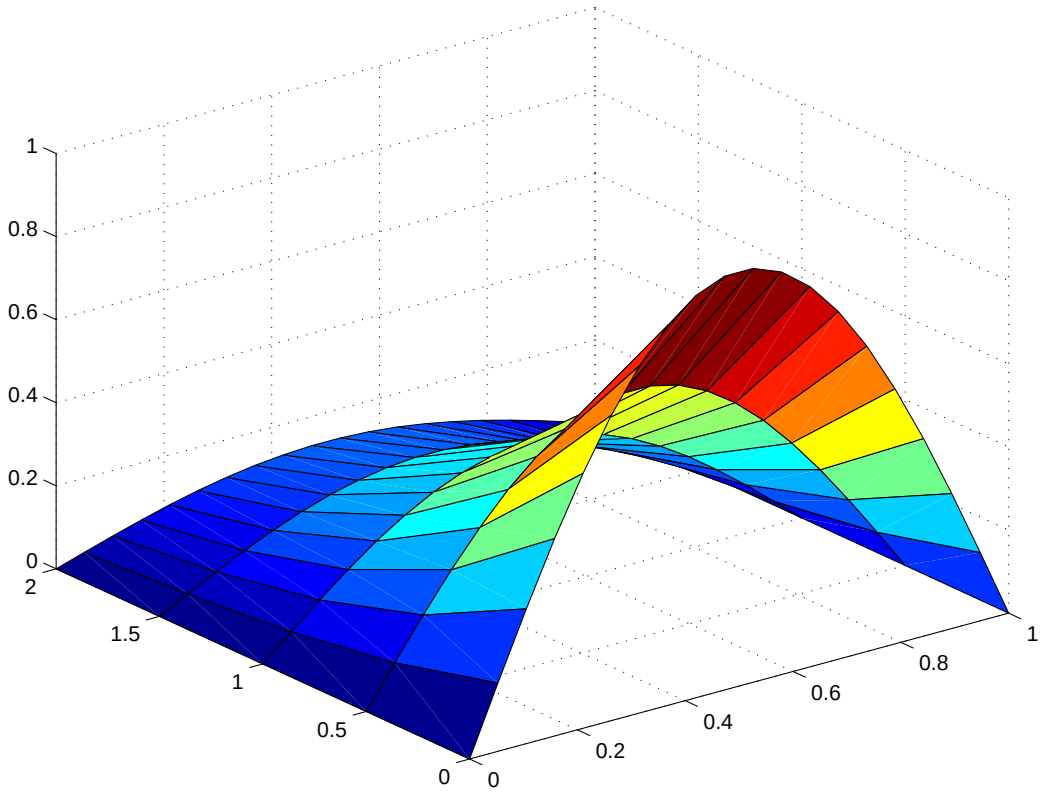
V následujícím skriptu si povšimněme volání funkce ode23s (změňte na ode23t, ode23tb, ode15s) jako alternativu ke Crank-Nicolsonově metodě.

