Math 226B Project 1

Theodore Kwan
tmkwan@uci.edu

January 25, 2016

Contents

1 Project Description

This project analyzes the finite element method on the Poisson equation with Dirichlet and Neumann Boundary Conditions.

2 Mesh Generation

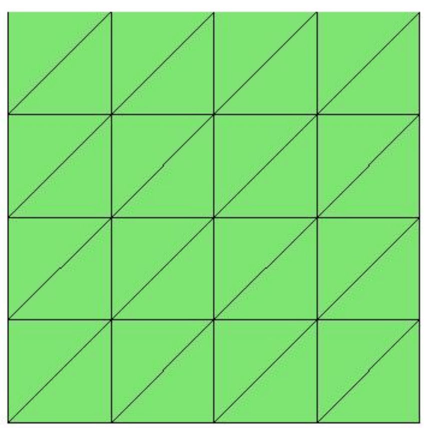
The first step is to use the mesh generation functions provided in the Ifem file. There are two types of meshes, a square and a circular mesh.

2.1 Square Mesh

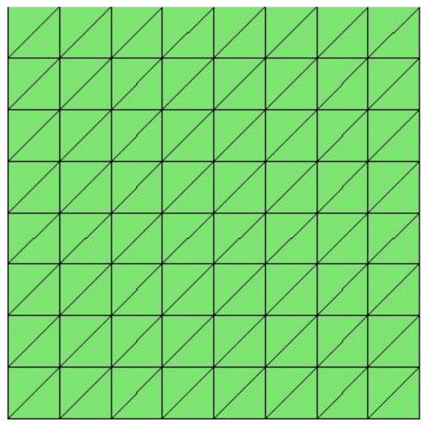
The square mesh is generated with the vector containing the corners:

$$\vec{s} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{\min} \\ x_{\max} \\ y_{\min} \\ y_{\max} \end{pmatrix}$$

And the initial element edge size of $h = 0.25$ The mesh generated was:



After refinement, the mesh becomes:

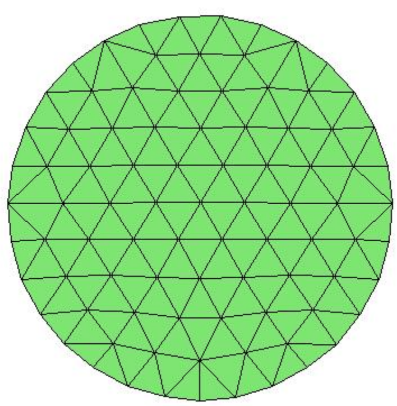


2.2 Circular Mesh

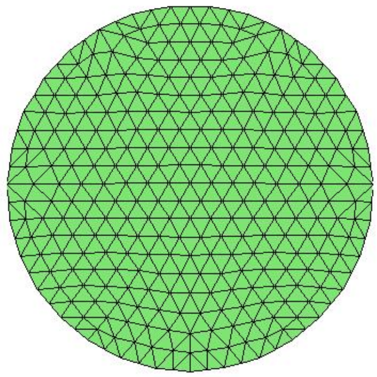
The circular mesh is generated by providing 4 values:

$$
\begin{aligned}
x &\equiv \text{Horizontal Center of Circle} \\
y &\equiv \text{Vertical Center of Circle} \\
r &\equiv \text{Radius of Circle} \\
h &\equiv \text{Element Edge size}
\end{aligned}
$$

Using an initial element edge size of $h = 0.2$ The mesh generated was:



After refinement, the mesh becomes:



3 Assembling Stiffness Matrix Comparison

There are three different versions used to assemble the stiffness matrix A.

3.1 Full Matrix

Listing 1: Assemble Full Stiffness Matrix

```
%% Assembling Standard
%   Copied by Ted Kwan from notes
%   written by Chen Long
%
%   Assembles full stiffness matrix
%   using the localstiffness method at
%   each element.
function [A]= assemblingstandard(node,elem)
    N=size(node,1); NT=size(elem,1);
    A=zeros(N,N); %A=sparse(N,N);
    for t=1:NT
        At=localstiffness(node(elem(t,:),:));
        for i=1:3
            for j=1:3
                A(elem(t,i),elem(t,j))= ...
                A(elem(t,i),elem(t,j))+At(i,j);
            end
        end
    end
end
```

3.2 Sparse Matrix

Listing 2: Assemble Sparse Stiffness Matrix

```
%% Assembling Sparse
%
%   Copied by Ted Kwan from notes
%   written by Chen Long.
%
%   Assembles sparse stiffness matrix
%   using the localstiffness method at
%   each element.
function [A] = assemblingsparse(node,elem)
    N=size(node,1); NT=size(elem,1);
    i=zeros(9*NT,1); j=zeros(9*NT,1); s=zeros(9*NT,1);
    index=0;
    for t=1:NT
        At=localstiffness(node(elem(t,:),:));
        for ti=1:3
            for tj=1:3
                index=index+1; i(index)=elem(t,ti);
                j(index)=elem(t,tj); s(index)=At(ti,tj);
            end
        end
    end
    A=sparse(i,j,s,N,N);
end
```

