Project 1

Bernt Jonas Fløde

23. september 2018

Bevis for omskriving (1a)

Vil vise at

$$\mathbf{A}\mathbf{v} = h^2\mathbf{f},$$

eller

$$\mathbf{A}\mathbf{v}/h^2 = \mathbf{f}$$

Ved å skrive ut $-\mathbf{A}\mathbf{v}/h^2$, får vi

$$-\frac{v_2 - 2v_1}{h^2} = f_1 \quad \text{for } i = 1, \ldots, n,$$
$$-\frac{v_3 + v_1 - 2v_2}{h^2} = f_2 \quad \text{for } i = 1, \ldots, n,$$
$$-\frac{v_4 + v_2 - 2v_3}{h^2} = f_3 \quad \text{for } i = 1, \ldots, n,$$

og så videre, med det siste leddet

$$-\frac{v_{n-1} - 2v_n}{h^2} = f_n \quad \text{for } i = 1, \ldots, n,$$

Mer kompakt betyr dette det samme som

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

som er definisjonen av den andrederivertes tilnærming.

Altså vil den kontinuerlige ekvivalenten til dette uttrykket være

$$-u''(x) = f(x),$$

som var utgangspunktet for utledningen, og beviset er ferdig.

Algoritme for tridiagonale

Vi skal her løse ligningen $\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$ med hensyn på $\mathbf{v}$. Det gjør vi ved å radredusere matrisen $[\mathbf{A}\ \tilde{\mathbf{b}}]$, vi skal da få $[\mathbf{I}\ \mathbf{v}]$ der $\mathbf{I}$ er identitetsmatrisen.

Altså skal vi radredusere følgende matrise:

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots & \tilde{b}_1 \\ a_1 & b_2 & c_2 & \dots & \dots & \dots & \tilde{b}_2 \\ 0 & a_2 & b_3 & c_3 & \dots & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} & \dots \\ & & & & a_{n-1} & b_n & \tilde{b}_n \end{bmatrix}$$

For å gjøre notasjonen lettere senere setter vi $b_1 = \beta_1$ og $\tilde{b}_1 = \tilde{\beta}_1$, slik at matrisen er:

$$\begin{bmatrix}
\beta_1 & c_1 & 0 & \dots & \dots & \dots & \tilde{\beta}_1 \\
a_1 & b_2 & c_2 & \dots & \dots & \dots & \tilde{b}_2 \\
0 & a_2 & b_3 & c_3 & \dots & \dots & \dots \\
 & \dots & \dots & \dots & \dots & \dots & \dots \\
 & & & a_{n-2} & b_{n-1} & c_{n-1} & \dots \\
 & & & & a_{n-1} & b_n & \tilde{b}_n
\end{bmatrix}$$

Vi starter radreduksjonen med å få vekk a-ene. For å radredusere vekk a_1, kan vi gjøre II: II - a_1/I (dvs. vi regner ut a_1 delt på hvert tall i første raden, og trekker resultatet fra tilsvarende tall i andre rad). Da får vi følgende matrise:

$$\begin{bmatrix}
\beta_1 & c_1 & 0 & \dots & \dots & \dots & \tilde{\beta}_1 \\
0 & \beta_2 & c_2 & \dots & \dots & \dots & \tilde{\beta}_2 \\
0 & a_2 & b_3 & c_3 & \dots & \dots & \dots \\
 & \dots & \dots & \dots & \dots & \dots & \dots \\
 & & & a_{n-2} & b_{n-1} & c_{n-1} & \dots \\
 & & & & a_{n-1} & b_n & \tilde{b}_n
\end{bmatrix}$$

der $\beta_2 = b_2 - a_1/c_1$ og $\tilde{\beta}_2 = \tilde{b}_2 - a_1/\tilde{\beta}_1$.

Ved å fortsette slik for $\beta_{i+1} = b_{i+1} - a_i/c_i$ og $\tilde{\beta}_{i+1} = \tilde{b}_{i+1} - a_i/\tilde{\beta}_i$, får vi følgende matrise:

$$\begin{bmatrix}
\beta_1 & c_1 & 0 & \dots & \dots & \dots & \tilde{\beta}_1 \\
0 & \beta_2 & c_2 & \dots & \dots & \dots & \tilde{\beta}_2 \\
 & \dots & \beta_3 & c_3 & \dots & \dots & \dots \\
 & & \dots & \dots & \dots & \dots & \dots \\
 & & & & \beta_{n-1} & c_{n-1} & \dots \\
 & & & & & \beta_n & \tilde{\beta}_n
\end{bmatrix}$$

Fra teorien i lineær algebra kan vi nå se hva svaret må bli. Den nederste raden forteller at $\beta_n v_n = \tilde{\beta}_n$, altså $v_n = \tilde{\beta}_n/\beta_n$. Raden over forteller at $\beta_{n-1}v_{n-1} + c_{n-1}v_n = \tilde{\beta}_{n-1}$, altså $v_{n-1} = \frac{\tilde{\beta}_{n-1} - c_{n-1}v_n}{\beta_{n-1}}$. Generelt forteller en rad at $\beta_i v_i + c_i v_{i+1} = \tilde{\beta}_i$, altså $v_i = \frac{\tilde{\beta}_i - c_i v_{i+1}}{\beta_i}$, og hvis vi setter $v_{n+1} = 0$ er sistnevnte formel også gyldig for $i = n$.

