

MAT-INF 1100

Obligatorisk oppgave 1 av 2

Innleveringsfrist

Torsdag 29. september 2022, klokken 14:30 i Canvas (canvas.uio.no).

Instruksjoner

Du velger selv om du skriver besvarelsen for hånd og skanner den, eller om du skriver løsningen direkte inn på datamaskin (for eksempel ved bruk av \LaTeX). Skannede ark må være godt lesbare. Det forventes at man har en klar og ryddig besvarelse med tydelige begrunnelser. Besvarelsen skal leveres som én PDF-fil. Husk å inkludere eventuell kode og kjøreeksempel, samt relevante plott og figurer i PDF-filen.

Merk at man har ett forsøk på oppgaven. Samarbeid og alle slags hjelpemidler er tillatt, men den innleverte besvarelsen skal være skrevet av deg og reflektere din forståelse av stoffet. Er vi i tvil om du virkelig har forstått det du har levert inn, kan du bli bedt om en muntlig redegjørelse.

Søknad om utsettelse av innleveringsfrist

Hvis du blir syk eller av andre grunner trenger å søke om utsettelse av innleveringsfristen, må du ta kontakt med studieadministrasjonen ved Matematisk institutt (e-post: studieinfo@math.uio.no) før innleveringsfristen. Vitenskapelig ansatte kan ikke innvilge utsettelse.

For å få adgang til avsluttende eksamen i dette emnet, må man bestå alle obligatoriske oppgaver i ett og samme semester.

For å få godkjent denne første obligatoriske oppgaven må du ha gjort seriøse forsøk på å løse alle deloppgavene, og minst halvparten av oppgavene må være tilfredsstillende besvart.

For fullstendige retningslinjer for innlevering av obligatoriske oppgaver, se her:

www.uio.no/studier/admin/obligatoriske-aktiviteter/mn-math-oblig.html

LYKKE TIL!

Oppgaver

Oppgave 1. Vi skal se på differensligningen

$$x_{n+2} - 6x_{n+1} + 3x_n = 0, \quad \text{med } x_0 = 1 \text{ og } x_1 = 1. \quad (1)$$

- a) Skriv et program som regner ut og skriver ut x_2, x_3, \dots, x_{100} .
b) Vi endrer den ene startverdien x_1 til $x_1 = 3 - \sqrt{6}$. Skriv et nytt program som regner ut og skriver ut x_2, x_3, \dots, x_{100} .

Løsning: Følgende kode løser både a) og b).

```
1 import math
2
3 print("Exercise 1a) ")
4 xpp = 1; xp = 1;
5 for n in range(99):
6     x = 6*xp - 3*xpp
7     print("x[" ,n+2, "] = ", x)
8     xpp =xp; xp = x;
9
10 print("Exercise 1 b)")
11 xpp = 1; xp = 3 -math.sqrt(6);
12 for n in range(450):
13     x = 6*xp - 3*xpp
14     print("x[" ,n+2, "] = ", x)
15     xpp =xp; xp = x;
```

- c) Vis at den generelle løsningen av $x_{n+2} - 6x_{n+1} + 3x_n = 0$ er

$$x_n = C(3 - \sqrt{6})^n + D(3 + \sqrt{6})^n$$

og at initialverdiene $x_0 = 1$ og $x_1 = 3 - \sqrt{6}$ gir den spesielle løsningen $x_n = (3 - \sqrt{6})^n$.

Løsning: Den karakteristiske ligningen har røtter $3 \pm \sqrt{6}$, og den generelle løsningen følger fra dette. Initialverdiene gir ligningene

$$C + D = 1 \quad C(3 - \sqrt{6}) + D(3 + \sqrt{6}) = 3 - \sqrt{6},$$

som har løsning $C = 1, D = 0$. Det følger at den spesielle løsningen er $x_n = (3 - \sqrt{6})^n$.

- d) Prøv deg frem i programmet fra b) med å øke antall iterasjoner ytterligere (til mye mer enn 100). Når antall iterasjoner blir stort, stemmer det du fant i b) med den analytiske løsningen du fant i c)?

Løsning: På grunn av avrundingsfeil i den ene initialverdien regner ikke maskinen ut eksakt $x_n = (3 - \sqrt{6})^n$, men noe på formen

$$x_n = (1 - \epsilon_1)(3 - \sqrt{6})^n + \epsilon_2(3 + \sqrt{6})^n,$$

der ϵ_1 og ϵ_2 er små tall. Den numeriske løsningen stemmer til å begynne med bra med den analytiske og går mot 0, men etter hvert

som n blir stor vil $\epsilon_2(3 + \sqrt{6})^n$ dominere, og maskinen vil til slutt gi overflow, deretter en ny overflow med samme fortegn, og så NaN (maskinen regner ut noe på formen $\infty - \infty$). I koden over har vi valgt å kjøre 450 iterasjoner, slik at dette blir illustrert på de siste iterasjonene.

