$Kode : Python \ , \ Dato : 09.09.2019 \ , \ Emne : Fys3150 \ , \ Navn : Erlend Akre$

Projekt 1.

Andre ordens lineære differensiallikninger

Abstract

Hensikten med dette projektet er å få erfaring med å skrive vitenskaplige atrikler, utvikle gode vaner når det kommer til struktur og gjennomføring av projekter generelt. I tillegg ønsker jeg å få god kjenskap til algoritmene og metodene, slik at disse kan brukes siden.

Forsøket er en simulering av potensialet fra en ladning fra en gitt avstand fra ladningen. Jeg har sett på potensialet fra ladningen opptill en meters avstand, for å simulere potensialet har vi brukt Poissons ligning til å finne annenderiverte av potensialet. Deretter brukte jeg annenderivertes diskritisering til lage et linjært ligningsystem.

Ligningssytemet har jeg løst ved hjelp av en agumentert matrise, ved bruk av enten Thomas algoriothmen og en baklengst substutisjon eller ved hjelp av LU Decomposisjon. Så har jeg sett på hvordan potensialet ser ut sammenlignet med en analytsik løsning.

Jeg har sett på hvordan nøyaktigheten til resultatene er avhengige av å ha en liten som mulig stegstørrelse, og hvordan stegstørrelsen påvirker tiden påkrevd til å gjennomføre simulasjonen. Jeg har også sett på hvor effektive de forskjellige algoritmene oppfører seg for forskjellige n, både i tidsforbruk og minne bruk.

Ved siden av simulasjonene og sammenligningen har jeg laget unittester, unit testene er designet slik at de skal finne feil hvis koden ikke fungerer slik den skal. Dette medfører at når koden blir skrevet om og nye funksjoner blir implementert så vil koden beholde funksjonalitet.

Sist men ikke minst har jeg laget et GitHub reposetory som inneholder kildekoden jeg har utviklet i dette projektet, i tillegg til et par håndplukkede resultater som skal forsøke å vise at koden virker som den skal. GitHub reposetoryet skal også vise en ganske detaljert logg av utviklingen av koden. GitHub reposetory finner du her ;

https://github.com/Erlendak/Fys3150_Projekt_1 (https://github.com/Erlendak/Fys3150_Projekt_1)

Hvis du ønsker å clone reposetory kan du bruke denne leken;

https://github.com/Erlendak/Fys3150_Projekt_1.git (https://github.com/Erlendak/Fys3150_Projekt_1.git)

Teori

a)

Jeg bruker Possions ligning til å utrykke det elektrostatisk potensial Φ, fra en gitt ladning jeg uttrykker med ρ, potensialet blir så beskrevet ved ;

$$\nabla^2\Phi = -4\pi\rho(\mathbf{r}).$$

Under disse beregningene av det elektriske potensialet Φ bruker vi kulekordinater, Dette medfører at vi får kulesymmetri mellom det elektriske potensialet Φ og ladningen ρ(r). Dette gir oss ;

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = -4\pi\rho(r)$$

Så substituerer vi $\Phi(r)$ med $\frac{\phi(r)}{r}$ og da får vi ;

$$\frac{d^2\phi(r)}{d^2r} = -4\pi r\rho(\mathbf{r}).$$

Under beregningene videre definerer jeg u som potensialet ϕ, og siden jeg bruker kulekordinatrer og har kulesymetri bruker jeg x som avstanden 'r'. Dette gir oss diffrensial ligningen ;

$$-u\,''(x) = f(x), \qquad x \in (0,1), \qquad u(0) = u(1) = 0$$

Jeg ser på potensialet fra ladningen opptil avstanden x = 1, og definerer Dirichlet grense betingelsene til $u(0) = u(1) = 0$. Jeg definerer den diskre approkismasjonen til u ved hjelp av v_i der de diskrete punktene x_i som er avstanden $i \cdot h$ fra ladningen. Jeg definnerer h til å være stegstørrelsen under approximasjonen, og kan uttrykes som h = $\frac{1}{(n+1)}$. Videre approximerer jeg u'' ved ;

$$-\frac{v_{i+1} + v_{i+1} + 2v_i}{h^2} = f_i\,, \qquad for\ i = 1, \ldots, n$$

For å gjøre det enkelt å lage en augmentert matrise skriver jeg det på formen ;

$$-v_{i+1} + 2v_i - v_{i+1} = h^2 f_i$$

Her setter jeg opp et ligningsystem, der vi får en (i x i) matrise med koeffisentene til v_{i-1}, v_i og v_{i+1} på tridiagonalen som jeg kaller A. Så definerer jeg vektoren v med lengde i som gir meg alle de diskrete verdiene fra v_1 til v_n. Skalarproduktet mellom A og v vil gi meg vektoren $\widetilde{b}$.

