FYS3150/FYS4150
Computational Physics
Project 1

Magnus Ulimoen
Krister Stræte Karlsen

September 8, 2015

1 Motivation

A great deal of differential equations in the natural sciences can be written as a linear second-order differential equation on the form

$$\frac{d^y}{dx^2} + k^2(x)y = f(x) \tag{1.1}$$

Some examples include

I Schrödinger's equation

II Airy function

III Economics?

IV Poisson's equation

V Laplace's equation (Poisson's with f=0)

So an accurate method of finding the solution to these problems should be found.

1.1 Example: Electromagnetism

The electrostatic potential Φ is generated from a charge distribution ρ. This gives

$$\nabla^2\Phi = -4\pi\rho(\mathbf{r}) \tag{1.2}$$

Letting ρ be spherically symmetric this leads to a symmetric Φ. This gives with a substitution $\Phi(r) = \phi(r)/r$

$$\frac{d^2\phi}{dr^2} = -4\pi r\rho(r) \tag{1.3}$$

Which is on the form of (1.1).

2 Discretization

Solving the equation

$$\frac{d^2v}{dr^2} = f(r) \tag{2.1}$$

Letting r be discretized into N pieces,

$$r_0, r_1, \ldots, r_i, \ldots, r_{N-1} \tag{2.2}$$

where

$$h = \frac{r_{N-1} - r_0}{N} \tag{2.3}$$

$$r_i = r_0 + i * h \tag{2.4}$$

And using the three-point formula from the symmetric Taylor-expansion for the second derivative of f,

$$\frac{d^2v}{dx^2} \simeq \frac{v_{i-1} + v_{i+1} - 2v_i}{h^2} \tag{2.5}$$

This gives the discretized formula

$$\frac{v_{i-1} + v_{i+1} - 2v_i}{h^2} = f(r_i) \tag{2.6}$$

$$v_{i-1} + v_{i+1} - 2v_i = \tilde{f}_i \tag{2.7}$$

with $\tilde{f}_i = f(r_i)h^2$.

With boundary conditions (dropping of an element)

$$v_1 - 2v_0 = f_0 \tag{2.8}$$

$$v_{N-2} - 2v_{N-1} = f_{N-1} \tag{2.9}$$

3 Thomas-algorithm

3.1 Formulation

Given the following ODE with boundary conditions,

$$-u''(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0. \tag{3.1}$$

and the dicretized form following a symmetric Taylor expansion

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for} \quad i = 1, \ldots, n, \quad v_0 = v_n = 0 \tag{3.2}$$

we are going to show it can be written as system of linear equations of the form:

$$\mathbf{Av} = \tilde{\mathbf{b}}, \tag{3.3}$$

3.2 Solution

Multipling the discretized equation (3.2) by h^2 we get:

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

Filling in for i and choosing $\tilde{b_i} = h^2 f_i$ we obtain the following set of equations:

$$\begin{aligned}
2v_1 - v_2 &= \tilde{b_1} \\
-v_1 + 2v_2 - v_3 &= \tilde{b_2} \\
&\vdots \\
-v_{i-1} + 2v_i - v_{i+1} &= \tilde{b_i} \\
&\vdots \\
-v_{n-1} + 2v_n &= \tilde{b_n}
\end{aligned}$$

Now one can easily see that this system of linear equations can written on the form of (3.3), where

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{pmatrix}, \quad \tilde{\mathbf{b}} = \begin{pmatrix} \tilde{b_1} \\ \tilde{b_2} \\ \dots \\ \dots \\ \dots \\ \tilde{b_n} \end{pmatrix}.$$

3.3 Algorithm for a tridiagonal system of linear equations

We start by looking at the system of equations:

$$b_1 v_1 + c_1 v_2 = \tilde{b_1} \tag{3.4}$$
$$a_2 v_1 + b_2 v_2 + c_3 v_3 = \tilde{b_2} \tag{3.5}$$
$$a_3 v_2 + b_3 v_3 + c_3 v_4 = \tilde{b_3} \tag{3.6}$$
$$\vdots$$
$$a_n v_{n-1} + b_n v_n = \tilde{b_n} \tag{3.7}$$

If we solve (3.4) for v_1 and insert it into (3.5) we obtain the following "modified second equation":

$$(b_1 b_2 - a_2 c_1) v_2 + b_1 c_2 v_3 = b_1 \tilde{b_2} - a_2 \tilde{b_1}$$

Now having successfully removed v_1 from the second equation we can go on and solve it for v_2 and insert it into the third equation obtaining:

$$(b_3(b_1 b_2 - a_2 c_1) - a_3 b_1 c_2) v_3 + c_3(b_1 b_2 - a_2 c_1) v_4 = \\ (b_1 b_2 - a_2 c_1)\tilde{b_3} - a_3 b_1 \tilde{b_2} + a_2 a_3 \tilde{b_1}$$

