MAT-INF1100 Oblig 1

Teodor Spæren, brukernavn teodors

September 16, 2015

Oppgave 1

I de oppgavene som krever at man gjør om et rasjonalt tall i intervallet (0, 1) om til en binærsifferutvikling, fant jeg denne sifferutviklingen ved å sette første siffer lik $(b * \beta)//c$ og så sette $b = (b * \beta)\%c$. Her er b/c lik det rasjonale tallet jeg omgjør og β er basen, med andre ord 2. Dette forsatte jeg med inntil b ble 0 eller fikk en tidligere verdi. Jeg har derfor valgt å ikke skrive opp alle omgjøringer av denne typen, da det ville ha brukt unødvendig mye plass.

a) For å gjøre om $\mathrm{da7}_16$ til en sifferutvikling i 2-tallsystemet, kan man bare omgjøre hvert siffer til den binære versjonen og så putte de sammen. Da får man:

$$\mathrm{da7}_{16} = 110110100111_2$$

b)

$$\frac{29}{64} = 0.011101_2$$

c)

$$\frac{1}{25} = 0.\overline{00001010001111010111}_2$$

d) Her brukte jeg samme regel som i 1.a. Gjør her bare om 4 til det tilsvarende binære tallet.

$$0.4_{16} = 0.01_2$$

e) Her gjør jeg om til 10-tallssystemet for så å bruke vanlig metode for å få den binære tallutviklingen.

$$\frac{\mathrm{b2c}_{16}}{1000_{16}} = \frac{2860}{4096} \Leftrightarrow \frac{715}{1024} = 0.1011001011_2$$

Oppgave 2

Skal vise at følgen $\{x_n\}$ gitt ved:

$$x_n = \left(\cos\left(x_{n-1}\right)\right)^2 \sin\left(x_{n-2}\right) \text{ for } n \geq 2$$

der $x_0 = \pi/2$ og $x_1 = 3$, ligger i de lukkede intervallet $[0, 1]$ for alle $n \geq 2$.

Jeg ønsker å bevise dette ved induksjon. Induksjonshypotesen er

$$P_n : 0 \leq x_n \leq 1 \text{ for } n \geq 2$$

og siden $n \geq 2$, så setter vi $n_0 = 2$. Det første vi må gjøre er å teste om P_{n_0} er sann

$$0 \leq x_2 \leq 1$$
$$0 \leq \left(\cos\left(x_{2-1}\right)\right)^2 \sin\left(x_{2-2}\right) \leq 1$$
$$0 \leq \left(\cos\left(3\right)\right)^2 \sin\left(\tfrac{\pi}{2}\right) \leq 1$$

og vi ser i fra dette at P_{n_0} sann. Da er det nok å vise at dersom P_m er sann for alle m slik at $2 \leq m < k$, så er P_k også sann. Dette viser vi slik:

$$0 \leq x_k \leq 1$$
$$0 \leq (\cos(x_{k-1}))^2 \sin(x_{k-2}) \leq 1$$

Siden vi antar at P_m er sann, ser vi at $0 \leq x_{k-1} \leq 1$ og $0 \leq x_{k-2} \leq 1$. Da er det lett å se at den største verdien P_k kan ha er 1 og den minste verdien P_k kan ha er 0.

■

Oppgave 3

a)

Programmet jeg skrev for å regne ut røttene til et andregrads polynom:

```
1  import math
2
3  a = float(raw_input("a: "))
4  b = float(raw_input("b: "))
5  c = float(raw_input("c: "))
6
7  p1, p2 = -b/(2*a), math.sqrt(b*b - 4*a*c)/(2*a)
8  x1, x2 = p1 - p2, p1 + p2
9
10 print("x1 = {:g}, x2 = {:g}".format(x1, x2))
```

For å sjekke om programmet fungerer, tester vi ut 3 eksempler. Venstre siden er program kjøringen og den høyre siden er manuell utregning.

```
a: 1
b: 2
c: 1
x1 = -1, x2 = -1
```

$$x = \frac{-2 \pm \sqrt{2^2 - 4 \times 1 \times 1}}{2 \times 1}$$

$$x = \frac{-2}{2}$$

$$x_1 = -1, \quad x_2 = -1$$

```
a: 1
b: 6
c: 9
x1 = -3, x2 = -3
```

$$x = \frac{-6 \pm \sqrt{6^2 - 4 \times 1 \times 6}}{2 \times 1}$$

$$x = \frac{-6}{2}$$

$$x_1 = -3, \quad x_2 = -3$$

```
a: 1
b: -1
c: -6
x1 = -2, x2 = 3
```

$$x = \frac{1 \pm \sqrt{(-1)^2 - 4 \times 1 \times -6}}{2 \times 1}$$

$$x = \frac{1 \pm 5}{2}$$

$$x_1 = -2, \quad x_2 = 3$$

b)