3.3 Vectorized Sparse Matrix

Listing 3: Quickly Assembling Sparse Stiffness Matrix

```
%% Assembling Quick
%
%   Copied by Ted Kwan from notes
%   written by Chen Long.
%
%   Assembles sparse stiffness matrix
%   using vector of areas quickly.
%
function [A] = assembling(node,elem)
   N=size(node,1); NT=size(elem,1);
   ii=zeros(9*NT,1); jj=zeros(9*NT,1); sA=zeros(9*NT,1);
   ve(:,:,3)=node(elem(:,2),:)-node(elem(:,1),:);
   ve(:,:,1)=node(elem(:,3),:)-node(elem(:,2),:);
   ve(:,:,2)=node(elem(:,1),:)-node(elem(:,3),:);
   area=0.5*abs(-ve(:,1,3).*ve(:,2,2)+ve(:,2,3).*ve(:,1,2));
   index=0;
   for i=1:3
       for j=1:3
           ii(index+1:index+NT)=elem(:,i);
           jj(index+1:index+NT)=elem(:,j);
           sA(index+1:index+NT)=dot(ve(:,:,i),ve(:,:,j),2)./(4*area);
           index=index+NT;
       end
   end
   A=sparse(ii,jj,sA,N,N);
end
```

3.4 Comparisons

The three different methods are compared for their time usage on both the square and the circle meshes. The sparse matrix assembly is slightly faster than the full matrix assembly, but the vectorized method is far quicker than either. The results are given below.

3.4.1 Square Mesh

Using 4 iterations of uniformrefine on a square mesh, the results are:

Profile Summary
Generated 23-Jan-2016 00:36:03 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
localstiffness	21760	2.952 s	2.952 s	
assemblingsparse	4	1.865 s	0.304 s	
assemblingstandard	4	1.640 s	0.249 s	
uniformrefine	4	0.088 s	0.013 s	
myunique	4	0.075 s	0.017 s	
unique	4	0.053 s	0.015 s	
assembling	4	0.048 s	0.046 s	

3.4.2 Circular Mesh

Using 3 iterations of uniformrefine on a circular mesh, the results are:

Profile Summary
Generated 24-Jan-2016 02:46:15 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
localstiffness	24024	3.993 s	3.993 s	
assemblingstandard	3	2.363 s	0.308 s	
assemblingsparse	3	2.274 s	0.336 s	
assembling	3	0.052 s	0.050 s	

4 Finite Element Method

The Matlab Script which does the finite element method on the PDE given, is listed in this section. The script has a fixup for the Neumann boundary conditions, where it removes the additive constant which must show up since the PDE solution is only unique up to an additive constant.