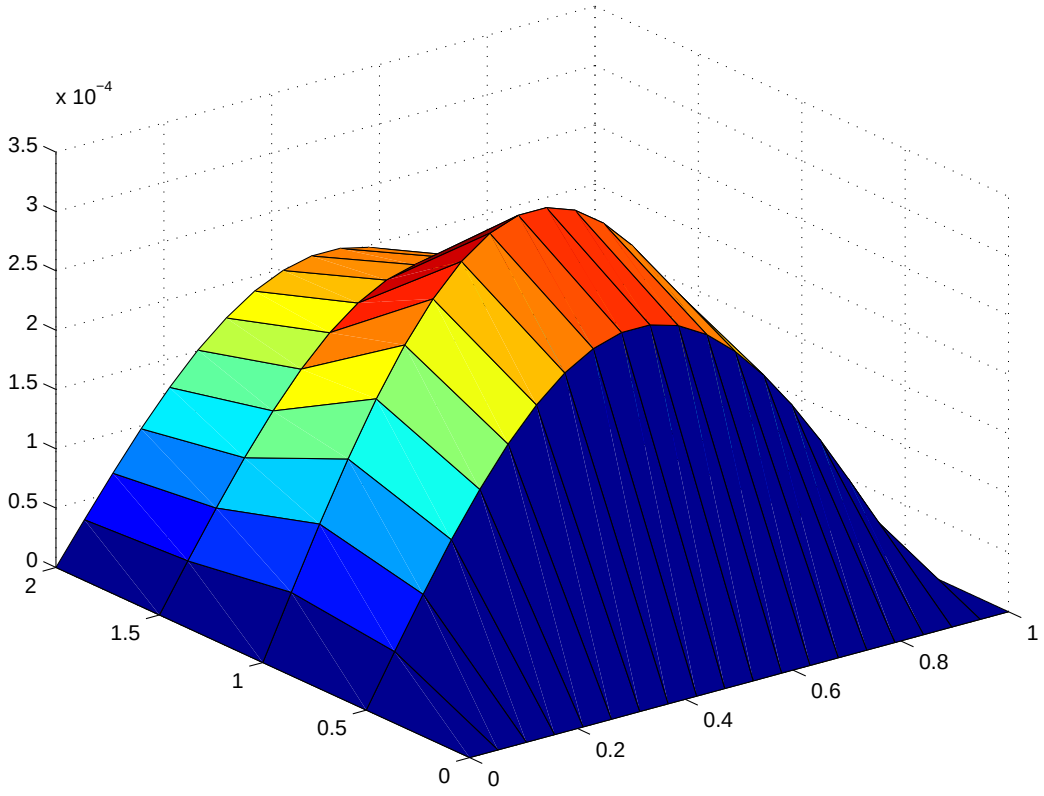
```
odefun = @(t,y) rC*(-K*y + evolF(elim,x,t,bvp));
[tt,y] = ode23s(odefun,t,bvp.phi(xin));
```

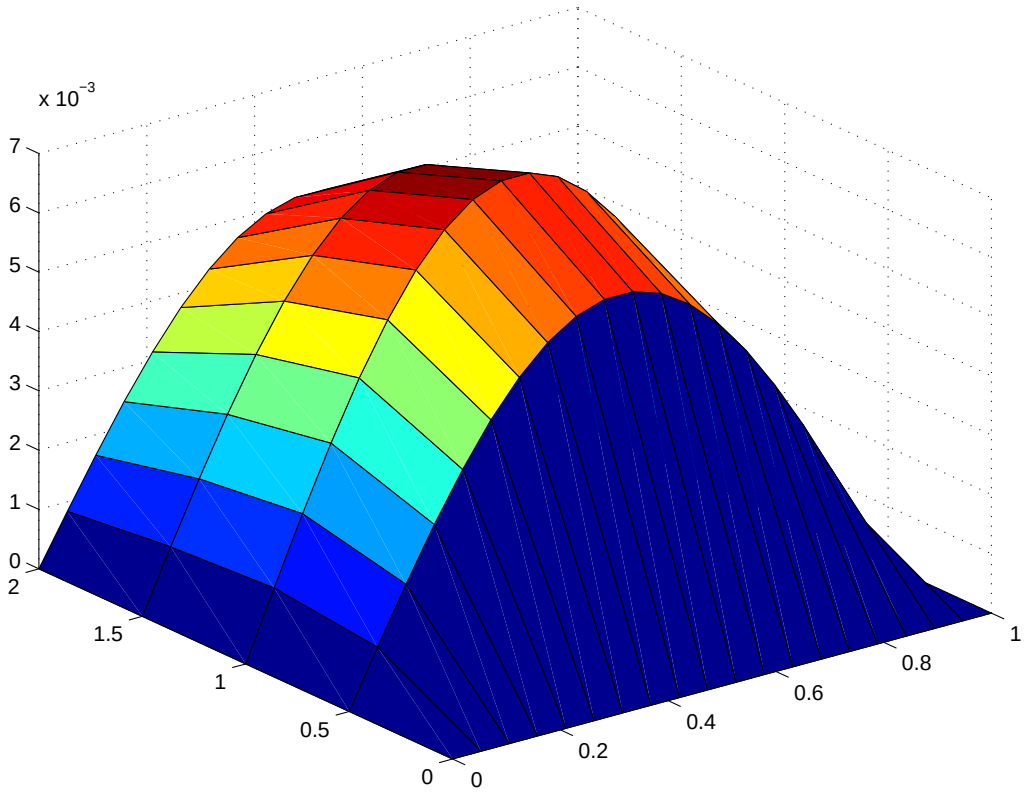
s23.m

```
cls

bvp = defEvol;

bvp.rho = @(x) 0;
bvp.c = @(x) pi^2;

u = @(x,t) sin(pi*x)*exp(-t);
bvp=exsolevol(bvp,u);

x = linspace(0,1,20);
t = linspace(0,2,5);

tic
U = theta(x,t,bvp);
toc

[K,elim] = evolK(x,bvp);
lt = length(t);
lx = length(x);


xin = x(2:end-1);
rC = diag(1./bvp.c(xin));

odefun = @(t,y) rC*(-K*y + evolF(elim,x,t,bvp));

tic
[tt,y] = ode23s(odefun,t,bvp.phi(xin));
toc
```

```
Y = [arrayfun(bvp.bc{1},t') y arrayfun(bvp.bc{2},t')];

[X,T] = meshgrid(x,t);

surf(X,T,U);
print -depsc o23

figure
surf(X,T,abs(Y-arrayfun(u,X,T)));
print -depsc o23a

figure
surf(X,T,abs(U-arrayfun(u,X,T)));
print -depsc o23b
```


