Numeriska metoder, grundkurs II
Övning 5 för I2

Övningsgrupp 1

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

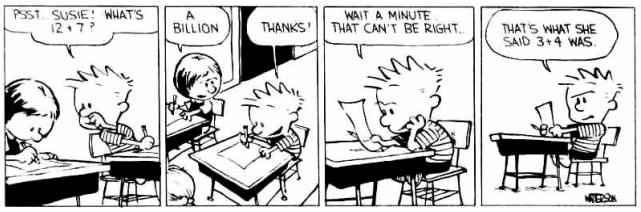
Kurshemsida:
`http://www.csc.kth.se/utbildning/kth/kurser/2D1240/numi07`

Material utdelat på övningarna:

`http://www.csc.kth.se/~hjorth/teaching/numi07`

Dagens program

- Gyllenesnittetminimering
  Hitta funktionsminimum utan att derivera
- Differentialekvationer
  Euler, Runge-Kutta, `ode45`
- Fråga om teorital
  Något speciellt tal vi ska gå igenom?
- Rättelse – övning 4
  Newtons metod har kvadratisk konvergens!



Gyllenesnittminimering, exempel 4.27

Förhållandet mellan olika segment hos snäckor uppfyller approximativt gyllene snittet.

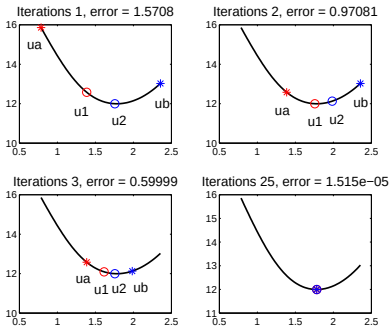
Vi ska dock använda $g = \frac{\sqrt{5}-1}{2}$ till att hitta minimum för en funktion. Inga derivator behövs.

```
function L = len(u)

Q1 = [0;10]; Q2 = [10;10]; R = 4;
P  = [8+R*cos(u); 3+R*sin(u)];

L = norm(Q1-P) + norm(Q2-P);
```

Det som är intressant här är while-slingan och hur punkterna u_a, u_b, u_1 och u_2 uppdateras för varje varv beroende på vilken av u_1 och u_2 som är minst.

```
% Kod för gyllenesnittminimering
clear all

g  = (sqrt(5) - 1)/2;
ua = pi/4;  ub = 3/4*pi;
u1 = ua*g + (1-g)*ub;
u2 = ua*(1-g) + g*ub;

tol = 1e-5;

while(abs(ua-ub) > tol)

  if(len(u1) < len(u2))
    ub = u2; u2 = u1;
    u1 = ua*g + (1-g)*ub;
  else
    ua = u1; u1 = u2;
    u2 = ua*(1-g) + g*ub;
  end

end

mean([ua ub])
```

Vi har valt de inre punkterna u_1 och u_2 så att vi hela tiden kan återanvända en av dem i nästa iteration.




```
>> gyllenesnittminimering
ans =
    1.7813
```

I första iterationen ser vi att u_2 är mindre än u_1, eftersom funktionen är unimodal kan minimum alltså inte finnas mellan u_1 och u_a, eller hur?

Vi låter u_1 bli vårt nya u_a. Då kan u_2 användas som det nya u_1, och vi behöver bara beräkna fram det nya u_2 värdet. Dubbelkolla i figuren!

Euler framåt, exempel 7.1

Antag att vi har en diff-ekvation

$$\frac{dy}{dt} = f(y,t) = 1 + t - y$$

och vi vet att $y(0) = 1$.

Vi kan då lösa den *för hand* med Euler framåt, vilket har förekommit några gånger i gamla tentor.

Med Euler framåt tar vi ett litet steg Δt framåt i derivatans riktning

$$y(t + \Delta t) = y(t) + \Delta t \cdot f(y,t)$$

för att beräkna det nya funktionsvärdet i $t + \Delta t$.

Rita figur och räkna för hand!

Räkna ett steg med $h = 0.2$ och gör sedan om beräkningen med $h = 0.1$, då behövs två steg.

ode45

För att kunna lösa diff ekvationer med matlabs inbyggda funktioner måste vi först skapa en funktion som ode45 kan anropa.

Första argumentet måste vara tiden, oavsett om funktionen har något tidsberoende eller ej.

```
function f = flode(t,h)

A = 0.8; C = 0.1; g = 9.81;

Qin = max(0,1-h/20);
Qut = C*sqrt(2*g*h);

f = (Qin - Qut)/A;
```

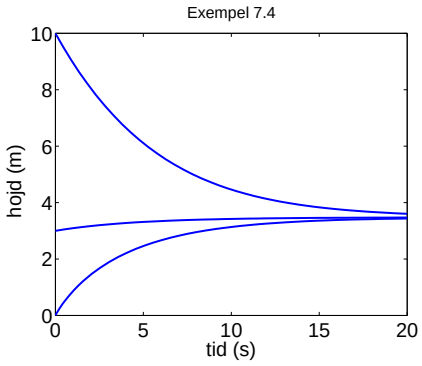
Anropet till ode45 har tre parametrar, namnet på funktionen, tidsintervallet och begynnelsevärden.

```
[t,h] = ode45(@flode,[0 20], h0);
```

Som svar får vi kolumnvektorer med tiden och höjden vid de olika tidpunkterna.