The two modified equations may be written as

$$v_2 = \frac{b_1 \tilde{b_2} - a_2 \tilde{b_1}}{b_1 b_2 - a_2 c_1} - \frac{b_1 c_2}{b_1 b_2 - a_2 c_1} v_3 = \beta_2 + \gamma_2 v_3$$

$$\begin{aligned} v_3 =& \frac{(b_1 b_2 - a_2 c_1)\tilde{b_3} - a_3(b_1 \tilde{b_2} - a_2 \tilde{b_1})}{b_3(b_1 b_2 - a_2 c_1) - a_3 b_1 c_2} - \frac{c_3(b_1 b_2 - a_2 c_1)}{b_3(b_1 b_2 - a_2 c_1) - a_3 b_1 c_2} v_4 \\ =& \frac{\tilde{b_3} - a_3 \beta_2}{a_3 \gamma_2 + b_3} + \frac{-c_3}{a_3 \gamma_2 + b_3} v_4 = \beta_3 + \gamma_3 v_4 \end{aligned}$$

This prossess can be repeated up untill the last equation. This is the forward substitution step. From the last equation we compute v_n and get all we need to compute v_{n-1}, then v_{n-2}, and so on. This is the backward substitution part of the algorithm. A shrewd reader might see that the coefficiants, β and γ, take a recursive form

$$\beta_i = \frac{\tilde{b_i} - a_i\beta_{i-1}}{a_i\gamma_{i-1} + b_i}, \qquad \gamma_i = \frac{-c_i}{a_i\gamma_{i-1} + b_i},$$

and the equation for v_{i-1} reads:

$$v_{i-1} = \beta_{i-1} + \gamma_{i-1}v_i \tag{3.8}$$

It follows from (3.4) that $\beta_1 = \frac{\tilde{b_1}}{b_1}$ and $\gamma_1 = \frac{-c_1}{b_1}$. From combining (3.7) and (3.8) we get

$$v_n = \frac{\tilde{b_n} - a_n\beta_{n-1}}{a_n\gamma_{n-1} + b_n} = \beta_n.$$

Having all the nessecary ingredients the algorithm reads as follows.

Algorithm I

$a_i = c_i = -1, \quad i = 1, 2, 3, .., n$
$b_i = 2, \quad i = 1, 2, 3, .., n$
$\tilde{b_i} = h^2 f_i \quad i = 1, 2, 3, .., n$
$\beta_1 = \frac{\tilde{b_1}}{b_1}$
$\gamma_1 = \frac{-c_1}{b_1}$
for $i = 2, 3, .., n$
    $\beta_i = \frac{\tilde{b_{i-1}} - a_{i-1}\beta_{i-1}}{a_{i-1}\gamma_{i-1} + b_{i-1}}, \qquad \gamma_i = \frac{-c_{i-1}}{a_{i-1}\gamma_{i-1} + b_{i-1}}$

$v_n = \beta_n$
for $i = n, n-1, .., 2$
    $v_{i-1} = \beta_{i-1} + \gamma_{i-1}v_i$

This is often refered to as *Thomas Algorithm*, an algorithm for solving tridiagonal systems of linear equations.

3.4 FLOPS

The forward substitution requires one pass forwards, and one pass backwards, and have a complexity of $\mathcal{O}(N)$

4 Matrix solutions

Can solve the equation (3.3) directly by LU-decomposition. The matrix A is then transformed into one upper triagonal matrix and one lower diagonal matrix. By doing this on a matrix this results in an easier way to solve several $Ax = y$ for different y.

Direct solver by arma::solve(A, y) is used to compare with the LU-method. This is ordinary quicker, since armadillo is a high-level wrapper around lapack etc.

4.1 FLOPS

The standard is $\mathcal{O}\left(N^3\right)$ complexity

5 Implementation

5.1 Thomas algorithm

Allocation of temporary arrays to hold bprime and dprime...

5.2 Matrix

Using the armadillo library for the solution...

5.3 Sparse matrix

The matrix solution above is very ineffective in both FLOPS and memory usage. This is improved by using sparse matrices, which saves the position and the value, and assumes everything else is zero, which saves a lot of space in the memory for our purpose ($(N-3)^2$ memory space saved).

6 Efficiency

Comparing the different algorithms for different h and against each other

Compare relative error. What is rel-error compared to FLOPS?