1
0.8
0.6
0.4
0.2
0
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1


x 10^-4
3.5
3
2.5
2
1.5
1
0.5
0
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1



4.2. Úloha hyperbolického typu

Lineární hyperbolická rovnice 2. řádu je tvaru

$$\rho(x)\frac{\partial^2 u}{\partial t^2} + c(x)\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x}\left(p(x)\frac{\partial u}{\partial x}\right) + q(x)u = f(x,t), \qquad x \in (0,l),\ t \in (0,T)$$

s okrajovými podmínkami Dirichletova typu

$$\begin{aligned} u(0,t) &= g_0(t), \\ u(l,t) &= g_l(t), \end{aligned}$$

rsp. s kombinací Dirichletovy a Robinovy podmínky

$$\begin{aligned} u(0) &= g_0(t), \\ -p(l)\,u'(l,t) &= \alpha u(l,t) - \beta_l(t) \end{aligned}$$

nebo Robinovy a Dirichletovy podmínky

$$\begin{aligned} p(0)\,u'(0) &= \alpha u(0,t) - \beta_0(t), \\ u(l) &= g_l(t) \end{aligned}$$

a početčním podmínkám

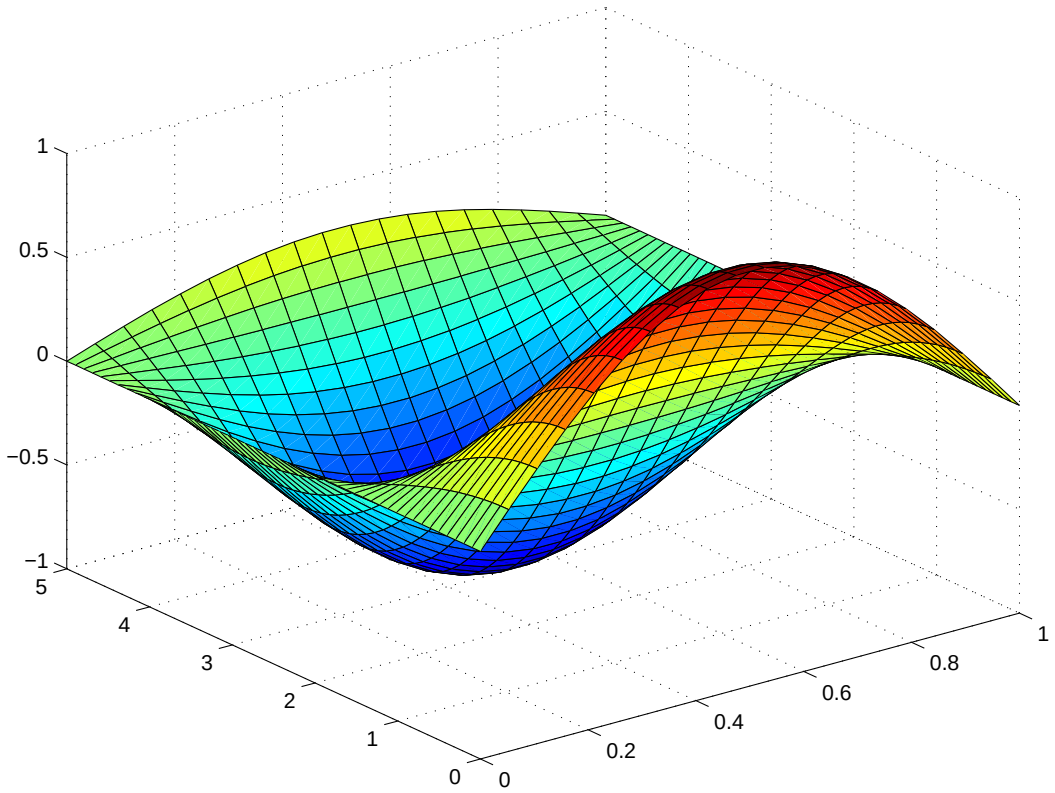
$$\begin{aligned} u(x,0) &= \varphi(x) \qquad x \in (0,l), \\ \frac{\partial u(x,0)}{\partial t} &= \psi(x) \qquad x \in (0,l). \end{aligned}$$

