Počítačový algebraický systém Maple jako pomůcka při studiu předmětu Matematika I a II

Pavel Pokorný
Praha, 3. dubna 2004

Soustavy nelineárních algebraických rovnic

Obsah

1 Úvod

Soustavy nelineárních algebraických rovnic (ani samostatné rovnice) nelze (až na výjimky) řešit přesně analyticky. Proto musíme k jejich řešení použít nějakou přibližnou numerickou metodu. Ukážeme si šest nezávislých způsobů, jak řešit soustavu nelineárních algebraických rovnic pomocí systému Maple:

- graficky,
- pomocí příkazu `fsolve`,
- Newtonovou metodou bez speciálních balíků,
- Newtonovou metodou s užitím balíku `linalg`,
- Newtonovou metodou s užitím balíku `LinearAlgebra`,
- pomocí kontinuace.

Porovnáme výhody a nevýhody jednotlivých způsobů. Přitom porovnáme balíky pro lineární algebru `linalg` a `LinearAlgebra`. Ukážeme si, jak lze z příkazů Maple sestavit program a jak lze tento program uložený v samostatném souboru spustit. Při vyšetřování závislosti řešení na počátečním odhadu si ukážeme několik fraktálů.

Vše budeme ukazovat na příkladě soustavy dvou rovnic

$$x^3 + 3xy + y^3 = 0 \tag{1}$$
$$\exp(x) - 2x + \exp(y) - y - 3 = 0. \tag{2}$$

Poznámka:
Když hovoříme o soustavě nelineárních algebraických rovnic, může vyvstat otázka, jestli popsané metody lze použít i pro jedinou rovnici, či pro lineární rovnice. Ano, lze, ale úloha se tím zjednoduší natolik, že zde popsané metody řešení jsou zbytečně složité. Nelineární algebraické rovnice obvykle řešíme posloupností kroků, jejichž výsledky se při správném použití blíží (říkáme, že konvergují) ke správnému řešení. Pro lineární rovnice dostaneme správné řešení již po prvním kroku.

2 Grafická metoda

Řešíme-li soustavu dvou rovnic

$$\begin{aligned} f(x,y) &= 0 && (3)\\ g(x,y) &= 0, && (4) \end{aligned}$$

pak každá z rovnic (v typickém případě) určuje křivku v rovině x-y.

Hledat řešení soustavy znamená hledat průsečíky těchto dvou křivek. To lze provést pomocí systému Maple následujícími příkazy:

Příkazem

```
> restart;
```

zrušíme uložené definice z předchozí práce se systémem Maple, aby nekolidovaly s novými, a také uvolníme část paměti. Tento příkaz je zbytečný těsně po spuštění systému Maple.

Definujeme levé strany rovnic

```
> f := x^3 + 3*x*y + y^3;
```

$$f := x^3 + 3\,x\,y + y^3$$

```
> g := exp(x) - 2*x + exp(y) - y - 3;
```

$$g := e^x - 2\,x + e^y - y - 3$$

Protoze příkaz `implicitplot`, který budeme používat, je v balíku `plots`, máme dvě možnosti:

- volat jej plným jménem `plots[implicitplot]`
- nebo příkazem `with(plots)` zpřístupnit všechny funkce z balíku `plots` a poté jej volat zkráceným jménem `implicitplot`. Příkaz `with(plots)` ukončíme dvojtečkou, abychom potlačili hlášení (seznam funkcí obsažených v balíku), která v této chvíli nepotřebujeme.

```
> with(plots):
```

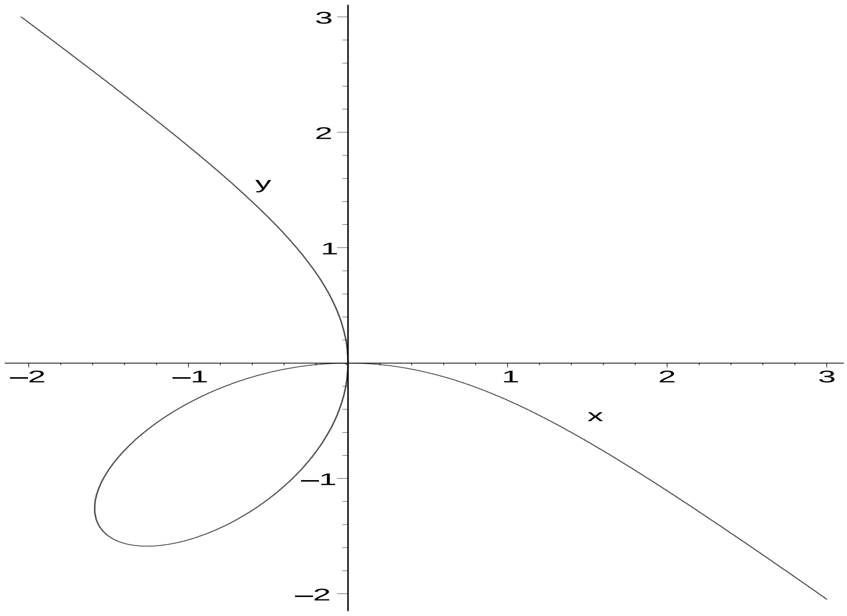
Definujeme rozsahy proměnných a počet bodů pro vykreslení grafu

```
> rozsahy := x=-3..3, y=-3..3, numpoints=100000;
```

