Indholdsfortegnelse

Abstract

This paper will explain the use of different numerical methods for solving nonlinear equations. The methods are Newton-Raphson's method, the bisection method and the secant method. The theory and how to use each of the methods will be explained including: convergence analysis, error estimation, stopping criteria and order of convergence. The paper will also give examples of how to find the roots of two different nonlinear equations. The roots will be calculated using the methods mentioned above. In addition I have written a program utilizing the Newton-Raphson's method and the secant method to determine the roots. It is often impossible to calculate an exact root of a nonlinear equation, therefore we use these different methods to make a very accurate approximation of the exact root. This approximation can be made as exact as required by carrying out enough iterations.

Indledning

I denne opgave vil jeg beskrive, hvordan man finder roden eller f(x)=0. Jeg vil benytte tre forskellige metoder hhv. sekant metoden, Newton's metode og bisektionsmetoden. Derudover har jeg også skrevet et program, som også kan benytte de forskellige metoder.

Kildehenvisninger i denne opgave vil være markeret med et tal som f.eks [2] for at se kilden, skal man kigge i litteraturlisten efter nummeret. Ligeledes vil tegn som f.eks. ∩ blive brugt til at navngive formler og ligheder, for at forøge overskueligheden.

Konvergens

Man taler om konvergens i sammenhæng med rækker. At en række konvergerer betyder at den går imod en konstant. Et eksempel på dette kunne være $2\text{-}n^{-1}$, når n går mod uendelig vil denne række gå mod 2, men den vil aldrig antage værdien 2. Dette da der altid bliver fratrukket en uendelig lille smule, og vi siger så at rækken konvergere mod 2.

Hvis rækken ikke går mod en konstant må den gå imod $-\infty$ eller $+\infty$ og rækken kaldes så divergent. Et lille eksempel 2^n vil divergere mod uendelig. Den enkelte metodes konvergens bliver beskrevet nærmere under metoden.

Konvergenskriterier

Alt efter hvilken metode man benytter, er der forskellige kriterier for konvergens. Benyttes en metode som bisektionsmetoden er der ingen konvergenskriterier. Benyttes derimod Newtons metode skal gættet være tæt nok på roden for at metoden konvergerer, dette ligeledes for sekant metoden.

Konvergensorden

Konvergensorden er et udtryk for hvor hurtigt en metode nærmer sig roden under de rette forudsætninger. I f.eks Newtons metode og sekant metoden skal gættet ligge tæt på roden ellers vil den enten divergere eller konvergere meget langsomt. Derfor bruger man tit en kombination af metoderne, som f.eks. at starte med bisektionsmetoden for hurtigt at nærme sig roden, hvor Newtons metode muligvis vil konvergere meget langsomt til start (hvis tangenten til gættet er meget stejl). Konvergensorden vedrørende specifikke metoder er skrevet under disse.

Fejlestimater

Et fejlestimat er et udtryk for hvor stor fejlen er, typisk vil der være uligheder, så du ved at din fejl er mindre end en bestemt værdi. Fejlestimater kan være vigtige, hvis man bliver bedt om at udregne noget med en bestemt præcision. Fejlen for et estimat må være roden fratrukket approximationen til roden. Et fejlestimat skal helst være faldende med antallet af iterationer hvis dette ikke er tilfældet er vores metode ikke ved at konvergere. Uddybende forklaringer af fejlestimater for de enkelte metoder står under disse.

Stopkriterier

Når man regner numerisk, regner man ikke eksakt. Det er derfor nødvendigt for os at indføre stopkriterier til vores numeriske metoder, for at de ikke fortsætter i al evighed. Et stopkriterie kan f.eks være at vores fejlestimat skal være mindre end en konstant. Når det er opnået stoppes den iterative proces og vi accepterer vores approximation som en løsning. Da man ikke på forhånd kan vide hvor mange iterationer det vil tage at nå en given præcision programmerer man tit et stopkriterie, således at der kun bliver gennemløbet et bestemt antal iterationer. På denne måde undgår man at sætte computeren igang med en evig udregning.

Taylor's Theorem

Taylor's theorem er tilknyttet til teorien om numeriske metoder. Jeg vælger derfor at opskrive teoremet da jeg anvender det i nogle af mine udregninger.

Hvis f er en C^{n+1} funktion, (C^{n+1} betyder at den n+1'te afledte er kontinuert) defineret på et lukket interval [a,b]. Da for et hvert punkt x og x+h som ligger i [a,b] gælder

$$f(x+h) = \sum_{k=0}^{n} \left(\frac{h^k}{k!} f^{(k)}(x) \right) + \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\epsilon)$$

Hvor Epsilon ligger mellem x og x+h. (når der skrives $f^{(k)}$ menes der den k'te afledte af f ikke at forveksle med f opløftet i k'te). [1]

Numeriske metoder

Numeriske metoder er de metoder vi bruger til at tilnærme os nulpunktet. Det er ofte umuligt at udregne det præcise nulpunkt og vi bliver derfor nødt til at benytte en eller flere af disse metoder.