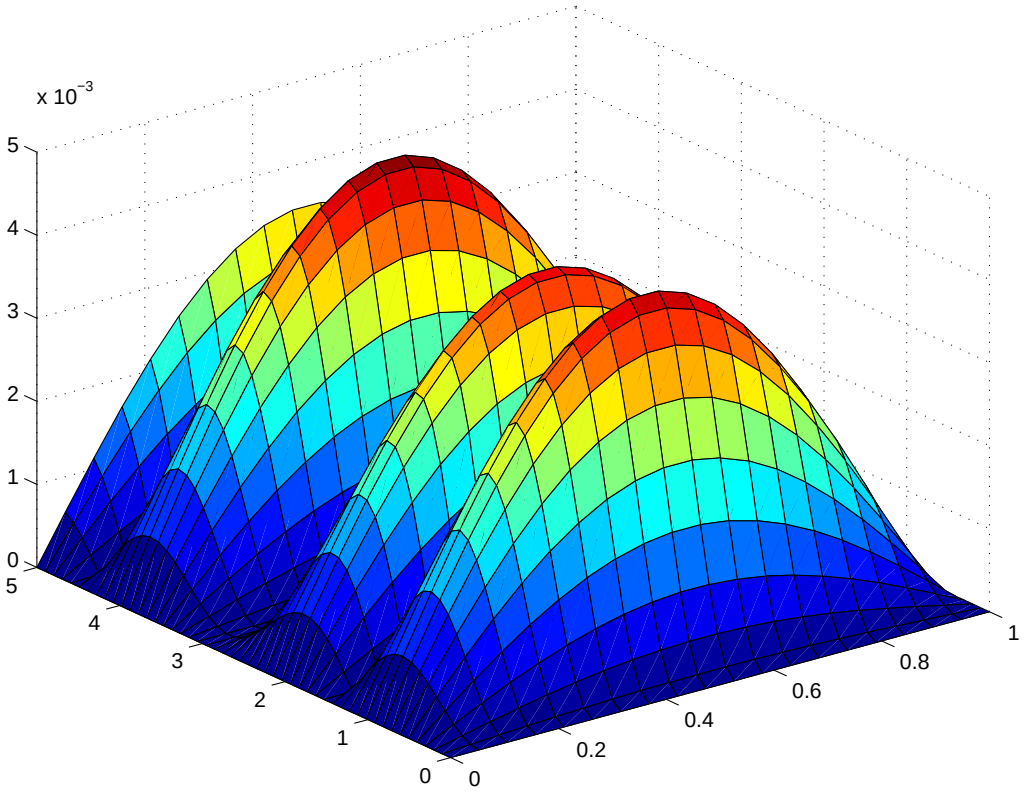
newmark.m

```
function [U,dU] = newmark( x, t, eq)
if ~exist('eq','var')
    eq = defEvol;
end
%[K,F,X,T] = evolF(x,t,eq);
[K,elim] = evolK(x,eq);
lt = length(t);
lx = length(x);
F = evolF(elim,x,t,eq);
U = zeros(length(x),lt);
dU = zeros(length(x),lt);
U(:,1) = eq.phi(x);
dU(:,1) = eq.psi(x);
dc = [length(eq.bc{1}) length(eq.bc{2})]==1;
r = 1+dc(1):lx-dc(2);
C = diag(arrayfun(eq.c,x(r)));
M = diag(arrayfun(eq.rho,x(r)));
for n=1:lt-1
    tau = t(n+1)-t(n);
    D = 4*M/tau^2+2*C/tau;
    U(r,n+1) = (D+K) \ ( (D-K)*U(r,n)+4*M/tau*dU(r,n)+F(:,n)+F(:,n+1) );
    dU(r,n+1) = 2/tau*(U(r,n+1)-U(r,n))-dU(r,n);
end
if dc(1)
    U(1,:) = eq.bc{1}(t);
end
if dc(2)
    U(end,:) = eq.bc{2}(t);
end
U=U';
end
```

