Simulering av differenslikninger

Forelesning uke 37, 2007

MAT-INF1100

Løsning av differenslikninger i formel

Mulig for lineære likninger med konst. koeff. og enkelte inhomogeniteter.
Eksempel: (b, c er konstante)

$$x_{n+2} + bx_{n+1} + cx_n = \cos(n), \quad x_0 = b_0, \quad x_1 = b_1.$$

1. Løs karakteristisk likning $r^2 + br + c = 0$
2. Bestem hvilket tilfelle vi har (2 reelle røtter, sammenfalne røtter, komplekse røtter) og sett opp generell $x_n^{(h)}$.
3. Finn en partikulærløsning $x_n^{(p)}$, dersom mulig
4. Bestem de valgbare konstantene i $x_n^{(h)}$ slik at $x_n^{(h)} + x_n^{(p)}$ oppfyller initialbetingelsene $x_0 = b_0$, $x_1 = b_1$

Hva kan løses i formel; eksempler

Likning	løsning
$x_{n+2} + 2x_n = n^2$	ja
$x_{n+2} + 2\sin(n)x_n = 0$	nei
$x_{n+2} + 2x_n = \sqrt{1+n}$	nei
$x_{n+2} + 3x_{n+1} + (x_n)^2 = 0$	nei
$x_{n+3} + 3x_{n+1} + x_n = e^n$	ja
$x_{n+1} + (n+1)x_n = 0$	ja, spesialtilfelle

Tallsvar fra differenslikninger

Vi har tre naturlige måter å **beregne** løsninger av differenslikninger

1. Kode formelløsningen.
   Grei jobb på kalkulator eller datamaskin
2. Kode stegene i prosedyren beskrevet på tidligere lysark.
   En litt krevende oppgave på vårt nivå– langt program
3. Direkte simulering.
   Enkelt og generelt

De to første kan bare brukes når en formelløsning kan finnes.
Vi skal se mer på det siste alternativet.

Differenslikning av orden 2

$$x_{n+2} + bx_{n+1} + cx_n = 0, \quad x_0 = b_0, \quad x_1 = b_1, \quad (1)$$

Klassifiering: Lineær, av annen orden, konstante koeffisienter, homogen $\Rightarrow$ formel for generell løsning

I stedet: direkte simulering av (1)
Program burde

- Spørre etter og lese parametere b, c, b_0, b_1 og maks. n (**nmax**) fra skjerm
- Skrive resultat til fil eller plotte

For ikke å foregripe INF1100: parametere settes i program og vi skriver til "skjerm"

Metode og testing

Metode

Likning skrives eksplisitt:

$x_{n+2} = -bx_{n+1} - cx_n, \quad n = 0, 1, 2...$

eller

$x_i = -bx_{i-1} - cx_{i-2}, \quad i = 2, 3, ...$

Beregner x_i fortløpende

Testeksempel: Fibonaccifølgen

$b = c = -1, \; b_0 = b_1 = 1 \Rightarrow$

$x_i = x_{i-1} + x_{i-2} = 0, \quad x_0 = 1, \quad x_1 = 1$

Svar: 1, 1, 2, 3, 5, 8, 13, ...

Merk: Vi koder ikke Fibonaccilikningen med de spesifikke koeffisientene $b = c = -1$, men tilpasser generelt program

Metode med array; program andarr.py

```
b=-1.0; c=-1.0; b0=1.0; b1=1.0; nmax=6 #Fibonacci

x=[b0,b1] #initialiserer array

for i in [0,1]: print i,x[i]

i=2

while (i <=nmax):
    x.append(-b*x[i-1]-c*x[i-2])
    print i,x[i]
    i=i+1
```

Simulering

Kjøring ($>$ er promptet fra et shell)

```
>python andarr.py
0 1.0
1 1.0
2 2.0
3 3.0
4 5.0
5 8.0
6 13.0
```

