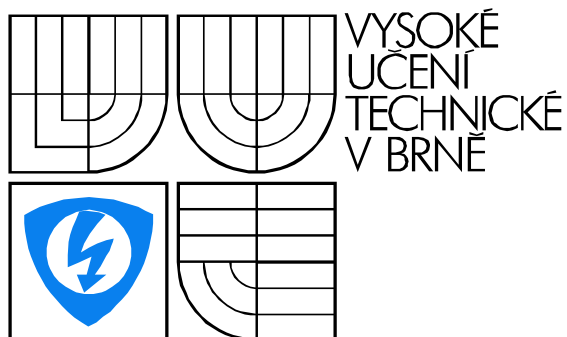




FAKULTA
ELEKTROTECHNIKY
A KOMUNIKAČNÍCH
TECHNOLOGIÍ

Diferenciální rovnice a jejich použití v elektrotechnice – práce s programem MATLAB

Prof. RNDr. Josef Diblík, DrSc.
Mgr. Irena Hlavičková, Ph.D.
RNDr. Zdeněk Svoboda, CSc.

Úvod

Tento text tvoří doplněk ke skriptům Diferenciální rovnice a jejich použití v elektrotechnice. Odkazy na příklady, rovnice, apod. uvedené zkratkou MDRE- se vztahují právě k příkladům a rovnicím v těchto skriptech. Cílem textu je poskytnout návody, jak používat počítač, konkrétně program MATLAB, při řešení různých problémů spojených s probíranou látkou. Dříve, než se pustíte do práce s počítačem, zopakujte si vždy příslušnou problematiku a projděte si „ruční“ způsob řešení příkladů. Očekáváme, že pro čtenáře tohle není první setkání s MATLABem a že alespoň základní práci s ním už ovládá.

Veškeré předvedené příklady byly napsány v MATLABu 7.8.0.347(R2009a) s tím, že pro symbolické výpočty se používá MuPAD. Používá-li někdo jinou verzi MATLABu nebo vybral-li si pro symbolické výpočty Maple (coby součást MATLABu, ne Maple jako samostatný program), je možné, že v některých případech se budou výsledky lišit. Zvlášť je to možné při řešení některých diferenciálních rovnic nebo při zjednodušování některých výrazů.

Také bychom chtěli upozornit, že některá řešení úmyslně nejsou úplně elegantní (obvykle je na takových místech v textu upozornění typu „Bylo by lepší udělat to tak a tak...“). Chceme tím čtenářům poskytnout prostor pro vlastní tvořivost. Je samozřejmě možné, že některá řešení jsou neelegantní nebo dokonce chybná zcela neúmyslně – v tom případě budeme vděční za upozornění.

1 Symbolický toolbox MATLABu

MATLAB je primárně určen pro výpočty numerické, obzvláště pro různé výpočty s maticemi. Symbolický toolbox jeho možnosti rozšiřuje navíc o výpočty symbolické, např. derivování, integrování, řešení diferenciálních rovnic a pod. V této kapitole si ukážeme nejjednodušší použití symbolického toolboxu. Zaměříme se na funkce, které se budou používat i v dalším textu.

1.1 Symbolické proměnné a výrazy

Symbolickou proměnnou definujeme pomocí funkce `sym`. Chceme-li proměnné hned přiřadit nějakou hodnotu, uděláme to např. takto:

```
a = sym(1/2);  
b = sym(2/3);
```

Tímto jsme definovali dvě symbolické proměnné, `a` a `b`, a přiřadili jim hodnoty $1/2$ a $2/3$. Nyní s těmito proměnnými můžeme provádět různé výpočty, např.:

```
soucet = a+b  
sucin = a*b
```

dá výsledek

```
soucet =  
7/6  
sucin =  
1/3
```

Výpočty byly provedeny symbolicky – přesně se zlomky. Podíváme-li se na seznam definovaných proměnných (Workspace), uvidíme, že nám tímto přibyly další dvě symbolické proměnné, `soucet` a `sucin`.

Jiná možnost je vytvořit symbolickou proměnnou s hodnotou, kterou má nějaká „obyčejná“ proměnná:

```
c = 5;  
d = sym(c)  
  
d =  
5
```

Potřebujeme-li vytvořit symbolickou matici, uděláme to podobně:

```
A = sym([1 2; 3 4]);
```

nebo

```
A = [1 2; 3 4];  
symA = sym(A);
```

Jestliže do symbolické proměnné zatím žádnou hodnotu přiřazovat nechceme, napíšeme např.

```
x = sym('x')
```

Tím jsme definovali symbolickou proměnnou x , která zatím žádnou hodnotu nemá. Dále ale musíme být při práci s touto proměnnou opatrní. Napíšeme-li teď např.

```
x = 1;
```

stane se tím z x opět „obyčejná“ proměnná s hodnotou 1. Chceme-li definovat více symbolických proměnných, aniž bychom jim přiřazovali nějaké hodnoty, je pohodlnější použít příkaz `syms`. Např. příkazem

```
syms x y z
```

jsem definovali symbolické proměnné x , y a z .

Potřebujeme-li definovat symbolickou matici, která zatím není vyplněná žádnými hodnotami, lze to udělat např. takto:

```
syms a11 a12 a21 a22  
A = [a11, a12; a21, a22]
```

1.2 Dosazování

Pro dosazení do symbolického výrazu je určena funkce `subs`. Můžeme ji použít různými způsoby. Jeden z nich je

```
výsledek = subs( do_čeho, za_co, hodnota)
```

Například

```
syms x  
f = x^2;  
subs(f,x,'2*t')
```

dá výsledek

```
ans =  
4*t^2
```

Do výrazu můžeme dosadit i celý vektor nebo matici hodnot. Výsledkem je pak vektor, resp. matice. Např.

```
subs(f,x,[1 2; 3 4])
```

(pracujeme s proměnnými f a x z předchozího příkladu) dá výsledek

```
ans =  
1      4  
9      16
```

Jiná možná syntaxe je

```
výsledek = subs(výraz)
```

aniž bychom zadali, za co a jaká hodnota se má dosazovat. Tato forma se používá, jestliže v některých nebo všech proměnných, na kterých výraz závisí, jsou přiřazeny nějaké hodnoty. Ty jsou pak do výrazu dosazeny. Např. takto:

```
f = sym('x+2*y-z');  
x = 2;  
y = 3;  
subs(f)  
  
ans =  
8 - z
```

1.3 Zjednodušování výrazů

Pro úpravy výrazů existuje více funkcí. Nejpoužívanější je funkce `simplify`:

```
f = sym('sin(x)^2+cos(x)^2');  
simplify(f)  
  
ans =  
1
```

Je tu ale jeden problém: chceme-li nějaký výraz zjednodušit, není předem jasné, která forma výrazu je ta nejlepší. Někdy se nám lépe hodí jeden tvar, pro jiné účely zase jiný. Funkce `simplify` nemusí dát vždy takový výsledek, jaký bychom si představovali.

1.4 Derivování

Pro symbolické derivování slouží funkce `diff`. Můžeme ji volat s různým počtem argumentů, vždy však zadáme výraz, který se má zderivovat. Tento výraz může být zadán jako textový řetězec nebo to může být symbolická proměnná.

Zavoláme-li funkci pouze s jedním argumentem, zderivuje se zadaný výraz podle `x`, je-li v něm `x` obsaženo. Např.

```
diff('2*a*x+3*b-4*x')  
  
ans =  
2*a - 4
```

Jestliže ve výrazu žádné `x` není, zderivuje se podle proměnné, která je k `x` abecedně nejbližší.

Přehlednější je zadat vždy i proměnnou, podle které se derivuje, jako druhý argument funkce:

```
syms t
f = t*sin(t);
diff(f,t)

ans =
sin(t) + t*cos(t)
```

Tímto způsobem počítáme i parciální derivace:

```
syms x y z
f = x^2+y*z;
dfdx = diff(f,x)
dfdy = diff(f,y)
dfdz = diff(f,z)

dfdx =
2*x
dfdy =
z
dfdz =
y
```

Pro výpočet derivací vyšších řádů zadáme ještě jako třetí argument, kolikrát se má funkce derivovat:

```
syms x
f = x^3;
d2 = diff(f,x,2)
d3 = diff(f,x,3)

d2 =
6*x
d3 =
6
```

Připomeňme, že funkce `diff` existuje i v základní verzi MATLABu. Dělá však něco jiného. Počítá difference neboli rozdíly po sobě jdoucích složek zadaného vektoru. Např.

```
x = [1 3 10 20];
diff(x)

ans =
     2     7    10
```

1.5 Integrovaní

V MATLABu integrujeme pomocí funkce `int`. Podobně jako u derivování, není-li řečeno jinak, integruje se podle x . V případě, že integrovaný výraz x neobsahuje, integruje se podle proměnné, která je abecedně k x nejbližší.

```
syms x
f = sin(x);
int(f)

ans =
-cos(x)
```

Všimněte si, že výsledek neobsahuje integrační konstantu „+c“. V některých případech to může působit problémy, ale o tom ještě bude řeč v některé z dalších kapitol. Integrační proměnnou můžeme zadat jako druhý parametr funkce `int`:

```
syms a t
f = cos(a*t);
int(f,t)

ans =
sin(a*t)/a
```

Pro výpočet určitého integrálu zadáme meze jako další argumenty:

```
syms x
f = x^2;
int(f,x,0,1)

ans =
1/3
```

Na závěr této kapitoly poznamenejme, že někdy trvá integrování počítači překvapivě dlouho a uživatel se musí obrnit trpělivostí.

2 Obyčejné diferenciální rovnice prvního řádu

V kapitole 2.2 skript MDRE jsme si připomněli obyčejné diferenciální rovnice prvního řádu. Nyní ukážeme, jak se tyto rovnice řeší pomocí MATLABu.

2.1 Obecné řešení

Obecné řešení diferenciální rovnice prvního řádu najdeme pomocí funkce

```
řešení = dsolve('rovnice','proměnná')
```

Vstupní argumenty funkce jsou řetězce, výsledek však je typu `sym`.

Proměnnou se myslí nezávislá proměnná v rovnici, nejčastěji tedy x nebo t . Jestliže žádné jméno proměnné nezadáme, jako default se bere proměnná t .

Derivace hledané funkce se značí písmenem D .

Příklad 2.1 Najděte obecné řešení rovnice $xy' + y = \sin x$ (viz skripta MDRE, rovnice [r1](#)).

Řešení:

```
dsolve('x*Dy+y=sin(x)','x')
```

Výsledek pak vypadá takto:

```
ans =  
-(C2 + cos(x))/x
```

Obecné řešení je tedy $y = -(c + \cos(x))/x$.

Kdybychom nezadali, že nezávislá proměnná je x , počítalo by se s nezávislou proměnnou t , zatímco x by se bralo jako parametr:

```
dsolve('x*Dy+y=sin(x)')
```

dá výsledek

```
ans =  
sin(x) - C4/exp(t/x)
```

Tímto příkazem bychom vyřešili rovnici $x \frac{dy}{dt} + y = \sin(x)$.

Na rozdíl od Maplu, který najde i řešení dané implicitně, MATLAB se vždy snaží najít explicitní řešení. Např. pro rovnici MDRE-[r2](#), $x + 1 + x(y - 1)y' = 0$, jejíž řešení lze vyjádřit rovnicí $\frac{y^2}{2} - y + x + \ln|x| + c = 0$, dostaneme dvě možnosti pro explicitní řešení:

```
dsolve('x+1+x*(y-1)*Dy=0','x')
```

```
ans =  
2^(1/2)*(C7 - x - log(x))^(1/2) + 1  
1 - 2^(1/2)*(C7 - x - log(x))^(1/2)
```

Jestliže se MATLABu explicitní řešení najít nepodaří, vypíše varování a vrátí prázdný výsledek. Někdy však může trvat delší dobu, než svou porážku uzná. V tom, které rovnice se podaří vyřešit, se jednotlivé verze MATLABu různí. Jako příklad uveďme rovnici $y'(1 + \cos y) = 1$, kterou lze „ručně“ vyřešit celkem snadno (řešení dané implicitně je $y + \sin y - x = c$):

```
dsolve('Dy*(1+cos(y))=1','x')
```

```
Warning: Explicit solution could not be found.
```

```
ans =
```

```
[ empty sym ]
```

2.2 Řešení počáteční úlohy

Dále si ukážeme, jak se hledá řešení počáteční úlohy. Počáteční podmínku zadáme jako druhý parametr funkce `dsolve`, ještě před nezávislou proměnnou.

Příklad 2.2 Najděte řešení rovnice $y' + 2xy = x$, které vyhovuje počáteční podmínce $y(0) = 1$.

Řešení:

```
dsolve('Dy+2*x*y=x','y(0)=1','x')
```

```
ans =
```

```
1/2+1/2*exp(-x^2)
```

Príslušné řešení je tedy $y = 1/2 + e^{-x^2}/2$.

Předvedeme ještě jinou možnost řešení, která je ovšem komplikovanější. Můžeme ji použít, jestliže si chceme zkontrolovat náš „ruční“ výpočet krok za krokem:

Nejprve najdeme obecné řešení, které si pro další použití uložíme do proměnné `res`:

```
res = dsolve('Dy+2*x*y=x','x')
```

```
res =
```

```
1/2+exp(-x^2)*C1
```

Do řešení dosadíme za x nulu. Použijeme k tomu funkci

```
subs(do_čeho, za_co, hodnota)
```

Argumenty této funkce mohou být symbolické proměnné nebo řetězce, `hodnota` může být i číselná.

```
res0 = subs(res,'x',0)
```

```
res0 =
```

```
1/2+C1
```

Nyní necháme vyřešit rovnici $1/2 + C1 = 1$. K řešení rovnic slouží funkce

```
solve(rovnice, neznámá)
```

Argumenty mohou být symbolické proměnné nebo řetězce. V případě, že je `rovnice` typu `sym` nebo řetězec, který neobsahuje rovnítko, vyřeší se rovnice `rovnice=0`.

My zatím máme k dispozici jen levou stranu rovnice a musíme k ní nějakým způsobem přidat pravou stranu. Můžeme to udělat např. takto:

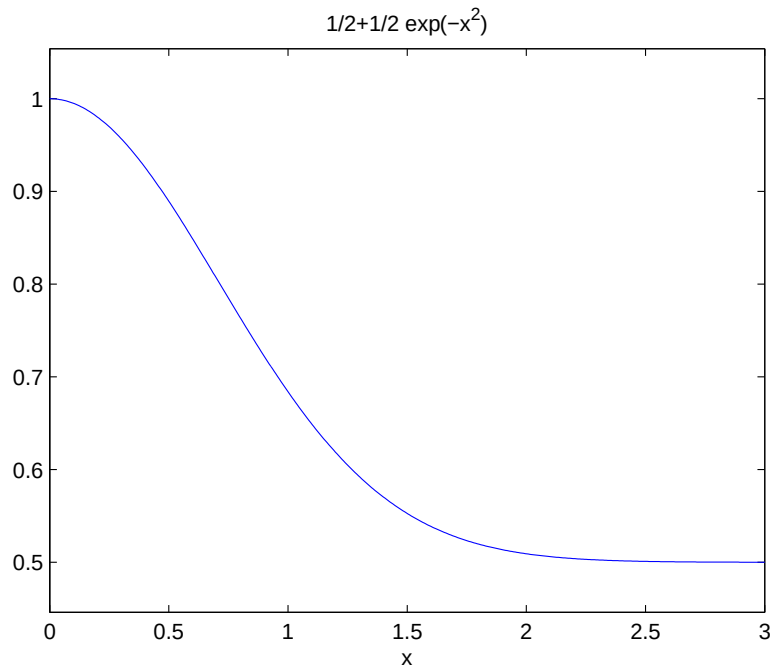
```
rovnice = res0-1;  
rovnice =  
-1/2+C1  
solve(rovnice,'C1')  
ans =  
1/2
```

Teď ještě do řešení uloženého v proměnné `res` dosadíme vypočtenou hodnotu za `C1`:

```
res = subs(res,'C1',ans)  
res =  
1/2+1/2*exp(-x^2)
```

Na závěr můžeme ještě nakreslit graf nalezeného řešení. Pro kreslení grafu funkce uložené v symbolické proměnné můžeme např. použít funkci `ezplot` (graf kreslíme na intervalu $\langle 0, 3 \rangle$):

```
ezplot(res,[0,3])
```



S téměř stejným výsledkem můžeme použít funkci `plot`:

```
xx = 0:0.05:3;  
yy = subs(res,'x',xx);  
plot(xx,yy)
```

Celý program pro nalezení řešení spolu s nakreslením grafu najdete v souboru `dsolve_poc_uloha.m`

2.3 Směrové pole

Nejprve stručně zopakujme, co je to směrové pole rovnice

$$y' = f(x, y). \quad (2.1)$$

Funkce $f(x, y)$ nám udává hodnotu derivace řešení v každém bodě (x, y) z definičního oboru funkce f . To znamená, že jestliže graf nějakého řešení rovnice (2.1) prochází určitým bodem (x, y) , pak tečna ke grafu tohoto řešení v tomto bodě má směrnici $f(x, y)$. Směrové pole rovnice (2.1) znázorňujeme tak, že ve vybrané síti bodů (x, y) nakreslíme šipky s příslušnými směrnici. Řešení rovnice (2.1) se pak „řídí“ těmito šipkami.

Pro kreslení směrového pole bohužel v MATLABu není hotová funkce. Můžeme však použít funkci `quiver`, která je určena pro kreslení obecného vektorového pole v rovině. Syntaxe je

```
quiver( x, y, u, v)
```

kde x a y jsou souřadnice bodů, ve kterých jsou umístěny začátky vektorů ve směru (u, v) . Například v bodě $(x(1), y(1))$ je umístěn vektor ve směru $(u(1), v(1))$. Velikost vektoru není dodržena, jde pouze o daný směr. Velikosti jednotlivých vektorů MATLAB automaticky škáluje, koeficient si můžeme volit.

Pro lepší pochopení této funkce si ukážeme jednoduchý příklad, který zatím nebude nijak souviset se směrovým polem.

