Math/CS 467/667 Programming Assignment 2

1. The two-dimensional Poisson equation is the elliptic partial differential equation $\Delta u = f$ for $x \in \Omega$ with boundary conditions $u(x) = \psi(x)$ for $x \in \partial\Omega$. To verify the performance of the finite-difference method start with a differentiable function that will serve as the exact solution. For this assignment let

$$u_{\text{ex}}(x) = 2x_1^4 \cos 3x_2 + e^{2x_2} \sin 2x_1.$$

Now, define $\psi(x) = u_{\text{ex}}(x)$ on $\partial\Omega$ and find f by computing Δu_{ex}. What is the value of $f(x)$?

Differentiating as

$$\frac{\partial u_{\text{ex}}}{\partial x_1} = 8x_1^3 \cos 3x_2 + 2e^{2x_2} \cos 2x_1, \qquad \frac{\partial^2 u_{\text{ex}}}{\partial x_1^2} = 24x_1^2 \cos 3x_2 - 4e^{2x_2} \sin 2x_1,$$

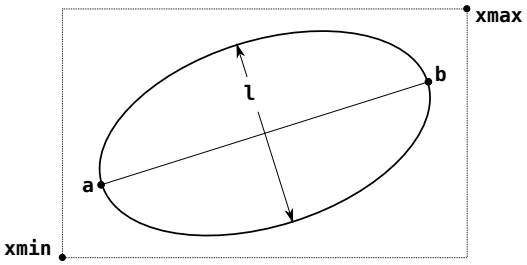
$$\frac{\partial u_{\text{ex}}}{\partial x_2} = -6x_1^4 \sin 3x_2 + 2e^{2x_2} \sin 2x_1 \qquad \text{and} \qquad \frac{\partial^2 u_{\text{ex}}}{\partial x_2^2} = -18x_1^4 \cos 3x_2 + 4e^{2x_2} \sin 2x_1$$

yields that

$$\begin{aligned}\Delta u_{\text{ex}} &= 24x_1^2 \cos 3x_2 - 4e^{2x_2} \sin 2x_1 - 18x_1^4 \cos 3x_2 + 4e^{2x_2} \sin 2x_1 \\ &= 24x_1^2 \cos 3x_2 - 18x_1^4 \cos 3x_2 = 6x_1^2(4 - 3x_1^2) \cos 3x_2.\end{aligned}$$

Therefore, $f(x) = 6x_1^2(4 - 3x_1^2) \cos 3x_2$.

2. Let Ω be an ellipse contained in a bounding box between $x_{\min}$ and $x_{\max}$ of the form



where a and b are the endpoints of the major axis and ℓ is the length of the minor axis. For

$$a = (1.5, 2), \qquad b = (6.5, 3), \qquad \ell = 2.6, \qquad x_{\min} = (1, 1) \qquad \text{and} \qquad x_{\max} = (7, 4)$$

find a function $\texttt{domain}(x)$ such that $\Omega = \{\, x \in \mathbf{R}^2 : \texttt{domain}(x) > 1 \,\}$. Note there are many choices that will work equally well. One idea is to use

$$\texttt{domain}(x) = 2 - (x - c)^T R^T S^2 R(x - c)$$

where c is the center of the ellipse, S is a diagonal matrix with the values $2/|a - b|$ and $2/\ell$ on the diagonal and R is a suitable rotation.

First compute c as

$$c = \frac{a + b}{2} = \frac{(1.5, 2) + (6.5, 3)}{2} = \frac{(8, 5)}{2} = (4, 2.5).$$

It follows that $b - c = (2.5, 0.5)$. In polar coordinates this point is given as $r(\cos\theta, \sin\theta) = (2.5, 0.5)$. Thus

$$(\cos\theta, \sin\theta) = \frac{(2.5, 0.5)}{r} = \begin{bmatrix} 5/\sqrt{26} \\ 1/\sqrt{26} \end{bmatrix} \approx \begin{bmatrix} 0.98058067569092016 \\ 0.19611613513818404 \end{bmatrix}.$$