$$\mathit{rozsahy} := x = -3..3,\ y = -3..3,\ \mathit{numpoints} = 100000$$

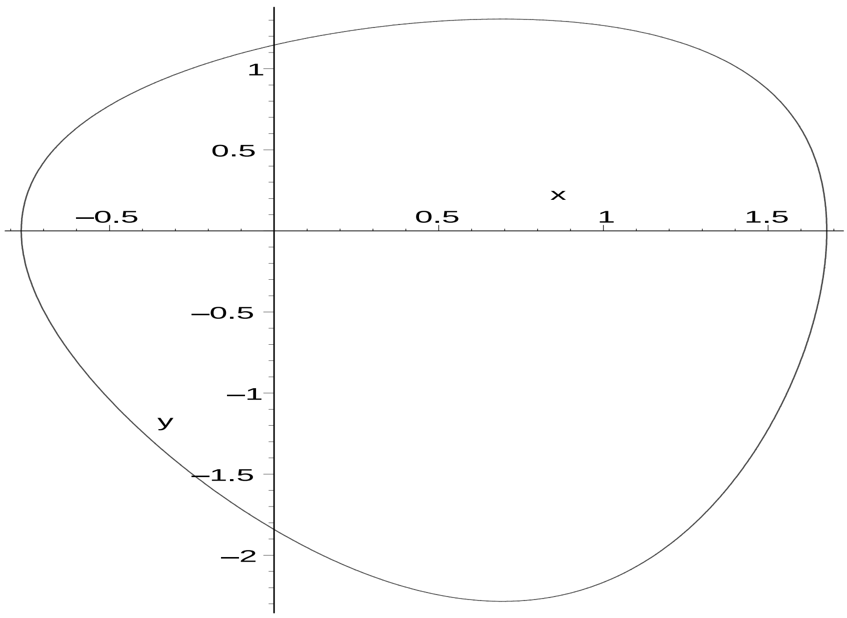
Nakreslíme křivku danou první rovnicí

```
> implicitplot(f,rozsahy);
```



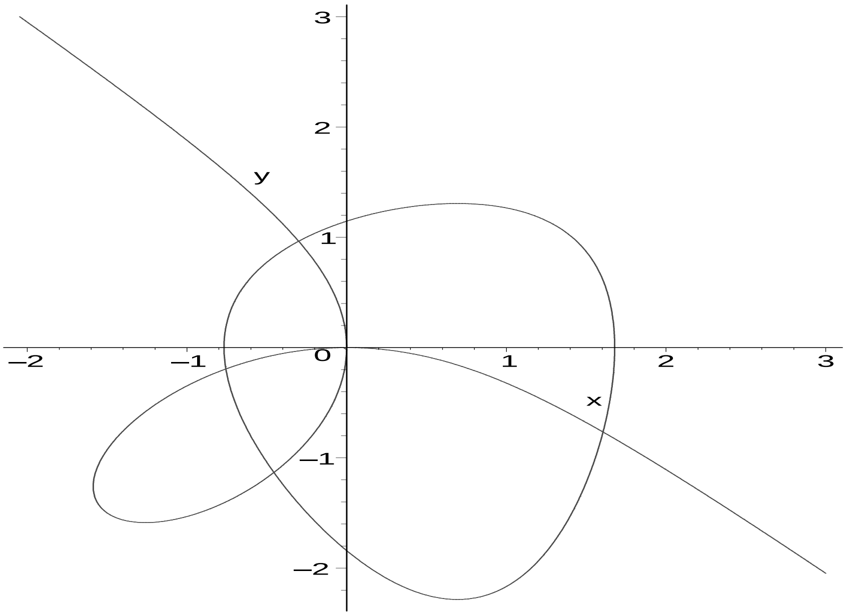
a křivku danou druhou rovnicí

```
>  implicitplot(g,rozsahy);
```



a obě křivky najednou

```
> implicitplot({f,g},rozsahy);
```



Z grafu je vidět, že daná soustava má čtyři reálná řešení (o komplexních to neříká nic, ale ta nás nezajímají) a lze je z grafu i přibližně určit. Chceme-li řešení určit přesněji, zúžíme rozsahy proměnných x a y pro vykreslení grafu.

Shrnutí:

Výhody grafické metody:

- je rychlá,
- jednoduchá,
- názorná,
- dává informaci o počtu všech řešení v daném rozsahu proměnných x, y,
- je vhodná pro první seznámení s danou soustavou rovnic.

Nevýhody grafické metody:

- je málo přesná,
- je těžko použitelná pro více než dvě rovnice.