Oppsummert kan vi finne svaret med de tre rekursive formlene

$$\beta_{i+1} = b_{i+1} - a_i/c_i$$

$$\tilde{\beta}_{i+1} = \tilde{b}_{i+1} - a_i/\tilde{\beta}_i$$

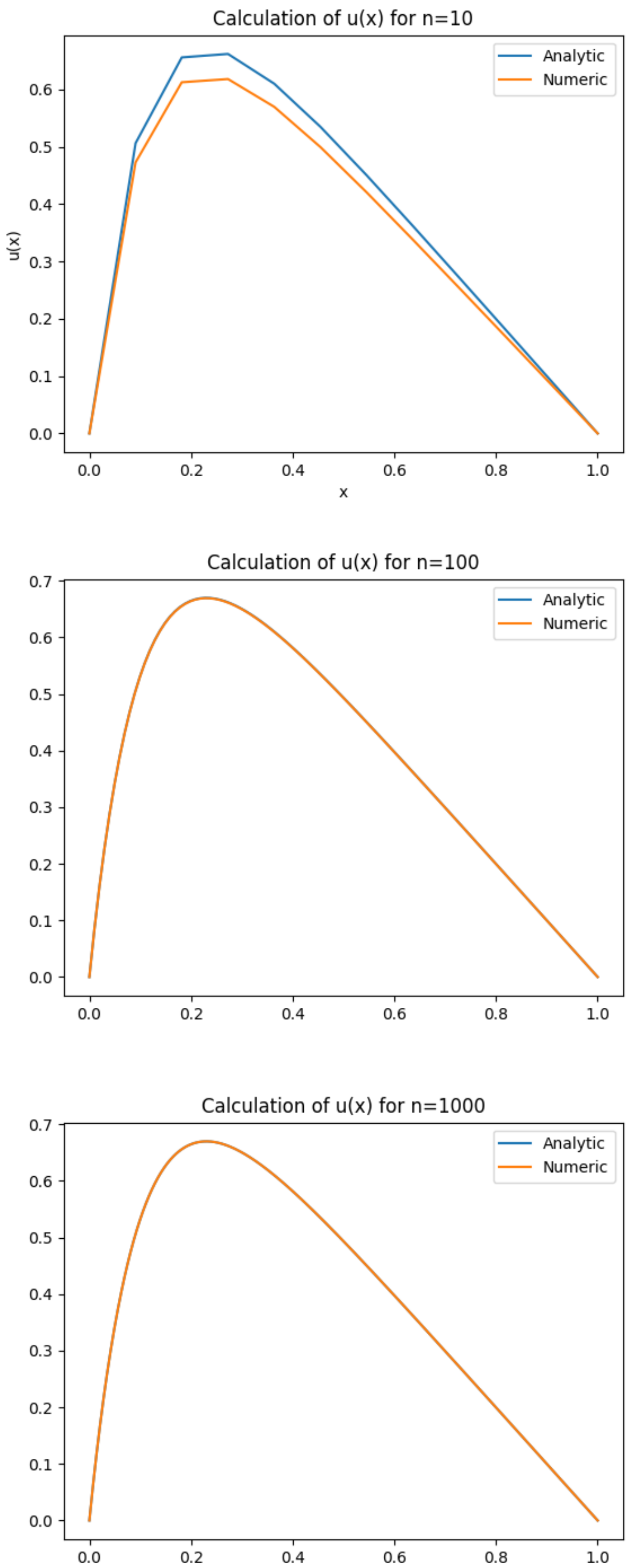
$$v_i = \frac{\tilde{\beta}_i - c_i v_{i+1}}{\beta_i}$$

med initialbetingelsene $b_1 = \beta_1$, $\tilde{b}_1 = \tilde{\beta}_1$ og $v_{n+1} = 0$.

