FYS3150: Prosjekt 1

Saurav Sharma, Delmon Arous

15. september 2013

Sammendrag

Vi diskuterer hvordan lineær algebra kan anvendes til å løse lineære, inhomogene andregrads differensialligninger i C/C++. Her får vi altså redusert differensialligningen til et tridiagonalt system (sett med lineære ligninger) som vi løser numerisk gjennom forover og bakover substitusjon. Plotfremstillinger av numerisk løsning og relativ avvik som funksjon av steglengde gir oss resultater som viser at vår numeriske løsning nærmer seg med minkende steglengde til det spesielle analytiske svaret. Disse resultatene sammenligner vi med resultatdata oppnådd fra LU dekomposisjon. Her viser det seg at LU dekomposisjon bruker mer tid på å løse et tridiagonalt system enn en egendefinert tridiagonal løser. Vi får da også en god innføring i bruk av viktige funksjoner i bibliotekpakker som f.eks. Armadillo, LAPACK og BLAS. Videre blir vi introdusert til håndtering av dynamisk minneallokering av vektorer og matriser utover prosjektet.

Introduksjon

Prosjektets hovedfokus er å gjøre seg kjent med diverse matrise operasjoner, alt fra dynamisk minneallokering til bruk av programmer og funksjoner i aktuelle bibliotekpakker.

I dette prosjektet skal vi se på hvordan et lineært ligningssystem implementeres i C++ ved hjelp av matriser. Her spiller minneallokering et sentralt rolle på hvor effektiv maskinsystemet kan løse ligningssystemet. Dette avhenger av størrelsen på ligningssystemet (dimensjonene av matrisene som implementeres) og tilgjengelighet av RAM i maskinen. Med det skal vi løse de lineære ligningene gjennom ulike metoder, ved å utnytte karakteristikk av matriser, og se hvilken løsningsmetode står frem som mest responsiv. Her tar vi bruk solvere og funksjoner som finnes i Armadillo-biblioteket for sammenligning av de mer generelle metodene, og ser på fordeler og ulemper ved disse.

Videre er antall flyttalsoperasjoner også viktig å få redusert med tanke på å finne en løsning av et lineært ligningssystem på kortest mulig tid.

Teori

I dette prosjektet skal vi betrakte Poissons ligning fra elektromagnetisme. Det elektrostatiske potensialet Φ er generert av en ladningsfordeling $\rho(\mathbf{r})$ som befinner seg i posisjonen $\mathbf{r}$. Poissons ligning i tre dimensjoner lyder,

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

Ved å anta at potensialet Φ og ladningsfordelingen ρ er sfærisk symmetriske, kan ligningen reduseres til en en-dimensjonal ligning i r,

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

som kan omskrives gjennom substitusjonen $\Phi(r) = \phi(r)/r$,

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

Så ved å denotere det inhomogene leddet som f (kilde-leddet), og la $\phi \to u$ og $r \to x$ får vi følgende Poissons ligning i en dimensjon,

$$-u''(x) = f(x) \tag{1}$$

Ligning (1) kan vi løse numerisk ved å approksimere $u''(x)$ gjennom en tre-punkts formel (oppnås gjennom Taylor-utvikling), og innføre en symmetrisk matrise $\mathbf{B}$. Altså,

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2) \tag{2}$$

og

$$\mathbf{B} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \tag{3}$$

der steglengden, gitt ved n antall steg, er $h = \frac{x_{max} - x_{min}}{n}$. Med det stykker vi opp intervallet vårt $x \in [x_{min}, x_{max}]$ i n subintervaller ved å la $x_i = x_{min} + ih$ for $i = 0, ..., n$. Derfor kan vi approksimere Poisson ligningen (1) til,

$$u''(x_i) = -\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} = f_i = f(x_i), \quad i = 1, 2, ..., n-1 \tag{4}$$

Her definerer vi approksimasjonen til u evaluert i x_i som v_i.

Problem

(a)

Vi vil løse den en-dimensjonale Poisson ligningen med Dirichlet grensebetingelser, som er gitt ved Den en-dimensjonale Poisson ligningen med Dirichlet grensebetingelser er,

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

Vi approksimerer ligningen og finner den numerisk ved å løse ligning (4). Med det lyder grensebetingelsene $v_0 = v_n = 0$. Til slutt innfører vi matrisen **B** (3) for å omskrive ligning (4) til et sett av lineære ligninger på formen $\mathbf{Av} = \tilde{\mathbf{b}}$, hvor

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}$$

og $\tilde{b}_i = h^2 f_i$.