3 Použití příkazu `fsolve`

Pro přibližné numerické řešení soustav nelineárních algebraických rovnic lze v systému Maple s výhodou použít příkazu `fsolve` (anglické slovo „solve" znamená řešit, písmeno f na začátku je z anglického „floating point number" a označuje práci s přibližnými čísly s pohyblivou desetinnou čárkou).

Nejdříve si opět vyčistíme paměť

```
> restart;
```

a definujeme levé strany rovnic

```
> f := x^3 + 3*x*y + y^3;
```

$$f := x^3 + 3\,x\,y + y^3$$

```
> g := exp(x) - 2*x + exp(y) - y - 3;
```

$$g := e^x - 2\,x + e^y - y - 3$$

Příkaz `fsolve` zavoláme s jedním argumentem, dvojicí levých stran daných rovnic (jsou-li pravé strany nulové, není třeba je psát)

```
> reseni1 := fsolve({f,g});
```

$$\mathit{reseni1} := \{x = 1.605448402,\ y = -0.7658800022\}$$

Chceme-li jiné řešení (z grafické metody již víme, že daná soustava má celkem čtyři reálná řešení), zavoláme příkaz `fsolve` se dvěma argumenty, druhým argumentem je počáteční odhad řešení. Ten zjistíme grafickou metodou nebo zkusmo nebo na základě našich dalších znalostí o úloze.

```
> reseni2 := fsolve({f,g},{x=-0.5,y= 1.0});
```

$$\mathit{reseni2} := \{y = 0.9642204252,\ x = -0.3005240445\}$$

Podobně najdeme i třetí a čtvrté řešení

```
> reseni3 := fsolve({f,g},{x=-0.8,y=-0.2});
```

$$\mathit{reseni3} := \{x = -0.7565242968,\ y = -0.1939930663\}$$

```
> reseni4 := fsolve({f,g},{x=-0.6,y=-1.2});
```

$$\mathit{reseni4} := \{y = -1.132719887,\ x = -0.4554952073\}$$

Vzniká otázka, jak závisí nalezené řešení na počátečním odhadu. Je-li odhad blízký jednomu z přesných řešení, pak nalezené řešení je obvykle rovno tomuto přesnému řešení (až na chybu metody a zaokrouhlovací chybu). Není-li odhad dostatečně blízký žádnému z přesných řešení, pak tato závislost může být složitá. Této otázce se budeme podrobně věnovat v samostatné kapitole.

Máme-li nalezené řešení, lze provést zkoušku tak, že nalezené řešení dosadíme do levých stran daných rovnic. To provedeme příkazem `eval` (z anglického „evaluate“ – vyhodnotit), který zavoláme se dvěma argumenty: levou stranou zadané rovnice a nalezeným řešením (ve tvaru rovnice, kterou nám vrátil příkaz `fsolve`)

```
> eval(f,reseni1);
```

$$-0.18\,10^{-8}$$

a dosazení do levé strany druhé rovnice

```
> eval(g,reseni1);
```

$$-0.1\,10^{-8}$$

Vidíme, že levé strany nejsou rovny nule, ale malému číslu. Tato chyba je dána chybou metody a zaokrouhlovací chybou. Tyto chyby lze v systému Maple ovlivnit pomocí proměnné `Digits`, která určuje, na kolik desetinných míst se provádí numerické výpočty. Lze zjistit její nastavení příkazem

```
> Digits;
```

$$10$$

Vidíme, že výpočty se provádí na 10 desetinných míst. Požadujeme-li větší přesnost, např. 16 desetinných míst (tato hodnota je výhodná, protože tuto přesnost poskytuje hardware většiny počítačů; lze požadovat i větší přesnost, ale pak rychle roste doba výpočtu), lze její hodnotu změnit takto

```
> Digits:=16;
```

$$\mathit{Digits} := 16$$

Zavoláme-li nyní příkaz `fsolve`, dá přesnější řešení

```
> presnejsireseni1 := fsolve({f,g});
```

$$\mathit{presnejsireseni1} := \{x = 1.605448402044969,\ y = -0.7658800021965751\}$$

o čemž se přesvědčíme dosazením do levých stran daných rovnic, které uzavřeme do hranatých závorek

```
> eval([f,g],presnejsireseni1);
```

$$[-0.5\,10^{-15},\ 0.]$$

Vidíme, že chyba je opravdu menší.

Shrnutí:

Výhody příkazu `fsolve`:

- snadné použití,
- použitelný i pro velký počet rovnic,
- vysoká přesnost, kterou lze navíc nastavit.

Nevýhody příkazu `fsolve`:

- nedá celkový obraz o počtu a rozmístění všech řešení.

