Studieretningscase Matematik og Programmering –

Newton-Raphsons metode til nulpunkts bestemmelse

Et billede, der indeholder fiskeri

Automatisk genereret beskrivelse

*Model over Newton-Raphsons Metode. MAT A HTX Figur 5.15*

*En opgave på 8.7858 gyldige normalsider*

Indholdsfortegnelse

[Indledning: 2](#_Toc2366380)

[Problemformulering: 2](#_Toc2366382)

[Hovedafsnit Teori: 2](#_Toc2366381)

[Matematik: 2](#_Toc2366382)

[Talfølger og rekursionsligninger: 2](#_Toc2366383)

[Differentiering af funktioner: 2](#_Toc2366383)

[Irrationelle tal: 2](#_Toc2366383)

[Newton-Raphons metode: 6](#_Toc2366386)

[Programmering: 6](#_Toc2366387)

[Variabler: 6](#_Toc2366388)

[Betinget udførelse: 8](#_Toc2366391)

[Funktioner / Metoder: 6](#_Toc2366389)

[Rekursion i programmering: 8](#_Toc2366392)

[Implementering af Newton-Raphsons metode: 9](#_Toc2366393)

[Perspektivering: 2](#_Toc2366381)

[Udvidelse af modellen: 2](#_Toc2366382)

[Konklusion: 2](#_Toc2366382)

[Litteraturliste 9](#_Toc2366394)

[Bilag: 10](#_Toc2366395)

Indledning

Jeg har tit gået og undret mig over hvordan lommeregnere og computere kan tilnærme sig værdier for tal som og . Jeg tænkte om der mon var en fiks måde at udregne disse tal, som man endda med noget snilde kunne gøre i hovedet. Da vi begyndte at lære om Newton-Raphsons metode blev jeg meget fascineret af hvor effektivt og hurtigt man vha. den kunne udregne tilnærmede værdier for de mystiske tal, som jeg personligt så dem for før. Den elegante metode hvorved tangenter en efter en nærmer sig det urørlige nulpunkt i en differentiabel funktion.

Jeg læste om hvordan man vha. Newton-Raphsons metode kunne løse den transcendente ligning . Med min viden indenfor analytisk ligningsløsning var denne ligning uløselig. Lige meget hvilke tricks og kneb man gjorde brug af kunne man aldrig associerer en værdi med . Ligninger som denne skal løses numerisk fx ved brug af metoder som Newton-Raphsons. I denne opgave vil jeg gennemgå den teori og de trin der skal til for at succesfuldt bruge Newton-Raphsons metode i matematik og programmering.

Problemformulering

Hvordan kan man bruge Newton-Raphsons metode til at finde nulpunkter for funktioner og løse ligninger numerisk, og hvordan fungerer den matematiske teori der ligger bag elementerne der indgår i udregningen? Hvordan kan man implementere Newton-Raphsons metode som programmering?

Hovedafsnit Teori

Matematik

Talfølger og rekursionsligninger

Newton-Raphsons metode gør brug af en rekursionsligning. Derfor må vi forstå teorien bag rekursionsligninger før vi kan kaste os ud i at skrive programmer der løser de såkaldte *transcendente* ligninger.

En *talfølge* er en ordnet liste af tal. Et eksempel ses herunder:

De tre prikker i slutningen viser her at talfølgen er uendelig.

Et tal i en talfølge kaldes for et *element*.

Man kan knytte en variabel til talfølger som set her:

Her ses det også at man kan skrive talfølgerne uden de krøllede parenteser.

For at angive et bestemt element i talfølgen bruger man tælletallet

Man vil dog i de fleste tilfælde se det første element betegnet med indeks 0. Det vil da gælde at det første og fjerde element i talfølgen kan skrives som:

For at beskrive talfølger gør man brug af *rekursionsligninger*.

En rekursionsligning kan forstås, som en regel der bestemmer hvordan hvert element i en talfølge skal beregnes. En talfølge kaldes rekursionsligningens løsning, som ikke skal misforstås som rekursionsligningens generelle løsning som er noget andet.

Vi ser igen på talfølgen:

Hvert element i talfølgen følger reglen at det er 2 større end det foregående element.