Příklad 2.3 *Mějme dva body, $(x_1, y_1) = (-1, 0)$ a $(x_2, y_2) = (2, 3)$. Do bodu (x_1, y_1) umístíme vektor ve směru $\vec{w}_1 = (u_1, v_1) = (1, 4)$ a do bodu (x_2, y_2) umístíme vektor ve směru $\vec{w}_2 = (u_2, v_2) = (-2, -8)$ (opačný směr než má $\vec{w}_1$, velikost je dvojnásobná).*

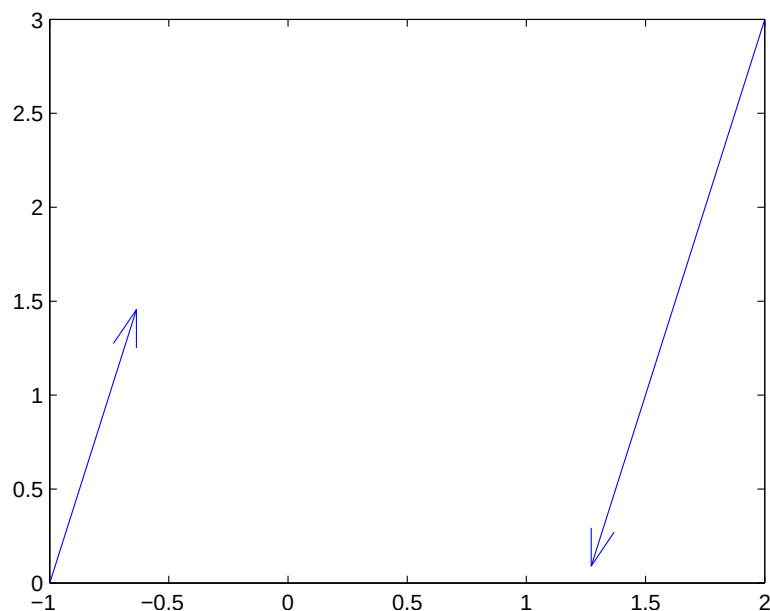
Řešení:

```
x = [-1, 2];
y = [0, 3];
u = [1, -2];
v = [4, -8];
quiver(x,y,u,v)
```

Všimněte si, že zůstal zachován poměr délek vektorů. Jinak ovšem nebyl zobrazen přímo vektor $(1, 4)$, ale pouze vektor, který má stejný směr. Velikost byla zvolena automaticky, aby se vektory dobře vešly do obrázku. Chceme-li, aby se vektory zobrazovaly delší nebo naopak kratší, můžeme zadat funkci `quiver` další parametr:

```
quiver(x,y,u,v,scale)
```

kde `scale` je parametr, kterým se relativní délka vektorů násobí.



Nyní už přikročíme ke kreslení směrového pole.

Příklad 2.4 *Znázorněte směrové pole rovnice*

$$y' = x + y. \quad (2.2)$$

Najděte řešení vyhovující počáteční podmínce $y(0) = -1/2$ a nakreslete do stejného obrázku jeho graf.

Řešení: Směrové pole nakreslíme např. na oblasti $\langle -1, 3 \rangle \times \langle -1, 3 \rangle$ (tj. $x \in \langle -1, 3 \rangle$, $y \in \langle -1, 3 \rangle$). Nejprve si sestavíme síť bodů, ve kterých pak znázorníme příslušné směrnice. Chceme, aby x -ové i y -ové souřadnice postupovaly od -1 do 3, např. s krokem 0,5:

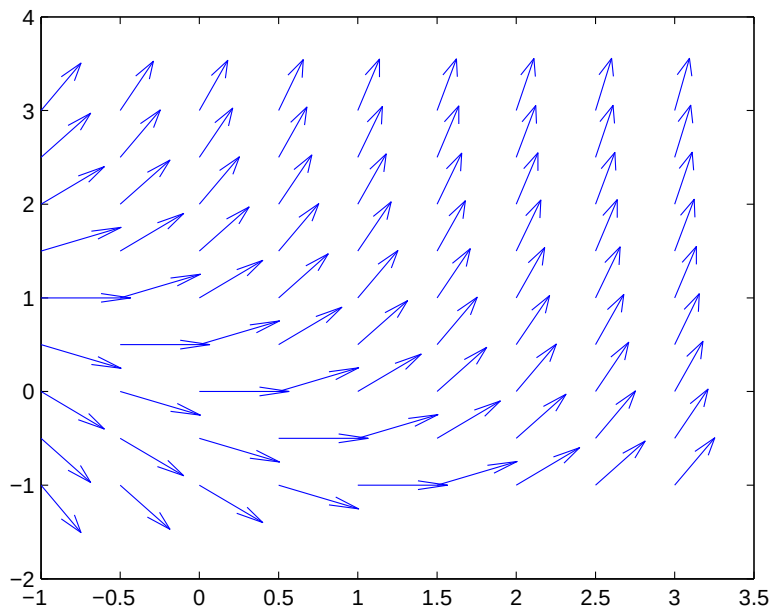
```
x=-1:0.5:3;
y=-1:0.5:3;
```

Tímto jsme zatím zadali dva vektory. My však potřebujeme rovinnou síť bodů – dvě matice, ve kterých budou x -ové i y -ové souřadnice příslušných bodů. K tomu použijeme funkci `meshgrid`:

```
[xx,yy] = meshgrid(x,y)
xx =
-1.0000    -0.5000         0    ...
-1.0000    -0.5000         0    ...
-1.0000    -0.5000         0    ...
...         ...         ...    ...
yy =
-1.0000    -1.0000    -1.0000    ...
-0.5000    -0.5000    -0.5000    ...
         0         0         0    ...
...         ...         ...    ...
```

V každém ze získaných bodů spočítáme směrnici – hodnotu funkce $f(x, y) = x + y$ (viz rovnice (2.2)). Pak bychom chtěli kreslit vektory v příslušných směrech. Každý takový vektor bude tvaru $\vec{w} = (u, v) = (1, f(x, y))$. Proč je první souřadnice 1? Uvědomme si, že tangens úhlu, který takovýto vektor svírá s osou x , je $\tan \varphi = v/u = f(x, y)$. Neboli směrnice vektoru bude opravdu $f(x, y)$. Spočítáme si souřadnice vektorů u a v a pak ještě vektory znormujeme, aby měly při kreslení všechny stejnou velikost. Pak už zbývá jen použít funkci `quiver`:

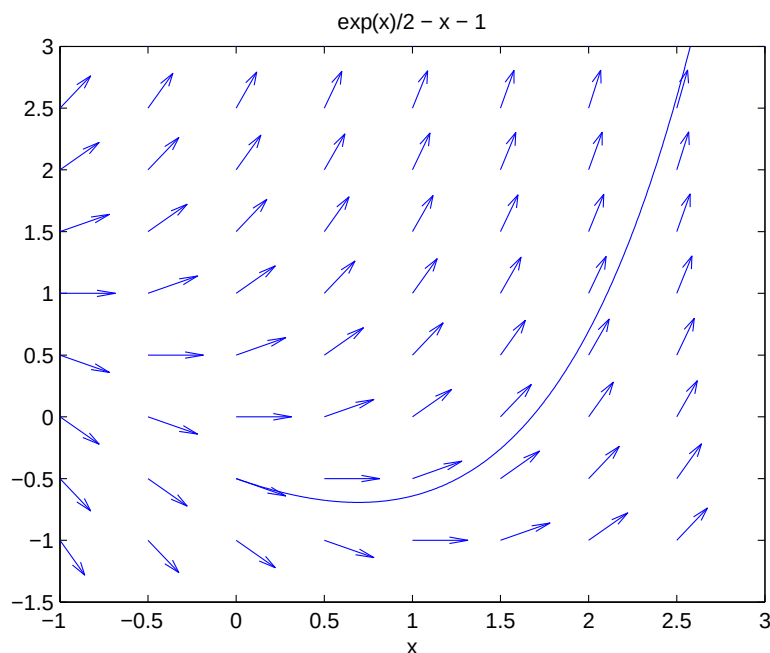
```
f=inline('x+y','x','y'); % definujeme funkci proměnných x a y
u=ones(size(xx)); % matice ze samých jedniček stejné velikosti jako xx
v=f(xx,yy);
norma=sqrt(u.^2+v.^2);
u0=u./norma;
v0=v./norma;
quiver(xx,yy,u0,v0)
```



Obrázek bude vypadat lépe, jestliže šipky necháme nakreslit menší. Necháme si směrové pole nakreslit znovu a spolu s ním i řešení rovnice, které splňuje počáteční podmínku $y(0) = -1/2$:

```
quiver(xx,yy,u0,v0,0.5)
hold on % kresli dal do stejneho obrazku
res = dsolve('Dy = x+y','y(0) = -0.5','x')
ezplot(res,[0,3])
axis([-1 3 -1.5 3]) % nastaveni rozsahu os

res =
exp(x)/2 - x - 1
```



Celý program pro kreslení směrového pole spolu s jedním řešením najdete v souboru `smerove_pole.m`.

2.4 Numerické řešení počáteční úlohy

Jestliže si MATLAB neporadí s analytickým řešením diferenciální rovnice, můžeme rovnici nechat vyřešit numericky. V tomto případě vždy hledáme řešení počáteční úlohy

$$y' = f(t, y), \quad y(t_0) = y_0.$$

Pro numerické řešení diferenciálních rovnic existuje v MATLABu více funkcí. Protože však numerické řešení není hlavním obsahem předmětu MDRE, předvedeme zde pouze jednu, a to

`[T,Y] = ode45(funkce, [t0, tn], y0)`

`funkce` je funkce, která je na pravé straně rovnice. Může být zadána různými způsoby, viz níže.

`[t0, tn]` je interval, na kterém hledáme řešení. `t0` je bod, ve kterém je zadána počáteční podmínka.

`y0` je předepsaná hodnota řešení v bodě `t0`.

Ve vektoru `T` jsou pak uloženy hodnoty času `t`, ve kterých byly vypočteny přibližné hodnoty řešení a v odpovídajících složkách vektoru `Y` je pak toto řešení.

Ve funkci `ode45` se používá kombinace metody Runge-Kutta 4. a 5. řádu (stručný popis metod Runge-Kutta najdete např. ve skriptech z BMA3). Délka kroku je volena automaticky, a to tak, aby chyba nepřesáhla určitou hodnotu, kterou si uživatel může nastavit.

Příklad 2.5 *Numericky najděte řešení počáteční úlohy*

$$y' = -2ty + t, \quad y(0) = 1$$

na intervalu $\langle 0, 3 \rangle$. (Jedná se o tutéž rovnici, kterou jsme analyticky řešili v příkladu 2.2, jen nezávislá proměnná se teď jmenuje t .)

Řešení: Nejprve si vytvoříme soubor s funkcí, která je na pravé straně řešení rovnice, tj. $f(t, y) = -2ty + t$. Výsledný soubor (M-file) pojmenujeme `f.m` a bude vypadat takto:

```
function dy = f(t,y)
dy = -2*t*y+t;
```

Nyní použijeme funkci `ode45`:

```
[t,y]=ode45('f',[0,3],1)

t =
    0
    0.0750
    0.1500
    0.2250
    ⋮
y =
    1.0000
    0.9972
    0.9889
    0.9753
    ⋮
```

Jméno funkce jsme zadali jako řetězec. Jiná možnost je použít tzv. handle funkce:

```
[t,y]=ode45(@f,[0,3],1)
```

Jestliže nechceme vytvářet soubor s funkcí, máme ještě možnost využít tzv. anonymní funkci:

```
f = @(t,y) -2*t*y+t;
[t,y]=ode45(f,[0,3],1)
```

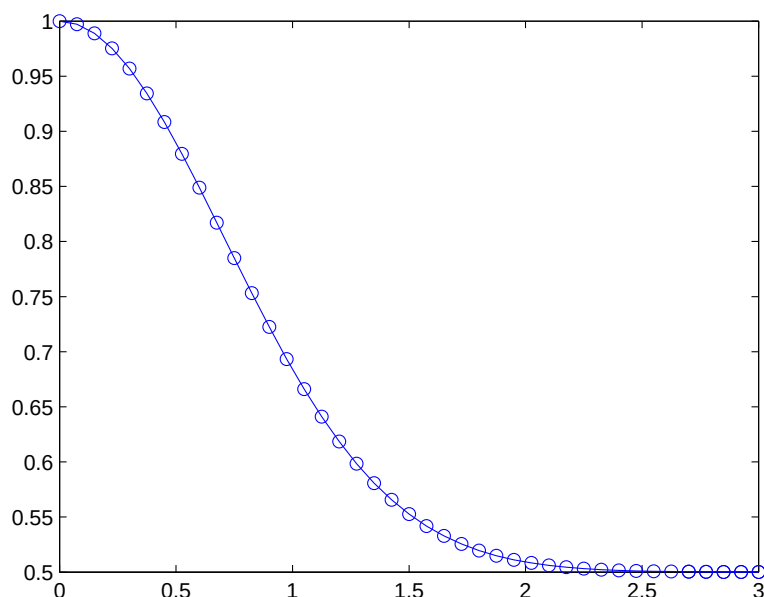
Ještě další možnost je inline funkce:

```
f = inline('-2*t.*y+t','t','y');
[t,y]=ode45(f,[0,3],1)
```

Jestliže nezadáme žádné výstupní argumenty a napíšeme pouze

```
ode45(f,[0,3],1)
```

zobrazí se graf nalezeného řešení:



2.5 Exaktní rovnice a integrační faktor

Řešíme-li exaktní rovnice pomocí MATLABu, je potřeba rovnici nejprve převést z tvaru obsahujícího diferenciály dx , dy do tvaru s y' , abychom mohli použít funkci `dsolve`. Protože se však MATLAB snaží vyjádřit řešení explicitně, můžeme dostat výsledek v odlišném tvaru než při řešení „ručním“ pomocí kmenové funkce nebo se řešení nemusí podařit najít vůbec.

Příklad 2.6 Najděte obecné řešení rovnice

$$2xy \, dx + (x^2 - 1) \, dy = 0.$$

(Jedná se o rovnici z příkladu MDRE-2.3.)

Řešení: Vydělíme-li rovnici dx , dostaneme

$$2xy + (x^2 - 1)y' = 0.$$

Pro řešení této rovnice použijeme funkce `dsolve`:

```
dsolve('2*x*y+(x^2-1)*Dy =0', 'x')
```

```
ans =  
C17/(x^2 - 1)
```

V příkladu MDRE-2.3 jsme řešení dostali vyjádřené implicitně rovnicí $x^2y - y = c$, zatímco MATLAB dal řešení explicitní. Zde snadno vidíme, že jde o totéž řešení. V jiných případech však řešení explicitně vyjádřit nelze.

Příklad 2.7 Najděte obecné řešení rovnice

$$(x + y)dx + (x + e^y)dy = 0.$$

Řešení: V tomto případě nás MATLAB zklame:

```
dsolve('(x+y)+(x+exp(y))*Dy = 0','x')
```

```
Warning: Explicit solution could not be found.
```

```
ans =
```

```
[ empty sym ]
```

Budeme tedy rovnici vyřešit „ručně“, s tím, že s některými kroky nám MATLAB pomůže. Nejprve ověříme, že se jedná o rovnici exaktní. V našem případě je $M(x, y) = x + y$, $N(x, y) = x + e^y$. Zadáme tyto funkce:

```
syms x y;
```

```
M = x+y;
```

```
N = x+exp(y);
```

Nyní vypočteme $\partial M/\partial y$ a $\partial N/\partial x$:

```
dMdy = diff(M,y)
```

```
dNdx = diff(N,x)
```

```
dMdy =
```

```
1
```

```
dNdx =
```

```
1
```

Na první pohled vidíme, že $\partial M/\partial y = \partial N/\partial x$. Kdyby však derivace byly složitější, můžeme se o této rovnosti přesvědčit MATLABem. Necháme zjednodušit výraz $\partial M/\partial y - \partial N/\partial x$. Pokud jsou si derivace rovny, musí vyjít 0:

```
simplify(dMdy - dNdx)
```

```
ans =
```

```
0
```

Nyní začneme hledat kmenovou funkci f . Nejprve zintegrujeme M podle x :

```
f = int(M,x)
```

```
f =
```

```
(x + y)^2/2
```

Na výsledek možná teď hledíme trochu zmateně, očekávali jsme přece $f(x, y) = x^2/2 + xy$. Když se chvíli zamyslíme, uvědomíme si, že derivace funkce $(x + y)^2/2$ podle x je skutečně $x + y$, takže výsledek integrace je správný, i když poněkud netradiční. Všimněte si také, že MATLAB (stejně jako Maple) vynechává ve výsledcích integrační konstantu, ono „+c“ na závěr. Zde se navíc nejedná o konstantu, ale o funkci proměnné y , kterou jsme značili jako $g(y)$ a kterou ještě musíme najít. To provedeme na základě vztahu MDRE-2.4:

```
g = int( N - diff(f,y) ,y)
```

```
g =
exp(y) - y^2/2
```

Nalezenou funkci g teď přičteme k f a výsledek necháme zjednodušit:

```
f = simplify(f+g)
f =
x^2/2 + y*x + exp(y)
```

Obecné řešení je tedy $x^2/2 + yx + e^y = c$.

Celý postup řešení najdete v souboru `exaktni_rovnice.m`.

2.5.1 Integrační faktor

Nyní předvedeme, jak lze pomocí MATLABu najít integrační faktor a pomocí něj převést rovnici na exaktní.

Příklad 2.8 *Najděte obecné řešení rovnice*

$$(2xy^2 - y) dx + (y^2 + x + y) dy = 0.$$

(Jedná se o rovnici z příkladu MDRE-2.6.)

Řešení: S touto rovnicí si MATLAB poradí, i když výsledek nezapiše zrovna tak, jak jsme zvyklí:

```
dsolve('(2*x*y^2-y)+(y^2+x+y)*Dy = 0','x')
ans =
0
solve(y*(x^2 + C35 + y + log(y)) = x, y)
```

Máme tedy dvě řešení: $y = 0$ a řešení dané implicitně rovnicí $y(x^2 + c + y + \ln y) = x$ (porovnejte s řešením získaným v příkladu MDRE-2.6).

Ukážeme si však ještě i to, jak se dají pomocí MATLABu jednotlivé kroky výpočtu. Zadáme funkce M a N a podíváme se, zda $\alpha = (\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x})/N$ závisí pouze na proměnné x :

```
syms x y;
M = 2*x*y^2-y;
N = y^2+x+y;
dMdy = diff(M,y)
dNdx = diff(N,x)
alfa = simplify((dMdy - dNdx)/N)
```

```
dMdy =
4*x*y - 1
dNdx =
1
alfa =
(2*(2*x*y - 1))/(y^2 + y + x)
```

Ve výrazu α se vyskytují obě proměnné, takže tato možnost nevyšla. Prozkoumáme, zda $\beta = (\frac{\partial N}{\partial x} - \frac{\partial M}{\partial y})/M$ závisí pouze na proměnné y :

```
beta = simplify((dNdx-dMdy)/M)
beta =
-2/y
```

Tentokrát máme úspěch. Pro zjištění, na jakých proměnných závisí funkce, lze také použít funkci `findsym`, např.

```
findsym(alfa)
ans =
x,y
```

Nyní vypočítáme integrační faktor $\mu(y) = e^{\int \beta(y) dy}$:

```
mu = exp(int(beta,y))
mu =
1/y^2
```

a nalezeným integračním faktorem rovnici vynásobíme. Tím dostaneme nové M a N , pro které už rovnice bude exaktní, a rovnici dál budeme řešit stejně jako v předchozím příkladu.

```
M = M*mu
N = N*mu
f = int(M,x)
g = int(N-diff(f,y),y)
f = simplify(f+g)
M =
-(y - 2*x*y^2)/y^2
N =
(y^2 + y + x)/y^2
f =
x^2 - x/y
g =
y + log(y)
f =
y + log(y) - x/y + x^2
```

Obecné řešení je tedy $y + \ln y - x/y + x^2 = c$.

Celý postup najdete v souboru `int_faktor.m`.

3 Některé typy obyčejných diferenciálních rovnic prvního řádu

3.1 Bernoulliho rovnice

S přímým řešením Bernoulliho rovnic v MATLABu zpravidla nebývají problémy.