Foreslår at det gis full pott så lenge de påpeker at det blir store avvik fra den analytiske løsningen for store n - Det er først rett før obligfristen at vi går gjennom teorien for dette.

Oppgave 2. Binomialkoeffisienten $\binom{n}{i}$ er definert som

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \quad (2)$$

der $n \geq 0$ er et ikke-negativt heltall og i er et heltall i intervallet $0 \leq i \leq n$. Disse dukker opp i mange ulike sammenhenger og må ofte beregnes på datamaskin. Siden n og i er heltall så er det rimelig å bruke heltallige variable i slike beregninger. For små verdier av n og i går det bra, men for større verdier kan vi få problemer siden både teller og nevner i (2) kan bli større enn det største heltallet som kan representeres med 32- eller 64-bits heltall (selv om $\binom{n}{i}$ i seg selv ikke er så stor). I mange språk vil dette føre til «overflow», men selv i et språk som Python (som unngår dette ved hjelp av innebygget programvare), vil beregningene gå sakte, med mindre man forenkler formelen (2). Alternativt til heltall, kan vi bruke flyttall til å beregne (2), men da vil vi lett få «overflow» underveis i beregningene. I denne oppgaven skal vi se på disse to måtene å beregne $\binom{n}{i}$ på, og hvordan man kan unngå problemer med «overflow».

Ser vi nærmere på (2) finner vi at vi kan forkorte i stor skala,

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

Fra dette finner vi ved hjelp av produktnotasjon to ekvivalente formler for $\binom{n}{i}$. En hvor multiplikasjon gjøres separat i teller og nevner:

$$\binom{n}{i} = \frac{\prod_{k=1}^i (n-k+1)}{\prod_{j=1}^i j}. \quad (3)$$

Og en bestående av et produkt av rasjonale tall:

$$\binom{n}{i} = \prod_{j=1}^i \frac{n-j+1}{j}. \quad (4)$$

- a) Vi skal bruke heltallsvariable til å beregne $\binom{n}{i}$ i Python, og se at slike beregninger er pålitelige, men kan ta lang tid.

Skriv et program som beregner $\binom{n}{i}$ ved hjelp av (3), dvs ved å beregne teller og nevner hver for seg som produkt av heltall.

Test metoden på følgende tre binomialkoeffisienter (siste eksempel kan ta rundt 10 sekunder å beregne med denne framgangsmåten):

$$\left. \begin{aligned} \binom{5000}{4} &= 26010428123750, \\ \binom{1000}{500} &\approx 2.7028824094543655 \cdot 10^{299}, \\ \binom{100000}{99940} &\approx 1.1806919799625687 \cdot 10^{218} \end{aligned} \right\} \quad (5)$$

Hvilke resultater får du?

Løsning: De tre binomialkoeffisientene beregnes korrekt. Koden under kan brukes:

```
1 import math
2 import time
3
4 print("Exercise 2a) ")
5
6 def compute_binom_2a(n,i):
7     teller = 1;
8     nevner = 1;
9
10    for j in range(1,i+1):
11        teller *= (n-j+1); nevner *=j;
12    return teller/nevner
13
14 print("Binom(5000,4) = ", compute_binom_2a(5000,4))
15 print("Binom(1000,500) = ", compute_binom_2a(1000,500))
16 print("Binom(100000, 99940) = ",
17       compute_binom_2a(100000, 99940))
18 print("Binom(100000, 100000 - 999940) = ",
19       compute_binom_2a(100000, 60))
```

- b) Vi skal nå i stedet bruke flyttallsvariable til å beregne $\binom{n}{i}$, og se at dette kan være mer effektivt enn heltallsvariable, men at det kan lede til overflow ved beregning av store binomialkoeffisienter.

Skriv et program som beregner $\binom{n}{i}$ ved hjelp av formelen (4), dvs. som et produkt av flyttall. Test metoden på de tre binomialkoeffisientene i ligning (5).

Hvilke resultater får du? Merk at avrundingsfeil her kan lede til små avvik, og at «overflow» kan oppstå. Selv om $\binom{n}{i}$ er mindre enn det største flyttallet maskinen kan representere, kan du likevel få

«overflow»?