s24.m

```
cls

bvp = defEvol;
bvp.c = @(x) 0*x;

gamma = 1;

%presne reseni
u = @(x,t) sin(pi*x)*cos(t);
bvp=exsolevol(bvp,u);
%bvp.f = @(x,t) 0;

x = linspace(0,1,20);
t = linspace(0,5,50);

U = newmark(x,t,bvp);
[X,T] = meshgrid(x,t);

surf(X,T,U);
print -depsc o24

figure
surf(X,T,abs(U-arrayfun(u,X,T)));
print -depsc o24a
```


1
0.5
0
−0.5
−1
5
4
3
2
1
0
0
0.2
0.4
0.6
0.8
1


x 10^−3
5
4
3
2
1
0
5
4
3
2
1
0
0
0.2
0.4
0.6
0.8
1

Označení

$\approx$ aproximuje, přibližně rovno

Index

Implicitní Eulerova metoda, 25

TR-metoda, 28

Matlab, funkce a skripty

ABStep, 44
Adams, 43

BDF, 46
BDFStep, 46
Butcher, 35

cls, 3

diff1d, 55
diff2d, 63

EE, 18
evolF, 77
evolK, 76

IE, 25
importmesh, 71

Kepler, 48

meshrect, 67
MKPd1, 57
MKPd2, 69

NBody, 50
newmark, 81

PCStep, 45
plotmesh, 68

RKStep, 37
RKStep2, 37

struct2pairs, 3

theta, 77

Literatura

[1] J. Alberty, C. Carstensen, and S. A. Funken, *Remarks around 50 lines of matlab: Short finite element implementation*, Numerical Algorithms, 20 (1998), pp. 117–137.

[2] K. Atkinson, W. Han, and D. Stewart, *Numerical Solution of Ordinary Differential Equations*, Pure and Applied Mathematics, Wiley, 2009.

[3] S. Attaway, *Matlab: A Practical Introduction to Programming and Problem Solving*, Elsevier Science, 2011.

[4] P. Deuflhard and F. Bornemann, *Numerische Mathematik 2: Gewöhnliche Differentialgleichungen*, De Gruyter Lehrbuch Series, De Gruyter, 2008.

[5] L. Čermák, *Numerické metody pro řešení diferenciálních rovnic*, Ústav matematiky FSI VUT v Brně, 2013.

[6] M. J. Gander, *A Non Spiraling Integrator for the Lotka Volterra Equation*.

[7] D. Griffiths and D. Higham, *Numerical Methods for Ordinary Differential Equations: Initial Value Problems*, Springer Undergraduate Mathematics Series, Springer, 2010.

[8] E. Hairer, S. Norsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer Series in Computational Mathematics, Springer, 2010.

[9] F. Hecht, *Freefem*, 2012.

[10] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2009.

[11] C. B. Moler, *Numerical computing with MATLAB*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2008.

[12] L. F. Shampine and M. W. Reichelt, *The matlab ode suite*.