Fibonaccifølgen er gjenskapt
Program bør alltid sjekkes mot kjente resultater

Modifisert program

$x_{n+2} = -bx_{n+1} - cx_n$

Vi regner ut en ny x fra bare de to siste foregående x'er $\Rightarrow$ unøvendig å ta vare på hele arrayen $\Rightarrow$ unødvendig bruk av minne
Viktig? Ikke her, men kan være det andre ganger

Alternativ løsning

Tar bare vare på to siste x: **xi, xim**
Ved inngang til sykel i inneholder **xi, xim** hhv. x_{i-1}, x_{i-2}
Ved utgang inneholder **xi, xim** hhv. x_i, x_{i-1}

Forsøk uten array; program andfeil.py

```
b=-1.0; c=-1.0; b0=1.0; b1=1.0; nmax=6 #Fibonacci

xim=b0; xi=b1 #initialisering

print 0,xim; print 1,xi

i=2

while (i <=nmax):
   xi=-b*xi-c*xim
   xim=xi  # maa oppdatere forrige x
   print i,xi
   i=i+1
```

Resultat

```
>python andfeil.py
0 1.0
1 1.0
2 2.0
3 4.0
4 8.0
5 16.0
6 32.0
```

HELT FEIL

Vi har overskrevet **xi** før vi setter **xim=xi**

Hvorfor får vi $x_i = 2^{i-1}$?

Bøtemiddel: melomlagring

Bruk av mellomlagring; program andmellom.py

```
b=-1.0; c=-1.0; b0=1.0; b1=1.0; nmax=6 #Fibonacci

xim=b0; xi=b1 #initialiserer

print 0,xim; print 1,xi

i=2

while (i <=nmax):
   tmp=xi # tar vare paa gammel xi
   xi=-b*xi-c*xim
   xim=tmp  # maa oppdatere forrige x
   print i,xi
   i=i+1
```

Kjøring

```
>python andmellom.py
0 1.0
1 1.0
2 2.0
3 3.0
4 5.0
5 8.0
6 13.0
```

-Tenk igjennom hvordan formelene i *Kalkulus, kap 4.1–2* kan kodes. Program blir da **mye** lengre enn **andmellom.py**.
-Det er mulig å trikse seg unna mellomlagring (oppgave i *Kompendium(gml.)*)

Nytt eksempel

Likner på et i *Kompendium(gml.), 4.2*.
$b = -31/3$, $c = 10/3$, $b_0 = 1$, $b_1 = 1/3 \Rightarrow$

$$x_{n+2} - \frac{31}{3}x_{n+1} + \frac{10}{3}x_n = 0, \quad x_0 = 1, \quad x_1 = \frac{1}{3}$$

Karakteristisk polynom=0:

$$r^2 - \frac{31}{3}r + \frac{10}{3} = 0 \Rightarrow r_1 = \frac{1}{3}, r_2 = 10$$

Løsning som oppfyller initialbetingelser

$$x_n = \left(\frac{1}{3}\right)^n$$

Utvidelser av program; program nyeks.py

Motivasjon

Vi trenger raskt å kunne variere på **nmax**
Bør ikke stadig editere fil $\Rightarrow$ stress og feil
nmax kunne vært lest fra skjerm, men hentes fra kommandolinje

Kommandolinjeargument

Kommando : >**python nyeks.py 4**
Argument kan hentes fra liste **sys.argv** ved

```
import sys
 ...
nmax=int(sys.argv[1]) #nmax fra kommandolinje
...
```

Se *Elements of computer programming, kap. 3.2.*
Merk **sys.argv[0]** inneholder programnavn