Listing 4: Finite Element Method Poisson Equation

```
%% Finite Element Method
%
%   Written by Ted Kwan for Math 226B using code provided
%   within the lectures by Professor Chen at UCI.
%
%   Returns u(x,t) to approximate the solution
%   to the Poisson equation -\Delta u=f
%
%%% Inputs
% * mesht - String for mesh type (square or circle).
% * bdryt - String for boundary type (Dirichlet, Neumann or Mixed).
% * f - function handle for the right hand side.
% * ref - number of refinements to be used.
% * varargin options:
% 1 function handle for g(x).
% 2 function handle for g_n (normal derivative).
% * For mesht='circle'
% 3 x coordinate.
% 4 y coordinate.
% 5 radius r.
% 6 Mesh size h.
% * For mesht='square'
% 3 vector with the four boundaries of the rectangle.
% 4 Mesh size h.
%
function [u,node,elem,A]= FiniteElem(mesht,bdryt,f,ref,varargin)
    d=2; % Sets Dimension to two.
    %% Mesh Generation
    %   Chooses a type of mesh generation based on
    %   the input parameter mesht.
    %
    if(strcmpi(mesht,'circle'))
        if(nargin<6) % Safety for access of elements.
            error('Not enough arguments');
        else
            %%% Create circle mesh
            x=varargin{3};y=varargin{4}; r=varargin{5};h=varargin{6};
            [node,elem] = circlemesh(x,y,r,h);
        end
    else
        if(nargin<4) % Safety for access of elements.
            error('Not enough input arguments');
        else
            %%% Create square mesh.
            squ=varargin{3}; h=varargin{4};
            [node,elem] = squaremesh(squ,h);
        end
    end
```

```
%% Mesh Refinement.
%
while(ref>0)
    [node,elem] = uniformrefine(node,elem);
    ref=ref-1;
end
%% Assemble Stiffness Matrix.
%
%   Quick method to generate sparse matrix
%   A, the stiffness matrix.
%
N=size(node,1); NT=size(elem,1);
ii=zeros(9*NT,1); jj=zeros(9*NT,1); sA=zeros(9*NT,1);
ve(:,:,3)=node(elem(:,2),:)-node(elem(:,1),:);
ve(:,:,1)=node(elem(:,3),:)-node(elem(:,2),:);
ve(:,:,2)=node(elem(:,1),:)-node(elem(:,3),:);
area=0.5*abs(-ve(:,1,3).*ve(:,2,2)+ve(:,2,3).*ve(:,1,2));
index=0;
for i=1:3
   for j=1:3
       ii(index+1:index+NT)=elem(:,i);
       jj(index+1:index+NT)=elem(:,j);
       sA(index+1:index+NT)=dot(ve(:,:,i),ve(:,:,j),2)./(4*area);
       index=index+NT;
   end
end
A=sparse(ii,jj,sA,N,N);
%% Calculate RHS
%
mid1 = (node(elem(:,2),:)+node(elem(:,3),:))/2;
mid2 = (node(elem(:,3),:)+node(elem(:,1),:))/2;
mid3 = (node(elem(:,1),:)+node(elem(:,2),:))/2;
bt1 = area.*(f(mid2(:,1),mid2(:,2))+f(mid3(:,1),mid3(:,2)))/6;
bt2 = area.*(f(mid3(:,1),mid3(:,2))+f(mid1(:,1),mid1(:,2)))/6;
bt3 = area.*(f(mid1(:,1),mid1(:,2))+f(mid2(:,1),mid2(:,2)))/6;
b = accumarray(elem(:),[bt1;bt2;bt3],[N,1]);
%%% End RHS calculation.

%% Boundary Condition.
%
if(strcmpi(bdryt,'Dirichlet'))
    %%% Dirichlet Boundary Conditions.
    %
    g_D=varargin{1};
    [bdNode,bdEdge,isBdNode]=findboundary(elem);
    freeNode = find(~isBdNode);
    u = zeros(N,1);
    u(bdNode) = g_D(node(bdNode,1),node(bdNode,2));
    b = b - A*u;
    u(freeNode)=A(freeNode,freeNode)\b(freeNode);
```

```
    elseif(strcmpi(bdryt,'Neumann'))
        %%% Neumann Boundary Conditions
        %
        g_N=varargin{2};
        [bdNode,bdEdge,isBdNode,isBdElem]=findboundary(elem);
        if(strcmpi(mesht,'circle'))
            bdFlag=SetBdFlagCir(isBdElem,elem,isBdNode,NT,d);
        else
            bdFlag=SetBdFlagSq(isBdElem,elem,isBdNode,bdEdge,NT,d);
        end
        u = zeros(N,1);
        totalEdge = [elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])];
        Neumann = totalEdge(bdFlag(:) == 2,:);
        Nve = node(Neumann(:,1),:) - node(Neumann(:,2),:);
        edgeLength = sqrt(sum(Nve.^2,2));
        mid = (node(Neumann(:,1),:) + node(Neumann(:,2),:))/2;
        b = b + accumarray([Neumann(:),ones(2*size(Neumann,1),1)], ...
          repmat(edgeLength.*g_N(mid(:,1),mid(:,2))/2,2,1),[N,1]);
        b=b-mean(b)*ones(N,1); % Compatibility Condition.
        %u=bicgstabl(A,b,1e-8,500); %Uncomment to use
        %u=gmres(A,b,3,1e-6,300);
        u=A\b;
        %%% Fixup for the additive constant.
        %   This ensures that the constant is 0 as it
        %   is supposed to be, to compare it to the chosen
        %   real solution.
        ma=max(u);mi=min(u);
        if(ma>0)
            u=u-((ma)*ones(N,1));
        else
            u=u+(abs(ma)*ones(N,1));
        end
    else
        %%% Mixed boundary conditions - Not implemented
        %
        %   This method is not implemented, because the variable
        %   bdFlag is not passed to the function.
        %
        error('Mixed Boundary Conditions Are not yet implemented.');
        totalEdge = [elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])];
        Dirichlet = totalEdge(bdFlag(:) == 1,:);
        Neumann = totalEdge(bdFlag(:) == 2,:);
        %---------- Dirichlet boundary conditions ----------
        isBdNode = false(N,1);
        isBdNode(Dirichlet) = true;
        bdNode = find(isBdNode);
        freeNode = find(~isBdNode);
        u = zeros(N,1);
        u(bdNode) = g_D(node(bdNode,1),node(bdNode,2));
        b = b - A*u;
```

```
        %----------- Neumann boundary conditions -----------
        if (~isempty(Neumann))
        Nve = node(Neumann(:,1),:) - node(Neumann(:,2),:);
        edgeLength = sqrt(sum(Nve.^2,2));
        mid = (node(Neumann(:,1),:) + node(Neumann(:,2),:))/2;
        b = b + accumarray([Neumann(:),ones(2*size(Neumann,1),1)], ...
        repmat(edgeLength.*g_N(mid(:,1),mid(:,2))/2,2,1),[N,1]);
        end
    end
end
```

