UiO : Fysisk institutt
Det matematisk-naturvitenskapelige fakultet

Prosjekt 1

FYS3150 - Computational Physics

Anders Johansson

Innhold

1 Sammendrag

I denne rapporten løses en andreordens differensiallikning på formen $-u''(x) = f(x)$ numerisk via trepunktsmetoden. Feilleddet utledes som $O(h^2)$, og denne h^2-avhengigheten bekreftes eksperimentelt. Algoritmer utvikles for både generelle, tridiagonale matriser og den spesielle matrisen som resulterer fra bruken av trepunktstilnærmingen. Disse algoritmene har tidsbruk proporsjonal med n, i motsetning til den mye brukte teknikken LU-dekomposisjon, hvis tidsbruk er proporsjonal med n^3. Begge de utviklede algoritmene implementeres i C++, og de resulterende funksjonene benyttes for å løse differensiallikningen. Tidsbruken sammenlignes med LU-dekomposisjon, og avhengigheten av h bekreftes.

Alle filene til dette prosjektet finnes på GitHub[1].

[1] https://github.com/anjohan/Offentlig/tree/master/FYS3150/Oblig1

2 Introduksjon

Differensiallikninger er en sentral del av fysikken, ettersom mange av de mest sentrale lovene og likningene (f.eks. Newtons lover og Schrödingerlikningen) er differensiallikninger. Majoriteten av disse likningene er vanskelige eller umulige å løse analytisk, så numeriske løsningsmetoder for differensiallikninger er et viktig verktøy for fysikere.

Differensiallikninger på formen $-u''(x) = f(x)$ dukker ofte opp. Eksempelvis vil Newtons andre lov føre til en slik likning dersom kreftene kun er posisjonsavhengige. Ettersom f kan være alle mulige, kontinuerlige funksjoner, må hvert tilfelle løses separat hvis man ønsker å finne en analytisk løsning - hvis det i det hele tatt er mulig å løse likningen analytisk. En numerisk algoritme bryr seg derimot ikke om hva f er, så en slik algoritme kan brukes på alle mulige problemer som resulterer i en differensiallikning på denne formen.

Det finnes mange numeriske metoder for å løse differensiallikninger. I dette prosjektet benyttes trepunktstilnærmingen til den andrederiverte, som fører til at differensiallikningen kan skrives som et lineært likningssystem. Slike løses generelt med LU-dekomposisjon. Trepunktstilnærmingen fører til en spesiell, tridiagonal matrise, som gjør det gunstig å utvikle en mer spesialisert algoritme som utnytter egenskapene til likningssettet man får.

I dette prosjektet har jeg utviklet to slike algoritmer - én som fungerer for alle, tridiagonale matriser, og én som radreduserer den spesifikke matrisen man får fra trepunktstilnærmingen. Begge disse algoritmene krever et antall regneoperasjoner som øker lineært med antall punkter, i motsetning til LU-dekomposisjonen, som øker kubisk med antall punkter.

Rapporten består hovedsakelig av to deler. Den første delen er teoretisk. Her utledes først trepunktstilnærmingen med feilledd, og et estimat gjøres for hvor stor steglengden bør være for å få minst relativ feil. Estimatet stemmer dessverre ikke med de faktiske resultatene. Deretter utvikles de to løsningsalgoritmene, og antall regneoperasjoner beregnes for hver av dem. I tillegg implementeres de i C++.

Den andre delen består av resultater av anvendelse av de utviklede algoritmene, samt LU-dekomposisjon fra `armadillo`[1] for sammenlikning, på differensiallikningen. Presisjonens avhengighet av steglengden bekreftes, og den estimerte optimale steglengden avkreftes. I tillegg sammenlignes kjøretid både mellom generell og spesiell løsningsalgoritme for tridiagonale matriser, og den spesielle og LU-dekomposisjon. De estimerte steglengdeavhengighetene viser seg å stemme svært godt.

3 Teori og metoder

Differensiallikning:

$$-u''(x) = f(x) = 100\mathrm{e}^{-10x}, \quad x \in [0,1]$$

Initialbetingelser:

$$u(0) = u(1) = 0$$

$u(x) = 1 - \left(1 - \mathrm{e}^{-10}\right)x - \mathrm{e}^{-10x}$ er en eksakt løsning av denne differensiallikningen, men likningen skal i dette prosjektet løses numerisk.

x- og funksjonsverdiene kan diskretiseres ved

$$u_i = u(x_i) = u(i \cdot h) \, \forall \, i \in \{0, 1, \ldots, n+1\}$$

hvor $n+2$ er antall valgte x-verdier og $h = 1/(n+1)$ er avstanden mellom disse verdiene. Definerer også $f_i = f(x_i)$.