Newtonsmetode

Newtonsmetode benytter sig af iterationen

$$x_{(n+1)} = x_n - \left(\frac{f(x_n)}{f'(x_n)} \right)$$

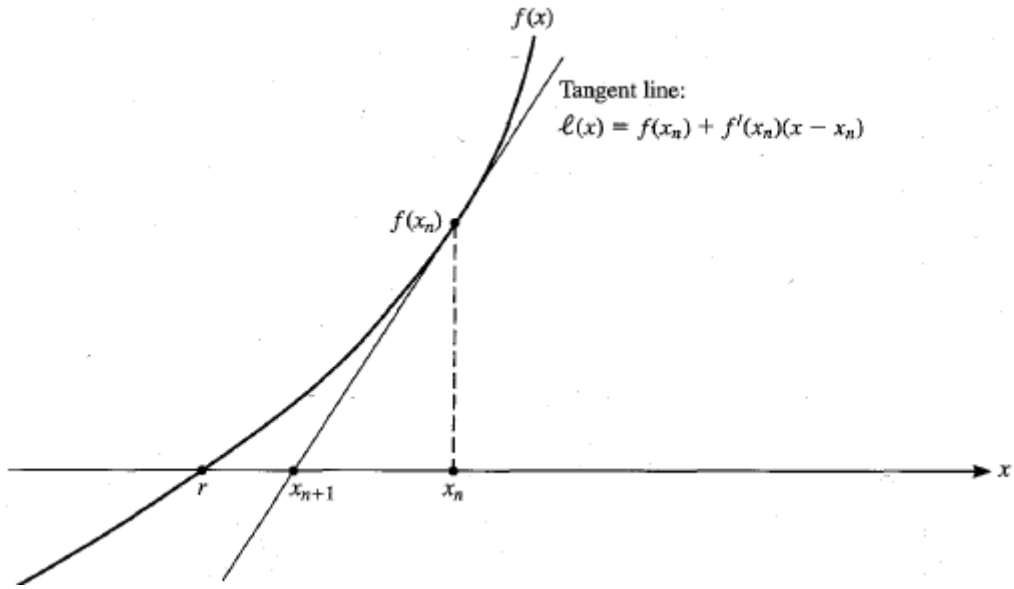
for $n \geq 0$.

Denne iteration kræver at man har den afledte af f altså f '.

Vi kan starte med et gæt på en rod x_0. Vi laver en iteration, som derefter vil give os x_1. Derefter indsætter vi x_1 i Newton-Raphsons iteratio og finder x_2. Dette fortsætter man med til vores stopkriterier er nået (se stopkriterier).

Vi ser på en grafisk beskrivelse af Newtons metode. Vi befinder os i den n'te iteration. Tangenten til $f(x_n)$ følges til skæringspunktet med x-aksen. Dette punkt er x_{n+1}. Når vi dividerer $f(x_n)$ med $f'(x_n)$ finder vi længden mellem x_n og x_{n+1}, når denne længde trækkes fra x_n får vi x_{n-1}. Dette fortsætter man med at gøre til man kommer tilpas tæt på r, som er roden.

2



Newtons metode er den hurtigste af mine tre metoder henholdsvis; Newtons metode, bisektionsmetoden og sekant metoden. Grunden til dette er at Newtons metode har en kvadratisk konvergens.

Fejlanalyse

Fejlen for den n'te iteration vil vi betegne med $e_n=x_n-r$. Vi antager at f '' er kontinuert og at $f'(r) \neq 0$. f '' antages kontinuert da vi ønsker at benytte Taylors teorem.

$$e_{n+1} = x_{n+1} - r = x_n - \frac{f(x_n)}{f'(x_n)} - r = e_n - \frac{f(x_n)}{f'(x_n)} = \frac{e_n \cdot f'(x_n)}{f'(x_n)} - \frac{f(x_n)}{f'(x_n)} = \frac{e_n \cdot f'(x_n) - f(x_n)}{f'(x_n)}$$

Her har jeg benyttet at $e_n=x_n-r$ og sat på fælles brøkstreg.

Jeg vil nu benytte Taylors teorem, dette kan jeg gøre eftersom f '' er kontinuert.

$$0 = f(r) = f(x_n - e_n) = f(x_n) - e_n \cdot f'(x_n) + \frac{1}{2} e_n^2 \cdot f''(\epsilon_n)$$

Hvor Epsilon er et tal mellem r og x_n. Ved at omskrive venstre siden og højre siden af ovenstående ligning fås.

$$e_n f'(x_n) - f(x_n) = \frac{1}{2} f''(\epsilon_n) e_n^2$$

Ved at indsætte ovenstående i fås venstre siden af nedenstående udtryk, når n er høj vil x_n være tæt på r og vi kan derfor lave approximationen set nedenunder.