Vi antar så at kilde-leddet vårt er $f(x) = 100e^{-10x}$, og holder lik intervall og grensebetingelser som før. Med det finner vi en analytisk løsning for $u(x)$ som kan brukes for videre sammenligninger med vår numeriske løsninger.

(b)

Vi omskriver matrisen **A** til å bestå av vektorene a, b og c av lengde $1 : n$ langs hhv. nedre, midt og øvre diagonal. Da gir ligning $\mathbf{Av} = \tilde{\mathbf{b}}$,

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_2 & b_2 & c_2 & \dots & \dots & \dots \\ & a_3 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{pmatrix} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \dots \\ \dots \\ \dots \\ \tilde{b}_n \end{pmatrix} \tag{5}$$

Dermed kan det tridiagonale systemet (5) uttrykkes som

$$a_i v_{i-1} + b_i v_i + c_i v_{i+1} = \tilde{b}_i, \quad i = 1, 2, ..., n-1 \tag{6}$$

Disse lineære ligningene løser vi gjennom forover og bakover substitusjon ved å sette opp passende algoritmer. Her sammenligner vi antall FLOPS utført med standard gaussisk eliminasjon og LU dekomposisjon.

Til slutt løser vi ligningene (6) for $n = 10, 100, 1000$ mesh punkter gjennom bruk av algoritmene satt opp. De numeriske løsningene vi oppnår sammenligner vi med de tilhørende analytiske løsningene gjennom plotframstilling.

(c)

Vi beregner det relative avviket for $i = 1, 2, ..., n-1$,

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

og ser hvordan det relative avviket forholder seg til $\log_{10}(h)$ for funksjonsverdiene til u_i og v_i. Her tabulerer vi maksimal verdi av ϵ_i for hvert sett med gitterpunkter $n = 10, 100, 1000, 10000, 10^5$.

(d)

Vi sammenligner våre resultater med resultater oppnådd tidligere med resultater ved LU dekomposisjon for matrise av størrelse $n = 10, 100, 1000, 10^5$. Her brukte vi funksjonen "*time*" og kjørte programmet for å sammenligne tidsbruket ved LU dekomposisjon og tridiagonal løsningsmetode. Antall flyttallsoperasjoner blir også vurdert.

(e)

I denne deloppgaven skal vi ta for oss hvordan matriseoperasjoner påvirker maskinens effektivitet. Dette gjør vi ved å allokere minne for matrisene definert og utføre matrise-matrise multiplikasjoner med dimensjoner $N = 10^4$. Vi utfører multiplikasjonen først med en egendefinert metode, for så å bruke funksjonene i BLAS for beregning av multiplikasjonen. Disse to metodene sammenligner vi med hverandre ved å estimere tiden brukt på evaluering av multiplikasjonen.

Matrise-matrise multiplikasjonen er gitt ved $\mathbf{A} = \mathbf{BC}$, hvor matrisene $\mathbf{B}$ og $\mathbf{C}$ er satt opp til å være tilfeldige.

Resultater

(a)

Ved å definere en $n \times n$-matrise, $\mathbf{B}$,

$$\mathbf{B} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}$$

og vektorene $\mathbf{v} = (v_0, v_1, ..., v_n)^T$ og $\mathbf{f} = (f_1, f_2, ..., f_n)^T$ kan vi omskrive ligning (4) til et sett av lineære ligninger,

$$\mathbf{Bv} = \mathbf{f}$$

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ \vdots \\ f_n \end{pmatrix}$$

Multipliserer begge sider med h^2 gir,

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = h^2\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ \vdots \\ f_n \end{pmatrix}$$

$$\Rightarrow \mathbf{Av} = \tilde{\mathbf{b}}$$

Antar nå at kilde-leddet er $f(x) = 100e^{-10x}$, får vi fra ligning (1),

$$\begin{aligned} u''(x) &= -f(x) = -100e^{-10x} \\ u'(x) &= 10e^{-10x} + C_1 \\ u(x) &= -e^{-10x} + C_1 x + C_2 \end{aligned}$$

der grensebetingelsene $u(0) = u(1) = 0$ gir $C_1 = e^{-10} - 1$ og $C_2 = 1$. Derfor er den analytiske løsningen

