Indian Institute of Technology, Madras

Finite Element Methods for Design

ED4030

Tutorial 1

Submitted by:
Athul Vijayan
ED11B004

August 17, 2014

1 Problem 5

We need to solve the differential equation

$$\frac{d\phi}{dx} = 3\phi + 4 \qquad | \ \phi = 0 \text{ at } x = 0$$

The exact solution for this can be found to be by solving the differential equation:

$$\phi = \frac{4}{3}(e^{3x} - 1)$$

But here we will use concepts of numerical methods to solve the differential equation to find values of ϕ at different points of x. For different aproaches, the aproximation we make are:

1. Forward difference
$$(\frac{d\phi}{dx})_n \approx \frac{\phi_{n+1} - \phi_n}{\Delta x}$$
2. Backward difference
$$(\frac{d\phi}{dx})_n \approx \frac{\phi_n - \phi_{n-1}}{\Delta x}$$
3. Central difference
$$(\frac{d\phi}{dx})_n \approx \frac{\phi_{n+1} - \phi_{n-1}}{2\Delta x}$$

We use these aproximations to solve the problems in this part.

1.1 Part A

We use 20 grid refinements to study error at $x = \frac{1}{3}$ and $x = \frac{2}{3}$.

$$\Delta x = [\frac{1}{3}, \frac{1}{6}, \frac{1}{9}, ..., \frac{1}{30}]$$

We use forward difference for this part of question. From the definition of forward difference:

$$\begin{aligned} (\frac{d\phi}{dx})_n &\approx \frac{\phi_{n+1} - \phi_n}{\Delta x} \\ \frac{\phi_{n+1} - \phi_n}{\Delta x} &= 3\phi + 4 \\ \phi_{n+1} - (3\Delta x + 1)\phi - 4\Delta x &= 0 \end{aligned} \tag{1}$$

When we write out all the equations in recurrence formula, we get a system of linear equations with n variables.

$$\text{AX} = \text{B}$$

Where matrices A , X and B are:

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -(3\Delta x + 1) & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & & \\ 0 & 0 & 0 & \cdots & -(3\Delta x + 1) & 1 \end{pmatrix}$$

and

$$B = \begin{bmatrix} 4\Delta x \\ 4\Delta x \\ \vdots \\ 4\Delta x \end{bmatrix}, \qquad X = \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}$$

We solve the above equations using the code:

```
numGrids = 20;                       % number of different grids over which solution is sought
numFirstGridElems = 3; xVal = 1/3; % number of elements in the first (coarsest) grid
xArray = cell(numGrids,1);          % array storing the x values at grid locations
phiFD = cell(numGrids,1);           % solution vectors
midValsFD = zeros(numGrids,1);      % an array storing all phi values at x , FD
stepVals = zeros(numGrids,1);       % array storing delta x values for different grids
phi_0 = 0;                          % phi b.c. at x = 0


for i = 1:numGrids
    numElem = numFirstGridElems*2^(i-1);  % number of elements
```

```
    numPoints = numElem;                    % number of grid points, N in class
    xArray{i} = linspace(0,1,numPoints+1)'; % {} references Cell elements, () references
        array or matrix elements
    xArray{i} = xArray{i}(2:numPoints+1); % [0,1] divided into numPoints interior grid
        points
    deltaX = 1/numElem;                     % step size
    %
    diagVec = ones(numPoints,1);            % main diagonal of length n
    offdiagVec = -1.0*(3*deltaX + 1)*ones(numPoints-1,1); % off-diagonal vectors of length
        (n-1)
    myMat = gallery('tridiag',offdiagVec,diagVec,zeros(numPoints-1,1)); % matrix of
        coefficients
    myVec = 4*deltaX*ones(numPoints,1);   % rhs vector
    myVec(1) = myVec(1) + phi_0*(3*deltaX + 1);
    phiFD{i} = myMat\myVec;                 % solution obtained by matrix inversion
    index = find(xArray{i}==xVal);          % index returns the location of Xval in the array
        xArray{i}
    midValsFD(i) = phiFD{i}(index);         % pick the computed value at x = xVal;
    stepVals(i) = deltaX;                   % store step size for plotting later

    errorAtX1(i) = midValsFD(i)-4/3*(exp(3*xVal)-1); % calculate the RMS error in every
        iteration of delta x=1/3

    xVal = 2/3;
    index = find(xArray{i}==xVal);          % index returns the location of Xval in the array
        xArray{i}
    midValsFD(i) = phiFD{i}(index);         % pick the computed value at x = xVal;
    errorAtX2(i) = midValsFD(i)-4/3*(exp(3*xVal)-1); % error in every iteration of delta
        x2/3
end
```

Now we use polyfit to find the slope of $log(E) Vs log(\Delta x)$.

