MA-178

Semesterprosjekt

Gruppe 56

Matte Prosjekt Høst 2020

Numerisk derivasjon og
Newton - Raphsons metode.

MERETHE FLÅT, TOBIAS BRAMBO, MAXIME R. CARA,
JOACHIM FREDHEIM, FRODE HITLAND, AHMAD H. ALKHALED
Vi som forfattere erklærer vårt selvstendige arbeid, og forstår konsekvensene av plagiat

Veileder: Michael R. Hansen

Universitetet i Agder, 2020
Fakultet for teknologi og realfag
Institutt for ingeniørvitenskap

Abstract

We were presented this project about four weeks ago. It included both differentiation and the use of Newton - Raphsons method. We made use of common pen and paper calculation and coding with the help of the programming language *python*.

When calculating the results in part one, numerical approximation to the derivative, we first of all found the derivative to *f(x)*, we then calculated an approximation to *f'(x)* by using a new function called *g(x)*. We also calculated an estimated error, by using a third function called *E(x)*. Both task 1d and task 2 of the first part of the project were done with the help of *python*. The last task of part one, we were asked to find with trial and error, the greatest Δx with to desimals, so that $E(x_0) \leq 0.001$. This specific task was done in *python*.

For the second part of this project we were asked to use the Newton - Rapshon method so that we can get approximate results for both solutions and the equation. We are also asked to use an estimated error for $E = 10^{-12}$ and to see how our choice of x_0 affected our results. For task two, part two of the project we were asked to calculate the degree θ_2, based on the knowledge of the degree θ_1. When we finished calculating the results to task 2a, 2b and 2c in part two, we were asked to explore the pyshical meaning of the results. For all the tasks on part two we used programs written in *python*

Innhold

Kapittel 1

Introduksjon

I dette matematikk prosjektet har vi de siste fire ukene jobbet med å forstå aspekter tilknyttet den deriverte av det vi kaller reelle funksjoner. En reell funksjon er en funksjon hvor både definisjonsmengden og verdimengden til funksjonen er reelle tall. Vi har også jobbet med å kunne forstå hvordan man skal kunne bruke Newton - Raphson metoden for å kunne finne tilnærminger til nullpunkter av ikke - lineære likninger. Vi har utforsket forskjellige måter å løse disse oppgavene på, og har endt opp med å bruke flere forskjellige midler til å løse oppgavene i dette prosjektet. Vi har løst noen oppgaver for hånd, mens andre oppgaver har vi brukt programmer som GeoGebra og også laget egne programmer i kodespråket Python til å finne løsninger til oppgavene. Vi har lært mye i løpet av dette prosjektet og vi skal i denne rapporten presentere teorien bak matematikken i prosjektet og presentere våre metoder og resultater.

Kapittel 2

Teori

2.1 Del 1: Numerisk tilnærming til den deriverte

I kapittel 2.2, definisjon nr. 4 i boken *Calculus*, blir den deriverte til en funksjon f definert som følgende:

$$f'(x) = \lim_{x \to h} \frac{f(x+h) - f(x)}{h}$$

For alle verdier av x som er innenfor grenseverdien.

Hvis f'(x) eksisterer, så sier vi da at f er deriverbar ved x. [2]
Hvis funksjonen $g(x) = c$ (konstant), da sier vi at den deriverte funksjonen til $g(x) = 0$.[2]

Når det er snakk om det vi kaller definisjonsområdet til en funksjon *f*, og definisjonsområdet til den deriverte av funksjonen *f'*. Så er det da snakk om alle *x*, hvor grafen til f har en tangent som ikke er vertikal.[2]

Når vi så skal finne en tilnærming til $f'(x_0)$ så benyttet vi oss av funkjsonen som vi kaller *g(x)*, likningen er som følgene:

$$f'(x) \approx g(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Ved å starte med en mindre Δx blir tilnærmingen bedre enn når vi bruker og sier $f'(x) \approx g(x)$. Når vi benytter $f'(x) \approx g(x)$ så oppstår det en liten feil, ettersom dette ikke er et eksakt svar. Denne feilen kan vi skrive som følgende:

$$E(x) = |f'(x) - \text{ tilnærmert verdi}| = |f'(x) - g(x)|$$

2.2 Del 2: Newton - Raphson

Newton-Raphsons metode lar oss ved hjelp av en rekke gjetninger finne en tilnærming til étt lokalt nullpunkt til en reell deriverbar funksjon.

Det vil si, en x_s som oppfyller

$$f(x_s) = 0$$

Ideelt sett vil vi bare velge en x_0 og gjette til vi finner x_s, dessverre er ikke dette alltid så enkelt. I stedet bruker vi tangentlinjen til f i punktet x_n og analyserer nullpunktet til denne.
Dersom tangentlinjens skjæringspunkt x_a tilfredsstiller kravet $|f(x_a)| < E$ hvor E er en forhåndsdefinert godkjent feilmargin har vi gjort en tilfredsstillende tilnærming for x_s

Kapittel 3

Metoder

3.1 Del 1

3.1.1 Oppgave 1

I oppgave 1 del 1 har vi regnet ut den deriverte av hver funksjon på papir ved å bruke kjente regler for derivasjon. vi har også regnet ut den deriverte av x_0, hvor verdien av x_0 var gitt i oppgavene. Tilnærmingen av den den deriverte av x er $g(x)$, den finner vi ut ved å bruke likningen

$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x} = g(x)$$

Feilen mellom den eksakt deriverte $f'(x)$ og den tilnærmete deriverte $g(x)$, er $E(x)$. Feilen $E(x)$ uttrykkes med kommende ligning:

$$E(x) = |f'(x) - g(x)|$$

I vårt tilfelle bruker vi $x_0 = x$. Vi har byttet verdien på x med verdiene som vi ble gitt i oppgavene for hver funksjon.

Underoppgave 1 D) løste vi som en del av python programmet som er mer detaljert beskrevet i 3.1.2. Kode vedlagt i A.1.
En del av dette programmet var dedikert til å finne en Δx som tilfredsstilte kravet:

$$\mathbf{0.001 \geq E(x) = \frac{F(x+\Delta x) - F(x)}{\Delta x}}$$

Dette ble gjort med en løkke som sjekket Δx verdier fra 0 og oppover med ekstremt små inkrementer. Hver av de fire versjonene av programmet var her finjustert til å sjekke et mer realistisk intervall for raskere utregning.

3.1.2 Oppgave 2

Oppgave 2 har blitt løst rent digitalt ved hjelp av fire programmer skrevet i Python med bibliotekene matplotlib og numpy.
Matplotlib er et flott bibliotek for grafing/plotting av funksjoner, med mange muligheter for farger, linjetyper, linje/punktkart osv.
Numpy er et bibliotek fullt av matematiske og vitenskapelige funksjoner som gjør programmatisk utregning av avansert matematikk mye enklere
I alt skrev vi fire programmer til denne oppgaven, ett til hver av funksjonene. Generelt sett er programmene nesten helt like, de eneste variasjonene mellom dem er definisjoner av selve funksjonene, samt argumentene gitt av oppgaven for intervall og x_0, i tillegg til det nevnt i 3.1.1.