4 Newtonova metoda bez speciálních balíků

4.1 Odvození Newtonovy metody

Použijeme-li Taylorův rozvoj prvního řádu pro vektorovou funkci F v okolí bodu $x_0 \in R^n$, dostaneme

$$F(x) = F(x_0) + J(x_0) \cdot (x - x_0) + R,$$

kde $J(x_0)$ je matice parciálních derivací funkce F v bodě x_0 a R je zbytek. Zanedbáme zbytek R a hledáme bod $x = x_1$, kde je funkce F nulová,

$$0 = F(x_0) + J(x_0) \cdot (x_1 - x_0).$$

Je-li matice $J(x_0)$ regulární, lze nalézt řešení takto

$$J(x_0) \cdot (x_1 - x_0) = -F(x_0)$$

$$x_1 - x_0 = -J^{-1}(x_0) \cdot F(x_0)$$

$$x_1 = x_0 - J^{-1}(x_0) \cdot F(x_0).$$

Při výpočtu je výhodnější řešit soustavu lineárních rovnic

$$J(x_0) \cdot \Delta x = F(x_0)$$

a potom dopočítat

$$x_1 = x_0 - \Delta x.$$

4.2 Spuštění samostatného programu

Chceme-li zadat více příkazů, máme dvě možnosti:

- můžeme psát jednotlivé řádky klávesnicí přímo v prostředí Maple,
- a nebo si je můžeme předem napsat (pomocí libovolného editoru, např. `vi`) do souboru, který nazveme např. `mujprogram1`. Příkazy obsažené v tomto programu lze spustit v Maple příkazem

```
read mujprogram1;
```

4.3 Použití Newtonovy metody

Pro použití Newtonovy metody může tento program vypadat takto:

```
restart:
Digits := 16:

f := x^3 + 3*x*y + y^3:
g := exp(x) - 2*x + exp(y) - y - 3:
fx := diff(f,x):
fy := diff(f,y):
gx := diff(g,x):
gy := diff(g,y):

mujnewton1 := proc (x0,y0)
 local f0,g0,detj,detx,dety,dx,dy,x1,y1:
 f0 := eval(f,{x=x0,y=y0}):
 g0 := eval(g,{x=x0,y=y0}):
 detj := eval(fx * gy - fy * gx, {x=x0,y=y0}):
 detx := eval(f0 * gy - fy * g0, {x=x0,y=y0}):
 dety := eval(fx * g0 - f0 * gx, {x=x0,y=y0}):
 dx := detx / detj:
 dy := dety / detj:
 x1 := x0 - dx:
 y1 := y0 - dy:
 x1,y1:
end:
```

```
odhad0 := 1.6, -0.8;
odhad1 := mujnewton1(odhad0);
odhad2 := mujnewton1(odhad1);
odhad3 := mujnewton1(odhad2);
odhad4 := mujnewton1(odhad3);
levestrany := eval([f,g],{x=odhad4[1],y=odhad4[2]});
```

Vysvětlení:

- nejdříve si vyčistíme paměť příkazem `restart;`
- zadáme pravé strany `f` a `g`,
- připravíme si jejich parciální derivace `fx, fy, gx, gy`,
- napíšeme proceduru `mujnewton1`, která má dva vstupní parametry `x0, y0` a počítá jednu iteraci Newtonovy metody takto:
  - nejdříve napíšeme seznam lokálních proměnných, které budeme používat,
  - vyčíslíme levé strany `f0, g0`,
  - vyčíslíme determinant `detj` matice parciálních derivací a determinanty `detx, dety` matic, které dostaneme náhradou jednotlivých sloupců sloupcem pravých stran lineární soustavy,
  - Cramerovým pravidlem najdeme řešení `dx,dy` lineární soustavy,
  - od vstupního odhadu řešení `x0, y0` odečteme korekci `dx, dy` a dostaneme upřesněné řešení `x1, y1`.
  - Posledním výrazem v proceduře je dvojice `x1, y1` a ta bude výsledkem procedury.

Použití je jednoduché: zvolíme nultý odhad `odhad0`, což je dvojice čísel oddělených čárkou, a zavoláme naši proceduru `mujnewton1` a do závorky jí dáme tento odhad. Výsledek uložíme do proměnné `odhad1`. Tak můžeme pokračovat, až se aproximace přestanou měnit. Zkoušku provedeme vyhodnocením levých stran např. pro čtvrtý odhad.

Výsledky příkazu `read mujprogram1` pak vypadají takto:

```
> read mujprogram1;
```

$$odhad0 := 1.6,\ -0.8$$
$$odhad1 := 1.605498579860371,\ -0.7662250746521963$$
$$odhad2 := 1.605448404509504,\ -0.7658800518916949$$
$$odhad3 := 1.605448402044969,\ -0.7658800021965758$$
$$odhad4 := 1.605448402044969,\ -0.7658800021965752$$
$$levestrany := [-0.7\,10^{-15},\ 0.]$$

Shrnutí:
Výhody Newtonovy metody bez použití speciálních balíků:

- víme přesně, co a jak počítáme,
- rychlý výpočet.

Nevýhody Newtonovy metody bez použití specálních balíků:

- poměrně dlouhý program, zejména pro více rovnic, tedy mnoho programátorské práce.