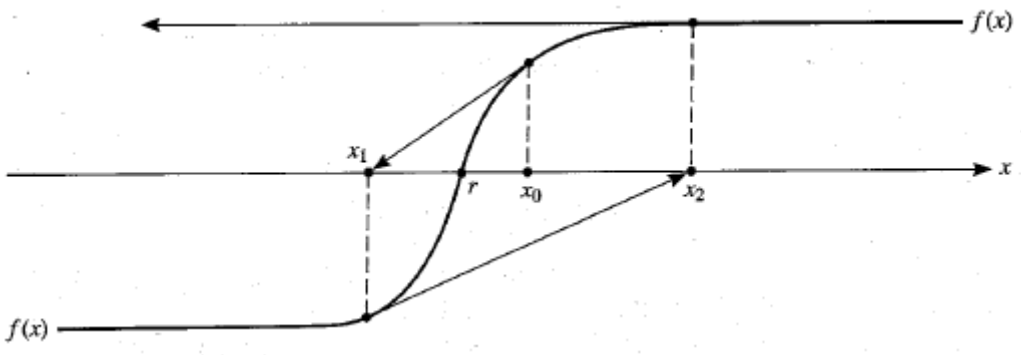
$$e_{n+1} = \frac{1}{2}\frac{f''(\epsilon_n)}{f''(x_n)}e_n^2 \approx \frac{1}{2}\frac{f''(r)}{f''(r)}e_n^2 = C \cdot e_n^2$$

Når vores x_n er tiltrækkeligt tæt på r vil e_n være mindre end 1. Vi ser at vores n+1'te fejl er kvadratet på den n'te fejl gange en konstant. Eftersom at e_n er under 1 vil den næste fejl blive mindre end den foregående fejl. Da fejlen bliver mindre med kvadratet på den foregående kaldes denne form for konvergens kvadratisk konvergens. Dette er en meget hurtig form for konvergens.

Divergens

Som sagt er Newtons metode ikke garanteret konvergens. Derfor bruger man tit Newtons metode i kombination med andre langsommere metoder som til gengæld konvergerer globalt (overalt). Denne hybrid metode kan så laves så den konvergerer globalt og er hurtigere end de enkelte metoder hver for sig. Et simpelt eksempel på en funktion som ikke er globalt konvergent med Newtons metode ses herunder. Det ses at hvis x_0 vælges så langt fra r så tangenten i $f(x_0)$ har hældning < 1 da vil tangenten i $f(x_1)$ få endnu mindre hældning og dernæst vil tangenten i $f(x_2)$ få endnu mindre hældning osv. X_n vil derfor bevæge sig længere og længere væk fra r.

3



Bisektionsmetoden

Bisektionsmetoden bruges normalt mest til grov placering af roden. Hvis funktionen har flere rødder i det interval man vælger, kan bisektionsmetoden volde problemer fordi produktet af f(a) og f(b) ikke nødvendigvis er negativt. Hvis produktet af f(a) og f(b) er positivt har vi et lige antal rødder, og er funktionsværdien negativ har vi et ulige antal rødder. Hvis produktet af f(a) og f(b) er 0 da må enten f(a) eller f(b) være 0 altså være funktionens rod. Det samme gælder hvis $f(a_n)$ gange $f(b_n)$ er 0 (det er dog meget usandsynligt at man gætter roden). Når man benytter bisektionsmetoden er det første man skal have et interval, hvor man ved at roden ligger inden for. Man finder dette interval ved at gange f(a) med f(b). Hvis f(a) gange f(b) er positivt er enten både f(a) og f(b) positive eller negative, og der er derfor ingen rod i dette interval. Hvis f(a) gange f(b) derimod er negativt, ved vi at roden må ligge i intervalet [a,b] fordi et positivt tal gange et negativt giver et nyt negativt. Vi kan kun have både et positivt tal og et negativt tal hvis funktionen krydser x-aksen i dette interval (forudsat vi har en kontinuert funktion på [a,b]).

Fremgangsmåden for bisektionsmetoden bliver således. Vi definerer midtpunktet i intervallet $c=½(a+b)$. Vi checker så om f(a) gange f(c) er negativt, er den negativ ved vi at roden ligger i dette interval og vi kalder så c for b_1 og a for a_1. Vi benytter så metoden forfra i intervallet $[a_1,b_1]$ hvor vi starter med at definere c_1 osv. Hvis produktet af f(a) og f(c) ikke er negativt ved vi at roden må ligge i [c,b], vi kalder så c for a_1 og b for b_1, og fortsætter metoden på dette interval.