Bruker programmet i a) til å løse ligningen

$$10^{-8}x^2 + 10x + 10^{-8} = 0$$

Kjørelogg:

```
a: 1e-8
b: 10
c: 1e-8
x1 = -1e+09, x2 = 0
```

Den roten som får en feil er x2. Denne skulle vært -1×10^{-9}, noe som gir oss en stor relativ feil

$$\frac{\left|10^{-9} - 0\right|}{|10^{-9}|} = 1$$

Dette skjer når $b^2 \gg 4ac$ og det oppstår en underflow slik at $b = \sqrt{b^2 - 4ac}$. En måte å få mer nøyaktige resultater på er å regne ut den roten hvor avrundingsfeilen oppstår ved hjelp av den andre roten og det faktum at produktet mellom de to røttene alltid er

$$
\begin{aligned}
r_1 r_2 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \times \frac{-b - \sqrt{b^2 - 4ac}}{2a} \\
r_1 r_2 &= \frac{(-b)^2 - \left(\sqrt{b^2 - 4ac}\right)^2}{4a^2} \\
r_1 r_2 &= \frac{b^2 - b^2 + 4ac}{4a^2} \\
r_1 r_2 &= \frac{c}{a}
\end{aligned}
$$

Dette betyr at hvis vi kan kalkulere en av røttene med høy presisjon, så kan vi få den andre roten også, ved hjelp av dette forholdet. Programmet under implementerer denne alternative løsningsmetoden:

```
import math

a = float(raw_input("a: "))
b = float(raw_input("b: "))
c = float(raw_input("c: "))

# Compute the parts.
p1, p2 = -b/(2*a), math.sqrt(b**2 - 4*a*c)/(2*a)

# Compute both roots as normal.
x1, x2 = p1 - p2, p1 + p2

# If we have an underflow, compute one of the two roots by
# using the fact that the product of the two roots are c/a
if b**2 == (b**2-4*a*c):
        if abs(x1) > abs(x2):
                x2 = c/(x1*a)
        else:
                x1 = c/(x2*a)

print("x1 = {:g}, x2 = {:g}".format(x1, x2))
```

Nå hånterer programmet ligning gitt i starten av b) og andre lik den, helt fint:

```
a: 1e-8
b: 10
c: 1e-8
x1 = -1e+09, x2 = -1e-09
```

```
a: 1e-8
b: -10
c: 1e-8
x1 = 1e-09, x2 = 1e+09
```

Oppgave 4

a)

Programmet sjekker om matematiske identiter også holder for maskiner. Den kjører en sjekk 10000 ganger, og ser om

$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

også holder når man bruker 3 tilfeldigvalgte flyttall for x, y. For hvergang dette ikke holder blir verdien av x, y og z skrevet ned og de to forskjellige resultatene notert. Til slutt printes antall feil i prosent, fulgt av verdiene x, y og z hadde når den siste feilen oppstod. Hvor stor forskjellen var skrives også ut.

Derfor forteller utskriften oss at hele 50.64% av testene fant forskjellige resultater for $(x+y)^3$ og $x^3 + 3x^2y + 3xy^2 + y^3$. Videre kan vi lese at den siste

feilen var svært liten, $-4.4408920985 \times 10^{-16}$, men forsatt nok til at resultatene ikke var like. Det å bruke == for å teste om to flyttall er like, er en vanlig feil i programmering. Tester som den over og mange andre forteller oss at slike naive måter å sammenligne flyttall på kan ha katastrofale følger!

b)

Det modifiserte programmet:

```
from random import random

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

for i in range(N):
        x = random(); y = random(); z = random();
        res1 = (x + y) + z
        res2 = x + (y + z)

        if res1 != res2:
                antfeil += 1
                x0 = x; y0 = y; z0 = z
                feil1 = res1
                feil2 = res2

print (100. * antfeil/N)
print x0, y0, z0, feil1 - feil2
```

og her har vi kjøreloggen:

```
17.33
0.497037317989 0.36282462109 0.707965949223 -2.22044604925e-16
```

Vi ser helt klart at feil prosenten er langt lavere denne gangen. Jeg tror grunnen til dette er vi i dette programmet gjør færre operasjoner, noe som leder til mindre avrundingsfeil.