As the standard rotation matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

rotates points counterclockwise about the origin, choose

$$R = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

to rotate the major and minor axis of the ellipse clockwise so they line up with the x and y axis of the cartesian plane. Code to implement and test domain in Julia is given by

```
using LinearAlgebra

function domain(x)
    a=[1.5,2.0]
    b=[6.5,3.0]
    l=2.6
    c=(a+b)/2
    w=norm(b-a)
    cs=(b-c)/norm(b-c)
    S=[2/w 0.0; 0.0 2/l]
    R=[cs[1] cs[2]; -cs[2] cs[1]]
    return 2-(x-c)'*R'*S*S*R*(x-c)
end

println("on boundary domain([1.5,2.0]=",domain([1.5,2.0]))
println("on boundary domain([6.5,3.0]=",domain([6.5,3.0]))
println("inside domain([4.0,2.5])=",domain([4.0,2.5]))
println("outside domain([7.0,3.5])=",domain([7.0,3.5]))
```

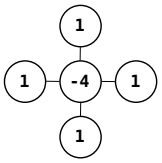
The output

```
on boundary domain([1.5,2.0]=0.9999999999999996
on boundary domain([6.5,3.0]=0.9999999999999996
inside domain([4.0,2.5])=2.0
outside domain([7.0,3.5])=0.39417387346381405
```

demonstrates for a few keys points that

$$\begin{aligned} &\texttt{domain}(x) > 1 \quad \text{for } x \text{ inside the ellipse,} \\ &\texttt{domain}(x) = 1 \quad \text{for } x \text{ on the boundary,} \\ &\texttt{domain}(x) < 1 \quad \text{for } x \text{ outside the ellipse.} \end{aligned}$$

3. Subdivide the bounding box horizontally using `mres` $= 100$ grid points and vertically with the same spacing. This is how the program `poisson.jl` from class creates the grid. Then use the 5-point stencil



to create an approximate solution u_{ap} to Poisson's equation. Define

$$\|u\|^2 = \sum_{x\in\mathcal{J}} \big|u(x)\big|^2 h^2 \qquad \text{where} \qquad h = \frac{6}{\texttt{mres}-1}, \qquad \mathcal{J} = \big\{\, x_{ij} : x_{ij} \in \Omega \,\big\}$$

and x_{ij} are the grid points. Find the error $\|u_{\text{ap}} - u_{\text{ex}}\|$ in your approximation and check that it is approximately 6.5213971543769285. What is the relative error $\|u_{\text{ap}} - u_{\text{ex}}\|/\|u_{\text{ex}}\|$?