5 Dirichlet Boundary Conditions

Consider the following Poisson Equation on $\Omega = [0,1] \times [0,1]$ with Dirichlet boundary conditions:

$$\begin{cases} \Delta u = 8\pi^2 \sin(2\pi x)\cos(2\pi y), & (x,y) \in \Omega \\ u = \sin(2\pi x)\cos(2\pi y), & (x,y) \in \partial\Omega \end{cases} \tag{1}$$

The PDE (1) has solution:

$$u(x,y) = \sin(2\pi x)\cos(2\pi y) \tag{2}$$

To apply the Dirichlet boundary conditions to the finite element method, first set:

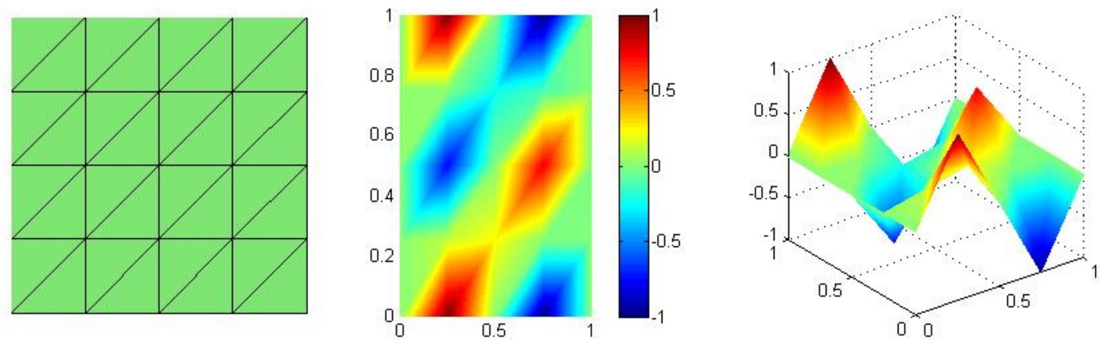
$$u(x_i,y_j) = \sin(2\pi x_i)\cos(2\pi y_j), \quad (x_i,y_j) \in \partial\Omega \tag{3}$$

And change the right hand side to be:

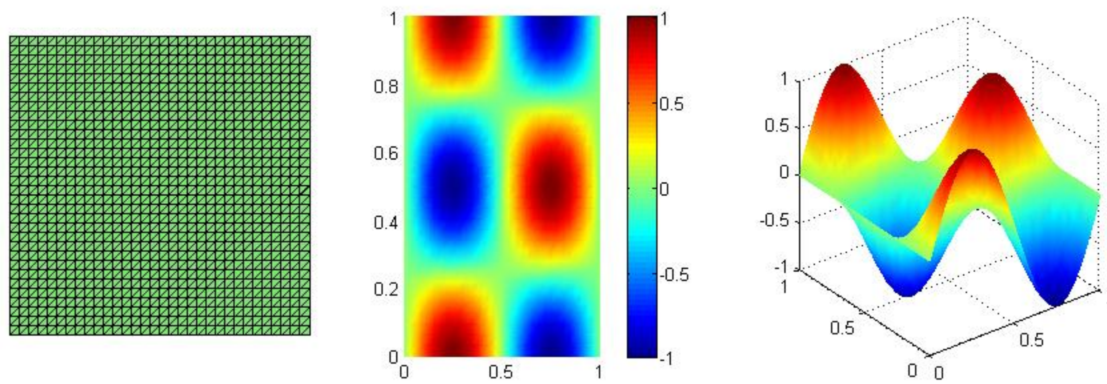
$$\vec{b} = \vec{b} - A\vec{u} \tag{4}$$

5.1 Solution Plots

First, we can plot the solution without any refinement:

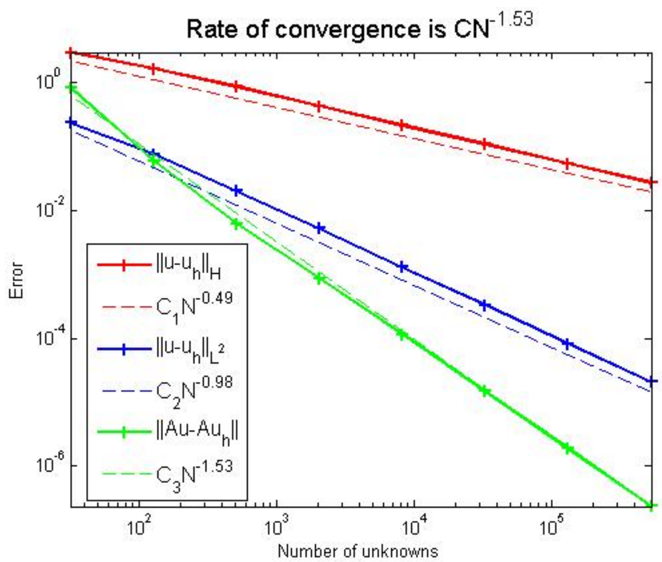


After refining the grid a couple of times, the picture looks more like the actual solution:



5.2 Convergence Rate

The convergence rate is high, since the dirichlet boundary conditions require no fixup like the Neumann boundary conditions. The convergence rate plot is:



6 Neumann Boundary Conditions

Consider the following PDE for the unit circle $\Omega = \{(x,y) \,:\, x^2+y^2<1\}$:

$$\begin{cases} \Delta u = -1, & (x,y) \in \Omega \\ \frac{\partial u}{\partial \vec{n}} = 0, & (x,y) \in \partial\Omega \end{cases} \tag{5}$$

Problem (5) is not well posed, since the compatibility condition is not satisfied:

$$\int_\Omega -1 dA = -\pi 1^2 = -\pi \neq \int_{\partial\Omega} 0 ds = 0 \tag{6}$$

But, the Poisson problem given by:

$$\begin{cases} -\Delta u = 1, & (x,y) \in \Omega \\ \nabla u \cdot \vec{n} = -\frac{1}{2}, & (x,y) \in \partial\Omega \end{cases} \tag{7}$$

Does satisfy the compatibility condition since:

$$\int_\Omega -f dA = \int_\Omega -1 dA = -\pi = \int_{\partial\Omega} \frac{ds}{2} = -\frac{2\pi}{2} = -\pi \tag{8}$$

Converting to polar coordinates, the laplacian becomes:

$$\Delta u = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}$$

Since the solution to (7) has radial symmetry,

$$u = u(r) \Rightarrow \frac{\partial^2 u}{\partial \theta^2} = 0$$

So the partial differential equation in (7) becomes:

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} = \frac{1}{r}\left(r\frac{\partial u}{\partial r}\right)_r = -1$$

Therefore:

$$\left(r\frac{\partial u}{\partial r}\right)_r = -r \Rightarrow r\frac{\partial u}{\partial r} = -\frac{r^2}{2} + c_1$$
$$\Rightarrow \frac{\partial u}{\partial r} = -\frac{r}{2} + \frac{c_1}{r} \Rightarrow u(r) = -\frac{r^2}{4} + c_1 \log|r| + c_2$$

Since the function is defined at 0, it must be the case that $c_1 = 0$. The only other boundary condition is the Neumann condition, so the solution is unique only up to an additive constant.

We can choose the constant to be 0 which gives:

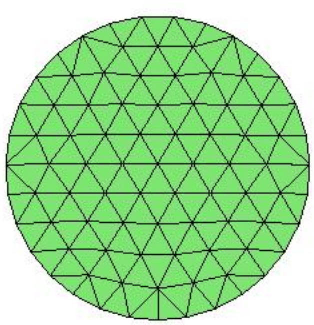
$$u(r,\theta) = \frac{-r^2}{4} \quad u(x,y) = \frac{-x^2 - y^2}{4} \tag{9}$$

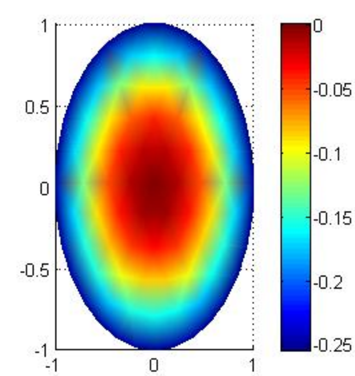
The gradient of (9) is:

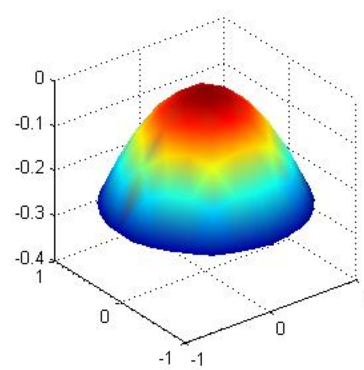
$$\nabla u = -\frac{1}{2}\begin{pmatrix} x \\ y \end{pmatrix}$$

