AE 5327 - Homework 2

James Grisham

10/21/2013

Problem Statement

Given the following model equation:

$$\frac{d^2\phi}{dx^2} + \phi = x^2$$

Perform the following tasks:

a. Derive the exact solution to the equation for the boundary conditions

$$\phi(0) = \frac{d\phi(1)}{dx} = 0$$

b. Write a program to solve the equation approximately using second-order accurate finite differencing. Use Thomas' algorithm to solve the linear system of equations.

c. On the same graph, plot the exact and approximate solutions for $\phi(x)$ vs x.

d. Plot the $\mathcal{L}^2$-norm of the truncation error vs Δx. Can you use this plot to confirm that the method is truly second-order accurate? (HINT: try using a log-log plot).

Solution

Exact Solution

The differential equation can be classified as a linear, second-order, nonhomogeneous ODE. The analytical solution is determine using the method of undetermined coefficients. First, the complimentary solution is determined by solving the homogeneous problem given by

$$\frac{d^2\phi}{dx^2} + \phi = 0 \tag{1}$$

The characteristic equation is

$$s^2 + 1 = 0$$

Therefore, the roots of the characteristic equation are both imaginary, yielding a solution of

$$\boxed{\phi_c(x) = C_1 \cos x + C_2 \sin x} \tag{2}$$

Now the particular solution is determined by assuming the form of the solution, taking derivatives and plugging in. Let

$$\phi_p(x) = A_1 x^2 + A_2 x + A_2 \tag{3a}$$

$$\frac{d\phi}{dx} = 2A_1 x + A_2 \tag{3b}$$

$$\frac{d^2\phi}{dx^2} = 2A_1 \tag{3c}$$

Inserting (3) into the original ODE

$$A_1 x^2 + A_2 x + (A_3 + 2A_1) = x^2 \tag{4}$$

Now, from (4),

$$A_1 = 1 \tag{5a}$$

$$A_2 = 0 \tag{5b}$$

$$A_3 = -2 \tag{5c}$$

Therefore, the particular solution is given by

$$\boxed{\phi_p(x) = x^2 - 2} \tag{6}$$

The total solution is determined by adding the complimentary and particular solutions, i.e.,

$$\phi(x) = \phi_c(x) + \phi_p(x)$$

Therefore, the exact solution is

$$\phi(x) = C_1 \cos x + C_2 \sin x + x^2 - 2 \tag{7}$$

where the constants C_1 and C_2 are determined by applying the boundary conditions given in the problem statement. After applying the boundary conditions, the result is

$$\boxed{\phi(x) = 2\cos x + \frac{2\left(\sin(1) - 1\right)}{\cos(1)} \sin x + x^2 - 2} \tag{8}$$

Thomas' Algorithm

The differential equation given in (1) can be expressed as a difference equation by using the central difference discretization scheme. That is,

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\Delta x)^2} + \phi_i = {x_i}^2 \tag{9}$$

where $i = 1, 2, \ldots, N$, where N is equal to the number of grid points. This equation can be rearranged as

$$\frac{1}{(\Delta x)^2}\,\phi_{i-1} + \left(1 - \frac{2}{(\Delta x)^2}\right)\,\phi_i + \frac{1}{(\Delta x)^2}\,\phi_{i+1} = {x_i}^2 \tag{10}$$

Now, (10) can be written as

$$b_i\,\phi_{i-1} + d_i\,\phi_i + a_i\,\phi_{i+1} = c_i \tag{11}$$

where

$$\begin{aligned} a_i &= \frac{1}{(\Delta x)^2} \\ b_i &= \frac{1}{(\Delta x)^2} \\ c_i &= {x_i}^2 \\ d_i &= \left(1 - \frac{2}{(\Delta x)^2}\right) \end{aligned}$$

To enforce the Dirichlet boundary condition, $\phi(0) = 0$, the solution of the first row is set so that the boundary condition is satisfied (i.e., $a_1 = 0$, $d_1 = 1$, $c_1 = 0$). The result is $\phi_1 = 0$, which is simply a statement of the boundary condition.

To enforce the Neumann boundary condition $d\phi(1)/dx = 0$, the a second-order accurate finite difference representation of the first derivative of $\phi(x)$ is used along with the difference equation given in Equation (9).