Løsning: De to første gir store tall, den tredje gir overflow. Koden under kan brukes

```
1 import math
2 import time
3
4 print("Exercise 2a) ")
5
6 def compute_binom_2a(n,i):
7     teller = 1;
8     nevner = 1;
9
10    for j in range(1,i+1):
11        teller *= (n-j+1); nevner *=j;
12    return teller/nevner
13
14 print("Binom(5000,4) = ", compute_binom_2a(5000,4))
15 print("Binom(1000,500) = ", compute_binom_2a(1000,500))
16 print("Binom(100000, 99940) = ",
17       compute_binom_2a(100000, 99940))
18 print("Binom(100000, 100000 - 99940) = ",
19       compute_binom_2a(100000, 60))
```

Hvis $i > n/2$ så vil faktorene i produktet der $j > n/2$ regne ut tall som er mindre enn 1. Det vil si at etter $n/2$ faktorer i produktet så har maskinen regnet ut et tall som er mye større enn svaret. Deretter blir disse nedskalert med tall som er mindre enn 1. Det kan derfor være slik at vi får overflow etter (eller før) $n/2$ iterasjoner, selv om svaret i seg selv ikke er for stort.

- c) Anta $i > n/2$. Kan du tenke ut en enkel modifisering av programmet fra b) som vil hjelpe oss med å unngå «overflow» i en del situasjoner?
Løsning: Hvis vi bruker at $\binom{n}{i} = \binom{n}{n-i}$ så unngår vi denne problematikken. Om man introduserer endringen i programmet i a) (se nederste linje i koden) reduseres beregningstiden for tredje eksempel drastisk. Og om man introduserer endringen i programmet i oppgave b) så unngår man overflow (se nederste linje i koden).

Oppgave 3. For tre vilkårlige reelle tall $x, y, z \in \mathbb{R}$ vet vi fra kalkulus at:

$$(x + y) + z = x + (y + z) \quad (\text{Assosiative loven for addisjon})$$

$$x(y+z) = xy+xz \quad (\text{Distributive loven for multiplikasjon mhp addisjon}).$$

I denne oppgaven skal vi se om disse disse to lovene også gjelder når vi bruker flyttall i Python. Følgende Python-program er gitt:

```
1 from random import random
2
3 antfeil = 0; N = 100000
```

```

4 x0 = 0; y0 = 0; z0 = 0; ikkelik1 = 0; ikkelik2 = 0;
5 for i in range(N):
6     x = random(); y = random(); z = random()
7     res1 = (x + y) * z
8     res2 = x*z + y*z
9     if res1 != res2:
10        antfeil += 1
11        x0 = x; y0 = y; z0 = z
12        ikkelik1 = res1
13        ikkelik2 = res2
14 print (100. * antfeil/N)
15 print (x0, y0, z0, ikkelik1 - ikkelik2)

```

Hvor “random()” genererer et tilfeldig tall i (0, 1). En kjøring ga som output

30.859

0.6087077776638925 0.9204274878392227 0.06851310883531125 -1.3877787807814457

- a) Forklar hva programmet gjør, og tolk output.

Løsning: Programmet regner ut i hvor mange prosent av tilfellene den distributive loven feiler når vi regner med flyttall. Programmet skriver også ut siste forekomst for x , y , z der loven feiler, og hvor stor feilen er.

- b) Forklar og kjør følgende program (koden kan du finne på kurssidene), og tolk output.

```

1 from random import random
2
3 antfeil = 0; N = 100000
4 x0 = 0; y0 = 0; z0 = 0; ikkelik1 = 0; ikkelik2 = 0;
5 for i in range(N):
6     x = random(); y = random(); z = random()
7     res1 = (x + y) + z
8     res2 = x + (y + z)
9     if res1 != res2:
10        antfeil += 1
11        x0 = x; y0 = y; z0 = z
12        ikkelik1 = res1
13        ikkelik2 = res2
14 print (100. * antfeil/N)
15 print (x0, y0, z0, ikkelik1 - ikkelik2)

```

Kan du eksperimentelt ved hjelp av en endring i programmet si i hvilken rekkefølge tre flyttall i Python blir addert (når du skriver $x + y + z$)? Er det $(x + y) + z$ eller er det $x + (y + z)$?

Løsning: Programmet regner ut i hvor mange prosent av tilfellene den assosiative loven feiler når vi regner med flyttall. Programmet skriver også ut siste forekomst for x , y , z der loven feiler, og hvor stor feilen er. Hvis du i koden først bytter ut linjen

`res1 = (x + y) + z`

med

```
res1 = x + y + z
```

finner du ut om Python legger sammen de to siste tallene først når du skriver $x + y + z$. På samme måte, bytter du ut linjen

```
res2 = x + (y + z)
```

med

```
res2 = x + y + z
```

finner du ut om Python legger sammen de to første tallene først. Gjør du dette vil du se at Python legger sammen de to første tallene først.