Numeriska metoder, grundkurs II
Övning 2 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

- Efter experimentell störningsräkning
  Hur avrundar vi svar med felgränser?
- Minstakvadratmetoden
  Trick för att skriva om ickelinjära ekvationer
- Tjusiga 3D plottar
  Ett första exempel
- Newtons ansats
  Smart sätt att räkna för hand
- Hermite-interpolation och kubiska Bézierkurvor
  Några olika sätt att interpolera

Repetition - Antal korrekta siffror

Kom ihåg att om felet är mindre än $0.5 \cdot 10^{-d}$ så har vi minst d korrekta decimaler.

Allt mellan $a = 3.1499\ldots$ och $b = 3.05$ avrundas till 3.1, så för att ha en korrekt decimal måste vi ligga inom detta intervall, det vill säga ha ett fel mindre än $\frac{a-b}{2} = 0.05$.

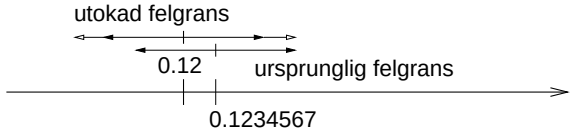
Hur ska vi avrunda svaret?

På förra övningen störde vi indata variabelvis, och summerade beloppen av de olika störningarna i utdata. Antag nu att vi från matlab fått:

$$0.12\underbrace{34567}_{?} \pm 0.01$$

Hur ska vi då avrunda svaret?

Den sista biten av decimalutvecklingen är inte direkt tillförlitlig, så vi avrundar till 0.12, men då har vi ökat felet med 0.0034567 — ajdå!



Vår utökade felgräns blir ± 0.0134567 vilket vi avrundar *uppåt* till ± 0.02 för att täcka osäkerheten.

Exempel 4.15

Vi ska anpassa $T = T_0 \cdot R^q$ till sex uppmätta värden på R och T. Funktionen är dock inte linjär i T_0 och q, och därför måste vi skriva om den först.

$$\ln T = ln(T_0 R^q) = \ln T_0 + q \ln R$$

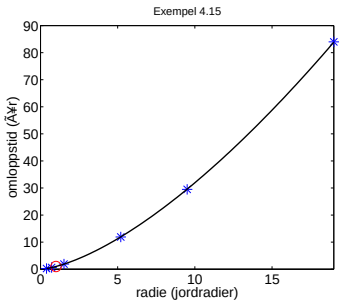
Observera att i det omskrivna ekvationssystemet är $\ln T_0$ och q våra obekanta.

$$\begin{pmatrix} 1 & \ln(0.39) \\ 1 & \ln(0.72) \\ \vdots & \vdots \\ 1 & \ln(19.0) \end{pmatrix} \begin{pmatrix} \ln T_0 \\ q \end{pmatrix} = \begin{pmatrix} \ln(0.24) \\ \ln(0.62) \\ \vdots \\ \ln(84) \end{pmatrix}$$

Sedan löser vi det överbestämda systemet med hjälp av matlab.

```
clear all

R = [0.39 0.72 1.52 5.20 9.50 19.0]';
T = [0.24 0.62 1.88 11.9 29.5 84.0]';

A = [ones(size(R)) log(R)];
y = A\log(T);

T0 = exp(y(1))
q = y(2)

r = linspace(R(1),R(end),100);
plot(R,T,'*',1,1,'o',r,T0*r.^q)
title('Exempel 4.15')
xlabel('radie (jordradier)')
ylabel('omloppstid (år)')
```

Kör vi koden får vi $T_0 = 1.0008$ och $q = 1.5040$.



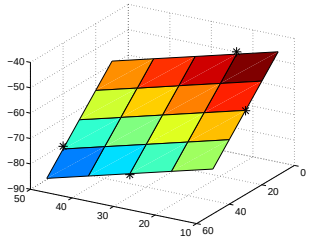
Exempel 4.17

Givet fem punkter ska vi bestämma ett plan.