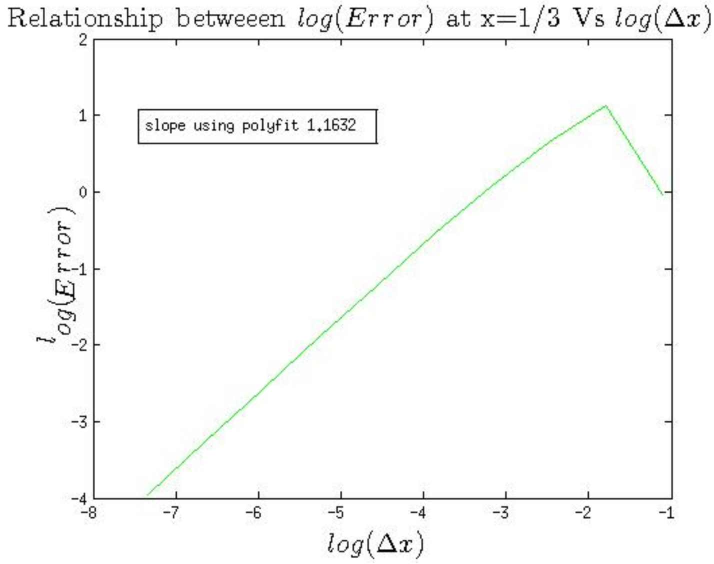
```
     % Now we have error at x1 and x2 for different delta x. we plot log(error) Vs
         log(delta x) for checking the order of error
    %% plot FD fit and true solution
    power = polyfit(log(abs(errorAtX1')), log(stepVals), 1)
```

and plot this using the following code

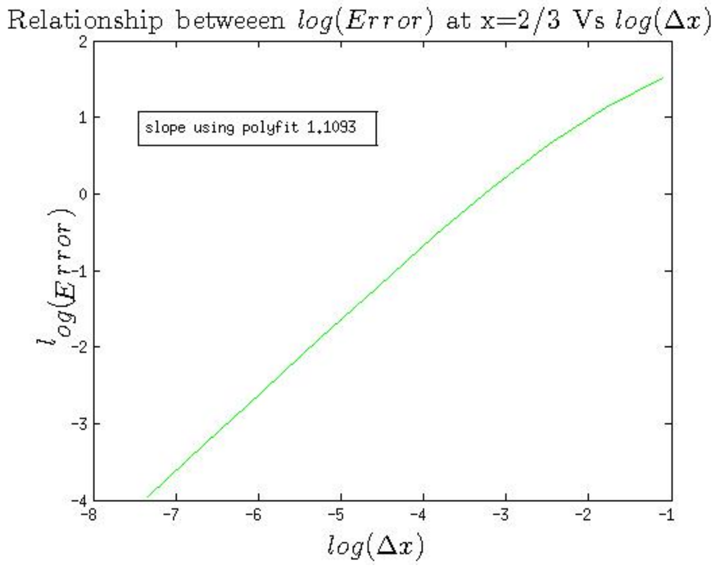
```
figure;
plot(log(stepVals), log(abs(errorAtX1')), 'g')
title('Relationship betweeen $$log(Error)$$ at x=$$1/3$$ Vs $$log(\Delta x)$$', ...
    'Interpreter','LaTeX','FontSize',18)   % plot title at the top
xlabel('$$log(\Delta x)$$',...             % Label for x-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
ylabel('$$log(Error)$$',...        % Label for y-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
annotation('textbox', [.2 .7 .1 .1], 'String', ['slope using polyfit ',num2str(power(1))]);
```

The respective plots are

x=$\frac{1}{3}$:



x=$\frac{2}{3}$:

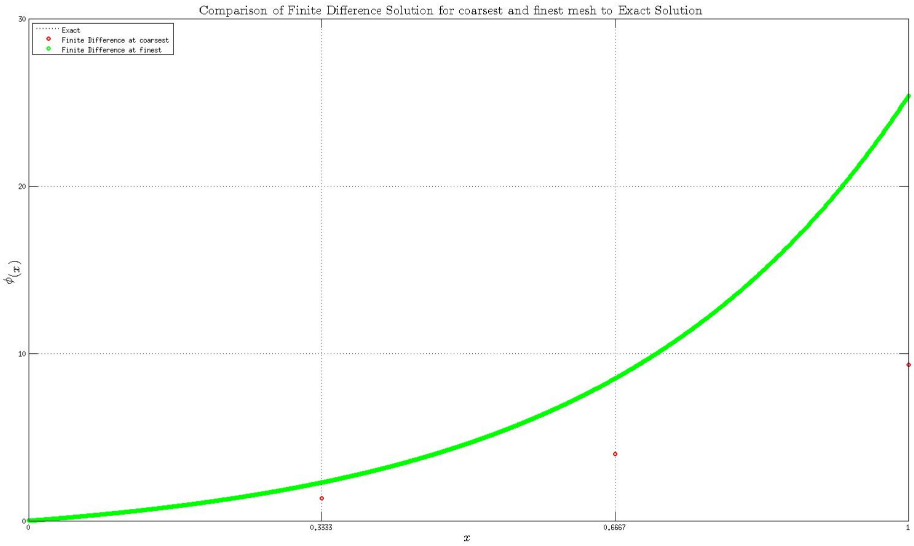


finally slopes of $log(E) Vs log(\Delta x)$ using polyfit are:

Item	
X	slope
1/3	1.1632
2/3	1.1093

1.2 Part B

We can compare the Finite Difference Solution with exact method solution by inspecting their plots. Below is the plot of $\phi(x)$ with x:



As the value of Δx decreases, the error also decreases. This happens because the definition of our $fracd\phi dx)_n$ is

$$(\frac{d\phi}{dx})_n = \lim_{x\to 0} \frac{\phi_{n+1}-\phi_n}{\Delta x}$$

But as Δx becomes a finite value, the aproximation goes away from the exact solution.

1.3 Part C

Only difference in backward difference will be the change of definition of $\frac{d\phi}{dx}$. In Backward difference, we use the formula:

$$(\frac{d\phi}{dx})_n \approx \frac{\phi_n-\phi_{n-1}}{\Delta x}$$

We use backward difference for this part of question. From the definition of backward difference:

$$\begin{aligned}(\frac{d\phi}{dx})_n &\approx \frac{\phi_n-\phi_{n-1}}{\Delta x} \qquad (2)\\ \frac{\phi_n-\phi_{n-1}}{\Delta x} &= 3\phi_n + 4\\ -\phi_{n-1} + (1-3\Delta x)\phi - 4\Delta x &= 0\end{aligned}$$

When we write out all the equations in recurrence formula, we get a system of linear equations with n variables.

$$\text{AX} = \text{B}$$

Where matrices A , X and B are:

$$A = \begin{pmatrix} (1-3\Delta x) & 0 & 0 & \cdots & 0 & 0\\ -1 & (1-3\Delta x) & 0 & \cdots & 0 & 0\\ \vdots & \vdots & \ddots & \vdots & & \\ 0 & 0 & 0 & \cdots & -1 & -(3\Delta x+1) \end{pmatrix}$$

and

$$B = \begin{bmatrix} 4\Delta x\\ 4\Delta x\\ \vdots\\ 4\Delta x \end{bmatrix}, \qquad X = \begin{bmatrix} \phi_0\\ \phi_1\\ \vdots\\ \phi_n \end{bmatrix}$$