Using central-differencing,

$$\left.\frac{d\phi}{dx}\right|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} + O[(\Delta x)^2] \tag{12}$$

Evaluating (12) at the last grid point, N, where $x = 1$,

$$\left.\frac{d\phi}{dx}\right|_{i=N} \approx \frac{\phi_{N+1} - \phi_{N-1}}{2\Delta x} \tag{13}$$

From the boundary condition, this value must be equal to zero. That is

$$\frac{\phi_{N+1} - \phi_{N-1}}{2\Delta x} = 0$$

Which yields

$$\phi_{N+1} = \phi_{N-1} \tag{14}$$

Now, evaluating the original difference equation, Eq. (9), at $i = N$,

$$\frac{\phi_{N+1} - 2\phi_N + \phi_{N-1}}{(\Delta x)^2} + \phi_N = {x_N}^2 \tag{15}$$

However, ϕ_{N+1} is not in the computational domain. The relation in (14) is used to mitigate this issue. Inserting (14) into (15),

$$\frac{2}{(\Delta x)^2}\,\phi_{N-1} + \left(1 - \frac{2}{(\Delta x)^2}\right)\,\phi_N = {x_N}^2 \tag{16}$$

Therefore,

$$\begin{aligned} b_N &= \frac{2}{(\Delta x)^2} \\ d_N &= 1 - \frac{2}{(\Delta x)^2} \\ c_N &= {x_N}^2 \end{aligned}$$

Now all of the necessary information is known and the process can be programmed. First, the terms a_i, b_i, c_i and d_i are filled in. Second, the tridiagonal matrix is converted to an upper triangular matrix using the Thomas algorithm. Finally, the solution at each point is computed using back substitution.

Plots

The problem was solved using MATLAB and C++ with Python for plotting. The truncation error is calculated by finding the difference between the exact and numberical solution as follows:

$$\mathrm{TE}_i = \phi(x_i) - \phi_i$$

where $\phi(x_i)$ is the exact solution at the i-th point and ϕ_i is the approximate solution at the i-th point. The $\mathcal{L}^2$ norm of the truncation error is approximated as:

$$\mathcal{L}^2 \approx \sqrt{\sum_{i=1}^{N} (\mathrm{TE}_i)^2 \, \Delta x}$$

The order of accuracy of the discretization scheme was determined by using a linear regression to find the slope of the data in Figures 2 and 4. In both cases, the order of accuracy was approximately 2.