5 Newtonova metoda s užitím balíku `linalg`

Při použití Newtonovy metody pro řešení soustavy nelineárních algebraických rovnic musíme v každém kroku vyřešit soustavu lineárních algebraických rovnic. Při řešení lineární soustavy je výhodné (zejména pro více rovnic) pracovat s vektory a maticemi. Pro práci s vektory a maticemi jsou v systému Maple k dispozici dva různé balíky: `linalg` a `LinearAlgebra`. Ukážeme si nejprve použití staršího balíku `linalg`, potom novějšího balíku `Linear Algebra` a poté je porovnáme.

Pro použití Newtonovy metody s užitím balíku `linalg` může program (nazveme jej např. `mujprogram2`) vypadat takto:

```
restart:
Digits:=16:
with(linalg):

f := x^3 + 3*x*y + y^3:
g := exp(x) - 2*x + exp(y) - y - 3:
fx := diff(f,x):
fy := diff(f,y):
```

```
gx := diff(g,x):
gy := diff(g,y):
F := vector([f,g]):
J := matrix([[fx,fy],[gx,gy]]):

mujnewton2 := proc (v0::vector)
 local F0,J0,v1:
 F0 := eval (eval(F),{x=v0[1],y=v0[2]}):
 J0 := eval (eval(J),{x=v0[1],y=v0[2]}):
 v1 := evalm(v0 - linsolve(eval(J0),eval(F0))):
end:

odhad0 := vector([1.6, -0.8]);
odhad1 := mujnewton2(eval(odhad0));
odhad2 := mujnewton2(eval(odhad1));
odhad3 := mujnewton2(eval(odhad2));
odhad4 := mujnewton2(eval(odhad3));
levestrany := eval (eval(F),{x=odhad4[1],y=odhad4[2]});
```

Vysvětlení:

- Příkazem `with(linalg):` načteme balík `linalg`, abychom mohli volat příkazy v něm definované krátkým jménem.
- Zadáme levé strany `f, g` a jejich parciální derivace `fx, fy, gx, gy`.
- Příkazem `F := vector([f,g]);` vytvoříme vektor levých stran a pojmenujeme si jej `F`, abychom se na něj mohli odvolávat.
- Podobně příkazem `J := matrix([[fx,fy],[gx,gy]]);` vytvoříme matici parciálních derivací a pojmenujeme si ji `J`.
- Procedura `mujnewton2` je výrazně kratší než v případě bez použití balíku `linalg`, zejména díky použití příkazu `linsove` pro řešení soustavy lineárních algebraických rovnic. Voláme jej se dvěma parametry, maticí soustavy `J0` a vektorem pravých stran `F0`. Při odkazu na matici nebo na vektor musíme při použití balíku `linalg` volat funkci `eval`. Kdybychom např. místo `eval(F)` napsali jen `F`, dostali bychom samotný symbol `F` místo složek vektoru `F`. To může být někdy výhoda, v našem případě je to ale spíše nevýhoda, která činí program zbytečně dlouhým a méně přehledným.

- Pro odčítání vektorů při použití balíku `linalg` nestačí napsat pouze `-` , ale musíme rozdíl vyhodnotit příkazem `evalm`, jinak se rozdíl neprovede. Tyto potíže jsou v balíku `LinearAlgebra` vyřešeny lépe.

- Nakonec můžeme opět provést zkoušku.

Použití programu `mujprogram2` a výsledky pak jsou:

```
> read mujprogram2;
```

```
Warning, the protected names norm and trace have been redefined and
unprotected
```

$$odhad0 := [1.6, -0.8]$$

$$odhad1 := [1.605498579860371, -0.7662250746521963]$$

$$odhad2 := [1.605448404509504, -0.7658800518916949]$$

$$odhad3 := [1.605448402044969, -0.7658800021965758]$$

$$odhad4 := [1.605448402044969, -0.7658800021965752]$$

$$levestrany := [-0.7\,10^{-15}, 0.]$$

Shrnutí:

Výhody použití Newtonovy metody s užitím balíku `linalg`:

- program je kratší a přehlednější než bez použití balíku.

Nevýhody použití Newtonovy metody s užitím balíku `linalg`:

- pomalejší výpočet než bez použití balíku,

- program je delší a méně přehledný než při použití balíku `LinearAlgebra`.