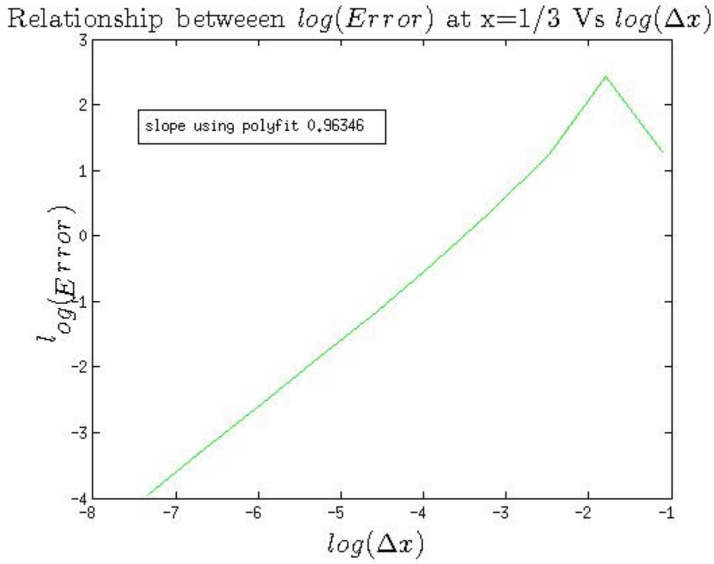
We solve the above equations using the code:
Explanation as same as before.

```
clear all
numGrids = 10;                  % number of different grids over which solution is
    sought
numFirstGridElems = 3; xVal = 1/3; % number of elements in the first (coarsest) grid
xArray = cell(numGrids,1);       % array storing the x values at grid locations
phiFD = cell(numGrids,1);        % solution vectors
midValsFD = zeros(numGrids,1);   % an array storing all phi values at x , FD
stepVals = zeros(numGrids,1);    % array storing delta x values for different grids
phi_0 = 0;                       % phi b.c. at x = 0

% Now we do the above solution using backward difference formula for first derivative
for i = 1:numGrids
    numElem = numFirstGridElems*2^(i-1);  % number of elements
    numPoints = numElem;                  % number of grid points, N in class
    xArray{i} = linspace(0,1,numPoints+1)'; % {} references Cell elements, () references
        array or matrix elements
    xArray{i} = xArray{i}(2:numPoints+1); % [0,1] divided into numPoints interior grid
        points
    deltaX = 1/numElem;                   % step size

    offdiagVec = -1*ones(numPoints-1,1);         % main diagonal of length n
    diagVec = (1 - 3*deltaX)*ones(numPoints,1); % off-diagonal vectors of length (n-1)
    myMat = gallery('tridiag',offdiagVec,diagVec,zeros(numPoints-1,1)); % matrix of
        coefficients
    myVec = 4*deltaX*ones(numPoints,1);   % rhs vector
    myVec(1) = myVec(1) + phi_0*(3*deltaX + 1);
    phiFD{i} = myMat\myVec;               % solution obtained by matrix inversion
    index = find(xArray{i}==xVal);        % index returns the location of Xval in the
        array xArray{i}
    midValsFD(i) = phiFD{i}(index);       % pick the computed value at x = xVal;
    stepVals(i) = deltaX;                 % store step size for plotting later

    errorAtX1(i) = midValsFD(i)-4/3*(exp(3*xVal)-1); % calculate the RMS error in every
        iteration of delta x=1/3

    xVal = 2/3;
    index = find(xArray{i}==xVal);        % index returns the location of Xval in the
        array xArray{i}
    midValsFD(i) = phiFD{i}(index);       % pick the computed value at x = xVal;
    errorAtX2(i) = midValsFD(i)-4/3*(exp(3*xVal)-1); % error in every iteration of delta
        x2/3
end

% Now we have error at x1 and x2 for different delta x. we plot log(error) Vs log(delta
    x) for checking the order of error
%% plot FD fit and true solution
power = polyfit(log(abs(errorAtX1')), log(stepVals), 1)
figure;
plot(log(stepVals), log(abs(errorAtX1')), 'g')
title('Relationship betweeen $$log(Error)$$ at x=$$1/3$$ Vs $$log(\Delta x)$$', ...
    'Interpreter','LaTeX','FontSize',18)   % plot title at the top
xlabel('$$log(\Delta x)$$',...              % Label for x-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
ylabel('$$log(Error)$$',...        % Label for y-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
annotation('textbox', [.2 .7 .1 .1], 'String', ['slope using polyfit
    ',num2str(power(1))]);
power = polyfit(log(abs(errorAtX2')), log(stepVals), 1);
figure;
plot(log(stepVals), log(abs(errorAtX2')), 'g')
```