MATLAB Results

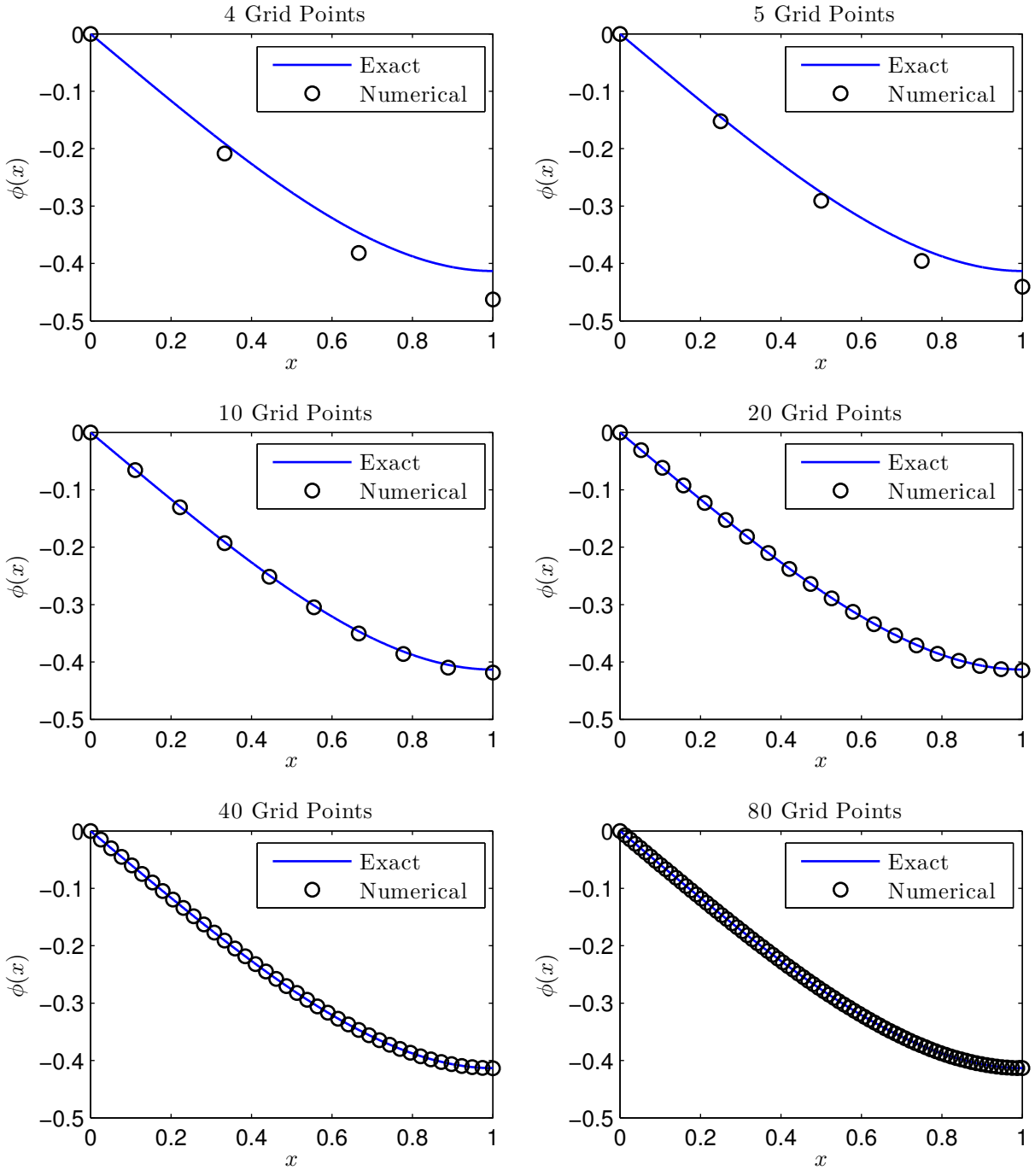

Figure 1. MATLAB solution with varying levels of grid resolution.

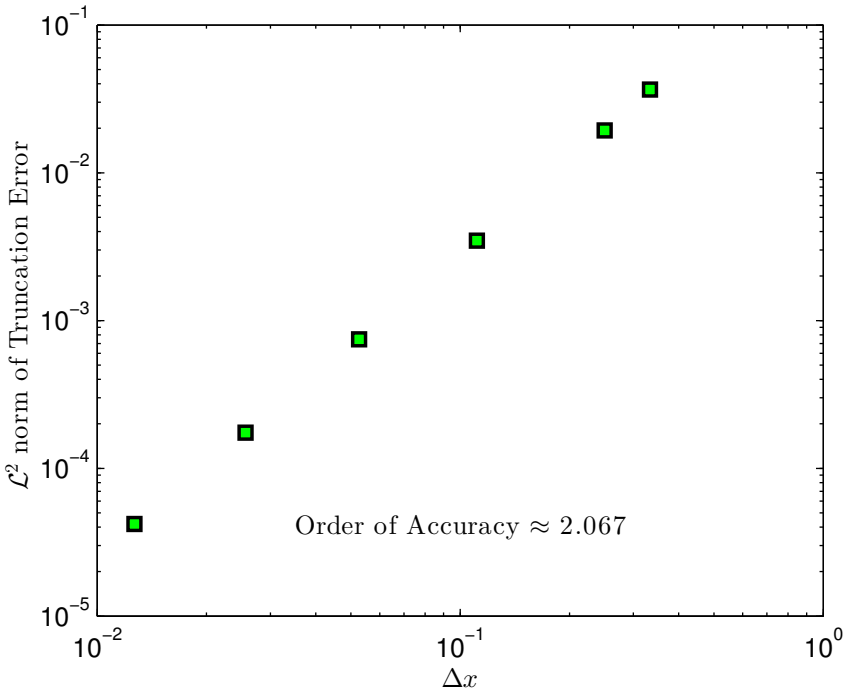

Figure 2. Plot of $\mathcal{L}^2$ norm of truncation error vs grid resolution (Δx).

C++ Results

A C++ program was written to solve the problem. The results were written out to data files and then post-processed using Python. See appendix for code.

