# MAT-INF 1100 Obligatorisk oppgave 1

Kenneth Ramos Eikrehagen

4. oktober 2016

## Innhold



## 1 Oppgave 1.

**a)**

Jeg lagde et dataprogram til differensligningen:

$x_{n+2} - 2x_{n+1} - x_n = 0$ med $x_0 = 1$ og $x_1 = 1$.

Jeg brukte at $x_{n+2} = 2x_{n+1} + x_n$ i programmet.

```
x = [1, 1]

for i in range(2,100+1):
    value = 2*x[-1] + x[-2]
    print  x%d %d  %(i,value)
    x.append(value)

"""
Terminal> python oblig1.py
x2 3
x3 7
x4 17
x5 41
x6 99
x7 239
x8 577
x9 1393
x10 3363
x11 8119
x12 19601
x13 47321
x14 114243
x15 275807
x16 665857
x17 1607521
x18 3880899
x19 9369319
x20 22619537
```

```
x21  54608393
x22  131836323
x23  318281039
x24  768398401
x25  1855077841
x26  4478554083
x27  10812186007
x28  26102926097
x29  63018038201
x30  152139002499
x31  367296043199
x32  886731088897
x33  2140758220993
x34  5168247530883
x35  12477253282759
x36  30122754096401
x37  72722761475561
x38  175568277047523
x39  423859315570607
x40  1023286908188737
x41  2470433131948081
x42  5964153172084899
x43  14398739476117879
x44  34761632124320657
x45  83922003724759193
x46  202605639573839043
x47  489133282872437279
x48  1180872205318713601
x49  2850877693509864481
x50  6882627592338442563
x51  16616132878186749607
x52  40114893348711941777
x53  96845919575610633161
x54  233806732499933208099
x55  564459384575477049359
x56  1362725501650887306817
x57  3289910387877251662993
x58  7942546277405390632803
x59  19175002942688032928599
x60  46292552162781456490001
x61  111760107268250945908601
x62  269812766699283348307203
x63  651385640666817642523007
x64  1572584048032918633353217
x65  3796553736732654909229441
x66  9165691521498228451812099
x67  22127936779729111812853639
x68  53421565080956452077519377
x69  128971066941642015967892393
x70  311363698964240484013304163
x71  751698464870122983994500719
x72  1814760628704486452002305601
x73  4381219722279095887999111921
x74  10577200073262678228000529443
x75  25535619868804452344000170807
x76  61648439810871582916000871057
x77  148832499490547618176001912921
x78  359313438791966819268004696899
x79  867459377074481256712011306719
x80  2094232192940929332692027310337
x81  5055923762956339922096065927393
x82  12206079718853609176884159165123
x83  29468083200663558275864384257639
x84  71142246120180725728612927680401
x85  171752575441025009733090239618441
x86  414647397002230745194793406917283
x87  1001047369445486500122677053453007
x88  2416742135893203745440147513823297
```

```
x89 5834531641231893991002972081099601
x90 14085805418356991727446091676022499
x91 34006142477945877445895155433144599
x92 82098090374248746619236402542311697
x93 198202323226443370684367960517767993
x94 478502736827135487987972323577847683
x95 1155207796880714346660312607673463359
x96 2788918330588564181308597538924774401
x97 6733044458057842709277507685523012161
x98 16255007246704249599863612909970798723
x99 39243058951466341909004733505464609607
x100 94741125149636933417873079920900017937
"""
```