3.1 Utledning av trepunktstilnærming til den andrederiverte

Fra Taylors formel:

$$u(x+h) = u(x) + hu'(x) + \tfrac{1}{2}h^2u''(x) + \tfrac{1}{6}h^3u'''(x) + \tfrac{1}{24}h^4u''''(c_1), \; c_1 \in (x, x+h)$$
$$u(x-h) = u(x) - hu'(x) + \tfrac{1}{2}h^2u''(x) - \tfrac{1}{6}h^3u'''(x) + \tfrac{1}{24}h^4u''''(c_2), \; c_2 \in (x-h, x)$$

For liten h og «pen» u kan vi tilnærme $u(c_1) \approx u(c_2) \approx u(x)$. Ved å legge sammen de to likhetene får vi

$$u(x+h) + u(x-h) = 2u(x) + h^2u''(x) + \tfrac{1}{12}h^4u''''(x)$$
$$u''(x) = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} - \tfrac{1}{12}h^2u''''(x) \tag{1}$$

3.2 Anvendelse av trepunktstilnærmingen

Med de diskretiserte x-verdiene og medfølgende notasjon, resulterer problemet i likningssettet

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \, \forall \, i \in \{1, 2, \ldots, n\} \tag{2}$$

hvor v_i er en tilnærming til u_i. Tilfellene $i = 0$ og $i = n+1$ gir liknende likninger, men disse har verdier gitt av initialbetingelsene og behøver derfor ikke å løses. For $i = 1$ og $i = n$ er henholdsvis v_{i-1} og v_{i+1} lik 0 fra initialbetingelsene.

Ved å multiplisere med h^2 på begge sider, kan likning (2) på forrige side skrives ut som

$$\begin{array}{rcrcrcl} & & 2v_1 & - & v_2 & = & h^2 f_1 \\ -v_1 & + & 2v_3 & - & v_1 & = & h^2 f_2 \\ -v_2 & + & 2v_4 & - & v_2 & = & h^2 f_3 \\ \vdots & & \vdots & & \vdots & & \vdots \\ -v_{n-2} & + & 2v_{n-1} & - & v_n & = & h^2 f_{n-1} \\ -v_{n-1} & + & 2v_n & & & = & h^2 f_n \end{array}$$

Dette kan skrives på matriseform:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} h^2 f_1 \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ h^2 f_{n-2} \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix}$$

3.3 Utregning av presisjonsmessig beste n

3.3.1 Matematisk feil

Feilleddet i trepunktstilnærmingen er i likning (1) på forrige side gitt ved

$$\varepsilon_{\mathrm{M}} := \left| -\tfrac{1}{12} h^2 u''''(x) \right| = \tfrac{1}{12} h^2 \left| u''''(x) \right|$$

Med den kjente, eksakte løsningen $u(x) = 1 - \left(1 - \mathrm{e}^{-10}\right)x - \mathrm{e}^{-10x}$ er $u''''(x) = -10^4 \mathrm{e}^{-10x}$. Siden $x \in [0,1]$, er

$$\varepsilon_{\mathrm{M}} \leq \tfrac{1}{12} h^2 \cdot 10^4 = \frac{10^4}{12(n+1)^2}$$

3.3.2 Numerisk feil

I uttrykket

$$\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = ((v_{i+1} - v_i) - (v_i - v_{i-1}))(n+1)^2$$

kan hvert av leddene i den første faktoren gi en feil lik den maksimale, numeriske presisjonen, som er omtrent $\varepsilon_{\mathrm{N}} \approx 10^{-16}$ for `double` i C++.

3.3.3 Total feil

Den totale feilen blir dermed

$$\varepsilon(n) = \frac{10^4}{12(n+1)^2} + 2\varepsilon_{\mathrm{N}}(n+1)^2 \overset{n \gg 1}{\approx} \frac{10^4}{12n^2} + 2\varepsilon_{\mathrm{N}} n^2$$

For å finne minimumet, må denne funksjonen deriveres med hensyn på n:

$$\begin{aligned}
0 &= \frac{\partial \varepsilon}{\partial n} \\
0 &= -\frac{10^4}{6n^3} + 4\varepsilon_{\mathrm{N}} n \\
4\varepsilon_{\mathrm{N}} n &= \frac{10^4}{6n^3} \\
n^4 &= \frac{10^4}{24\varepsilon_{\mathrm{N}}} \\
n &= \frac{10}{\sqrt[4]{24\varepsilon_{\mathrm{N}}}} \\
n &\approx 4{,}5 \cdot 10^4
\end{aligned}$$

3.4 Algoritmer for å løse differensiallikningen

Trepunktsmetoden fører til en matriselikning $A\vec{v} = \vec{d}$ hvor A er en helt spesiell, tridiagonal matrise. Matriselikninger løses vanligvis med LU-dekomposisjon, hvor antall regneoperasjoner går som n^3[2]. Ettersom matrisen for differensiallikningen er på en spesiell form, er det mulig å utvikle algoritmer hvor antallet regneoperasjoner øker lineært med n. Ved å utnytte at A alltid består av samme elementer når man bruker trepunktsformelen (-1, 2 og -1 rundt diagonalen), kan man ytterligere spare regneoperasjoner.