Modifications to the code developed in class for the domain specified above and the functions f and ψ led to

```
#  poisson.jl -- Solve the Poisson Equation
#
#  del^2 y = f for x such that domain(x)>c
#  y = psi for x such that domain(x)=c

using SparseArrays, LinearAlgebra, Plots

xmin=[1,1]
xmax=[7,4]
mres=100

# specify the boundary condition
function psi(x)
    return 2*x[1]^4*cos(3*x[2])+exp(2*x[2])*sin(2*x[1])
end

# specify the distribution
function f(x)
    return 6*x[1]^2*(4-3*x[1]^2)*cos(3*x[2])
end

# create spatial grid to fit the desired resolution
if xmax[1]-xmin[1]<xmax[2]-xmin[2]
    dx=(xmax[2]-xmin[2])/(mres-1)
else
    dx=(xmax[1]-xmin[1])/(mres-1)
end
nu=Int.(round.((xmax-xmin)/dx).+1)
image=zeros(Int,nu...)
xmax=xmin+dx*(nu.-1)
x1s=[xmin[1]+(i1-1)*dx for i1=1:size(image)[1]]
x2s=[xmin[2]+(i2-1)*dx for i2=1:size(image)[2]]
function xi(i)
    return [x1s[i[1]],x2s[i[2]]]
end

# specify the domain
function domain(x)
    a=[1.5,2.0]
    b=[6.5,3.0]
    l=2.6
    c=(a+b)/2
    w=norm(b-a)
    cs=(b-c)/norm(b-c)
    S=[2/w 0.0; 0.0 2/l]
    R=[cs[1] cs[2]; -cs[2] cs[1]]
    return 2-(x-c)'*R'*S*S*R*(x-c)
end

# count the grid points inside the domain
function enumerate()
    count=0
```

```
    for i1=2:size(image)[1]-1
        for i2=2:size(image)[2]-1
            if domain(xi([i1,i2]))>1.0
                count+=1
                image[i1,i2]=count
            end
        end
    end
    return count
end

# create A and F using the 4-point stencil
function stencil(count)
    xs=[]; ys=[]; as=[]
    fv=Vector{Float64}(undef,count)
    for i1=1:size(image)[1]
        for i2=1:size(image)[2]
            function look(j1,j2)
                if image[j1,j2]>0
                    push!(xs,image[i1,i2])
                    push!(ys,image[j1,j2])
                    push!(as,1.0)
                else
                    fv[image[i1,i2]]-=psi(xi([j1,j2]))
                end
            end
            if image[i1,i2]>0
                fv[image[i1,i2]]=dx^2*f(xi([i1,i2]))
                push!(xs,image[i1,i2])
                push!(ys,image[i1,i2])
                push!(as,-4.0)
                look(i1,i2-1)
                look(i1-1,i2)
                look(i1+1,i2)
                look(i1,i2+1)
            end
        end
    end
    return (sparse(Int.(xs),Int.(ys),Float64.(as)),fv)
end

# by construction the exact solution is psi
function exact(count)
    xv=zeros(count)
    for i1=1:size(image)[1]
        for i2=1:size(image)[2]
            if image[i1,i2]>0
                xv[image[i1,i2]]=psi(xi([i1,i2]))
            end
        end
    end
    return xv
end
```