Příklad 3.1 Najděte obecné řešení rovnice

$$y' = -\frac{y}{x} + xy^2.$$

(Jedná se o rovnici z příkladu MDRE-3.1)

Řešení:

```
dsolve('Dy = -y/x+x*y^2','x')
```

```
ans =
```

$$0$$

$$1/(C2*x - x^2)$$

Všimněte si, že MATLAB našel i singulární řešení $y = 0$. Obecné řešení $y = 1/(cx - x^2)$ vypadá sice trochu jinak než řešení nalezené v příkladu MDRE-3.1, tj. $y = -1/(x^2 + kx)$, ale snadno se přesvědčíme, že řešení vypočtené MATLABem dostaneme z tohoto řešení volbou $k = -c$ a jednoduchou úpravou.

Chceme-li si pomoci MATLABu spíš kontrolovat postup řešení vypočteného na papíře, je to možné. Některé kroky však stejně budeme muset provést „ručně“. MATLABem můžeme najít řešení příslušné homogenní rovnice $y' = -y/x$ – to vyjde $y_0(x) = c/x$. Dále pak najdeme funkci $C(x)$ jako řešení diferenciální rovnice $C'(x) = b(x) \cdot C^m(x) \cdot y_0^{n-1}(x)$, tj. pro náš příklad $C'(x) = x \cdot C^2(x) \cdot (1/x)$:

```
dsolve('DC = C^2','x')
```

```
ans =
```

$$0$$

$$-1/(C9 + x)$$

Máme $C(x) = -1/(c + x)$ a obecné řešení je $y = C(x) \cdot y_0(x) = -1/(cx + x^2)$.

V MATLABu by se sice dal napsat program, který by pro zadanou Bernoulliho rovnici $y' = a(x)y + b(x)y^n$ příslušnou homogenní rovnici a následně rovnici pro neznámou funkci $C(x)$ sestavil a vyřešil automaticky, ale nebylo by to tak úplně jednoduché. Tento úkol ponecháme zájemcům jako námět na hraní.

3.2 Riccatiova rovnice

Příklad 3.2 Najděte obecné řešení rovnice

$$y' = -\frac{4}{x^2} - \frac{1}{x}y + y^2 \quad (3.1)$$

(viz příklad MDRE-3.2).

Řešení:

```
dsolve('Dy = -4/x^2-1/x*y+y^2','x')
```

```
ans =
```

$$\frac{-2/x}{1/(C12*x^5 + x/4) - 2/x}$$

MATLAB opět našel i singulární řešení.

Jestliže u Riccatiho rovnice chceme používat MATLAB jen jako pomocníka pro dílčí výpočty, začneme tím, že ověříme, jestli jsme správně uhodli řešení y_1 . V příkladu MDRE-3.2 je psáno, že jedno řešení je $y_1 = 2/x$. Ověříme pomocí MATLABu, že se skutečně jedná o řešení rovnice (3.1)– dosadíme do levé a pravé strany:

```
syms x y y1
prava_strana = -4/x^2-y/x+y^2;
y1 = 2/x;
P = subs(prava_strana,y,y1);
L = diff(y1,x);
simplify(L-P)
```

```
ans =
```

```
0
```

Zjistili jsme, že rozdíl levé a pravé strany je nulový, a $y_1 = 2/x$ je tudíž opravdu řešení zadané rovnice. Nyní zavedeme substituci $y = y_1 + u$. Necháme si dosadit do pravé strany rovnice (3.1) a od výsledku pak odečteme y_1' . Tím dostaneme pravou stranu nové rovnice s neznámou funkcí u :

```
syms u
prava_dosad = subs( prava_strana , y , y1+u );
nova_prava = simplify( prava_dosad - diff(y1,x) )

nova_prava =
(3*u)/x + u^2
```

Vzniklou Bernoulliho rovnici $u' = 3u/x + u^2$ už můžeme řešit podobně jako v příkladu 3.1.

3.3 Clairautova rovnice

Příklad 3.3 *Najděte řešení rovnice*

$$y = xy' + \frac{1}{2}(y')^2$$

(viz příklad MDRE-3.3).

Řešení:

```
reseni = dsolve('y = x*Dy + 1/2*(Dy)^2', 'x')
```

```
reseni =  
      -x^2/2  
      C21^2/2 + x*C21
```

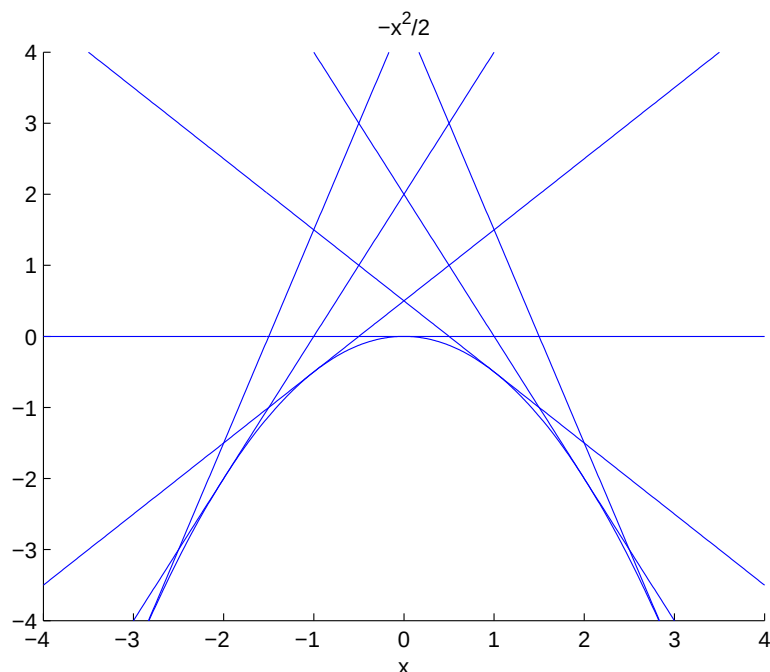
Tentokrát nebudeme předvádět žádné pomocné výpočty, ale ukážeme si, jak lze snadno nakreslit obrázek několika vybraných řešení spolu s řešením singulárním. Budeme kreslit grafy funkcí $y = c^2/2 + cx$ pro různé hodnoty konstanty c . Nejprve do druhé složky **reseni** dosadíme za **C21** čísla -3, -2, ..., 3. Funkci **subs** můžeme použít i pro vektor dosazovaných hodnot, výsledkem bude opět vektor.

```
ruzna_reseni = subs(reseni(2), 'C21', -3:3)
```

```
ruzna_reseni =  
      [ 9/2 - 3*x, 2 - 2*x, 1/2 - x, 0, x + 1/2, 2*x + 2, 3*x + 9/2]
```

Nyní do jednoho obrázku budeme postupně kreslit grafy těchto funkcí na intervalu $\langle -4, 4 \rangle$ a na závěr nakreslíme graf singulárního řešení $y = x^2/2$, které je uloženo v první složce **reseni**. Rozmezí os nastavíme v x i v y od -4 do 4.

```
hold on  
for i=1:length(C21)  
    ezplot(ruzna_reseni(i), [-4 4]);  
end  
ezplot(reseni(1), [-4 4]);  
axis([-4 4 -4 4])
```



3.4 Picardova metoda postupných aproximací

Příklad 3.4 *Picardovou metodou postupných aproximací řešte počáteční problém*

$$y' = y - 1, \quad y(0) = 2$$

(viz příklad MDRE-3.4).

Řešení: Budeme postupovat jako při výpočtu na papíře, krok za krokem. Nejprve definujeme symbolické proměnné, funkci f a počáteční podmínky:

```
syms x y t
f = y-1;
x0 = 0;
y0 = 2;
```

Tedy vypočteme y_1 jako $y_0 + \int_{x_0}^x f(t, y_0(t)) dt$ (viz MDRE-(3.11)):

```
y1 = y0 + int ( subs(f,[x,y],[t,y0]), t, x0,x)

y1 =
x+2
```

Dál pokračujeme podobně. Pro přehlednost nejprve do y_1 dosadíme t za x . Pak vypočítáme y_2 :

```
y1t = subs(y1,x,t);
y2 = y0 + int ( subs(f,[x,y],[t,y1t]), t, x0,x)
```

```
y2 =
(x*(x + 2))/2 + 2
```

Tento výsledek můžeme upravit (roznásobit) pomocí funkce `expand`:

```
y2 = expand(y2)

y2 =
x^2/2 + x + 2
```

Podobně bychom mohli postupovat dál. Pro šikovnějšího čtenáře jistě nebude problém program upravit tak, aby se automaticky spočítal předem daný počet postupných aproximací. V příkladu 3.4 jsme ukázali, že

$$y_n(x) = 1 + \sum_{k=0}^n \frac{x^k}{k!}.$$

Řešení bude tedy $y(x) = 1 + \sum_{k=0}^{\infty} \frac{x^k}{k!}$. Nekonečnou řadu můžeme sečíst v MATLABu pomocí funkce `symsum`:

```
soucet = symsum( výraz, proměnná, odkud, kam )
```

V našem případě tedy takto (nejprve ještě definujeme symbolickou proměnnou `k`, její faktoriál v sumě pak definujeme jako `sym('k!')`):

```
syms k
reseni = 1+symsum(x^k/sym('k!'),k,0,Inf)

reseni =
exp(x)+1
```

Pro oživení ještě vytvoříme animaci, ve které se nejprve vykreslí graf řešení spolu s y_1 a pak spolu s y_2 .

```
ezplot(y1,[0,2])
hold on
ezplot(res,[0,2])
obr(1) = getframe % první snímek
hold off

ezplot(y2,[0,2])
hold on
ezplot(res,[0,2])
obr(2) = getframe % druhý snímek
hold off
```

Obrázky kreslíme obvyklým způsobem. Pomocí funkce `getframe` si vždy nakreslený obrázek zachytíme. Celou animaci pak spustíme pomocí funkce `movie`:

`movie(obr)`

Protože se naše animace skládá jen ze dvou snímků, může se stát, že si při jejím spuštění ani nevšimneme, že se něco děje. Proto animaci raději zpomalíme, použijeme funkci `movie` s více parametry:

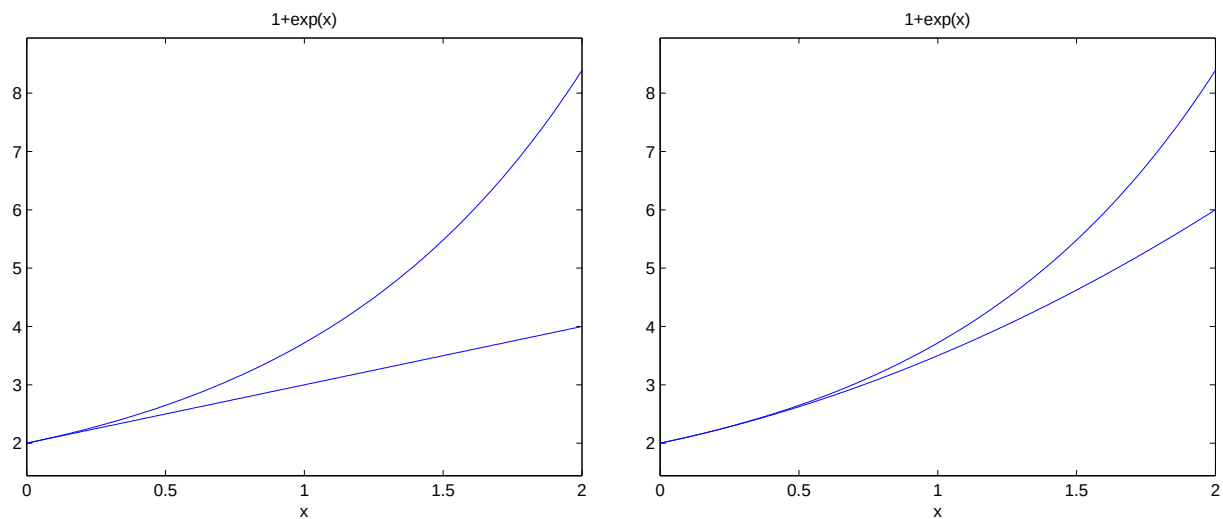
`movie( M , n , fps )`

kde `M` je matice snímků, `n` je počet opakování a `fps` je počet snímků za sekundu (frames per second).

Chceme-li naši animaci přehrát jednou, rychlostí 2 snímky za sekundu, napíšeme

`movie(obr, 1, 2)`

Animace se bude skládat z těchto dvou snímků:



Právě popsaný program naleznete vcelku v souboru `picard.m`

4 Lineární systémy obyčejných diferenciálních rovnic

4.1 Lineární homogenní rovnice n -tého řádu s konstantními koeficienty

Příklad 4.1 Najděte obecné řešení diferenciální rovnice

$$y^{(5)} + y^{(4)} + 2y''' + 2y'' + y' + y = 0$$

(viz příklad z kapitoly MDRE-4.3).

Řešení: Pro přímé řešení rovnice opět použijeme funkci `dsolve`. Derivace vyšších řádů zadáváme pomocí symbolu `D` následovaného číslem, které udává řád derivace. Např. y'' zapisujeme jako `D2y`. Pro naši rovnici tedy napíšeme

```
dsolve('D5y+D4y+2*D3y+2*D2y+Dy+y=0','x')
ans =
C1*exp(-x)+C2*sin(x)+C3*cos(x)+C4*sin(x)*x+C5*cos(x)*x
```

Jestliže chceme pomocí MATLABu najít kořeny charakteristické rovnice

$$\lambda^5 + \lambda^4 + 2\lambda^3 + 2\lambda + 1 = 0,$$

můžeme použít funkci `solve` ze symbolického toolboxu:

```
solve('1^5+1^4+2*1^3+2*1^2+1+1=0','1')
ans =
-1
sqrt(-1)
-sqrt(-1)
sqrt(-1)
-sqrt(-1)
```

Neboli reálný kořen je -1 a pak ještě máme kořeny $\pm i$, každý z nich násobnosti 2.

Jiná možnost je nechat kořeny vypočítat numericky pomocí funkce `roots`:

```
kořeny = roots( polynom )
```

přičemž `polynom` je zadán jako vektor tvořený koeficienty polynomu. Např. polynom $P(x) = x^2 + 2x - 3$ zadáme jako `P = [1 2 -3]`. Pro naši rovnici dostaneme

```
roots([1 1 2 2 1 1])
ans =
-1.0000
0.0000 + 1.0000i
0.0000 - 1.0000i
-0.0000 + 1.0000i
-0.0000 - 1.0000i
```

4.2 Řešení systémů diferenciálních rovnic

Příklad 4.2 Najděte obecné řešení systému

$$\begin{aligned} y_1' &= 2y_1 + y_2, \\ y_2' &= -3y_1 + 6y_2 \end{aligned}$$

(viz příklad MDRE-4.18).

Pak najděte řešení vyhovující počátečním podmínkám $y_1(0) = -1$, $y_2(0) = 2$.

Řešení: Funkci `dsolve` můžeme použít i pro systém diferenciálních rovnic. Rovnice zadáme postupně jako argumenty funkce `dsolve`. Pro přehlednost si rovnice uložíme nejprve do proměnných `rovnice1` a `rovnice2`:

```
rovnice1 = 'Dy1 = 2*y1 + y2';
rovnice2 = 'Dy2 = -3*y1 + 6*y2';
reseni = dsolve(rovnice1, rovnice2, 'x')

reseni =
      y1: [1x1 sym]
      y2: [1x1 sym]
```

Řešení jsme tentokrát dostali jako strukturu o dvou složkách `y1` a `y2`. Na tyto složky se dostaneme takto:

```
reseni.y1

ans =
C1*exp(3*x)+C2*exp(5*x)
```

```
reseni.y2

ans =
C1*exp(3*x)+3*C2*exp(5*x)
```

Pro nalezení řešení vyhovujícího počátečním podmínkám zadáme tyto podmínky jako další argumenty funkce `dsolve`:

```
part_reseni = dsolve(rovnice1, rovnice2, 'y1(0)=-1', 'y2(0)=2', 'x');
part_reseni.y1
part_reseni.y2

ans =
-5/2*exp(3*x)+3/2*exp(5*x)
ans =
-5/2*exp(3*x)+9/2*exp(5*x)
```

Poznámka. Ve funkci `dsolve` v MATLABu se bohužel musejí všechny rovnice systému zadávat jednotlivě a jako řetězce. Nemůžeme systém zapsat v celku stylem $Dy = A*y$, kde A by byla matice, kterou jsme předem zadali. Můžeme si však ze zadané matice systém vyrobit, i když je to možná poněkud krkolomné. Předvedeme to na soustavě, kterou jsme právě řešili.

```
syms y1 y2
A = [2 1; -3 6];
Ay = A*[y1;y2]
rovnice1 = ['Dy1 = ',char(Ay(1))]
rovnice2 = ['Dy2 = ',char(Ay(2))]

Ay =
    2*y1+y2
   -3*y1+6*y2
rovnice1 =
Dy1 = 2*y1+y2
rovnice2 =
Dy2 = -3*y1+6*y2
```

Tímto jsme si poskládali řetězce `rovnice1` a `rovnice2`, které pak můžeme zadat jako argumenty funkci `dsolve`. Ještě dlužíme vysvětlení, jak se v MATLABu řetězce skládají. Stačí řetězce, které chceme spojit, zapsat do hranatých závorek za sebe, např. `['abc', 'def']` dá výsledek `'abcdef'`. Použitá funkce `char` konvertuje symbolický výraz na textový řetězec.

4.2.1 Numerické řešení systémů diferenciálních rovnic

Pro numerické řešení systémů diferenciálních rovnic slouží v MATLABu stejné funkce jako pro řešení jediné rovnice. Hledané funkce musí tvořit sloupcový vektor. Opět si musíme nejprve vytvořit funkci, která je na pravé straně řešeného systému.

Příklad 4.3 *Najděte přibližné řešení počáteční úlohy*

$$\begin{aligned} y_1' &= 2y_1 + y_2 + t, & y_1(0) &= -1, \\ y_2' &= -3y_1 + 6y_2 + 1, & y_2(0) &= 2. \end{aligned}$$

Řešení: Napíšeme si funkci pro výpočet hodnoty pravé strany a uložíme ji do souboru `fce.m`:

```
function dy = fce(t,y)
A = [2 1; -3 6];
b = [t; 1];
dy = A*y + b;
```

Dále použijeme funkci `ode45` (popis jejích parametrů viz příklad 2.5). Počáteční hodnota řešení je zadána jako sloupec.

```
[t,y] = ode45(@fce,[0,1],[-1;2])
```

```
t =
```

```
0
0.0067
0.0134
⋮
```

```
y =
```

```
-1.0000  2.0000
-0.9997  2.1025
-0.9986  2.2092
⋮        ⋮
```

4.2.2 Řešení systémů diferenciálních rovnic pomocí exponenciály matice

Příklad 4.4 Najděte obecné řešení systému

$$\begin{aligned} y_1' &= 2y_1 + y_2, \\ y_2' &= -3y_1 + 6y_2 \end{aligned}$$

pomocí exponenciály matice (viz příklad MDRE-4.18).

Řešení: V symbolickém toolboxu MATLABu existuje přímo funkce `expm` pro nalezení exponenciály matice:

```
syms t c1 c2
A = [2 1; -3 6];
eA = expm(t*A)
reseni = eA*[c1;c2]

eA =
[ (3*exp(3*t))/2 - exp(5*t)/2, exp(5*t)/2 - exp(3*t)/2]
[ (3*exp(3*t))/2 - (3*exp(5*t))/2, (3*exp(5*t))/2 - exp(3*t)/2]
reseni =
c1*((3*exp(3*t))/2 - exp(5*t)/2) - c2*(exp(3*t)/2 - exp(5*t)/2)
c1*((3*exp(3*t))/2 - (3*exp(5*t))/2) - c2*(exp(3*t)/2 - (3*exp(5*t))/2)
```

Právě předvedené příkazy najdete též v souboru `exp_mat.m`.