Fejlanalyse

Vi kalder vores originale interval $[a,b]=[a_0,b_0]$ det er klart at længden på $[a_1,b_1]$ er den halve af $[a_0,b_0]$. Længden (b_n-a_n) er altså længden på (b_0-a_0) halveret n gange. Jeg kan således skrive længden $b_n-a_n = 2^{-n}\,(b_0-a_0)$. Når vi ikke gider lave flere halveringer er vi tilbage med et interval hvori roden ligger. Det bedst kvalifecerede gæt vi kan lave på roden er i midten af det sidste interval. Altså vores gæt $c_n=(a_n+b_n)/2$. Hvis c_n er vores gæt og roden skal ligge i $[a_n,b_n]$ så må fejlen være $e_n \leq ½(b_n-a_n)$. Samlet set er vi altså kommet frem til at:

$$e_n = \frac{1}{2}e_{n-1} = |r - c_n| \leq \frac{1}{2}(b_n - a_n) = 2^{-(n+1)} \cdot (b_0 - a_0)$$

Hvis man kun ser på det første lighedstegn er det tydeligt at konvergensordenen er lineær, e_{n-1} er nemlig ikke opløftet i noget, men bare ganget med en konstant.

Vores fejlestimat er altså begrænset af følgende ulighed.

$$e_n \leq 2^{-(n+1)} \cdot (b_0 - a_0)$$

Et lille eksempel på et fejlestimat $[a_0,b_0]=[3,7]$

$e_5 \leq 2^{-(5+1)}\,(7-3) = 1/16$

Divergens

Givet fremgangsmåden for bisektionsmetoden er det tydeligt, at vi ikke kan have problemer med divergens i denne metode.

Sekant metoden

En af ulemperne ved Newtons metode er at man skal bruge den afledte af funktionen. I sekant metoden undgår vi denne ulempe ved at lave en approximation. Tricket er at f '(x) skrives ved hjælp af definitionen på en differentionskvotient

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Denne approximation bliver bedre og bedre jo tættere x_{n-1} er på x_n.

Vi erstatter nu f '(x_n) i Newtons metode med ovenstående approximation og får:

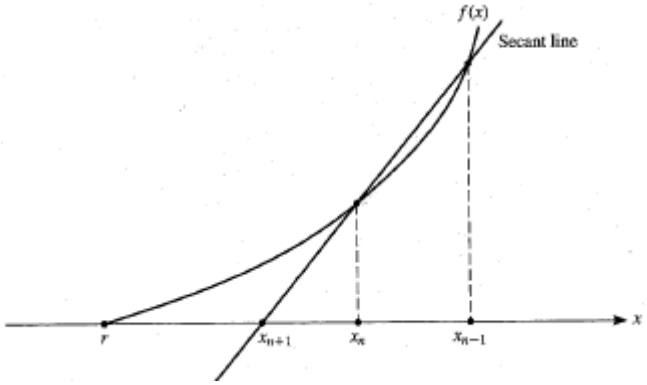
$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

for $n \geq 1$.

Vores iteration er altså ikke længere afhængig af den afledte f(x). Tilgengæld kræver sekant

metoden to start gæt (x_0 og x_1) for at kunne bestemme x_2.

På billedet nedenfor ses sekant-metoden benyttet på en funktion. Som vi kan se tegnes sekanten mellem to punkter på grafen, denne sekantlinies skæring med x-aksen giver os den næste værdi af x (foreløbigt gæt på rod).[4]



Fejlanalyse

Jeg vil bestemme konvergensordenen af sekant metoden.

Fra definitionen på sekant metoden samt at $e_n=x_n-r$ kan vi skrive:

$$e_{n+1} = x_{n+1} - r = \frac{[f(x_n)x_{n-1} - f(x_{n-1})x_n]}{[f(x_n) - f(x_{n-1})]} - r = \frac{[f(x_n)(e_{n-1}+r) - f(x_{n-1})(e_n+r)]}{[f(x_n) - f(x_{n-1})]} - r$$

$$\frac{[f(x_n)e_{n-1} + f(x_n)r - f(x_{n-1})e_n - f(x_{n-1})r]}{[f(x_n) - f(x_{n-1})]} - r = \frac{[f(x_n)e_{n-1} - f(x_{n-1})e_n]}{[f(x_n) - f(x_{n-1})]} + \frac{[f(x_n)r - f(x_{n-1})r]}{[f(x_n) - f(x_{n-1})]} - r =$$

$$\frac{[f(x_n)e_{n-1} - f(x_{n-1})e_n]}{[f(x_n) - f(x_{n-1})]} + r\frac{[f(x_n) - f(x_{n-1})]}{[f(x_n) - f(x_{n-1})]} - r = \frac{[f(x_n)e_{n-1} - f(x_{n-1})e_n]}{[f(x_n) - f(x_{n-1})]}$$

Ved at faktorisere e_ne_{n-1} ud og gange med 1 på en smart måde, $(x_n-x_{n-1})/(x_n-x_{n-1})$ får vi:

$$e_{n+1} = \left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right] \cdot \left[\frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{x_n - x_{n-1}}\right] e_n e_{n-1}$$