Jeg kan så skrive ligningsettet som ;

$$Av = \widetilde{b}$$

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix}, \qquad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ \vdots \\ v_n \end{bmatrix}, \qquad \tilde{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

For å kunne gjøre beregningene trenger jeg en løsning for f(x), altså $-4\pi r\rho(x)$. I dette projektet antar jeg at det kan utttrykkes som f(x) = $100e^{-10x}$. Dette gjør at jeg kan finne en analytsik løsning også ;

$$\frac{d^2u(x)}{d^2x} = 100e^{-10x}$$

$$u(x) = -\int_0^x \int_0^x 100e^{-10x} = -\int_0^x \frac{100}{-10}e^{-10x} + c_1$$

$$-\left(-\frac{10}{-10}e^{-10x} + c_1x + c_2 \right) = c_1x + c_2 - e^{-10x}$$

Så tester vi for grense verdiene, u(0), u(1) = 0 ;

$$u(0) = 0 = c_1 0 + c_2 - e^{-10\cdot 0} = c_2 - 1 \implies c_2 = 1$$
$$u(1) = 0 = c_1 \cdot 1 + 1 - e^{-10\cdot 1} = c_1 + 1 - e^{-10} \implies c_1 = -1 + e^{-10}$$

Dette gir oss utrykket for den analytisk løsningen til potensialet ;

$$\underline{\underline{u(x) = 1 - (1 - e^{-10})x - e^{-10x}}}$$

b)

Tilbake til approksimasjonen, for å kunne løse ligningsystemet kunne jeg brukt Gauss eliminasjon, men siden matrisen A er en tridiagonal matrise, finnes det andre algoritmer som skal gjøre det enklere å løse systemet. Matrise A har kun elementer som er foskjellig fra null ved diagonalen, øvre diagonalen og den nedre diagonalen. Alle elementene i diagonalen er like store og elementene i den øvre- og nedre- diagonalen er like store, dette gjør at vi kan se på dem som konstanter, men i første omgang gjør vi ingen antagleser for elementene langs tridiagonalen.

Altså så starter jeg med å utvikle et system som løser generelle tridiagonale matriser, siden jeg ser på tridiagonal matriser kan jeg behandle de tre diagonalene som 3 separate vektorer a_i, d_i og c_i. Videre utleder jeg hvordan jeg bruker en augmentert matrise, Thomas algoritmen og en baklengst subsstisjon til å løse ligningsystemt ;

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & \dots & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & \dots & \dots & a_{n-1} & b_n \end{bmatrix}$$

$$\left[\begin{array}{cccccc|c} d_1 & c_1 & 0 & \dots & \dots & 0 & b_1 \\ a_1 & d_2 & c_2 & \dots & \dots & 0 & b_2 \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

Når jeg har et ligningsystem på augmentert matrise form kan jeg bruke Gauss eliminasjon til å løse diffrential ligningene. Vi starter med dividere første rad med d_1 ;

$$\left[\begin{array}{cccccc|c} \frac{d_1}{d_1} & \frac{c_1}{d_1} & 0 & \dots & \dots & 0 & \frac{b_1}{d_1} \\ a_1 & d_2 & c_2 & \dots & \dots & 0 & b_2 \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right] \Longrightarrow \left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ a_1 & d_2 & c_2 & \dots & \dots & 0 & b_2 \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

Deretter trekker vi fra rad 2., a_1 multiplisert med rad 1. ;

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ a_1-(a_1\cdot 1) & d_2-(a_1\cdot c_1^{(1)}) & c_2-(a_1\cdot 0) & \dots & \dots & 0 & b_2-(a_1\cdot b_1^{(1)}) \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

Dette kan vi videre skrive som ;

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & d_2^{(1)} & c_2 & \dots & \dots & 0 & b_2^{(1)} \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