6 Newtonova metoda s užitím balíku `LinearAlgebra`

Při použití Newtonovy metody s využitím balíku `LinearAlgebra` bude program (nazveme jej `mujprogram3`) vypadat takto:

```
restart:
Digits:=16:
with(LinearAlgebra):
```

```
f := x^3 + 3*x*y + y^3:
g := exp(x) - 2*x + exp(y) - y - 3:
fx := diff(f,x):
fy := diff(f,y):
gx := diff(g,x):
gy := diff(g,y):
JF :=  <<fx,gx>|<fy,gy>|<f,g>>:

mujnewton3 := proc (v0::Vector)
 local JF0,v1:
 JF0:= eval (JF,{x=v0[1],y=v0[2]}):
 v1 := v0 - LinearSolve(JF0):
end:

odhad0 := <1.6, -0.8>;
odhad1 := mujnewton3(odhad0);
odhad2 := mujnewton3(odhad1);
odhad3 := mujnewton3(odhad2);
odhad4 := mujnewton3(odhad3);
levestrany := eval (Column(JF,3),{x=odhad4[1],y=odhad4[2]});
```

Vysvětlení:

- vyčištění paměti, načtení balíku `LinearAlgebra`, zadání levých stran a jejich parciálních derivací je obdobné jako v předchozích případech,

- pro řešení soustavy lineárních algebraických rovnic se při použití balíku `LinearAlgebra` používá příkaz `LinearSolve`. Tento příkaz vyžaduje jediný argument – rozšířenou matici soustavy, tedy matici koeficientů u neznámých doplněnou sloupcem pravých stran. Proto nebudeme vytvářet zvlášť matici parciálních derivací a zvlášť vektor pravých stran soustavy lineárních rovnic, ale připravíme si rozšířenou matici najednou. Pro přípravu matic (i vektorů) se při práci s balíkem `Linear-Algebra` používají „úhlové“ závorky, tedy menšítko `<` a většítko `>`; v jedné úrovni pro přípravu vektoru, ve dvou do sebe vnořených úrovních pro přípravu matice. Takže rozšířenou matici dostaneme příkazem `JF := <<fx,gx>|<fy,gy>|<f,g>>;` pojmenujeme si ji `JF`. Na rozdíl od balíku `linalg` se při práci s balíkem `LinearAlgebra` matice zadávají ne po řádcích, ale po sloupcích. Jednotlivé sloupce se oddělují svislou čarou `|`.

- Pro zkoušku potřebujeme vyčíslit pravý (zde třetí) sloupec rozšířené matice. Třetí sloupec matice `JF` dostaneme příkazem `Column`, tedy `Column(JF,3)`.

Takže použití programu `mujprogram3` a jeho výsledky jsou

```
> read mujprogram3;
```

$$odhad0 := \begin{bmatrix} 1.6 \\ -0.8 \end{bmatrix}$$

$$odhad1 := \begin{bmatrix} 1.605498579860371 \\ -0.7662250746521963 \end{bmatrix}$$

$$odhad2 := \begin{bmatrix} 1.605448404509504 \\ -0.7658800518916949 \end{bmatrix}$$

$$odhad3 := \begin{bmatrix} 1.605448402044969 \\ -0.7658800021965758 \end{bmatrix}$$

$$odhad4 := \begin{bmatrix} 1.605448402044969 \\ -0.7658800021965752 \end{bmatrix}$$

$$levestrany := \begin{bmatrix} -0.7\,10^{-15} \\ 0. \end{bmatrix}$$

Shrnutí:
Výhody použití Newtonovy metody s užitím balíku `LinearAlgebra`:

- program je ještě kratší a ještě přehlednější než při použití balíku `linalg`. Pokud je naším cílem napsat opravdu krátký program, pak proceduru `mujnewton3`, která má pět řádků, lze napsat v ještě kratší podobě, na jediném řádku (nazveme ji např. `kn` jako „krátký Newton“)

```
kn:=proc(v::Vector) v-LinearSolve(eval(JF,{x=v[1],y=v[2]})); end;
```

Nevýhody použití Newtonovy metody s užitím balíku `LinearAlgebra`:

- pomalejší výpočet než bez použití balíku.

7 Srovnání balíků `linalg` a `LinearAlgebra`

Oba balíky jsou určeny pro usnadnění práce s vektory a maticemi, tedy obecně pro řešení úloh lineární algebry v systému Maple.

Balík `linalg` je starší; balík `LinearAlgebra` se objevil až v roce 2000 ve verzi Maple 6 a od té doby je upravován pouze balík `LinearAlgebra`, zatímco balík `linalg` je ponecháván v systému Maple již bez úprav pouze z důvodů slučitelnosti, tedy aby starší programy pracovaly i s novými verzemi systému Maple.

Balík `linalg` používá pro jména příkazů zkratky psané malými písmeny, balík `LinearAlgebra` používá pro jména příkazů celá anglická slova psaná s velkými počátečními písmeny.

Rozdíly v používání příkazů z obou balíků snad nemohou být větší. Jmenujme jen několik z nich. Matice se v balíku `linalg` zadávají po řádcích, v balíku `LinearAlgebra` po sloupcích. Funkce `linsolve` pro řešení soustavy lineárních rovnic v balíku `linalg` vyžaduje zvlášť matici a vektor pravých stran; funkce `LinearSolve` v balíku `LinearAlgebra` vyžaduje rozšířenou matici soustavy. Vektory a matice při práci s balíkem `linalg` musíme vyhodnocovat funkcí `eval`, jinak dostaneme pouze jejich jména, a ne jejich obsah. Aritmetické operace, jako součet a rozdíl, musíme v balíku `linalg` vyhodnocovat funkcí `evalm`.