Ved brug af Taylors teorem får jeg nedenstående. Når jeg skriver $O(e_n^3)$ betyder dette at udtrykket er højest samme størrelsesorden som e_n^3. Hvilket vil sige at $O(e_n^3) \leq Ce_n^3$. Alt dette bunder i at det dominerende led i Taylorrækken er e_n^3 leddet, da de senere led vil indeholde e_n i højere potens, og da e er mindre end 1, vil leddet blive mindre des højere e_n's potens er.

$$f(x_n) = f(r + e_n) = f(r) + e_n f'(r) + \frac{1}{2} e_n^2 f''(r) + O(e_n^3)$$

Jeg dividerer ovenstående udtryk med e_n på begge sider af lighedstegnet og husker at f(r) = 0.

$$\frac{f(x_n)}{e_n} = f'(r) + \frac{1}{2} e_n f''(r) + O(e_n^2)$$ 

Her har jeg blot ændret indekset til n-1 ingen matematik er udført.

$$\frac{f(x_{n-1})}{e_{n-1}} = f'(r) + \frac{1}{2} e_{n-1} f''(r) + O(e_{n-1}^2)$$ 

Ved at trække  fra  får vi nedenstående lighed.

$$\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}} = \frac{1}{2}(e_n - e_{n-1}) f''(r) + O(e_{n-1}^2)$$

Da $x_n - x_{n-1} = e_n - e_{n-1}$ får vi nedenstående. Dette er skrevet som en approximation, da vi har undladt leddet $O(e^2_{n-1})$ denne approximation har ikke stor betydning, da e gerne skulle blive meget lille og e^2 derfor endnu mindre.

$$\frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{x_n - x_{n-1}} \approx \frac{1}{2} f''(r)$$ 

Den første firkantede parentes i  er opskrevet nedenfor. Vi ser at hvis man vender brøken har vi vores approximation for f '(r) nemlig differenskvotienten. Nedenstående udtryk på altså være den inverse af f '(r).

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \approx \frac{1}{f'(r)}$$ 

Vi ser at  og  er approximationer for henholdsvis den første og anden firkantede parentes i  . Vi indsætter disse approximationer og får følgende.

$$e_{n+1} \approx \frac{1}{2} \cdot \frac{f''(r)}{f'(r)} e_n e_{n-1} = C \cdot e_n e_{n-1}$$

Vi kan her se at vores fejl minder meget om fejlen fra Newtons metode. Den eneste forskel er at i Newtons metode havde vi C e^2_n og her har vi C $e_n e_{n-1}$. Denne konvergens vil derfor ikke være kvadratisk medmindre at $e_n = e_{n-1}$. Dette giver dog ingen mening, da fejlen må antages at blive mindre og mindre da vores metode ellers ikke ville konvergere. Vi kan således nu konkludere at konvergensordenen er mindre end kvadratisk (for $e^a_n = e_n e_{n-1}$ må a < 2 da e_{n-1} er mindre end e_n, husk a =2 er kvadratisk konvergensorden). $e_n e_{n-1} < e_n$ da e_n og e_{n-1} er mindre end 1, da vi har $e^a_n = e_n e_{n-1}$ og vi ved at højresiden bliver mindre end e_n må venstre siden også skulle blive mindre end e_n. Dette kan kun ske på én måde og det er hvis a>1 da e_n<1. Jeg har således vist at 1<a<2 altså mindre end

kvadratisk konvergensorden, men større end lineær konvergensorden. Denne form for konvergensorden har derfor fået navnet superliniær konvergensorden.

Divergens

Sekant metoden har samme divergens muligheder som Newtons metode. Da en sekant, som en tangent, også kan bringe os længere væk fra løsningen. Se Newtons metode divergens for illustration.

Opgaver

Herunder vil der blive løst to opgaver. Den første opgave har jeg valgt at løse med Newtons metode og det program jeg har programeret vil også løse opgaven med Newtons metode.

Opgave 1

Jeg skal bestemme nulpunktet for funktionen.

$$f(x) := 10^{-x} - 10$$

Jeg vil benytte Newton-Raphsons metode.

$$x_{(n+1)} = x_n - \left(\frac{f(x_n)}{f'(x_n)} \right)$$

Det første man gør er at gætte på der hvor nulpunktet er. Jeg gætter på -2.

Så indsætter jeg mit gæt i Newton-Raphsons metode.

$$-2 - \left(\frac{f(-2)}{f'(-2)} \right)$$

Derefter får jeg udregnet det tal jeg skal fortsætte udregningerne med. Dette er det næste bud på min rod. Jeg benytter metoden igen for at udregne et mere præcist bud på den rigtige rod.

$$= -1.60913$$

Så indsætter jeg det nye tal og udregner næste.

$$-1.60913 - \left(\frac{f(-1.60913)}{f'(-1.60913)} \right)$$

$$= -1.28166$$

Nu gentages processen til mine stopkriterer nås.