I exemplet ska vi lösa för tre olika startvärden, vilket görs enklast med en for-loop.

```
clear all, close all

for h0 = [0 3 10];

  % Vi löser från t = 0 till t = 20
  [t,h] = ode45(@flode,[0 20], h0);
  plot(t,h), hold on

end

title('Exempel 7.4')
xlabel('tid (s)'), ylabel('höjd (m)')
```



Andra ordningens Runge-Kutta

Andra ordningens Runge-Kutta använder två lutningar för att stega framåt

$$y(t+\Delta t) = y(t) + \frac{h}{2}(f(t,y) + f_2)$$

Här är f_2 en approximation till lutningen i nästa punkt

$$f_2 = f(t+\Delta t, y + h \cdot f(t,y))$$

Steget vi tar utnyttjar medelvärdet av dessa två lutningarna. Metoden är mest av teoretiskt intresse.

Rita figur!

Fjärde ordningens Runge-Kutta

Vi kommer att använda oss av fjärde ordningens Runge-Kutta. Nästa steg beräknas ur

$$y(t+\Delta t) = y(t) + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

där

$$\begin{aligned} f_1 &= f(t,y) \\ f_2 &= f(t+\frac{h}{2}, y+\frac{h}{2}\cdot f_1) \\ f_3 &= f(t+\frac{h}{2}, y+\frac{h}{2}\cdot f_2) \\ f_4 &= f(t+h, y+h\cdot f_3) \end{aligned}$$

Exempel 7.7

Vi har att $\frac{dy}{dx} = f(x,y)$ och ska beräkna en lösning till differentialekvationen med hjälp av fjärde ordningens Runge-Kutta.

```
nX = 8; x = 0; y = 0; h = 0.8/nX;

xv = x; yv = y; % Vektorer att spara x,y värdena i

for i = 1:nX

  f1 = funk(x,y);
  f2 = funk(x+h/2,y + h/2*f1);
  f3 = funk(x+h/2,y + h/2*f2);
  f4 = funk(x+h,y + h*f3);

  x = x + h;
  y = y + h/6 *(f1 + 2*f2 + 2*f3 + f4);

  xv = [xv x]; yv = [yv y];

end
```

Koden fortsätter på nästa slide...

För sista biten löser vi x som en funktion av y.

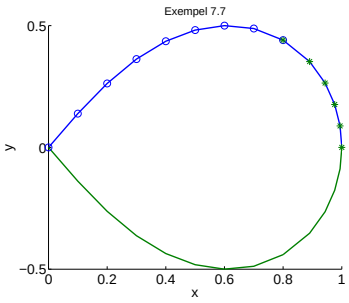
```
nY = 5; h = y/nY; x = 1; y = 0;

xv2 = x; yv2 = y;

for i = 1:nY

  f1 = ifunk(y,x);
  f2 = ifunk(y+h/2,x + h/2*f1);
  f3 = ifunk(y+h/2,x + h/2*f2);
  f4 = ifunk(y+h,x + h*f3);

  y = y + h;
  x = x + h/6 *(f1 + 2*f2 + 2*f3 + f4);

  xv2 = [xv2 x]; yv2 = [yv2 y];

end

clf, hold on, plot(xv,yv,'o',xv2,yv2,'*')

plot(xv,yv, xv,-yv), plot(xv2,yv2,xv2,-yv2)

title('Exempel 7.7'), xlabel('x'), ylabel('y')
```

Vi behöver funktionen för $y' = f(x, y)$, observera att nollan specialbehandlas.

```
function f = funk(x,y)

if(x == 0)
  f = sqrt(2); % Fås mha L'Hospital
else
  f = 2*x/y * (2/(1 + 2*x^2 + y^2) - 1);
end
```

och funktionen för $x' = 1/f(x, y)$

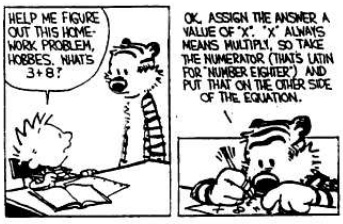
```
function f = ifunk(y,x)

f = y/(2*x)/(2/(1+2*x^2+y^2)-1);
```

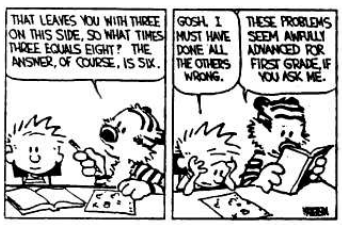


Teorital

Titta igenom teoritalen, är det något speciellt tal som är extra svårt?

