Numeriska metoder, grundkurs II
Övning 4 för I2

Övningsgrupp 1

Johannes Hjorth
hjorth@nada.kth.se
Rum 163:006, Roslagstullsbacken 35
08 - 790 69 00

Kurshemsida:
http://www.csc.kth.se/utbildning/kth/kurser/2D1240/numi07

Material utdelat på övningarna:

http://www.nada.kth.se/∼hjorth/teaching/numi07

Dagens program

- Ickelinjära ekvationssystem
  Newtons metod
- Överbestämda ickelinjära ekvationssystem
  Gauss-Newtons metod
- Störningsräkning (repetition)
  Hur tillförlitligt är vårt svar?

Newtons metod

Om vi har ett antal ickelinjära ekvationer på formen

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

bildar vi Jacobianen

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix}$$

Nu löser vi systemet $\mathbf{J} \cdot \mathbf{dx} = -\mathbf{f}(\mathbf{x})$

$$\begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} dx \\ dy \end{pmatrix} = - \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

och uppdaterar x och y

$$\begin{aligned} x_1 &= x_0 + dx \\ y_1 &= y_0 + dy \end{aligned}$$

Varför ser ekvationen ut som den gör?

Antag att vi har en startgissning x_0, y_0, vi befinner oss då vid $f(x_0, y_0)$.

Om vi tar ett steg dx i x-led flyttar vi oss $\frac{\partial f}{\partial x} \cdot dx$.

Ett steg dy i y-led förflyttar oss $\frac{\partial f}{\partial y} \cdot dy$.

Så om vi kan hitta dx och dy sådana att

$$\frac{\partial f}{\partial x} \cdot dx + \frac{\partial f}{\partial y} \cdot dy = -f(x_0, y_0)$$

kommer vi förhoppningsvis att komma närmare nollan.

Vi passar givetvis på att lösa för g samtidigt.

Exempel 3.9, Newtons metod

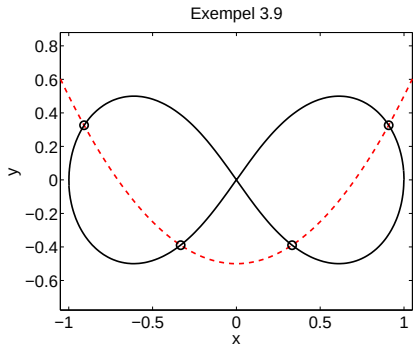
Vi ska hitta skärningspunkterna mellan en lemniskata

$$\left(x^2 + \frac{y^2}{2}\right)^2 = x^2 - \frac{y^2}{2}$$

och en parabel

$$y = x^2 - \frac{1}{2}$$

med andra ord de punkter där båda ekvationerna är uppfyllda.



Vi skriver en funktion på formen $\mathbf{f}(\mathbf{x}) = 0$

```
function z = funk(xv)
x = xv(1); y = xv(2);

z = [(x^2 + y^2/2)^2 - x^2 + y^2/2
     y - x^2 + 1/2];
```

och en funktionsfil `Jac.m` för Jacobianen

```
function J = Jac(xv)
x = xv(1); y = xv(2);

J = [2*(x^2 + y^2/2)*2*x - 2*x   2*(x^2 + y^2/2)*y + y
     -2*x                        1];
```

Sen löser vi med Newtons metod

```
clear all, format compact
xStart = [-1 -0.5 0.5 1]; tol = 1e-7;

for i=1:length(xStart)
  x = [xStart(i); xStart(i)^2 - 0.5]; dx = 1;

  while(norm(dx) > tol)
    dx = -Jac(x)\funk(x);
    x = x + dx
  end
end
```

Newtons metod har kvadratisk konvergens precis som Newton-Raphsons metod, men för vissa svåra problem kan den gå över i linjär konvergens.

```
x =
    0.932065217391304
    0.364130434782609
x =
    0.910588376031954
    0.328709935849736
x =
    0.908848729379520
    0.326002986524293
x =
    0.908838244450336
    0.325986954465636
x =
    0.908838244076430
    0.325986953895928
```

Exempel 4.23, Gauss-Newton

Vi ska bestämma parametrarna a, b, q och t_0

$$F = a\,e^{qt} + b\,e^{-(t-t_0)^2}$$

så att kurvan går nära de givna punkterna.

Först skapar vi en funktionsfil

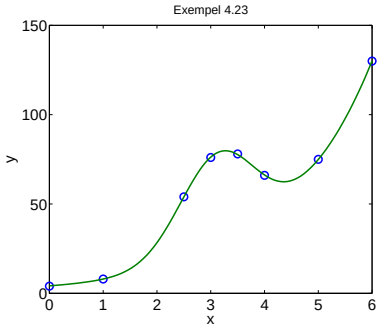
```
function F = funk(y,t,x)

a = x(1); b = x(2); q = x(3); t0 = x(4);

F = a*exp(q*t) + b*exp(-(t-t0).^2) - y;
```

och sedan en fil för Jacobianen

```
function J = Jac(t,x)

a = x(1); b = x(2); q = x(3); t0 = x(4);

J = [exp(q*t) exp(-(t-t0).^2) a*t.*exp(q*t) ...
     b*(2*(t-t0)).*exp(-(t-t0).^2)];
```