$$-1.28166 - \left(\frac{f(-1.28166)}{f'(-1.28166)} \right)$$

$$= -1.07442$$

$$-1.07442 - \left(\frac{f(-1.07442)}{f'(-1.07442)}\right)$$

$$= -1.00603$$

$$-1.00603 - \left(\frac{f(-1.00603)}{f'(-1.00603)}\right)$$

$$= -1.00004$$

$$-1.00004 - \left(\frac{f(-1.00004)}{f'(-1.00004)}\right)$$

$$= -1.$$

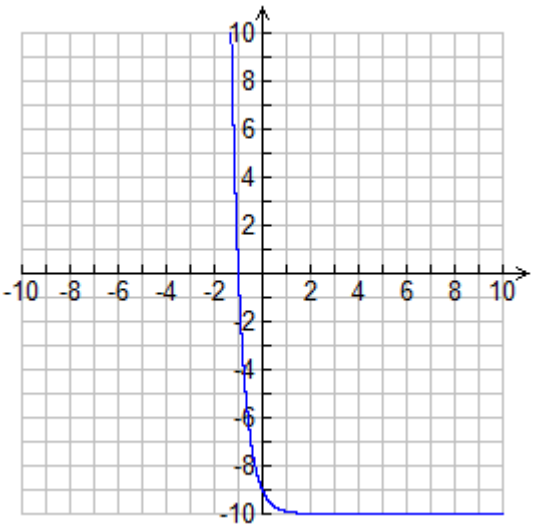
$$-1. - \left(\frac{f(-1.)}{f'(-1.)}\right)$$

$$= -1.$$

Nu sker der ingen ændringer i tallet længere og jeg må derfor gå ud fra at jeg har nået nulpunktet.

Det skal dog siges at det program jeg bruger til udregningerne (TI interactive) udregner med et bestemt antal decimaler og selv om den siger at svaret er -1 er det muligvis bare meget tæt på -1, men rent faktisk ikke -1, der er roden.

Derefter ser jeg på en graf for at se om mit resultat ser nogenlunde rigtigt ud.



Som det ses på grafen ovenfor ser -1 ud til at være roden i min funktion. Ofte vil man kun kunne lave en meget præcis tilnærmelse. Mit gæt har været udemærket og derfor har mine udregninger hurtigt konvergeret mod roden.

Nedenunder har jeg intastet samme startgæt, altså -2 i mit computerprogram som jeg har skrevet.

```
Indtast 1, 2 eller 3.
Hvis 1 Newtons metode til at finde 0 punkt af funktion f(x)
Hvis 2 Newtons metode til at finde 0 punkt af funktion g(x)
Hvis 3 Sekant metode til at finde 0 punkt af funktion g(x)
1

Her vil vi ved hjælp af Newton's metode lave en 0 punkts søgning af f(x)
indtast det ønskede startgæt
-2
nr 1 udregning er -1.60913
nr 2 udregning er -1.28166
nr 3 udregning er -1.07442
nr 4 udregning er -1.00603
nr 5 udregning er -1.00004
nr 6 udregning er -1
nr 7 udregning er -1
nr 8 udregning er -1
Vil du tilbage til startmenu?
Tast j, J, n eller N
```

Som det kan ses nærmer programmet sig nulpunktet præcis som da jeg selv lavede en tilnærmelse til nulpunktet. (for forklaring bag programmet se computerprogram).

Opgave 2

Jeg skal bestemme 0 punktet for funktionen.

$$g(x) := 10^{-x} - \left(\frac{1}{10}\right)$$

Jeg vil benytte sekant metoden.

$$x_{(n+1)} := x_n - f(x_n)\cdot\left[\frac{x_n - x_{(n-1)}}{f(x_n) - f(x_{(n-1)})}\right]$$

Når man benytter sekant metoden skal man bruge to gæt jeg gætter på 0 og -1.

Så indsætter jeg mine gæt i sekant metoden og får derefter udregnet mit nye gæt på roden.

$$0 - g(0)\cdot\left(\frac{0 - (-1)}{g(0) - g(-1)}\right)$$

Mit nye gæt på roden bliver.

$$= .1$$

Så indsætter jeg mit nye tal som x_{n-1} og mit gamle x_{n-1} bliver det nye x_n og udregner næste gæt.

$$-1 - g(-1)\cdot\left(\frac{-1 - 0.1}{g(-1) - g(0.1)}\right)$$

$$= .182966$$

Nu gentages processen til mine stopkriterier nåes.