Jestliže si spíš chceme kontrolovat „ruční“ výpočet, s některými kroky nám MATLAB také může pomoci. Vrátime se tedy zpátky na začátek a budeme exponenciálu hledat postupně. Nejprve sestavíme charakteristický polynom matice (předpokládáme, že matici A už máme zadanou, stejně tak symbolickou proměnnou t). K tomu slouží funkce `poly`. Jestliže je matice A zadána „obyčejně“, výsledkem budou koeficienty charakteristického polynomu:

```
p = poly(A)
```

```
p =
    1    -8    15
```

Kdybychom matici zadali symbolicky, dostali bychom výsledek zapsaný opravdu jako polynom:

```
A = sym(A);
p = poly(A)
```

```
p =
x^2-8*x+15
```

Charakteristický polynom je tedy

$$p(\lambda) = \lambda^2 - 8\lambda + 15.$$

Budeme hledat řešení diferenciální rovnice $y'' - 8y' + 15y = 0$ s počátečními podmínkami $y(0) = 0$, $y'(0) = 1$:

```
y = dsolve('D2y - 8*Dy + 15*y = 0', 'y(0)=0', 'Dy(0)=1', 't')
```

```
y =
exp(5*t)/2 - exp(3*t)/2
```

Mohli bychom program upravit tak, aby se příslušná diferenciální rovnice sestavila automaticky z koeficientů polynomu p , postupovalo by se podobně jako v poznámce za příkladem 4.2, ponecháme to však čtenářům jako cvičení. Nyní vypočítáme vektor z :

```
z = [ -8 1; 1 0]*[y;diff(y,t)]
```

```
z =
(5*exp(3*t))/2 - (3*exp(5*t))/2
exp(5*t)/2 - exp(3*t)/2
```

Teď již můžeme sestavit exponenciálu matice. Poznamenejme, že funkce `eye(n)` sestaví jednotkovou matici o n řádcích a n sloupcích.

```
eA = z(1)*eye(2)+ z(2)*A
```

```
eA =
[ (3*exp(3*t))/2 - exp(5*t)/2, exp(5*t)/2 - exp(3*t)/2
  (3*exp(3*t))/2 - (3*exp(5*t))/2, (3*exp(5*t))/2 - exp(3*t)/2]
```

Celý program pro výpočet exponenciály matice krok za krokem najdete v souboru `exp_mat_kroky.m`.

5 Soustavy lineárních diferenciálních rovnic s konstantními koeficienty - řešení na bázi zobecněných vlastních vektorů

Pro začátek poznamenejme, že všechny rovnice řešené v této kapitole by samozřejmě bylo možné vyřešit jediným použitím funkce `dsolve`. Zde je však cílem ukázat, jak lze pomocí MATLABu provádět jednotlivé kroky „ručního“ výpočtu.

Při řešení systému diferenciálních rovnic

$$x' = Ax, \quad (5.1)$$

metodou vlastních čísel a vlastních vektorů potřebujeme vlastní čísla matice A a jim odpovídající vlastní vektory najít. K tomu slouží v MATLABu funkce `eig`:

```
lambda = eig(A)
```

Takto získáme pouze vlastní čísla, která budou uložena ve sloupcovém vektoru `lambda`. Jestliže zadáme dva výstupní argumenty (pozor na jejich pořadí):

```
[V,D] = eig(A)
```

dostaneme vlastní čísla i vlastní vektory. Vlastní vektory tvoří sloupce matice V a jsou normované, tj. upravené tak, aby jejich velikost byla rovna 1.

Vlastní čísla tvoří diagonálu matice D . Tato matice je tzv. kanonický tvar matice A .

Poznamenejme, že vlastní čísla a vlastní vektory MATLAB hledá numericky.

Chceme-li vlastní čísla a vlastní vektory najít symbolicky, zadáme matici A jako symbolickou proměnnou. V tomto případě vlastní vektory normovány nejsou.

Příklad 5.1 *Metodou vlastních čísel a vlastních vektorů určete obecné řešení soustavy rovnic:*

$$\begin{aligned} x_1' &= -2x_1 - 2x_2, \\ x_2' &= -3x_1 - x_2. \end{aligned}$$

Řešení: Najdeme vlastní čísla a jim odpovídající vlastní vektory. Můžeme porovnat řešení numerické a symbolické. Nejprve numericky:

```
A = [-2 -2; -3 -1];
```

```
[V,D] = eig(A)
```

```
V =
```

```
 -0.7071    0.5547
 -0.7071   -0.8321
```

```
D =
```

```
 -4    0
  0    1
```

Vlastní čísla jsou tedy $\lambda_1 = -4$ a $\lambda_2 = 1$. Vlastní vektory jsou

$$v_1 \doteq \begin{pmatrix} -0,7071 \\ -0,7071 \end{pmatrix}, \quad v_2 \doteq \begin{pmatrix} 0,5547 \\ -0,8321 \end{pmatrix}.$$

Obecné řešení je

$$x(t) = C_1 v_1 e^{-4t} + C_2 v_2 e^t.$$

Nyní pro srovnání problém vyřešíme symbolicky:

```
A = [-2 -2; -3 -1];
```

```
A = sym(A);
```

```
[V,D] = eig(A)
```

```
V =
```

```
[ 1, 1]
```

```
[-3/2, 1]
```

```
D =
```

```
[ 1, 0]
```

```
[ 0, -4]
```

Vlastní čísla jsou uvedena v jiném pořadí (což je jen náhoda), $\lambda_1 = 1$ a $\lambda_2 = -4$. Vlastní vektory jsou

$$v_1 = \begin{pmatrix} 1 \\ -3/2 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Symbolické řešení vypadá na pohled lépe. Pro větší matice s iracionálními vlastními čísly se však výpočet nakonec stejně provede numericky a výsledek už na pohled pěkně vypadat nebude.

Chceme-li MATLAB použít pro kontrolu jednotlivých kroků našeho „ručního“ výpočtu, můžeme si ještě dříve, než budeme počítat samotná vlastní čísla, nechat sestavit charakteristický polynom matice. K tomu slouží funkce `poly`, viz příklad 4.4.

Nyní si ukážeme příklad s komplexními vlastními čísly:

Příklad 5.2 *Určete obecné řešení soustavy rovnic:*

$$x'_1 = x_1 + 3x_2,$$

$$x'_2 = -3x_1 + x_2.$$

Řešení: Vypočítáme vlastní čísla a vlastní vektory:

```
A = sym([1 3; -3 1])
```

```
[V,D] = eig(A)
```

```
V =
```

```
[ 1, 1]
```

```
[ i, -i]
```

```
D =
```

```
[ 1+3*i, 0]
```

```
[ 0, 1-3*i]
```

Vidíme, že $\lambda_{1,2} = 1 \pm i$ a že vlastní vektor příslušný vlastnímu číslu $\lambda_1 = 1 + i$ je $v_1 = (1, i)^T$. Uložíme si tento vektor, tj. první sloupec matice A , do proměnné $v1$. Také si uložíme první vlastní číslo neboli prvek v prvním řádku a prvním sloupci matice D :

```
v1 = V(:,1)
lambda1 = D(1,1)

v1 =
    1
    i
lambda1 =
    1+3*i
```

Následně vypočítáme reálnou a imaginární část $v_1 \cdot e^{\lambda_1 t}$. Přitom musíme zadat, že proměnná t bude reálná:

```
syms t real
x1 = real(exp(lambda1*t)*v1)
x2 = imag(exp(lambda1*t)*v1)

x1 =
    -1/2*i*exp((1+3*i)*t)+1/2*i*exp((1-3*i)*t)
    1/2*exp((1+3*i)*t)+1/2*exp((1-3*i)*t)
x2 =
    -1/2*i*(-i*exp((1+3*i)*t)-i*exp((1-3*i)*t))
    -1/2*i*(exp((1+3*i)*t)-exp((1-3*i)*t))
```

Tento tvar výsledku je asi poněkud překvapivý. Vůbec se v něm nevyskytuje očekávaný sinus a kosinus. Chceme-li výsledek zjednodušit, použijeme funkci `simplify`:

```
x1 = simplify(x1)
x2 = simplify(x2)

x1 =
    sin(3*t)*exp(t)
    cos(3*t)*exp(t)
x2 =
    cos(3*t)*exp(t)
    -sin(3*t)*exp(t)
```

Poznamenejme, že starší verze MATLABu tohoto zjednodušení nebyly schopny (vyzkoušeno s verzí R2007b) a nechávaly výsledek ve tvaru s exponenciálami.

Obecné řešení je tedy

$$x(t) = c_1 \begin{pmatrix} e^t \sin 3t \\ e^t \cos 3t \end{pmatrix} + c_2 \begin{pmatrix} e^t \cos 3t \\ -e^t \sin 3t \end{pmatrix}.$$

Zatím jsme řešili příklady, kde vlastní čísla nebyla násobná. Zbývá nám probrat případ násobných vlastních čísel.

Příklad 5.3 *Určete obecné řešení soustavy rovnic:*

$$\begin{aligned}x'_1 &= x_1 + x_2 - x_3, \\x'_2 &= -x_1 + x_2 - x_3, \\x'_3 &= -x_2 + 2x_3.\end{aligned}$$

(Viz příklad MDRE-5.4.)

Řešení: Najdeme vlastní čísla a vlastní vektory matice soustavy:

```
A = sym([1 1 -1; -1 1 -1; 0 -1 2]);
[V,D] = eig(A)
```

```
V =
[ -1, -1]
[  0,  1]
[  1,  1]
D =
[ 2, 0, 0]
[ 0, 1, 0]
[ 0, 0, 1]
```

Vidíme, že matice má vlastní číslo $\lambda_1 = 2$, ke kterému přísluší vlastní vektor $v_1 = (-1, 0, 1)^T$ a dvojnásobné vlastní číslo $\lambda_2 = 1$, ke kterému přísluší vlastní vektor $v_2 = (-1, 1, 1)^T$. Musíme proto najít zobecněný vlastní vektor v_3 hodnoty 2, a to jako řešení soustavy $(A - \lambda_2 E)v_3 = v_2$. V MATLABu je pro řešení soustav lineárních rovnic určen operátor `\`. Ten je však primárně určen pro řešení soustav s regulární maticí (tj. s jednoznačně daným řešením). V našem příkladu je matice soustavy singulární (její řádky jsou lineárně závislé), a proto je použití operátoru `\` poněkud problematické – jako výsledek dostaneme jen jedno z nekonečně mnoha řešení a MATLAB nás na to upozorní:

```
lam2 = D(2,2);
v2 = V(:,2);
A2 = A-lam2*eye(3)
v3 = A2\v2

A2 =
[  0,  1, -1]
[ -1,  0, -1]
[  0, -1,  1]
```

Warning: System is rank deficient. Solution is not unique.

```
v3 =
-1
-1
0
```

Pro účely našeho konkrétního příkladu by tento výsledek postačil, protože hledáme pouze jeden zobecněný vlastní vektor, a ten jsme tímto našli. Pro úplnost ale ještě ukážeme, jak lze v MATLABu najít všechna řešení zadané soustavy, vyjádřená v závislosti na parametrech. Místo operátoru `\` musíme použít funkci `solve`. Ta ale očekává jednotlivé rovnice jako řetězce, nemůžeme jí, bohužel, zadat soustavu vcelku. Podnikavější čtenář se může pokusit sestavit příslušné řetězce pomocí programu. My je zde zadáme pro náš příklad.

```
rce1 = 'v32 - v33 = -1';
rce2 = '- v31 - v33 = 1';
rce3 = '- v32 + v33 = 1';
v3=solve(rce1,rce2,rce3,'v31','v32','v33')
```

```
v3 =
    v31: [1x1 sym]
    v32: [1x1 sym]
    v33: [1x1 sym]
```

Výsledkem funkce `solve` pro soustavu rovnic je struktura. Podíváme se na její jednotlivé složky:

```
v3 = [v3.v31; v3.v32; v3.v33]

v3 =
    - z - 1
     z - 1
      z
```

Vidíme, že $v_3 = (-z - 1, z - 1, z)^T$, kde $z \in \mathbb{R}$ je parametr, který si můžeme libovolně zvolit. Řešení získané pomocí operátoru `\` vzniklo volbou $z = 0$, mohli bychom však zvolit např. $z = 1$. Tím dostaneme vektor $v_3 = (-2, 0, 1)^T$ (viz „ruční“ řešení tohoto příkladu – MDRE-5.4). Zůstaneme-li u vektoru $v_3 = (-1, -1, 0)^T$, získáváme obecné řešení

$$x(t) = c_1 e^{2t} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} + c_2 e^t \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} + c_3 e^t \left(t \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} \right).$$

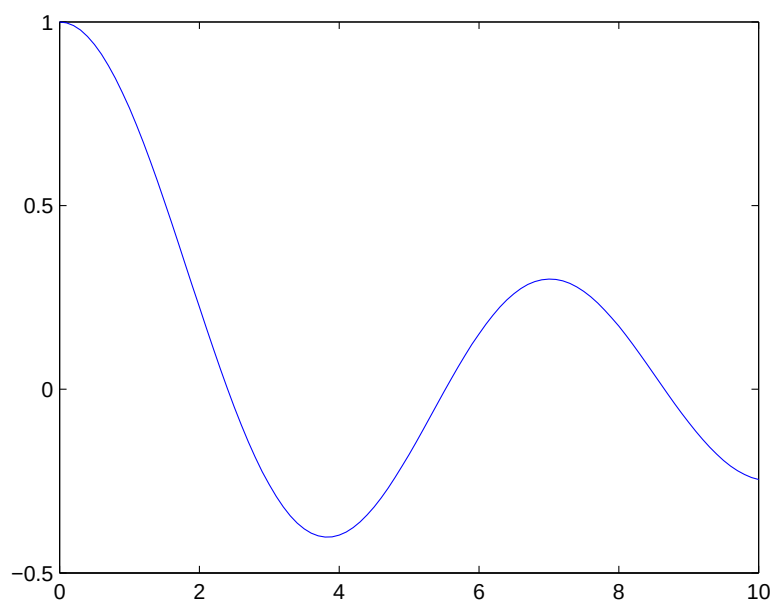
6 Besselova rovnice a Besselovy funkce

Besselovy funkce jsou standardní součástí MATLABu, pro jejich použití není potřeba mít symbolický toolbox. Besselova funkce 1. druhu, $J_\nu(x)$, se v MATLABu zapíše jako

```
J = besselj(nu,x)
```

Chceme-li např. nakreslit graf funkce J_0 na intervalu $\langle 0, 10 \rangle$, můžeme to udělat takto:

```
x = 0:0.1:10;
y = besselj(0,x);
plot(x,y)
```



Besselovu rovnici MATLAB také umí vyřešit (k tomu ovšem už symbolický toolbox potřebujeme).

Příklad 6.1 Najděte obecné řešení rovnice

$$x^2 y'' + xy' + \left(x^2 - \frac{1}{9}\right) y = 0.$$

Řešení:

```
rce = 'x^2*D2y + x*Dy + (x^2-1/9)*y = 0';
res = dsolve(rce,'x')

res =
C5*besselj(-1/3, x) + C6*bessely(-1/3, x)
```

Vidíme, že řešení je zapsané v jiném tvaru, než bychom očekávali – viz MDRE-7.13. Místo funkce $J_{1/3}$ (neboli `besselj(1/3,x)`) se zde vyskytuje jakési `bessely`. Jedná se o Besselovu funkci 2. druhu, $Y_\nu(x)$, která je také řešením Besselovy rovnice. V kapitole

skript o Besselově rovnici se probírá pouze případ, kdy ν není celé číslo. Tehdy jsou funkce J_ν a $J_{-\nu}$ lineárně nezávislé a obecné řešení můžeme vyjádřit pomocí nich. Jestliže však ν celé je, s Besselovými funkcemi 1. druhu bychom nevystačili. Druhé nezávislé řešení v tomto případě tvoří právě Besselova funkce 2. druhu.

Jestliže není ν celé číslo, platí mezi J_ν a Y_ν vztah

$$Y_\nu(x) = \frac{J_\nu(x) \cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}.$$

Jestliže ν je celé, $\nu = k \in \mathbb{Z}$, pak bychom Y_ν předchozím vztahem definovat nemohli, protože $\sin(k\pi) = 0$. V tomto případě je

$$Y_k(x) = \lim_{\nu \rightarrow k} Y_\nu(x).$$

Pro některé hodnoty ν se řešení Besselovy rovnice podaří vyjádřit i bez Besselových funkcí.

Příklad 6.2 Najděte obecné řešení rovnice

$$x^2 y'' + xy' + \left(x^2 - \frac{1}{4}\right)y = 0.$$

Řešení:

```
rce = 'x^2*D2y + x*Dy + (x^2-1/4)*y = 0';
res = dsolve(rce, 'x')
```

```
res =
(2^(1/2)*C2*cos(x))/(pi^(1/2)*x^(1/2)) + (2^(1/2)*C3*sin(x))/(pi^(1/2)*x^(1/2))
```

7 Stabilita řešení systémů diferenciálních rovnic

7.1 Stabilita lineárních systémů

Budeme zkoumat stabilitu triviálního řešení

$$x' = Ax, \quad (7.1)$$

kde A je konstantní reálná čtvercová matice typu (n, n) .

Podle věty MDRE-8.3 stačí najít vlastní čísla matice A a podívat se na jejich reálné složky.

Příklad 7.1 *Vyšetřete stabilitu systému*

$$\begin{aligned} x_1' &= -3x_1 - x_2 - x_4 \\ x_2' &= x_1 - 2x_2 \\ x_3' &= -x_2 - x_3 + x_4 \\ x_4' &= x_2 - 2x_4. \end{aligned}$$

Řešení: Zadáme matici soustavy a najdeme její vlastní čísla:

```
A = [-3 -1 0 -1; 1 -2 0 0; 0 -1 -1 1; 0 1 0 -2];
```

```
lambda = eig(A)
```

```
lambda =  
-1.0000  
-2.0000 + 1.0000i  
-2.0000 - 1.0000i  
-3.0000
```

Všechna vlastní čísla matice A mají záporné reálné části. Systém je tedy asymptoticky stabilní.

7.1.1 Hurwitzovo kritérium

Najdeme-li vlastní čísla matice, stabilitu určíme rychle. Kdybychom si spíš chtěli kontrolovat „ruční“ výpočet prováděný pomocí Hurwitzova kritéria, může nám být MATLAB také užitečný. Vyřešíme znovu příklad 7.1.