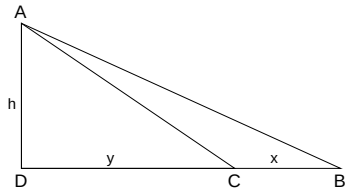
```
clear all, format compact, format long

t = [0 1 2.5 3 3.5 4 5 6]';
y = [4 8 54 76 78 66 75 130]';

q = 0.5; t0 = 3; a = 6; b = 46;

dx = 1; tol = 1e-9;

x = [a b q t0]';

while(norm(dx) > tol)
   dx = -Jac(t,x)\funk(y,t,x);
   x = x + dx
end

tv = linspace(0,6,100);
yv = funk(zeros(size(tv)),tv,x);

plot(t,y,'o',tv,yv,'-')
xlabel('x'), ylabel('y'), title('Exempel 4.23')
```

```
>> tal423
x =                         x =
   4.017371944444375           4.164131868186754
  52.956093842139907          53.779521089482209
   0.568027509070274           0.573537552856143
   3.141929262404280           3.123823447096213
x =                         x =
   4.164745884015918           4.164131900176109
  53.762196527743605          53.779520981171395
   0.573808363763991           0.573537551518655
   3.123525415707931           3.123823446632913
x =                         x =
   4.164148092636323           4.164131900074707
  53.779453113736075          53.779520981514523
   0.573537124281598           0.573537551522895
   3.123823421265514           3.123823446634611
```



Exempel 8.7, störningsräkning

Vi känner sträckan BC, och vinklarna ABC $= \alpha$ samt ACD $= \beta$. Vår uppgift är att beräkna AD.



BC$= 50 \pm 0.2$ m, $\alpha = 32.6 \pm 0.3$, $\beta = 53.8 \pm 0.3$.

Vi låter BC$= x$, CD$= y$ och AD$= h$

$$\begin{aligned} \tan\alpha &= \frac{h}{x+y} \\ \tan\beta &= \frac{h}{y} \end{aligned}$$

löser ut y i båda leden, och skriver om

$$h = \frac{x\tan\alpha}{1-\frac{\tan\alpha}{\tan\beta}}$$

```
function h = funk(x,a,b)
alpha = a*pi/180; beta = b*pi/180; % Radianer!
h = x * tan(alpha) / (1 - tan(alpha)/tan(beta));
```

Separat fil för vårt program:

```
clear all, format compact
x = 50; dx = 0.2;
a = 32.6; da = 0.3;
b = 53.8; db = 0.3;

hRef = funk(x,a,b)
hx = funk(x+dx,a,b);
ha = funk(x,a+da,b);
hb = funk(x,a,b+db);

hErr = abs(hx-hRef) + abs(ha-hRef) + abs(hb-hRef)

h = []; % Alternativt, testa alla kombinationer

for X = [x x+dx x-dx]
    for A = [a a+da a-da]
        for B = [b b+db b-db]
            h(end+1) = funk(X,A,B);
        end
    end
end

hMean= mean(h), hMax = max(h), hMin = min(h)
```

Kör vi koden får vi följande svar

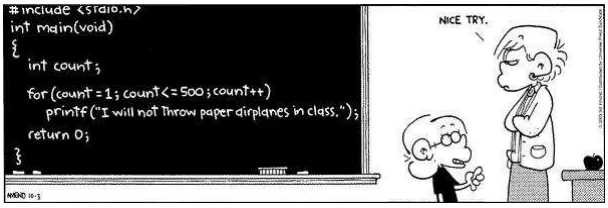
```
>> tal87
hRef =
   60.1130
hErr =
    2.1353
hMean =
   60.1300
hMax =
   62.2981
hMin =
   58.0444
>>
```

Vi kan antingen använda oss av `hRef` $\pm$ `hErr`, eller svara med `hMean` och ange en felgräns som täcker in `hMin` och `hMax`.

Kom ihåg att avrunda felet uppåt!

Störningsräkningen ger oss $h = 60 \pm 3\,\mathrm{m}$.

Undvik for-loopar



Varje gång ni tänker använda en for-loop, fundera på om det inte går att skriva det med någon matrisoperation istället.

```
v = []; dv = 2*pi/10;
for i=1:10
  v = [v; dv*i];
end
```

Matlab är nämligen bra på matrisoperationer

```
v = (1:10)*2*pi/10;
```

eller i det här fallet, kanske hellre

```
v = linspace(0,2*pi,10);
```

Hur plottar man sneda ellipser?

Antag att vi har en ellips som beskrivs av formeln

$$x^2 + 0.7 \cdot y^2 - x \cdot y = 12$$

om vi sätter in

$$\begin{aligned} x &= r \cdot \cos\phi \\ y &= r \cdot \sin\phi \end{aligned}$$

kan vi lösa ut r ur

$$r^2(\cos^2\phi + 0.7 \cdot \sin^2\phi - \cos\phi \sin\phi) = 12$$

Vi kan sedan plotta den sneda ellipsen

```
linspace(0,2*pi,100);

r = sqrt(12./(cos(fi).^2 + 0.7*sin(fi).^2 ...
             - sin(fi).*cos(fi)));

plot(r.*cos(fi), r.*sin(fi))
```

Kör vi koden får vi en tjusig ellips!