Python, matplotlib og numpy er alle svært dynamiske verktøy som har gjort denne oppgaven ganske enkel å programmere.
Her definerte vi bare funksjonene F(x), f(x) og g(x), lagde en liste på 1000 tall i det gitte intervallet $min \leq x \leq max$ med fast intervall $\frac{max-min}{1000}$, videre lagde vi tre variabler, Fy, fy og gy, til å holde y-koordinatene tilhørende listen x.
Python lar oss definere disse listene veldig enkelt med Fy = F(x), hvor dersom x er en liste med 1000 verdier, vil også Fy bli en liste med 1000 verdier. Deretter mates bare listene x og y til matplotlib som plotter og tegner grafene for oss i ønsket stil. 2D ble løst med en egen løkke som itererte over samme liste x, og tilordnet hver utregnede $E(x_n)$, $n\epsilon\{0, 999\}$, til sin egen index i en liste E.
Kode tilhørende oppgaven finnes som vedlegg: A.1

3.2 Del 2

3.2.1 Oppgave 1

I oppgave 1 på del 2 har vi skrevet en kode som gjør Newton-Raphson metoden for oss, slik at alt vi trengte å gjøre var å sette inn en ønsket startverdi. Da vil programmet konvergere mot det nærmeste nullpunktet og stoppe når den kom innenfor den maximale feilen $E = 10^{-12}$. Koden stopper om startverdien vi setter inn gjør slik at den deriverte blir lik 0, fordi man kan ikke dele på 0, og den stopper også om den ikke når en nøyaktig nok verdi før max antall iterasjoner er nådd.

For at koden skal fungere må vi sette inn $f(x)$ og $f'(x)$ i koden, og så vil koden gjøre hele utregningen frem til den treffer ett nullpunkt som tilfredsstiller feilen E. I denne oppgaven får vi $f(x)$ og kan da regne ut $f'(x)$ ved å bruke kjente regler for derivasjon.

Koden til oppgave 1 finner du i Tillegg A.2.1

3.2.2 Oppgave 2

I oppgave 2 på del 2 bruker vi samme koden som vi brukte på oppgave 1, men vi får ikke en funksjon. Vi fikk informasjon om de forskjellige leddene som vi kunne bruke til å utlede likningen $f(\theta_2)$. Når vi utledet en likning og satte inn θ_1 fikk vi uttrykk som vi kunne derivere, og da kunne vi bare sette inn likningen og den deriverte i den samme koden vi brukte på oppgave 1. Vi brukte programvaren GeoGebra til å utlede likningene og de deriverte til likningene i denne oppgaven. Måten vi utledet likningene på var å sette inn den informasjonen vi fikk i skriv innfeltet i GeoGebra som en enhet, og da fikk vi fine uttrykk med én ukjent, som vi lett kunne derivere i samme programmet ved å skrive inn f' i skriv innfeltet.

På oppgave 2 var vi nødt til å ha 3 forskjellige koder med de ulike likningene i, fordi θ_1 er ikke lik i alle deloppgavene, og vi får da ulike likninger og ulike deriverte.

Kodene til oppgave 2 finner du i Tillegg A.2.2 - A.2.4

Kapittel 4

Resultater

4.1 Del 1

4.1.1 Oppgave 1

#	Funksjonsuttrykk $f(x)$	Intervall	x_0
1	$f(x) = 7x^2 - 8x + 1$	$0 \leq x \leq 2$	$x_0 = 1$
2	$f(x) = sin(x)$	$0 \leq x \leq 2\pi$	$x_0 = \pi/4$
3	$f(x) = \frac{1-x}{(x+3)^2}$	$-1 \leq x \leq 2$	$x_0 = 1$
4	$f(x) = \sqrt{1+x^2}$	$0 \leq x \leq 10$	$x_0 = 5$

A)

Finn den deriverte $f'(x)$ (ikke en tilnærming) ved å bruke kjente regler for derivasjon.

1. $f'(x) = 14x - 8$
2. $f'(x) = cos(x)$
3. $f'(x) = \frac{x-5}{(x+3)^3}$
4. $f'(x) = \frac{x}{\sqrt{(1+x)^2}}$

B)

Regn ut $f'(x_0)$, samt en tilnærming ved å gjøre bruk av $g(x)$ fra (2) med $\Delta x = 0.1$.

1	$f'(x_0) = 14(1) - 8 = 6$	$g(1) = \frac{(7*1.1^2-8*1.1+1)-(7*1-8*1+1)}{0.1} \approx 6.7$
2	$f'(x_0) = cos(\frac{\pi}{4}) = \frac{\sqrt{2}}{2} \approx 0.71$	$g(\frac{\pi}{4}) = \frac{sin(0.885)-sin(\frac{\pi}{4})}{0.1} \approx 0.67$
3	$f'(x_0) = \frac{1-5}{(1+3)^2} = -\frac{1}{16}$	$g(1) = \frac{\frac{1-1.1}{(1.1+3)^2}-\frac{1-1}{(1+3)^2}}{0.1} \approx -0.059$
4	$f'(x_0) = \frac{5}{\sqrt{1+5^2}} = \frac{5\sqrt{26}}{26}$	$g(5) = \frac{\sqrt{1+5.1^2}-\sqrt{1+5^2}}{0.1} \approx 0.981$

C)

Beregn E(x_0).

1. $E(x_0) = |6 - 6.7| = 0.7$
2. $E(x_0) = |0.7071 - 0.67| = 0.0371$
3. $E(x_0 = |0.062 - 0.059| = 0.003$
4. $E(x_0 = |0.9805 - 0.981| = 0.005$

D)

Finn, ved prøving og feiling, den største Δx avrundet til to signifikante desimaler, slik at $E(x_0) \leq 0.001$.
Δx verdiene her ble funnet ved hjelp av et python program. Utklipp mellom hvert liste-punkt for hver funksjon. Se vedlegg for kode. Vedlegg: A.1.1. Web: [1]

1. $E \leq 0.001$ ved $\Delta x \leq 0.00014$

```
E:0.0009940000064636578, x:0.00014199999999999998
```

2. $E \leq 0.001$ ved $\Delta x \leq 0.0028$

```
E:0.0009997281895267607, x:0.002825
```

3. $E \leq 0.001$ ved $\Delta x \leq 0.032$

```
E:0.0009999941212291974, x:0.03238900000000039
```

4. $E \leq 0.001$ ved $\Delta x \leq 0.27$

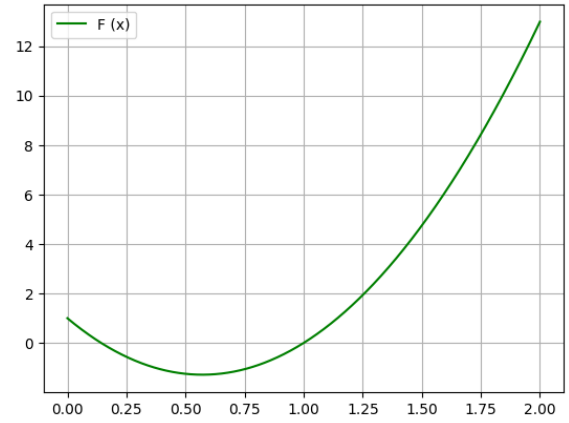
```
E:0.0009999996683186518, x:0.27940299999998924
```