Příklad 7.2 *Pomocí Hurwitzova kritéria vyšetřete stabilitu systému*

$$\begin{aligned} x_1' &= -3x_1 - x_2 - x_4 \\ x_2' &= x_1 - 2x_2 \\ x_3' &= -x_2 - x_3 + x_4 \\ x_4' &= x_2 - 2x_4. \end{aligned}$$

Řešení: Opět zadáme matici soustavy. Pak si necháme spočítat její charakteristický polynom:

```
A = [-3 -1 0 -1; 1 -2 0 0; 0 -1 -1 1; 0 1 0 -2];
```

```
p = poly(A);
```

```
p =
    1.0000    8.0000   24.0000   32.0000   15.0000
```

Charakteristický polynom je tedy

$$p(\lambda) = \lambda^4 + 8\lambda^3 + 24\lambda^2 + 32\lambda + 15.$$

Z jeho koeficientů sestavíme Hurwitzovu matici (viz MDRE-(8.15)). Pozor na pořadí koeficientů! Např. a_1 je koeficient u λ^1 apod. Matici zde zadáme ručně, odvážný čtenář si může napsat program, který bude matici z koeficientů polynomu sestavovat automaticky. Pak vypočítáme jednotlivé determinanty. K tomu připomeňme, že chceme-li pracovat pouze s částí určité matice, můžeme zadat indexy příslušných řádků a sloupců. Např. zápis $H(1:3, 1:3)$ znamená, že se mají vzít pouze první tři řádky a první tři sloupce matice H .

```
H = [32 15 0 0; 8 24 32 15; 0 1 8 24; 0 0 0 1];
d1 = det(H(1,1))
d2 = det(H(1:2,1:2))
d3 = det(H(1:3,1:3))
d4 = det(H(1:4,1:4))

d1 =
    32
d2 =
    648
d3 =
   4160
d4 =
   4160
```

Vidíme, že všechny determinanty jsou kladné. Všechny kořeny polynomu p tedy mají záporné reálné části a systém je asymptoticky stabilní.

8 Rovinný autonomní diferenciální systém

Budeme se zabývat systémem dvou diferenciálních rovnic, ve kterých se na pravé straně nevyskytuje proměnná t :

$$\begin{aligned}x' &= f_1(x, y) \\ y' &= f_2(x, y).\end{aligned}\tag{8.1}$$

8.1 Fázový obraz řešení

Fázový obraz je projekce řešení systému (8.1), $x = x(t)$, $y = y(t)$, do roviny (x, y) .

Příklad 8.1 Najděte řešení počáteční úlohy

$$\begin{aligned}x' &= -x + 2y, & x(0) &= 2, \\ y' &= -2x - y, & y(0) &= 3\end{aligned}$$

a nakreslete jeho fázový obraz pro $t \in \langle 0, 5 \rangle$.

Řešení: Nejprve vyřešíme soustavu:

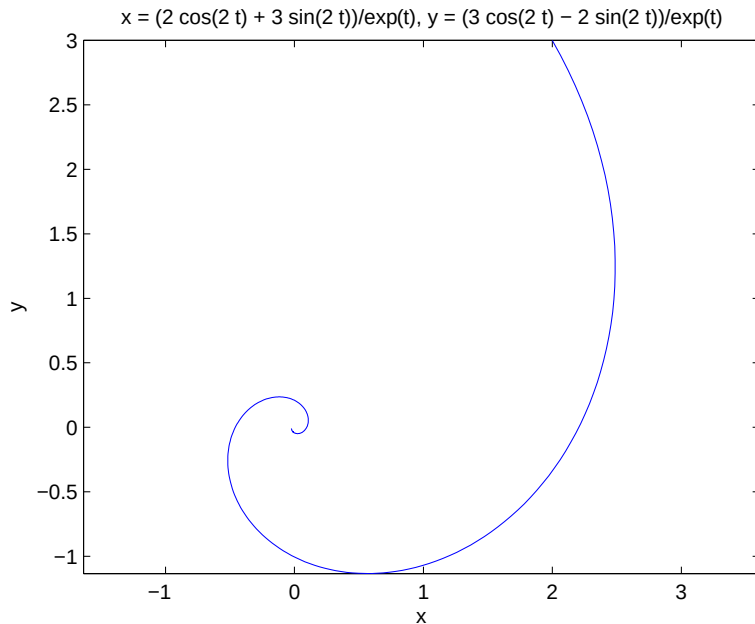
```
rce1 = 'Dx = -x+2*y';
rce2 = 'Dy = -2*x-y';
res = dsolve(rce1,rce2,'x(0)=2','y(0)=3');
x = simplify(res.x)
y = simplify(res.y)

x =
(2*cos(2*t) + 3*sin(2*t))/exp(t)
y =
(3*cos(2*t) - 2*sin(2*t))/exp(t)
```

Pro kreslení fázového obrazu opět můžeme použít funkci `ezplot`. Tentokrát ale nekreslíme graf jediné funkce, ale parametricky zadanou křivku. To funkce `ezplot` také umí – zadáme-li dvě funkce jedné proměnné a rozsah pro parametr, nakreslí příslušnou křivku:

```
ezplot(x,y,[0,5])
```

Celý program pro kreslení fázového obrazu řešení najdete v souboru `fazovy_obraz.m`.



8.2 Směrové pole pro autonomní rovinný systém

Směrové pole už jsme kreslili pro jednu diferenciální rovnici. U autonomního rovinného systému je situace podobná. Řešením systému (8.1) je parametricky zadaná křivka $x = x(t), y = y(t)$. Jestliže tato křivka prochází v určitém čase t bodem (x, y) , pak tečný vektor k této křivce je dán pravými stranami soustavy (8.1). Lze to říci i jinak: vektor $(f_1(x, y), f_2(x, y))$ udává rychlost (velikost i směr), jakou se řešení pohybuje v rovině (x, y) . Takto získané směrové pole si opět můžeme nechat nakreslit.

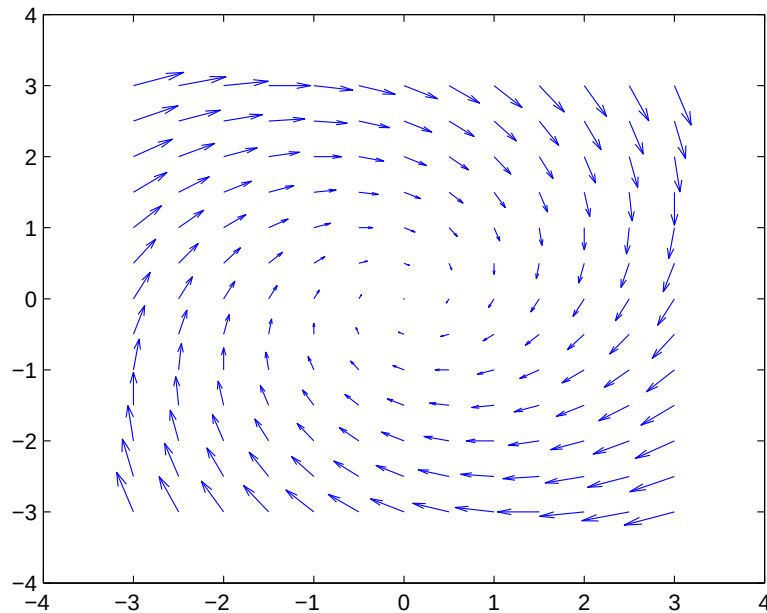
Příklad 8.2 *Znázorněte směrové pole soustavy*

$$\begin{aligned}x' &= -x + 2y, \\y' &= -2x - y\end{aligned}$$

(jedná se o soustavu z příkladu 8.1). Pak do téhož obrázku nakreslete fázový obraz řešení procházejícího bodem $(2, 3)$ (neboli řešení nalezeného v předchozím příkladu).

Řešení: Směrové pole nakreslíme např. na oblasti $\langle -3, 3 \rangle \times \langle -3, 3 \rangle$. Nejprve sestojíme síť bodů, pak vypočteme hodnoty pravých stran soustavy a necháme vykreslit vektorové pole. Tentokrát nebudeme vektory normalizovat, protože zajímavá je i jejich velikost.

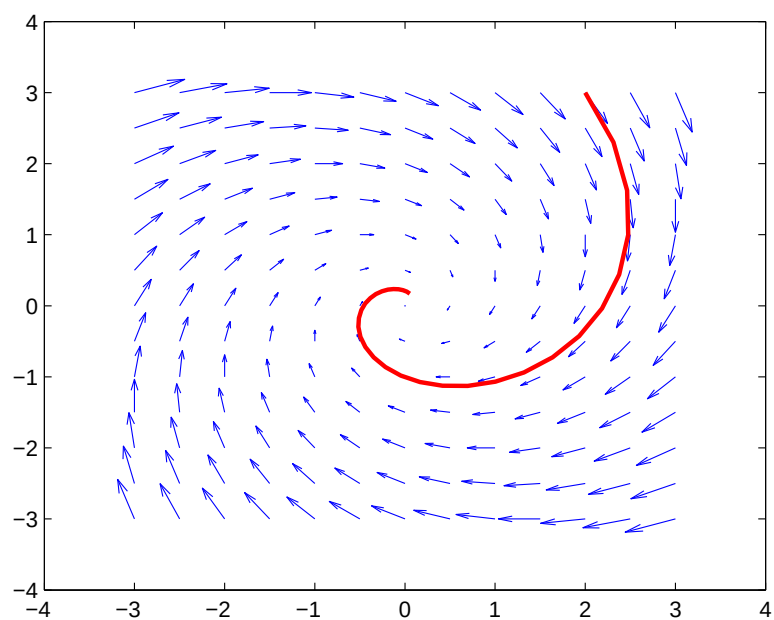
```
x=-3:0.5:3;
y=-3:0.5:3;
[xx,yy] = meshgrid(x,y);
A = [ -1 2; -2 -1];
u = A(1,1)*xx+A(1,2)*yy;
v = A(2,1)*xx+A(2,2)*yy;
quiver(xx,yy,u,v)
```



Nyní ještě přidáme fázový obraz řešení, které jsme našli v předchozím příkladu. Tam jsme pro jeho nakreslení použili funkci `ezplot`. Zde pro změnu použijeme funkci `plot`. Do nalezeného řešení dosadíme za t čísla z intervalu od 0 do 3 s krokem 0,1. K tomu použijeme funkci `subs`. Zatím jsme tuto funkci používali ve tvaru `subs(do_čeho, za_co, hodnota)`. Máme však i jinou možnost. Jestliže chceme dosadit do výrazu, který závisí např. na proměnné t , za t určitou hodnotu, stačí do t něco přiřadit a pak použít funkci `subs` pouze je jedním parametrem: `subs(do_čeho)`. Právě tak to provedeme v následující ukázce. Pak nakreslíme graf řešení, a to červenou barvou a silnější čarou.

```
hold on
res = dsolve('Dx = -x+2*y', 'Dy = -2*x-y', 'x(0) = 2', 'y(0)=3');
t = 0:0.1:3;
x = subs(res.x);
y = subs(res.y);
plot(x,y,'r','LineWidth',2)
```

Program vcelku najdete v souboru `smer_pole2.m`.



9 Parciální diferenciální rovnice, základní pojmy, zobrazení řešení

9.1 Pojem řešení parciální diferenciální rovnice

V této kapitole si ukážeme, jak ověřit, že určitá funkce je řešením zadané parciální diferenciální rovnice.

Příklad 9.1 *Ověřte, že funkce $u(x, y) = e^x \sin y$ je řešením rovnice*

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0.$$

Řešení: Pro výpočet parciálních derivací použijeme funkci `diff`. Pro výpočet parciálních derivací vyšších řádů zadáme proměnnou, podle které se derivuje, a řád derivace jako další argumenty funkce:

`diff( výraz, proměnná, řád_derivace )`

Pro náš příklad to tedy bude takto:

```
syms x y
u = exp(x)*sin(y);
diff(u,x,2)+diff(u,y,2)

ans =
0
```

Funkce $u = e^x \sin y$ je tedy opravdu řešením zadané diferenciální rovnice.

9.2 Zobrazení řešení

Budeme se zabývat zobrazením řešení pro případ, že řešení závisí na dvou prostorových proměnných x a y neboli že řešením je funkce $u = u(x, y)$.

Příklad 9.2 *Funkce $u(x, y) = x^2 + y^2$ je řešením rovnice*

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 4.$$

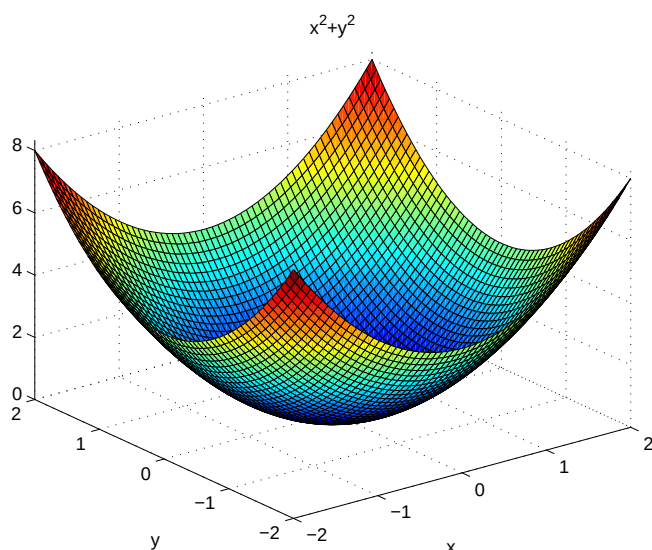
Nakreslete graf tohoto řešení na oblasti $\langle -2, 2 \rangle \times \langle -2, 2 \rangle$.

Řešení: Pro kreslení grafů funkcí dvou proměnných nebo vůbec prostorových ploch je v MATLABu několik funkcí. My si zde ukážeme funkce `mesh`, `surf` a `ezsurf`. Nejjednodušší je použití funkce `ezsurf` („surf“ je z anglického *surface* = povrch). Syntaxe je

`ezsurf( funkce, [xmin, xmax, ymin, ymax])`

Funkci můžeme zadat jako řetězec, symbolický výraz, může to být inline funkce, anonymní funkce nebo funkční handle. Pro náš případ to může vypadat např. takto:

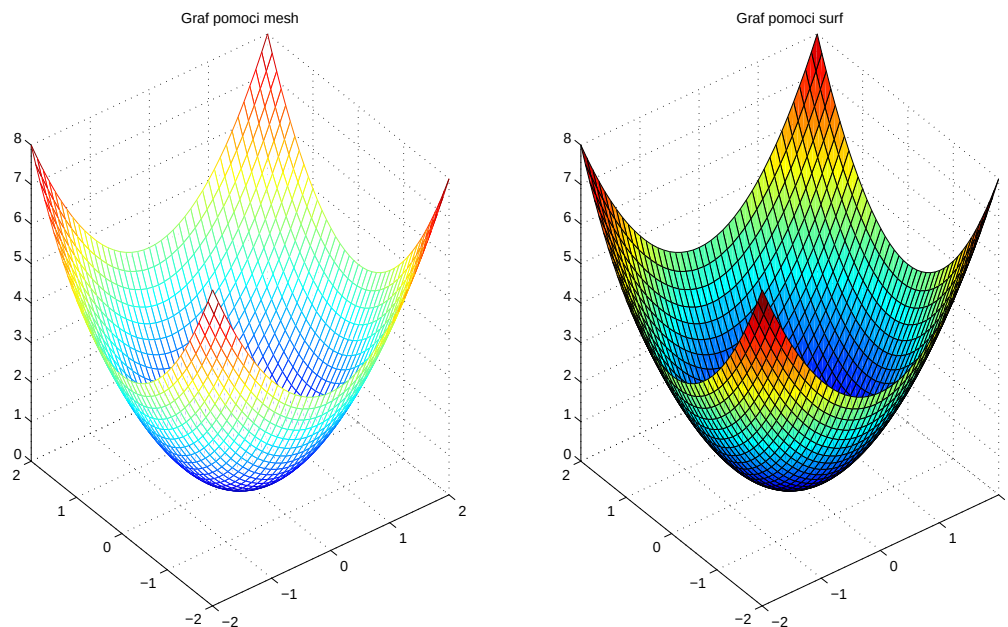
```
ezsurf('x^2+y^2', [-2,2,-2,2])
```



Funkce `ezsurf` toho umí více, například plochy zadané parametricky, ale to zde potřebovat nebudeme.

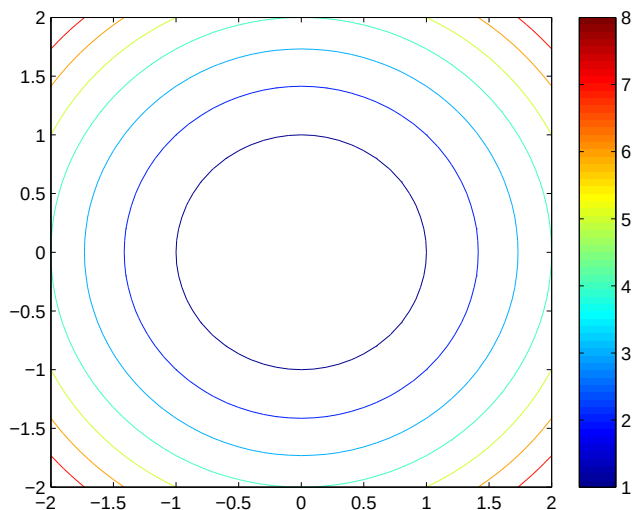
Použití funkcí `mesh` a `surf` je o něco složitější a vypadá podobně jako použití funkce `plot`. Jako argumenty zadáme x-ové, y-ové a z-ové souřadnice bodů, které chceme kreslit. Funkce `mesh` pak nakreslí síťový graf („drátěnku“), zatímco funkce `surf` jednotlivé plošky kreslí plně. Chceme-li kreslit graf na oblasti $\langle -2, 2 \rangle \times \langle -2, 2 \rangle$, vytvoříme si nejprve síť bodů v rovině. K tomu použijeme funkci `meshgrid`, se kterou jsme se již seznámili v příkladu 2.4. Ve všech bodech sítě pak spočítáme funkční hodnoty a vše necháme vykreslit. Pro srovnání si necháme nakreslit vedle sebe obrázky získané oběma funkcemi, `mesh` i `surf`:

```
x = -2:0.1:2;
y = -2:0.1:2;
[xx,yy] = meshgrid(x,y);
zz = xx.^2+yy.^2;
subplot(1,2,1)
mesh(xx,yy,zz)
title('Graf pomoci mesh')
subplot(1,2,2)
surf(xx,yy,zz)
title('Graf pomoci surf')
```



Další možnost je nezobrazovat graf prostorově, ale nechat nakreslit vrstevnice, tj. křivky, na kterých je řešení konstantní. K tomu slouží funkce `contour`. Její použití je podobné jako u funkcí `mesh` a `surf`. Použijeme už vypočtené hodnoty `xx`, `yy` a `zz`. Vrstevnice se vykreslí různými barvami, podle toho, jak velké funkční hodnotě odpovídají. Protože bychom však z takového obrázku poznali, kde je řešení větší a kde menší, ale nepoznali bychom, jakých hodnot nabývá, necháme si navíc zobrazit barevnou stupnici (`colorbar`):

```
contour(xx,yy,zz)
colorbar
```



Programy pro kreslení najdete v souborech `graf3D.m` a `vrstevnice.m`.