Závěr:
pro psaní nových programů lze doporučit balík `LinearAlgebra`, protože je novější, obsahuje více funkcí a pro složité problémy (např. pro velký počet rovnic) jsou jeho funkce efektivnější. Naopak pro úlohy, kde je třeba provádět jednoduchý výpočet mnohokrát, je často dobré si napsat vlastní program v co možná nejnižším jazyce, protože pak je výrazně rychlejší, jak ukážeme v následující kapitole.

8 Závislost nalezeného řešení na počátečním odhadu

Zvolíme-li počáteční odhad blízko správného řešení, bude nalezené přibližné řešení (až na chybu metody a zaokrouhlovací chybu) rovno tomuto správnému řešení. Nebude-li počáteční odhad blízký správnému řešení, mohou se výsledky jednotlivých iterací Newtonovy metody vyvíjet i velmi složitým způsobem. V lepším případě dokonvergují k jednomu ze správných řešení, ale mohou i divergovat, či neustále se měnit. Někdy se žertem říká, že Newtonova metoda je dobrá až tehdy, známe-li řešení. Tomu je třeba rozumnět tak, že je dobrá, známe-li odhad s dostatečnou přesností. Proto se Newtonova metoda často používá v numerických metodách typu prediktor–korektor jako druhá část výpočtu, tedy korektor, pro upřesnění výslekdu, který nejdříve odhadneme (předpovíme) prediktorem.

Závislost na počátečním odhadu lze dobře znázornit graficky tímto postupem: Ve zvoleném rozsahu neznámých, např. $x \in \langle -3, 3 \rangle$ a $y \in \langle -3, 3 \rangle$ si vybereme např. 100 ekvidistantních hodnot x a např. 100 ekvidistantních hodnot y. Tím dostaneme 10000 bodů v obdélníkové síti. Pro každý z těchto 10000 bodů zvolí počáteční odhad roven tomuto bodu a provedeme několik iterací Newtonovy metody. Jestliže je výsledek blízký prvnímu správnému řešení, obarvíme obdélníček kolem bodu jednou barvou (např. červenou), jestliže je výsledek blízký druhému správnému řešení, obarvíme obdélníček kolem bodu druhou barvou (např. modrou), a podobně pro třetí a čtvrté správné řešení obarvíme obdélníček nějakou třetí či čtvrtou barvou.

To lze provést v systému Maple např. tímto programem.

```
restart;
Digits:=16;
f  := x^3 + 3*x*y + y^3;
g  := exp(x) - 2*x + exp(y) - y - 3;
fx := diff(f,x);
fy := diff(f,y);
gx := diff(g,x);
gy := diff(g,y);

koren1 := eval([x,y],fsolve({f,g},{x= 1.6,y=-0.8}));
koren2 := eval([x,y],fsolve({f,g},{x=-0.3,y= 1.0}));
koren3 := eval([x,y],fsolve({f,g},{x=-0.8,y=-0.2}));
koren4 := eval([x,y],fsolve({f,g},{x=-0.5,y=-1.1}));
koreny := array([koren1,koren2,koren3,koren4]);

#de := fx * gy - fy * gx;
#plots[implicitplot](de, x=-3..3,y=-3..3, numpoints=100000);

mujnewton1 := proc (x0,y0)
 local f0,g0,detj,detx,dety,dx,dy;
```

```
 f0 := eval(f,{x=x0,y=y0});
 g0 := eval(g,{x=x0,y=y0});
 detj := eval(fx * gy - fy * gx, {x=x0,y=y0});
 detx := eval(f0 * gy - fy * g0, {x=x0,y=y0});
 dety := eval(fx * g0 - f0 * gx, {x=x0,y=y0});
 dx := detx / detj;
 dy := dety / detj;
 x0 - dx, y0 - dy;
end;

xa := -3; xb :=  3.001; ya := -3; yb :=  3.002; nx := 100; nn := 10;
b  := array(1..nx,1..nx);  # barvy
v  := array(1..4);         # vzdalenosti od korenu

for i from 1 to nx do
for j from 1 to nx do
 xy := xa + (xb-xa)*i/nx, ya + (yb-ya)*j/nx; # nastrel
 for k from 1 to nn do xy := mujnewton1(xy); end do;
 for k from 1 to 4  do v[k]:=(xy[1]-koreny[k,1])^2+
                             (xy[2]-koreny[k,2])^2; end do;
 if   v[1]<=v[2] and v[1]<=v[3] and v[1]<=v[4] then b[i,j]:=1;
 elif v[2]<=v[1] and v[2]<=v[3] and v[2]<=v[4] then b[i,j]:=2;
 elif v[3]<=v[1] and v[3]<=v[2] and v[3]<=v[4] then b[i,j]:=3;
 elif v[4]<=v[1] and v[4]<=v[2] and v[4]<=v[3] then b[i,j]:=4;
 else print("tak ted nevim jakou barvu mu mam lidicky dat?");
 end if;
end do;
end do;
with(plots):
listdensityplot(b,colorstyle=HUE,range=1..5);
```