4.1.2 Oppgave 2

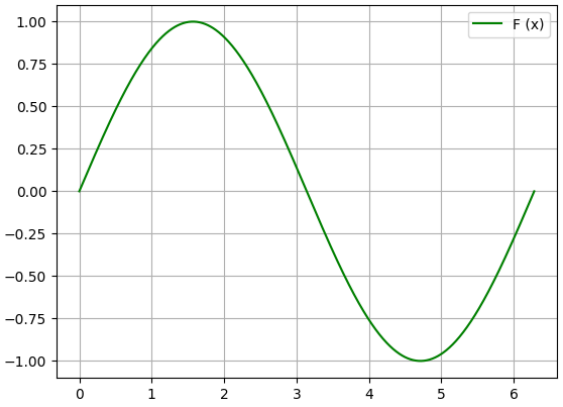
Plott grafen til følgende funksjoner i intervallet fra tabellen over:
Notat: Plotting er gjort med python bibloteket matplotlib og den tilhørende pyplot klassen. Plotting er gjort ved å lage en liste med 1000 x-verdier i intervallet for så å regne $f(x), f'(x), g(x)$ eller $E(x)$ på dem alle. Pyplot lar oss tegne disse punktene som en rekke punkter med linjesegmenter mellom seg, eller som en rekke punkter. Vi har valgt å tegne grafene som en sum av linjesegmenter av visuelle grunner.

A)

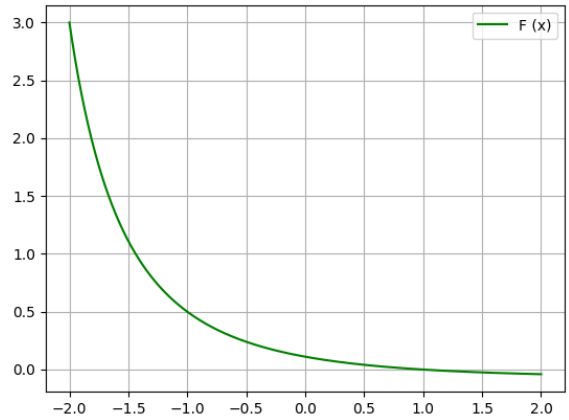
$f(x)$.



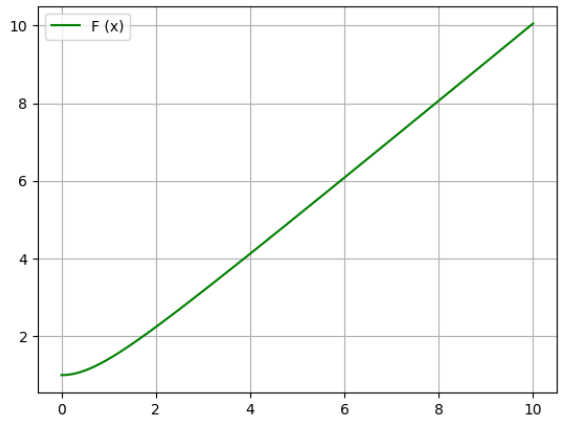
$f(x) = 7x^2 - 8x + 1$



$f(x) = sin(x)$



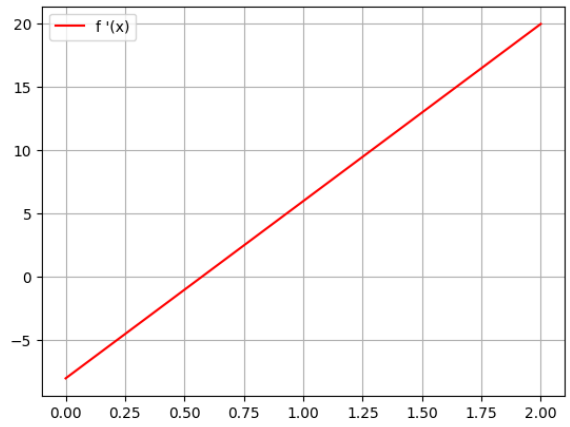
$f(x) = \frac{1-x}{(x+3)^2}$



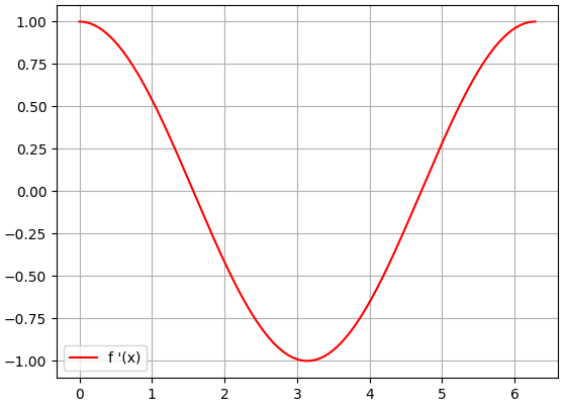
$f(x) = \sqrt{1+x^2}$

B)

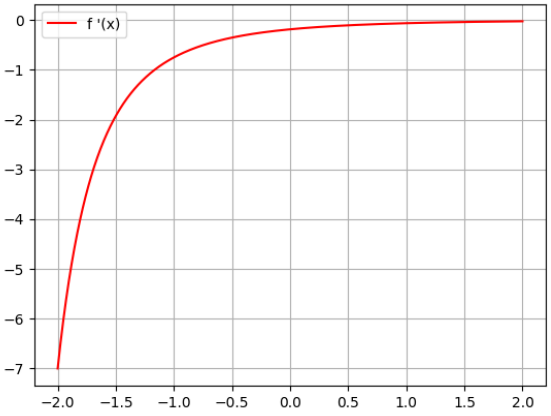
$f'(x)$



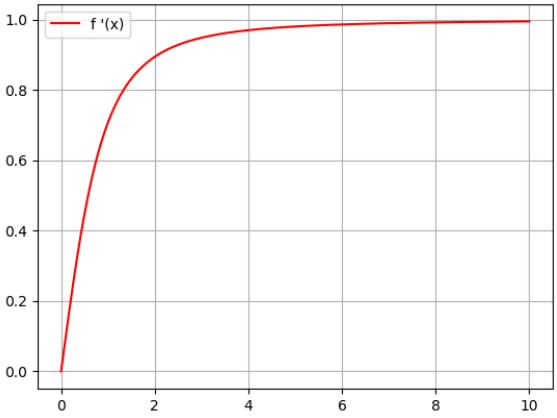
$f'(x) = 14x - 8$



$f'(x) = cos(x)$



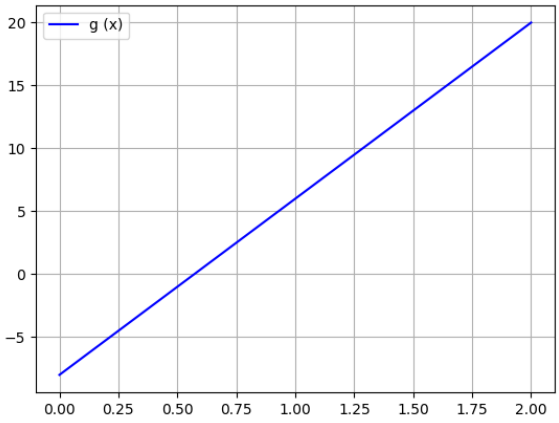
$f'(x) = \frac{x-5}{(x+3)^3}$



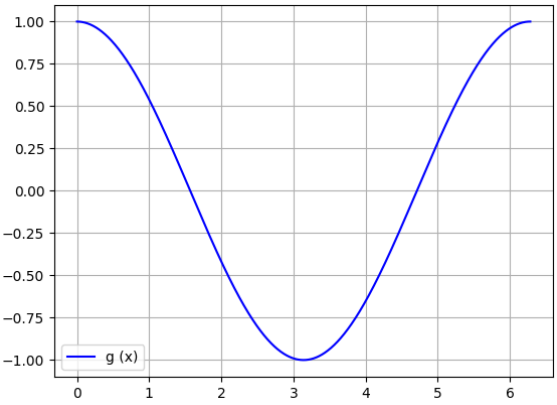
$f'(x) = \frac{x}{\sqrt{(1+x)^2}}$

C)

