INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Numerické výpočty III

Jiří Zelinka

Tento učební text vznikl za přispění Evropského sociálního fondu a státního rozpočtu ČR prostřednictvím Operačního programu Vzdělávání pro konkurenceschopnost v rámci projektu Univerzitní výuka matematiky v měnícím se světě (CZ.1.07/2.2.00/15.0203).

Obsah

Úvod

V rámci předmětu Numerické výpočty III bude hlavní pozornost věnována numerickému řešení nelineárních a diferenciálních rovnic. Z toho důvodu budeme pro příklady používat hlavně Matlab, který je pro tyto účely velmi vhodný a obsahuje řadu zpracovaných algoritmů vhodných pro dané téma.

Kromě uvedených metod si také ukážeme, jak s pomocí počítače řešit úlohy pro funkce více proměnných – zda přijde ke slovu Sage, abychom se vrátili k Matlabu při zpracování statistických problémů.

Pokud by se čtenář chtěl seznámit s porblematikou hlouběji, může se zaměřit na publikace [2], [3] nebo [1].

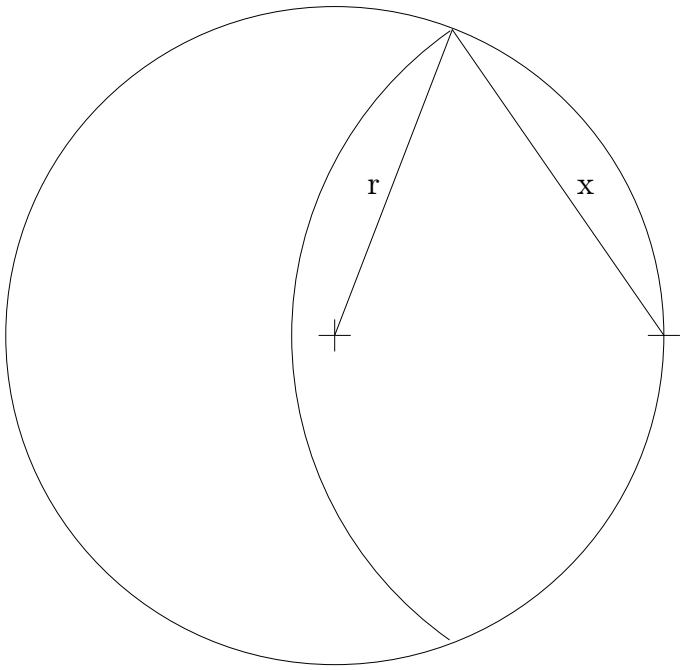
Kapitola 1

Řešení nelineárních rovnic

Tak studenti, dneska si povíme o důležité aplikaci Banachovy věty o pevném bodě v numerické matematice. Copak, Kropáčku, něco se vám nezdá?
Ale, jenom mne tak napadlo, pane profesore, že je škoda, že se toho nedožil Archimedes.

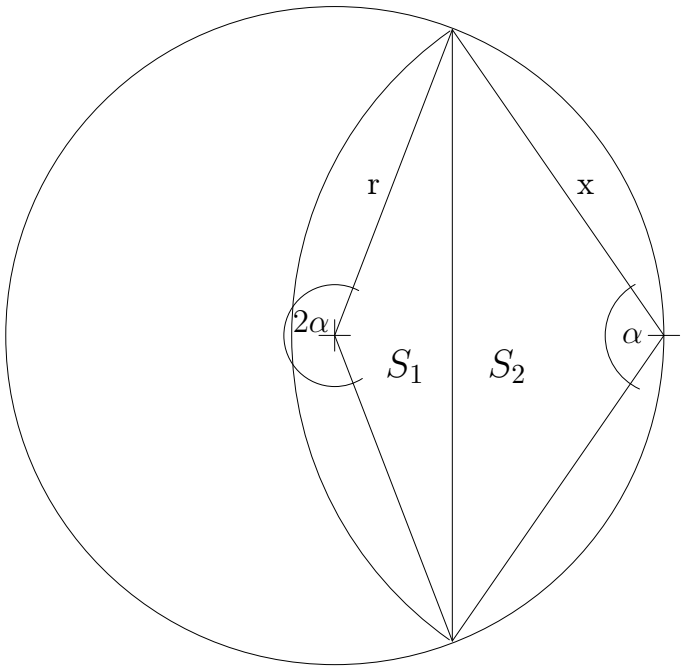
1.1 Motivační úloha

Začneme geometrickou úlohou s jednoduchým zadáním. Mějme kruh v rovině o zadaném poloměru r. Máme z hranice kruhu udělat kruhový oblouk o neznámém poloměru x tak, aby část ohraničená oběma křivkami měla plochu, která je rovna polovině plochy původního kruhu. Situaci si můžeme znázornit na obrázku:



Tato úloha je matematickým vyjádřením úlohy ze zemědělství. Sedlák má kruhový pozemek, na kterém se pase koza. Protože sedlák chce, aby jí tráva na pozemku vystačila na dva dny, uváže ji ke kůlu na okraji pozemku tak dlouhým provazem, aby za první den spásla polovinu trávy. Druhý den ji nechá k dispozici celý pozemek, kde může spást zbylou trávu.

Pro řešení úlohy si do obrázku doplníme několik údajů.



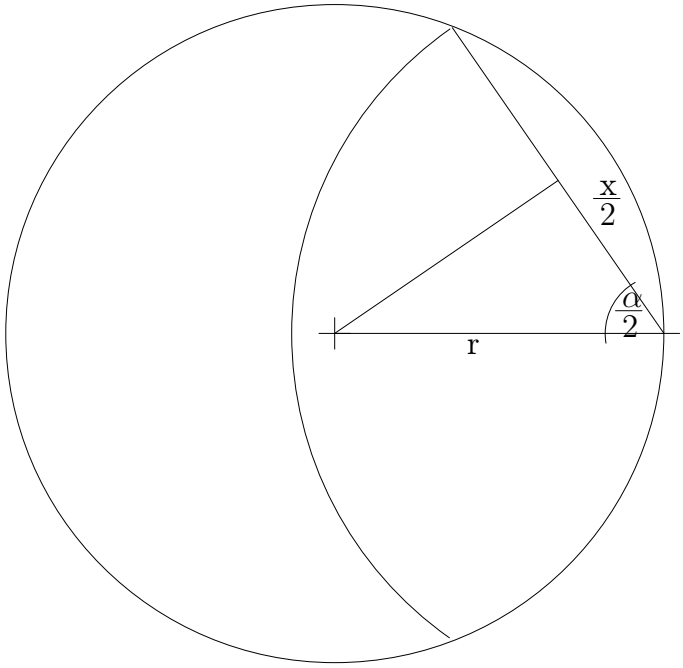
Spojíme střed kružnice i oblouku s průsečíky na kružnici. Průvodiče vycházející od kraje kružnice svírají úhel α, průvodiče vycházející ze středu kružnice tedy svírají úhel 2α. Oblast, která nás zajímá, se skládá ze dvou kruhových úsečí, první z nich má obsah S_1 a je dána poloměrem x a úhlem α, druhá má obsah S_2, poloměr r a úhel $2\pi - 2\alpha$. Označíme-li obsah celého kruhu S, dostáváme

$$S_1 + S_2 = \frac{1}{2}S.$$

Obsah kruhové úseče určíme tak, že od obsahu kruhové výseče odečte obsah trojúhelníka o stranách rovných poloměru výseče,jež svírají úhel, jež určuje výseč. Tedy

$$S_1 = \frac{1}{2}x^2(\alpha - \sin\alpha), \qquad S_2 = \frac{1}{2}r^2((2\pi - 2\alpha) - \sin(2\pi - 2\alpha))$$

Ještě potřebujeme vztah mezi r a x. K tomu můžeme použít pravoúhlý trojúhelník, který dostaneme, když spojíme střed kružnice se středem tětivy délky x podle následujícího obrázku:



V uvedeném obrázku platí

$$\frac{x/2}{r} = \cos\frac{\alpha}{2},$$

tedy

$$x = 2r\cos\frac{\alpha}{2}.$$

Můžeme tedy namísto x hledat úhel α, z předchozího vztahu pak x snadno vypočítáme.

Nyní už máme všechny potřebné vztahy pohromadě a použitím goniometrických vzorečků (toto cvičení jistě čtenář snadno zvládne) získáme rovnici

$$\alpha = \tan\alpha - \frac{\pi}{2\cos\alpha}.$$

Přesné řešení této rovnice ovšem získat neumíme. Nezbývá, než použít nějakou numerickou metodu, kterýmžto se budou věnovat následující řádky.

1.2 Základní metody řešení nelineární rovnice

Budeme se zabývat hledáním řešení rovnice

$$f(x) = 0, \qquad x \in I = [a, b] \tag{1.1}$$

pro spojitou funkci f. To že řešení na uvedeném intervalu existuje nám zaručí například podmínka $f(a) \cdot f(b) \leq 0$, tedy v krajních bodech intervalu I má funkce f opačná znaménka. Řešení ovšem může být více. Označme řešení rovnice jako ξ, nazýváme jej také kořenem funkce f.

Všechny metody a tvrzení, které zde budou vedeny, může čtenář nalézt v detailním znění ve skriptech [2].

1.2.1 Půlení intervalu

Nejjednodušší metodou pro nalezení řešení je metoda půlení intervalu, zvané též metoda bisekce. Její myšlenka je velmi jednoduchá: jestliže máme splněnu podmínku $f(a) \cdot f(b) \leq 0$, rozpůlíme interval $[a, b]$ a jako nový interval vezmeme ten, pro který platí, že v jeho krajních bodech má funkce f opět opačná znaménka, tedy tento poloviční interval opět obsahuje řešení rovnice. Pokračujeme s půlením tak dlouho, dokud nedostaneme interval dostatečné malý (stále obsahující řešení), tedy máme přibližné řešení s požadovanou přesností. Jako přibližné řešení $\bar{\xi}$ stanovíme střed konečného intervalu.

U metody půlení intervalu je možné dokonce předem určit, kolik iterací budeme potřebovat. Jestliže například požadujeme, aby chyba přibližného řešení byla menší než δ, dostáváme pro počet kroků k nerovnost

$$\frac{b-a}{2^{k+1}} \leq \delta$$

odkud logaritmování při základu 2 dostáváme

$$k \geq \log_2 \frac{b-a}{2\delta}.$$

Metoda půlení intervalu má jednu základní výhodu, že totiž za uvedených předpokladů vždy konverguje. Její nevýhodou je, že konverguje poměrně pomalu. A jelikož jsme v podstatě vyčerpali všechny důležité informace o ní, budeme se věnovat metodám dalším.