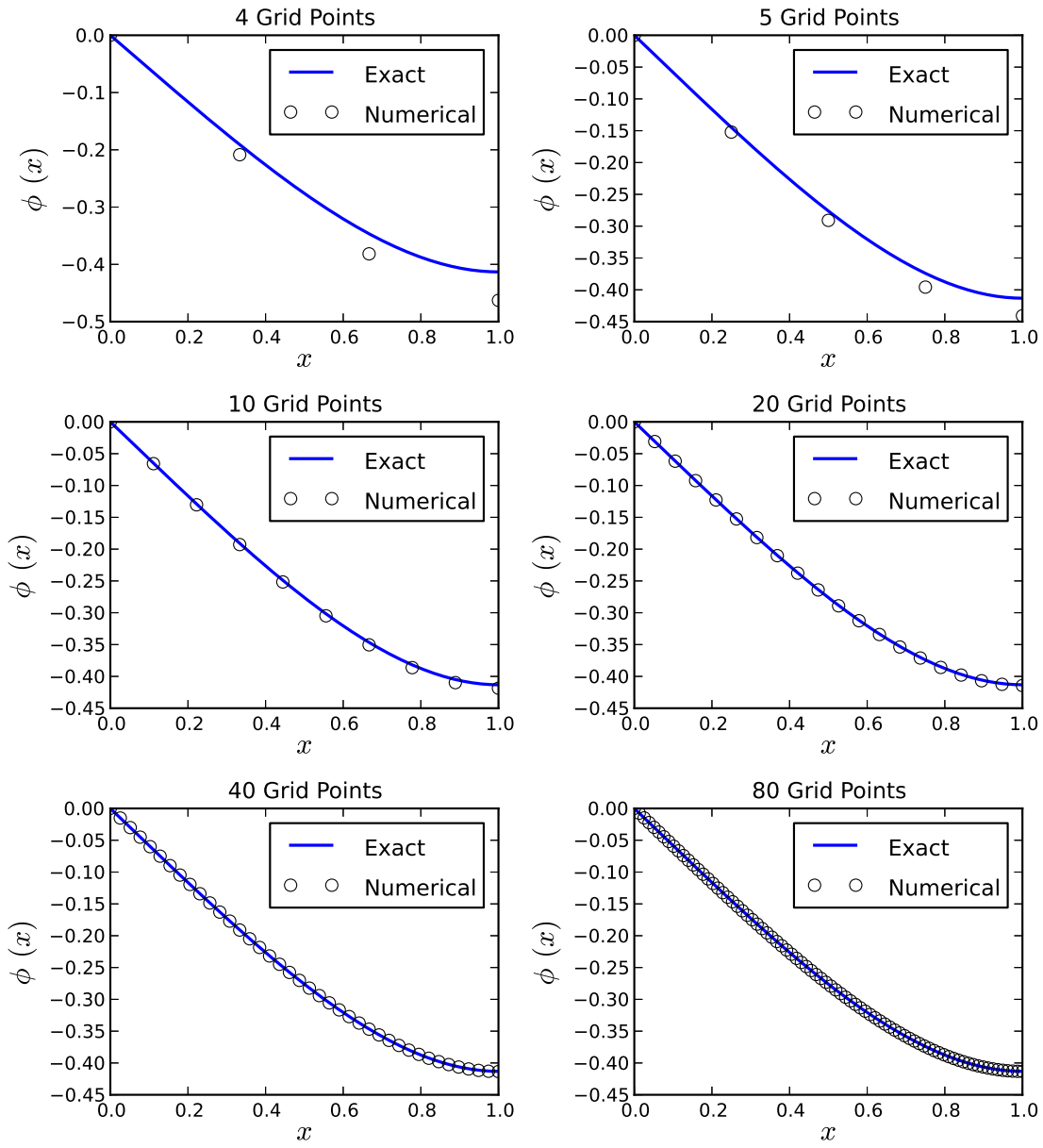

Figure 3. C++ solution with varying levels of grid resolution.