**b)**

Her er hvordan programmet ser ut når startverdiene endres til $x_0 = 1$ og $x_1 = 1 - \sqrt{2}$

```
from math import sqrt

x = [1, (1 - sqrt(2))]

for i in range(2,100+1):
    value = 2*(float(x[-1])) + float(x[-2])
    print i, value
    x.append(value)

"""
Terminal> python oblig1_2.py
2 0.171572875254
3 -0.0710678118655
4 0.0294372515229
5 -0.0121933088198
6 0.00505063388334
7 -0.00209204105308
8 0.000866551777181
9 -0.000358937498718
10 0.000148676779744
11 -6.15839392302e-05
12 2.55089012837e-05
13 -1.05661366627e-05
14 4.37662795827e-06
15 -1.81288074619e-06
16 7.50866465893e-07
17 -3.11147814402e-07
18 1.28570837088e-07
19 -5.40061402265e-08
20 2.05585566349e-08
21 -1.28890269568e-08
22 -5.21949727883e-09
23 -2.33280215145e-08
24 -5.18755403078e-08
25 -1.2707910213e-07
26 -3.06033744568e-07
27 -7.39146591267e-07
28 -1.7843269271e-06
29 -4.30780044547e-06
30 -1.0399927818e-05
31 -2.51076560815e-05
32 -6.06152399811e-05
33 -0.000146338136044
34 -0.000353291512069
35 -0.000852921160181
36 -0.00205913383243
37 -0.00497118882504
```

```
38 -0.0120015114825
39 -0.0289742117901
40 -0.0699499350627
41 -0.168874081915
42 -0.407698098894
43 -0.984270279703
44 -2.3762386583
45 -5.7367475963
46 -13.8497338509
47 -33.4362152981
48 -80.7221644471
49 -194.880544192
50 -470.483252832
51 -1135.84704986
52 -2742.17735254
53 -6620.20175494
54 -15982.5808624
55 -38585.3634798
56 -93153.307822
57 -224891.979124
58 -542937.26607
59 -1310766.51126
60 -3164470.2886
61 -7639707.08846
62 -18443884.4655
63 -44527476.0195
64 -107498836.504
65 -259525149.028
66 -626549134.561
67 -1512623418.15
68 -3651795970.86
69 -8816215359.88
70 -21284226690.6
71 -51384668741.1
72 -1.24053564173e+11
73 -2.99491797087e+11
74 -7.23037158346e+11
75 -1.74556611378e+12
76 -4.21416938591e+12
77 -1.01739048856e+13
78 -2.45619791571e+13
79 -5.92978631998e+13
80 -1.43157705557e+14
81 -3.45613274313e+14
82 -8.34384254183e+14
83 -2.01438178268e+15
84 -4.86314781954e+15
85 -1.17406774218e+16
86 -2.8344502663e+16
87 -6.84296827479e+16
88 -1.65203868159e+17
89 -3.98837419065e+17
90 -9.62878706289e+17
91 -2.32459483164e+18
92 -5.61206836958e+18
93 -1.35487315708e+19
94 -3.27095315112e+19
95 -7.89677945932e+19
96 -1.90645120697e+20
97 -4.60258035988e+20
98 -1.11116119267e+21
99 -2.68258042134e+21
100 -6.47632203535e+21
"""
```

**c)**

Jeg bruker at $x_n = C \times r^n$ på ligningen $x_{n+2} - 2x_{n+1} - x_n = 0$ som gir den karakteristiskeligningen $r^2 - 2r - 1$. Jeg løser den ved hjelp av abc formelen:

$$\frac{2^{+}_{-}\sqrt{2^2 - (4 \times 1 \times (-1))}}{2} = \frac{2^{+}_{-}\sqrt{8}}{2} = \frac{2^{+}_{-}2\sqrt{2}}{2} = 1^{+}_{-}\sqrt{2} \tag{1}$$

Jeg får to røtter $r_1 = 1 - \sqrt{2}$ og $r_2 = 1 + \sqrt{2}$
Ut i fra Lemma 4.1.4 blir den generelle løsningen til $x_{n+2} - 2x_{n+1} - x_n = 0$

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

Får å få en endelig løsning setter jeg inn startverdiene $x_0 = 1$ og $x_1 = 1 - \sqrt{2}$ inn i den generelle løsningen.

$$1 = x_0 = C(1 - \sqrt{2})^0 + D(1 + \sqrt{2})^0 = C + D \Rightarrow C = 1 - D$$