$$0.1 - g(0.1)\cdot\left(\frac{0.1 - .182966}{g(0.1) - g(.182966)}\right)$$

$$= .517034$$

$$.182966 - g(.182966)\cdot\left(\frac{.182966 - .517034}{g(.182966) - g(.517034)}\right)$$

$$= .71063$$

$$.517034 - g(.517034)\cdot\left(\frac{.517034 - .71063}{g(.517034) - g(.71063)}\right)$$

$$= .878273$$

$$.71063 - g(.71063)\cdot\left(\frac{.71063 - .878273}{g(.71063) - g(.878273)}\right)$$

$$= .965255$$

$$.878273 - g(.878273)\left(\frac{.878273 - .965255}{g(.878273) - g(.965255)}\right)$$

$$= .995414$$

$$.965255 - g(.965255)\left(\frac{.965255 - .995414}{g(.965255) - g(.995414)}\right)$$

$$= .999819$$

$$.995414 - g(.995414)\left(\frac{.995414 - .999819}{g(.995414) - g(.999819)}\right)$$

$$= .999999$$

$$.999819 - g(.999819)\left(\frac{.999819 - .999999}{g(.999819) - g(.999999)}\right)$$

$$= 1.$$

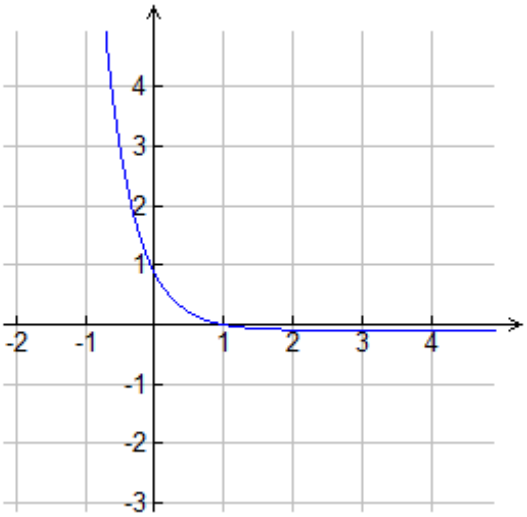
$$.999999 - g(.999999)\left(\frac{.999999 - 1.}{g(.999999) - g(1.)}\right)$$

$$= 1.$$

Nu sker der ikke længere ændringer i tallet og jeg går derfor ud fra at jeg har nået nulpunktet.

Det skal dog siges at jeg bruger et program til udregningerne (TI interactive) og programmet bruger et bestemt antal decimaler, så selv om den siger at svaret er 1 er det muligvis bare meget tæt på 1, men rent faktisk ikke 1 der er roden.

Derefter ser jeg på en graf for at se om mit resultat ser nogenlunde rigtigt ud.



Som det ses på grafen ovenfor ser 1 ud til at være roden i min funktion. Mine gæt har været udemærkede og derfor har mine udregninger hurtigt konvergeret mod roden.

Nedenunder ses udregningerne udregnet i det computerprogram jeg selv har skrevet.

```
Her vil vi ved hjælp af sekantmetoden laves en 0 punkts søgning af g(x)
intast det første ønskede startgæt
0
Intast det andet ønskede startgæt
-1
nr 1 udregning er -1
nr 2 udregning er 0.1
nr 3 udregning er 0.182966
nr 4 udregning er 0.517035
nr 5 udregning er 0.710631
nr 6 udregning er 0.878273
nr 7 udregning er 0.965255
nr 8 udregning er 0.995414
nr 9 udregning er 0.999819
nr 10 udregning er 0.999999
nr 11 udregning er 1
nr 12 udregning er 1
```

Som det kan ses tilnærmer programmet sig på præcis samme måde som udregningerne, fordi jeg har benyttet sekant metoden begge steder.

Computerprogram

Jeg har skrevet et computerprogram der kan løse både f(x) og g(x) ved hjælp af Newton-Rapsons metode. Programmet starter med en lille introduktion hvor det fortæller, hvad det kan. Derefter kommer man ind i en menu med forskellige valgmuligheder hvor man kan vælge hvad man vil lave. Efter hvilken menu man vælger kan man enten: løse f(x) og g(x) ved hjælp af Newton-Raphsons metode eller løse g(x) ved hjælp af sekant metoden. Efter programmet har regnet, bliver man spurgt om man vil tilbage til startmenuen. Dette for at man ikke behøver at genstarte programmet, hvis man ønsker at prøve forskellige metoder. Jeg har også lavet et break inde i løkken der går, at hvis man får det samme resultat flere gange stopper udregningerne. Hvis ikke programmet når et resultat