For første runde med Gauss eliminasjon får vi at a_0 ikke er definert, $d_1 = 1$, $c_1 = \frac{c_1}{d_1}$ $b_1 = \frac{b_1}{d_1}$. Så må vi passe på at vi har klar gjort neste rad for neste runde, dette betyr at $a_1 = 0$, $d_2 = d_2 - (a_1 \cdot c_1^{new})$, $c_2 = c_2$, $b_2 = b_2 - (a_1 \cdot b_1^{new})$ Så gjentar vi prossessen for neste rad ;

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & \frac{d_2^{(1)}}{d_2^{(1)}} & \frac{c_2}{d_2^{(1)}} & \dots & \dots & 0 & \frac{b_2^{(1)}}{d_2^{(1)}} \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right] \Longrightarrow \left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & a_2 - (a_2 \cdot 1) & d_3 - (a_2 \cdot c_2^{(2)}) & c_3 - (a_2 \cdot 0) & \ddots & \vdots & b_3 - (a_2 \cdot b_2^{(2)}) \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & 0 & d_3^{(2)} & c_3 & \ddots & \vdots & b_3^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & \vdots \\ 0 & 0 & \dots & \dots & a_{n-1} & d_n & b_n \end{array}\right]$$

Her ser jeg at matriser med tridiagonal får pivot elementer på diagonalen og alle elemente til den nedre diagonalen blir nullet ut. Alle elementene på den øvre diagonalen eller på den augmenterte kolonnen må beregnes for førte runde har vi allerede bestemt $c_1^{new} = \frac{c_1}{d_1}$ og $b_1^{new} = \frac{b_1}{d_1}$. For resten må vi ta hensyn til at vi har forberedt raden først, dette gir oss ;

$$c_n^{new} = \frac{c_n}{d_n - (a_{n-1} \cdot c_{n-1}^{new})}, \qquad b_n^{new} = \frac{b_n - (a_{n-1} \cdot b_{n-1}^{new})}{d_n - (a_{n-1} \cdot c_{n-1}^{new})}$$

Da sitter jeg igjen med matrisen ;

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & 0 & 1 & c_3^{(3)} & \ddots & \vdots & b_3^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 1 & c_{n-1}^{(n-1)} & b_{n-1}^{(n-1)} \\ 0 & 0 & \dots & \dots & 0 & 1 & b_n^{(n)} \end{array}\right]$$

Så foretar jeg en baklengs substution ;

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & 0 & 1 & c_3^{(3)} & \ddots & \vdots & b_3^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 1 & c_{n-1}^{(n-1)} - (c_{n-1}^{(n-1)} \cdot 1) & b_{n-1}^{(n-1)} - (c_{n-1}^{(n-1)} \cdot b_n^{(n)}) \\ 0 & 0 & \dots & \dots & 0 & 1 & b_n^{(n)} \end{array}\right]$$

$$\left[\begin{array}{cccccc|c} 1 & c_1^{(1)} & 0 & \dots & \dots & 0 & b_1^{(1)} \\ 0 & 1 & c_2^{(2)} & \dots & \dots & 0 & b_2^{(2)} \\ 0 & 0 & 1 & c_3^{(3)} & \ddots & \vdots & b_3^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 1 & 0 & b_{n-1}^{(n-1\,,\,1)} \\ 0 & 0 & \dots & \dots & 0 & 1 & b_n^{(n)} \end{array}\right]$$

$$\left[\begin{array}{cccccc|c} 1 & 0 & 0 & \dots & \dots & 0 & b_1^{(1\,,\,n-1)} \\ 0 & 1 & 0 & \dots & \dots & 0 & b_2^{(2\,,\,n-2)} \\ 0 & 0 & 1 & 0 & \ddots & \vdots & b_3^{(3\,,\,n-3)} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 1 & 0 & b_{n-1}^{(n-1\,,\,1)} \\ 0 & 0 & \dots & \dots & 0 & 1 & b_n^{(n)} \end{array}\right]$$

Dette gir oss løsningen av ligningsystemet ;

$$V_{(n-1)-i} = b_{(n-1)-i}^{new} - \left(c_{(n-1)-i}^{new} \cdot V_{n-i} \right)$$

Hvor V_n = b_n^{new} og i går fra 0 til n-2.