6.1 Solution Plots

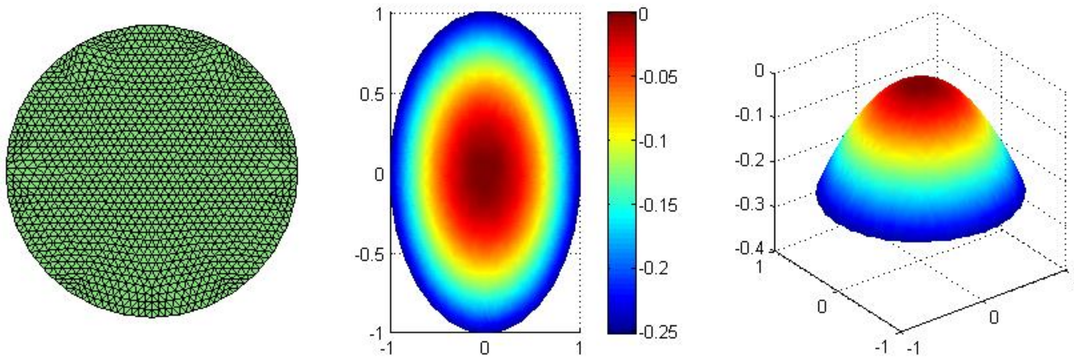
First, we can plot the solution without any refinement:





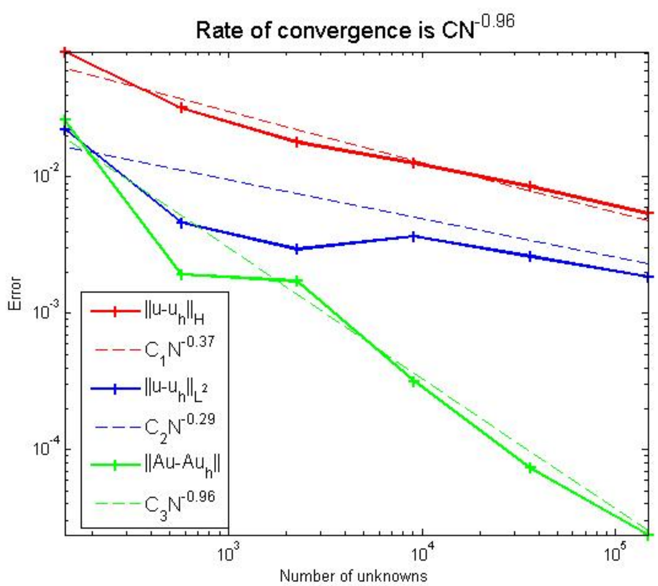


After refining the grid a couple of times, the picture looks more like the actual solution:

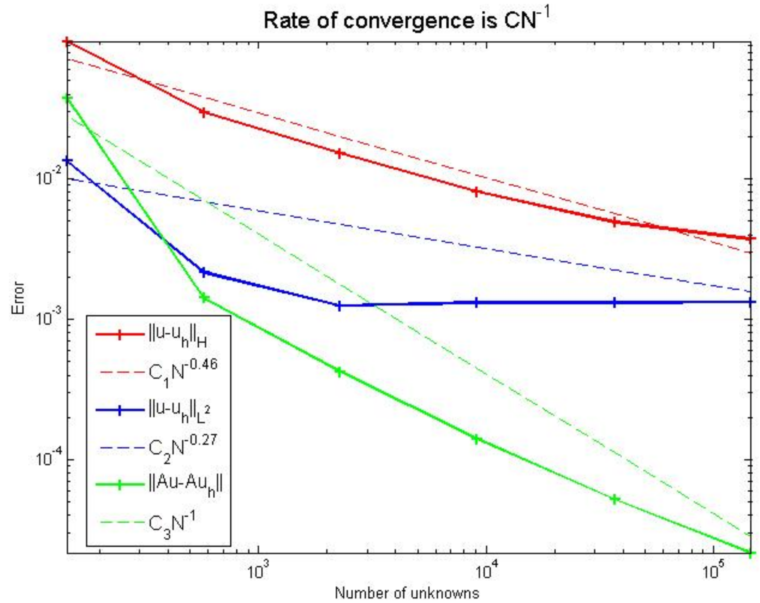


6.2 Convergence Rate

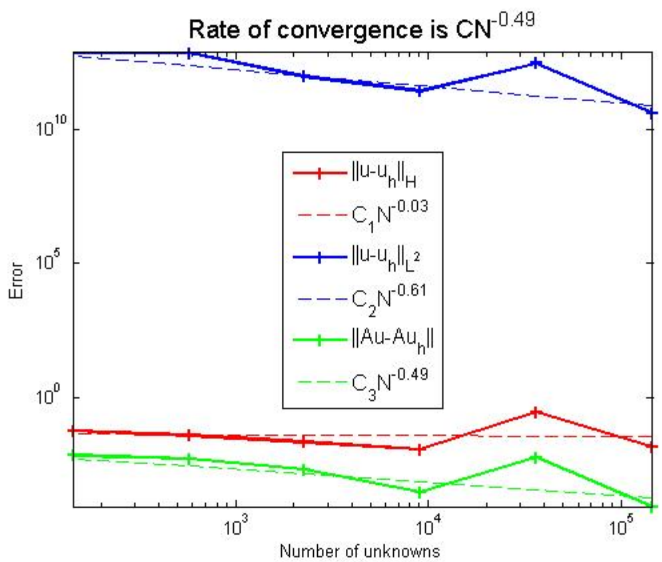
There were two methods of solving the linear system attempted. First, the mldivide algorithm was applied to give:



Secondly, the biconjugate gradient method was applied to give:



So the convergence is slightly worse when the mldivide algorithm is used, but both work as long as the fixup for the constant is ensured. Without the fixup for the arbitrary constant, the results can be off by an extremely large amount:



7 Supporting Code

This section contains a list of all of the supporting code used for both the tests, and within the FiniteElem matlab function.

7.1 Mesh Generation

This method performs the mesh generation, and profile viewing for the different ways to assemble the stiffness matrix.

Listing 5: Generate and Compare Different Meshes

```
%% Generate Mesh
%
%   Written by Ted Kwan for Math 226B Project 1
%
%   This method generates the meshes, plots them,
%   then compares them in the profile viewer.
%
clear all;
[node,elem] = squaremesh([0,1,0,1],0.25);
% [node,elem] = circlemesh(0,0,1,0.2);
% showmesh(node,elem);
profile on
for i=1:3
    [node,elem] = uniformrefine(node,elem);
    tic; assemblingstandard(node,elem); toc;
    tic; assemblingsparse(node,elem); toc;
    tic; assembling(node,elem); toc;
end
profile viewer
```

7.2 Finite Element Method Tests

Listing 6: Test Finite Element Method

```
%% Test Finite Element Method
%
%   Written by Ted Kwan for Math 226B Project 1.
%
%   This checks the error and plots the convergence rate for both
%   the PDE with either Dirichlet or Neumann boundary conditions
%   given in the project.
%

clear all;

%% Dirichlet Boundary Setup
%
% mesht='square';
% bdryt='dirichlet';
% f=@(x,y) (8*(pi^2)*sin(2*pi*x).*cos(2*pi*y));
% uHi=inline('sin(2*pi*pxy(:,1)).*cos(2*pi*pxy(:,2))','pxy');
% Du = inline('[2*pi*cos(2*pi*pxy(:,1)).*cos(2*pi*pxy(:,2)) ...
% ,-2*pi*sin(2*pi*pxy(:,1)).*sin(2*pi*pxy(:,2))]','pxy'); %Split into
% two lines to fit in window. Needs to be one line to run.
% squ=[0,1,0,1]; h=0.25;
% g=inline('sin(2*pi*x).*cos(2*pi*y)');
% gn=@(x) NeumBdry(x);
%% Neumann Problem Setup
%
mesht='circle';
bdryt='Neumann';
f=@(x,y) (1); g=@(x,y) (-0.5);
DuN=inline('[-pxy(:,1)/2,-pxy(:,2)/2]','pxy');
uHi=inline('(-(pxy(:,1).^2)-(pxy(:,2).^2))/4','pxy');
x=0;y=0;r=1;h=0.2;
%% Refine and Plot
%
%   This section refines the grid and plots the different
%   errors after recording the amount of elements used
%   in the particular refinement.
%
N=5; err=zeros(N,3); ns=zeros(N,1);
for ref=0:N
%     [us,node,elem,A]=FiniteElem(mesht,bdryt,f,ref,g,0,squ,h);
    [us,node,elem,A]=FiniteElem(mesht,bdryt,f,ref,g,g,x,y,r,h);
    ns(ref+1)=length(elem(:,1)); uh1 = uHi(node);
    err(ref+1,3) = max(abs((A*uh1)-(A*us)));  clear A; %Retrieve memory.
    err(ref+1,1) = getH1error(node,elem,DuN,us);
    err(ref+1,2) = getL2error(node,elem,uHi,us);
end
```

```matlab
%% Plot Convergence Rate
%
%   This plots the convergence rate with the showrate function
%   given in the ifem folder.
%
sty1='||u-u_h||_H'; sty2='||u-u_h||_{L^{2}}'; sty3='||Au-Au_h||';
r1=showrate(ns,err(:,1),[],'r-+'); r2=showrate(ns,err(:,2),[],'b-+');
r3=showrate(ns,err(:,3),[],'g-+');
%%% Fixup for older versions on Matlab.
%
%   The round command does not work with a second argument
%   in older versions on Matlab, but it can be re—implemented
%   by just rounding the numbers manually.
%
r1nd=round((100*r1))/100; r2nd=round((100*r2))/100;
r3nd=round((100*r3))/100;
h_legend = legend(sty1,['C_1N^{' num2str(r1nd) '}'],...
                  sty2,['C_2N^{' num2str(r2nd) '}'],...
                  sty3,['C_3N^{' num2str(r3nd) '}'],'LOCATION','Best');
set(h_legend,'FontSize',12);
```