I tillegg krever LU-dekomposisjonen at hele matrisen lagres, dvs. n^2 elementer som hver tar 8 bytes. De spesialiserte algoritmene krever derimot kun at elementene rundt diagonalen lagres, hvilket krever mye mindre minne.

3.4.1 En algoritme for generelle, tridiagonale matriser

Et lineært likningssett med tridiagonal matrise kan skrives på den generelle formen

$$\begin{bmatrix}
b_1 & c_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\
a_1 & b_2 & c_2 & 0 & 0 & \cdots & 0 & 0 \\
0 & a_2 & b_3 & c_3 & 0 & \cdots & 0 & 0 \\
\vdots & & & \ddots & & & & \vdots \\
0 & 0 & \cdots & 0 & a_{n-3} & b_{n-2} & c_{n-2} & 0 \\
0 & 0 & \cdots & 0 & 0 & a_{n-2} & b_{n-1} & c_{n-1} \\
0 & 0 & \cdots & 0 & 0 & 0 & a_{n-1} & b_n
\end{bmatrix}
\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix}
=
\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

For å finne $(v_1, v_2, \ldots, v_n)$ må dette radreduseres. Hvis $a_1 = 0$ kan rad 2 få stå i fred, ellers skal produktet av rad 1 og a_1/b_1 trekkes fra. Dette gir

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & b_2' & c_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & a_{n-3} & b_{n-2} & c_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & \cdots & 0 & 0 & 0 & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

hvor $b_2' = b_2 - c_1 a_1/b_1$ og $d_2' = d_2 - d_1 a_1/b_1$. Hvis $a_2 \neq 0$, må så produktet av rad 2 og b_2'/a_2 trekkes fra rad 3.

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & b_2' & c_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & b_3' & c_3 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & a_{n-3} & b_{n-2} & c_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & \cdots & 0 & 0 & 0 & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' \\ d_3' \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

hvor $b_3' = b_3 - c_2 a_2/b_2'$ og $d_3' = d_3 - d_2' a_2/b_2'$. Denne prosessen kan gjentas for alle rader, hvilket resulterer i

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & b_2' & c_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & b_3' & c_3 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & b_{n-2}' & c_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & b_{n-1}' & c_{n-1} \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & b_n' \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' \\ d_3' \\ \vdots \\ d_{n-2}' \\ d_{n-1}' \\ d_n' \end{bmatrix}$$

hvor

$$b_i' = \begin{cases} b_i - c_{i-1}\dfrac{a_{i-1}}{b_{i-1}'}, & a_{i-1} \neq 0, i \neq 1 \\ b_i, & \text{ellers} \end{cases} \quad \text{og} \quad d_i' = \begin{cases} d_i - d_{i-1}'\dfrac{a_{i-1}}{b_{i-1}'}, & a_{i-1} \neq 0, i \neq 1 \\ d_i, & \text{ellers} \end{cases}$$

Nå kan v_i finnes ved substitusjon:

$$b_n' v_n = d_n' \quad \Longrightarrow \quad v_n = \frac{d_n'}{b_n'}$$

$$b_{n-1}' v_{n-1} + c_{n-1} v_n = d_{n-1}' \Longrightarrow v_{n-1} = \frac{d_{n-1}' - c_{n-1} v_n}{b_{n-1}'}$$

$$b_{n-2}' v_{n-2} + c_{n-2} v_{n-1} = d_{n_1}' \Longrightarrow v_{n-2} = \frac{d_{n-2}' - c_{n-2} v_{n-1}}{b_{n-2}'}$$

Generelt:

$$v_i = \frac{d_i' - c_i v_{i+1}}{b_i'}, \; i \in \{1, 2, \ldots, n-1\}$$

3.4.1.1 Antall regneoperasjoner (med $a_i \neq 0 \forall i$)

For hver av de $n-1$ siste radene må man:

- Regne ut $r := a_{i-1}/b'_{i-1}$
- Multiplisere r med c_{i-1}
- Trekke resultatet fra b_i
- Multiplisere r med d'_{i-1}
- Trekke resultatet fra d_i

Dette gir 5 regneoperasjoner per rad, totalt $5(n-1)$. For å finne v_i, må man for hver av de $n-1$ første radene

- Multiplisere v_{i+1} med c_i.
- Trekke resultatet fra d'_i.
- Dividere med b'_i.

I tillegg må d'_n divideres med b'_n, som gir $3(n-1)+1$ regneoperasjoner.

Hele radreduksjonen krever derfor $5(n-1)+3(n-1)+1 = 8(n-1)+1 = 8n-7$ regneoperasjoner.