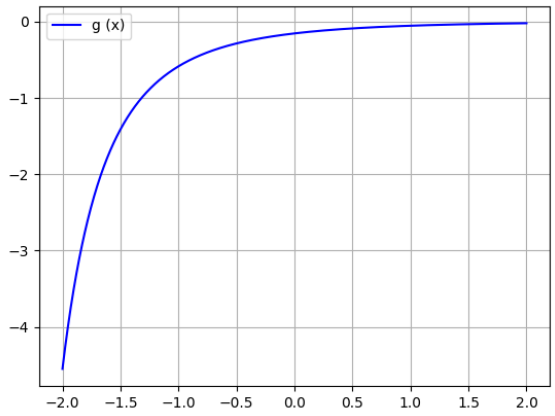
Tilnærmingen $g(x)$ med Δx verdien fra Oppgave 1 d.



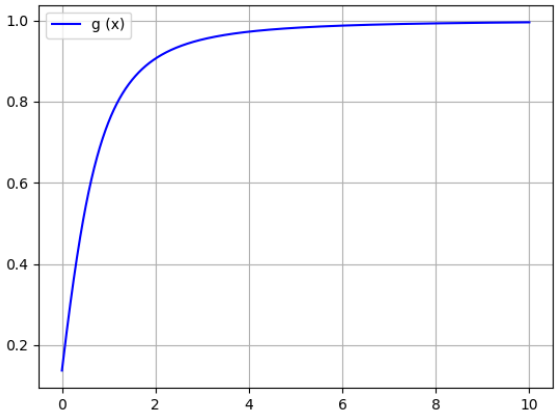
$$g(x) = \frac{F(x+0.00014)-F(x)}{0.00014}$$



$$g(x) = \frac{F(x+0.0028)-F(x)}{0.0028}$$



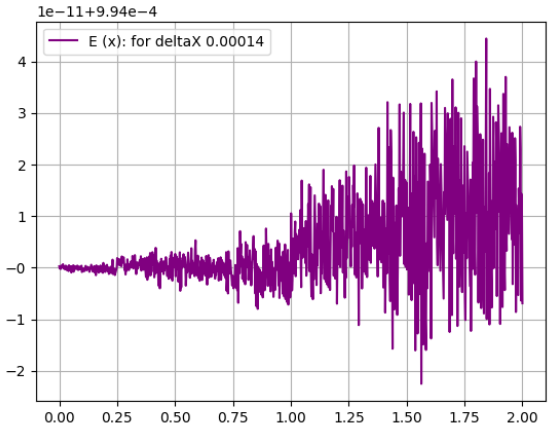
$$g(x) = \frac{F(x+0.032)-F(x)}{0.032}$$



$$g(x) = \frac{F(x+0.27)-F(x)}{0.27}$$

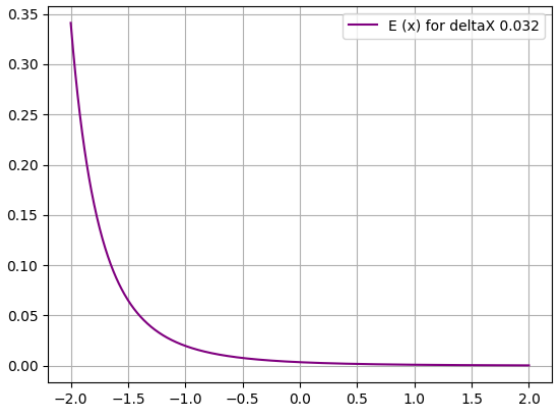
D)

Feilen $E(x) = |f'(x) - g(x)|$
MERK: I figuren for Funksjon 1 skal feilen være konstant. Variasjonen vi ser i grafen her er grunnet tallstøy/avrundingsproblemer og unøyaktigheter hos datamaskinen ved utregning av tall med mange desimaler.

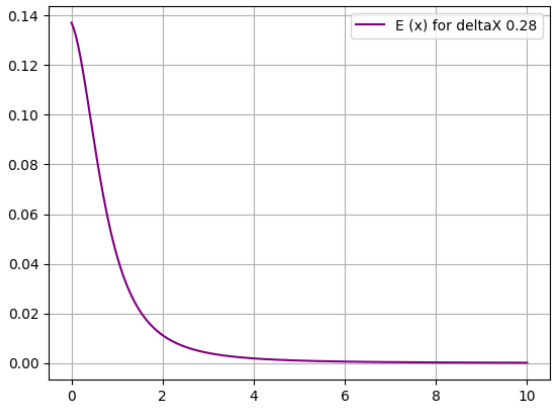


$$E(x) = |f'(x) - \frac{F(x+0.00014) - F(x)}{0.00014}|$$

$$E(x) = |f'(x) - \frac{F(x+0.0028) - F(x)}{0.0028}|$$



$$E(x) = |f'(x) - \frac{F(x+0.032) - F(x)}{0.032}|$$



$$E(x) = |f'(x) - \frac{F(x+0.27) - F(x)}{0.27}|$$

4.2 Del 2

4.2.1 Oppgave 1

Etter vi har satt inn $f(x)$ og $f'(x)$ ser vi at programmet konvergerer mot disse nullpunktene:

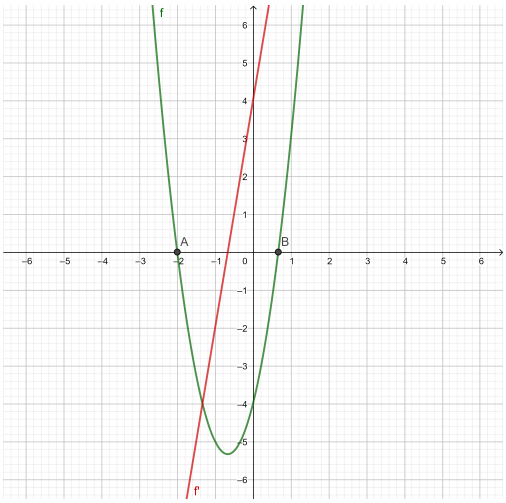
```
Newton Raphson metoden:
Iterasjon-1, x1 = 0.700000000000 og f(x1) = 0.270000000000
Iterasjon-2, x1 = 0.667073170732 og f(x1) = 0.003252528257
Iterasjon-3, x1 = 0.666666728615 og f(x1) = 0.000000495586
Iterasjon-4, x1 = 0.666666666667 og f(x1) = 0.000000000000

Det ene nullpunktet er i: 0.666666666667
```