1.2.2 Prostá iterační metoda

U této metody se zaměříme na rovnici v jiném tvaru:

$$x = g(x),$$

hledáme tedy hodnotu ξ, kterou funkce g zobrazí samu na sebe. Proto se bod ξ nazývá pevný bod funkce g.

Pro funkci g samozřejmě požadujeme spojitost na intervalu $I = [a, b]$. Pro existenci pevného bodu na tomto intervalu pak postačuje, aby se interval I zobrazil sám do sebe, tedy $\forall x \in I : g(x) \in I$. Tato podmínka ovšem nezaručí jednoznačnost pevného bodu. Pro ni potřebujeme, aby funkce g byla kontrakcí na intervalu I, to jest musí existovat konstanta L, $0 \leq L < 1$, že pro každé $x, y \in I$ platí

$$|g(x) - g(y)| \leq L \cdot |x - y|.$$

V případě platnosti této podmínky nám jednoznačnost pevného bodu zaručí Banachova věta o pevném bodě, kterou mohl čtenář náhodou zaslechnout na přednáškách věnovaných metrickým prostorům. Konstanta L, která v podmínce vystupuje, se nazývá Lipschitzova konstanta, a funkce, které tuto podmínku splňují (i bez omezení $L < 1$), se nazývají lipschitzovsky spojité.

Pakliže je funkce g kontrakcí na intervalu I, platí navíc, že pokud vezmeme libovolnou počáteční aproximaci $x_0 \in I$ a sestrojíme posloupnost rekurentně předpisem

$$x_1 = g(x_0), \ldots, x_{k+1} = g(x_k), \ldots,$$

bude tato posloupnost konvergovat k pevnému bodu ξ funkce g.

Tato vlastnost je zásadní pro hledání přibližného řešení rovnice. Ověření Lipschitzovské spojitosti s konstantou $L < 1$ by ovšem bylo poněkud komplikované. Naštěstí pro funkce, které mají na intervalu I spojitou derivaci je tato vlastnost zaručena, pokud pro každé $x \in I$ platí $|g'(x)| \leq L < 1$, což vyplývá bezprostředně z Lagrangeovy věty o střední hodnotě.

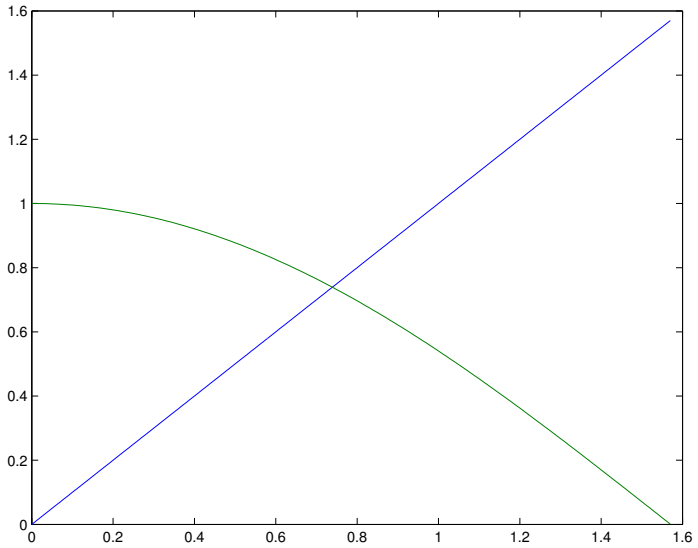
Protože výpočet posloupnosti je dán jednoduchým předpisem $x_{k+1} = g(x_k)$, nazývá se metoda založená na Banachově větě o pevném bodě *prostou iterační metodou*.

Ukažme si na jednoduchém příkladu postup ověření uvedených podmínek a nalezení přibližného řešení rovnice:

Příklad 1. Mejme rovnici

$$x = \cos x.$$

Pokud si problém představíme graficky, hledáme v podstatě průsečík grafu funkce $g(x) = \cos x$ s přímkou $y = x$ (viz obrázek):



Nejprve je potřeba najít interval, který funkce g zobrazí do sebe. Snadno zjistíme, že takovým intervalem je například interval $I = [0, \pi/2]$. Na tomto intervalu je navíc funkce $\cos x$ klesající, takže stačí ověřit, že se do intervalu I zobrazí jeho krajní body.

Dále musíme ověřit, zda na daném intervalu je maximu derivace funkce g v absolutní hodnotě ostře menší než jedna. Pokud tomu tak bude, stačí zvolit jako konstantu L ono maximum absolutní hodnoty derivace. Zde ovšem narazíme na problém – maximum je v pravém krajním bodě intervalu a je rovno jedné. Musíme tedy interval o něco zkrátit a to z obou stran, aby se i po zkrácení zobrazovala do sebe.

Vhodnou volbou (nikoliv jedinou možnou) je $I = [\cos 1.5, 1.5)$, pak $L = \sin 1.5 \doteq 0.9975$. Pokud pak budeme hledat přibližné řešení například v Matlabu, stačí zvolit libovolnou počáteční aproximaci a pak počítat jednoduchým způsobem další aproximace, dokud nedostaneme dostatečné přesné přibližné řešení. Takto jednoduché výpočty můžeme zadávat i ručně, napsat si jednoduchý program by čtenář jistě také zvládl.

```
>> presnost=0.0001;
>> x=1;
```

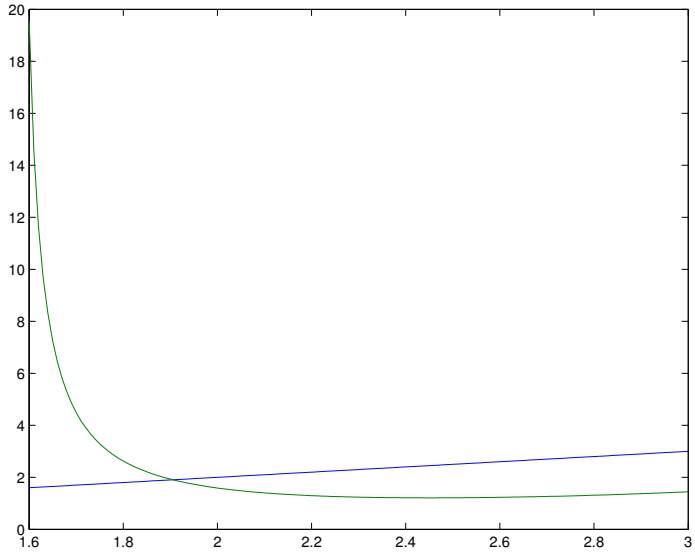
```
>> while abs(x-cos(x))>presnost, x=cos(x), end
x =
    0.5403
x =
    0.8576
x =
    0.6543
x =
    0.7935
x =
    0.7014
  .
  .
  .
x =
    0.7392
x =
    0.7390
x =
    0.7391
>>
```

Na dosažení požadované přesnosti je potřeba něco přes dvacet iterací.

Pokud bychom chtěli použít prostou iterační metodu na řešení úvodního příkladu pro rovnici

$$x = \tan x - \frac{\pi}{2\cos x},$$

tedy pro funkci $g(x) = \tan x - \frac{\pi}{2\cos x}$, můžeme si vypomoci obrázkem, abychom mohli lépe odhadnout interval, kde se pevný bod nachází. Při tom využijeme toho, že musí ležet někde v intervalu $[\pi/2, \pi]$, který je třeba o něco zkrátit, jelikož v levém krajním bodě jde funkce g do nekonečna.



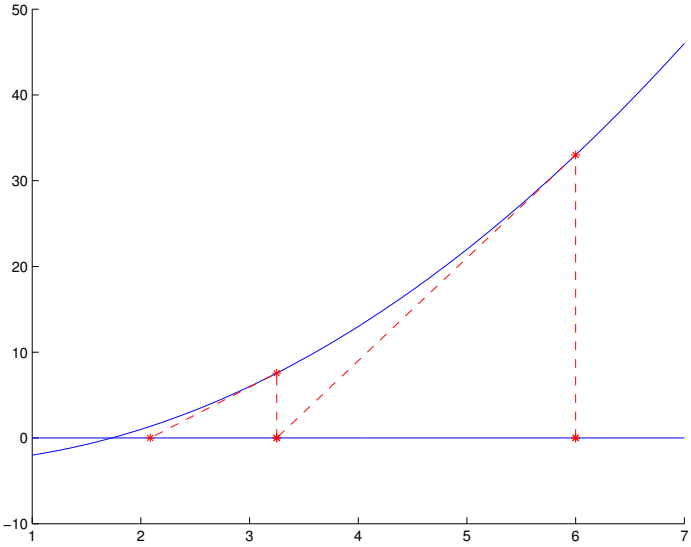
Zjistíme, že pevný bod leží v intervalu $[1.8, 2]$. Na tomto intervalu ale použít prostou iterační metodu není možné, neboť absolutní hodnota derivace funkce g je zde větší než jedna.

1.2.3 Newtonova metoda

Vrátíme se nyní zpět k rovnici

$$f(x) = 0.$$

Na řešení této rovnice můžeme nahlížet jakožto na průsečík grafu funkce f s osou x. Myšlenka Newtonovy metody je prostá. Zvolíme počáteční iteraci x_0 a další iterací je průsečík tečny ke grafu s osou x. V podstatě hledáme přibližné řešení na lineární aproximaci funkce f. Tento postup opakujeme tak dlouho, dokud nemáme přibližné řešení s dostatečnou přesností. Metodu si můžeme dobře ilustrovat graficky:



Je zřejmé, že v tomto případě kromě spojitosti funkce f potřebujeme také spojitost její derivace. Vyjádříme-li si z rovnice tečny k f v bodě $[x_k.f(x_k)]$ její průsečík z osou x, dostáváme iterační vztah

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Stejný vztah bychom obdrželi použitím Taylorova rozvoje, v němž bychom zanedbali členy od druhé derivace včetně.

Máme vlastně opět prostou iterační metodu, tentokrát pro funkci g ve speciálním tvaru

$$g(x) = x - \frac{f(x)}{f'(x)},$$

můžeme tedy opět zkoumat podmínky, za nichž je tato funkce kontrakcí. Obecné vyjádření je ovšem složité, platí ale tvrzení, že pokud je derivace funkce f nenulová na intervalu obsahujícím řešení rovnice, pak existuje okolí tohoto řešení, na němž je funkce g kontrakcí, tedy pro libovolnou počáteční aproximaci z tohoto okolí Newtonova metoda konverguje.