3.4.1.2 Implementasjon i C++

```
#include <ctime>

double generell(double* a, double* b, double* c, double* d, double* v, int n){
    /* Parameterne er som beskrevet i algoritmen
     * for generelle, tridiagonale matriser.
     */
    clock_t t1, t2;
    double t, r;
    int i;
    double *b_ny = new double[n];
    double *d_ny = new double[n];
    b_ny[0] = b[0];
    d_ny[0] = d[0];
    t1 = clock();

    //Radreduksjon:
    for(i=1; i<n; i++){
```

```
        if(a[i-1]!=0){
            r = a[i-1]/b_ny[i-1];
            b_ny[i] = b[i]-c[i-1]*r;
            d_ny[i] = d[i]-d_ny[i-1]*r;
        }
        else {
            b_ny[i] = b[i];
            d_ny[i] = d[i];
        }
    }

    //Tilbakesubstitusjon:
    v[n-1] = d_ny[n-1]/b_ny[n-1];
    for(i=n-2; i>=0; i--){
        v[i] = (d_ny[i] - c[i]*v[i+1])/b_ny[i];
    }
    t2 = clock();
    t = ((double) (t2 - t1))/CLOCKS_PER_SEC;
    delete [] b_ny;
    delete [] d_ny;
    return t;
}
```

3.4.2 Algoritme for spesialtilfellet med $a_i = c_i = -1, b_i = 2$

Trepunktstilnærmingen til den andrederiverte resulterer i en helt spesiell, tridiagonal matrise, slik at likningssettet blir seende slik ut:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

Radreduksjon

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{2} & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' = d_2 + \frac{1}{2}d_1 \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{2} & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{4}{3} & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' \\ d_3' = d_3 + \frac{2}{3}d_2' \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{bmatrix}$$

Ved å fortsette slik, ender likningssettet opp som

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \frac{3}{2} & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \frac{4}{3} & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \frac{n-1}{n-2} & -1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \frac{n}{n-1} & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \frac{n+1}{n} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2' \\ d_3' \\ \vdots \\ d_{n-2'} \\ d_{n-1}' \\ d_n' \end{bmatrix}$$

Diagonalelementene, igjen kalt b_i', kan altså regnes ut ved $b_i' = \frac{i+1}{i}$. Høyresideverdiene kan dermed skrives som $d_i' = d_i + \frac{i-1}{i}d_{i-1}' = d_i + \frac{d_{i-1}'}{b_{i-1}'}$.

Substitusjon finner v_i:

$$b_n' v_n = d_n' \quad \Longrightarrow \quad v_n = \frac{d_n'}{b_n'}$$

$$b_{n-1}' v_{n-1} - v_n = d_{n-1}' \Longrightarrow v_{n-1} = \frac{d_{n-1}' + v_n}{b_{n-1}'}$$

$$b_{n-2}' v_{n-2} - v_{n-1} = d_{n-2}' \Longrightarrow v_{n-2} = \frac{d_{n-2}' + v_{n-1}}{b_{n-2}'}$$

Generelt[2]:

$$b_i' v_i - v_{i+1} = d_i' \Longrightarrow v_i = \frac{d_i' + v_{i+1}}{b_i'}$$

3.4.2.1 Antall regneoperasjoner For å finne b_i' og d_i' må man for hver av de $n-1$ siste radene

- Addere i og 1.
- Dele resultatet på i.
- Dele d_{i-1}' på resultatet.
- Legge d_i til resultatet.

[2] d_i'/b_i' inngår også i utregningen av b_i', men antallet regneoperasjoner reduseres ikke ved å lagre dette forholdet og bruke det i $v_i = (d_i'/b_i') + (v_{i+1}/b_i')$

Dette gir 4 regneoperasjoner per rad, totalt $4(n-1)$ regneoperasjoner. For å finne v_i, må man for hver av de $n-1$ første radene

- Legge d_i' sammen med v_{i+1}.
- Dele resultatet på b_i'.

Dette gir 2 regneoperasjoner per rad, totalt $2(n-1)$. I tillegg må man dele d_n' på b_n'.

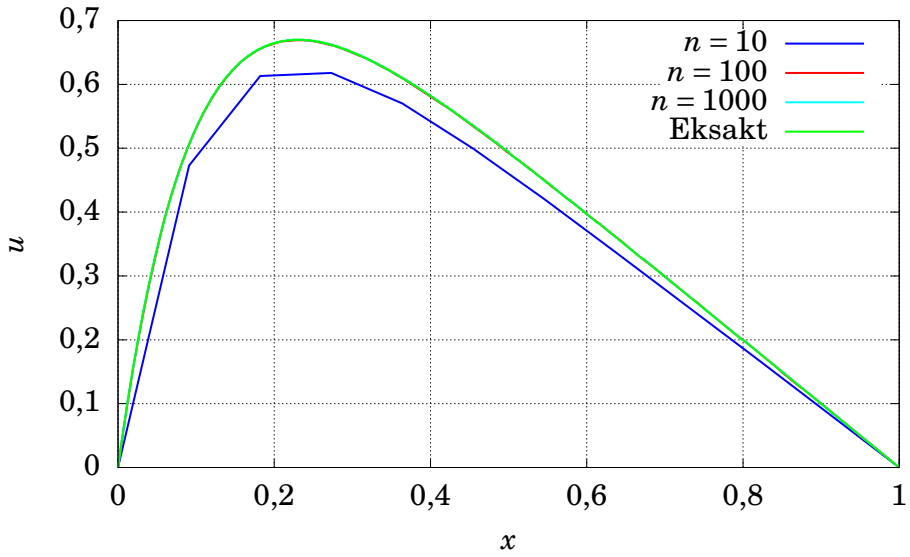
Algoritmen krever derfor $4(n-1)+2(n-1)+1 = 6(n-1)+1 = 6n-5$ regneoperasjoner. Dette er $2(n-1)$ færre enn den generelle algoritmen.

