UNIVERSITY OF OSLO

COMPUTATIONAL PHYSICS

Project 1

Authers:

Birgitte Madsen

Soumya ??

Autumn 2015

UiO : University of Oslo

Department of Physics
University of Oslo
Sem Slands vei 24
0371 Oslo, Norway
+47 22 85 64 28
http://www.mn.uio.no/fysikk/english/

Course:

Computational Physics

Project number:

1

Hand-in deadline:

Monday 7 September, 2015

Synopsis:

Project Members:

Birgitte Madsen
Soumya ??

Copies: ??
Page count: 9
Appendices: ??
Completed: ??

PREFACE

This project is written by 6th semester physics group 4.207a at the Department of Physics and Nanotechnology at Aalborg University, Denmark, in the Spring semester, 2014, as a 10 ECTS-point bachelor project.

Reading Guide

Succeeding chapters support each other, and it is therefore recommended to read the report chronologically. When referring to equations or the like in the text, *equation* will be shortened Eq., *table* will be shortened Tab., and so forth. In App. **??** a list of frequently used symbols and constants are given. The external references used in this work appear in numbered order in brackets in the text and are listed in the bibliography at the end of the report in order of succession.

Signatures

The group member's signatures below express that the entire group is accountable for all aspects of the project and all chapters of the report.

Andreas V. Pedersen	Birgitte Madsen

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

Hello, we love computational physics!!

$$\sqrt(2) = \sin(\Theta + \phi \cdot 23)$$

1.1 Nature of the problem

1.2 Description of the Algorithm

The algorithm written to solve the problem of computing the u in [1] uses the Gaussian elimination method. However, since the linear problem includes a special matrix, namely a tridiagonal matrix, the number of floating points operation needed to solve this specific problem can be meget reduced by modifying the Gaussian elimination method.

The algorithm used to solve this problem, however still, consists of the same two steps as in the Gaussian elimination: forward substitution and backward substitution.

Let us first adress the forward substitution. the aim of this first part of the algorithm is essentially to make the matrix A into an upper triangular matrix by suitable subtractions of multiplas of the first row from the other rows in the matrix. This gives rise to a change in the matrix elements. However, the elements in the first row of the matriw will not be changed, and hence we have that

$$\tilde{b}_1 = b_1 \tag{1.1}$$

in which $\tilde{b}_1$ is the element in the first column and first row of the computed triangular matrix. By writing out the computed matrix elements after subtracting multipla of the first row from the other rows to create zeros below the diagonal and using the fact that all elements right above and below the diagonal of A is equal to -1 whilst the remaining elements are 0, it is seen that the elements in the diaginal for $i > 1$,

[1] FiXme Note: eq-ref

named b_i in the tridagonal matrix A and $\tilde{b}$ in the computed triangular matrix, get the value

$$\tilde{b}_i = b_i - \frac{1}{b_{i-1}} \tag{1.2}$$

Likewise, the elements in the vector $\vec{f}$ are changed to

$$\tilde{f}_i = f_i + \frac{f_{i-1}}{b_{i-1}} \tag{1.3}$$

whilst the elements named a_i in A become equal to zero, and the elements c_i are unchanged.

This gives rise to the following code for the forward substitution.

```
// Forward substitution

   double abtemp[n];
   double btemp = b[0];

   for (int i=1 ; i<n ; i++)
   {
      abtemp[i] = - 1/btemp;
      btemp = b[i] + abtemp[i];
      f[i] = f[i] - f[i-1]*abtemp[i];
      b[i] = btemp;
   }
```

[2] Notice that in the above lines of code, the first element of a vector is $i = 0$.

For every time the loop runs, there are 4 flops We have chosen to calculate $1/b_{i-1}$, which is used in both (1.2) and (1.3), to reduce the number of flopsy 1 for every time the loop is run. Since the loop runs from $i = 2$ to $i = n$, if $i = 1$ is the first element of a vector, the loop runs $n-1$ times, which gives a total number of flopsor the forward substitution of

$$\#flops = 4(n-1) \tag{1.4}$$

In the back substitution, the values of the entrances of vector $\vec{u}$ in [3] are computed. Since the result from the forward substitution is an upper triangular matrix, it is evident that

$$u_n = \frac{\tilde{f}_n}{\tilde{b}_n} \tag{1.5}$$

in which $\tilde{f}_n$ and $\tilde{b}_n$ are elements of $\vec{f}$ and $\vec{b}$ after the forward substitution. From the determined value of u_n, the values of the rest of the u_i's can be determined using the fact, that all elements in the upper triangular matrix is zero apart from the elements in the diagonal and the elements just above the diagonal, which all have the value -1. This gives

$$u_i = \frac{\tilde{f}_i + u_{i+1}}{\tilde{b}_i} \tag{1.6}$$

yielding a source code:

[2] FiXme Note: skriv evt. -= osv i stedet
[3] FiXme Note: eq-ref

```
// Back substitution

    u[n-1] = f[n-1]/b[n-1];

    for(int i=n-1 ; i>= 0; i--)
    {
        u[i] = (f[i]-ac*u[i+1])/b[i];
    }
```

For each time the loop runs, there are 2 flops Like in the forward substitution, the loop runs $n-1$ times, yielding

$$\#flops = 2(n-1)+1 \tag{1.7}$$

Hence, the total number of flopsor both the forward substitution and back substitution is

$$\#flops_{total} = 4(n-1)+2(n-1)+1 = 6n-6 \tag{1.8}$$

which gives that the number of flops goes as $6n$ or $\mathcal{O}(n)$.

4

1.3 Source Code

[4] FiXme Note: compare flops m. GE and LU

CHAPTER

2

RESULTS

2.1 Reliabilty and Numerical Stability of Results

2.2 Interpretation of Results

CHAPTER

3

CRITIQUE

CHAPTER

4

CONCLUSION