I C++ kan dette implementeres ved hjelp av for-løkker. De to første rekursive formlene implementeres i en forlengs for-løkke, mens den siste implementeres i en baklengs for-løkke. De to første initialbetingelsene gis direkte i koden, mens den siste er implisitt gitt i C++. Koden blir slik:

```
beta[1]       = b[1];
beta_tilde[1] = b_tilde[1];
for (int i=1; i<n; i++) {
    a_div_beta = a[i] / beta[i];
    beta[i+1]       = b[i+1]    - a_div_beta * c[i];
    beta_tilde[i+1] = beta[i+1] - a_div_beta * beta_tilde[i];
}
for (int i=n; i>0; i--) {
    v[i] = (beta_tilde[i] - c[i] * v[i+1]) / (b_tilde[i]);
}
return v;
```

der `a_div_beta` er en hjelpevariabel som sparer programmet fra å regne ut dette to ganger, og dermed gjøre programmet raksere.



Figur 1: Analytisk og numerisk løsning av differensialligning for n=10, n=100 og n=1000.

Hvis vi ser på koden er det åtte flyttallsoperasjoner inne i for-løkkene, som vi si at tiden som utregningen bruker går som 8 n.

Denne koden finnes også i dataprogrammet `tridiagonal.cpp` lenger ned. Dataprogrammet finner en numerisk løsning av $-u''(x) = f(x)$ der $f(x) = 100e^{-10x}$ som er funnet med metoden som her har blitt beskrevet, og sammenligner det med den analytiske løsningen $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. Dette gjør det for $n = 10$, $n = 100$ og $n = 1000$, se figur 1.

Spesialtilfelle

Vi ser nå på et spesialtilfelle der $a_i = a$, $b_i = b$ og $c_i = c$ for alle i, for tre gitte tall a, b og c. Da er det mulig å forenkle dataprogrammet ved å gjøre om arrayene `a`, `b` og `c` til enkle flyttall. Det er også nærliggende å tro at det skulle være mulig med flere forenklinger i koden får å få den til å gå enda raskere, men noen slik forenkling har jeg ikke funnet. Tiden som utregningen bruker går altså fortsatt som 8 n. Koden blir slik:

```
beta[1]        = b;
beta_tilde[1] = b_tilde[1];
for (int i=1; i<n; i++) {
    a_div_beta = a / beta[i];
    beta[i+1]        = b          - a_div_beta * c;
    beta_tilde[i+1] = beta[i+1] - a_div_beta * beta_tilde[i];
}
for (int i=n; i>0; i--) {
    v[i] = (beta_tilde[i] - c * v[i+1]) / (b_tilde[i]);
}
return v;
```

Her er tabellen som blir skrevet ut:

	$log(h)$	ϵ_{max}
$n = 10^1$	-1.04	-1.17970
$n = 10^2$	-2.00	-3.08804
$n = 10^3$	-3.00	-5.08005
$n = 10^4$	-4.00	-7.07929
$n = 10^5$	-5.00	-8.84297
$n = 10^6$	-6.00	-6.07547
$n = 10^7$	-7.00	-5.52523

Vi ser at nøyaktigheten blir bedre og bedre for mindre h, fram til et punkt der avrundingsfeil blir dominerende.

tridiagonal.cpp

```
#include "src/head.h"
#include "src/find_nums.cpp"
#include "src/make_vecs.cpp"
#include "src/timing.cpp"
#include "src/make_table_file.cpp"
using namespace arma;

vec rref_tridiagonal(int n, double* a, double* b, double* c, double h, vec f) {
    double *b_new = new double[n+2];
    double *f_new = new double[n+2];
    vec v = zeros<vec>(n+2);
    double a_div_bnew;

    b_new[1] = b[1];
    f_new[1] = f[1];
    for (int i=1; i < n; i++) {
        a_div_bnew = a[i] / b_new[i];
        b_new[i+1] = b[i+1] - a_div_bnew*c[i];
        f_new[i+1] = f[i+1] - a_div_bnew*f_new[i];
    }
    for (int i=n; i>0; i--) {
        v[i] = (f_new[i] - c[i]*v[i+1])/b_new[i];
    }

    return v;
}

vec rref_tridiagonal_3num(int n, double a, double b, double c, double h, vec f) {
    double *b_new = new double[n+2];
    double *f_new = new double[n+2];
    vec v = zeros<vec>(n+2);
    double a_div_bnew;

    b_new[1] = b;
    f_new[1] = f[1];
    for (int i=1; i < n; i++) {
        a_div_bnew = a / b_new[i];
```

```
        b_new[i+1] = b - a_div_bnew*c;
        f_new[i+1] = f[i+1] - a_div_bnew*f_new[i];
    }
    for (int i=n; i>0; i--) {
        v[i] = (f_new[i] - c*v[i+1])/b_new[i];
    }
    return v;
}

// int main(int argc, char *argv[]) {
int do_for_one_n(int n) {
    // Define numbers
    double h = 1. / (n + 1);

    // Numerical calculation
    double *a = make_abc(n, -1);
    double *b = make_abc(n, 2);
    double *c = make_abc(n, -1);
    vec y = make_y(n, f);
    auto start = time_start();
    vec v = rref_tridiagonal(n, a, b, c, h, y);
    cout << "Tridiagonal time: "; time_finish(start);
    start = time_start();
    vec v_3num = rref_tridiagonal_3num(n, -1., 2., -1., h, y);
    cout << "Tridiagonal 3num time: "; time_finish(start);


    // Analytical calculation
    vec u = make_u_anal(n);

    // Make file
    if (n < 10000) {
        vec *out_array = new vec[4];
        out_array[0] = make_h(n);
        out_array[1] = u;
        out_array[2] = v;
        out_array[3] = v_3num;
        make_table_file("tridiagonal_n=" + to_string(n), n+2, 4, out_array);
    }


    // Print
    cout << "Error when n=" << n << ": " << error_max(n, v, u) << endl;
    cout << "Error 3num when n=" << n << ": " << error_max(n, v_3num, u) << endl;
    cout << "log(h): " << log10(h) << endl;
    cout << '\n';

    return 0;
}

int main() {
    for (int i = 1; i < 8; i++) {
        do_for_one_n(pow(10, i) );
    }

    return 0;
}

// Tables:
// x | Analytix | Numerical
```