3.4.2.2 Implementasjon i C++

```
#include <ctime>

double spesiell(double* d, double* v, int n){
    /* Parameterne er som beskrevet i algoritmen
     * for generelle, tridiagonale matriser.
     */
    clock_t t1, t2;
    double t;
    double *b_ny = new double[n];
    double *d_ny = new double[n];
    int i;
    t1 = clock();

    //Radreduksjon:
    b_ny[0] = 2;
    d_ny[0] = d[0];
    for(i=1; i<n; i++){
        b_ny[i] = (i+2.0)/(i+1); //C++ indekserer fra 0
        d_ny[i] = d[i] + d_ny[i-1]/b_ny[i-1];
    }

    //Tilbakesubstitusjon:
    v[n-1] = d_ny[n-1]/b_ny[n-1];
    for(i=n-2; i>-1; i--){
        v[i] = (d_ny[i] + v[i+1])/b_ny[i];
    }
    t2 = clock();
    t = ((double) (t2 - t1))/CLOCKS_PER_SEC;
    delete [] d_ny;
    delete [] b_ny;
    return t;
}
```

4 Resultater og diskusjon

4.1 Sammenligning av eksakt løsning med resultater fra løsningsalgoritmen for generelle, tridiagonale matriser



Figur 1: Sammenligning av det eksakte uttrykket med løsninger funnet med algoritmen for generelle, tridiagonale matriser for $n = 10, 100, 1000$. Se avsnitt 6.3.2 på side 18 for programkode.

Figuren viser at den tilnærmede løsningen raskt blir umulig å skille fra den eksakte, hvilket viser at trepunktsformelen er en god tilnærming til den andrederiverte. Se tabell 2 på side 14 for en kvantisering av presisjonen.

4.2 Sammenligning av kjøretid for generell og spesiell løsningsalgoritme

Tabell 1: Sammenlikning av tidsforbruk for generell og spesiell løsningsalgoritme med forskjellige verdier av n. Se avsnitt 6.3.3 på side 19 for programkode.

n	Generell	Spesiell	Forhold
10	$1 \cdot 10^{-6}$ s	0 s	0
100	$3 \cdot 10^{-6}$ s	$2 \cdot 10^{-6}$ s	1,5
1000	$2{,}9 \cdot 10^{-5}$ s	$1{,}8 \cdot 10^{-5}$ s	1,61
10 000	0,000 331 s	0,000 182 s	1,82
100 000	0,003 45 s	0,002 37 s	1,46
1 000 000	0,0306 s	0,0217 s	1,41
10 000 000	0,281 s	0,204 s	1,38
100 000 000	5 s	2,41 s	2,07

Ut fra antallet regneoperasjoner for hver av algoritmene - $8n - 7$ for den generelle og $6n - 5$ for den spesielle - bør tidsbruken øke lineært som funksjon av n, hvilket stemmer godt overens med resultatene for $n \geq 100$. I tillegg bør forholdet i tidsbruk nærme seg $8/6 = 4/3 \approx 1{,}33$ når n øker. Dette samsvarer også forholdsvis godt med resultatene. For større n (f.eks. 10^8) bruker programmet mer minne enn tilgjengelig, hvilket endrer tidsbruken kraftig.

4.3 Den spesielle løsningsalgoritmens presisjon for forskjellige n

Et mål for presisjonen er logaritmen til den relative feilen, dvs.

$$\epsilon_i = \log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right)$$

Tabell 2: Logaritmen av den maksimale, relative feilen for den spesielle løsningsalgoritmen med forskjellige verdier av n. Se avsnitt 6.3.3 på side 19 for programkode.

n	$\max\{\epsilon_i\}$
10	−1,18
100	−3,09
1000	−5,08
10 000	−7,08
100 000	−9,08
1 000 000	−10,2
10 000 000	−9,09
100 000 000	−8,13

Trepunktstilnærmingen til den andrederiverte har et feilledd $O(h^2)$. Dette betyr at når h deles på 10, dvs. når n multipliseres med 10, bør $\epsilon \propto \log_{10}\left(h^2\right) = 2\log_{10}(h)$ avta med 2, siden $\log_{10}(h)$ avtar med 1. Dette stemmer godt for $n \leq 10^5$.