Figur 4.1: Startverdi: 1

```
Newton Raphson metoden:
Iterasjon-1, x1 = -3.500000000000 og f(x1) = 18.750000000000
Iterasjon-2, x1 = -2.397058823529 og f(x1) = 3.649437716263
Iterasjon-3, x1 = -2.045554907515 og f(x1) = 0.370665008920
Iterasjon-4, x1 = -2.000752508266 og f(x1) = 0.006021764935
Iterasjon-5, x1 = -2.000000212231 og f(x1) = 0.000001697848
Iterasjon-6, x1 = -2.000000000000 og f(x1) = 0.000000000000

Det ene nullpunktet er i: -2.000000000000
```

Figur 4.2: Startverdi: -1

I denne oppgaven er det bare 2 nullpunkter som du kan se på grafen av likningen under:

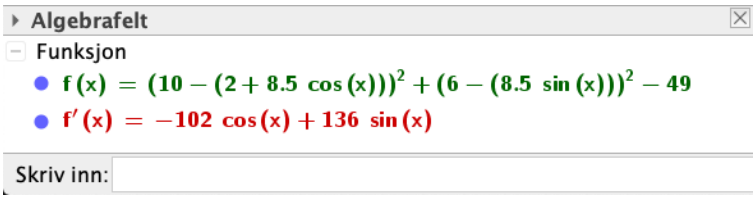


Figur 4.3: Graf til oppgave 1, del 2

4.2.2 Oppgave 2

a)

Utledningen av likningen i denne deloppgaven:



Figur 4.4: Utledningen gjort i GeoGebra

Etter at vi har satt inn $f(\theta_2)$ og $f'(\theta_2)$ ser vi at programmet konvergerer mot disse nullpunktene:

```
Newton Raphson metoden:
Iterasjon-1, x1 = -0.125000000000 og f(x1) = 1.027940062104
Iterasjon-2, x1 = -0.116300434092 og f(x1) = 0.004637969237
Iterasjon-3, x1 = -0.116260824540 og f(x1) = 0.000000096682
Iterasjon-4, x1 = -0.116260823714 og f(x1) = -0.000000000000

Det ene nullpunktet er i: -0.116260823714
```

Figur 4.5: Startverdi: 0

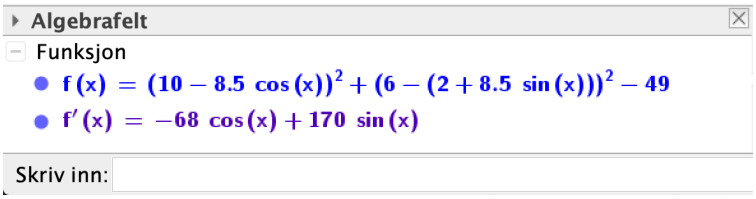
```
Newton Raphson metoden:
Iterasjon-1, x1 = 1.607814409445 og f(x1) = 26.353188780616
Iterasjon-2, x1 = 1.419148549321 og f(x1) = 1.875463832707
Iterasjon-3, x1 = 1.403392366282 og f(x1) = 0.015143344508
Iterasjon-4, x1 = 1.403263050101 og f(x1) = 0.000001030450
Iterasjon-5, x1 = 1.403263041301 og f(x1) = 0.000000000000

Det ene nullpunktet er i: 1.403263041301
```

Figur 4.6: Startverdi: 1

Siden denne likningen er trigonometrisk gjelder det også for alle nullpunktene $+2k\pi$ der 'k' er større enn null og en integer. Altså for hver gang leddet i θ_1 gjør en full rotasjon, kommer θ_2 tilbake til samme vinkel.

b)

Utledningen av likningen i denne deloppgaven:



Figur 4.7: Utledningen gjort i GeoGebra

Etter at vi har satt inn $f(\theta_2)$ og $f'(\theta_2)$ ser vi at programmet konvergerer mot disse nullpunktene:

```
Newton Raphson metoden:
Iterasjon-1, x1 = -0.452205882353 og f(x1) = 16.050128284672
Iterasjon-2, x1 = -0.333708084392 og f(x1) = 0.901493446353
Iterasjon-3, x1 = -0.326191381136 og f(x1) = 0.003916873336
Iterasjon-4, x1 = -0.326158435426 og f(x1) = 0.000000075571
Iterasjon-5, x1 = -0.326158434791 og f(x1) = 0.000000000000

Det ene nullpunktet er i: -0.326158434791
```

Figur 4.8: Startverdi: 0

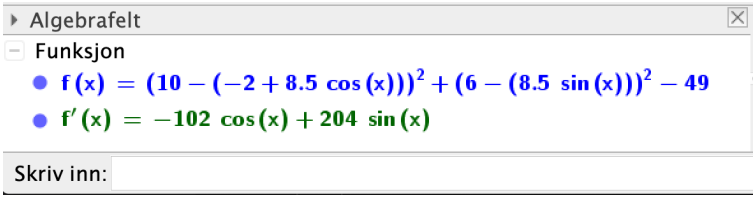
```
Newton Raphson metoden:
Iterasjon-1, x1 = 1.092385139461 og f(x1) = 0.621745949768
Iterasjon-2, x1 = 1.087186966693 og f(x1) = 0.001875732847
Iterasjon-3, x1 = 1.087171189161 og f(x1) = 0.000000017332
Iterasjon-4, x1 = 1.087171189015 og f(x1) = 0.000000000000

Det ene nullpunktet er i: 1.087171189015
```

Figur 4.9: Startverdi: 1

Siden denne likningen er trigonometrisk gjelder det også for alle nullpunktene $+2k\pi$ der 'k' er større enn null og en integer. Altså for hver gang leddet i θ_1 gjør en full rotasjon, kommer θ_2 tilbake til samme vinkel.

c)

Utledningen av likningen i denne deloppgaven:



Figur 4.10: Utledningen gjort i GeoGebra

Etter at vi har satt inn $f(\theta_2)$ og $f'(\theta_2)$ ser vi at programmet konvergerer mot disse nullpunktene:

```
Newton Raphson metoden:
Iterasjon-1, x1 = -0.007352941176 og f(x1) = 0.005507922836
Iterasjon-2, x1 = -0.007299723106 og f(x1) = 0.000000287813
Iterasjon-3, x1 = -0.007299720325 og f(x1) = -0.000000000000

Det ene nullpunktet er i: -0.007299720325
```

Figur 4.11: Startverdi: 0

```
Newton Raphson metoden:
Iterasjon-1, x1 = 0.938238217595 og f(x1) = 0.378377982223
Iterasjon-2, x1 = 0.934607864490 og f(x1) = 0.001337699254
Iterasjon-3, x1 = 0.934594938490 og f(x1) = 0.000000016980
Iterasjon-4, x1 = 0.934594938326 og f(x1) = 0.000000000000

Det ene nullpunktet er i: 0.934594938326
```

Figur 4.12: Startverdi: 1

Siden denne likningen er trigonometrisk gjelder det også for alle nullpunktene $+2k\pi$ der 'k' er større enn null og en integer. Altså for hver gang leddet i θ_1 gjør en full rotasjon, kommer θ_2 tilbake til samme vinkel.

Kapittel 5

Diskusjon

•Sammenhengen mellom $f(x)$ og $f'(x)$. F.eks. hvordan $f'(x)$ varierer i forhold til $f(x)$.

