PHY 480 Project 1

Benjamin Brophy

February 12, 2016

In this project I will numerically solve the Poisson equation with Dirchelet boundary conditions by rewriting it as a set of linear equations. I use the Thomas algorithm for solving a tridiagonal matrix, but simplified for the case that the off-diagonal elements of the tridiagonal matrix are all -1. I will first discuss the methods and results, and the code is included at the end of the report.

We are solving the equation $-u''(x) = f(x)$ for $u(x)$, where $f(x) = 100e^{-10x}$, on $x \in [0, 1]$ with $u(0) = u(1) = 0$. This is discretized to n grid points $x_i = ih$ with spacing $h = \frac{1}{n+1}$. The discretized approximate of $u(x_i)$ is v_i. The h^2 order approximation of $f(x)$ is then

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

. Let $b_i \in \mathbf{b}$ be defined as $b_i = h^2 f_i$. Then

$$-v_{i+1} + 2v_i - v_{i+1} = b_i$$

$$(0 \quad \cdots \quad 0 \quad -1 \quad 2 \quad -1 \quad 0 \quad \cdots \quad 0)(v_0 \quad \cdots \quad v_{i-1} \quad v_i \quad v_{i+1} \quad \cdots \quad v_n)^T = b_i$$

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots \\ 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ \cdots \\ v_i \\ \cdots \\ v_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \cdots \\ b_i \\ \cdots \\ b_n \end{bmatrix}$$

Thus,

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

To solve this matrix I will use a simplified version of Gaussian elimination for tridiagonal matrices called the Thomas algorithm, further simplified for the values of this tridiagonal matrix. The first step is forward substitution. After this step, all elements other than the diagonal and elements 1 to the right are 0, all values 1 right of the diagonal are -1, so we only need to compute the new diagonal values, $\tilde{d}_i$, and new values of the right hand side, $\tilde{b}_i$. ($\tilde{d}_1 = d_1 = 2$, $\tilde{b}_1 = b_1$)

$$\tilde{d}_i = d_i - \frac{1}{\tilde{d}_{i-1}}$$

$$\tilde{b}_i = b_i + \frac{\tilde{b}_{i-1}}{\tilde{d}_{i-1}}$$

Then I completed backwards substitution to solve for v_i with the following algorithms ($v_n = \tilde{b}_n$):

$$v_i = \frac{\tilde{b}_i + v_{i+1}}{\tilde{d}_i}$$

General Gaussian Elimination or LU factorization solutions would take about $frac23N^3$ flops to solve the equation. The Thomas algorithm would in general require about $4n$ flops for each forwards and backwards substitution for a total of $8n$ flops, but the simplified algorithm for this matrix reduces that to $2n$ flops for each forwards and backwards substitution, for a total of about $4n$ flops.

The resulting solutions for $N = 10, 100, 1000, 10000$ are plotted along with the analytical solution $u(x) = 1 - (1 - exp(-10))x - exp(-10x)$ on the following page. The code used is at the end of the report. The solution is a rough but somewhat looking approximation at N=10, and as far as can be seen here, looks like a near perfect representation of the f(x) as n takes on values of 1000 and higher.

For each value of N, for each step i the log of the relative error was measured with regard to the actual value $u_i = u(x_i)$, where $u(x)$ is the analytical solution $u(x) = 1 - (1 - exp(-10))x - exp(-10x)$.

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

Here are is the maximum error at any point x_i for each N. The error seems to decrease about as the increase in the order of magnitude of N.

N	**MaxError**
10	0.301744
100	0.0374884
1000	0.00384884
10000	0.000385926

variousNplots.jpg

Figure 1: The solution is a rough but somewhat looking approximation at N=10, and as far as can be seen here, looks like a near perfect representation of the f(x) as n takes on values of 1000 and higher.

When measuring the time for various my solver using "time.h" in C++, even for $N = 10^5$, the solution did not take long enough to register on a scale of seconds. My solver performs on the order of N flops. However, LU decomposition takes on the order of N^3 flops. Here are the results of my measurement of solving using LU decomposition.

N	**Time(s)**
10	0
100	0
1000	2
10000	5194
100000	N/A

So, it took about 1.44 hours to solve the problem with $N = 10^4$. When I attempted to solve the $N = 10^5$ matrix using LU decomposition, it never finished in hours, which makes sense since it should take on the order of 10^3 longer than the solution of the $N = 10^4$ case.

Here is the code used in this project, in C++:

```
using namespace std;

#include <iostream>
#include <cmath>
#include <fstream>
#include <string>
#include "time.h"
#include "../../ComputationalPhysicsMSU/doc/Programs/LecturePrograms/progra

int main()
{
clock_t start_alg, finish_alg;
cout << "N?" << endl; //size of matrix
int N, i, j;
double h;
cin >> N;
double n = N;
h = 1/(n+1);
double *A = new double[N];
// initialize A array, representing diagonal of A, all elements = 2
for(i=0 ; i < N ; i++) {
A[i] = 2.0;}
```

```
double *B = new double[N];
// initialize B = h^2*f(x) = 100*exp(-10x)
for(i=0 ; i < N ; i++) {
B[i]=h*h*100*exp(-10*h*i);
}

start_alg=clock();
// forward substitution
for(i=1 ; i < N ; i++) {
A[i]=A[i]-(1/A[i-1]);
B[i]=B[i]+B[i-1]/A[i-1];
}

// backward substitution
double *X = new double[N];;
X[N-1] = B[N-1]/A[N-1];
for(i=1 ; i < N ; i++) {
X[N-1-i]=(B[N-1-i]+X[N-i])/A[N-1-i];
}
finish_alg=clock();

double *E = new double[N];
double u, x;
double maxerror = 0.0;

//Finding error at each point, and maximum error
for(i=1 ; i < N ; i++) {
x = i*h;
u = 1 - (1 - exp(-10))*x - exp(-10*x);
E[i]= log10(abs((X[i]-u)/u));
if (E[i]>maxerror){maxerror = E[i];}
}

cout << "Max Error: " << maxerror << endl;

//write results to output file
std::string N_str = std::to_string(N);
std::string filename = "N" + N_str + ".txt";
ofstream myfile (filename);
```

```
if (myfile.is_open())
{
for(int count = 0; count < N; count ++){
myfile << X[count] << "\n" ;
}
myfile.close();
}

cout << "Time Alg: " << ((finish_alg-start_alg)/CLOCKS_PER_SEC)<<endl;

delete [] A; delete [] B; delete [] X; delete [] E;

int *indx;
indx = new int[N];
double d;
double **Z;
Z=new double*[N];
for ( i = 0; i < N; i++){
Z[i] = new double[N];
}
for ( i = 0; i < N; i++){
for ( j = 0; j < N; j++){
if (i==j){
Z[i][j]=2.0;
}
else if (i==j+1 || j == i+1){
Z[i][j]=-1.0;
}
else{
Z[i][j]=0.0;
}

}
}
double *b = new double[N];
// initialize b = h^2*f(x) = 100*exp(-10x)
for(i=0 ; i < N ; i++) {
b[i]=h*h*100*exp(-10*h*i);
}
```

```
clock_t start_LU, finish_LU;
start_LU = clock();
// LU decompose the matrix
ludcmp(Z,N,indx,&d);
// Then backward substitution
lubksb(Z, N, indx, b);
finish_LU = clock();

cout << "Time LU: " << ((finish_LU-start_LU)/CLOCKS_PER_SEC)<<endl;

// Free space
delete [] b;
for ( i = 0 ; i < N; i++) delete [] Z[i];
delete [] Z;

return 0;
}
```