10 Vlnová rovnice

Budeme zkoumat řešení tzv. vlnové rovnice, která též popisuje kmitání struny:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}. \quad (10.1)$$

10.1 D'Alembertův vzorec

Uvažujeme nekonečně dlouhou strunu, tj. $-\infty < x < \infty$. Hledáme řešení, které vyhovuje podmínkám

$$\begin{aligned} u(x, 0) &= \varphi(x), \\ \frac{\partial u}{\partial t}(x, 0) &= \psi(x). \end{aligned} \quad (10.2)$$

V tomto případě je řešení rovnice (10.1), které splňuje podmínky (10.2), dáno vzorcem

$$u(x, t) = \frac{1}{2} (\varphi(x - ct) + \varphi(x + ct)) + \frac{1}{2c} \int_{x-ct}^{x+ct} \psi(\xi) d\xi. \quad (10.3)$$

Příklad 10.1 Najděte řešení rovnice

$$\frac{\partial^2 u}{\partial t^2} = 4 \cdot \frac{\partial^2 u}{\partial x^2}$$

za předpokladu, že

$$u(x, 0) = 0, \quad \frac{\partial u}{\partial t}(x, 0) = \sin x.$$

Řešení: Definujeme symbolické proměnné x , t a ξ , zadáme funkce $\varphi(x) = 0$ a $\psi(\xi) = \sin \xi$ a vypočítáme řešení podle vzorce (10.3). Pro dosazení argumentu $x - ct$, resp. $x + ct$ do funkce φ použijeme funkci `subs`:

```
syms x t xi
phi = 0;
psi = sin(xi);
c = 2;
u = 1/2*(subs(phi,x,x-c*t)+subs(phi,x,x+c*t)) + 1/(2*c)*int(psi,xi,x-c*t,x+c*t)
u =
cos(t)*sin(t)*sin(x)
```

Případně bychom mohli nechat výsledek zjednodušit pomocí funkce `simplify`, ale v tomto případě to není potřeba.

Dále si vytvoříme animaci kmitání struny. Animaci už jsme jednou dělali, a to v příkladu 3.4.

Budeme zobrazovat řešení pro t od 0 do 10 s krokem 0,1. Do proměnné t přiřadíme příslušné hodnoty. Do řešení je pak dosadíme, čímž dostaneme vektor funkcí proměnné x – průběhy řešení v jednotlivých časových krocích:

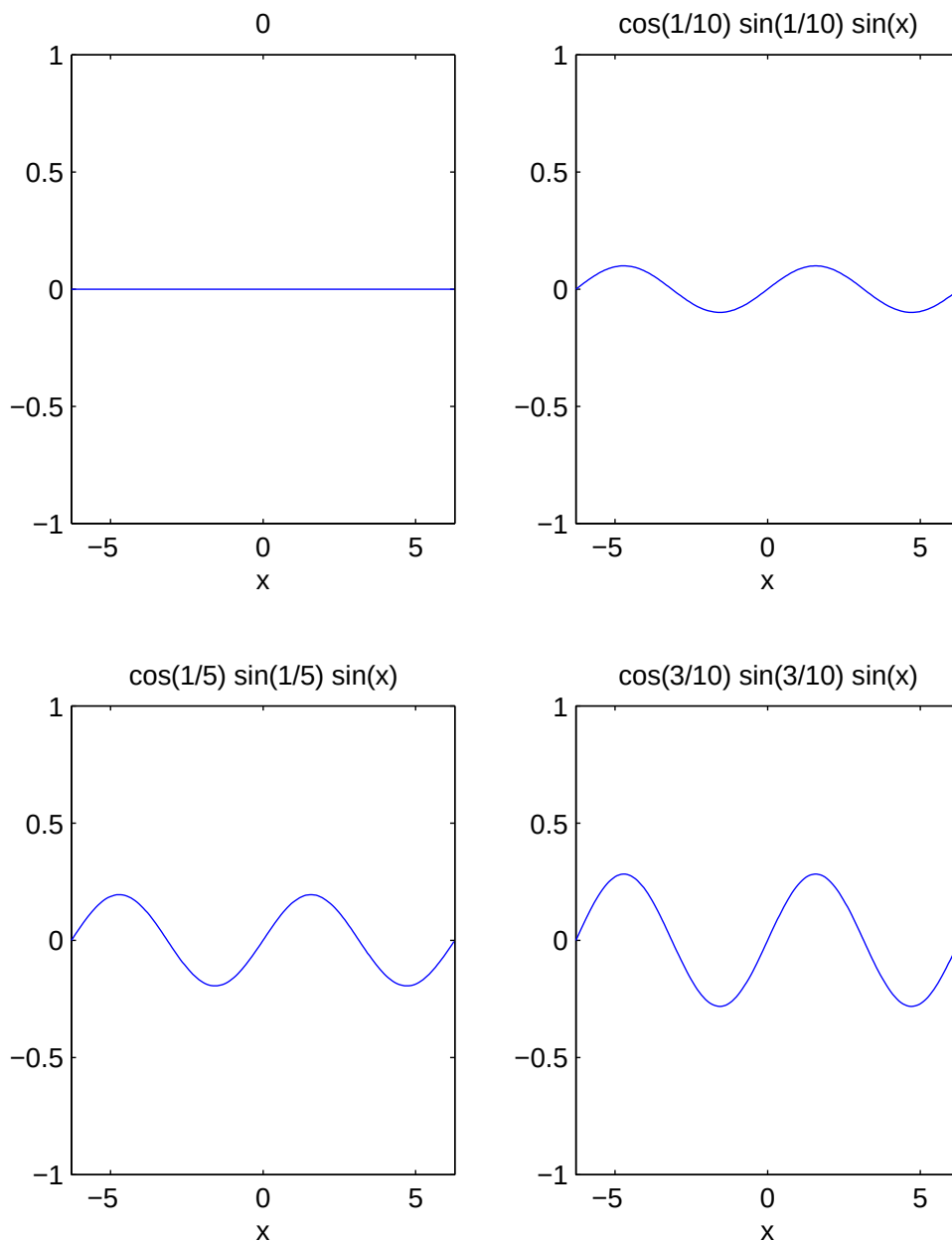
```
t = 0:0.1:10;
uu = subs(u)

uu =
[ 0, cos(1/10)*sin(1/10)*sin(x), cos(1/5)*sin(1/5)*sin(x), ...]
```

Tato řešení budeme postupně kreslit na intervalu $\langle -2\pi, 2\pi \rangle$. Obrázek si vždy uložíme pomocí funkce `getframe`.

```
x1 = -2*pi;
x2 = 2*pi;
for i=1:length(t)
    ezplot(uu(i), [x1,x2])
    axis([x1 x2 -1 1])
    obr(i) = getframe;
end
```

Vznikne animace, níže je předvedení prvních pár snímků.
Celý program najdete v souboru `dAlembert.m`.



Chceme-li animaci přehrát znovu, použijeme funkci `movie` (viz příklad 3.4). Jistá komplikace je v tom, že napíšeme-li pouze např.

```
movie(obr,1,5)
```

animace se pustí, obrázky jsou správné, ale popis os může být špatný. Jestliže chceme mít osy nastavené správně, můžeme si napřed pro animaci vytvořit nové okno s obrázkem, nastavit rozsah os, a pak teprve spustit animaci (nesmíme samozřejmě v mezinápravě smazat proměnné `x1`, `x2` a `obr`):

```
figure;
axis([x1 x2 -1 1])
movie(obr,1,5)
```

10.2 Fourierova metoda separace proměnných

Opět hledáme řešení rovnice (10.1) s počátečními podmínkami (10.2). Struna je tentokrát konečné délky l , tj. $x \in \langle 0, l \rangle$, a na koncích je upevněná, tj. platí okrajové podmínky

$$u(0, t) = 0, \quad u(l, t) = 0 \quad \text{pro } t \geq 0.$$

Podle MDRE-(13.12) lze řešení najít ve tvaru nekonečné řady

$$u(x, t) = \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi ct}{l} + b_n \sin \frac{n\pi ct}{l} \right) \sin \frac{n\pi x}{l}, \quad (10.4)$$

kde

$$a_n = \frac{2}{l} \int_0^l \varphi(x) \sin \frac{n\pi x}{l} dx, \quad b_n = \frac{2}{n\pi c} \int_0^l \psi(x) \sin \frac{n\pi x}{l} dx. \quad (10.5)$$

Příklad 10.2 Najděte profil kmitající struny délky $l = 2$, jestliže v rovnici (10.1) je $c = 1$ a počáteční tvar struny je lomená čára,

$$u(x, 0) = 1 - |x - 1|.$$

Řešení: Definujeme symbolické proměnné x , t a n , konstanty c a l a funkce $\varphi(x) = 1 - |1 - x|$, $\psi(x) = 0$. V našem příkladu bychom s funkcí ψ vůbec nemuseli pracovat, vidíme hned, že koeficienty b_n budou nulové. Vytvoříme však skript, který by měl fungovat univerzálně, stačí jen přepsat funkce `phi` a `psi` a konstanty c a l .

Pak vypočítáme a_n a b_n podle vzorců (10.5):

```
syms x t n
c = 1;
l = 2;
phi = 1-abs(x-1);
psi = 0;

a = 2/l*int(phi*sin(n*pi*x/l),x,0,l)
b = 2/(n*pi*c)*int(psi*sin(n*pi*x/l),x,0,l)

a =
-(4*(sin(pi*n) - 2*sin((pi*n)/2)))/(pi^2*n^2)
b =
0
```

Vidíme, že dle očekávání je $b_n = 0$. Abychom se lépe vyznali ve výrazu pro a_n (těch závorek je tam trochu moc), necháme si jej „pěkně“ vypsát:

```
pretty(a)
```

$$-\frac{4 \left(\sin(\pi n) - 2 \sin\left(\frac{n\pi}{2}\right) \right)}{\pi^2 n^2}$$

Takže

$$a_n = -\frac{4}{\pi^2 n^2} \left(\sin(n\pi) - 2 \sin \frac{n\pi}{2} \right).$$

MATLABu se, bohužel, nedá sdělit, že číslo n je celé. Lze zadávat pouze předpoklady, že určitá symbolická proměnná je reálná (viz příklad 5.2) nebo kladná. Takže MATLAB už není schopen výsledek dále zjednodušit. My však víme, že pro n celé je $\sin n\pi = 0$. Dále, pro n sudé je i $\sin \frac{n\pi}{2} = 0$, zatímco pro n liché, tj. $n = 2k - 1$, $k = 1, 2, \dots$, je $\sin \frac{n\pi}{2}$ střídavě ± 1 neboli $\sin \frac{n\pi}{2} = (-1)^{k-1}$. Celkem se a_n zredukuje na

$$a_{2k} = 0, \quad a_{2k-1} = -\frac{4}{\pi^2 (2k-1)^2} \cdot (-2) \cdot (-1)^{k-1} = \frac{8(-1)^{k-1}}{\pi^2 (2k-1)^2}, \quad k = 1, 2, \dots$$

Můžeme se o tom přesvědčit i pomocí MATLABu, necháme-li si několik prvních hodnot a_n spočítat. Za **n** dosadíme postupně 1 až 10, a to symbolicky. Jinak bychom dostali numerické hodnoty koeficientů a_n , podle kterých bychom předchozí vzorec tak snadno neověřili.

```
subs(a,n,sym(1:10))
```

```
ans =
```

```
[ 8/pi^2, 0, -8/(9*pi^2), 0, 8/(25*pi^2), 0, -8/(49*pi^2), 0, 8/(81*pi^2), 0]
```

Řešení jsme tedy dostali ve tvaru nekonečné řady,

$$u(x, t) = \sum_{k=1}^{\infty} \left(\frac{8(-1)^{k-1}}{\pi^2 (2k-1)^2} \cos \frac{(2k-1)\pi t}{2} \right) \sin \frac{(2k-1)\pi x}{2}.$$

Nyní opět vytvoříme animaci zobrazující vývoj řešení v čase. Proti řešení získanému pomocí d'Alembertova vzorce, které bylo v uzavřeném tvaru, je zde ale problém s nekonečnou řadou. Mohli bychom se pokusit nechat sumu vypočítat pomocí funkce **symsum** (viz příklad 3.4), ale to se podaří jen velmi zřídka. Konkrétně pro tento příklad MATLAB dlouho bojuje a pak vypíše výsledek opět ve tvaru nekonečné řady.

Proto budeme pracovat pouze s přibližným řešením, které získáme tak, že vezmeme pouze prvních N členů sumy.

Vypočítáme hodnoty koeficientů a_n a b_n pro $n = 1, \dots, N$. Pak vypočítáme prvních N členů sumy a jejich součet:

```

N = 20;
n = sym(1:N);
an = subs(a);
bn = subs(b);

un = (an.*cos(n*pi*c*t/l) + bn.*sin(n*pi*c*t/l)).*sin(n*pi*x/l);
u = sum(un)

u =
(8*cos((pi*t)/2)*sin((pi*x)/2))/pi^2 -
(8*cos((3*pi*t)/2)*sin((3*pi*x)/2))/(9*pi^2) + ...
- (8*cos((19*pi*t)/2)*sin((19*pi*x)/2))/(361*pi^2)

```

Nyní symbolický výraz u převedeme na funkci, se kterou se pohodlněji pracuje. Přitom použijeme funkci `char`, která symbolický výraz převede na textový řetězec.

```
uu = inline(char(u), 'x', 't');
```

Teď budeme postupně kreslit řešení na intervalu $\langle 0, l \rangle$ postupně pro t od 0 do 10 s krokem 0,1 a každý obrázek uložíme:

```

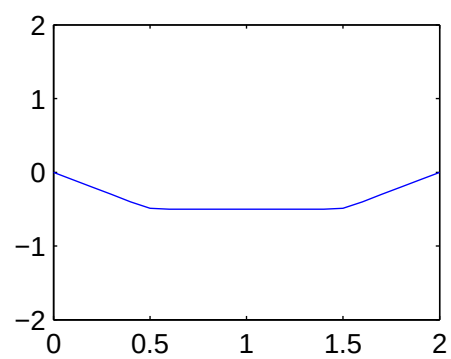
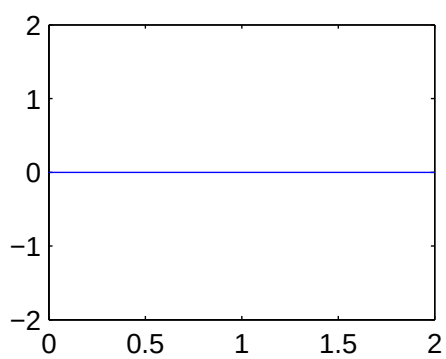
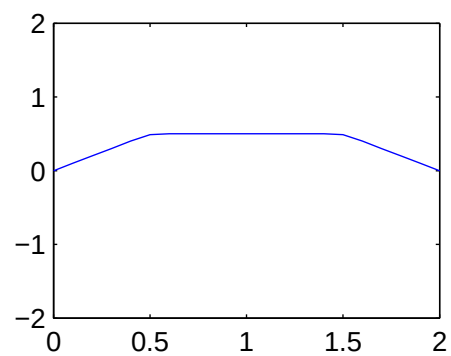
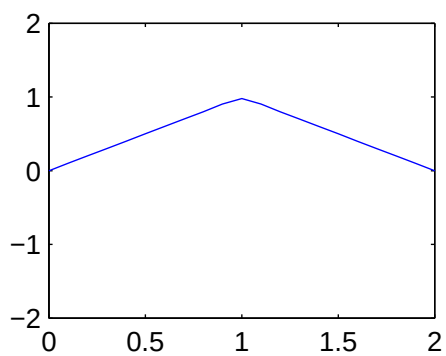
xx = linspace(0,1,21); % interval [0,1] dělíme na 20 dílků
tt = 0:0.1:10;
for i=1:length(tt)
    uxt = uu(xx,tt(i)); % řešení v i-tém časovém kroku
    plot(xx,uxt)
    axis([0, 1, -2, 2])
    obr(i) = getframe;
end

```

Vznikne animace, na ukázkou předvedeme její 1., 6., 11. a 16. snímek.

Chceme-li si animaci znovu přehrát, použijeme funkci `movie`, ale pozor na rozsah os – viz předchozí příklad.

Celý program najdete v souboru `fourier_vlny.m`.



11 Metoda konečných prvků pro PDR - PDE toolbox MATLABu

11.1 Cíl kapitoly

Cílem této kapitoly je předvést, jak můžeme parciální diferenciální rovnice numericky řešit s použitím Matlabu. Matlab nepoužívá pro řešení PDR metodu konečných diferencí, popsanou v kapitole 15 skript MDRE, nýbrž metodu konečných prvků, která je jednou z nejdůležitějších a nepoužívanějších numerických metod pro řešení parciálních diferenciálních rovnic. Princip této metody zde bude nastíněn jen velmi, velmi zhruba, protože na podrobnější vysvětlení bohužel nemáme prostor.

11.2 Metoda konečných prvků

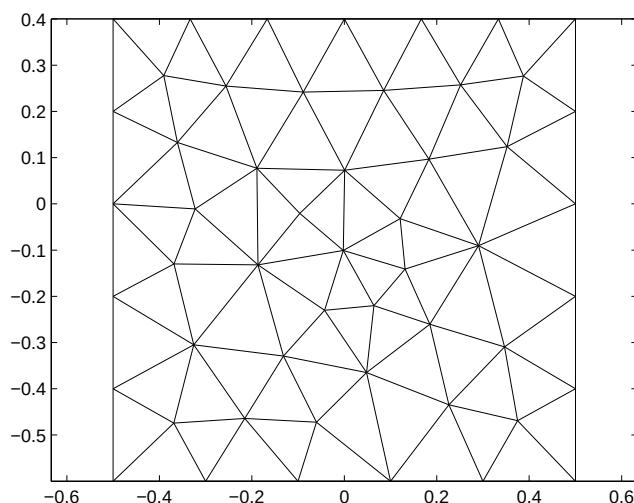
Metoda konečných prvků (MKP, anglicky FEM – „finite element method“) má mnoho společných rysů s metodou konečných diferencí (MKD). Opět hledáme řešení nějaké parciální diferenciální rovnice na zadané oblasti Ω . Matlab umožňuje řešit parciální diferenciální rovnice druhého řádu na rovinných oblastech (tj. hledaná funkce u závisí na prostorových proměnných x a y a případně na čase t), ale obecně lze metodou konečných prvků řešit i rovnice vyššího než druhého řádu a ve vyšších dimenzích než v rovině (ovšem už v třírozměrném prostoru se problém technicky velmi zkomplikuje). Všude dál v této kapitole budeme uvažovat pouze rovinné oblasti.

Stejně jako u metody konečných diferencí oblast Ω pokryjeme sítí. Na rozdíl od MKD se však v základní verzi metody konečných prvků používají trojúhelníkové sítě, viz obrázek 11.1. Říkáme, že **oblast Ω ztriangulujeme**. Na obrázku vidíme, že síť nemusí být nikterak pravidelná. Je však vhodné, aby jednotlivé trojúhelníky nebyly příliš „placaté“, tj. aby jejich vnitřní úhly nebyly příliš malé.

Výhodou trojúhelníkových sítí (oproti obdélníkovým, které se používají u MKD) je to, že mohou dobře vystihnout i velmi složité oblasti. Tím odpadají problémy s realizací okrajových podmínek, které jsme řešili u MKD (viz příklad MDRE-15.1).