c)

Så bruker jeg at alle elementene i a_i, d_i og c_i er konstanter og vi får da;

$$c_1^{new} = \frac{c}{d}, \qquad b_1^{new} = \frac{b_1}{d}$$

For resten blir det da slik ;

$$c_n^{new} = \frac{c}{d - (a \cdot c_{n-1}^{new})}, \qquad b_n^{new} = \frac{b_n - (a \cdot b_{n-1}^{new})}{d - (a \cdot c_{n-1}^{new})}$$

d)

I programene implementerer jeg en funkjson som gjør at jeg kan se hvor lang tid hver beregning tok, slik at jeg kan sammenligne hvor lang tid det tar med forkjellige mengder med iterasjoner. Både for en generell matrise og for den spesielle algoritmen.

Jeg implementerer også to funksjoner som skal sammenligne det approksimerte potensiale med det analytiske. Funksjonen jeg kaller _error() finner det største aviket fra den analytiske løsningen. I tillegg implementerer jeg enda en funksjon som jeg kaller error() som finner den relative felien i forhold til hvor stor aviket er og hvor stort potensialet er, og gir meg et logaritmisk svar.

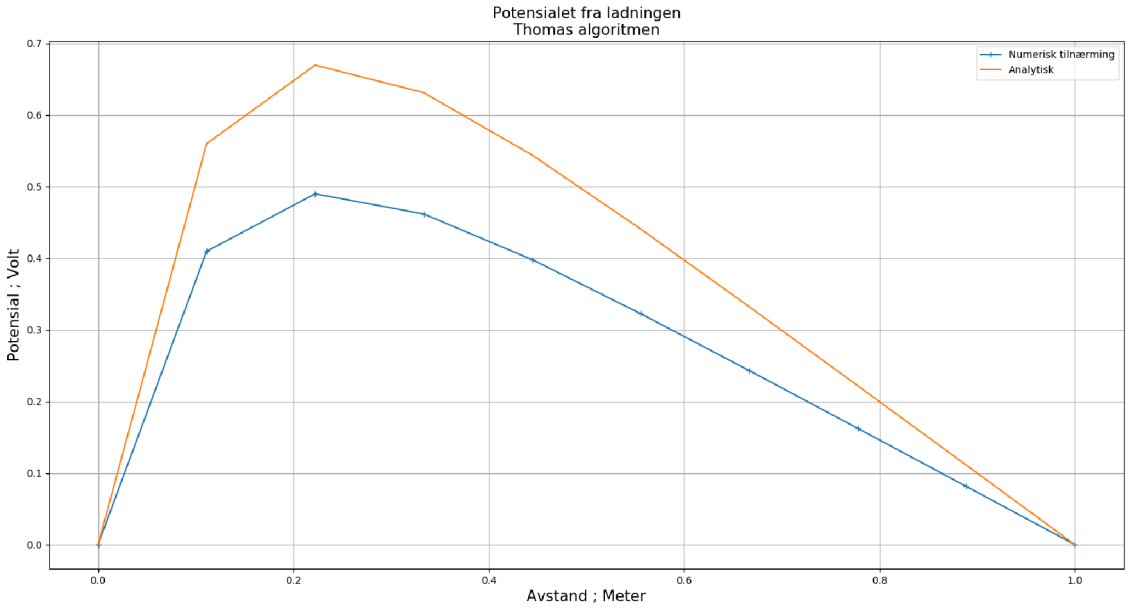
e)

Jeg har også utviklet et program som bruker numpys lu_factor(A) og lu_solve(A,b) til gjøre en LU decomposisjon av matrisen A og så løse den med hensyn på vektor $\widetilde{b}$. I tillegg har jeg utviklet et program som tester alle 3 metodene for ;