stopper programmet med at regne efter 100 udregninger. Dette har jeg sat den til for at den ikke skal regne for længe. Nedenfor kan man se det program jeg har skrevet i Dev C++.

```
#include <iostream>
#include <cmath>//matematik program til bland andet power og squareroot (potens)

using namespace std;

const char ae = -111; //lille æ
const char oe = -101; //lille ø
const char aa = -122; //lille å
int main()
{
   int besked;
char igen;
//do-while-løkke, fordi den skal kører mindst én gang
//og så længe man taster j eller J
  double x1, x, x2, f, diff,y, g, gn, difg; //liste af ting med kommatal der skal huskes.

    cout<<"Dette er et program der skal udregne og tiln"<<ae<<"rme sig 0 punkter"<<endl;
    cout<<"Programmet vil benytte Newtons motode og sekant metoden"<<endl;
    cout<<"Tryk p"<<aa<<" en hvilken som helst tast for at forts"<<ae<<"tte til startmenu"<<endl;
      system("PAUSE"); //stopper programmet til der trykkes på en tast
    do
{
  system("cls"); //sletter skærmen
    cout << "Indtast 1, 2 eller 3." << endl;
    cout << "Hvis 1 Newtons metode til at finde 0 punkt af funktion f(x)"<<endl;
    cout << "Hvis 2 Newtons metode til at finde 0 punkt af funktion g(x)"<<endl;
    cout << "Hvis 3 Sekant metode til at finde 0 punkt af funktion g(x)"<<endl;
cin >> besked;
cout << endl;
//switch-case fungerer på samme måde som if-else
switch(besked)
{
case 1:
cout << "Her vil vi ved hj"<<ae<<"lp af Newton's metode lave en 0 punkts s"<<oe<<"gning af f(x)" <<
cout<< "indtast det "<<oe<<"nskede startg"<<ae<<"t" << endl;
cin>>x;


for (int q = 1 ; q <= 100 ; q++)//en løkke der kører mange gange
{
f=pow(10,-x)-10;//selve funktionen
diff=log(0.1)*pow((0.1),x);//den afledte funktion
  x1=x-(f/diff);//Newton-ralpson's metode der udregnes
  cout<<"nr "<<q<<" udregning er "<<x1<<endl;//fortæller om udregningen
```

```
 if (x==x1) //stopper løkken hvis vi får samme rod
 {
    break;
  }
  x=x1;//den gamle værdi udskiftes med det nye udregnede tal og starter forfra
}

break;
case 2:
cout << "Her vil vi ved hj"<<ae<<"lp af Newton's metode lave en 0 punkts s"<<oe<<"gning af g(x)" <<
cout<< "indtast det "<<oe<<"nskede startg"<<ae<<"t" << endl;
cin>>x;


for (int q = 1 ; q <= 100 ; q++)//løkke
{
g=pow(10,-x)-0.1;//funktionen g(x)
difg=log(0.1)*pow((0.1),x);//den afledte funktion
  x1=x-(g/difg);//udregningen til næste punkt
  cout<<"nr "<<q<<" udregning er "<<x1<<endl;//hvad nummer udregning vi er nået til
  if (x==x1) //stopper løkken hvis vi får samme resultat
 {
  }
  x=x1;//indsætter det nye gæt på roden i det gamle gæts plads
}


break;
case 3:
cout << "Her vil vi ved hj"<<ae<<"lp af sekant metoden lave en 0 punkts s"<<oe<<"gning af g(x)" <<
cout << "indtast det f"<<oe<<"rste "<<oe<<"nskede startg"<<ae<<"t" << endl;
cin>>x;//første gæt
cout << "Indtast det andet "<<oe<<"nskede startg"<<ae<<"t" << endl;
cin>>x1;//andet gæt

for (int q = 1 ; q <= 100 ; q++)//løkke
{
g=pow(10,-x)-0.1;//selve funktionen
gn=pow(10,-x1)-0.1;//funktionen til punkt 2
  x2=x-g*((x-x1)/(g-gn));//sekant metoden
  cout<<"nr "<<q<<" udregning er "<<x1<<endl;//hvad nummer udregning
  if (x1==x2) //stopper løkken når vi får samme resultat
 {
    break;
```

```
90    }
91    x=x1;//tager 2. gæt og indsætter det som første
92    x1=x2;//tager resultatet af sekantmetoden og indsætter som andet gæt.
93  }
94
95
96  break;
97  case 4:
98
99  cout<<"Intet at vise"<<endl;
100
101 break;
102 default:
103 cout << "Indtastet forkert talv" << ae << "rdi" << endl << endl;
104 break;
105 }
106 cout << "Vil du tilbage til startmenu?" << endl;
107 //do-while-løkke, fordi den skal kører mindst én gang
108 //og så indtil man taster j, J, n eller N
109 do
110 {
111 cout << "Tast j, J, n eller N" << endl;
112 cin >> igen;
113 } while (igen!='j' && igen!='J' && igen!='n' && igen!='N');
114 } while (igen=='j' || igen=='J');
115
116
117     system("PAUSE");
118     return 0;
119 }
120
```

Litteraturliste

Bøger:
Introduction to Numerical Computation - analysis and Matlab illustrations" af Lars Eldén, Linde Wittmeyer-Koch og Hans Bruun Nielsen.

"Numerical Analysis Mathematics of scientific computing Third edition" af David Kincaid og Ward Cheney.

Specifikke kilder:

1: "Numerical Analysis Mathematics of scientific computing Third edition" af David Kincaid og Ward Cheney. Side 10.

2: "Numerical Analysis Mathematics of scientific computing Third edition" af David Kincaid og Ward Cheney. Side 83.

3: "Numerical Analysis Mathematics of scientific computing Third edition" af David Kincaid og Ward Cheney. Side 84.

4: "Numerical Analysis Mathematics of scientific computing Third edition" af David Kincaid og Ward Cheney. Side 94.