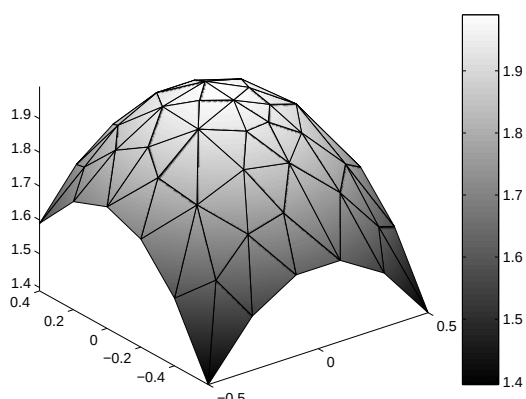
Další analogie s metodou konečných diferencí je v tom, že **původní parciální diferenciální rovnici převedeme na soustavu algebraických rovnic** (lineárních či nelineárních, dle povahy původní úlohy) s neznámými $u_1, u_2, \dots, u_n$, kde u_i je přibližná hodnota řešení v i -tém uzlu sítě a n je počet uzlů. Postup, jakým je tato soustava tzv. diskretizačních rovnic získána, je však zcela odlišný a daleko komplikovanější než u MKD a popisovat jej zde nebudeme. Soustavu diskretizačních rovnic vyřešíme – tím získáme hodnoty řešení v uzlových bodech – a za přibližné řešení rovnice na celé oblasti Ω bereme destičkovou plochu (což je prostorová analogie lomené čáry v rovině, viz obrázek 11.2) danou těmito hodnotami. To, že získáme řešení na celé oblasti, a nikoli jen v uzlech sítě, je další výhodou metody konečných prvků oproti MKD.

Pro ilustraci slouží obrázek 11.2, na kterém vidíme přibližné řešení rovnice $-\Delta u = 4$ na oblasti Ω z obrázku 11.1 s okrajovou podmínkou $u(x, y) = 2 - x^2 - y^2$ na $\partial\Omega$. Snadno bychom ověřili, že přesným řešením této rovnice s touto okrajovou podmínkou je funkce $u(x, y) = 2 - x^2 - y^2$. Grafem řešení tedy je rotační paraboloid. Vidíme, že přibližné řešení

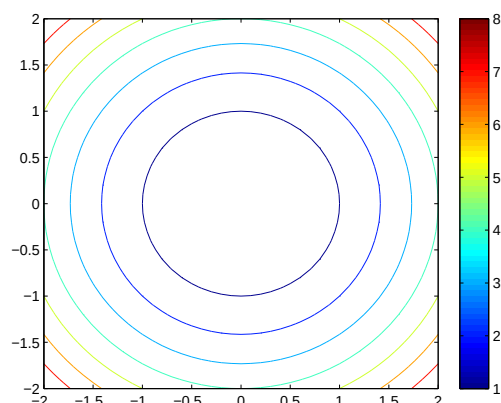


Obrázek 11.1: Triangulovaná oblast Ω

nalezené metodou konečných prvků tvarem paraboloidu vcelku odpovídá, pro přesnější srovnání bychom museli porovnat příslušné numerické hodnoty. Prostorový graf nemusí být ovšem zrovna nejpřehlednější, řešení se proto častěji znázorňuje pomocí vrstevnic, viz obrázek 11.3.



Obrázek 11.2: Graf přibližného řešení



Obrázek 11.3: Přibližné řešení téže rovnice znázorněné pomocí vrstevnic

11.3 Příklad řešení pomocí Matlabu

Poznámka 11.1 Všem jazykovým puristům se omlouváme za to, že v dalším textu budeme do češtiny míchat různá anglická slova (a občas je ještě navíc potvrdit skloňováním). Autoři z vlastní zkušenosti soudí, že v oblasti počítačových programů je příliš důsledný překlad

do češtiny spíš na škodu věci.

V Matlabu lze parciální diferenciální rovnice řešit velmi snadno, máme-li nainstalovaný tzv. PDE Toolbox (PDE = „partial differential equations“). Nejpohodlnější je pracovat s grafickým uživatelským rozhraním (GUI). Ukážeme zde řešení jednoho příkladu právě v tomto prostředí. Budeme řešit téměř totožný příklad, jako byl př. MDRE-15.1).

Příklad 11.1 Pomocí Matlabu řešte metodou konečných prvků okrajovou úlohu

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 8x \quad \text{na } \Omega, \quad (11.1)$$

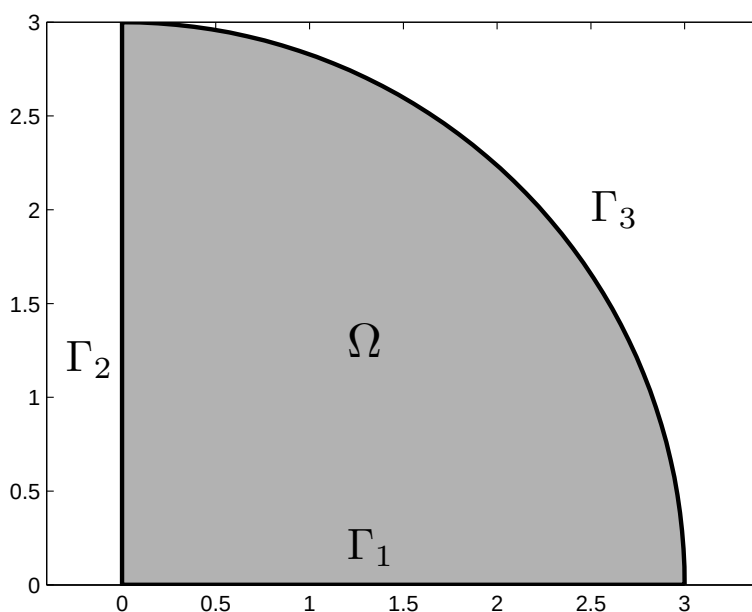
kde oblast Ω je čtvrtina kruhu se středem v počátku souřadné soustavy a poloměrem 3, která leží v prvním kvadrantu:

$$\Omega = \{(x, y) : x \geq 0, y \geq 0, x^2 + y^2 \leq 9\},$$

s Dirichletovými okrajovými podmínkami

$$\begin{aligned} u(x, y) &= x^3 & \text{na } \Gamma_1 &= \{(x, y) : 0 \leq x \leq 3, y = 0\}, \\ u(x, y) &= 0 & \text{na } \Gamma_2 &= \{(x, y) : x = 0, 0 \leq y \leq 3\}, \\ u(x, y) &= 9x(y + 1) & \text{na } \Gamma_3 &= \{(x, y) : x \geq 0, y \geq 0, x^2 + y^2 = 9\}. \end{aligned} \quad (11.2)$$

Oblast Ω s vyznačenými částmi hranice Γ_1 , Γ_2 a Γ_3 vidíme na obrázku 11.4.

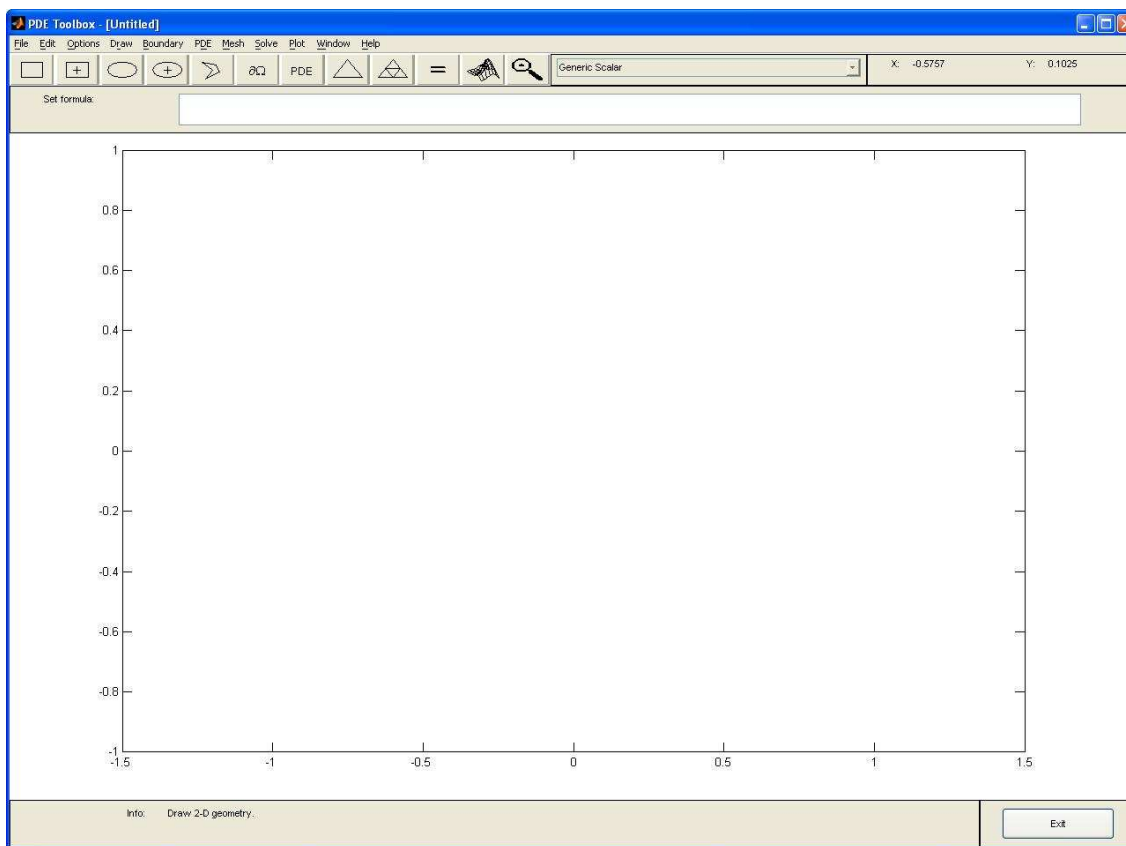


Obrázek 11.4: K příkladu 11.1: Oblast Ω a její hranice, rozdělená na tři části

Řešení: Spustíme Matlab. Do příkazového okna napíšeme příkaz k otevření prostředí pro řešení parciálních diferenciálních rovnic:

```
>> pdetool
```

Otevře se následující okno



Řešení příkladu v tomto prostředí popíšeme krok za krokem. Každá akce, kterou je třeba provést, bude označena symbolem ●.

Zadání oblasti Ω

Oblast, na které řešíme parciální diferenciální rovnici, můžeme zadat interaktivně. Na bílou plochu uprostřed můžeme umisťovat různé geometrické obrazce (obdélníky, kruhy, elipsy a polygony) a pak z nich pomocí množinových operací (sjednocení, průniku a rozdílu) sestavit požadovanou oblast. Naši oblast (čtvrtkruh) dostaneme jako průnik kruhu se středem v počátku a poloměrem 3 a čtverce, který má levý dolní vrchol v počátku a stranu o délce 3 (případně cokoli většího než 3).

Rozmezí os na ploše neodpovídá našim potřebám, a proto je musíme změnit:

- V menu v položce **Options** vybereme **Axis Limits...** a nastavíme správné rozmezí - pro náš příklad můžeme zadat pro obě osy např. meze -1 až 4.

Dále nastavíme, aby se ukazovala mřížka a aby se body, které budeme za chvíli klikáním zadávat (vrcholy čtverce a pod.) k mřížce přichytávaly.

- V **Options** klikneme na položku **Grid**.
- V **Options** klikneme na položku **Snap** („snap to grid“ = „přilnout k mřížce“).

Nyní zadáme kruh:

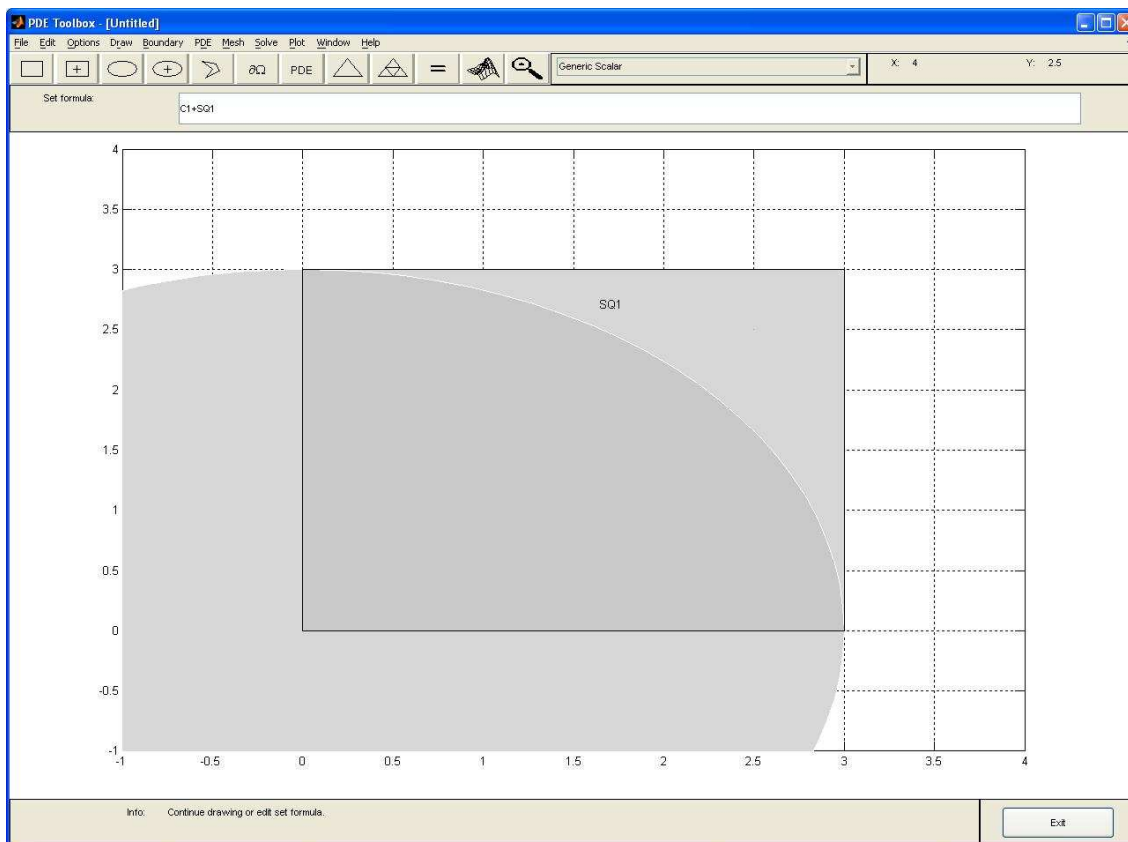
- Klikneme na tlačítko se symbolem $\oplus$. Tím budeme zadávat elipsu, s tím, že první bod, na který klikneme, je jejím středem.
- Najedeme myší na bod $(0, 0)$, stiskneme tlačítko myši (díky tomu, že jsme zvolili „snap

to grid“, se nemusíme trefit úplně přesně) a táhneme - objeví se obrys elipsy. Táhneme, až elipsu (v našem případě vlastně kruh) natáhneme do požadovaných rozměrů.

Podobným způsobem zadáme čtverec:

- Klikneme na tlačítko se symbolem . Tím budeme zadávat obdélník.
- Opět najedeme na bod (0, 0) (levý dolní roh čtverce) a dotáhneme kurzor do bodu (3, 3) (pravý horní roh).

Výsledek by měl vypadat nějak takto:



Kdybychom některý z objektů zadali jinak, než jsme chtěli, můžeme jej vcelku snadno opravit: Nejprve na požadovaný objekt klikneme a tím jej vybereme pro další úpravy (aktuálně vybraný objekt má černě zvýrazněnou hranici). Je-li nevhodně umístěn, ale velikost má přitom správnou, můžeme jej pomocí myši přetáhnout na jiné místo. Chceme-li změnit velikost, stačí, když na vybraný objekt „doubleklikneme“ (Jak je tohle správně česky? Poklepeme? Dvojklikneme?). Otevře se dialog, ve kterém můžeme změnit rozměry, umístění i název. Pokud jsme to zkazili úplně, můžeme vybraný objekt stisknutím klávesy Delete odstranit a začít znovu.

Zatím jsme jen zadali kruh a čtverec, ale nijak jsme počítači nesdělili, že nás zajímá jejich průnik. To provedeme nyní. Podíváme se na políčko nadepsané **Set formula**. Do tohoto políčka zadáváme množinový vzorec („set“ má kromě mnoha jiných i význam „množina“, „formula“ snad překládat netřeba), pomocí něhož je z jednotlivých zadaných oblastí sestavena oblast výsledná. Můžeme používat operátory + (sjednocení), * (průnik)

a – (množinový rozdíl). Při našem zadávání kruhu a čtverce se v poli Set formula automaticky objevil text $C1+SQ1$. Kdybychom to tak nechali, za oblast Ω by se vzalo sjednocení oblastí $C1$ (C jako „circle“ = kruh) a $SQ1$ (SQ jako „square“ - čtverec). My ale potřebujeme průnik, a proto

- změníme text v editačním poli **Set formula** na $C1*SQ1$. Navenek nepoznáme žádnou změnu, obrázek bude vypadat pořád stejně.

Nyní nastal vhodný okamžik pro uložení výdobytků naší práce.

- Uložíme rozpracovanou úlohu např. jako soubor `priklad1.m` (klasicky: v menu **File**, **Save as...**).

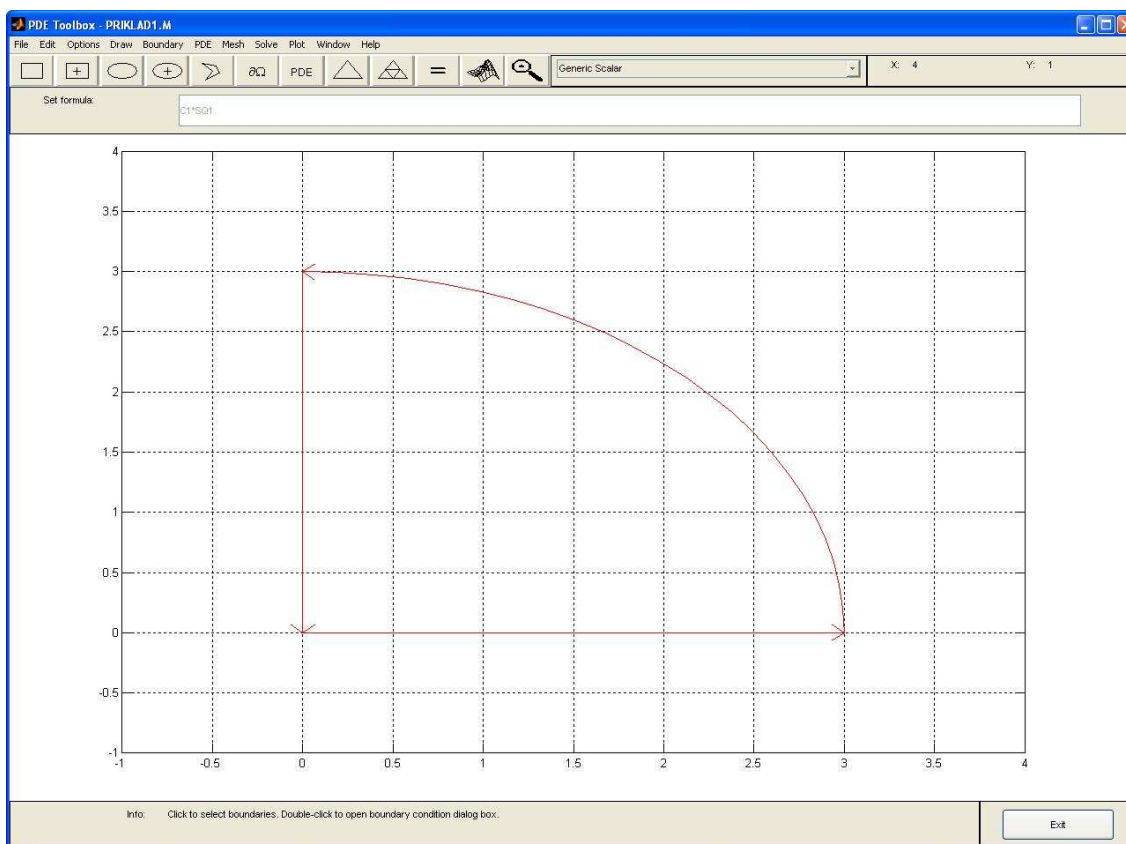
Můžeme se do uloženého souboru podívat, např. v editoru Matlabu nebo třeba v Poznámkovém bloku, jak kdo chce. Uvidíme, že Matlab automaticky vytvořil poměrně dlouhý soubor, v němž většině věcí nerozumíme, a proto do něj nebudeme vrtat.