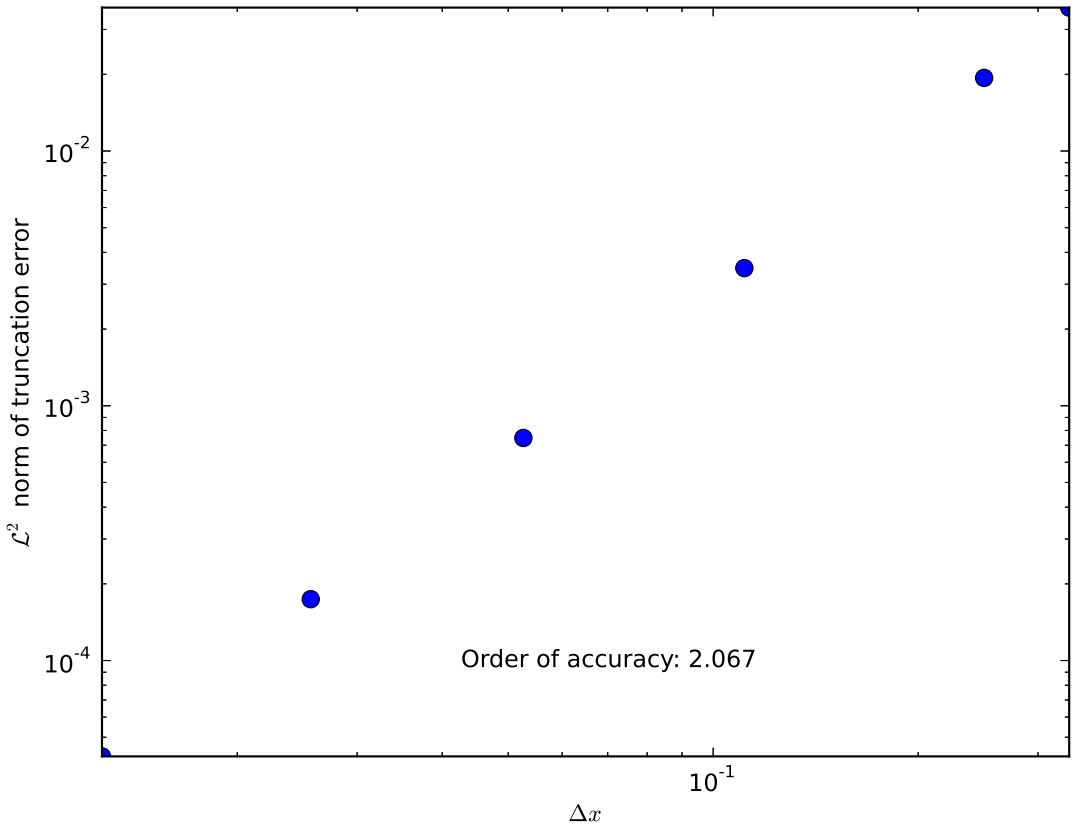

Figure 4. Plot of $\mathcal{L}^2$ norm of truncation error vs grid resolution (Δx).

Appendices

MATLAB code

Main Script

```
%=========================================================================
%% Clearing workspace
%=========================================================================

clc,clear,close all

%=========================================================================
```

```
%% Inputs
%==========================================================================

xStart = 0;
xStop = 1;
nPoints = [4 5 10 20 40 80];

% Path for images
ImgPath = '../Images/';

%==========================================================================
%% Exact solution
%==========================================================================

x_e = linspace(0,1,1e3);
phi_e = 2*cos(x_e) + 2*(sin(1) - 1)/cos(1)*sin(x_e) + x_e.^2 - 2;

%==========================================================================
%% Calling Thomas to determine numerical solution
%==========================================================================

% Setting up figure
figure('Position',[2250 40 660 730])

% Creating structure in which solutions will be stored
soln = struct('x',[],'phi',[],'dx',[],'nPoints',[],'TE',[],'L2',[]);
k = 1;

for n = nPoints

    [soln(k).x,soln(k).phi,soln(k).dx] = Thomas(n);
    soln(k).nPoints = n;

    % Plotting solution
    subplot(3,2,k);
    plot(x_e,phi_e,'-b',soln(k).x,soln(k).phi,'ok','LineWidth',1)

    % Adding labels and legend
    [lh1,lh2] = legend('Exact','Numerical');
    txtobj = findobj(lh2,'type','text');
    set(txtobj,'Interpreter','LaTeX')
    xlabel('$x$','Interpreter','LaTeX')
    ylabel('$\phi(x)$','Interpreter','LaTeX')
    title([num2str(n),' Grid Points'],'Interpreter','LaTeX')

    % Improving aesthetics
    set(gca,'YLim',[-0.5 0])
    loc = get(lh1,'Position');
    set(lh1,'Position',[loc(1)-0.025 loc(2) loc(3)+0.025 loc(4)])

    % Incrementing counter
    k = k + 1;

end

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw2MATLABSoln.eps'])

```

```
%==========================================================================
%% Plotting L^2 norm vs truncation error
%==========================================================================

% Anonymous function for exact solution
phi = @(x) 2*cos(x) + 2*(sin(1) - 1)/cos(1)*sin(x) + x.^2 - 2;

% Determining norm of the truncation error
for I = 1:numel(nPoints)

    for J = 1:soln(I).nPoints

        % Separating Delta x
        Dx = soln(I).dx;

        % Simplifying referencing
        if J > 1
            x_jm1 = soln(I).x(J-1);
        end
        x_j = soln(I).x(J);
        if J < soln(I).nPoints
            x_jp1 = soln(I).x(J+1);
        end

        % According to class
        if J == 1
            soln(I).TE1(J) = 0;
        elseif J > 1 && J < soln(I).nPoints
            soln(I).TE1(J) = -((phi(x_jp1) - 2*phi(x_j) + phi(x_jm1))/ ...
                (Dx)^2 + phi(x_j) - x_j^2);
        else
            soln(I).TE1(J) = -((2*(phi(x_jm1)) -  ...
                2*phi(x_j))/(Dx)^2 + phi(x_j) ...
                - x_j^2);
        end

        % According to the book (overwriting previous values)
        soln(I).TE(J) = phi(soln(I).x(J)) - soln(I).phi(J);

    end

    % Calculating the L^2 norm of the truncation error
    sum = 0;
    for J = 1:soln(I).nPoints

        sum = sum + soln(I).TE(J)^2*soln(I).dx;

    end

    soln(I).L2 = sqrt(sum);

end

% Plot truncation error
figure('Position',[2300 130 500 340])
plot(soln(end).x,soln(end).TE,'-k',soln(end).x,abs(soln(end).TE1),'--r',...
    'LineWidth',1.3)
set(gca,'YScale','log')

