MAT-INF1110 - Oblig 1

Joakim Flatby

4. november 2016

1

$$x_{n+2} - 2x_{n+1} - x_n = 0$$

a)

$$x_{n+2} = 2x_{n+1} + x_n$$

$x_0 = 1$ og $x_1 = 1$

Lager en liste som inneholder x_0 og x_1. Looper 98 ganger og setter inn verdiene opp til x_{100}. Siden x_0 og x_1 allerede er satt, vil x.append sette x_2 første gang, x_3 andre gang, osv.

```
x = [1, 1]
for n in range(0, 99):
    x.append(2 * x[n+1] + x[n])
print x
```

b)

Samme, med $x_0 = 1$ og $x_1 = 1 - \sqrt{2}$

```
from math import sqrt

x = [1, 1-sqrt(2)]
for n in range(0, 99):
    x.append(2 * x[n+1] + x[n])
print x
```

c)

Den karakteristiske ligningen til $x_{n+2} - 2x_{n+1} - x_n = 0$ blir

$$r^2 - 2r - 1 = 0$$

Finner røtter ved å sette inn i abc-formelen.

$$r = \frac{2 \pm \sqrt{4 - 4 * 1 * (-1)}}{2 * 1}$$
$$r = \frac{2 \pm \sqrt{8}}{2}$$
$$r = \frac{2 \pm 2 * \sqrt{2}}{2}$$
$$r = 1 \pm \sqrt{2}$$

$r_1 = 1 - \sqrt{2}$

og

$r_2 = 1 + \sqrt{2}$

Siden ligningen har to reelle røtter, vil den ha generell løsning:

$$x_n = Cr_1^n + Dr_2^n$$

Setter inn røttene som regnet ut over, og har dermed vist at den generelle løsningen er

$$x_n = C(1 - \sqrt{2})^n + D(1 + \sqrt{2})^n$$

For den endelige løsningen har vi

$x_0 = 1 = C + D$

og

$x_1 = 1 - \sqrt{2} = C(1 - \sqrt{2}) + D(1 + \sqrt{2})$

Løser for C og D.

$$1 = C + D \quad \text{(I)}$$
$$C = 1 - D \quad \text{(I)}$$
$$1 - \sqrt{2} = C(1 - \sqrt{2}) + D(1 + \sqrt{2}) \quad \text{(II)}$$

Setter inn $1 - D$ for C

$$1 - \sqrt{2} = (1 - D)(1 - \sqrt{2}) + D(1 + \sqrt{2}) \quad \text{(II)}$$
$$1 - \sqrt{2} = 1 - \sqrt{2} - D + D\sqrt{2} + D + D\sqrt{2} \quad \text{(II)}$$
$$1 - \sqrt{2} = 1 - \sqrt{2} + 2D\sqrt{2} \quad \text{(II)}$$
$$2D\sqrt{2} = 0 \quad \text{(II)}$$
$$D = 0 \quad \text{(II)}$$

Setter inn 0 for D

$$1 = C + 0 \quad \text{(I)}$$
$$C = 1 \quad \text{(I)}$$

Altså er $C = 1$ og $D = 0$

Hvis vi setter dette inn i den generelle løsningen får vi:

$$x_n = 1(1 - \sqrt{2})^n + 0(1 + \sqrt{2})^n$$

dermed er den endelige løsningen

$$x_n = (1 - \sqrt{2})^n$$

d)

Den analytiske løsningen i c stemmer for lave n, men blir feil etterhvert som n øker. Jeg vet ikke helt hvorfor, men tenker det kan ha noe med at tallene i utregningen blir så små at maskinen ikke kan behandle mindre tall.

2

a)

```
1
2  def binomialkoeffisient(n, i):
3      produkt = 1
4      for j in xrange(1, n-i+1):
5          produkt *= (float(i)+j)/j
6
7      return produkt
8
9  print binomialkoeffisient(9998, 4)
10 print binomialkoeffisient(100000, 70)
11 print binomialkoeffisient(1000, 50000)
12 #4.16083629103e+14
13 #8.14900007814e+249
14 #2.70288240945e+299
```

Resultatene blir riktige(hvis man bruker float)

Grunnen til at man må bruke flyttall er så python skal forstå at den må gjøre desimaltallsdivisjon. Når alle variablene i en utregning er heltall, så vil python bare gjøre heltallsdivisjon.

Dermed vil f.eks $\frac{1}{2} = 0$ istedenfor 0.5

Hadde jeg derimot skrevet $\frac{1.0}{2}$ ville svaret vært 0.5

Det holder altså at én av variablene er float, derfor har jeg satt float(i) i koden.

(Men man må passe på at den variablen er med i den første delen av utregningen. For eksempel vil $1/2 * 4.0$ være lik 0.0, mens $1.0/2 * 4 = 2.0$)

b)

Ja, drar man det langt nok med størrelsen på tallene vil maskinen få overflow likevel.

c)

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$\binom{n}{i} = \frac{(n-i+1)}{1}\frac{(n-i+2)}{2}\cdots\frac{n}{i}$$

$$\binom{n}{i} = \prod_{j=1}^{i}\frac{n-i+j}{j}$$

3

a)

Programmet tar to utregninger som matematisk sett gir samme svar:

$$(x + y)(x - y)$$

og

$$x^2 - y^2$$

og prøver med 10000 forskjellige tilfeldige tall for x og y, og ser om de gir samme svar.

Utskriften forteller oss at 48.76% av utregningene ga forskjellige svar.

Det siste paret med tilfeldige tall som ga forskjellig svar var 0.8206445131955394 og 0.6733852440761772, og forskjellen mellom svarene på de to utregningene var -2.7755575615628914e-17

b)

Hvorfor antallet feil er så forskjellig mellom de to utregningene vet jeg ikke, men det ser ut til å være ca. dobbelt så mange feil i a). Jeg tenker det har noe med at å opphøye små desimaltall i 2 gir så små tall at maskinen ikke klarer å regne med det.