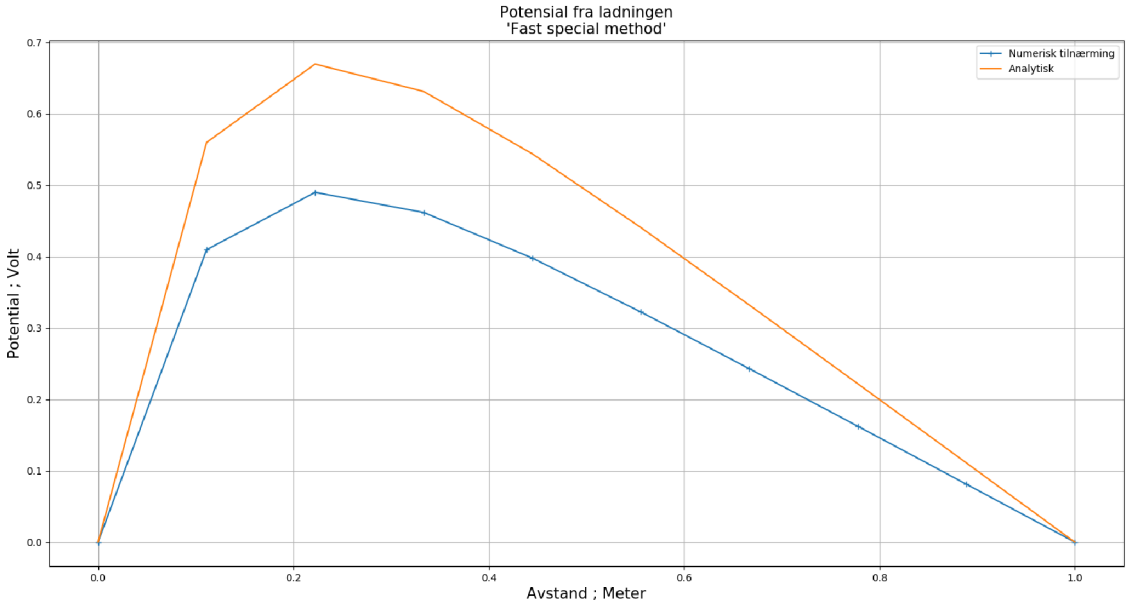
$$n = \Big[10,\ 20,\ 30,\ 50,\ 80,\ 100,\ 150,\ 250,\ 400,\ 600,\ 1 \cdot 10^3,\ 1.500,$$
$$2 \cdot 10^3,\ 4 \cdot 10^3,\ 1 \cdot 10^4,\ 1 \cdot 10^5,\ 1 \cdot 10^6, 1 \cdot 10^7,\ 1 \cdot 10^8\Big]$$

Unntatt LU dekomposisjonen som kun testes opp til 10^4. For så å sammenligne tidsforbruket, relativ nøyaktighet og avvik i potensialet av de forskjellige approksimasjonene.