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

$$= (1 - \sqrt{2} - D + D\sqrt{2}) + D + D\sqrt{2} = 1 - \sqrt{2} + 2D\sqrt{2} = 2D\sqrt{2} \Rightarrow D = 0 \Rightarrow C = 1 - 0$$

Setter inn C og D i den generelle løsningen og får :

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

**d)**
Løsningen i c) stemmer ikke med beregningene mine i b). Dette er pga avrundnings feil. I c) ser man at $(1 + \sqrt{2})^n$ forsvinner, altså blir akkurat 0.0, men på datamaskinen skjer det en avrundnings feil her. I steden for å bli 0.0 blir svaret $10^{-16}$ dermed faller ikke den bort. Dette medfører at det blir feil når n blir stor. Avviket i starten er liten, men alikvel vokser den. Vi ser allerede når n = 20 ($x_{20}$) at svaret endre seg fra å gå mot null til å stige. Når jeg kommer til n = 40 ($x_{40}$) er avviket allerede på $6.99 \cdot 10^{-2}$ på n = 60 ($x_{60}$) er avviket på $3.16 \cdot 10^{6}$! på n = 100($x_{100}$) er avviket kommet på $6.48 \cdot 10^{21}$! Avrundningsfeilen har store konsekvenser for denne følgen. Jeg legger ved en kode jeg brukte for å studere feilen nærmere.

```
from math import sqrt

x = [1, (1 - sqrt(2))]

for i in range(2,100+1):
    value = 2*(float(x[-1])) + float(x[-2])
    x.append(value)

def xn(n):
    return (1 - sqrt(2))**n

n = 100
for i in range(n+1):
    u = xn(i)
    b = x[i]
    w = u - b
    print  x%g General solution = %12g  |  Computed solution = %12g  |
        Avvik = %g  %(i, u, b, w)
```

# 2 Oppgave 2.

**a)**

```
n = eval(raw_input( n?  ))
i = eval(raw_input( i?  ))

s = 1

for j in range(1,(n+1-i)):
    s = s * (i+j)/j

print  %.14e  %s

"""
Terminal> python oblig1_3.py
n? 9998
i? 4
416083629102505

n? 100000
i? 70
8.14900007813826e+249

n? 1000
i? 500
2.70288240945437e+299

"""
```

Jeg programmer i python og da trenger jeg ikke bruke flyttall, dette er på grunn av at python setter opp presisjonen når tallene blir store. Her er ikke "integer overflow" så farlig, det som skjer er at kjøringen av programmet går saktere enn vanlig. Heltall aritmetikk er mer presis og svarerne man får er eksakte, og det er lett å se om noe går feil (resultatene er fullstendig feil!). Her er det greit å bruke heltall siden binomialkoeffisientene er heltall og divisjonen gir ingen rest. Hvis jeg hadde brukt et annet programerings språk kan det tenkes at jeg burde ha brukt flyttall og tilnærminger for å forhindre "overflow".
Svarene jeg fikk ser man under koden.

**b)**

Ja. Dette kan skje når enten i eller n blir for stor i forhold til hverandre. I mitt program i a) så blir det overflow når n blir mye større en i selv om binomialkoeffisienten er innen for det største flyttallet som kan representere på min maskin. Jeg får memory error når jeg prøver f.eks $n = 2^{63}$ og $i = 2$ eller $n = 2 * 10^9$ og i $= 2$

**c)**

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!} = \frac{n(n-1)(n-2)\cdot\ldots\cdot(n-i+1)(n-i)(n-i-1)(n-i-2)\cdot 3\cdot 2\cdot 1}{i!(n-i)(n-i-1)(n-i-2)\cdot\ldots\cdot 3\cdot 2\cdot 1} \tag{2}$$

Jeg forkorter og får:

$$\frac{n(n-1)(n-2)\cdot\ldots\cdot(n-i+1)}{i!} \tag{3}$$

Ved hjelp av produktnotasjon kan vi derfor skrive $\binom{n}{i}$ som

$$\binom{n}{i} = \prod_{j=1}^{i} \frac{n-i-j}{j} = \frac{n(n-1)(n-2)\cdot \ldots \cdot (n-i+1)}{i!} \tag{4}$$

Denne metoden er bedre å bruke når n er mye større enn i. F.eks $n = 2^{63}$ og $i = 2$ . Den forrige er best å bruke når i er mye større enn n.