```

```
% Adding annotations and improving aesthetics
xlabel('$x$','Interpreter','LaTeX')
ylabel('Truncation Error','Interpreter','LaTeX')
[lh1,lh2] = legend('Method 1','Method 2');
set(lh1,'Location','North')
txtobj = findall(lh2,'type','text');
set(txtobj,'Interpreter','LaTeX')
lPos = get(lh1,'Position');
set(lh1,'Position',[lPos(1) lPos(2) 1.1*lPos(3) lPos(4)])

% Determining the slope
logDx = zeros(1,numel(nPoints));
logTE = logDx;
for n = 1:numel(nPoints)

    logDx(n) = log10(soln(n).dx);
    logTE(n) = log10(soln(n).L2);

end

% Doing linear regression
A(:,1) = logDx;
A(:,2) = 1;
b(:,1) = logTE;
xstar = A\b;

% Plotting
figure('Position',[2300 130 500 340])
hold on
for n = 1:numel(nPoints)

    plot(soln(n).dx,soln(n).L2,'sk','LineWidth',1.3,'MarkerFaceColor', ...
        'g')

end

% Adding labels
xlim = get(gca,'XLim');
ylim = get(gca,'YLim');
text(0.1*xlim(2),0.001*ylim(2),['Order of Accuracy $\approx$ ', ...
    num2str(xstar(1))],'Interpreter','LaTeX')
xlabel('$\Delta x$','Interpreter','LaTeX')
ylabel('$\mathcal{L}^2$ norm of Truncation Error','Interpreter','LaTeX')

% Improving aesthetics
set(gca,'box','on','XScale','log','YScale','log')

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw2MATLABL2norm.eps'])
```

Thomas function

```
function [x_n,phi_n,Dx] = Thomas(nPoints,plotDec)
% Thomas(nPoints) uses finite difference discretization to solve the
```

```
% following differential equation: phi_xx + phi = x^2.
%
%   Thomas(nPoints,plotDec) makes use of finite difference discretization
%   to solve a differential equation given by
%
%                       phi_xx + phi = x^2
%
%   with boundary conditions phi(0) = 0 and phi_x(1) = 0.  The tridiagonal
%   matrix is converted to an upperdiagonal matrix using the Thomas
%   algorithm.  Then, the solution at each grid point is found using back
%   substitution.
%
%   Inputs
%           nPoints:    number of grid points.
%           plotDec:    optional input that allows user to plot the
%                       numerical and exact solutions.
%
%   Outputs
%           x_n:        coordinates of grid points.
%           phi_n:      numerical solution.
%           Dx:         distance between grid points.

%=========================================================================
% Inputs
%=========================================================================

if nargin < 2
    plotDec = 'no plot';
end

xStart = 0;
xStop = 1;

%=========================================================================
% Exact solution
%=========================================================================

x_e = linspace(0,1,1e3);
phi_e = 2*cos(x_e) + 2*(sin(1) - 1)/cos(1)*sin(x_e) + x_e.^2 - 2;

%=========================================================================
% Numerical solution
%=========================================================================

% Discretization
Dx = (xStop - xStart)/(nPoints - 1);
x_n = xStart:Dx:xStop;

% Preallocating
a = zeros(1,numel(Dx));
d = zeros(1,numel(Dx));
b = zeros(1,numel(Dx));
c = zeros(1,numel(Dx));

% Filling in matrix
for I = 1:numel(x_n)

    if I == 1

```

```
        % Enforcing Dirichlet BC
        a(I) = 0;
        d(I) = 1;
        c(I) = 0;

    elseif I > 1 && I < numel(x_n)

        % Interior points
        a(I) = 1/Dx^2;
        b(I) = 1/Dx^2;
        d(I) = (1 - 2/Dx^2);
        c(I) = x_n(I)^2;

    else

        % Enforcing Neumann BC
        b(I) = 2;
        d(I) = Dx^2 - 2;
        c(I) = Dx^2*x_n(I)^2;

    end

end

% Using Thomas algorithm to change tridiagonal matrix to upper triangular
for J = 2:nPoints

    d(J) = d(J) - b(J)/d(J-1)*a(J-1);
    c(J) = c(J) - b(J)/d(J-1)*c(J-1);

end

% Using back substitution to determine the unknowns
phi_n = zeros(1,nPoints);
phi_n(nPoints) = c(nPoints)/d(nPoints);
for n = 2:nPoints

    % Counter
    K = nPoints - n + 1;

    % Back substitution
    phi_n(K) = (c(K) - a(K)*phi_n(K+1))/(d(K));

end

%=========================================================================
% Plotting solution
%=========================================================================

if strcmp(plotDec,'plot')

    plot(x_e,phi_e,'-b',x_n,phi_n,'ok','LineWidth',1.5)
    xlabel('x')
    ylabel('\phi')
    legend('Exact Solution','Numerical Solution')

end

end
```

