Project 1, FYS 3150 / 4150, fall 2013

Nathalie Bonatout and Odd Petter Sand

September 9, 2013

1 Introduction

In this project we will solve the one-dimensional Poissson equation with Dirichlet boundary conditions by rewriting it as a set of linear equations.

To be more explicit we will solve the equation

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

and we define the discretized approximation to u as v_i with grid points $x_i = ih$ in the interval from $x_0 = 0$ to $x_{n+1} = 1$. The step length or spacing is defined as $h = 1/(n+1)$. We have then the boundary conditions $v_0 = v_{n+1} = 0$. We approximate the second derivative of u with

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

where $f_i = f(x_i)$.

(Author's note: This text, and the text introducing the various exercises, is taken from the project description provided at the course website.)

2 Exercises

2.1 Exercise a)

We start with the given equation

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

where $i \in [1,n] \cap \mathbb{N}$ and $n \in \mathbb{N}$. We will assume that $n \geqslant 3$ and that $v_0 = v_{n+1} = 0$.

$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i \equiv \widetilde{b}_i$

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix} = \widetilde{b}_i$$

Now we expand these vectors from 3 elements to n elements. Note that the ith element of the row vector should be 2 and the ith element of the column vector should be v_i after the expansion:

$$\begin{bmatrix} 0 & \cdots & -1 & 2 & -1 & \cdots & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_{i-1} \\ v_i \\ v_{i+1} \\ \vdots \\ v_n \end{bmatrix} = \widetilde{b}_i$$

Note that in the row vector 2 can very well be the first or last element, in which case the preceding presentation can be a little misleading. We further recognize the row vector as the ith row of the $n \times n$ matrix $\mathbf{A}$ (defined such that $A_{ij} = 2\delta_{ij} - \delta_{i(j-1)} - \delta_{i(j+1)}$) and get the inner product

$\mathbf{A}_i \cdot \mathbf{v} = \widetilde{b}_i$

and remembering that $i \in [1,n]$, by the definition of matrix multiplication

$\mathbf{Av} = \widetilde{\mathbf{b}}$

which is what we wanted to show.

2.2 Exercise b)

The algorithm we will use is as follows: First we do forward subsitution by subtracting from the current row a multiple of the row before it.

$A_i = A_i - x_i A_{i-1}$

Here, x_i is the factor that cause the term a_i to cancel out. Before the first step, the rows $i-1$ and i look like this (the * indicates a value that has been changed by the algoritm)

$$\begin{array}{ccccccc} \cdots & 0 & b^*_{i-1} & c_{i-1} & 0 & 0 & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \end{array}$$

and our goal is that it look like this after the forward substitution

$$\begin{array}{ccccccc} \cdots & 0 & b^*_{i-1} & c_{i-1} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & b^*_i & c_i & 0 & \cdots \end{array}$$

(note that c_i is unchanged). When we reach the bottom, we do a backward subsitution by adding to the current row a multiple of the row below it and then dividing the last element by itself to make the last element equal to 1:

$A_i = A_i - c_i A_{i+1}$

$A_i = \frac{A_i}{b^*_i}$

That is to say, we go from

$$\begin{array}{ccccccc} \cdots & 0 & b^*_i & c_i & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \end{array}$$

to

$$\begin{array}{ccccccc} \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \end{array}$$

Naturally, we also have to do equivalent operations on the vector $\widetilde{\mathbf{b}}$ on the other side of the equation. We will not calculate any unnecessary values, i.e. values

that will not be used by the program later on. Hence the algoritm looks like this (number of flops in parantheses):

For $i : 2 \longrightarrow n$

1. $x_i = \frac{a_i}{b^*_{i-1}}$ (n)
2. $b_i = b_i - x_i c_{i-1}$ $(2n)$
3. $\widetilde{b}_i = \widetilde{b}_i - x_i \widetilde{b}_{i-1}$ $(2n)$

$\widetilde{b}_n = \frac{\widetilde{b}_n}{b_n}$ (1)

For $i : n-1 \longrightarrow 1$

1. $\widetilde{b}_i = \widetilde{b}_i - c_i \widetilde{b}_{i+1}$ $(2n)$
2. $\widetilde{b}_i = \frac{\widetilde{b}_i}{b_i}$ (n)

This makes the total running time of the algorihm $8n$ (actually, it is $8(n-1)$, but we are mostly concerned with the performance for large n, where $n \approx n-1$).

If the matrix is symmetric, so that $a_i = c_{i-1}$, then $x_i c_{i-1} = \frac{a_i^2}{b^*_{i-1}}$.

If $a_i = c_{i-1} = k$ for all i, we have a special case where $k = \pm 1$. Then $x_i = \pm \frac{1}{b^*_{i-1}}$, $k^2 = 1$ and $x_i c_{i-1} = \frac{1}{b^*_{i-1}} = -x_i \equiv y_i$.

Furthermore, $c_i \widetilde{b}_{i+1} = k \widetilde{b}_{i+1} = \mp \widetilde{b}_{i+1}$ (note: opposite of the sign of k). The algorithm will then look like this for $k = -1$:

For $i : 2 \longrightarrow n$

1. $y_i = \frac{1}{b^*_{i-1}}$ (n)
2. $b_i = b_i - y_i$ (n)
3. $\widetilde{b}_i = \widetilde{b}_i + y_i \widetilde{b}_{i-1}$ $(2n)$

$\widetilde{b}_n = \frac{\widetilde{b}_n}{b_n}$ (1)

For $i : n-1 \longrightarrow 1$

1. $\widetilde{b}_i = \widetilde{b}_i + \widetilde{b}_{i+1}$ (n)

2. $\widetilde{b}_i = \frac{\widetilde{b}_i}{b_i}$ (n)

This gives a total running time of $6n$ for this implementation of the algorithm, which we ended up using for this project.