$f'(x)$ viser stigningen i $f(x)$ og hvordan den endrer seg, med tanke på x. Vi bruker det hvis vi vil komme frem til hvor mye en funksjon stiger i ett visst punkt.

•Hvordan valg av Δx påvirker nøyaktigheten av tilnærmingene. F.eks. hvor liten kan vi velge Δx ?

Valget av Δx påvirker nøyaktigheten av tilnærmingene vi gjør. Fordi jo lavere Δx vi velger, jo nærmere den faktiske x-verdien er vi. Men når vi velger Δx kommer vi til ett punkt hvor det ikke gir mening og velge en lavere verdi. Hvis vi bare vil vise 2 desimaler i svaret gir det ikke mening og velge en $\Delta x > 0.01$.

•Hvordan feilen $E(x)$ varierer som en funksjon av x.

Feilen $E(x)$ vil bli større jo større verdi for x vi putter inn. Ettersom tilnærmingenene og nøyaktigheten vil bli mindre desto mindre verdi for x man velger for Δx. Hvis funksjonen $f(x)$ ikke er lineær, vil man se at desto lengre vekk man kommer fra det punktet man deriverer i desto større blir feilen.
I flere av funksjonene ser vi også en sammenheng mellom $E(x)$ og den annenderiverte $f''(x)$ til funksjonen. De tydeligste eksemplene på dette er Funksjon 1, hvor den annenderiverte er en konstant, og funksjon 2, hvor E, når amplituden blåses opp tilstrekkelig, er en ganske tro gjenskapning av |-sin(x)|, dvs absoluttverdien til den annenderiverte av sin(x).
•Begrensninger ved numeriske tilnærminger av den deriverte.

Begrensningene med den numeriske tilnærmingen av den deriverte er det at vi ikke får en nøyaktig derivert. Altså den vil alltid ha en viss feilmargin. Eneste måten å unngå dette er ved å bruke den eksakte deriverte.

•Utforsk den fysiske betydningen av de to løsningene i hvert tilfelle.

Når vi finner nullpunktene til θ_2 får vi 2 verdier, her er den ene verdien for når armen til derricken er bøyd opp, og den andre verdien er når d en er bøyd ned, og den vil aldri bytte mellom disse tilstandene.

•Finn sammenhengen mellom initialverdien brukt i løsningsmetoden, og hvilken av de to løsningene metoden konvergerer mot.

Sammenhengen mellom initialverdien og nullpunktet den konvergerer mot er at hvis du velger initialverdi hvor stigningstallet til funksjonen er negativt konvergerer du mot det de nullpunktet der derricken er bøyd nedover. Velger du initialverdi hvor stigningstallet til funksjonen er positivt konvergerer du mot de nullpunktene hvor derricken er bøyd oppover. Dette gjelder derimot ikke hvis vi velger verdier nærme topp- og bunnpunkter, fordi da er stigningstallet 0.

Kapittel 6

Konklusjon

I dette prosjektet har vi gått igjennom og lært om de matematiske metodene Numerisk tilnærming av den deriverte og Newton-Raphson metoden. Prosjektet har hatt fokus på å utfordre oss innenfor disse områdene og vi har lært ulike konsepter og metoder vi kan bruke til å løse slike problemer på. Vi har blitt bedre kjent med reglene for disse regnemetodene og generelt fått en god forståelse for de forskjellige metodene vi bruker.

Etter å ha jobbert med det prosjektet, lærte vi kjente derivasjon regler og bruk av dem for å komme frem til løsningen. I tillegg har vi lært og brukt Limit til å finne en tilnærming til den deriverte av en gitt funksjon når differansen på x-verdiene går mot null.

Gjennom Newton-Raphson metoden lærte vi hvordan vi kunne bruke en funksjon og dens deriverte til å finne en god tilnærming til nullpunktene til funksjonen, så lenge gjennomsnittsveksten ikke er lik null. Andre ting vi oppdaget igjennom oppgaven var at Newton-Raphson metoden ikke alltid fungerer. Det er visse punkter den ikke fungerer i, som for eksempel når $f'(x) = 0$.

Alt i alt har dette prosjektet vært en bra lære opplevelse for vår gruppe, og vi har fått stort utbytte av det. Det har vært generelt gode og overkommelige oppgaver som fortsatt ga oss en viss utfordring.

Bibliografi

[1] Gruppe 56. *Matteprosjekt Matematikk 1 UIA H2020*. URL: https://github.com/jofredh/mat1proj.

[2] Christopher Essex Robert A. Adams. *Calculus A Complete Course Ninth edition*. Pearson, 2018. ISBN: 9780134154367.

Tillegg A

Kode

Kodekommentarer kommer ikke frem like tydelig som ønskelig her, den wrappes også videre over til ny side ved flere steder, noe som ødelegger kontinuitet i lesningen. Dette er beklagelig.
Ved ønske om nærmere gransking av koden i mer kode-vennlig medium, se Github.[1]

A.1 Del 1

Koden for Oppgave 1D og Oppgave 2 er gjort i samme fil, separert for hver av de fire funksjonene.

A.1.1 Funksjon 1

```
import numpy as np
import matplotlib.pyplot as plt

x0 = 1
min = 0
max = 2

def f(x):
    return (14*x) - 8

def g(dx,x=x0):
    return np.longdouble((F(x+dx)-F(x))/dx)

def G(x): #To funksjoner for tilnærmingen g fordi dette gjør det enklere å ...
   tegne grafen separert fra f'
    return np.longdouble((F(x+0.00014)-F(x))/0.00014) #DeltaX hardkodet ...
       inn etter funn i Del1:Oppgave1D

def F(x):
    return 7*x**2 -8*x +1


for x in np.arange(0,0.001,0.000001): #Loop for å finne største gyldige ...
   DeltaX for E<=0.001. Loop grenser og stepsize funnet ved prøv&feil
    Ex = abs(f(x0)-g(dx=x))
    if(Ex<=0.001):
        valid=x
        #print("F:{0}, G:{1}, f:{2}, E:{3}, ...
           x:{4}".format(F(x0+x),g(x),f(x0),Ex,x))#Fjern '#' tegnet foran ...
           denne linjen for å kjøre koden med printing av E og deltaX
    elif(Ex>0.001):
```

```
        break

i=0
E = [0]*1000
for x in np.linspace(min,max,1000): #Loop for å beregne variasjon i E over ...
    et gitt intervall min -> max
    E[i] = abs(f(x)-g(x=x,dx=valid))
    i+=1


fig = plt.gcf()
fig.canvas.set_window_title('Funksjon 1: 7 x   - 8x + 1')
x=np.linspace(min,max,1000)
y=F(x)
yd=f(x)
yg=G(x)
ye=E
plt.grid(True)
plt.plot(x,yd,'r-', label='f \'(x)')
plt.plot(x,y,'g-',label="F (x)")
plt.plot(x,yg,'b-',label="g (x)")
plt.plot(x,ye,color='purple',label='E (x): for deltaX %.5f'%(valid))
plt.legend()
plt.show()
```