Toto kriterium vypadá v praxi jen špatně použitelné, pokud se týče hledání počáteční aproximace. Naštěstí existují jiné podmínky, snadněji ověřitelné, umožňující její volbu, jsou to například Fourierovy podmínky:

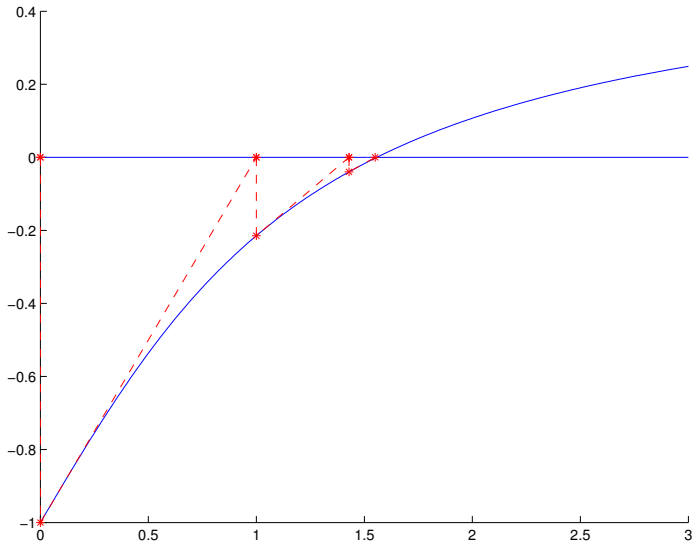
1. Funkce f má na intervalu I spojité druhé derivace a v krajních bodech I opačná znaménka.

2. Derivace funkce f je nenulová na celém intervalu, tedy ani nemění znaménko.

3. Druhá derivace funkce f nemění na I znaménko.

Při splnění těchto podmínek Newtonova metoda konverguje na I, a to dokonce monotonně, pakliže jako počáteční iteraci zvolíme ten krajní bod intervalu, v němž má funkční hodnota stejné znaménko jako je znaménko druhé derivace.

Uvedené podmínky vlastně říkají, že funkce f je na intervalu I ryze monotonní (rostoucí nebo klesající) a to buď konvexní nebo konkávní. Celkem jsou tedy čtyři případy, následující obrázek ukazuje situaci, kdy je f rostoucí a konkávní. V tom případě volíme jako počáteční aproximaci ten krajní bod, v němž je funkční hodnota záporná.



Další tři možnosti by si čtenář jistě dokázal snadno představit.

Příklad 2. Zkusme požít Newtonovu metodu na úvodní příklad, tedy na rovnici

$$x = \tan x - \frac{\pi}{2\cos x}.$$

Nejprve všechny členy převedeme na jednu stranu, abychom dostali funkce f:

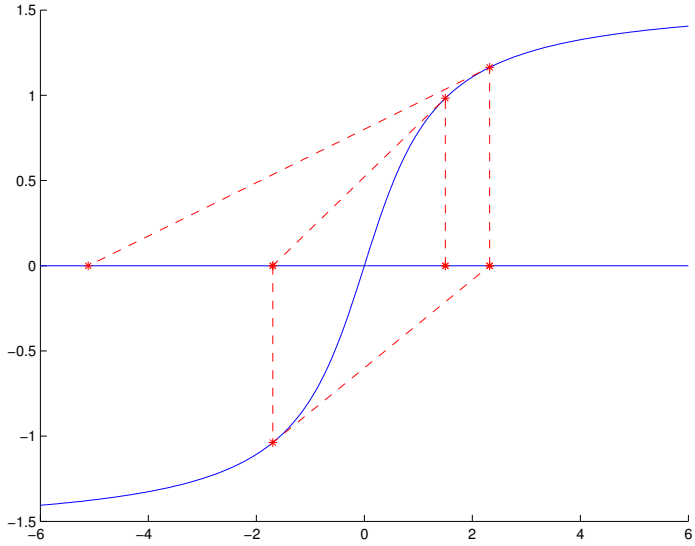
$$x - \tan x + \frac{\pi}{2\cos x} = 0$$

Pak zapojíme Matlab, přičemž použijeme toho, že z už známe interval, kde řešení leží.

```
>> f=sym('x-tan(x)+pi/(2*cos(x))')
f =
x - tan(x) + pi/(2*cos(x))
>> df=diff(f)
df =
(pi*sin(x))/(2*cos(x)^2) - tan(x)^2
>> x=1.8;
>> x=x-eval(f)/eval(df)
x =
    1.8735
>> x=x-eval(f)/eval(df)
x =
    1.9028
>> x=x-eval(f)/eval(df)
x =
    1.9057
>> x=x-eval(f)/eval(df)
x =
    1.9057
>>
```

Vidíme, že aproximaci řešení na čtyři desetinná místa jsem nalezli velmi rychle. Mohli bychom samozřejmě nastavit větší přesnost a počítat dál, ale i tak bychom dosáhli během několika dalších iterací nejlepší možné aproximace v rámci počítačové přesnosti.

Vypadá to tedy, že Newtonova metoda je velmi rychlá, i při jejím použití ovšem nesmíme zapomínat na jistou míru bedlivost. zejména co se týče volby počáteční aproximace. To si můžeme dobře demonstrovat na jednoduchém příkladě – totiž na funkci $f(x) = \arctan x$. Zvolíme-li jako počáteční aproximaci hodnotu 1.5, iterační posloupnost nekonverguje, jak naznačuje obrázek. Volbou $x_0 = 1$ by vše bylo v pořádku.



1.2.4 Metoda sečen a regula falsi

Další metody, o nichž si něco řekneme, jsou podobné. Jejich hlavní myšlenka spočívá v náhradě derivace v bodě x_k. To je nutné v případech, že derivaci nelze vyjádřit, například když hodnoty funkce f nejsou dány nějakým vzorcem, ale jsou to třeba výsledky fyzikálního měření. Derivaci pak nahradíme poměrnou diferencí, potřebujeme ovšem funkční hodnoty ve dvou bodech:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Použijeme-li tuto náhradu v iteračním vztahu pro Newtonovu metodu, dostáváme

$$x_{k+1} = x_k - f(x_k) \cdot \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}.$$

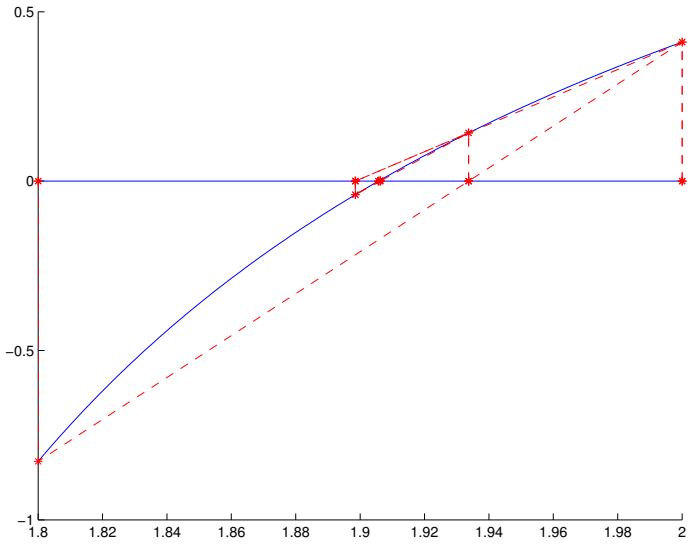
Metoda daná tímto iteračním vztahem se nazývá *metoda sečen*. Tento název je odvozen z toho, že tečná z Newtonovy metody je nahrazena sečnou protínající graf funkce f ve dvou bodech.

Z iteračního vztahu metody sečen také plyne, že na jejím začátku potřebujeme dvě iterace – x_0 a x_1. Také v každém kroku používáme dvě funkční hodnoty, stačí ale počítat jen jednu a tu druhou si pamatovat od minule. Vyzkoušejme si opět metodu sečen v Matlabu na úvodním příkladu:

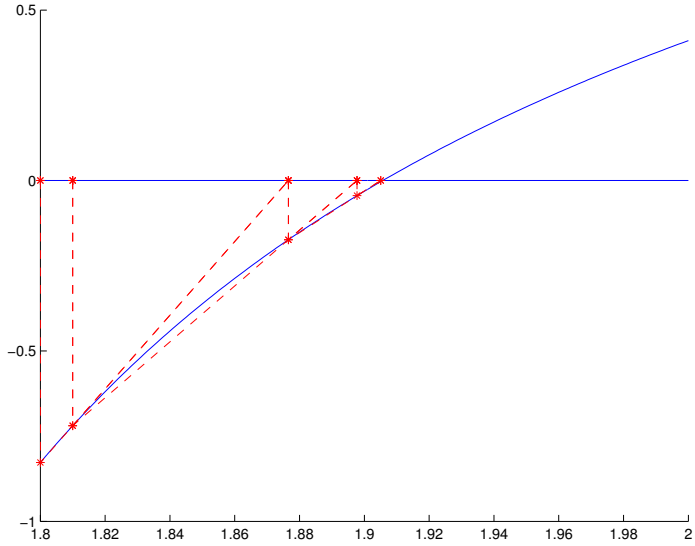
```
>> f=inline('x-tan(x)+pi/(2*cos(x))');
>> x0=1.8;
>> x1=2;
>> f0=f(x0)
f0 =
   -0.8274
>> f1=f(x1)
f1 =
    0.4104
>> x2=x1-f1*(x1-x0)/(f1-f0)
x2 =
    1.9337
>> f2=f(x2)
f2 =
    0.1422
>> x3=x2-f2*(x2-x1)/(f2-f1)
x3 =
    1.8985
>> f3=f(x3)
f3 =
   -0.0401
>> x4=x3-f3*(x3-x2)/(f3-f2)
x4 =
    1.9063
>> f4=f(x4)
f4 =
    0.0031
>> x5=x4-f4*(x4-x3)/(f4-f3)
x5 =
    1.9057
>> f5=f(x5)
f5 =
   6.1196e-05
>> x6=x5-f5*(x5-x4)/(f5-f4)
x6 =
    1.9057
>> f6=f(x6)
```

```
f6 =
  -9.5082e-08
>>
```

Počítat tímto způsobem je jistě poněkud nepraktické, zde to slouží jen jako ukázka. Čtenář by jistě zvládl celý problém řešit elegantněji, například pomocí cyklu. Grafické znázornění metody sečen ukazuje následující obrázek:



Není ovšem potřeba volit počáteční aproximace tak, aby funkční hodnoty měly opačná znaménka. Naopak, pokud budou blízko sebe, bude příslušná sečna bližší tečně z Newtonovy metody:

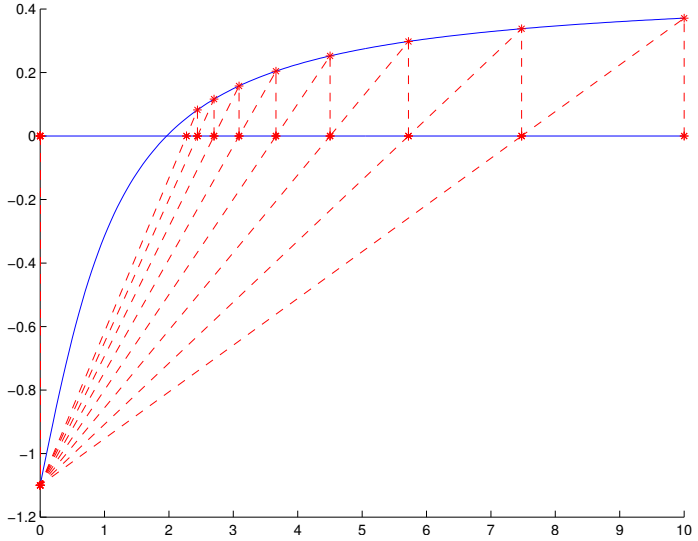


U metody *regula falsi* naopak počáteční aproximace z opačnými funkčními hodnotami požadujeme a tento požadavek pak platí pro všechny další iterace. Jedná se o podobnou myšlenku jako u metody půlení intervalu, iterace ale počítáme podobně jako u metody sečen:

$$x_{k+1} = x_k - f(x_k) \cdot \frac{x_k - x_s}{f(x_k) - f(x_s)},$$

kde s je největší index menší než k takový, že $f(x_s) \cdot f(x_k) \leq 0$.

Pro tuto metodu je typické, že pokud je funkce f na intervalu I konvexní nebo konkávní, jeden z bodů počátečních aproximací je tzv. *fixní*, tedy index s je pořád stejný, jak vidíme i z následujícího obrázku:



1.3 Rychlost konvergence

Z uvedených příkladů vidíme, že rychlost konvergence se liší u různých metod. Proto si definujme *řád konvergence* iterační metody, který udává rychlost její konvergence:
Nechť posloupnost (x_k) je iterační posloupnost získaná nějakou metodou, $x_k \to \xi$. Označme e_k chybu v k-tém kroku, t.j. $e_k = x_k - \xi$. Řekneme, že iterační metoda je řádu $p \geq 1$, jestliže platí

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C \neq 0.$$

Řád konvergence vlastně říká, jak musíme umocnit chybu v k-tém kroku, abychom limitně dostali chybu v kroku $k+1$ (až na násobení konstatnou). Je jasné, že čím vyšší je řád metody, tím je konvergence rychlejší. Například pro $p = 1$ (říkáme také, že metoda je lineárního řádu) je přechod mezi jednotlivými chybami jen přibližně násobení konstantou (ta tedy musí být menší než 1), kdežto pro metodu řádu 2 (kvadratického řádu) platí, že pokud je chyba v některém kroku v desetinách, v dalším bude v setinách, v následujícím už v desetitisícinách atd.

Pro metody, které jsem si ukazovali je řád znám. Bisekce je lineární, Newtonova metoda kvadratická, metoda sečen má řád roven zlatému řezu $\frac{1+\sqrt{5}}{2}$, metoda regula falsi je opět jen lineární. Pro prostou iterační metodu s

funkcí g platí, že pokud g má spojité derivace do řádu p včetně, ξ je pevný bod funkce a $g'(\xi) = \ldots = g^{(p-1)}(\xi) = 0$, $g^{(p)}(\xi) \neq 0$, pak metoda je řádu p.

Je známa metoda, pomocí níž se dá rychlost konvergence urychlit. Jedná se o Aitkenovu δ^2 metodu, která je založena na tom, že pokud by iterační posloupnost konvergovala přesně lineárně, je možné pomocí jednoduchého výpočtu ze tří členů určit limitu.

Mějme tedy posloupnost (x_k). Z ní sestrojíme posloupnost $(\hat{x}_k)$ s použitím vztahu

$$\hat{x}_k = x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}.$$

Platí, že pokud původní posloupnost konverguje k ξ, konverguje k této hodnotě i nová posloupnost. Dále pokud je řád konvergence původní posloupnosti lineární, je nová posloupnost řádu alespoň kvadratického, je li řád původní posloupnosti $p > 1$, je řád nové posloupnosti $2p-1$. Tento postup tedy urychlí i Newtonovu metodu.

1.3.1 Steffensenova metoda

Aitkenovu metodu nelze použít tak, jak je uvedena, jelikož bychom potřebovali znát celou původní posloupnost, dokázali bychom tedy odhadnout i její limitu, nebylo by tedy potřeba počítat ji znovu, byť rychleji. Nicméně kombinací Aitkenovy metody s prostou iterační metodou získáme Steffensenovu metodu, která zpravidla dává rychle konvergující posloupnost. Myšlenka Steffensenovy metody je prostá: určíme tři iterace prostou iterační metodou (počáteční iteraci a dvě další) a pak použijeme Aitkenovu metodu na urychlení výpočtu. Pak určíme další tři iterace prostou iterační metodou a zase provedeme urychlení, což opakujeme, dokud nedosáhneme požadované přesnosti. Celý postup můžeme zapsat třeba takto (pro počáteční iteraci x_0):

$$y_0 = g(x_0), \quad z_0 = g(y_0), \quad x_1 = x_0 - \frac{(y_0 - x_0)^2}{z_0 - 2y_0 + x_0}$$

a obecný krok je pak

$$y_k = g(x_k), \quad z_k = g(y_k), \quad x_{k+1} = x_k - \frac{(y_k - x_k)^2}{z_k - 2y_k + x_k}.$$

Steffensenova metoda konverguje pro počáteční iteraci z nějakého okolí pevného bodu ξ funkce g, jestliže $g'(\xi) \neq 1$.

Příklad 3. Vyzkoušíme Steffensenovu metodu na řešení úvodního příkladu pro funkci $g(x) = \tan x - \frac{\pi}{2\cos x}$. Jako počáteční iteraci zvolíme $x_0 = 1.8$ a výpočet provedeme v Matlabu:

```
>> g=inline('tan(x)-pi/(2*cos(x))');
>> x=1.8;
>> y=g(x),z=g(y)
y =
    2.6274
z =
    1.2392
>> x=x-(x-y)^2/(z-2*y+x)
x =
    2.1090
>> y=g(x),z=g(y)
y =
    1.3894
z =
   -3.2542
>> x=x-(x-y)^2/(z-2*y+x)
x =
    2.2410
>> y=g(x),z=g(y)
y =
    1.2672
z =
   -2.0623
>> x=x-(x-y)^2/(z-2*y+x)
x =
    2.6435
>> y=g(x),z=g(y)
y =
    1.2442
z =
   -1.9440
>> x=x-(x-y)^2/(z-2*y+x)
x =
    3.7379
```

```
>>
```

Vidíme, že nevhodně zvolená počáteční iterace ke konvergenci nevede. Zkusíme počáteční iteraci poněkud zpřesnit:

```
>> x=1.85;
>> y=g(x),z=g(y)
y =
    2.2117
z =
    1.2865
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9517
>> y=g(x),z=g(y)
y =
    1.7283
z =
    3.7177
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9291
>> y=g(x),z=g(y)
y =
    1.8087
z =
    2.5417
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9121
>> y=g(x),z=g(y)
y =
    1.8775
z =
    2.0449
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9062
```

```
>> y=g(x),z=g(y)
y =
    1.9034
z =
    1.9159
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9057
>> y=g(x),z=g(y)
y =
    1.9057
z =
    1.9058
>> x=x-(x-y)^2/(z-2*y+x)
x =
    1.9057
>>
```

1.4 Řešení systému nelineárních rovnic

Mějme systém nelineárních rovnic

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= 0 \end{aligned}$$

Můžeme jej stručně zapsat ve tvaru

$$F(\mathbf{x}) = \mathbf{0},$$

kde F i $\mathbf{x}$ bereme jako vektory. Nejpožívanější metodou je Newtonova metoda, kterou dostaneme tak, že vezmeme Taylorův rozvoj funkce více proměnných do prvního řádu.

Označme J_F matici derivací vektoru funkcí F, tedy

$$J_F(\mathbf{x}) = \left(\frac{\partial f_i(\mathbf{x})}{x_j} \right).$$

Pak Newtonovu metodu můžeme vyjádřit podobným způsobem jako je to pro funkci jedné proměnné:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_F^{-1}(\mathbf{x_k}) \cdot F(\mathbf{x_k}).$$

Protože ale práci s funkcemi více proměnných jsme zatím v Matlabu ani v Sage nezkoušeli, odložíme prozatím praktické vyzkoušení Newtonovy metody.

Kapitola 2

Funkce více proměnných

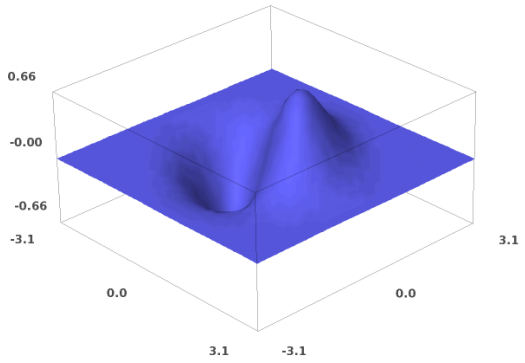
Kryšpín Kropáček: Matematický boj
Bojovat jsem začli směle s funkcí více proměnných.
Výsledky prý byly skvělé, skoro všechny – kromě mých.