Talfølgen er dermed løsning til rekursionsligningen:

Denne ligning skal læses som: ”Et element i talfølgen , er lig med det foregående element i talfølgen, , plus 2.”

I rekursionsligninger beregnes enhver værdi, udover *begyndelsesbetingelsen*, med afsæt i en eller flere foregående værdier, som så igen regnes med afsæt i deres foregående værdier.

Denne rekursionsligning beskriver dog ikke blot talfølgen fra før, men også alle andre talfølger der opfylder dens regler. For eksempel:

Hver talfølge er bestemt af sin rekursionsligning, men også af sin *begyndelsesbetingelse*.

Begyndelsesbetingelsen har betegnelsen: . Begyndelsesbetingelsen indeks er derved også

Talfølgen har altså begyndelsesbetingelsen , mens talfølgen har begyndelsesbetingelsen , dog er begge talfølger løsning til rekursionsligningen.

Differentiering af funktioner

I Newton-Raphsons metode indgår der differentiering så naturligvis skal jeg også gennemgå en del af dette emne.

Jeg tager udgangspunkt i ligningen fra indledningen, , jeg skal blot omformulere den til en funktion før jeg kan differentiere den.

Det er også sådan her jeg omformulerer ligningen når jeg skal bruge Newton-Raphsons metode til at finde .

For at differentiere simple funktioner som denne er det tit mere effektivt at gøre brug af regnereglerne for elementære funktioners afledte funktioner end af tretrinsreglen. Til denne funktion bruger jeg disse to regler:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Med disse regneregler finder jeg at:

bliver til ved første regneregel hvor så bare er og bliver til som bliver positiv af det andet minustegn.

Nu har vi altså fundet differentialkvotienten til vores funktion og vi har derved styr på denne del af Newton-Raphsons metode.

Irrationelle tal

Newton-Raphsons metode kan godt bruges til at løse ligninger der ellers også kan løses analytisk, men dens egentlige værdi ligger i evnen til, at numerisk løse transcendente ligninger som , og at tilnærme sig værdier for irrationelle tal som og .

Det er vigtigt at forstå hvad det vil sige at et tal er irrationelt, så man kan forstå værdien i Newton-Raphsons metodens evne til at tilnærme sig en værdi for tallet. Tal som er irrationelle, idet at de ikke kan beskrives med en brøk. Dette kan bevises med et modstridsbevis som i dette tilfælde vil sige at man beviser at ikke kan være rationel og derved må være irrationel.

Jeg vil nu demonstrere modstridsbeviset for at er irrationel:

Vi antager, at der findes et rationelt tal , sådan at:

Det vil sige at der må findes tal , sådan at . Vi kan gyldigt antage at , da .

Vi antager også at brøken ikke kan forkortes.

Det fås altså nu at:

Hvilket vil sige at:

Dette vil sige at er lige, derved følger at også må være lige. Når er lige må det betyde at der findes et helt tal der opfylder:

Indsat på ’s plads i tidligere ligning giver os:

Ved at opløse parentesen får vi så:

Derved bevises på samme måde som før at også må være et lige tal.

Da vi nu ved at både og er lige må deres brøk så nødvendigvis være forkortelig med 2, men dette strider imod vores antagelse om at brøken ikke kan forkortes og derved har jeg bevist at ikke kan være rationel og så derfor må være irrationel.

Dette faktum om irrationelle tal er grunden til at vi må bruge numeriske løsningsmetoder som Newton-Raphsons metode til at tilnærme os decimalværdierne, idet at vi ikke kan skrive dem på brøk.