C++ code

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

/*------------------------------------------------------------------------------
   This program must be compiled using g++ -o Hw2 -std=c++0x Hw2.cpp because
   of a num to string conversion that requires C++ 11 capabilities.
------------------------------------------------------------------------------*/

// Function to return the exact solution

double phiExact( double x_i)
{
  return 2.0*cos(x_i) + 2.0*(sin(1.0) - 1.0)/cos(1.0)*sin(x_i) +
   x_i*x_i - 2.0;
}

int main() {

  //============================================================================
  // Declaring variables
  //============================================================================
  int j,k;
  double xStart,xStop,dx,sum,L2;
  vector<double> x,phi,a,b,c,d,TE;

  cout << "\n=========================================================" << endl;
  cout << "AE 5327 -- Homework 2 " << endl;
  cout << "=========================================================\n" << endl;

  //============================================================================
  // Asking user to input number of grid points
  //============================================================================

  int nPoints;
  cout << "Enter the number of grid points: ";
  cin >> nPoints;

  // Preallocating vectors for nPoints
  x.reserve(nPoints);
  phi.reserve(nPoints);
  a.reserve(nPoints);
  b.reserve(nPoints);
  c.reserve(nPoints);
  d.reserve(nPoints);
```

```
52    TE.reserve(nPoints);
53
54    //========================================================================
55    // Discretizing the computational domain
56    //========================================================================
57
58    xStart = 0.0;
59    xStop = 1.0;
60    dx = (xStop - xStart)/((double)nPoints - 1.0);
61
62    // filling in values of x vector
63    for(int i=0; i<nPoints; i++){
64      if (i==0){
65      x[i] = xStart;
66      }
67      else {
68      x[i] = x[i-1] + dx;
69      }
70    }
71
72    //========================================================================
73    // Setting up matrix
74    //========================================================================
75
76    for(int i=0; i<nPoints; i++){
77
78      if (i == 0) {
79
80      // Enforcing Dirichlet Boundary Condition
81      a[i] = 0.0;
82      c[i] = 0.0;
83      d[i] = 1.0;
84
85      }
86      else if ((i > 0) && (i < (nPoints-1))){
87
88      // Interior points (from finite difference)
89      a[i] = 1.0/(dx*dx);
90      b[i] = 1.0/(dx*dx);
91      c[i] = x[i]*x[i];
92      d[i] = (1.0 - 2.0/(dx*dx));
93
94      }
95      else {
96
97      // Enforcing Neumann BC
98      b[i] = 2.0/(dx*dx);
99      c[i] = x[i]*x[i];
100     d[i] = 1.0 - 2.0/(dx*dx);
101
102     }
103
104   }
105
106   //========================================================================
107   // Use Thomas' algorithm to solve system of equations
108   //========================================================================
109
110   // Changing tridiagonal matrix to upper triangular
```

```
111   for (int index=0; index<(nPoints-1); index++){
112
113     // Setting up index
114     j = 1 + index;
115
116     // Using Thomas algorithm
117     d[j] = d[j] - b[j]/d[j-1]*a[j-1];
118     c[j] = c[j] - b[j]/d[j-1]*c[j-1];
119
120   }
121
122   //=========================================================================
123   // Using back substitution to determine the solution
124   //=========================================================================
125
126   // Assigning the solution for the last row
127   phi[nPoints-1] = c[nPoints-1]/d[nPoints-1];
128
129   // Iterating through and calculating unknowns
130   for (int N=0; N<(nPoints-1); N++){
131
132     // Setting up index
133     k = nPoints - N - 2;
134
135     // Back substitution
136     phi[k] = (c[k] - a[k]*phi[k+1])/d[k];
137
138   }
139
140   //=========================================================================
141   // Calculating the truncation error and the norm
142   //=========================================================================
143
144   // Truncation error
145   for (int i=0; i<nPoints; i++){
146
147     TE[i] = phiExact(x[i]) - phi[i];
148
149   }
150
151   // L^2 norm
152   sum = 0.0;
153   for (int i=0;i<nPoints; i++){
154
155     sum += TE[i]*TE[i]*dx;
156
157   }
158
159   // Taking the square root of the sum
160   L2 = sqrt(sum);
161
162   // Outputting the L2 norm
163   cout << "\nL2 norm of truncation error: " << L2 << "\n\n";
164
165   //=========================================================================
166   // Outputting results to data file
167   //=========================================================================
168
169   // Constructing file names
```