# 3 Oppgave 3.

**a)**

```
from random import random #implementerer falsk/uekte-tilfeldige(pseudo-random) nummer

antfeil = 0; N = 10000
x0 = y0 = z0 = 0.0
feil1 = feil2 = 0.0

for i in range(N):
    x = random(); y = random(); #random() genererer et tilfeldig flyttall mellom [0.0,1.0)
    res1 = (x + y)*(x - y) #legger sammen og trekker fra de tilfeldige siffrene og multipliserer de
    res2 = x**2 - y**2 #kvadrerer begge de tilfeldige tallene ogsaa trekker i fra

    if res1 != res2: #hvis res1 ikke er det samme som res2
        antfeil += 1 #legger til 1 i antfeil
        x0 = x; y0 = y #x blir x0 og y blir y0
        feil1 = res1 #res1 blir feil1
        feil2 = res2 #res2 blir feil2

print (100. * antfeil/N) #100. multiplisert med antfeil/10000
print (x0, y0, feil1 - feil2) #skriver ut hva x og y var sist gang de var feil og differansen mellom res1 og res2
```

Programmet importer funksjonen random.
Den oppgir variabler (antfeil, x0, y0, z0, feil1, feil2) som er tomme, altså med verdi 0.0, og N til 10 000.
x = random() genererer et tilfeldig flyttall i intervallet [0.0, 1.0)
y = random() genererer et tilfeldig flyttall i intervallet [0.0, 1.0).
res1 adderer og subtraherer de tilfeldige flyttallene x og y, før den multipliserer resultatene.$(x + y) * (x - y)$
res2 kvadrerer de tilfeldige flyttallene x og y også subtraherer de.$x^2 - y^2$
Hvis res1 er forskjellig fra res2 legger programmet til 1 i antfeil. Tallet x blir satt til $x_0$ og tallet y blir satt til $y_0$. res1 blir satt til feil1 og res2 blir satt til feil2. Programmet skriver ut antall feil som har forekommet i prosent, skriver også ut x og y verdien på den siste feilen som ble gjort samt differansen mellom res1 og res2 på den siste feilen.
**b)**

En mulig forklaring kan være at det gjøres flere opperasjoner i det første programmet i forhold til det andre. Siden i første programmet skal programmet addere x og y, så subtrahere x og y før den multipliserer svarerene fra addisjonen og subtraksjonen.$(x + y) \times (x - y)$ og den skal sammenligne dette med $x^2 - y^2$ Altså 6 opperasjoner som gir 6 muligheter for feil.
I det andre programmet skal den dele x på y.$\frac{x}{y}$ Og sammenligner dette med 1 over y delt på x. $\frac{1.0}{\frac{y}{x}}$ Her er det altså 3 opperasjoner som medfører 3 muligheter for feil. Jeg tror dette kan være en grunn til at feilen blir mindre i program 2.
En annen forklaring er at når man adderer og subtraherer på en datamaskin kan det fort hende at noen tall forsvinner, spesielt hvis disse tallene er veldig like, denne effekten kalles "cancellation". Cancellation forekommer ikke når man skal dividere eller multiplisere, den verste feilen som kan skje når man gjør en av disse operasjonene er at det siste sifferet er feil. Siden både addisjon og subtraksjon forekommer i program 1 og kun divisjon forekommer i program 2 er nok dette mest sansynlig grunnen til at det er mindre feil i program 2.