PHY 480 - Computational Physics
Project 1: Linear Algebra Methods

Thomas Bolden

February 12, 2016

Github Repository at https://github.com/ThomasBolden/PHY-480-Spring-2016

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

Introduction

An important part of physics is being able to efficiently solve differential equations. There are many situations in which differential equations can be solved as a system of linear equations. Such equations are called linear second-order differential equations, of the form

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + k^2(x)y = f(x)\ , \tag{1}$$

where $f(x)$ is the inhomogenous term, and k^2 is a real function.

An example of this being useful is in electromagnetism. Poisson's equation describes the electrostatic potential energy field Φ caused by a given charge density distribution ρ. The equation in three dimensions is

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

where the electrostatic potential and charge density are spherically symmetric. This allows one to simplify the equation to one dimension

$$\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left(r^2\frac{\mathrm{d}\Phi}{\mathrm{d}r}\right) = -4\pi\rho(r) \tag{3}$$

Methods

Given a differential equation of the form

$$-\frac{\mathrm{d}^2}{\mathrm{d}x^2}u(x) = f(x) \tag{4}$$

where $f(x)$ is continuous on the domain $x \in (0,1)$. We also assume the boundary conditions $u(0) = u(1) = 0$. The second derivative can be approximated as

$$u'' = \frac{u_{i+1} + u_{i-1} - 2u_i}{u^2} \tag{5}$$

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 2 & -1 & \ddots & & 0 \\ \vdots & 0 & -1 & 2 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & -1 \\ 0 & 0 & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix}\ ,\ \mathbf{v} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix}$$

Results

.

$$\tilde{di} = i/i - 1$$

relative error should give a flat line

Conclusions

.

Code

../Code/Project1.cpp

```cpp
// Project 1 – Vector and Matrix Operations

#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>
#include <string>
#include <armadillo>

using namespace std;
using namespace arma;

ofstream myfile;

// –~– Functions –~– \\

double f(double x){
    return 100*exp(–10*x);
}

double analyze(double x){
    return 1.0–(1–exp(–10))*x–exp(–10*x);
}

// –~– Main –~– \\

int main(){

    // –~– Declaration of Variables –~– \\

    double n;
    string outfilename;

    cout << "Dimensions of the nxn matrix: ";
    while(!(cin >> n)){
        cout << "Not a valid number! Try again: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    cout << "Enter a name for the output file: ";
    cin >> outfilename;

    // body of the program


```

```
46
47      // writing value to file, to be read and graphed in python later
48      myfile.open(outfilename);
49      myfile << setiosflags(ios::showpoint | ios::uppercase); //sci notation
50      myfile << n << endl;
51
52      myfile.close();
53
54      return 0;
55
56  }
```

../Code/plots.py

```
1   # From matplotlib examples
2
3   # obvi not real useful yet
4
5   import numpy as np
6   import matplotlib.pyplot as plt
7
8   plt.subplots_adjust(hspace=0.4)
9   t = np.arange(0.01, 20.0, 0.01)
10
11  # log y axis
12  plt.subplot(221)
13  plt.semilogy(t, np.exp(-t/5.0))
14  plt.title('semilogy')
15  plt.grid(True)
16
17  # log x axis
18  plt.subplot(222)
19  plt.semilogx(t, np.sin(2*np.pi*t))
20  plt.title('semilogx')
21  plt.grid(True)
22
23  # log x and y axis
24  plt.subplot(223)
25  plt.loglog(t, 20*np.exp(-t/10.0), basex=2)
26  plt.grid(True)
27  plt.title('loglog base 4 on x')
28
29  # with errorbars: clip non-positive values
30  ax = plt.subplot(224)
31  ax.set_xscale("log", nonposx='clip')
32  ax.set_yscale("log", nonposy='clip')
33
34  x = 10.0**np.linspace(0.0, 2.0, 20)
35  y = x**2.0
36  plt.errorbar(x, y, xerr=0.1*x, yerr=5.0 + 0.75*y)
37  ax.set_ylim(ymin=0.1)
38  ax.set_title('Errorbars go negative')
39
40
```

```
41 plt.show()
```

References

[1] M. Hjorth-Jensen, *Computational Physics*, University of Oslo (2013).

[2] W. McLean, *Poisson Solvers*, Northwestern University (2004).