A.1.2 Funksjon 2

```
import numpy as np
import matplotlib.pyplot as plt

x0 = np.pi/4
min = 0
max = 2*np.pi

def f(x):
    return np.cos(x)

def g(dx,x=x0):
    return np.longdouble((F(x+dx)-F(x))/dx)

def G(x):  #To funksjoner for tilnærmingen g fordi dette gjør det enklere å...
    tegne grafen separert fra f'
    return np.longdouble((F(x+0.002825)-F(x))/0.002825)#DeltaX hardkodet ...
       inn etter funn i Del1:Oppgave1D

def F(x):
    return np.sin(x)

for x in np.arange(0,0.01,0.000001):#Loop for å finne største gyldige ...
   DeltaX for E<=0.001. Loop grenser og stepsize funnet ved prøv&feil
    Ex = abs(f(x0)-g(x))
    if(Ex<=0.001):
        valid=x
        #print("F:{0}, G:{1}, f:{2}, E:{3}, ...
           x:{4}".format(F(x0+x),g(x),f(x0),Ex,x)) #Fjern '#' tegnet ...
           foran denne linjen for å kjøre koden med printing av E og deltaX
    elif(Ex>0.001):
        break
```

```
i=0
E = [0]*1000
for x in np.linspace(min,max,1000): #Loop for å beregne variasjon i E over ...
    et gitt intervall min -> max
     E[i] = np.longdouble(abs(f(x)-g(x=x,dx=valid)))
     i+=1

fig = plt.gcf()
fig.canvas.set_window_title('Funksjon 2: sin(x)')
x=np.array(np.linspace(min,max,1000))
y=F(x)
yd=f(x)
yg=G(x)
ye=E
plt.grid(True)
plt.plot(x,yd,'r-', label='f \'(x)')
plt.plot(x,y,'g-',label="F (x)")
plt.plot(x,yg,'b-',label="g (x)")
plt.plot(x,ye,color='purple',label='E (x) for deltaX %.4f'%(valid))
plt.legend()
plt.show()
```

A.1.3 Funksjon 3

```
import numpy as np
import matplotlib.pyplot as plt

x0 = 1
min = -2
max = 2

def f(x):
    return (x-5)/(x+3)**3

def g(dx,x=x0):
    return np.longdouble((F(x+dx)-F(x))/dx)

def G(x):  #To funksjoner for tilnærmingen g fordi dette gjør det enklere å...
    tegne grafen separert fra f'
    return np.longdouble((F(x+0.32389)-F(x))/0.32389)#DeltaX hardkodet inn ...
       etter funn i Del1:Oppgave1D

def F(x):
    return (1-x)/((x+3)**2)


for x in np.arange(0.032,0.1,0.000001):#Loop for å finne største gyldige ...
   DeltaX for E<=0.001. Loop grenser og stepsize funnet ved prøv&feil
    Ex = abs(f(x0)-g(x))
    if(Ex<=0.001):
        valid=x
        #print("F:{0}, G:{1}, f:{2}, E:{3}, ...
           x:{4}".format(F(x0+x),g(x),f(x0),Ex,x))#Fjern '#' tegnet foran ...
           denne linjen for å kjøre koden med printing av E og deltaX
    elif(Ex>0.001):
        break

i=0
E = [0]*1000
```

```
for x in np.linspace(min,max,1000): #Loop for å beregne variasjon i E over ...
    et gitt intervall min -> max
    E[i] = np.longdouble(abs(f(x)-g(x=x,dx=valid)))
    i+=1

fig = plt.gcf()
fig.canvas.set_window_title('Funksjon 3: (1-x)/(x+3)  ')
x=np.linspace(min,max,1000)
y=F(x)
yd=f(x)
yg=G(x)
ye=E
plt.grid(True)
plt.plot(x,yd,'r-', label='f \'(x)')
plt.plot(x,y,'g-',label="F (x)")
plt.plot(x,yg,'b-',label="g (x)")
plt.plot(x,ye,color='purple',label='E (x) for deltaX %.3f'%(valid))
plt.legend()
plt.show()
```

A.1.4 Funksjon 4

```
import numpy as np
import matplotlib.pyplot as plt

x0 = 5
min = 0
max = 10

def f(x):
    return x/np.sqrt(1+x**2)

def g(dx,x=x0):
    return np.longdouble((F(x+dx)-F(x))/dx)

def G(x):  #To funksjoner for tilnærmingen g fordi dette gjør det enklere å...
    tegne grafen separert fra f'
    return np.longdouble((F(x+0.28)-F(x))/0.28)#DeltaX hardkodet inn etter ...
        funn i Del1:Oppgave1D

def F(x):
    return np.sqrt(1+x**2)


for x in np.arange(0.279,max-x0,0.000001):#Loop for å finne største ...
    gyldige DeltaX for E<=0.001. Loop grenser og stepsize funnet ved prøv&feil
    Ex = abs(f(x0)-g(x))
    if(Ex<=0.001):
        valid=x
        #print("F:{0}, G:{1}, f:{2}, E:{3}, ...
            x:{4}".format(F(x0+x),g(x),f(x0),Ex,x))#Fjern '#' tegnet foran ...
            denne linjen for å kjøre koden med printing av E og deltaX
    elif(Ex>0.001):
        break

i=0
E = [0]*1000
for x in np.linspace(min,max,1000): #Loop for å beregne variasjon i E over ...
    et gitt intervall min -> max
```

```
    E[i] = np.longdouble(abs(f(x)-g(x=x,dx=valid)))
    i+=1


fig = plt.gcf()
fig.canvas.set_window_title('Funksjon 4: sqrt(1+x )')
x=np.linspace(min,max,1000)
y=F(x)
yd=f(x)
yg=G(x)
ye=E
plt.grid(True)
plt.plot(x,yd,'r-', label='f \'(x)')
plt.plot(x,y,'g-',label="F (x)")
plt.plot(x,yg,'b-',label="g (x)")
plt.plot(x,ye,color='purple',label='E (x) for deltaX %.2f'%(valid))
plt.legend()
plt.show()
```

A.2 Del 2

A.2.1 Oppgave 1

```
'''
Denne koden tregner 1 input fra brukeren. Den trenger en startverdi på hvor
man tror det ene nullpunktet er. Hvis startverdien er nærme nok til at det ...
    kommer innenfor feilmarginen
før antall iterasjoner er nådd vil den si hvor nullpunktet er.

Hvis man ikke skulle komme frem til nullpunktet innen antallet iterasjoner ...
    jeg har lagt inn i koden
kan man i praksis øke dette tallet, men man har generelt valgt en dårlig ...
    start verdi om den skulle trenge
flere itersajoner...
'''

def f(x):
    return 3*x**2 + 4*x - 4

def g(x):
    return 6*x + 4

def newtonRaphson(x0,e,I):
    print('\nNewton Raphson metoden:')
    step = 1
    flag = 1
    condition = True

    while condition:
        if(g(x0) == 0.0): # Hvis startverdien vi skriver inn gjør den ...
            deriverte lik 0 stopper koden, kan ikke dele på 0
            print("kan ikke dele på 0")
            break

        x1 = x0 - f(x0)/g(x0)
        print('Iterasjon-%d, x1 = %0.12f og f(x1) = %0.12f' % (step, x1, ...
            f(x1)))
        x0 = x1
        step = step + 1
```

```
        if step > I:
            flag = 0
            break

        condition = abs(f(x1)) > e

    if flag == 1:
        print('\nDet ene nullpunktet er i: %0.12f' % x1)
    else:
        print("\nIkke konvergent.")

x0 = input("Startverdi: ") #Startverdi
e = 10**(-12) #max tillatte feilen
I = 100 #Max antall iterasjoner

x0 = float(x0)
e = float(e)
I = int(I)

newtonRaphson(x0,e,I)
```