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

(b)

Ved å omskrive matrisen **A** til en tridiagonal matrise gjennom vektorene **a**, **b** og **c**,

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_2 & b_2 & c_2 & \dots & \dots & \dots \\ & a_3 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix}$$

kan de ligning (4) utrykkes ved ligning (6). Her kan vi sette $a_1 = c_n = 0$, siden både v_0 og v_{n+1} er ikke inkludert i de lineære ligningene $\mathbf{Av} = \tilde{\mathbf{b}}$.

Algoritmen for forover substitusjon på ligningene (6) er gitt ved,

```
for(i = 2; i <=n; i++) {
    m = a[i]/b[i-1]; // line a
    b[i] -= m*c[i-1]; // line b
    b_tilde[i] -= m*b_tilde[i-1]; // line c
}
```

også tilsvarende bakover substitusjon er gitt ved,

```
v[n] = b_tilde[n]/b[n]; // line d
for(i = n-1; i >= 1; i--) {
v[i] = (b_tilde[i] - c[i]*v[i+1])/b[i] // line e
}
```

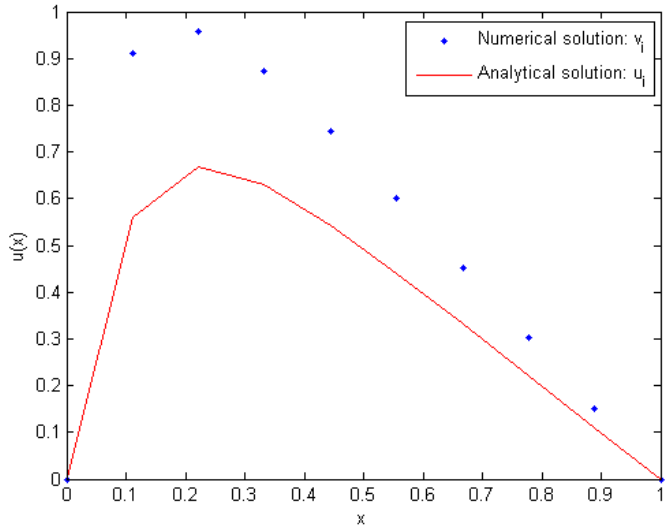
Sammenligning av numerisk og analytisk løsninger gjennom plot for $n = 10, 100, 1000$ mesh punkter er vist i figurene 1, 2 og 3.

(c)

Maksimal beregnet relativ avvik i settet for n er gitt i tabell 1. Tabellen gir oss $\log_{10}$ verdiene for det relative avviket og steglengden h for funksjonsverdiene u_i og v_i.

Tabell 1: Tabell over $\log_{10}$ verdier for det relative avviket ϵ_i og steglengden h, evaluert ved x_i som gir maks ϵ_i.

n	x_i	$\log_{10}(h)$	ϵ_i
10	0.8889	-1.04	-0.4455
100	0.9899	-2.00	-1.0745
1000	0.9899	-3.00	-2.0484
10000	0.9999	-4.00	-3.0458
10^5	0.9999	-5.00	-4.0456



Figur 1: Analytisk løsning sammenlignet med numerisk løsning ved bruk av $n = 10$ gitterpunkter.

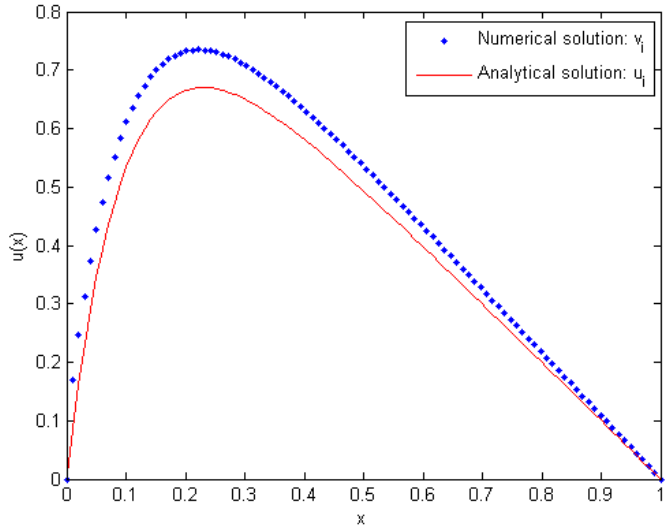
(d)

Tabell 2: Tabell over tidsbruk og antall flyttallsoperasjoner på å løse et system med n ligninger, og dette ved bruk av en tridiagonal løser og LU dekomposisjon.