For større n, dvs. mindre h, begynner differansene mellom verdiene i x_i og x_{i+1} å bli så små at den numeriske presisjonen ikke lenger er tilstrekkelig god, slik at den numeriske feilen tar over for den matematiske feilen som dominerende feilkilde. Feilen er derfor økende for $n \geq 10^6$.

Estimatet av 45 000 som beste n-verdi i avsnitt 3.3 på side 5 ser ikke ut til å stemme så godt. Dette kan for eksempel skyldes at den numeriske feilen blir mindre enn antatt, noe som vil føre til større optimal n. I tillegg oppgir tabellen relativ feil, mens estimatet gjelder absolutt feil.

4.4 Sammenligning av kjøretid for LU-dekomposisjon og spesiell løsningsalgoritme

Tabell 3: Sammenlikning av tidsforbruk for LU-dekomposisjon fra `armadillo` og spesiell løsningsalgoritme med forskjellige verdier av n. Se avsnitt 6.3.4 på side 20 for programkode.

n	LU	Spesiell	Forhold
10	0,000 389 s	0 s	0
100	0,000 716 s	$3 \cdot 10^{-6}$ s	239
1000	0,355 s	$1{,}8 \cdot 10^{-5}$ s	$1{,}97 \cdot 10^4$
10 000	447 s	0,000 253 s	$1{,}77 \cdot 10^6$

Antallet regneoperasjoner skal gå som n^3 for LU-dekomposisjon, og n for den spesielle løsningsalgoritmen. Dette betyr at forholdet i tidsbruk bør gå som n^2, hvilket stemmer ganske godt fra

$n = 100$ til $n = 1000$ og veldig godt fra $n = 1000$ til $n = 10\,000$.

LU-dekomposisjonen krever at hele matrisen lagres i minnet, hvilket med `double` krever $8n^2$ bytes. For $n = 10\,000 = 10^4$ gir dette 800 MB, hvilket går greit på de fleste datamaskiner, mens $n = 10^5$ ville kreve 80 GB, noe ingen ikke-supre datamaskiner har tilgjengelig.

5 Konklusjon

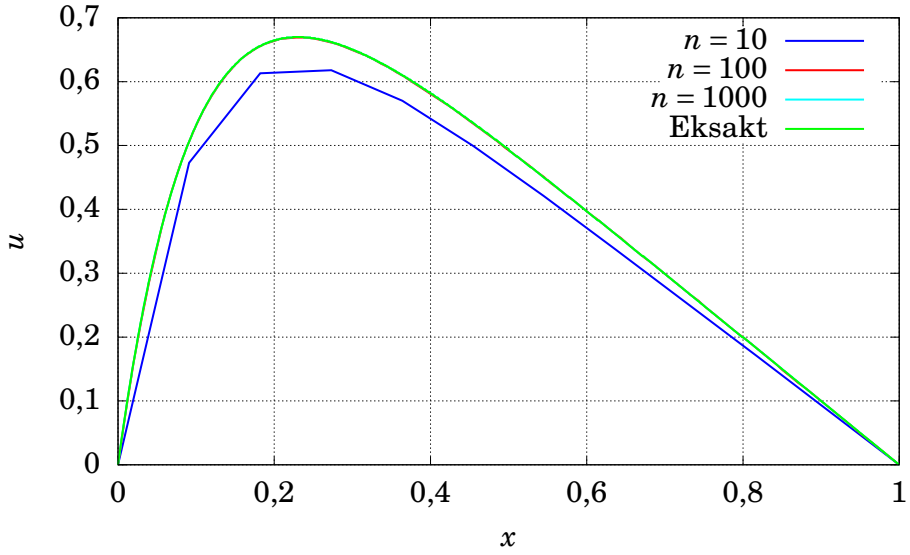
Fra tabell 1 på side 13 og tabell 3 på forrige side er det veldig tydelig at det er store fordeler ved å utvikle algoritmer for spesialtilfeller slik som tridiagonale matriser, i stedet for å alltid bruke generelle algoritmer som for eksempel LU-dekomposisjon.

6 Appendiks

6.1 Spesifikasjoner for datamaskinen benyttet under kjøringer

- Type: HP NoteBook 15
- Minne: 6 GB, samt 6 GB «swap».
- Prosessor: Intel i3-5005U, 2,00 GHz.

6.2 Sammenligning av eksakt løsning med resultater fra løsningsalgoritmen for den spesielle, tridiagonale matrisen



Figur 2: Sammenligning av det eksakte uttrykket med løsninger funnet med algoritmen for spesielle, tridiagonale matriser for $n = 10, 100, 1000$. Se avsnitt 6.3.2 på side 18 for programkode.