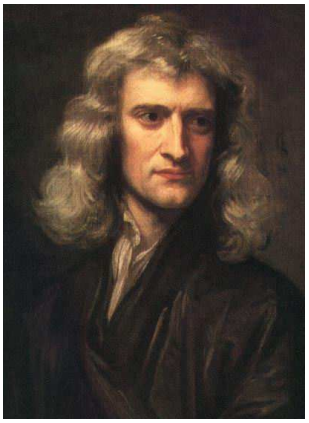
```
clear all, clf
format compact

x = [30 40 10 20 50]';
y = [50 20 30 10 40]';
z = [-81.3 -63.5 -57.0 -44.8 -80.7]';

A = [ones(size(x)) x y];
c = A\z;

xv = linspace(10,50,5); % 5 punkter
yv = linspace(10,50,5);
[X,Y] = meshgrid(xv,yv)

Z = c(1) + c(2)*X + c(3)*Y

% Tjusig 3d-plot
surf(X,Y,Z), hold on
plot3(x,y,z,'*')
```

Normalt använder man fler än 5 punkter i x- och y-led. Här vill jag dock visa hur X och Y matriserna ser ut och stora matriser får inte plats på pappret.

```
>> mkm2d
X =
    10    20    30    40    50
    10    20    30    40    50
    10    20    30    40    50
    10    20    30    40    50
    10    20    30    40    50
Y =
    10    10    10    10    10
    20    20    20    20    20
    30    30    30    30    30
    40    40    40    40    40
    50    50    50    50    50
Z =
  -41.4138  -45.7040  -49.9941  -54.2842  -58.5743
  -49.1468  -53.4369  -57.7270  -62.0171  -66.3073
  -56.8798  -61.1699  -65.4600  -69.7501  -74.0402
  -64.6127  -68.9029  -73.1930  -77.4831  -81.7732
  -72.3457  -76.6358  -80.9259  -85.2160  -89.5062
```



Newtons ansats



Gubben Newton var inte dum. Kan du se varför hans ansats är smartare än den naiva ansatsen?

$$P(x) = c_0 + c_1(x-x_1) + c_2(x-x_1)(x-x_2) + c_3(x-x_1)(x-x_2)(x-x_3) + \dots$$

Vad händer om du beräknat ett andragradspolynom och vill utöka det till ett tredjegradspolynom. Kan du återanvända dina gamla beräkningar?

Exempel 5.3

Vi har punkterna (1,3),(3,3),(4,0),(5,11) och ska beräkna ett tredjegradspolynom genom dem.

Vi sätter in $x_1 = 1$ i ansatsen på föregående slide:

$$c_0 + c_1 \cdot 0 + c_2 \cdot 0 + c_3 \cdot 0 = 3$$

Vi får att $c_0 = 3$. Sätter vi sedan in $x_2 = 3$ får vi,

$$3 + c_1 \cdot (3-1) + c_2 \cdot 0 + c_3 \cdot 0 = 3$$

vilket ger oss $c_1 = 0$. Vi fortsätter med $x_3 = 4$,

$$3 + 2 \cdot (4-1) + c_3 \cdot (4-1)(4-3) = 0$$

som ger $c_2 = -1$. Slutligen ger $x_4 = 5$ oss

$$3+2\cdot(5-1)-(5-1)(5-3)+c_4\cdot(5-1)(5-3)(5-4) = 11$$

och vi får $c_3 = 2$.

Polynomet blir

$$P(x) = 3 - (x-1)(x-3) + 2(x-1)(x-3)(x-4)$$

Notera alla nollorna i räkningarna på föregående slide. Vi kan enkelt utöka vårt tredjegradspolynom till ett fjärdegradspolynom med punkten (7,6),

$$3 - (7-1)(7-3) + 2 \cdot (7-1)(7-3)(7-4) + c_5 \cdot (7-1)(7-3)(7-4)(7-5) = 6$$

vilket ger oss $c_4 = -\frac{117}{144}$.

Vi har utökat Newtons ansats, men eftersom termen

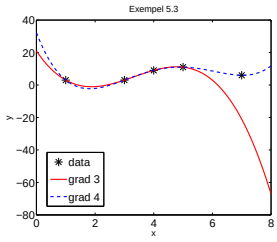
$$c_4 \cdot (x-x_1)(x-x_2)(x-x_3)(x-x_4)$$

kommer vara noll för x_1, x_2, x_3 och x_4 behöver ingen av de andra konstanterna c_0, c_1, c_2 eller c_3 räknas om, underbart!