Maila eller säg till så kan vi försöka hinna gå igenom det på sista övningen.





Härledning av feltermer

Vi kan härleda feltermerna för framåt och central differanskvoten med hjälp av taylorutvecklingar.

$$f'(x+h) = f(x)+h\cdot f'(x)+\frac{h^2}{2!}\cdot f''(x)+\frac{h^3}{3!}\cdot f'''(x)+\ldots$$

Börja med att få $f'(x)$ ensamt på en sida, så faller pusselbitarna på plats. **Vänta inte, gör det nu!**

För centraldifferanskvoten behöver vi utvecklingen för $f'(x-h)$ också. Subtrahera sedan den från utveckligen av $f(x+h)$. Återstår få $f'(x)$ ensam.

Gauss-kvadratur

När man ska beräkna en integral är ett sätt att placera ut punkterna jämnt i intervallet. Det finns dock andra sätt att göra det på.

Vi kan tillåta oss att placera ut punkterna var som helst på intervallet, samt att multiplicera de olika funktionsvärdena med konstanter.

$$\int_{-1}^{1} f(x)dx \approx w_1 f(x_1) + w_2 f(x_2)$$

Tanken med Gauss-kvadratur är att det ska vara exakt för polynom upp till en viss given grad.

Hur väljer vi punkter och vikter?

Lösningen ska vara exakt för

$$f(x) = \begin{cases} 1 \\ x \\ x^2 \\ x^3 \end{cases}$$

Härledning följer

$$\begin{aligned} w_1 + w_2 &= \int_{-1}^{1} 1dx = [x]_{-1}^{1} = 1 + 1 = 2 \\ w_1 x_1 + w_2 x_2 &= \int_{-1}^{1} x dx = \left[\frac{x^2}{2}\right]_{-1}^{1} = \frac{1}{2} - \frac{1}{2} = 0 \\ w_1 x_1^2 + w_2 x_2^2 &= \int_{-1}^{1} x^2 = \left[\frac{x^3}{3}\right]_{-1}^{1} = \frac{1}{3} + \frac{1}{3} = \frac{2}{3} \\ w_1 x_1^3 + w_2 x_2^3 &= \int_{-1}^{1} x^3 = \left[\frac{x^4}{4}\right]_{-1}^{1} = \frac{1}{4} - \frac{1}{4} = 0 \end{aligned}$$

En lösning till detta är (sätt in och kontrollera)

$$x_1 = -\frac{1}{\sqrt{3}}, x_2 = \frac{1}{\sqrt{3}}, w_1 = 1, w_2 = 1$$

Praktisk användning

Gauss-kvadratur är härlett för

$$\int_{-1}^{1} f(x)dx$$

men vi har ofta andra integrationsgränser

$$\int_{a}^{b} f(x)dx$$

Vi måste då göra en variabeltransformation, tex

$$\begin{aligned} x &= \frac{(b-a)\cdot t + a + b}{2} \\ dx &= \frac{b-a}{2} \cdot dt \end{aligned}$$

Vilket ger oss för två punkter, $n = 2$

$$\begin{aligned} \int_{a}^{b} f(x)dx &= \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{(b-a)\cdot t + a + b}{2}\right) dt \\ &\approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{(b-a)\cdot t + a + b}{2}\right) \end{aligned}$$

Horners algoritm

Säg att vi har ett polynom

$$y = c_0 + c_1 \cdot x + c_2 \cdot x^2 + c_3 \cdot x^3$$

Att beräkna en punkt kräver då många multiplikationer, med Horners algoritm kan man reducera antalet.

$$y = \underbrace{(\underbrace{(\underbrace{c_3 \cdot x}_{en} + c_2) \cdot x}_{två} + c_1) \cdot x}_{tre} + c_0$$

Tre multiplikationer istället för $1 + 2 + 3 = 6$.

Tidsåtgång

För gausseliminering är tidsåtgången proportionell mot n^3. Det betyder att om vi fördubblar storleken på systemet tar det $2^3 = 8$ gånger längre tid.

Tidsåtgången för lösning av triangulära matriser är n^2, vi kan använda detta sambandet för att uppskatta körtiden.

Tentatal på detta bygger ofta på att vi vet att tex $t = C \cdot n^3$, så får man hur lång tid det tar för ett visst n, då kan C bestämmas. Sedan är uppgiften att räkna fram hur lång tid det tar för ett större n.

För tridiagonala system växer tidsåtgången linjärt.

Då lagrar vi bara de nollskillda elementen, dvs $3 \cdot n$ för tridiagonalen istället för n^2. Tar mindre plats.

För stora system kan det vara bra att använda glesa matriser i matlab (`sparse`), tex då vi använder oss av bandmatrismetoden.