Vysvětlení:

- vyčištění paměti, nastavení přesnosti, zadání levých stran a jejich derivací je stejné jako v předchozích příkladech,
- příkazem `fsolve` si připravíme čtyři správná řešení,
- budeme používat proceduru mujnewton1, kterou už také známe,

- zvolíme rozsahy proměnných `x` a `y`, počet hodnot `nx` a počet iterací Newtonovy metody `nn`,
- připravíme si pole `b` barev a pole `v` vzdáleností nalezeného řešení od čtyřech správných řešení,
- dvojitou `for` smyčkou nastavíme počáteční odhad na jednotlivé body obdélníkové sítě a pro každý takto nastavený bod provedeme `nn` iterací Newtonovy metody,
- spočítáme si vzdálenosti nalezeného řešení od čtyřech správných řešení,
- podle toho, která z těchto čtyřech vzdáleností je nejmenší, uložíme do pole `b` číslo 1, 2, 3 nebo 4,
- příkazem `listdensityplot` vykreslíme pole `b`.

Toto lze sice spočítat pomocí systému Maple, ale výsledek trvá (na počítači s procesorem Pentium III 1 GHz) 94 s, použijeme-li Newtonovu metodu bez speciálních balíků. Použijeme-li balík `linalg`, trvá tentýž výpočet 647 s. Použijeme-li balík `LinearAlgebra`, trvá tentýž výpočet dokonce 863 s. Použijeme-li k řešení příkaz `fsolve`, trvá výpočet 200 s. Jestliže naprogramujeme výpočet v jazyce C, trvá výpočet několik sekund.

Z tohoto příkladu jsou dobře vidět přednosti a nevýhody počítačových algebraických systémů jako jsou Maple, Mathematica a další:

- výhody PAS: pohodlná práce, krátký program,
- nevýhody: výpočet trvá mnohem déle a spotřebuje mnohem více paměti než program napsaný v „nízkém“ programovacím jazyce jako např. C, Fortran nebo Pascal.

Takže tyto „vysoké“ programovací nástroje, jako je Maple apod. použijeme s výhodou pro vyzkoušení a odladění nového algoritmu nebo postupu. Jestliže poté potřebujeme provést výpočet, který by trval příliš dlouho, naprogramujeme si jej podruhé v jazyce, ve kterém přeložený program poběží rychleji.

Tak jsme postupovali i v případě vyšetřování závislosti nalezeného řešení Newtonovou metodou na počátečním odhadu. Vyzkoušeli jsme si navržený algoritmus v systému Maple na mřížce 100 x 100 bodů. Poté jsme zvolili mřížku 1000 x 1000 a napsali program v jazyce C, který vypočtená data zapsal do výstupního textového souboru. Pro vykreslení těchto dat jsme použili systém Mathematica, protože dává větší možnosti nastavit požadované parametry obrázku. Následující grafy ukazují, jak to dopadlo. Obrázky mají tzv. fraktální strukturu, to znamená, že vykreslíme-li si detail pro užší rozsah proměnných, objeví se podobné struktury jako na větším měřítku.

9 Použití kontinuace

Pro velice obtížné úlohy, tj. úlohy, kde levé strany algebraických rovnic rychle mění svůj průběh již v malém okolí správného řešení, mohou metody řešení založené na Newtonově metodě selhat, jestliže neznáme počáteční odhad řešení dostatečně přesně. V takových případech lze s úspěchem použít kontinuační metody.

Princip spočívá v tom, že vedle řešené rovnice

$$F(x) = 0,$$

jejíž kořen $x_F \in R^n$ neznáme, uvažujeme ještě pomocnou rovnici

$$G(x) = 0,$$

jejíž kořen $x_G \in R^n$ známe, např. $G(x) = x - x_G$, kde x_G si zvolíme. Z těchto dvou rovnic vytvoříme třetí rovnici

$$\alpha G(x) + (1 - \alpha) F(x) = 0,$$

kde $\alpha \in \langle 0; 1 \rangle$ je parametr. Řešení $x = x(\alpha)$ této rovnice závisí na parametru α a představuje křivku v R^n. Pro $\alpha = 0$ její řešení $x = x_G$ (počáteční bod této křivky) známe. Tuto křivku lze spočítat metodou kontinuace a tak dostaneme koncový bod $x = x_F$, tedy řešení původní rovnice.

Protože ale moderní počítačové systémy jako Maple či Mathematica obsahují čím dál tím kvalitnější nástroje pro řešení soustav nelineárních algebraických rovnic, které jsou uživateli k dispozici jediným příkazem (např. příkaz `fsolve` v systému Maple), ustupuje důležitost metod založených na kontinuaci či vlastních metod založených na Newtonově metodě.