2.1 Zobrazovnání grafů

První věcí, kterou si ukážeme, či spíše zopakujeme, pro funkce více proměnných je nakreslení jejich grafu. To samozřejmě dokážeme jen pro funkce dvou proměnných. V Sage můžeme postupovat třeba takto:

```
sage: x,y = var('x,y')
sage: plot3d(sin(x+y)*exp(-(x^2+y^2)/2),(-pi,pi),(-pi,pi))
```

Tím dostaneme následující obrázek:

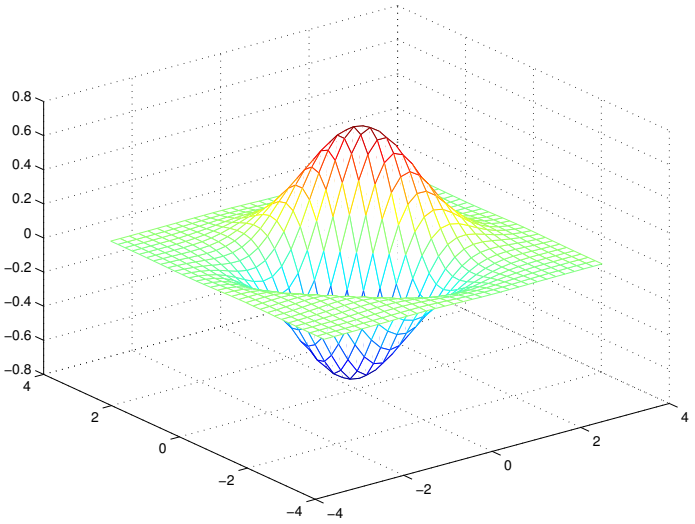


Získat podobný obrázek v Matlabu dá ovšem o něco víc práce. Nejprve musíme definovat síť bodů, v nichž se má funkce vyčíslit. K tomu se hodí matlabovská funkce `meshgrid`. Ukážeme její použití i výsledek, který dává:

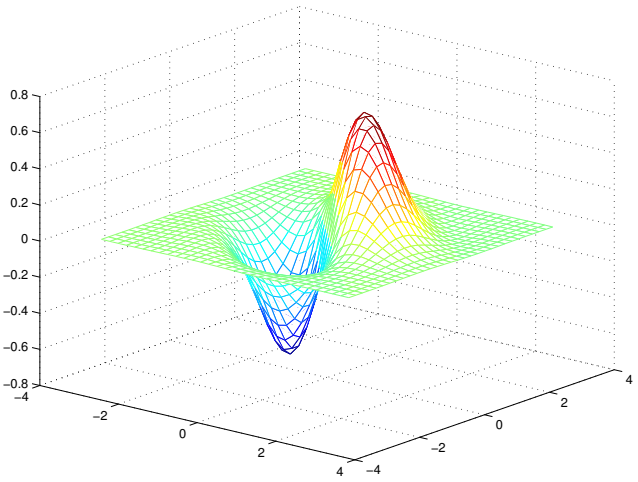
```
>> x=linspace(-pi,pi,31);
>> y=linspace(-pi,pi,31);
>> [X,Y]=meshgrid(x,y);
>> X(1:5,1:5)
ans =
   -3.1416   -2.9322   -2.7227   -2.5133   -2.3038
   -3.1416   -2.9322   -2.7227   -2.5133   -2.3038
   -3.1416   -2.9322   -2.7227   -2.5133   -2.3038
   -3.1416   -2.9322   -2.7227   -2.5133   -2.3038
   -3.1416   -2.9322   -2.7227   -2.5133   -2.3038
>> Y(1:5,1:5)
ans =
   -3.1416   -3.1416   -3.1416   -3.1416   -3.1416
   -2.9322   -2.9322   -2.9322   -2.9322   -2.9322
   -2.7227   -2.7227   -2.7227   -2.7227   -2.7227
   -2.5133   -2.5133   -2.5133   -2.5133   -2.5133
   -2.3038   -2.3038   -2.3038   -2.3038   -2.3038
>> size(X)
ans =
    31    31
>>
```

Vidíme, že ve výsledku se opakují vstupní vektory, v jedné výstupní matici po sloupcích, ve druhé po řádcích. Tyto výstupní matice se pak dají použít pro výpočet funkčních hodnot a výsledek si pak zobrazíme:

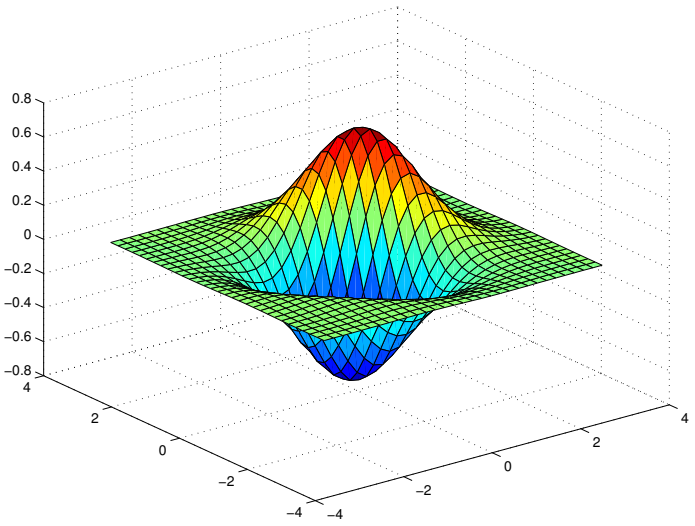
```
>> f=sin(X+Y).*exp(-(X.^2+Y.^2)/2);
>> mesh(x,y,f)
>>
```



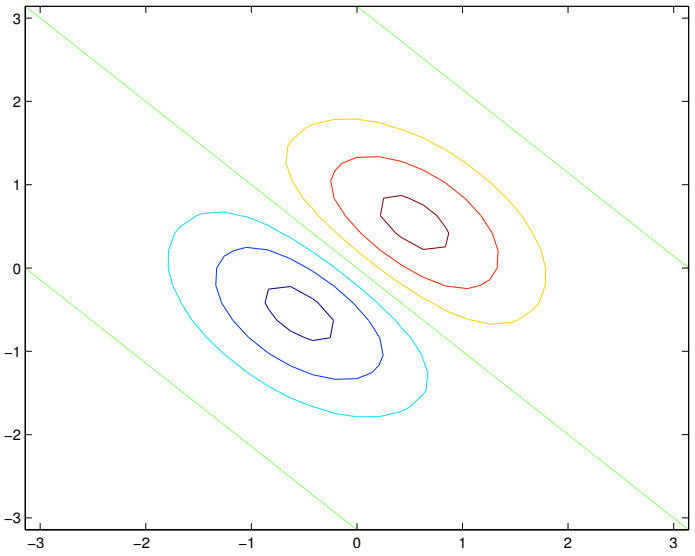
Graf funkce je zobrazen z jiné strany, pomocí ovládacích prostředků Matlabu si jej ovšem můžeme natočit do stejné polohy jako je to na předchozím obrázku.



Kromě funkce `mesh` můžeme také pro zobrazení použít funkci `surf`, která graf funkce f zobrazuje pomocí plošek:

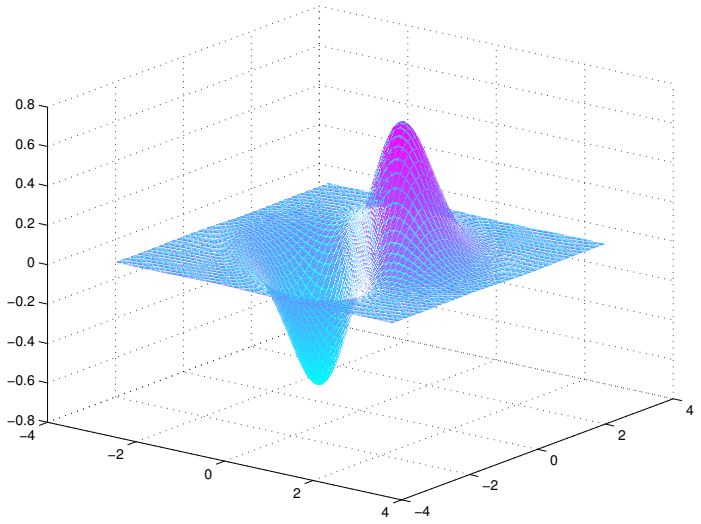


Další možností je zobrazit vrstevnice grafu, k tomu slouží funkce `contour`:



Můžeme samozřejmě graf funkce f zobrazit na jemnější síti, abychom dostali podobný výsledek jako u Sage. Je také možné použít jiné barevné zobrazení, k tomu slouží funkce `colormap`, a nastavit přechody mezi jednotlivými ploškami. Další podrobnosti je možné získat v dokumentaci:

```
>> x=linspace(-pi,pi,51);
>> y=linspace(-pi,pi,51);
>> [X,Y]=meshgrid(x,y);
>> f=sin(X+Y).*exp(-(X.^2+Y.^2)/2);
>> surf(x,y,f)
>> colormap(cool)
>> shading interp
>>
```



2.2 Výpočet parciáních derivací

Výpočet parciálních derivací je v obou našich používaných softwarech velmi jednoduchý. V Sage použijeme následující postup:

```
sage: x,y=var('x,y')
sage: f=sin(x+y)*exp(x-y)
sage: diff(f,x)
e^(x - y)*sin(x + y) + e^(x - y)*cos(x + y)
sage: diff(f,y)
-e^(x - y)*sin(x + y) + e^(x - y)*cos(x + y)
sage:
```

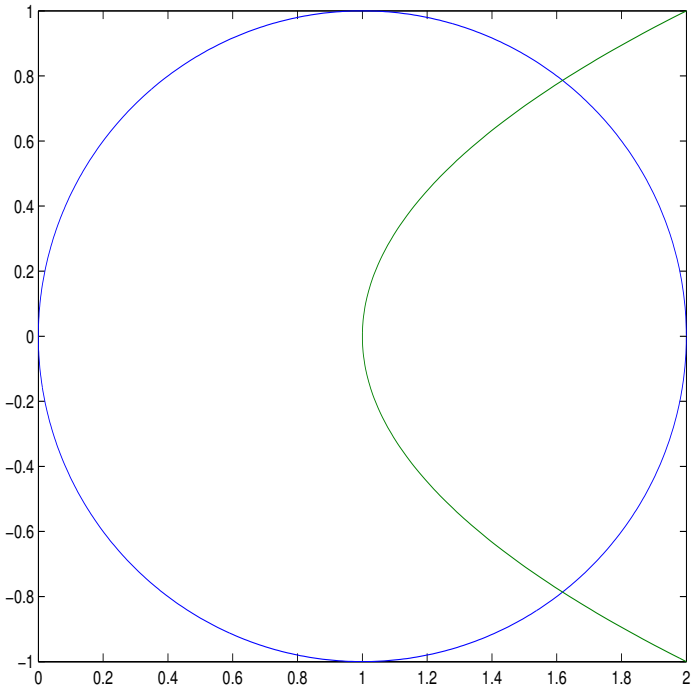
V Matlabu je postup obdobný:

```
>> f=sym('sin(x+y)*exp(x-y)')
f =
exp(x - y)*sin(x + y)
>> diff(f,'x')
ans =
exp(x - y)*cos(x + y) + exp(x - y)*sin(x + y)
>> diff(f,'y')
ans =
exp(x - y)*cos(x + y) - exp(x - y)*sin(x + y)
>>
```

Příklad 4. Nyní si můžeme v Matlabu ukázat, jak bychom řešili systém dvou nelineárních rovnic pomocí Newtonovy metody. Uvažujme rovnice

$$\begin{aligned} f_1(x,y) &= x^2 - 2x + y^2 = 0 \\ f_2(x,y) &= x - y^2 - 1 = 0 \end{aligned}$$

Hledáme vlastně průsečík kružnice a paraboly.