	Tridiagonal løser		LU dekomposisjon	
n	Tid [s]	FLOPS	Tid [s]	FLOPS
10	0.0	9×10	0.002	$\simeq 6.6 \times 10^2$
100	0.0	9×10^2	0.007	$\simeq 6.6 \times 10^5$
1000	0.0	9×10^3	0.171	$\simeq 6.6 \times 10^8$
10^5	0.011	9×10^5	Error	Error

(e)

Vi bestemmer multiplikasjonen mellom matrisene til å være av dimensjon $N = 2000$, og ikke $N = 10^4$. Med det fikk vi estimert tid gjennom vanlig metode på 48 *sekunder*, mens gjennom funksjonene i BLAS ble tiden estimert til 8 *sekunder*.



Figur 2: Analytisk løsning sammenlignet med numerisk løsning ved bruk av $n = 100$ gitterpunkter.

Diskusjon

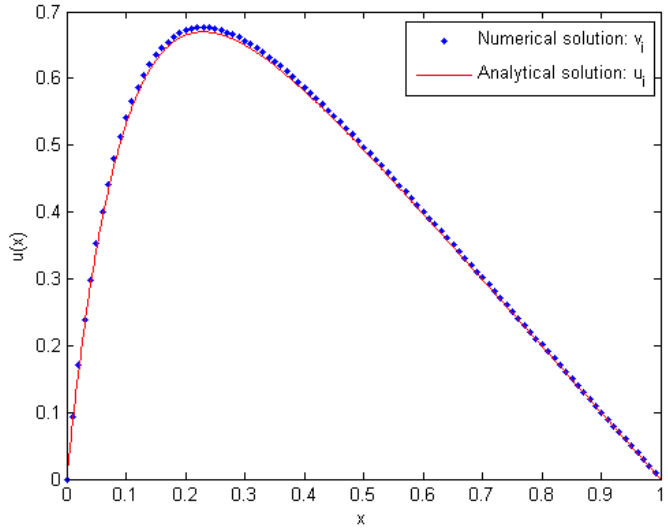
(a)

I denne deloppgaven har vi vist at approksimasjon av den en-dimensjonale Poisson ligningen gjennom tre-punkts formelen (2) er mulig. Vi kunne evt. ha brukt en fem-punkts formel istedenfor sistnevnt for å øke vår presisjon.

(b)

I denne deloppgaven har vi løst det tridiagonale systemet (6) gjennom bruk av algoritmene for forover og bakover substitusjon. Forover substitusjon eliminerer a_i'ene, mens bakover substitusjon produserer løsningen. Så i **linje a** definerer vi en multiplikasjonsfaktor m for ligning $(i-1)$, også trekker vi dette resultatet fra ligning i for å eliminere a_i'ene i ligningene nedover. Dette viser **linje b** i algoritmen. Tilsvarende blir også høyresiden av ligning $(i-1)$ multiplisert med m, for så trukket fra høyresiden av ligning i. Dette er **linje c**. I **linje d** løser vi for v_n i den siste ligningen, ligning n, og jobber oss bakover gjennom loopen.

Videre sammenligner vi antall flyttallsoperasjoner (FLOPS) mellom løsningsmetodene på et tridiagonalt system, gaussisk eliminasjon og LU dekomposisjon. I vårt tilfelle ble løsningen oppnådd med $8n$ FLOPS, mens gaussisk eliminasjon krever et større antall operasjoner. Her krever forover og bakover eliminasjon hhv. $2n^3/3 + O(n^2)$ og $n^2 + O(n)$, altså totalt antall



Figur 3: Analytisk løsning sammenlignet med numerisk løsning ved bruk av $n = 1000$ gitterpunkter.