<INSERT RESULTS + PLOTS HERE>

2.3 Exercise c)

In thie exercise we will compute the relative error in the data set $i = 1, \ldots, n$,by setting up

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

as function of $log_{10}(h)$ for the function values u_i and v_i. For each step length we will extract the max value of the relative error.

...

2.4 Exercise d)

(TODO: Change the text below to sound less like an exercise and more like a description.)

Compare your results with those from the LU decomposition codes for the matrix of sizes 10×10, 100×100 and 1000×1000. Here you should use the library functions provided on the webpage of the course. Use for example the unix function *time* when you run your codes and compare the time usage between LU decomposition and your tridiagonal solver. Alternatively, you can use the functions in C++, Fortran or Python that measure the time used.

Make a table of the results and comment the differences in execution time How many floating point operations does the LU decomposition use to solve the set

of linear equations? Can you run the standard LU decomposition for a matrix of the size $10^5 \times 10^5$? Comment your results.

To compute the elapsed time in c++ you can use the following statements

Time in C++

```
using namespace std;
...
#include "time.h"   //  you have to include the time.h header
int main()
{
    // declarations of variables
    ...
    clock_t start, finish;  //  declare start and final time
    start = clock();
    // your code is here, do something and then get final time
    finish = clock();
    ( (finish - start)/CLOCKS_PER_SEC );
...

...
```

2.5 Exercise e)

In this exercise we are investigating matrix multiplication in row major order versus column major order with regards to running time.

The task here is to write a small program which sets up two random (use the ran0 function in the library lib.cpp to initialize the matrix) double precision valued matrices of dimension 5000×5000. (NOTE: The original value of $10^4 \times 10^4$ proved too memory intensive for our poor laptops.)

The multiplication of two matrices $\mathbf{A} = \mathbf{BC}$ could then take the following form in standard row-major order

```
        for(j=0 ; j < n ; j++) {
            for(k=0 ; k < n ; k++) {
                a[i][j]+=b[i][k]*c[k][j]
            }
        }
    }
```

and in a column-major order as

```
        for(i=0 ; i < n ; i++) {
            for(k=0 ; k < n ; k++) {
                a[i][j]+=b[i][k]*c[k][j]
            }
        }
    }
```

(NOTE: We implemented this using dynamic memory allocation, as lib.cpp proved difficult to add to the project in Visual C++.)

The output of our program:

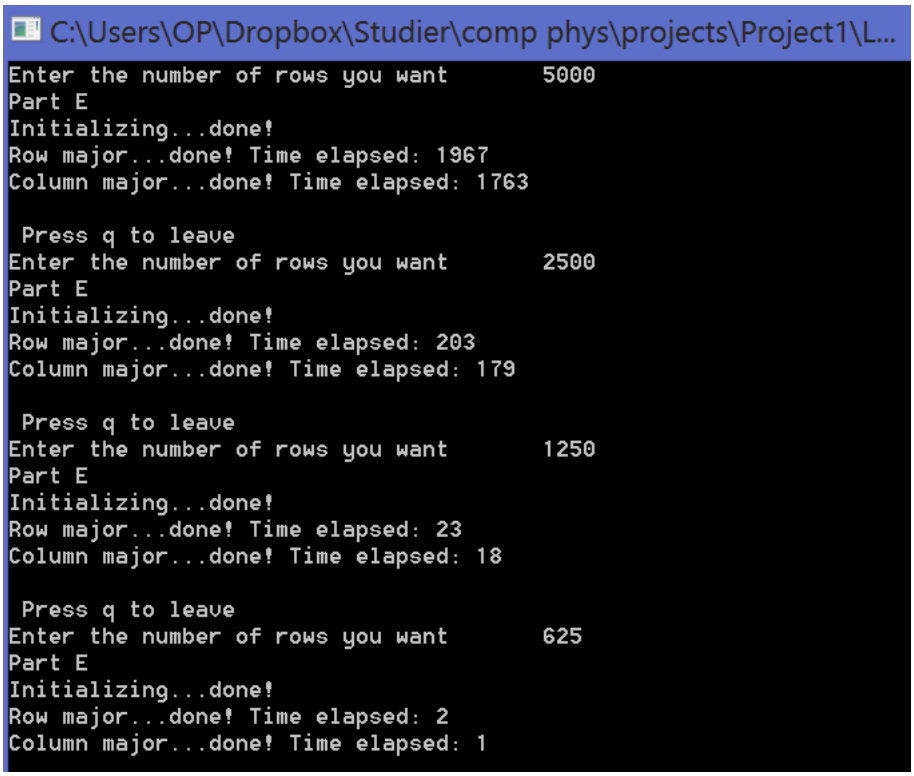

Figure #: -

TOTALLY BOGUS due to coffee break. :p

Figure N: (Note that the "Part B" is a typo. It should read "Part E".)

3 Conclusion

In this project we learned that...

3.1 Critique

We would like to provide the following items of feedback for future versions of this project:

- lib.cpp sucks!

- we need more RAM. you made our laptops feel sad.
- ???