Program nyeks.py

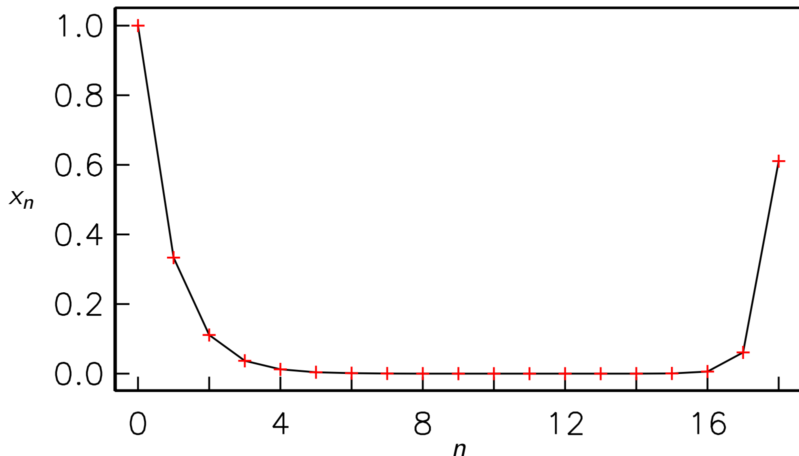
```
import sys
b=-31.0/3.0; c=10.0/3.0; b0=1.0; b1=1.0/3.0
nmax=int(sys.argv[1]) #nmax fra kommandolinje
xim=b0; xi=b1 #initialiserer

print 0,xim; print 1,xi

i=2

while (i <=nmax):
   tmp=xi # tar vare paa gammel xi
   xi=-b*xi-c*xim
   xim=tmp  # maa oppdatere forrige x
   print i,xi
   i=i+1
```

Kjøring

```
>python nyeks.py 4
0 1.0
1 0.333333333333
2 0.111111111111
3 0.037037037037
4 0.0123456790124
```

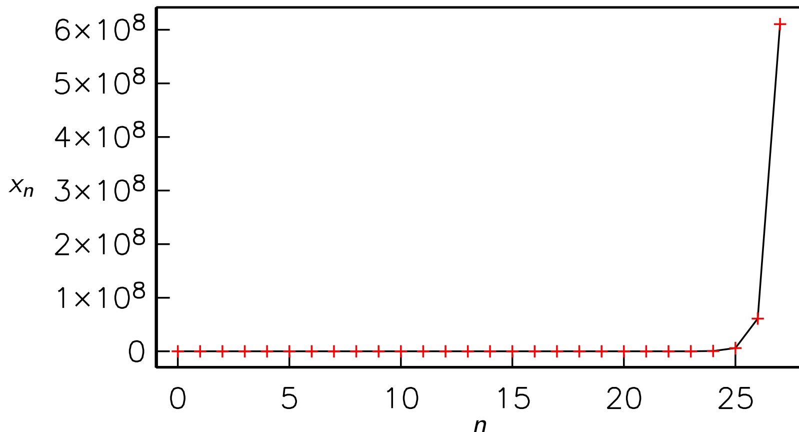
Løsning nær $x_n = (\frac{1}{3})^n$ (sjekk selv med kalkulator).
Skal ha $x_n \to 0$ når $n \to \infty$.
Vi bruker grafisk framstilling av løsning for større n

Graf for større n



Det ser bra ut fram til ca. $n = 15$. Da går det galt!

Enda større n



"Til himmels!". Hvorfor?

Hva går galt ?

Feilkilder i simulering

1. Initialbetingelser representeres ikke eksakt
2. Koeffisienter er heller ikke eksakte
3. Avrundingsfeil påvirker utregning av hver ny x_n

pkt. 1 gir litt gale initialbetingelser; løsning modifiseres ala

$$x_n = (1+\delta)\,(1/3)^n + \epsilon\,(10)^n\,, \qquad (2)$$

Nå vil $x_n \to \pm\infty$ når $n \to \infty$. Python bruker 64 bits flyttall $\Rightarrow$ 15-17 desimale siffer (*Kompendium(nytt), Fact 4.4*) $\Rightarrow$ feil av orden 10^{-17} (minst) $\Rightarrow$ δ og ϵ er av størrelse 10^{-17}
Før ca. $n = 17$ blir det andre leddet i (2) av orden 1.

Uansett hvor mange bit vi bruker i flyttallene vil dette fenomenet dukke opp for tilstrekkelig stor n pga. avrunding i pkt. 1–3