FLOPS er $2n^3/3 + O(n^2)$ for gaussisk eliminasjon. For LU dekomposisjon kan man finne en løsning med $O(n^2)$ FLOPS dersom vi allerede har en LU dekomponert matrise. Om ikke matrisen er allerede LU dekomponert, så trengs det $O(n^3)$ FLOPS å beregne matrisen.

Til slutt sammenlignet vi de analystiske og numeriske løsningene med hverandre gjennom plot av forskjellige steglengder h. Figur 1 viser at resultater med $n = 10$ mesh punkter. Her avviker den numeriske løsningen fra den analytiske, så mindre steglengde h trengs for å få høyere nøyaktighet. Figur 2 viser resultater med $n = 100$, og den numeriske løsningen legger seg mye nærmere det analytiske svaret. Med $n = 1000$ så overlapper nesten den numeriske og den analytiske løsningen med hverandre, som vist i figur 3. Herfra trenger vi å evaluere det relative avviket for grundigere analyse.

(c)

I denne deloppgaven beregnet vi det relative avviket for $i = 1, 2, ..., n$ for et sett med gitt antall gitterpunkter n. Fra hvert sett ble maksimal relativ avvik trukket ut og tabulert i tabell 1. Fra tabell 1 ser vi at det er en lineær sammenheng mellom $\log_{10}$ verdiene for det relative avviket og steglengden h. Tabellen viser at det relative avviket minker omtrentlig med et bestemt forhold med mindre $\log_{10}(h)$. Med $n = 10^5$ har vi stor nok nøyaktighet, så det skal være unødvendig å øke antall gitterpunkter n etter det. En konsekvens med å øke n etter 10^5, er at det relative avviket kan bli større grunnet

numerisk presisjons- og trunkeringsavvik blir større med mindre steglengde. Med det overskriver vi maskinpresisjonen.

(d)

Ved å utføre beregningene ved hjelp av de to forskjellige måtene kan vi se at vår egendefinerte metode, tridiagonal løseren, er kjappere og krever også mindre utregning enn det biblioteklløseren krever. Dette er fordi vi har laget en løser som spesialiserer seg på akkurat tridiagonale matriser, og dermed unngår vi "overflødige" flyttallsoperasjoner. fungerer bare på disse ved . Hadde vi hatt en generell matrise ville det ikke være mulig å løse det med vår løser. Dette på grunn av løseren vår bruker mindre tid, færre utregninger og mindre minne for en matrise som er av samme dimensjon. Det betyr i gjengjeld at vi kan ha større matriser med vår metode.

(e)

Vi gjennomførte matrise-matrise multiplikasjoner på egendefinert, tung måte og gjennom bruk av funksjonene i BLAS. Her viser resultater at tiden utført gjennom BLAS er seks ganger mindre enn tiden utført med vanlig metode. Dette indikerer at Armadillos bibliotek av lineær algebraiske funksjoner er mindre belastende på maskinsystemet og mer effektive. Her brukte vi heller matriser av dimensjoner $N = 2000$ enn $N = 10^4$, med tanke på tidsbruket ved bruk av systemminnet. For $N = 10^4$ hvor hvert element i de tre matrisene er av typen som gir dobbel presisjon, har vi $3 \cdot (10^4 \times 10^4 \cdot 16\,bits) = 4.8\,GB$ av systemminnet blir brukt. Dette er i overkanten av det vi har tilgjengelig. Derfor reduserte vi dimensjonene betraktelig for å demonstrere oppgaven.

Konklusjon

I dette prosjektet har vi løst en inhomogen, andreordens differensialligning numerisk ved bruk av lineær algebra. Vi approksimerte differensialligningen, gitt av Poisson-ligningen med Dirichlet grensebetingelser, ved tre-punkts formelen for den andrederiverte. Ved så å innføre en tridiagonal matrise, var vi i stand til å unngå unødvendige flyttallsoperasjoner for å finne en numerisk løsning av problemet. Dette gjorde det mulig å finne en approksimasjon av den spesielle analytiske løsningen, der vår numeriske løsning nærmet seg med minkende steglengde. Her fant vi en lineær oppførsel mellom det maksimale relative avviket og steglengden. Til sammenligning, løste vi Poisson-ligningen med LU dekomposisjon og fant ut at denne løsningsmetoden brukte lengere tid enn vår tridiagonale løser. Dette skyldes rett og slett at vi hadde i utgangspunktet en ligning for et tridiagonalt system.