```
  string fname = "Hw2_" + to_string((int)nPoints) + "pts.dat";
  string efname = "Hw2_" + to_string((int)nPoints) + "error.dat";

  // Outputting name of data files
  cout << "Results in file named:        " << fname << endl;
  cout << "Error results in file named:  " << efname << "\n\n";

  // Opening data file for results
  ofstream data_file("./DataFiles/"+fname);

  // Writing data to data file
  for (int i=0; i<nPoints; i++){
    data_file << setprecision(10) << x[i] << "," << phi[i]  << "," <<
    TE[i] << endl;
  }
  data_file.close();

  // Opening data file for error results
  ofstream edata_file("./DataFiles/"+efname);

  // Writing data to file
  edata_file << setprecision(10) << dx << "," << L2 << endl;
  edata_file.close();

  return 0;
}
```

Python code

```
#!/usr/bin/python -tt

import sys
import numpy as np
import matplotlib.pyplot as plt
import math
import ast

#=============================================================
# Setting up
#=============================================================

# Setting up font
font = {'size'   : 14}
nGridPts = [4,5,10,20,40,80]

# Setting up exact solution for comparison
def phi_e(x):
    return 2*np.cos(x) + 2*(np.sin(1) - 1)/np.cos(1)*np.sin(x) + x**2 - 2

# Setting up figure
fig = plt.figure()

# Setting up lists for dx vector and L2 vector
dx = [0]*len(nGridPts)
```

```
L2 = [0]*len(nGridPts)

#================================================================
# Iterating through data sets and plotting solution
#================================================================

# Setting counter
i = 1

# Iterating through data
for nPts in nGridPts:

    # Adding subplots
    actAx=fig.add_subplot(3,2,i)

    # Importing data
    fname = './DataFiles/Hw2_'+str(nPts)+'pts.dat'
    efname = './DataFiles/Hw2_'+str(nPts)+'error.dat'
    edata = np.genfromtxt(efname,delimiter=',')
    data = np.genfromtxt(fname,delimiter=',',dtype=float)

    # Separating data
    dx[i-1] = edata[0]
    L2[i-1] = edata[1]
    x_n = data[:,0]
    phi_n = data[:,1]
    TE = data[:,2]

    # Plotting exact solution
    x_e = np.arange(0,1,0.001)
    p_e, = actAx.plot(x_e,phi_e(x_e),'b',linewidth=1.5)

    # Plotting numerical solution
    p_n, = actAx.plot(x_n,phi_n,'o',linewidth=1.5,mfc='none')

    # Adding labels
    plt.title(str(nPts)+' Grid Points')
    plt.xlabel('$x$',fontdict=font)
    plt.ylabel('$\phi\,(x)$',fontdict=font)
    legend = actAx.legend([p_e,p_n],["Exact","Numerical"])

    # Incrementing counter
    i = i + 1

# Improving aesthetics
plt.rcParams.update({'font.size': 9})
fig.subplots_adjust(wspace=0.35, hspace=0.35)
fSize = fig.get_size_inches()
fig.set_size_inches((fSize[0],fSize[1]*1.5))

# Saving plot
saveName = '../Images/Hw2CPPSoln.eps'
fig.savefig(saveName)

#================================================================
# Plotting L2 norm vs dx
#================================================================

# Setting up figure
```

```
efig = plt.figure()
plt.plot(dx,L2,marker='o',color='b',linestyle='',markersize=8)

# use log scales
plt.axis('tight')
plt.gca().set_xscale('log')
plt.gca().set_yscale('log')

# Adding labels
plt.xlabel('$\Delta x$')
plt.ylabel('$\mathcal{L}^2$ norm of truncation error')

# Improving aesthetics
plt.rcParams.update({'font.size': 11})

#=================================================================
# Determining order of accuracy
#=================================================================

# Taking the logarithms
dxlog = [0]*len(dx)
L2log = [0]*len(L2)
i = 0
for n in dx:
    dxlog[i] = math.log10(dx[i])
    L2log[i] = math.log10(L2[i])
    i = i + 1

# Finding a linear regression
A = np.array([dxlog, np.ones(len(dxlog))])
xstar = np.linalg.lstsq(A.T,L2log)[0]
print '\nOrder of accuracy is approximately %1.3f\n' % xstar[0]

# Annotating the plot
ax = plt.gca()
ann_str = 'Order of accuracy: ' + str(round(xstar[0],3))
ax.text(0.07,0.0001,ann_str,ha='center',va='center')

# Saving plot
saveName = '../Images/Hw2CPPL2Soln.eps'
efig.savefig(saveName)

# Showing plots
plt.show()
```