Při další práci je vhodné čas od času úlohu opět uložit, dále již to zdůrazňovat nebudeme.

Zadání okrajových podmínek

Pokračujeme zadáním okrajových podmínek. Nejprve si necháme zobrazit hranici oblasti.

- Klikneme na tlačítko se symbolem $\partial\Omega$. (Jiná možnost: v menu **Boundary**, pak **Boundary Mode**.) Ukáže se nám hranice oblasti:



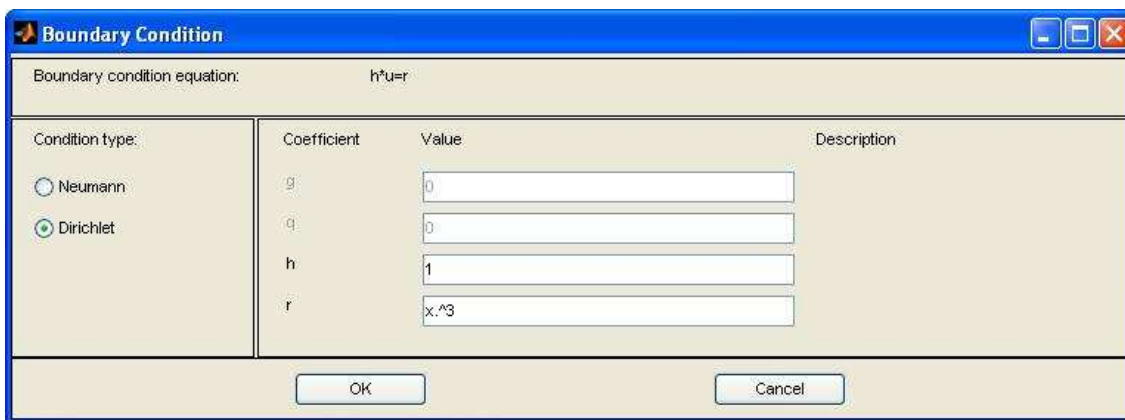
Postupně zadáme okrajové podmínky podle předpisů (11.2). Nejprve zadáme podmínku na vodorovné části hranice Γ_1 (viz též obrázek 11.4).

- Doubleklikneme na vodorovnou část hranice. Otevře se dialog pro zadávání okrajové podmínky. Jiná možnost: Na příslušnou část hranice klikneme (jednou). Tím bude tato část hranice vybrána pro další práci. Pak vybereme v menu **Boundary** a **Specify Boundary Conditions...**

V dialogu nyní zadáme okrajovou podmínku $u(x, y) = x^3$. V Matlabu lze zadávat dva základní druhy okrajových podmínek, Dirichletovy (je přímo zadáno, čemu se má řešení na hranici rovnat) a Neumannovy (které obsahují též derivaci řešení ve směru normály k hranici). Naše okrajové podmínky jsou Dirichletova typu.

- V dialogu proto vybereme (či spíš necháme nastaveno) **Condition type** na **Dirichlet**. Matlab očekává nyní podmínku ve tvaru (viz horní část dialogu) $h \cdot u = r$, kde h a r jsou zadané funkce, případně konstanty. Pro naši podmínku $u(x, y) = x^3$ je $h = 1$ a $r(x, y) = x^3$.

- V dialogu vyplníme kolonku pro funkci r výrazem `x.^3`. (Funkce h je na jedničku nastavená automaticky.) Vyplněný dialog by měl vypadat takto:



Podobným způsobem zadáme okrajové podmínky na ostatních částech hranice:

- Doubleklikneme na svislou část hranice. V dialogu zkontrolujeme, zda je zatržen Dirichletův typ okrajových podmínek a vyplníme kolonku pro funkci r výrazem 0.
- Totéž provedeme pro obloukovou část hranice, tentokrát zadáme $9 \cdot x \cdot (y+1)$.

Zadání rovnice

Nyní zadáme samotnou parciální diferenciální rovnici, kterou chceme vyřešit.

- Klikneme na tlačítko **PDE** nebo v menu vybereme **PDE** a pak **PDE Specification...** Otevře se dialog pro zadání rovnice. V levé části dialogu vybíráme typ rovnice - na výběr máme rovnici eliptickou, parabolickou, hyperbolickou a problém vlastních čísel. Naše rovnice (11.1) je typu eliptického, proto
- vybereme (případně jen zkontrolujeme, zda je vybrán) z možností pro **Type of PDE** typ **Elliptic**.

Rovnice je nyní očekávána (viz horní část dialogu) ve tvaru

$$-\operatorname{div}(c \cdot \operatorname{grad} u) + au = f \quad (11.3)$$

a po nás se chce, abychom doplnili funkce c , a a f . Doufáme, že tvarem (11.3) nejste příliš zaskočeni, pro jistotu však připomeňme, že je-li f funkce proměnných $x_1, x_2, \dots, x_n$, pak gradient funkce f je

$$\text{grad } f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right),$$

a jsou-li $g_1, g_2, \dots, g_n$ funkce proměnných $x_1, x_2, \dots, x_n$, pak divergence zobrazení $g = (g_1, \dots, g_n)$ je

$$\text{div } g = \frac{\partial g_1}{\partial x_1} + \frac{\partial g_2}{\partial x_2} + \dots + \frac{\partial g_n}{\partial x_n}.$$

Těž si snad pamatujete, že

$$\text{div}(\text{grad } f) = \frac{\partial}{\partial x_1} \left(\frac{\partial f}{\partial x_1} \right) + \dots + \frac{\partial}{\partial x_n} \left(\frac{\partial f}{\partial x_n} \right) = \frac{\partial^2 f}{\partial x_1^2} + \dots + \frac{\partial^2 f}{\partial x_n^2} = \Delta f.$$

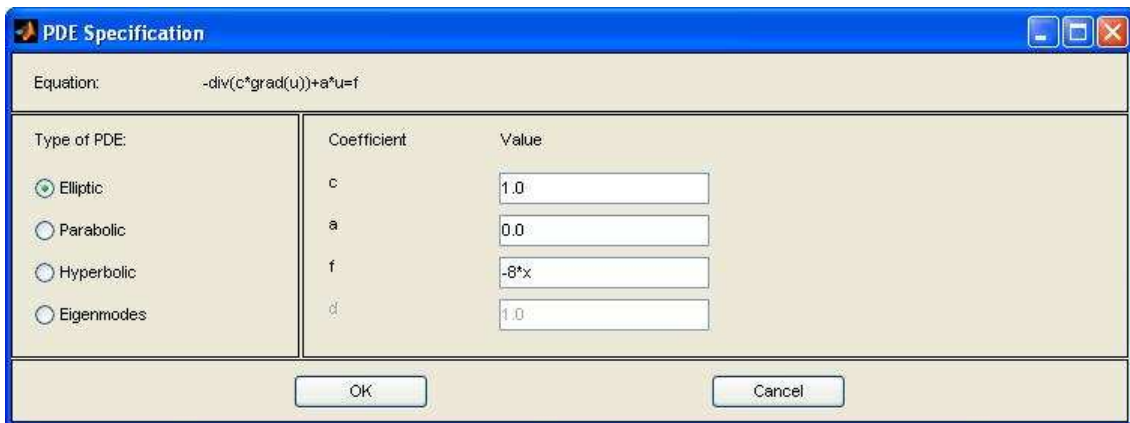
Řešená rovnice (11.1), tj. $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 8x$, po snadné úpravě $-\Delta u = -8x$, se tedy do tvaru (11.3) přepíše jako

$$-\text{div}(1 \cdot \text{grad } u) + 0 \cdot u = -8x.$$

Proto dialog pro zadání rovnice doplníme takto (některé z uvedených hodnot už tam možná jsou „samy od sebe“, pak je pochopitelně necháme být):

- Do kolonky pro funkci c doplníme konstantu 1, do kolonky pro a napíšeme nulu a do kolonky pro f napíšeme $-8*x$.

Celá věc by pak měla vypadat následovně:



Triangulace oblasti

Oblast Ω ztriangulujeme:

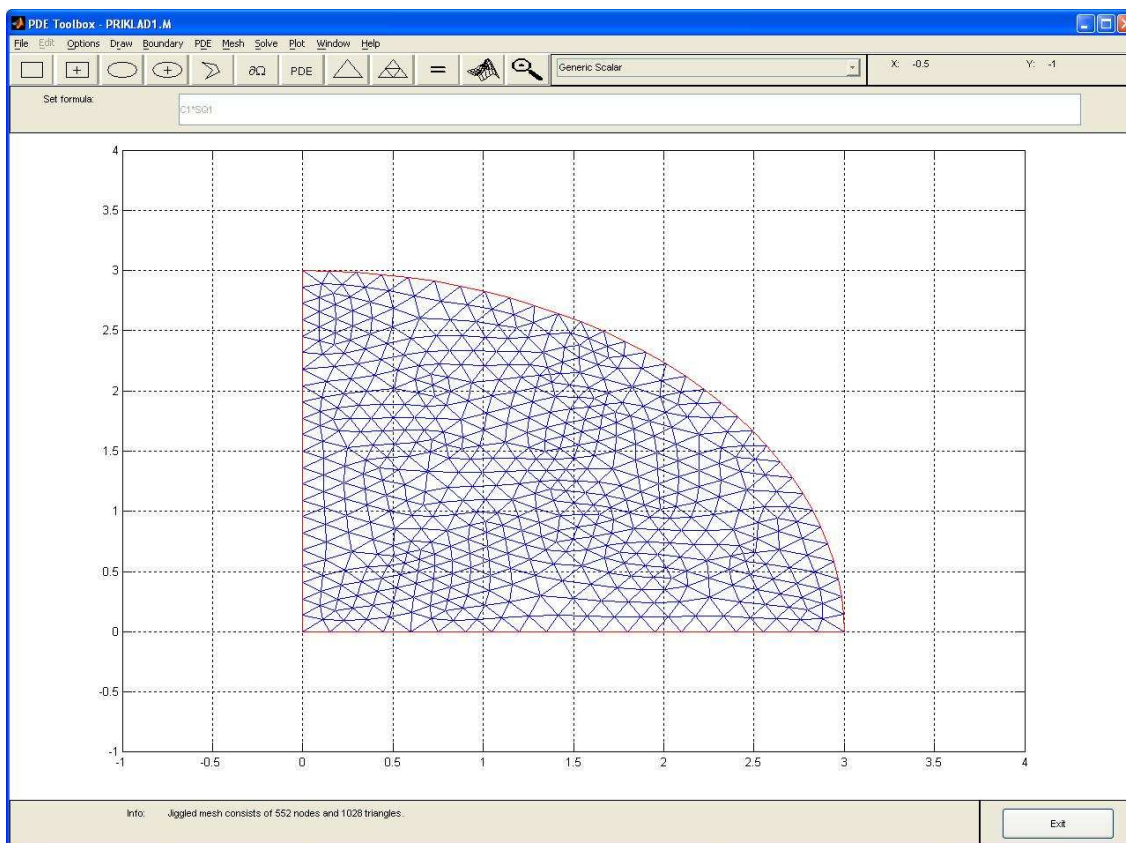
- Klikněte na tlačítko s trojúhelníkem nebo v menu vyberte **Mesh** a pak **Initialize Mesh**. Chceme-li mít síť jemnější, můžeme
- kliknout na tlačítko s trojúhelníkem rozděleným na čtyři menší trojúhelníky nebo v menu vybrat **Mesh** a pak **Refine Mesh**.

Zjemněnou síť pak můžeme ještě poněkud vylepšit (zpravidelnit, odstranit z ní některé úzké trojúhelníky) pomocí funkce Jiggle. (Původní význam slova „jiggle“ je „pohupovat“ či „trhaně se pohybovat“, v souvislosti se sítí se tím myslí zhruba to, že uzly sítě se trochu popřemísťují, každý uzel se přesune do průměru svých sousedů.)

- Můžeme v menu vybrat **Mesh** a pak **Jiggle Mesh** - tuto operaci můžeme provádět i opakovaně.

Pokud se nám to, co se se sítí děje, nelíbí, můžeme úpravy brát zpět pomocí menu: Mesh, pak Undo Mesh Change. První návrh sítě můžeme ovlivnit pomocí nastavení parametrů (v menu Mesh, Parameters...). Zde můžeme např. nastavit maximální povolenou délku hrany (maximum edge size).

Jestliže jsme zjemnění a „jigglování“ provedli právě jednou, měli bychom mít na obrazovce toto:



Síť můžeme exportovat pro její případné další použití: v menu Mesh, pak Export Mesh... Síť bude uložena pomocí tří matic, jejichž jména si můžeme vybrat. Matlab nám nabízí jména **p**, **e** a **t**. První matice bude obsahovat informace o uzlech sítě (points), druhá o hranách (edges) a třetí o trojúhelnících (triangles). S těmito maticemi pak v Matlabu můžeme dále pracovat dle potřeby, např. si jejich prvky můžeme nechat vypsát do souboru, který pak můžeme přenést a používat v jiném programu.

Řešení rovnice a jeho grafické znázornění

Teď když máme všechno připravené, můžeme konečně najít přibližné řešení rovnice.

- Klikneme na tlačítko s rovnítkem nebo v menu vybereme **Solve**, pak **Solve PDE**.

Řešení bude asi chvilku trvat (je nutno ne zcela jednoduchým způsobem sestavit a pak vyřešit systém zhruba 500 rovnic o 500 neznámých – pokud pracujeme se sítí, která je na předchozím obrázku). Až je počítač hotov, řešení se zobrazí pomocí dvourozměrného obrázku – hodnoty řešení na oblasti Ω jsou rozlišeny pomocí barev, vpravo máme zobrazenu stupnici (angl. colorbar), pomocí níž poznáme, jaká barva odpovídá jaké hodnotě řešení.

Chceme-li řešení exportovat pro další použití, uděláme to přes menu: Solve, pak Export Solution... Řešení je uloženo ve formě vektoru (jméno si můžeme vybrat, automaticky se nabízí u), jehož složky jsou hodnoty řešení v jednotlivých uzlech sítě. Chceme-li s tímto řešením dále pracovat, případně je (přes soubor) přenést do nějakého jiného programu, musíme k němu samozřejmě mít i příslušnou síť, hlavně její uzly, jinak je zcela bezcenné. Parametry zobrazení řešení můžeme měnit. Formulář pro nastavení parametrů se nám zobrazí po stisknutí tlačítka s 3-D grafem, případně se k němu dostaneme z menu: Plot, Parameters... Zde již necháme každému čtenáři prostor pro experimentování.

11.4 Cvičení

Příklad 1 Pomocí Matlabu najděte přibližné řešení úlohy

$$-\Delta u = 5 \sin \left(10 \arctg \frac{y}{x} \right) \quad \text{na } \Omega,$$

kde Ω je část mezikruží o poloměrech $r = 1, R = 3$, která leží v prvním kvadrantu,

$$\Omega = \{(x, y) : x \geq 0, y \geq 0, 1 \leq x^2 + y^2 \leq 9\},$$

s okrajovými podmínkami

$$\begin{aligned} u(x, y) &= 0 \quad \text{na } \Gamma_1 = \{(x, y) : x \geq 0, y \geq 0, x^2 + y^2 = 1\}, \\ u(x, y) &= 5 \quad \text{na } \Gamma_2 = \{(x, y) : x \geq 0, y \geq 0, x^2 + y^2 = 9\}, \\ \vec{n} \cdot \text{grad } u &= 0 \quad \text{na } \Gamma_3 = \{(x, y) : 1 \leq x \leq 3, y = 0\} \text{ a } \Gamma_4 = \{(x, y) : x = 0, 1 \leq y \leq 3\}. \end{aligned}$$

Řešení: Spíše několik poznámek a tipů:

Pozor na okrajovou podmínku zadanou na Γ_3 a Γ_4 . Symbolem $\cdot$ se zde nemyslí obyčejné násobení, ale skalární součin. Podmínka mohla být též zapsána jako $\partial u / \partial \vec{n} = 0$. Jedná se o homogenní Neumannovu okrajovou podmínku, které se též říká podmínka kolmosti. Až budete mít řešení, nechte si zobrazit vrstevnice (contours) a pokuste se odhadnout, proč se v souvislosti s touto podmínkou zmiňuje zrovna kolmost.

Při zadávání této podmínky si můžete všimnout, že v Matlabu bude očekáván tvar `n*c*grad(u)+qu=g`. Písmenem $\vec{n}$ se myslí normála $\vec{n}$, takže nezadáme nic. Funkce c je tatáž jako v rovnici (viz (11.3)) a zadáme ji (nebo spíš necháme nastavenou na 1) až při zadávání rovnice. Jediné, co musíme zadat přímo zde, jsou funkce q a g .

Pravá strana rovnice je poněkud komplikovaná, ale není to ze zlomyslnosti, spíš k tomu vedla snaha o to, aby výsledný obrázek byl zajímavější než u řešeného příkladu. Pozor na správný zápis operátorů - nezapomeňte na patřičné místo napsat tečku. Funkce $\arctg$ se v Matlabu zadává jako `atan`, ne třeba `arctan`! Z toho, že zlomek y/x na části hranice oblasti Ω není definován, si nemusíte dělat hlavu - Matlab si s tím poradí, zvlášť, když se tento zlomek dále dosazuje do funkce `arctg`.

Reference

- [1] J. Diblík, J. Baštinec, I. Hlavičková, Z. Šmarda: *Diferenciální rovnice a jejich použití v elektrotechnice*, Brno, FEKT VUT. (elektronický učební text)
- [2] R. S. Esfandiari: *MATLAB Manual for Advanced Engineering Mathematics*, Atlantis Publishing Company, 2008.
- [3] D. Hanselman, B. Littlefield: *Mastering MATLAB 7*, Pearson Education, Inc., 2005.
- [4] B. R. Hunt, R. L. Lipsman, J. M. Rosenberg, K. R. Coombes, J. E. Osborn, G. J. Stuck: *A Guide to MATLAB for Beginners and Experienced Users*, Cambridge University Press, 2009.
- [5] Documentation for MathWorks Products,
http://www.mathworks.com/access/helpdesk_r13/help/helpdesk.html