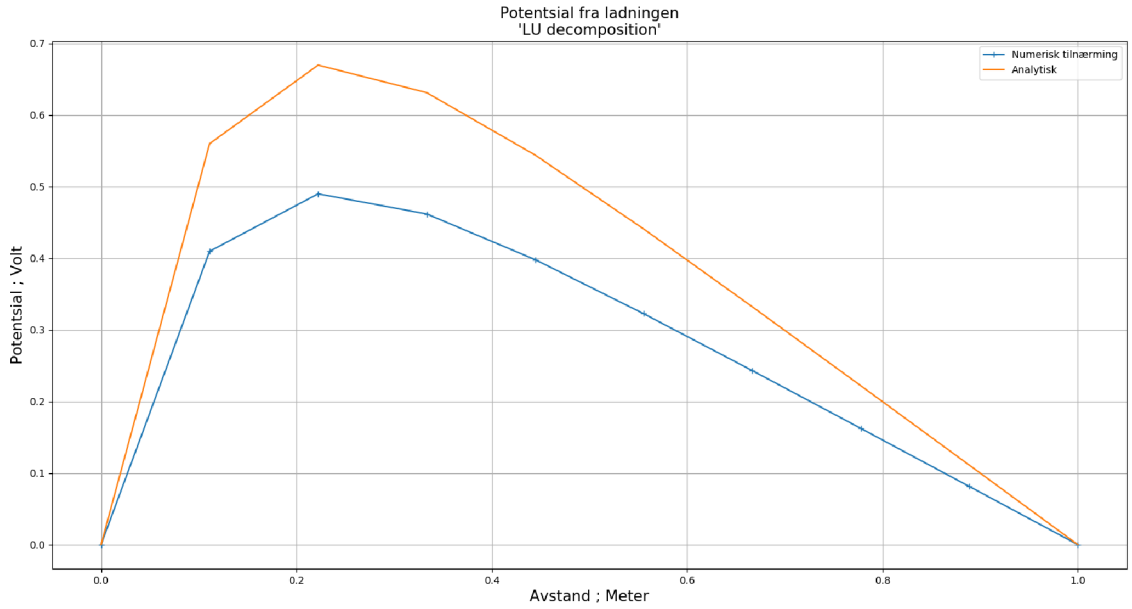
Resultat



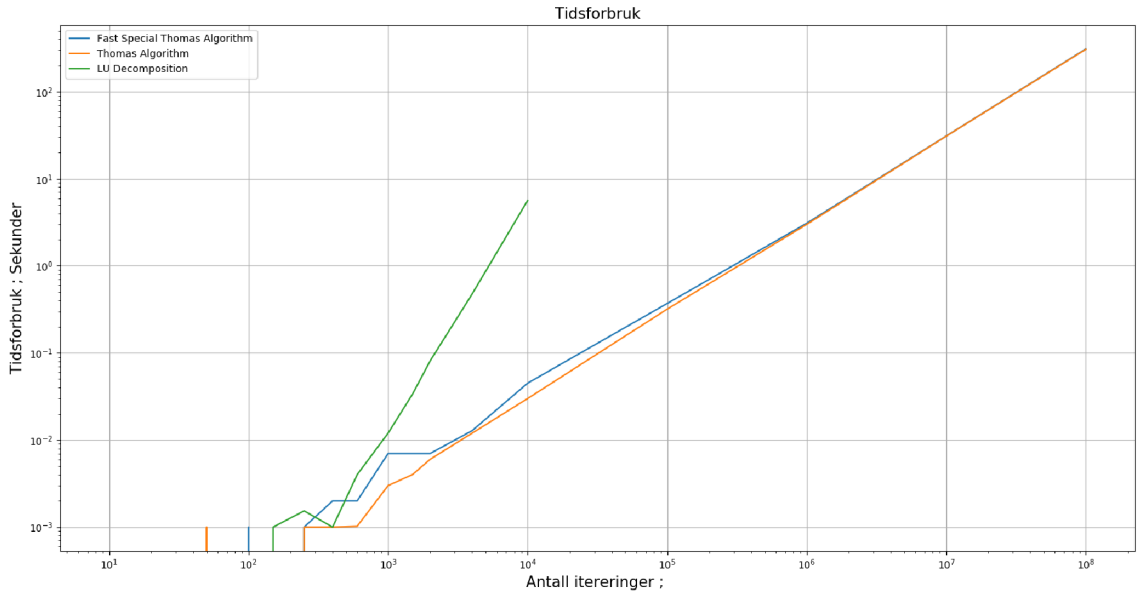
Her har vi approksimasjonen ved bruk av n = 10, ved hjelp av Thomas algoritmen for en generell tridiagonal matrise.



Her har vi approksimasjonen ved bruk av n = 10, ved hjelp av Thomas algoritmen, men der vi bruker konstanter for alle elementene i den nedre diagonalen og diagonalen.