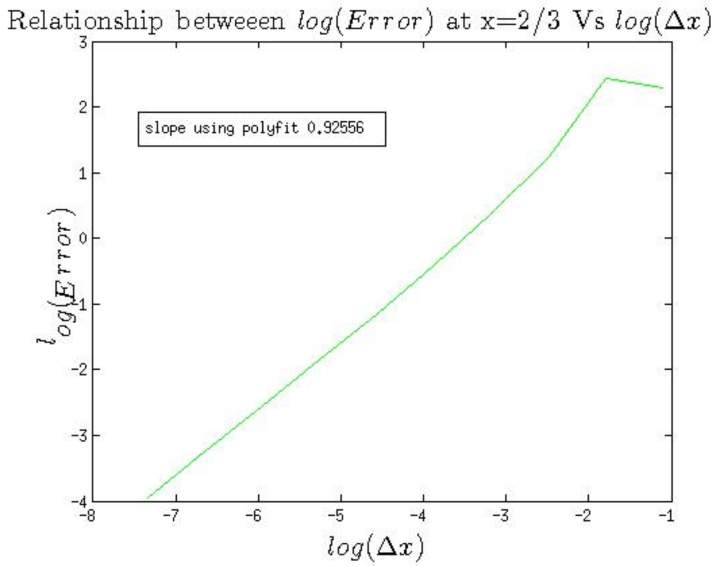
```
title('Relationship betweeen $$log(Error)$$ at x=$$2/3$$ Vs $$log(\Delta x)$$', ...
    'Interpreter','LaTeX','FontSize',18)   % plot title at the top
xlabel('$$log(\Delta x)$$',...              % Label for x-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
ylabel('$$log(Error)$$',...        % Label for y-axis
       'Interpreter','LaTeX', ...
       'FontSize',18)
annotation('textbox', [.2 .7 .1 .1], 'String', ['slope using polyfit
    ',num2str(power(1))]);
```

And the Plots using backward difference are:

x=$\frac{1}{3}$:



x=$\frac{2}{3}$:



2 Problem 6

2.1 Part A

Here we use aproximation for double derivative in terms of finite elements. And we can use that definition to create a system of equations.

$$(\frac{d^2\phi}{dx^2})_n \approx \frac{\phi_{n+1}-2\phi_n+\phi_{n-1}}{(\Delta x)^2}$$

We use forward difference for this part of question. From the definition of forward difference:

$$\begin{aligned}\frac{\phi_{n+1} - 2\phi_n + \phi_{n-1}}{\Delta x} - \phi_n = 0 \\ -\phi_{n+1} - (2 + \Delta x^2)\phi_n + \phi_{n-1} = 0\end{aligned} \tag{3}$$

When we write out all the equations in recurrence formula, we get a system of linear equations with n variables.

$$\text{AX} = \text{B}$$

Where matrices A , X and B are:

$$A = \begin{pmatrix} (2+\Delta x^2) & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & (2+\Delta x^2) & -1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots\vdots\vdots & & & \\ 0 & 0 & 0 & \cdots & -1 & (2+\Delta x^2) & \end{pmatrix}$$

and

$$B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \qquad X = \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}$$

We solve the above equations using the code:

```
%% Initialize arrays and allocate space
clear all
numGrids = 10;                     % number of different grids over which solution is
    sought
numFirstGridElems = 4; xVal = 1/2; % number of elements in the first (coarsest) grid
xArray = cell(numGrids,1);        % array storing the x values at grid locations
phiFD = cell(numGrids,1);         % Forward difference solution vectors
midValsFD = zeros(numGrids,1);    % an array storing all phi values at x = 0.5, FD
stepVals = zeros(numGrids,1);     % array storing delta x values for different grids
phi_0 = 0;                        % phi b.c. at x = 0
phi_1 = 1;
%
%% Solve using Forward Differences
for i = 1:numGrids
    numElem = numFirstGridElems*2^(i-1);  % number of elements
    numPoints = numElem-1;                 % number of grid points, N in class
    xArray{i} = linspace(0,1,numPoints); % {} references Cell elements, () references
        array or matrix elements
    % xArray{i} = xArray{i}(2:numPoints+1); % [0,1] divided into numPoints interior grid
        points
    deltaX = 1/numElem;                   % step size
    %
    diagVec = (2 + deltaX^2)*ones(numPoints,1);          % main diagonal of length n
    myMat = gallery('tridiag',-1*ones(numPoints-1,1) ,diagVec, -1*ones(numPoints-1,1));
        % matrix of coefficients
    myVec = [phi_0 zeros(1,numPoints-2) phi_1]';
    phiFD{i} = myMat\myVec;               % solution obtained by matrix inversion
    midValsFD(i) = phiFD{i}((numPoints+1)/2);       % pick the computed value at x = xVal;
    stepVals(i) = deltaX;                 % store step size for plotting later
```

```
    end
```