Til slutt tok vi for oss matrise-matrise multiplikasjoner gjennom en egendefinert algoritme og gjennom funksjoner i BLAS. Tidsbruket på hver av disse gjennomføringene ble sammenlignet, og funksjoner i BLAS stod frem som mer effektive og mindre belastende med antall flyttallsoperasjoner på maskinsystemet.

Link til programmer finnes her.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include "time.h"
#include "armadillo"

using namespace std;
using namespace arma;

void initialize(int &n);
void initialize_vectors(int n, double *x, double *h_step, double *a, ↩
    double *b, double *c, double *b_tilde, vec &b_tilde_lu);
void fw_bw_subst(int n, double *a, double *b, double *c, double *v, ↩
    double *b_tilde);
void output(int n, double *x, double *h_step, double *v);

ofstream outfile1, outfile2;

int main() {

    // Declare variables
    int n;
    double *x, *h_step, *a, *b, *c, *v, *b_tilde;
    clock_t start1, start2, finish1, finish2; // declare start and final ↩
        time

    // Read in output file, abort if there are too few command-line ↩
        arguments
    outfile1.open("data_plot.txt", ios::out);
    outfile2.open("data.txt", ios::out);

    // Read in input data from screen
    initialize(n);

    // Allocate space in memory for the one-dimensional arrays h_step and↩
         computed_derivative
    x = new double[n];
    h_step = new double[n];
    a = new double[n];
    b = new double[n];
    c = new double[n];
    v = new double[n];
    b_tilde = new double[n];
    vec b_tilde_lu(n);

    // Initialize vectors needed
    initialize_vectors(n, x, h_step, a, b, c, b_tilde, b_tilde_lu);

    // Commence forward substitution, and then backward substitution
    start1 = clock();
    fw_bw_subst(n, a, b, c, v, b_tilde);
    finish1 = clock();

    cout << "Number of FLOPS for selfmade tridiagonal solver: " << 9*n <<↩
         endl;
    cout << "Execution time for tridiagonal solver: " << ( (finish1 - ↩
```

```
        start1)/float(CLOCKS_PER_SEC) ) << " s" <<endl;

    // LU decomposition initialization
    mat A(n,n), L(n,n), U(n,n);
    vec v_lu(n), y(n), off_diag(n-1);

    // Defining the tridiagonal matrix
    off_diag.fill(-1); // off-diagonal elements are defined in the vector↩
         off_diag of length (n-1)
    A = 2.0*eye(n,n); // diagonal elements of the tridiagonal matrix A ↩
        are defines
    A.diag(1)= off_diag; A.diag(-1)= off_diag; // elements of off_diag ↩
        are placed in the sub- and upper diagonal in A
    A(0,0) = 1.0; A(n-1,n-1) = 1.0; // taking boundary conditions into ↩
        account by "expanding" A
    A(0,1) = 0.0; A(1,0)= 0.0; A(n-1,n-2)= 0.0; A(n-2,n-1) = 0.0;

    // LU decomposition
    start2 = clock();
    lu(L, U, A);
    y = inv(L)*b_tilde_lu;
    v_lu = inv(U)*y;
    finish2 = clock();

    cout << "Number of FLOPS for selfmade tridiagonal solver: " << floor↩
        (2.0/3.0*pow(n,3)) << endl;
    cout << "Execution time for LU decomposition: " << ( (finish2 - ↩
        start2)/float(CLOCKS_PER_SEC)) << " s" << endl;

    // Print out results to file
    output(n, x, h_step, v);

    // Clear memory
    delete [] x;
    delete [] h_step;
    delete [] a;
    delete [] b;
    delete [] c;
    delete [] v;
    delete [] b_tilde;

    outfile1.close();
    outfile2.close();
    return 0;
}

void initialize(int &n) {
    cout << "Read in from screen:" << endl;
    cout << "\n Number of grid points: n = "; cin >> n;
    return;
}

void initialize_vectors(int n, double *x, double *h_step, double *a, ↩
    double *b, double *c, double *b_tilde, vec &b_tilde_lu) {
    double h = 1.0 / (n-1.0);
    for(int i = 0; i < n; i++) {
        h_step[i] = h;
        x[i] = i*h;
        a[i] = -1;
        b[i] = 2;
        c[i] = -1;
        b_tilde[i] = h*h*100*exp(-10*x[i]);
```

