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 is Poisson

Methods

Given a differential equation of the form

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

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

$$u'' = \frac{u_{i+1} + u_{i-1} - 2u_i}{u^2} \tag{3}$$

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

Results

.

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

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



    // writing value to file, to be read and graphed in python later
    myfile.open(outfilename);
    myfile << setiosflags(ios::showpoint | ios::uppercase); //sci notation
    myfile << n << endl;
```

```

    myfile.close();

    return 0;

}
```

../Code/plots.py

```python
# From matplotlib examples

# obvi not real useful yet

import numpy as np
import matplotlib.pyplot as plt

plt.subplots_adjust(hspace=0.4)
t = np.arange(0.01, 20.0, 0.01)

# log y axis
plt.subplot(221)
plt.semilogy(t, np.exp(-t/5.0))
plt.title('semilogy')
plt.grid(True)

# log x axis
plt.subplot(222)
plt.semilogx(t, np.sin(2*np.pi*t))
plt.title('semilogx')
plt.grid(True)

# log x and y axis
plt.subplot(223)
plt.loglog(t, 20*np.exp(-t/10.0), basex=2)
plt.grid(True)
plt.title('loglog base 4 on x')

# with errorbars: clip non-positive values
ax = plt.subplot(224)
ax.set_xscale("log", nonposx='clip')
ax.set_yscale("log", nonposy='clip')

x = 10.0**np.linspace(0.0, 2.0, 20)
y = x**2.0
plt.errorbar(x, y, xerr=0.1*x, yerr=5.0 + 0.75*y)
ax.set_ylim(ymin=0.1)
ax.set_title('Errorbars go negative')


plt.show()
```

References

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

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