Om vi ändrar (7,6) till (7,7) får vi istället $c_4 = -\frac{116}{144}$

Vi beräknar den naiva ansatsen mha matlab

```
clear all, clf
xp = [1 3 4 5]'; yp = [3 3 9 11]';

A = [xp.^3 xp.^2 xp ones(size(xp))];
c3 = A\yp;

x = linspace(0,8,100);
% y3 = c3(1)*x.^3 + c3(2)*x.^2 + c3(3)*x + c3(4)
y3 = polyval(c3,x); % funkar om x.^3 till vänster

xp = [xp; 7]; yp = [yp; 6];
c4 = polyfit(xp,yp,4); % Alternativ metod
y4 = polyval(c4,x);

p = plot(xp,yp,'k*',x,y3,'r-',x,y4,'b--');
xlabel('x'), ylabel('y'), title('Exempel 5.3')

legend(p, 'data', 'grad 3','grad 4') % namnge kurvorna
```

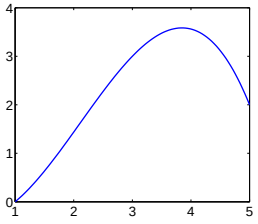


Hermite-interpolation

Vi vill interpolera mellan två punkter, till exempel (1,0) och (5,2). Vi känner även lutningen för dessa, säg $k_1 = 1$ och $k_2 = -3$.

$$\begin{aligned} x(t) &= x_1 + t \cdot h \\ y(t) &= y_1 + t \cdot \Delta y + t(1-t) \cdot g + t^2(1-t) \cdot c \\ g &= h \cdot k_1 - \Delta y \\ c &= 2\Delta y - h(k_1 + k_2) \end{aligned}$$

```
clear all
x1 = 1; y1 = 0; k1 = 1;
x2 = 5; y2 = 2; k2 = -3;
t = linspace(0,1,100);

h = x2-x1; dy = y2 - y1;
g = h*k1-dy;
c = 2*dy - h*(k1 + k2);

x = x1 + (x2 - x1)*t;
y = y1 + t*dy + t.*(1-t).*g ...
    + t.^2.*(1-t).*c;

plot(x,y)
```



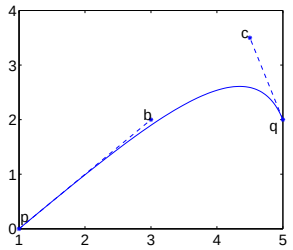
Kubiska Bézierkurvor

Vi vill dra en Bézierkurva mellan **p** och **q**, med hjälp av två styrpunkter **b** och **c**, här är $t \in [0, 1]$.

$$\mathbf{r}(t) = (1-t)^3\mathbf{p} + 3t(1-t)^2\mathbf{b} + 3t^2(1-t)\mathbf{c} + t^3\mathbf{q}$$

Tangenten i **p** är i riktning mot **b**, och i **q** går från **c**.

```
clear all, close all

p = [1,0]; q = [5,2];
b = p + 2*[1,1];
c = q - 0.5*[1,-3];

t = linspace(0,1,100)';

r = (1-t).^3*p + 3*t.*(1-t).^2*b ...
   + 3*t.^2.*(1-t)*c + t.^3*q;

plot(r(:,1),r(:,2)), hold on

v = [p;q;b;c];
plot(v(:,1),v(:,2),'*')
```



Repetition - funktioner

En funktion kan retunera flera värden. Här skapar vi en funktion som beräknar summan.

$$z = a_1 - a_2 + a_3 - a_4 + \ldots$$

```
% Denna fil heter altSum.m
function [x,y,z] = altSum(a)

x = a;
y = ones(size(a));
y(2:2:end) = -1;
z = sum(y.*a);
```

Oftast går det att undvika for-loopar med olika smarta matrisoperationer.

```
>> [x,y,z] = altSum(logspace(-2,2,5))
x =
    0.0100    0.1000    1.0000   10.0000  100.0000
y =
     1    -1     1    -1     1
z =
   90.9100
```

- Icke-linjära ekvationer kan ibland skrivas om på en linjär form, tex genom att logaritmera. Sen kan vi lösa med minstakvadratmetoden.
- Newtons ansats är bra vid handräkning.
- Vi behöver kunna interpolera på Lab1, det handlar mest om att stoppa in i kända formler.
- Använd funktioner för att undvika koduppprepning

Fråga om det är något som är oklart!