```
        b_tilde_lu(i) = b_tilde[i];
    }
    b_tilde[0]=0.0; b_tilde[n-1]=0.0;
    b_tilde_lu(0)=0.0; b_tilde_lu(n-1)=0.0;
    a[0] = 0; c[n-1] = 0;
    return;
}

void fw_bw_subst(int n, double *a, double *b, double *c, double *v, ↩
    double *b_tilde) {
    double m;
    // Forward substitution
    for(int i = 1; i < n-1; i++) {
            m = a[i]/b[i-1]; // line a
            b[i] -= m*c[i-1]; // line b
            b_tilde[i] -= m*b_tilde[i-1]; // line c
        }
    // Backward substitution
    v[n-1] = b_tilde[n-1]/b[n-1]; // line d
    for(int i = n-2; i > 0; i--) {
        v[i] = (b_tilde[i] - c[i]*v[i+1])/b[i]; // line e
    }
    v[0] = 0; v[n-1] = 0;
    return;
}

void output(int n, double *x, double *h_step, double *v) {
    double u, x_temp, temp_error, rel_error = 0;
    outfile1 << setiosflags(ios::showpoint | ios::uppercase);
    outfile2 << setiosflags(ios::showpoint | ios::uppercase);

    for(int i = 1; i < n; i++) {
        u = 1.0-(1.0-exp(-10.0))*x[i]-exp(-10.0*x[i]);
        temp_error = log10(fabs( (v[i] - u)/u ));
        outfile1 << setw(15) << setprecision(8) << x[i];
        outfile1 << setw(15) << setprecision(8) << v[i];
        outfile1 << setw(15) << setprecision(8) << u << endl;
        outfile2 << setw(15) << setprecision(8) << log10(h_step[i]);
        outfile2 << setw(15) << setprecision(8) << temp_error << endl;
        if (fabs(temp_error) > fabs(rel_error)) {
            rel_error = temp_error; // update rel_error when temp_error ↩
                is larger than rel_error
            x_temp = x[i]; // saves the position x_i for the update
        }
    }
    cout << "Maximum relative error: " << rel_error << " for x = " << ↩
        x_temp << endl;
    return;
}
```

```
#include <iostream>
#include <cstdlib>
#include "time.h"
#include "armadillo"

using namespace std;
using namespace arma;

int main() {
```

```
int N = 2000; // dimension of matrices
clock_t start1, start2, finish1, finish2; // declare start and final ↩
    time

// Memory allocation of matrices
double **A, **B, **C;

A = new double*[N];
B = new double*[N];
C = new double*[N];

for(int i = 0; i < N; i++) {
    A[i] = new double[N];
    B[i] = new double[N];
    C[i] = new double[N];
}

// Initialization of matrices
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        A[i][j] = 0.0; // initialize all elements to zero
        B[i][j] = ((double) rand() / (RAND_MAX + 1)); // initialize all ↩
            elements to be random
        C[i][j] = ((double) rand() / (RAND_MAX + 1)); // initialize all ↩
            elements to be random
    }
}

// Matrix-matrix multiplication
start1 = clock();
for(int k = 0; k < N; k++) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            A[i][j] += B[i][k]*C[k][j];
        }
    }
}
finish1 = clock();
cout<<"Execution time for tridiagonal solver: " << ( (finish1 - start1)/↩
    CLOCKS_PER_SEC )<< " s" << endl;

// Free space by deleting the matrices
for(int i = 0; i < N; i++) {
    delete [] A[i];
    delete [] B[i];
    delete [] C[i];
}
delete [] A;
delete [] B;
delete [] C;

// Declarations of variables
mat A_;
mat B_ = randu(N,N);
mat C_ = randu(N,N);

start2 = clock();
A_ = B_*C_;
finish2 = clock();

cout<<"Execution time for Library Solver: "<< ( (finish2 - start2)/↩
    CLOCKS_PER_SEC ) << " s" << endl;
```

```
  return 0;
}
```