Newton-Raphsons metode

Med alt vores viden om byggestenene i Newton-Raphsons metode kan vi nu opstille dens rekursionsligning:

Funktionen er her den funktion man vil finde et nulpunkt for. Hvis vi tager vores gennemløbende funktion, , ville rekursionsligningen se sådan her ud:

Udover at vælge min startbetingelse skal jeg i denne metode også vælge mit præcisionstal, , som effektivt bliver til den decimal jeg finder nulpunktsværdien ved. Det vil sige hvis mit er vil jeg finde en tilnærmet værdi for , hvor de fire første decimaler vil være korrekte.

bestemmer hvornår jeg er tilfreds med tilnærmelsen af sådan:

til præcision, når

For nu at bruge Newton-Raphsons metode til numerisk at tilnærme mig værdien for nulpunktet i funktionen , og derved løse ligningen , vælger jeg præcisionstallet og begyndelsesbetingelsen , idet at jeg gætter på at er relativt tæt på den egentlige værdi. For at overholde mit præcisionstal må jeg algoritmisk tjekke efter hver udregning af et element, om forskellen imellem det og det forrige er lavere end mit præcisionstal. Jeg indsætter det hele i ligningen og begynder at danne min talfølge:

til præcision

Dette kan også skrives som:

Afstanden fra til er mindre end

Jeg har nu gennemgået implementering af Newton-Raphsons metoden og derved tilnærmet mig en værdi for løsningen til den transcendente ligning .

Programmering

For at kunne implementere Newton-Raphsons metode i programmering må vi forstå de elementer der indgår i programmet sådan at vi har en samlet forståelse for de mindre funktionerende dele der tilsammen giver os et resultat. Med det sagt er implementeringen af Newton-Raphsons metode nogenlunde simpel i programmering.

Variabler

Variabler i programmering fungerer langt hen ad vejen på samme måde, som de gør i matematik. Man deklarerer variablen med et navn og en værdi. Denne del af processen er fælles for de fleste moderne programmeringssprog. Herefter bliver der forskelle fra sprog til sprog, men jeg ville tage udgangspunkt i JavaScript, idet vi har arbejdet mest med det i timerne.

Variablerne opbevarer data. Denne data har så forskellige former som kaldes datatyper. I JavaScript deklarerer man dem med nøgleordet *let*. Her er et eksempel:

let tal = 2;

*let* er altså nøgleordet der viser at der laves et variabel. *tal* er det navn jeg giver min variabel og så sætter jeg det lige med . Der findes mange forskellige datatyper, men her er fire af de mest vigtige:

let heltal = 5; //Et heltals variabel også kaldet en int eller integer

let tekst = "Hej verden!" //En tekst variabel også kaldet en streng eller string

let falsk = false //Et boolsk udtryk der enten er sandt eller falsk. Tit kaldet en bool

let decimaltal = 0.73909 //En decimaltalsvariabel også kaldet en float. Kan også være over 1

Variabler har også et såkaldt *scope*. Det vil sige hvor de er tilgængelige i koden. Deklarerer man en variabel inden i en metode vil den kun være tilgængelig der. Det kalder man et lokalt variabel. Hvis man så omvendt deklarerer en variabel udenfor nogen metoder, vil det blive en global variabel der er tilgængelig i alle dele af koden.

Betinget udførelse

Betinget udførsel går ud på at kunne afvikle nogle bestemte instruktioner til computeren hvis en betingelse er opfyldt. En simpel konstruktion for betinget udførsel ser sådan her ud:

if(betingelse){

  //Udfør sekvens af betingede instruktioner

}

Nøgleordet *if* er det der signalerer at der sker en betinget udførsel. I parenteserne bagefter følger den betingelse som *if* skal evaluere for om der skal ske en udførelse. Betingelsen skal være noget der enten er sandt eller falsk. Se dette eksempel hvor betingelsen er sand, idet at 10 er større end 5.

if(5 < 10){

  //Udfør sekvens af betingede instruktioner

}

Når betingelsen er sand, vil alle instruktioner og indenfor de to kurvede parenteser blive udført og programmet vil gå videre.

Udover *if* er der også et andet meget vigtigt nøgleord i forbindelse med betinget udførsel. *else* er et nøgleord til at udvide *if* konstruktionen. *else* fungerer sådan at vi kan gøre noget andet hvis vores oprindelige betingelse var falsk.

if(betingelse){

  //Udfør sekvens af betingede instruktioner

}

else if(betingelse2){

  //Sekvens af instruktioner afvikles hvis betingelsen var falsk og betingelse 2 er sand

}

else {

  //Sekvens af instruktioner afvikles hvis betingelse1 og betingelse2 var falsk

}

*else* kan kombineres med *if* for at danne en udvidet konstruktion der kører forskellige sekvenser af instruktioner baseret på evalueringen af flere forskellige betingelser.