7.3 Set Boundary Flag

These methods set the boundary flag for the Neumann boundary conditions, depending on whether or not there is a square mesh, or a circular mesh:

7.3.1 Circular Boundary

Listing 7: Set Circular Boundary Flag

```
%% Set the boundary flag
%
%   Written by Ted Kwan for Math 226B  Project 1.
%
%   This method sets the boundary flag for Neumann
%   Boundary conditions. It checks each boundary element
%   then sets the non-boundary node to 2 so that the
%   opposite edge will be counted as a boundary edge.
%
function [bdFlag] = SetBdFlagCir(isBdElem,elem,isBdNode,NT,d)
    bdFlag=int8(zeros(NT,d+1));
    for i=1:NT
       if(isBdElem(i))
        %%% Find Boundary node
        %
        %   Once the node is found, then it is checked to see if there
        %   are two boundary nodes. If there are, the flag is set.
        if(((isBdNode(elem(i,1))==1) && (isBdNode(elem(i,2))==1) ...
           || ((isBdNode(elem(i,1))==1) && (isBdNode(elem(i,3))==1))...
           || ((isBdNode(elem(i,2))==1) && (isBdNode(elem(i,3))==1)))
           %%% Change bdFlag for a boundary node.
           for j=1:(d+1)
              if(isBdNode(elem(i,j))==0)
                  bdFlag(i,j)=2;
              end
           end
        end
       end
    end
end
```

7.3.2 Square Boundary

Listing 8: Set Square Boundary Flag

```
%% Set the boundary flag
%
%   Written by Ted Kwan for Math 226B  Project 1.
%
%   This method sets the boundary flag for Neumann
%   Boundary conditions. It checks each boundary element
%   then sets the node opposite each edge to 2.
%
function [bdFlag] = SetBdFlagSq(isBdElem,elem,isBdNode,bdEdge,NT,d)
    bdFlag=int8(zeros(NT,d+1));
    Nbde=length(bdEdge(:,1));
    for i=1:NT
       if(isBdElem(i))
            %%% Boundary Element has been found.
            %
            %   checks to ensure that the edges are
            %   actually on the boundary, then sets node.
            el=elem(i,:);
            if(ismember([el(1),el(2)],bdEdge,'rows') ||...
               ismember([el(2),el(1)],bdEdge,'rows'))
               bdFlag(i,3)=2;
            end
            if(ismember([el(1),el(3)],bdEdge,'rows') ||...
                   ismember([el(3),el(1)],bdEdge,'rows'))
                bdFlag(i,2)=2;
            end
            if(ismember([el(2),el(3)],bdEdge,'rows') ||...
                   ismember([el(3),el(2)],bdEdge,'rows'))
                bdFlag(i,1)=2;
            end
       end
    end
end
```

7.4 Plot Different Solutions

This method plots the different approximations made using the finite element method.

Listing 9: Plot Approximated Solutions

```
%% Graph Solutions
%
%   Written by Ted Kwan for Math 226B Project 1.
%
%   This plots the solution to either of the PDEs
%   given in the project description.

%% Setup initial properties
%


%%% Dirichlet Problem on the unit square
%
% mesht='square';
% bdryt='dirichlet';
% f=@(x,y) (8*(pi^2)*sin(2*pi*x).*cos(2*pi*y));
% squ=[0,1,0,1]; h=0.25;
% g=inline('sin(2*pi*x).*cos(2*pi*y)');

%%% Neumann Problem on the unit disk
%
mesht='circle';
bdryt='Neumann';
g=@(x,y) (-0.5);
f=@(x,y) (1);
x=0;y=0;r=1;h=0.2;

%% Find Approximation
%
%[us,node,elem,A]=FiniteElem(mesht,bdryt,f,0,g,gn,squ,h);
[us,node,elem,A]=FiniteElem(mesht,bdryt,f,0,g,g,x,y,r,h);

%% Plot
%
showresult(node,elem,u);
```