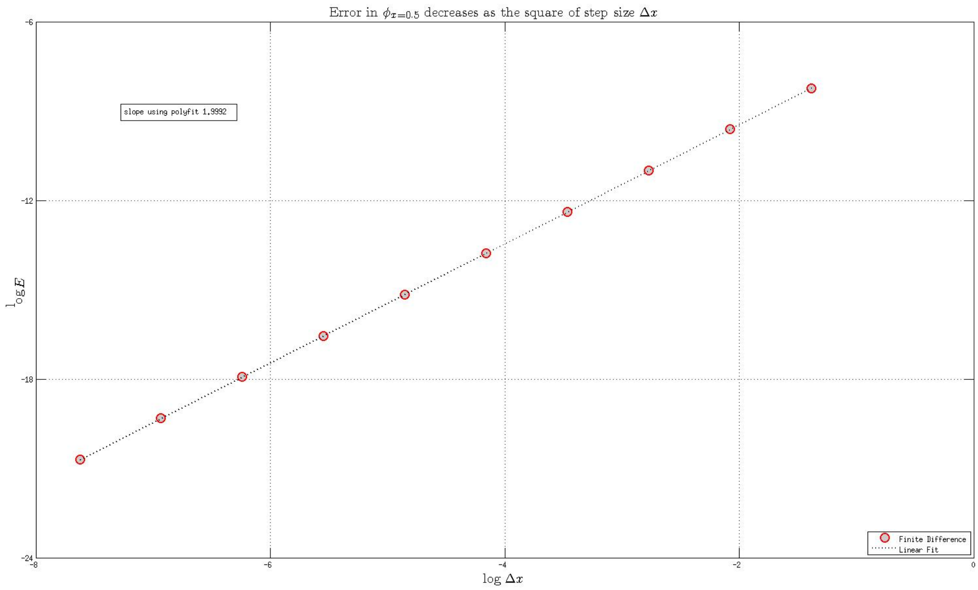
Now we do a polyfit between $log(\Delta x)$ and $log(Error)$ linerly:

```
x=0.5;
trueVal = (exp(1)/(exp(1)^2 -1 ))*(exp(x)-exp(-x));
errorAtX1 = midValsFD - trueVal;
%% plot FD fit and true solution

power = polyfit(log(stepVals), log(errorAtX1), 1)
```

The slope is found to be ≈ 1. Which means the order of error is quadratic.
Now we plot this using code:

```
    figure;
plot(log(stepVals), log(abs(errorAtX1')), 'bo','MarkerSize',12, ...
                                        'MarkerFaceColor',[0.8 0.8 0.8], ...
                                        'MarkerEdgeColor','r',...
                                        'LineWidth',2)
annotation('textbox', [.2 .7 .1 .1], 'String', ['slope using polyfit ',num2str(power(1))]);

fitVal = polyval(power,log(stepVals)); % obtain straight line fit to data
hold on % set hold = on to make next plot appear in the same figure
plot(log(stepVals),fitVal,'k:','LineWidth',2) % plot the straight line fit, 'k' for black,
    ':' for dotted line
%
% set various plot options
set(gca,'XTick',-8:2:0) % tick marks on x-axis
set(gca,'YTick',-24:6:-6) % tick marks on y-axis
set(gca,'FontSize',14) % font size for both axes
xlim([-8,0]); % x-range for plot
ylim([-24 -6]); % y-range for plot
xlabel('$$\log\Delta x$$',... % Label for x-axis
 'Interpreter','LaTeX', ...
 'FontSize',18)
ylabel('$$\log E$$',... % Label for y-axis
 'Interpreter','LaTeX', ...
 'FontSize',18)
grid('on') % grid lines to read off values easier
legend('Finite Difference','Linear Fit','Location','SouthEast') % plot legend
title('Error in $$\phi_{x=0.5}$$ decreases as the square of step size $$\Delta x$$', ...
 'Interpreter','LaTeX','FontSize',18)
```