Funktioner / Metoder

Funktioner og metoder kan forstås som mindre afsluttede programmer i det større program. Hvis de er brugt korrekt, har de mange fordele der tilsammen gør koden mere forståelig, nemmere at rette fejl i og forøger potentialet i at arbejde flere personer sammen om et fælles program.

function navn(parameter1, parameter2) //Dette kaldes funktionshovedet

{

  //instruktioner - Denne del af funktionen kaldes funktionskroppen

  //...

  //...

}

navn(2, 3) //Dette er et funktionskald, hvor parametrene er henholdsvis 2 og 3

Dette er et eksempel på en funktion. *navn* er blot det man vil kalde sin funktion. Det er dog vigtigt at bruge sigende funktionsnavne så det ikke forværrer læsbarheden af koden. I funktionshovedet er der 2 parametre skrevet ind efter navnet. Disse fungerer som lokale variabler i funktionen hvis værdi bliver defineret når man kalder funktionen. Funktionskroppen indeholder al kode der vil blive kørt når funktionen bliver kaldt. Til sidst, udenfor funktionen, er der det omtalte funktionskald som ændrer kodens flow ved at køre alt koden i funktionen før den går videre til næste linje. I funktionskaldet bliver parameter et og to også defineret som 2 og 3. Denne værdi vil de have indtil funktionen er helt afviklet.

Rekursion i programmering

Rekursion og rekursive funktioner i programmering er en teknik, der elegant løser problemer som ellers ville være svære at programmere ved brug af iterative løsninger som løkker.

En rekursiv metode eller funktion kalder sig selv. Det bliver kaldt det rekursive kald. De implementeres typisk ved brug af kombinationer af *if-*else eller noget der hedder en *switch*, som er et nøgleord der fungerer lidt som en kombination af *if-else*.Rekursive funktioner har også brug for at vide hvornår de skal stoppe, lidt ligesom præcisionstallet. Dette kaldes basistilfældet i programmering som også implementeres med *if*. Her er et eksempel på en rekursiv funktion i JavaScript der kan udregne fakultet:

function Fakultet(n)

{

  if(n === 0) return 1; //Basistilfældet

  else return n\*Fakultet(n-1) //Det rekursive kald

}

I denne funktion har jeg med blot 2 linjer i funktionskroppen kunnet implementere en løsning til udregning af fakultet.

Rekursive funktioner kan med få linjers kode løse relativt komplicerede problemer, men man vil dog opdage at funktioner som denne er mindre egnede til at udregne høje *n* værdier, idet at den først skal udregne hver eneste foregående værdi.

Rekursive funktioner og teknikker er altså fikse til elegant at løse problemer der kan være sværere for iterative implementeringer, men de har så også ulemper der ligger naturligt for enhver rekursiv løsning i programmering og matematik.

Implementering af Newton-Raphsons metode

Her er en simpel implementering af Newton-Raphsons metode til at finde nulpunktet i den matematiske funktion :

function NewtonRaphsonsMetodeNulPunkt(x0, precision)

{

  //Udførsel og indtastning af ligning.

  var xn = x0 - ((x0-cos(x0))/(1+sin(x0)))

  //Basistilfælde:

  if((x0-xn) < precision && (xn-x0) < precision)

  {

    return xn;

  }

  //Det rekursive kald:

  NewtonRaphsonsMetodeNulPunkt(xn, precision)

}

Denne funktion har to parametre. , som er vores ’gætværdi’ og så *precision* som er vores præcisionstal, altså ved hvilken decimal sikkerhed vi er tilfredse med tilnærmelsen. Denne rekursive funktion er en smule anderledes end andre, fordi den udregner næste værdi før den tjekker basistilfældet, som også er lidt specielt. Basistilfældet er nemlig bare hvornår præcisionstallet bliver opfyldt, for når den er opfyldt, har vi fundet den værdi vi vil have. , den næste værdi på talfølgen, udregnes ved implementering af rekursionsligningen for Newton-Raphsons metode. I denne udgave af ligningen har jeg bare differentieret i forvejen og så skrevet det ind, fordi JavaScript og P5JS biblioteket ikke har nogen indbygget differentierings funktion. Den sidste brik i puslespillet er så selve rekursion delen som foregår i bunden af funktionskroppen. Her kaldes funktionen igen, men denne gang kører vi funktionen med vores nyligt udregnede værdi, som matematisk ville være , som så igen finder den næste værdi der bliver . Sådan bliver det ved indtil basistilfældet er fundet og vi har en tilfredsstillende tilnærmet værdi for nulpunktet.