LU-dekomposisjon

Prøver også å skrive et program for ordinært LU-dekomposisjon til å løse de samme ligningene som over. Dette er ventet å bruke mer tid fordi det er flere flyttallsoperasjoner. Dessuten får programmet en feilmelding som je*/g ikke vet hvordan kan løses.

LU.cpp

```
#include "src/head.h"
#include "src/find_nums.cpp"
#include "src/make_vecs.cpp"
#include "src/timing.cpp"
using namespace arma;

int do_for_one_n(int n) {
    // Define numbers and arrays
    double h = 1. / (n + 1);
    vec y = make_y(n, f);
    double *a_i = make_abc(n, -1.);
    double *b_i = make_abc(n, 2.);
    double *c_i = make_abc(n, -1.);
```

```
    mat A = make_A(n);
    vec v;

    // Numerical calculation
    auto start = time_start();
    solve(v, A, y);
    cout << "LU time: "; time_finish(start);
    vec u = zeros<vec>(n+2);
    for (int i=0; i<n+1; i++) {
        u[i] = u_anal(i*h);
    }

    // Print
    cout << "Error when n=" << n << ": " << error_max(n, v, u) << endl;
    cout << "log(h): " << log10(h) << endl;
    cout << '\n';

    return 0;
}

int main() {
    for (int i = 1; i < 4; i++) {
        do_for_one_n(pow(10, i) );
    }

    return 0;
}
```

Annet kode

Følgende Python-program blir brukt til å kjøre de andre programmene: project1.py

```
import os, sys
import matplotlib.pyplot as plt
from math import log10

dirname = os.path.dirname(sys.argv[0])
if len(dirname) > 0:
    dirname += '/'

# Running C++ program
os.popen('c++ tridiagonal.cpp')
os.system('./' + dirname + 'a.out > out/tridiagonal_output.dat')
#os.popen('c++ LU.cpp')
#os.system('./' + dirname + 'a.out > out/LU_output.dat')
# os.popen()

# Preparing plots
plt.xlabel('x')
plt.ylabel('u(x)')

# Making plots
for n in [10, 100, 1000]:
    infile = open('out/tridiagonal_n=%g.dat' % n, 'r')

    x = []
    analytic = []
    numeric  = []

    for line in infile:
        numbers = line.split()
        x.append(eval(numbers[0]))
        analytic.append(eval(numbers[1]))
        numeric.append(eval(numbers[2]))

    plt.plot(x, analytic, label='Analytic')
    plt.plot(x, numeric, label='Numeric')
    plt.title('Calculation of u(x) for n=%g' % n)
    plt.legend()
    plt.savefig('plots/tridiagonal_n=%g.png' % n)
    plt.clf()

    infile.close()

# Collecting data for table
infile = open('out/tridiagonal_output.dat', 'r')

epsilons = []
loghs    = []

for i in range(1, 8):
    infile.readline()
    infile.readline()
```

```
    epsilons.append(eval(infile.readline().strip().split()[-1]))
    infile.readline()
    loghs.append(eval(infile.readline().strip().split()[-1]))
    infile.readline()

infile.close()

# Making table
outfile = open('out/project1d_table.dat', 'w')



outfile.write('\\begin{tabular}{ c c c } ')
outfile.write(' '*9 + '& $log(h)$ & $\epsilon_{max}$ \\\\ \n')
outfile.write('\\hline')
for i, e, l in zip(range(1, 8), epsilons, loghs):
    outfile.write(' $n=10^%1.0f$ & %5.2f & %8.5f \\\\ \n' % (i, l, e))
outfile.write('\\end{tabular}\n')

outfile.close()
```