And the plot is:



3 Problem 7

We use forward difference to aproximate for double derivative.

$$\frac{\phi_{n+1} - 2\phi_n + \phi n - 1}{\Delta x^2} = k(H - n\Delta x)^2 \tag{4}$$

$$\phi_{n+1} - 2\phi_n + \phi n - 1 = k(H - n\Delta x)^2 \Delta x^2 \tag{5}$$

where k is the constant in the equation. This is solved using the code:

```
clear all
numGrids = 10;                    % number of different grids over which solution is
    sought
numFirstGridElems = 4; xVal = 1/2; % number of elements in the first (coarsest) grid
xArray = cell(numGrids,1);        % array storing the x values at grid locations
phiFD = cell(numGrids,1);         % Forward difference solution vectors
midValsFD = zeros(numGrids,1);    % an array storing all phi values at x = 0.5, FD
stepVals = zeros(numGrids,1);     % array storing delta x values for different grids
phi_0 = 0;                        % phi b.c. at x = 0
phi_1 = 1;
k = -4*9*0.1/3^4*0.08;
%
%% Solve using Forward Differences
for i = 1:numGrids
    numElem = numFirstGridElems*2^(i-1);  % number of elements
    numPoints = numElem-1;                  % number of grid points, N in class
    xArray{i} = linspace(0,1,numPoints); % {} references Cell elements, () references
        array or matrix elements
    % xArray{i} = xArray{i}(2:numPoints+1); % [0,1] divided into numPoints interior grid
        points
    deltaX = 1/numElem;                   % step size
    %
    diagVec = -2*ones(numPoints,1);          % main diagonal of length n
```

```
    myMat = gallery('tridiag',ones(numPoints-1,1) ,diagVec, ones(numPoints-1,1)); %
        matrix of coefficients
    for j=1:numPoints
        myVec(j) = k*deltaX^2*(3-j*deltaX)^2*ones(1,numPoints);
    end
    myVec(1) = myVec(1) - phi_0;
    myVec(numPoints) = myVec(numPoints) - phi_1;
    myVec = [phi_0 zeros(1,numPoints-2) phi_1]';
    phiFD{i} = myMat\myVec;               % solution obtained by matrix inversion
    midValsFD(i) = phiFD{i}((numPoints+1)/2);      % pick the computed value at x = xVal;
    stepVals(i) = deltaX;                 % store step size for plotting later

    trueVal = (-5/108)*(0.25^4-12*0.25^3 + 54*0.25^2 -126*0.25);
    errorAtX1(i) = 4/3*(trueVal - midValsFD(i)); % calculate the error in every
        iteration of delta x=1/3

    xVal = 1/2;
    trueVal = (-5/108)*(0.5^4-12*0.5^3 + 54*0.5^2 -126*0.5);
    index = find(xArray{i}==xVal);        % index returns the location of Xval in the
        array xArray{i}
    midValsFD(i) = phiFD{i}(index);       % pick the computed value at x = xVal;
    errorAtX2(i) = 4/3*(trueVal - midValsFD(i)); % error in every iteration of delta x2/3

    xVal = 3/4;
    trueVal = (-5/108)*(0.75^4-12*0.75^3 + 54*0.75^2 -126*0.75);
    index = find(xArray{i}==xVal);        % index returns the location of Xval in the
        array xArray{i}
    midValsFD(i) = phiFD{i}(index);       % pick the computed value at x = xVal;
    errorAtX3(i) = 4/3*(trueVal - midValsFD(i)); % error in every iteration of delta x2/3
end
 % To get error as a function of $$\Delta x$$ x we use polyfit.
plot(log(stepVals), log(errorAtX1))
p1 = polyfit(log(stepVals'), log(errorAtX1), 1)
p2 = polyfit(log(stepVals'), log(errorAtX2), 1)
p3 = polyfit(log(stepVals'), log(errorAtX3), 1)
```