I det mere udvidede program i Bilag 1 har jeg også funktioner der vha. samme metode kan tilnærme sig værdier for pi og kvadratrødder samt en visualisering af hvordan nulpunktet findes for

Perspektivering

Udvidelse af modellen

I denne del af opgaven vil jeg perspektivere hvordan jeg kunne have forbedret og udvidet mit program, hvis jeg havde haft mere tid.

Til at starte med ville jeg få hver af de tre Newton-Raphsons metode funktioner til selv at printe resultatet og holde styr på deres talfølger så jeg ikke behøvede at have så mange funktionskald i *setup*. Dette ville gøres ved at have en parameter der var talfølge datastrukturen, som så samlede værdierne imens den fulgte med som en rekursiv parameter, ved at skrive sig selv ind som parameter i det rekursive kald.

Hvis jeg nu skulle skrive hele programmet fra bunden igen, ville jeg lave det helt anderledes. Jeg ville kun lave en Newton-Raphsons metode funktion som skulle kunne tage imod en hvilken som helst funktion som parameter og så finde nulpunkt for den. Til dette skulle jeg også installere et matematikfunktionsbibliotek for at kunne differentiere vilkårlige funktioner, eller også kunne jeg forsøge at skrive sådan en funktion selv.

Konklusion

I denne opgave har jeg været godt omkring teori fra begge fag der skal til for at implementere Newton-Raphsons metode. Mit eget skrevne program, i bilag 1, kunne sagtens være bedre og jeg havde mange visioner for hvad det kunne være blevet. Den nuværende udgave får dog stadig jobbet klaret, idet at den kan bruge Newton-Raphsons metode til at tilnærme sig værdier for både pi, kvadratrødder og løse ligningen . Med det sagt er Newton-Raphsons metode ikke den eneste måde at løse ligninger numerisk på, men alligevel er den klart værd at kende til. Min hensigt med denne opgave var også at være meget forståelig og forklare al teori sådan at en elev der ikke havde haft om emnet rekursion også ville forstå det. Alt i alt synes jeg at min SRC-opgave er gået godt og jeg har nu endelig en metode til at regne kvadratrødder i hovedet. Det tager bare lidt tid.

Litteraturliste

**MAT A HTX**

Jensen, M., Marthinus, K. & Hansen, B. (2017). MAT A HTX. SYSTIME A/S. <https://mathtxa.systime.dk/> Systime MAT B HTX

**MAT B HTX**

Marthinus, K., Jensen, M., Pedersen, J.S., Pedersen, N.P. & Hansen, P. (2017). MAT B HTX. SYSTIME A/S. <https://matbhtx.systime.dk/>

**P5JS**

McCarthy L.L, (2020). P5JS, programmerings bibliotek, <https://p5js.org/>

Bilag

Bilag 1: Model over Newton-Raphsons Metode. MAT A HTX Figur 5.15Et billede, der indeholder fiskeri

Automatisk genereret beskrivelse

Bilag 2: Koden til implementering af Newton-Raphsons metode:

*Denne kode er udarbejdet af mig selv og min klassekammerat*

/\*Opretter vores array hvori de forskellige tal for hvert trin af Newton Raphsons

metoden sættes ind.\*/

let talfølge = [];

//gemmer resultatet af NR

let result;

//Indstilling der Bestemmer størrelsen af kurven.

let curveSize = 100;

function setup() {

  createCanvas(windowWidth, windowHeight);

  //Sørger for at alle vinkle beregninger der fortages gøres i radianer.

  angleMode(RADIANS);

  /\*Selve funktionen og de betingleser Newton Raphsons metoden skal bruge, samt en

  konsol log af de tal man for i ens array.\*/

  NewtonRaphsonsMetodeNulPunkt(1, 0.0000000001)

  console.log("Tilnærmelse af x i ligningen x = cos(x) er: " + talfølge)

  /\*Tømmer arrayet så det kan printe for kvadratrod 2 istedet for. Skriver

  parameter ind i kvadratrodsfunktionen så den finder for kvadratrod 2.