Najít v Matlabu přibližné řešení za pomocí Newtonovy metody můžeme třeba takto:

```
>> f1=sym('x^2-2*x+y^2');
>> f2=sym('x-y^2-1');
>> f1x=diff(f1,'x')
f1x =
2*x - 2
>> f1y=diff(f1,'y')
f1y =
2*y
>> f2x=diff(f2,'x')
f2x =
1
>> f2y=diff(f2,'y')
f2y =
-2*y
>> J=[f1x,f1y;f2x,f2y]
J =
[ 2*x - 2,  2*y]
[       1, -2*y]
>> F=[f1;f2]
F =
 x^2 - 2*x + y^2
   - y^2 + x - 1
>> x=2;y=1; %pocatecni aproximace
>> xy=[x;y]-inv(eval(J))*eval(F)
xy =
    1.6667
    0.8333
>> x=xy(1);y=xy(2);
>> xy=[x;y]-inv(eval(J))*eval(F)
xy =
    1.6190
    0.7881
>> x=xy(1);y=xy(2);
>> xy=[x;y]-inv(eval(J))*eval(F)
xy =
```

```
    1.6180
    0.7862
>> x=xy(1);y=xy(2);
>> xy=[x;y]-inv(eval(J))*eval(F)
xy =
    1.6180
    0.7862
>>
```

Newtonova metoda i v tomto případě konverguje velmi rychle. Zde by samozřejmě bylo možné najít analytické vyjádření řešení, neboť dosazením druhé rovnice do první bychom dostali kvadratickou rovnici. Čtenář tak může učinit pro jeho porovnání s výše získaným přibližným řešením.

2.3 Integrování ve více proměnných

Určit primitivní funkci podle jedné či druhé proměnné s použitím Sage nebo Matlabu je jednoduché:

```
sage: x,y=var('x,y')
sage: f=sin(x+y)*cos(x-y)^2
sage: f1=integral(f,x);f1
-1/12*cos(-3*x + y) + 1/4*cos(-x + 3*y) - 1/2*cos(x + y)
sage: f2=integral(f1,y);f2
-1/12*sin(-3*x + y) + 1/12*sin(-x + 3*y) - 1/2*sin(x + y)
sage:
```

```
>> f=sym('sin(x+y)*cos(x-y)^2');
>> f1=int(f,'y')
f1 =
cos(y - 3*x)/4 - cos(3*y - x)/12 - cos(x + y)/2
>> f2=int(f1,'x')
f2 =
sin(3*x - y)/12 - sin(x - 3*y)/12 - sin(x + y)/2
>>
```

Pak také není problémem spočítat určitý integrál přes obdélníkovou oblast, například pro uvedenou funkci přes $[0, \pi] \times [0, \pi/2]$, přitom by nemělo záležet na pořadí integrování:

```
sage: I1=integral(f,x,0,pi);I1
-1/2*cos(3*y) + 7/6*cos(y)
sage: Ix=integral(f,x,0,pi);Ix
-1/2*cos(3*y) + 7/6*cos(y)
sage: II=integral(Ix,y,0,pi/2);II
4/3
sage:
```

```
>> Iy=int(f,'y',0,pi/2)
Iy =
(4*cos(x))/3 + sin(x)/3 - cos(x)^3 + cos(x)^2*sin(x)
>> II=int(Iy,'x',0,pi)
II =
4/3
>>
```

Jestliže ale potřebujeme spočítat integrál přes jinou oblast, nezbývá, než si meze oblasti vyjádřit ručně, neboť zdá se, že integrovat přes obecné oblasti zatím námi používaný software neumí.

Takže pokus bychom potřebovali určit integrál z funkce $f(x, y) = (x + y)^2$ přes kruh se středem v počátku a poloměrem 2, musíme si pro dané y určit, v jakých mezích se pohybuje x. To je celkem snadné meze jsou $[-\sqrt{4 - y^2}, \sqrt{4 - y^2}]$. Protože ale v mezích je obsaženo ometení pro y, musíme ho zadat i do výpočtu:

```
sage: f=(x+y)^2
sage: assume(abs(y)<=2)
sage: I1=integral(f,x,-sqrt(4-y^2),sqrt(4-y^2));I1
4/3*sqrt(-y^2 + 4)*(y^2 + 2)
sage: I2=integral(I1,y,-2,2);I2
8*pi
sage:
```

U jednoduchých oblastí, jako je například kruh či elipsa, si můžeme nechat hranice spočítat:

```
sage: S=solve(x^2+y^2-4,x);S
[x == -sqrt(-y^2 + 4), x == sqrt(-y^2 + 4)]
sage: I1=integral(f,x,S[0].right(),S[1].right());I1
4/3*sqrt(-y^2 + 4)*(y^2 + 2)
sage: I2=integral(I1,y,-2,2);I2
8*pi
sage:
```

V Matlabu se obejdeme bez předpokladů, nutno ovšem dodat, že výpočet posledního integrálů zde trval daleko delší dobu, než tomu bylo v Sage:

```
>> f=sym('(x+y)^2');
>> ylims=solve('x^2+y^2-4','y')
ylims =
  (2 - x)^(1/2)*(x + 2)^(1/2)
 -(2 - x)^(1/2)*(x + 2)^(1/2)
>> f=sym('(x+y)^2');
>> ylims=solve('x^2+y^2-4','x')
ylims =
  (2 - y)^(1/2)*(y + 2)^(1/2)
 -(2 - y)^(1/2)*(y + 2)^(1/2)
>> I1=int(f,'x',ylims(2),ylims(1))
I1 =
(4*(y^2 + 2)*(2 - y)^(1/2)*(y + 2)^(1/2))/3
>> I2=int(I1,'y',-2,2)
I2 =
8*pi
>>
```

Kapitola 3

Řešení diferenciálních rovnic

Pane profesore, nevím si rady s jedním problémem.
No, jestli je to matematický problém, můžete nás s ním seznámit.
Marně přemýšlím,jakou diferenciální rovnici bych měl použít na popis pohybu diferenciálu auta při průjezdu zatáčkou.

Budeme se zabývat zejména hledáním řešení rovnice ve tvaru

$$y' = f(x, y)$$

pro nějakou funkci f, s případnou počáteční podmínkou $y(x_0) = y_0$ pak hovoříme o počáteční úloze. Zobecněním této úlohy jsou rovnice vyšších řádů, z nichž velkou důležitost mají okrajové úlohy – tedy rovnice druhého řádu se zadanými podmínkami v krajních bodech nějakého intervalu $[a, b]$:

$$y'' = f(x, y, y'), \quad y(a) = y_1, \quad y(b) = y_2$$

3.1 Analytické řešení

Matlab i Sage obsahují nástroje pro nalezení analytického řešení diferenciální rovnice. V Matlabu se používá funkce `dsolve`, které v nejjednodušší podobě stačí jeden vstupní parametr – textový řetězec s rovnicí, kde derivace je označena jako `D`: Základní typy rovnic zvládá bez problémů:

```
>> dsolve('Dy = y')
ans =
```

```
C4*exp(t)
>> dsolve('Dy = -y')
ans =
C2/exp(t)
>> dsolve('Dy = t*y')
ans =
C6*exp(t^2/2)
>> dsolve('Dy = y/t')
ans =
C8*t
>>
```

Je možné použít i rovnici s parametrem a pokud Matlab rovnici nedokáže vyřešit, slušně nám to oznámí:

```
>> dsolve('Dy = a*y*sin(t)')
ans =
C10/exp(a*cos(t))
>> dsolve('Dy = cos(t*y)')
Warning: Explicit solution could not be found.
> In dsolve at 161
ans =
[ empty sym ]
>>
```

Lze také samozřejmě označit jinak výstupní nezávislou proměnnou nebo funkci, kterou hledáme, či zadat počáteční podmínku:

```
>> dsolve('Dy = y^2','x')
ans =
            0
 -1/(C38 + x)
>> dsolve('Dx = -s*x^2','s')
ans =
                  0
 -1/(- s^2/2 + C54)
>> dsolve('Dy = -y^2','y(1)=1','x')
ans =
```

```
1/x
>>
```

Problémem nejsou ani okrajové úlohy pro rovnice druhého řádu:

```
>> dsolve('D2y = -y+cos(x)','y(0)=0','y(pi/2)=1','x')
ans =
cos(3*x)/8 - cos(x)/8 + sin(x)*(x/2 + sin(2*x)/4)
- sin(x)*(pi/4 - 1)
>>
```

Řešení diferenciálních rovnic v Sage je také snadné. Nejprve definujeme `x` jako nezávislou proměnnou, `y` jakožto funkci proměnné `x` a pro nalezení řešení rovnice použijeme funkcei `desolve`. Zde se implicitně předpokládá, že pokud pravá strana rovnice není uvedena, je nulová. Stejnou rovnici tedy mohu zadat dvěma způsoby:

```
sage: x=var('x')
sage: y=function('y',x)
sage: desolve(diff(y,x) ==x*y, y)
c*e^(1/2*x^2)
sage: desolve(diff(y,x) -x*y, y)
c*e^(1/2*x^2)
sage:
```

V některých případech je potřeba z výsledku funkci `y` vyjádřit a to i při zadání počáteční podmínky:

```
sage: desolve(diff(y,x) == y^2, y)
-1/y(x) == c + x
sage: desolve(diff(y,x) == y^2, y,[1,-1])
-1/y(x) == x
sage:
```

Pro rovnice vyšších řádu se uvádí řád derivace jako třetí parametr funcke `diff`. Zkusíme-li vyřešit v Sage stejnou okrajovou úlohu jako v Matlabu, dostaneme o něco jednodušší vyjádření výsledku:

```
sage: desolve(diff(y,x,2) == -y+cos(x), y,[0,0,pi/2,1])
-1/4*(pi - 4)*sin(x) + 1/2*x*sin(x)
sage:
```

Čtenář si jako cvičení může zkusit ověřit, zda oba výsledky jsou totožné.

V případě, že Sage nějakou rovnici nedokáže vyřešit (nebo uděláme chybu v zadání), objeví se chybové hlášení, ze kterého je vidět, že Sage pro řešení diferenciálních rovnic používá software zvaný Maxima. Více informací o něm mohou zájemci zjistit na internetových stránkách `http://maxima.sourceforge.net/`.