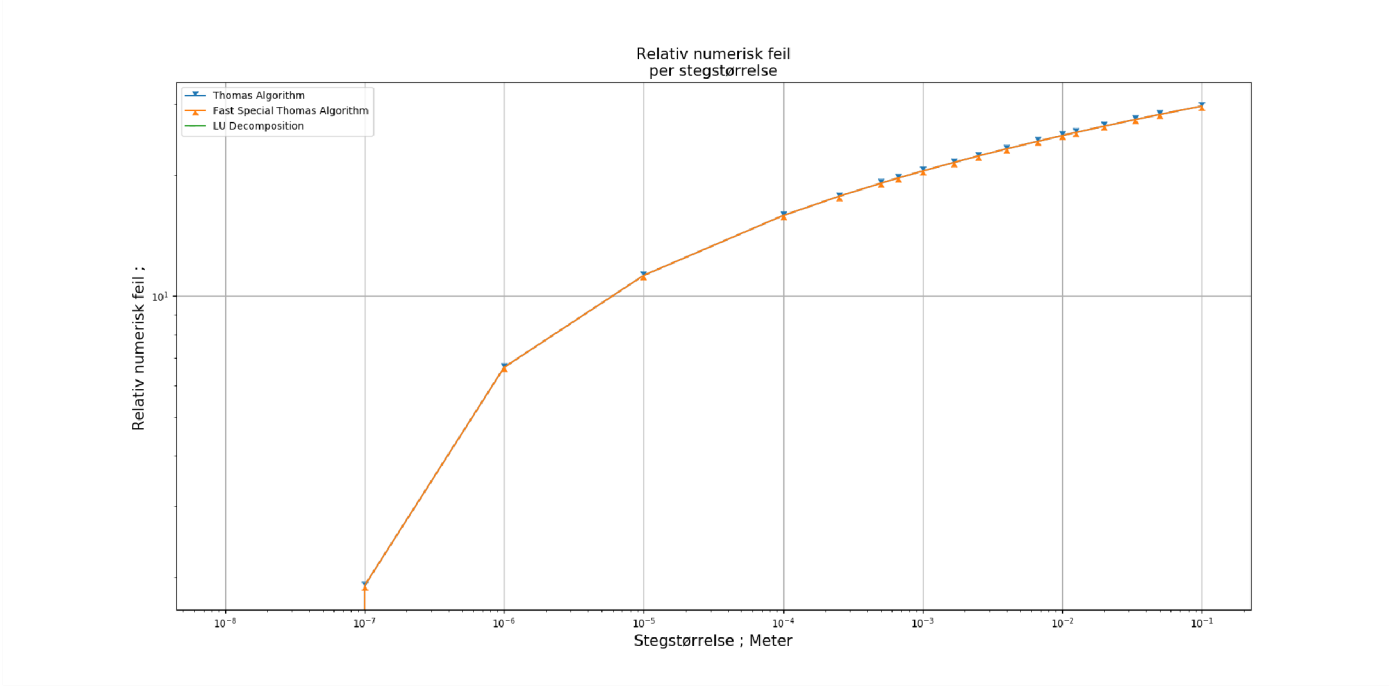


Her har vi approksimasjonen ved bruk av n = 10, ved hjelp av numpys LU dekomposisjon.