Resultatene er de samme som i figur 1 på side 12, ettersom trepunktstilnærmingen til den deriverte benyttes i begge tilfeller.

6.3 Programmer

6.3.1 Nyttige funksjoner

```
#include <cmath>

double f(double x){
    return 100*exp(-10*x);
}

double eksakt(double x){
    return 1 - (1 - exp(-10))*x - exp(-10*x);
}

void lagmatrise(double* a, double* b, double* c, double* d, int n, double h){
    double x;
    int i;
    for(i=0; i < n-1; i++){
        a[i] = -1;
        b[i] = 2;
        c[i] = -1;
        x = (i+1) * h;
        d[i] = h*h*f(x);
    }
    b[n-1] = 2;
    d[n-1] = h*h*f(n*h);
}

double maksimal_relativ_forskjell(double* a, double* b, int n){
    int i; double maks, relativ_forskjell;

    /* Algoritme for aa finne stoerste forskjell:
     * - Anta at det er stoerst forskjell i foerste element.
     * - Gaa gjennom resten, se om noen forskjeller er stoerre.
     */
    maks = fabs((b[0]-a[0])/a[0]);
    for(i=1; i<n; i++){
        relativ_forskjell = fabs((b[i]-a[i])/a[i]);
        maks = relativ_forskjell > maks ? relativ_forskjell : maks;
    }
    return maks;
}
```

6.3.2 Program som genererer figur 1 på side 12 og figur 2 på side 16

```
#include <cstdio>
#include "felles.h"
#include "generell.h"
#include "spesiell.h"

int main(){
    double *a, *b, *c, *d, *v;
    double h, x;
    int n, i;
    char *filnavn;
    FILE* fil;
    for(n=10; n<1001; n*=10){
        h = 1.0/(n+1);
        a = new double[n-1]; c = new double[n-1];
        b = new double[n]; d = new double[n]; v = new double[n+2];

        //Oppsett av variable
        v[0] = 0; v[n+1] = 0;
        lagmatrise(a,b,c,d,n,h);

        //Finn v:
        generell(a,b,c,d,&v[1],n);
        delete [] a; delete [] b; delete [] c;

        //Skriv resultatene til fil:
        filnavn = new char[20];
        sprintf(filnavn,"generell%d.dat",n);
        fil = fopen(filnavn,"w");
        for(i=0; i < n+2; i++){
            fprintf(fil, "%.3f %.3f\n", i*h, v[i]);
        }
        fclose(fil);
        delete [] filnavn;
        //Lag plott av spesiell i tillegg
        spesiell(d,&v[1],n);
        filnavn = new char[20];
        sprintf(filnavn,"spesiell%d.dat",n);
        fil = fopen(filnavn,"w");
        for(i=0; i < n+2; i++){
            fprintf(fil, "%.3f %.3f\n", i*h, v[i]);
        }
        fclose(fil);
        delete [] d; delete [] v; delete [] filnavn;
    }
    // Eksakt
    fil = fopen("bu.dat","w");
    n /= 10;
    for(i=0; i < n+2; i++){
        x = i*h;
        fprintf(fil, "%.3f %.3f\n",x,eksakt(x));
    }
    fclose(fil);
}
```

6.3.3 Program som genererer tabell 1 på side 13 og tabell 2 på side 14

```
#include <cstdio>
#include <cmath>
#include "felles.h"
#include "generell.h"
#include "spesiell.h"

int main(){
    double *a, *b, *c, *d, *v, *u, h, t_generell, t_spesiell, r, epsilon;
    int n, i;
    FILE *fil1 = fopen("c.dat","w");
    FILE *fil2 = fopen("d.dat","w");
    // Skriv ferdige LaTeX-tabeller (krever booktabs-pakken):
    fprintf(fil1,"\\begin{tabular}{rrrr}\\toprule\n\\multicolumn{1}{c}{\\(n\\)} &
 ↪  \\multicolumn{1}{c}{Generell} & \\multicolumn{1}{c}{Spesiell} &
 ↪  \\multicolumn{1}{c}{Forhold}\\\\");
    fprintf(fil2,"\\begin{tabular}{rr}\\toprule\n\\multicolumn{1}{c}{\\(n\\)} &
 ↪  \\multicolumn{1}{c}{\\(\\max\\qty{\\epsilon_i}\\)}\\\\");
    for(n=10; n < 1.1E8; n*=10){
        // Sett opp variable:
        h = 1.0/(n+1);
        a = new double[n-1]; c = new double[n-1];
        b = new double[n]; d = new double[n];
        v = new double[n+2]; u = new double[n+2];

        //Finn tidsbruk:
        lagmatrise(a,b,c,d,n,h);
        t_generell = generell(a,b,c,d,&v[1],n);
        delete [] a; delete [] b; delete [] c;
        t_spesiell = spesiell(d,&v[1],n);
        r =  t_spesiell != 0 ? t_generell/t_spesiell : 0;

        //Finn eksakt svar:
        for(i=1; i < n+1; i++){
            u[i] = eksakt(i*h);
        }

        //Relativ feil:
        epsilon = log10(maksimal_relativ_forskjell(&u[1],&v[1],n));

        //Skriv resultatene til fil:
        fprintf(fil1,"\\midrule\n \\(\\num{%d}\\) & \\(\\SI{%.3g}{\\second}\\) &
 ↪  \\(\\SI{%.3g}{\\second}\\) & \\(\\num{%.3g}\\)\\\\",n,t_generell,t_spesiell,r);
        fprintf(fil2,"\\midrule\n \\(\\num{%d}\\)& \\(\\num{%.3g}\\)\\\\",n,epsilon);

        delete [] d; delete [] v; delete [] u;
    }
    fprintf(fil1,"\\bottomrule\n\\end{tabular}");
    fprintf(fil2,"\\bottomrule\n\\end{tabular}");
    fclose(fil1); fclose(fil2);
}
```