3.2 Numerické řešení

Pro nalezení numerického řešení budeme používat Matlab. Numerické řešení hledáme na nějaké síti bodů $(x_i)_{i=0}^n$, takže nelze hledat obecné řešení, ale pouze přibližné partikulární řešení nějaké počáteční úlohy.

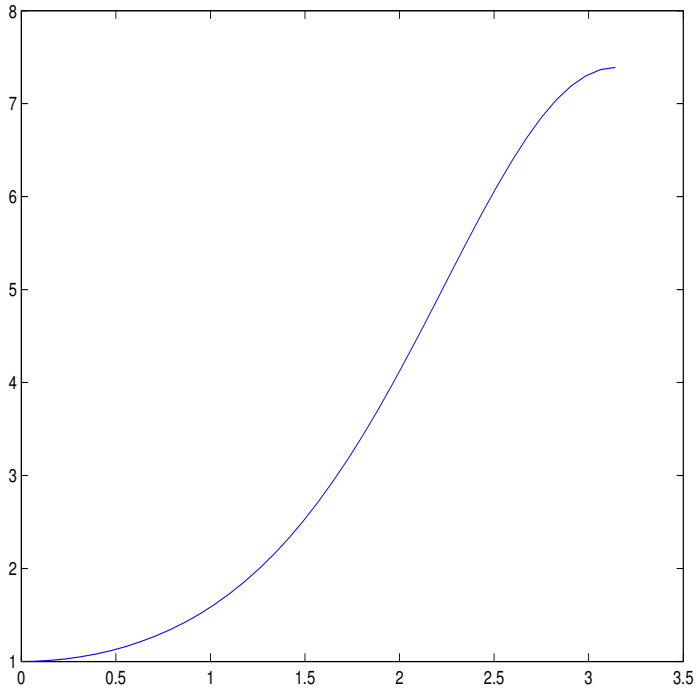
V současné době patrně nejpoužívanější metody jsou metody Runge–Kutta, které patří mezi jednokrokové metody, to znamená, že výpočet přibližných hodnot hledané funkce y v bodě x_{k+1} probíhá na základě hodnoty v předchozím bodě x_k. Kromě toho známe také vícekrokové metody (pro výpočet v x_{k+1} se použije více předchozích hodnot), ale ty jsou v základním matlabovském balíku využívány jen velmi málo (funkce `ode113`).

Nejpoužívanějšími metodami Runge–Kutta jsou metody čtvrtého řádu, v Matlabu je jedna z nich implementována ve funkci `ode45`. Nápověda k této funkci je poměrně obsáhlá a taky v ní můžeme zjisti další funkce použitelné pro numerické řešení diferenciálních rovnic. Zaměříme se jen na tuto funkci a postačí nám základní použití.

V nejjednodušší verzi musíme zadat funkci f z pravé strany diferenciální rovnice, interval, na němž chceme přibližné řešení spočítat a také počáteční podmínku. Výsledkem jsou dva vektory – síť bodů, a příslušné funkční hodnoty. Výsledek si zobrazíme:

```
>> f=inline('y*sin(t)');
>> I=[0,pi];
>> cond=[1];
>> [t,y]=ode45(f,I,cond);
```

```
>> plot(t,y)
>>
```



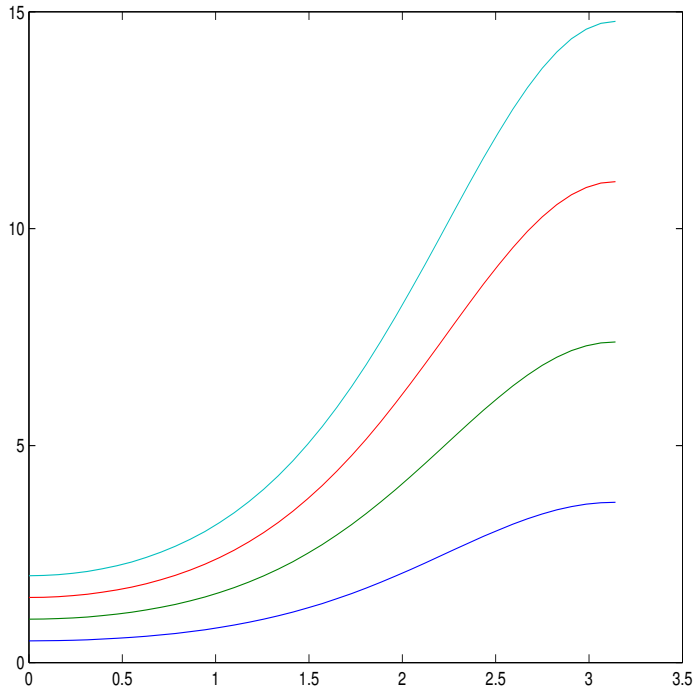
Protože nás zajímá, s jakou chybou je řešení určeno, zkusíme je porovnat s hodnotami řešení přesného:

```
>> dsolve('Dy = y*sin(x)','y(0)=1','x')
ans =
exp(1)/exp(cos(x))
>> ff=inline('exp(1)./exp(cos(x))')
ff =
     Inline function:
     ff(x) = exp(1)./exp(cos(x))
>> yy=ff(t);
>> max(abs(y-yy))
ans =
   2.1830e-05
>>
```

Chyba je tedy poměrně velmi malá, na grafickém znázornění by se přesné řešení od přibližného nedalo rozlišit.

Co se týče počátečních podmínek, berou se automaticky v krajním bodě zadaného intervalu, ale je možné jich zadat více najednou:

```
>> cond=[0.5 1 1.5 2];
>> [t,y]=ode45(f,I,cond);
>> plot(t,y)
>>
```



Kapitola 4

Statistické metody

Kropáčkův zpěv na nápěv Mládkovy písně Pochod Praha–Prčice:
„Jsou plné pohody statistické metody
s využitím náhody já problém vyřeším.“

V této kapitole se nebudeme věnovat teorii pravděpodobnosti a teoretické statistice, zkusíme se zaměřit na praktické statistické výpočty.

Co se týče pravděpodobnosti a statistiky, je Matlab skvěle vybaven, zejména pokud máte k dispozici statistický toolbox. Zde jsou implementovány všechny běžné metody a algoritmy, ale také spousta metod a algoritmů speciálních. Protože popis celého statistického toolboxu je nad rámec tohoto textu, omezíme se toliko na základní použití.

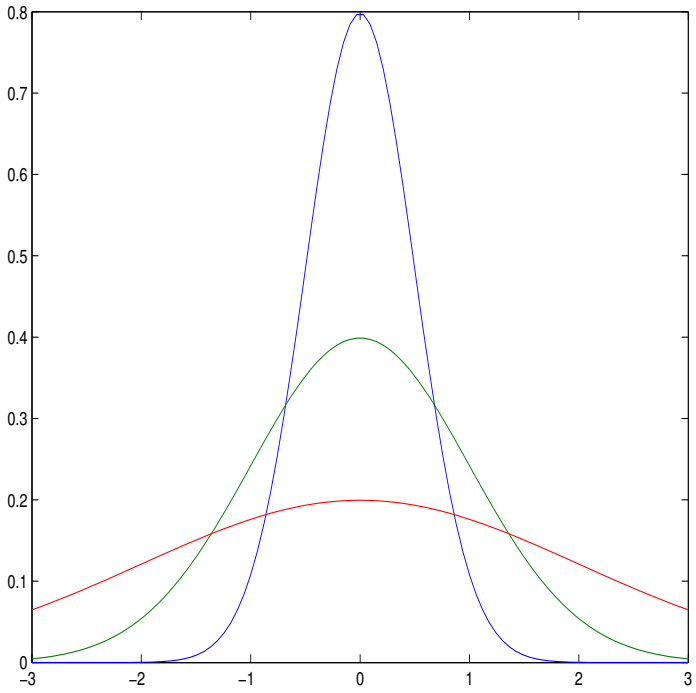
Pokud chceme vyzkoušet některé dostupné programy a nemáme k dispozici vhodná data, není problém si je vygenerovat. nabízí se generátor náhodných čísel, který umí vytvořit data s požadovaným rozdělením. Tak například pro data s normálním rozdělením lze použít funkci `normrnd`, s exonenciálním `exprnd`, s rovnoměrným `unifrnd`, s beta rozdělením `betarnd` atd.

Podobně pro výpočet hustoty náhodné veličiny jsou k použití funkce končící „`pdf`“ (z anglického *probability density function*), tedy `normpdf`, `unifpdf`, `exppdf` apod., pro distribuční funkce můžeme požít funkce `normcdf`, `unifcdf`, `expcdf` atd. (z anglického *cumulative distribution function*).

Pro kvantily se používají funkce `norinv`, `unifinv`, `expinv` apod. jako hodnoty inverzní distribuční funkce. Co se týče parametrů uvedených funkcí, zaleží samozřejmě na tom, jakými parametry je určeno konkrétní rozdělení, takže nejlepší bude poradit se v každém konkrétním případě s nápovědou.

Pokud bychom se třeba chtěli podívat, jak se mění hustota normální rozdělení v závislosti na směrodatné odchylce, stačí zadat:

```
>> x=linspace(-3,3);
>> s1=0.5;s2=1;s3=2;
>> y1=normpdf(x,0,s1);
>> y2=normpdf(x,0,s2);
>> y3=normpdf(x,0,s3);
>> plot(x,[y1;y2;y3])
```



Když už jsem se zmínili o směrodatné odchylce, pro její výpočet se používá příkaz `std`, pro rozptyl příkaz `var`. A nesmíme zapomenout na střední hodnotu a medián – `mean` a `median`.

Tyto čtyři funkce nejsou součástí statistického toolboxu ale jsou již v základní matlabovské výbavě. Jedná se samozřejmě o odhady příslušných charakteristik náhodných veličin, nikoliv o přesné hodnoty.

Zkusme porovnat u vygenerovaných dat teoretické hodnoty s vypočítanými. Pokud bychom v podobném duchu udělali rozsáhlé testování, mohou

nám výsledky leccos napovědět o kvalitě generátoru náhodných čísel, který Matlab používá.

Vygenerujeme 100 normálně rozdělených hodnot se střední hodnotou 1 a rozptylem 0.25. Nesmíme ovšem zapomenout, že jako parametr zadáváme směrodatnou odchylku:

```
>> x=normrnd(1,0.5,1,100);
>> mean(x)
ans =
    1.0615
>> median(x)
ans =
    1.0477
>> var(x)
ans =
    0.3378
>> std(x)
ans =
    0.5812
>>
```

Vidíme, že u rozptylu je relativně největší chyba, z jednoho vzorku ovšem nemůžeme vyvozovat žádné závěry.