A.2.2 Oppgave 2a

```
'''
Denne koden trenger 1 input fra brukeren. Den trenger en startverdi på hvor
man tror det ene nullpunktet er som da er . Hvis startverdien er nærme nok ...
    til at det kommer innenfor feilmarginen
før antall iterasjoner er nådd vil den si hvor nullpunktet er.

Hvis man ikke skulle komme frem til nullpunktet innen antallet iterasjoner ...
    jeg har lagt inn i koden
kan man i praksis øke dette tallet, men man har generelt valgt en dårlig ...
    start verdi om den skulle trenge
flere itersajoner...

Gode startverider er typisk verdier som er relativt nærme nullpunkter til ...
    grafen.
For denne grafen er gode startverdier f.eks.: "0" og "1"

Siden denne grafen er en trigonometrisk funksjon fortsetter den ut i ...
    uendelighet, derfor kan vi også plusse på 2*pi*k
der k er en integer, fordi det er nullpunkter i alle disse verdiene ut i ...
    uendeligheten.
'''

import math

x0 = input("Startverdi: ") #Startverdi
e = 10**(-12) #Max tillatte feilen
I = 100 #Max antall iterasjonerr

x0 = float(x0)
e = float(e)
I = int(I)

x = x0
cos = math.cos
sin = math.sin
```

```
def f(x):
    return (10 - (2 + 8.5*cos(x)))**2 + (6 - (8.5*sin(x)))**2 - 49

def g(x):
    return -102*cos(x) + 136*sin(x)

def newtonRaphson(x0,e,I):
    print('\nNewton Raphson metoden:')
    step = 1
    flag = 1
    condition = True

    while condition:
        if(g(x0) == 0.0): # Hvis startverdien vi skriver inn gjør den ...
            deriverte lik 0 stopper koden, kan ikke dele på 0
            print("kan ikke dele på 0")
            break

        x1 = x0 - f(x0)/g(x0)
        print('Iterasjon-%d, x1 = %0.12f og f(x1) = %0.12f' % (step, x1, ...
            f(x1)))
        x0 = x1
        step= step + 1

        if step > I:
            flag = 0
            break

        condition = abs(f(x1)) > e

    if flag == 1:
        print('\nDet ene nullpunktet er i: %0.12f' % x1)
    else:
        print("\nIkke konvergent.")

newtonRaphson(x0,e,I)
```

A.2.3 Oppgave 2b

```
'''
Denne koden tregner 1 input fra brukeren. Den trenger en startverdi på hvor
man tror det ene nullpunktet er. Hvis startverdien er nærme nok til at det ...
    kommer innenfor feilmarginen
før antall iterasjoner er nådd vil den si hvor nullpunktet er.

Hvis man ikke skulle komme frem til nullpunktet innen antallet iterasjoner ...
    jeg har lagt inn i koden
kan man i praksis øke dette tallet, men man har generelt valgt en dårlig ...
    start verdi om den skulle trenge
flere itersajoner...

Gode startverider er typisk verdier som er relativt nærme nullpunkter til ...
    grafen.
For denne grafen er gode startverdier f.eks.: "0" og "1"

Siden denne grafen er en trigonometrisk funksjon fortsetter den ut i ...
    uendelighet, derfor kan vi også plusse på 2*pi*k
```

```
der k er en integer, fordi det er nullpunkter i alle disse verdiene ut i ...
    uendeligheten.
'''

import math

x0 = input("Startverdi: ") #Startverdi
e = 10**(-12) #Max tillatte feilen
I = 100 #Max antall iterasjoner

x0 = float(x0)
e = float(e)
I = int(I)

x = x0
cos = math.cos
sin = math.sin

def f(x):
    return (10 - (8.5*cos(x)))**2 + (6 - (2 + 8.5*sin(x)))**2 - 49

def g(x):
    return -68*cos(x) + 170*sin(x)

def newtonRaphson(x0,e,I):
    print('\nNewton Raphson metoden:')
    step = 1
    flag = 1
    condition = True

    while condition:
        if(g(x0) == 0.0): # Hvis startverdien vi skriver inn gjør den ...
           deriverte lik 0 stopper koden, kan ikke dele på 0
            print("kan ikke dele på 0")
            break

        x1 = x0 - f(x0)/g(x0)
        print('Iterasjon-%d, x1 = %0.12f og f(x1) = %0.12f' % (step, x1, ...
           f(x1)))
        x0 = x1
        step = step + 1

        if step > I:
            flag = 0
            break

        condition = abs(f(x1)) > e

    if flag == 1:
        print('\nDet ene nullpunktet er i: %0.12f' % x1)
    else:
        print("\nIkke konvergent.")

newtonRaphson(x0,e,I)
```

A.2.4 Oppgave 2c

```
'''
Denne koden tregner 1 input fra brukeren. Den trenger en startverdi på hvor
```

```
man tror det ene nullpunktet er. Hvis startverdien er nærme nok til at det ...
    kommer innenfor feilmarginen
før antall iterasjoner er nådd vil den si hvor nullpunktet er.

Hvis man ikke skulle komme frem til nullpunktet innen antallet iterasjoner ...
    jeg har lagt inn i koden
kan man i praksis øke dette tallet, men man har generelt valgt en dårlig ...
    start verdi om den skulle trenge
flere itersajoner...

Gode startverider er typisk verdier som er relativt nærme nullpunkter til ...
    grafen.
For denne grafen er gode startverdier f.eks.: "0" og "1"

Siden denne grafen er en trigonometrisk funksjon fortsetter den ut i ...
    uendelighet, derfor kan vi også plusse på 2*pi*k
der k er en integer, fordi det er nullpunkter i alle disse verdiene ut i ...
    uendeligheten.
'''

import math

x0 = input("Startverdi: ") #Startverdi
e = 10**(-12) #Max tillatte feilen
I = 100 #Max antall iterasjoner

x0 = float(x0)
e = float(e)
I = int(I)

x = x0
cos = math.cos
sin = math.sin

def f(x):
    return (10 - (-2 + 8.5*cos(x)))**2 + (6 - (8.5*sin(x)))**2 - 49

def g(x):
    return -102*cos(x) + 204*sin(x)

def newtonRaphson(x0,e,I):
    print('\nNewton Raphson metoden:')
    step = 1
    flag = 1
    condition = True

    while condition:
        if(g(x0) == 0.0): # Hvis startverdien vi skriver inn gjør den ...
            deriverte lik 0 stopper koden, kan ikke dele på 0
            print("kan ikke dele på 0")
            break

        x1 = x0 - f(x0)/g(x0)
        print('Iterasjon-%d, x1 = %0.12f og f(x1) = %0.12f' % (step, x1, ...
            f(x1)))
        x0 = x1
        step = step + 1

        if step > I:
            flag = 0
            break
```

```
        condition = abs(f(x1)) > e

    if flag == 1:
        print('\nDet ene nullpunktet er i: %0.12f' % x1)
    else:
        print("\nIkke konvergent.")

newtonRaphson(x0,e,I)
```