6.3.4 Program som genererer tabell 3 på side 14

```
#include <cstdio>
#include <ctime>
#include <armadillo>
#include "felles.h"
#include "spesiell.h"

using namespace arma;

int main(){
    double *d, *v, h, t_spesiell, t_lu, y, r;
    clock_t t1, t2;
    int i, n;
    mat A; vec d2, v2;
    FILE *fil = fopen("e.dat","w");
    //Skriv ferdige LaTeX-tabeller (krever booktabs-pakken):
    fprintf(fil,"\\begin{tabular}{rrrr}\\toprule\n\\multicolumn{1}{c}{\\(n\\)} &
 ↪  \\multicolumn{1}{c}{LU} & \\multicolumn{1}{c}{Spesiell} &
 ↪  \\multicolumn{1}{c}{Forhold}\\\\");
    for(n=10; n < 10001; n*=10){
        A = zeros<mat>(n,n);
        d2 = zeros<vec>(n);

        v = new double[n+2];
        d = new double[n];
        h = 1.0/(n+1);
        for(i=0; i<n; i++){
            y = h*h*f(i*h);
            d[i] = y;
            d2[i] = y;
        }
        t_spesiell = spesiell(d,&v[1],n);
        delete [] v;

        A.at(0,0) = 2; A.at(0,1) = -1;
        A.at(n-1,n-2) = -1; A.at(n-1,n-1) = 2;
        for(i=1; i < n-1; i++){
            A.at(i,i-1) = -1;
            A.at(i,i) = 2;
            A.at(i,i+1) = -1;
        }
        t1 = clock();
        v2 = solve(A,d2);
        t2 = clock();
        t_lu = ((double) (t2 - t1))/CLOCKS_PER_SEC;
        r =  t_spesiell != 0 ? t_lu/t_spesiell : 0;

        delete [] d;
        fprintf(fil,"\\midrule\n \\(\\num{%d}\\) & \\(\\SI{%.3g}{\\second}\\) &
 ↪  \\(\\SI{%.3g}{\\second}\\) & \\(\\num{%.3g}\\)\\\\",n,t_lu,t_spesiell,r);
    }
    fprintf(fil,"\\bottomrule\n\\end{tabular}");
    fclose(fil);
}
```

6.3.5 Gnuplotprogram som lager figur 1 på side 12 og figur 2 på side 16

```
set terminal epslatex size 5,3 color colortext
set style line 1 lt 1 linecolor rgb "blue" linewidth 3
set style line 2 lt 1 linecolor rgb "red" linewidth 3
set style line 3 lt 1 linecolor rgb "cyan" linewidth 3
set style line 4 lt 1 linecolor rgb "green" linewidth 3

set output "b.tex"
set grid
set decimalsign "${,}$"
set key top right

set xlabel "$x$"
set ylabel "$u$"

plot "generell10.dat" title "$n=10$" with lines ls 1,\
"generell100.dat" title "$n=100$" with lines ls 2,\
"generell1000.dat" title "$n=1000$" with lines ls 3,\
"bu.dat" title "Eksakt" with lines ls 4

set output "b_spesiell.tex"
set grid
set decimalsign "${,}$"
set key top right

set xlabel "$x$"
set ylabel "$u$"

plot "spesiell10.dat" title "$n=10$" with lines ls 1,\
"spesiell100.dat" title "$n=100$" with lines ls 2,\
"spesiell1000.dat" title "$n=1000$" with lines ls 3,\
"bu.dat" title "Eksakt" with lines ls 4
```

Referanser

[1] Conrad Sanderson og Ryan Curtin. “Armadillo: a template-based C++ library for linear algebra”. I: *Journal of Open Source Software* 1.2 (jun. 2016). DOI: 10.21105/joss.00026. URL: http://dx.doi.org/10.21105/joss.00026.

[2] Morten Hjorth-Jensen. *Computational Physics*. Lecture notes. 2015. URL: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.