4.1 Intevalové odhady

Jednou ze základních statistických úloh je určení intervalu, v němž s danou pravděpodobností leží nějaký parametr náhodného rozdělení. Nejčastěji předpokládáme, že máme data s normálním rozdělením nebo s rozdělením s normálního odvozeným.

Podíváme se, jak se mění interval spolehlivosti pro odhad střední hodnoty normálních dat, pokud roste jejich počet. Nejprve předpokládejme, že známe hodnotu rozptylu, pak se používají kvantily standardizovaného normálního rozdělení:

```
>> n=100;
>> mu=1;
>> sigma=0.5;
```

```
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    0.9813
>> alpha=0.05;
>> D=M-norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> H=M+norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> I=[D,H]
I =
    0.8833    1.0793
>> n=500;
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    0.9878
>> D=M-norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> H=M+norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> I=[D,H]
I =
    0.9439    1.0316
>> n=1000;
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    1.0266
>> D=M-norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> H=M+norminv(1-alpha/2,0,1)*sigma/sqrt(n);
>> I=[D,H]
I =
    0.9956    1.0576
>>
```

V případě, že rozptyl není známý, použijeme kvantily Studentova rozdělení:

```
>> n=100;
>> alpha=0.05;
>> mu=1;
>> sigma=0.5;
```

```
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    0.9590
>> S=std(X)
S =
    0.5047
>> D=M-tinv(1-alpha/2,n-1)*S/sqrt(n);
>> H=M+tinv(1-alpha/2,n-1)*S/sqrt(n);
>> I=[D,H]
I =
    0.8589    1.0592
>> n=500;
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    1.0160
>> S=std(X)
S =
    0.4909
>> D=M-tinv(1-alpha/2,n-1)*S/sqrt(n);
>> H=M+tinv(1-alpha/2,n-1)*S/sqrt(n);
>> I=[D,H]
I =
    0.9729    1.0592
>> n=1000;
>> X=normrnd(mu,sigma,1,n);
>> M=mean(X)
M =
    1.0172
>> S=std(X)
S =
    0.4930
>> D=M-tinv(1-alpha/2,n-1)*S/sqrt(n);
>> H=M+tinv(1-alpha/2,n-1)*S/sqrt(n);
>> I=[D,H]
I =
    0.9866    1.0478
```

```
>>
```

Je jasné, že inerval se zkracuje (délka klesá s o odmocninou n), ale rozdíl mezi oběma případy není patrný.

4.2 Testování hypotéz

Pro testování hypotéz je v Matlabu možné využít již hotových programů. Základním z nich je `ttest`, který v nejjednodušší verzi testuje, zda rozptyl náhodného výběru je nulový. Hladina významnosti je implicitně 5 procent:

```
>> n=100;
>> mu=1;
>> sigma=0.5;
>> X=normrnd(mu,sigma,1,n);
>> ttest(X)
ans =
     1
```

Výsledek 1 znamená zamítnutí hypotézy. Pokud bychom chtěli vědět také pravděpodobnost (tzv. p-hodnota), přidáme výstupní parametr:

```
>> [H,P]=ttest(X)
H =
     1
P =
   1.1433e-32
>>
```

Vidíme, že střední hodnota není nulová téměř jistě. Provedeme tady drobnou úpravu:

```
>> M=mean(X)
M =
    0.9926
>> [H,P]=ttest(X,M)
H =
```

```
     0
P =
     1
>>
```

Ještě si otestujeme, jak dopadne skutečná střední hodnota použitá při generování náhodného výběru:

```
>> [H,P]=ttest(X,1)
H =
     0
P =
    0.8946
>>
```

Funkce `ttest` se dá také použít na testování rovnosti středních hodnot dvou náhodných výběru stejného rozsahu:

```
>> Y=normrnd(mu,sigma,1,n);
>> [H,P]=ttest(X,Y)
H =
     0
P =
    0.5040
>> Y=normrnd(mu-0.1,sigma,1,n);
>> [H,P]=ttest(X,Y)
H =
     0
P =
    0.2128
>>
```

K testování rovnosti středních hodnot u dvou výběrů se dá ooužít také funkce `ttest2`, zdá se ale, že pracuje poněkud jinak než funkce `ttest`:

```
>> [H,P]=ttest(X,Y)
H =
     0
```

```
P =
    0.5040
>> [H,P]=ttest2(X,Y)
H =
     0
P =
    0.5276
>>
```

4.3 Lineární regrese

Studenti si lineární regresi občas pletou s konstrukcí regresní přímky, což je ale jen speciální případ lineární regrese. Na tuto problematiku jsme už narazili v prvním semestru, když jsem hovořili o pseudoinverzní matici.

V lineární regresi předpokládáme, že pozorovaná data, která zpracováváme, jsou lineárními kombinacemi funkcí předem daných hodnot, přičemž v pozorování je nějaký náhodná chyba. Pozorování je tedy náhodná veličina $Y(x) = \beta_1 f_1(x) + \cdots + \beta_k f_k(x) + \varepsilon$. Měříme hodnoty Y v zadaných bodech $x_1,\ldots,x_n$, $n \geq k$, tím dostáváme celkem soustavu n rovnic pro k neznámých. Matice soustavy se ve statistice nazývá *matice plánu* a zpravidla se označuje $\mathbf{X}$. Při řešení se snažíme minimalizovat reziduum (rozdíl mezi levou a pravou stranou), k čemuž se výborně hodí pseudoinverzní matice.

Jestliže navíc předpokládáme normalitu chyby *veps* se známým rozptylem, dají se spočítat i intervalové odhady pro parametry β_j.

Příklad 5. Mějme třídící algoritmus a chceme otestovat jeho kvalitu. Dobré třídící algoritmy pro n dat potřebují $O(n \log n)$ času, horší algoritmy potřebují $O(n^2)$ času. Algoritmus jsme otestovali pro náhodné výběry různých rozsahů a následná tabulka udává průměrné časy pro jednotlivé rozsahy v milisekundách:

n	100	200	300	400	500
t	0.42.883	127.39	264.08	435.69	665.0

n	600	700	800	900	1000
t	920.24	1220.4	1565.8	1941.4	2348.6

Vyrobíme matici plánu, přičemž předpokládáme, že výsledná funkce může být polynom druhého stupně doplněná funkcemi $\log n$ a $n \log n$.

```
>> n
n =
  Columns 1 through 5
     100          200          300          400          500
  Columns 6 through 10
     600          700          800          900         1000
>> t
t =
  Columns 1 through 5
    42.883    127.39    264.08    435.69    665.08
  Columns 6 through 10
    920.24    1220.4    1565.8    1941.4    2348.6
>> X1=ones(10,1);
>> X2=n';
>> X3=(n.^2)';
>> X4=(log(n))';
>> X5=(n.*log(n))';
>> X=[X1,X2,X3,X4,X5]
X =
       1       100        10000     4.6052     460.52
       1       200        40000     5.2983     1059.7
       1       300        90000     5.7038     1711.1
       1       400       1.6e+05    5.9915     2396.6
       1       500       2.5e+05    6.2146     3107.3
       1       600       3.6e+05    6.3969     3838.2
       1       700       4.9e+05    6.5511     4585.8
       1       800       6.4e+05    6.6846     5347.7
       1       900       8.1e+05    6.8024     6122.2
       1      1000         1e+06    6.9078     6907.8
>>
```

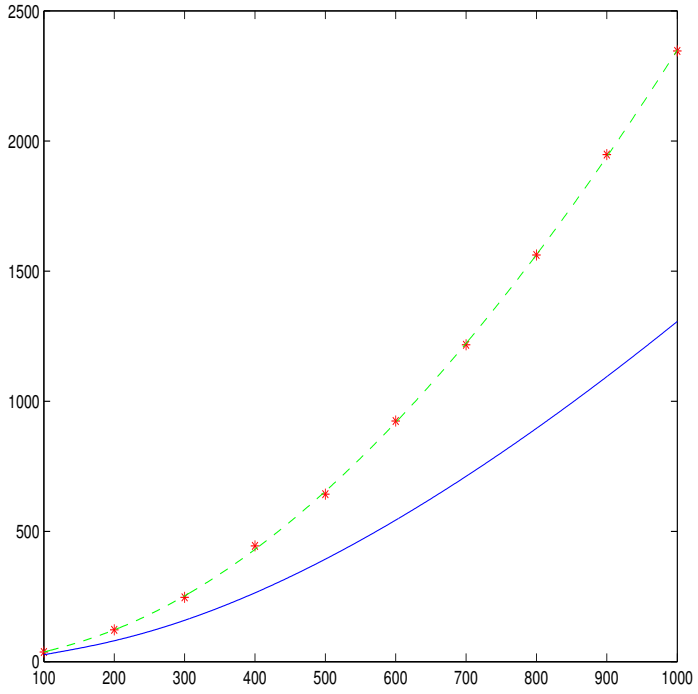
Nakonec provedeme výpočet odhadu parametrů s použitím pseudoinverzní matice:

```
>> Y=t';
>> beta=pinv(X)*Y
beta =
```

```
      -278.98
      -5.6652
    0.0013371
       99.146
      0.90803
>>
```

Zdá se že koeficient u druhé mocniny je zanedbatelný oproti ostatním koeficientům. Podíváme se ještě, jak funkce, kterou jsme získali, prochází aproximuje data. Přitom ověříme rozdíl mezi tím když uvedený koeficient zanedbáme a když nikoliv:

```
>> nn=100:1000;
>> f1=beta(1)+beta(2)*nn+beta(4)*log(nn)+...
beta(5)*nn.*log(nn);
>> f2=beta(1)+beta(2)*nn+beta(3)*nn.^2+beta(4)*log(nn)+...
beta(5)*nn.*log(nn);
>> plot(n,t,'*r',nn,f1,'b',nn,f1,'g--')
>>
```



Vidíme, že koeficient zanedbat nemůžeme, protože je sice řádově daleko menší než ostatní koeficienty, ale vzhledem k tomu, že se jím násobí velká čísla (druhé mocniny n), nelze jej vynechat.

Literatura

[1] Anděl, J.: *Základy matematické statistiky.* Praha: Matfyzpress, druhé vydání, 2007, ISBN 978-80-7378-001-2.

[2] Horová, I.; Zelinka, J.: *Numerické metody.* Brno: Masarykova univerzita, druhé vydání, 2008, ISBN 978-80-210-3317-7.

[3] Ralston, A.: *Základy numerické matematiky.* Praha: Academia, druhé vydání, 1978.