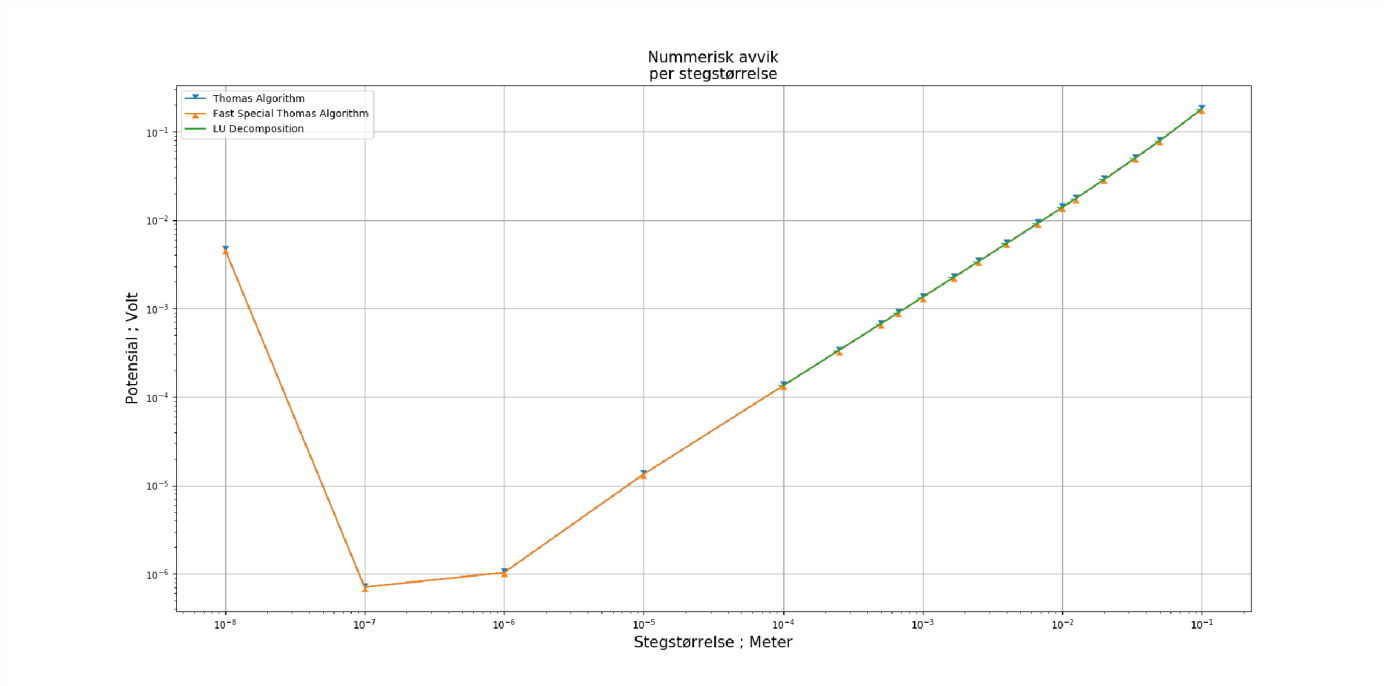


Her ser vi en sammenligning i tidsforburket mellom de forskjellige algoritmene, vi ser at resultatene for n mindre enn 10^2 er ujevne og at den generelle Thomas algoritmen starter som den raskeste av dem, før den spesielle ved n = 10^8 bruker tilnærmet like lang tid som den generelle. LU decomposisjon bruker en del lengere tid enn de andre algoritmene.

Ved 10^8 sjekket jeg manuelt minnebruk, den generelle Thomas algoritmen prossess brukte alene 5 380 MB med minne, men den spessielle brukte kun 3 091 MB med minne. LU Decomposisjon virket kun opp til 10^4 og ble avbrutt av memory error.



Den relative feilen blir mindre, desto mindre stegstørelse som blir brukt, men for stegstørrelsen h = 10^{-8} ser det ut som at den relative feilen ut til å gå mot 10^{∞} = 0.



Avviket i potensialet blir mindre desto mindre stegstørelse som blir brukt, bortsett fra når stegstørelsen passerer 10^{-7}.

Konklusjon

Jeg synes at approksimasjonene ser veldig veldig lovende ut, jeg ser at de har riktige grenseverdier og følger godt etter den analytiske løsningen. De nærmer seg også den analytiske løsningen neden ifra noe som også er svært fornuftig.

Når det kommer til den relative feilen ser det veldig fornuftig ut helt h = 10^{-8}, hvor den relative feilen stuper mot 10^{-8}. Den relative feilen burde egentlig være en bedre måte å måle feil på, siden den vil måle den relative feilen,imotsettning til å måle feilen der utslaget er størst. Det er tydelig på potensial avviket at ved 10^{-8} så

begynner numerisk avrundingsfeil å ødelegge for nøyaktigheten for approksimasjonen, det burde være synlig for den relative feilen, det kan tyde på at den relative feilen er implementert feil.

Den spesielle metoden for tridiagonale matriser bruker mindre minne enn den generelle, slik som forventet. Likvel bruker den spesielle metoden lengere tid enn den generelle, noe jeg ikke hadde forventet, ettersom de mer eller mindre gjør mye av det samme.

Dette kan komme av optimalisering i numpys sine arrays, som gjør det raskere å multiplisere array elementer enn python integer. Som forventet blir tidsforburket mellom dem tilnærmet likt ved n = 10^8. Dette kan komme av at vi ser på en logarittmisk skala og at det er et linjært forhold mellom dem tidsforbruket av de to metodene, men sansynligvis av at "read and write" hastigheten blir avgjørende under store operasjonener. LU dekomposisjonen virket ikke for n større enn 10^4, siden dette er en veldig inneffektiv måte å lagre matrisen på. Som forventet var tidsforbruket mye større enn de to andre metodene.

Dette projektet viser at den generelle metoden er den mest tidseffektive metoden for å løse den augmenterte matrisen, men hvis minne er et problem vil den spesielle metoden være fortetrukket. LU dekomposisjon er en tregere operasjon enn de andre, men har sine fordeler hos andre matriser enn tridiagonale.

Projektet har vært veldig gøy og lærerikt, men det har vært en del å gjøre som har gjort det litt stressende, spesielt om det ikke hadde vært for fantastisk gode lab-timer, forelesninger og poster på Piazza. Personlig hadde det vært greit med mer tid slik at jeg kunne prøvd å gjøre projektet i c++ og teste den relative feilen mer.