  Logger resultatet i konsollen\*/

  talfølge.splice(0, talfølge.length)

  NewtonRaphsonsMetodeKvadratrod(2, 2, 0.000000001)

  console.log("Tilnærmelse af kvadratroden af 2 er: " + talfølge)

  /\*Tømmer arrayet så det kan printe for PI istedet for.

  Logger resultatet i konsollen\*/

  talfølge.splice(0, talfølge.length)

  NewtonRaphsonsMetodePI(3, 0.000000001)

  console.log("Tilnærmelse af PI er: " + talfølge)

}

function draw() {

  background(220);

  //Sætter origo til midten af skærmen.

  translate(width/2, height/2)

  //Koordinatsystemets linjer og tekst

  strokeWeight(.5);

  line(0, height, 0, -height);

  line(-width,0, width, 0 );

  fill(0)

  textSize(25);

  text("X", 15, 25)

  text("Y", -25, -18)

  /\*Laver et punkt ved nulpunktet samt tekst med x værdien,

  skaleret efter kurvestørrelsen\*/

  circle(result\*curveSize, 0, 5)

  textSize(15)

  text(result, result\*curveSize, -15)

  //Tegner kurven for funktionen f(x) = x-cos(x)

  noFill();

  strokeWeight(2);

  beginShape();

  for(let i = -width; i < width; i++)

  {

    curveVertex((i\*curveSize),((i - cos(i))\*curveSize))

  }

  endShape();

}

function NewtonRaphsonsMetodeNulPunkt(x0, precision)

{

  //Gemmer de nuværende x0 værdi ind i arrayet.

  talfølge.push(x0)

  //Udførsel og indtastelse af ligning.

  var xn = x0 - ((x0-cos(x0))/(1+sin(x0)))

  /\*Returner og stopper rekursionen hvis vores tal kommer inde for en definerbar

  værdi af det forige tal. Vi tjekker for både positiv og negativ

  sådan at det blot er differensen der betyder noget.\*/

  if((x0-xn) < precision && (xn-x0) < precision)

  {

    result = xn;

    return xn;

  }

  /\*Rekursionsdelen af funktionen. Vi kalder funktionen igen, men med den næste

  værdi på talfølgen\*/

  NewtonRaphsonsMetodeNulPunkt(xn, precision)

}

function NewtonRaphsonsMetodeKvadratrod(x0, a, precision)

{

  //Gemmer de nuværende x0 værdi ind i arrayet.

  talfølge.push(x0)

  //Udførsel

  var xn = x0 - (((sq(x0))-a)/(2\*x0))

  /\*Returner og stopper rekursionen hvis vores tal kommer inde for en definerbar

  værdi af det forige tal. Vi tjekker for både positiv og negativ

  sådan at det blot er differensen der betyder noget.\*/

  if((x0-xn) < precision && (xn-x0) < precision)

  {

    return xn;

  }

  /\*Rekursionsdelen af funktionen. Vi kalder funktionen igen, men med den næste

  værdi på talfølgen\*/

  NewtonRaphsonsMetodeKvadratrod(xn, a, precision)

}

function NewtonRaphsonsMetodePI(x0, precision)

{

  //Gemmer de nuværende x0 værdi ind i arrayet.

  talfølge.push(x0)

  //Udførsel

  var xn = x0 - (sin(x0)/cos(x0))

  /\*Returner og stopper rekursionen hvis vores tal kommer inde for en definerbar

  værdi af det forige tal. Vi tjekker for både positiv og negativ

  sådan at det blot er differensen der betyder noget.\*/

  if((x0-xn) < precision && (xn-x0) < precision)

  {

    return xn;

  }

  /\*Rekursionsdelen af funktionen. Vi kalder funktionen igen, men med den næste

  værdi på talfølgen\*/

  NewtonRaphsonsMetodePI(xn, precision)

}