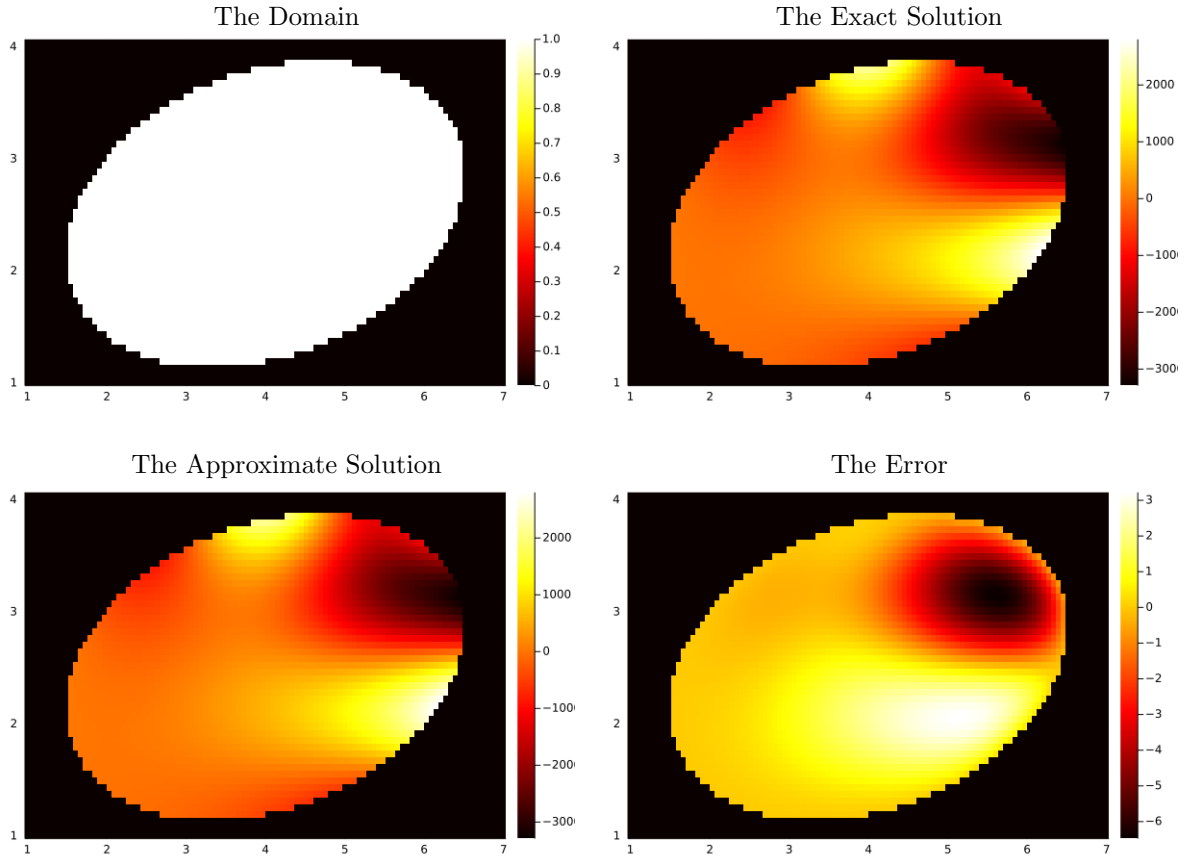
```
106
107 # convert a solution back to the grid
108 function physical(x)
109     black=minimum(x)
110     p=zeros(nu...)
111     for i1=1:size(image)[1]
112         for i2=1:size(image)[2]
113             if image[i1,i2]>0
114                 p[i1,i2]=x[image[i1,i2]]
115             else
116                 p[i1,i2]=black
117             end
118         end
119     end
120     return p
121 end
122
123 # various plotting functions
124 function pltdomain()
125     heatmap(x1s,x2s,(x->min(1,x)).(image)',c=:hot)
126 end
127 function plterror()
128     heatmap(x1s,x2s,physical(Uap-Uex)',c=:hot)
129 end
130 function pltexact()
131     heatmap(x1s,x2s,physical(Uex)',c=:hot)
132 end
133 function pltapprox()
134     heatmap(x1s,x2s,physical(Uap)',c=:hot)
135 end
136
137 # solve the problem and print the error
138 count=enumerate()
139 Uex=exact(count)
140 A,F=stencil(count)
141 Uap=A\F
142 nerr=norm(Uap-Uex)*dx
143 println("When mres=",mres," then |Uap-Uex|=",nerr)
144 println("Relative error |Uap-Uex|/|Uex|=",nerr/(norm(Uex)*dx))
145
146 pltdomain()
147 savefig("domain.png")
148 plterror()
149 savefig("error.png")
150 pltexact()
151 savefig("exact.png")
152 pltapprox()
153 savefig("approx.png")
```

with output

```
When mres=100 then |Uap-Uex|=6.521397154376893
Relative error |Uap-Uex|/|Uex|=0.002093026176747157
```

Note the computed error agrees with the value given in the problem.

For reference, we also include plots of the domain, exact solution, approximation and error.



4. Let $E_n = \|u_{\text{ap}} - u_{\text{ex}}\|$ be the error when `mres` $= n$. Verify the order of convergence of the 5-point stencil is $\mathcal{O}(h^2)$ by checking that $E_n/E_{2n} \approx 4$ for $n = 100$, 200, 400 and 800. Although the boundary is only resolved with $\mathcal{O}(h)$, why in this case does that not affect the convergence of the solution?

All code from the previous question was enclosed into a routine called `en(mres)` so that the calculation could easily be repeated for different values of `mres` $= n$. This led to the program

```
#  poisson.jl -- Solve the Poisson Equation
#
#  del^2 y = f for x such that domain(x)>c
#  y = psi for x such that domain(x)=c

using SparseArrays, LinearAlgebra, Plots, Printf

function en(mres)
    xmin=[1,1]
    xmax=[7,4]

    # specify the boundary condition
    function psi(x)
        return 2*x[1]^4*cos(3*x[2])+exp(2*x[2])*sin(2*x[1])
    end

    # specify the distribution
    function f(x)
```

```
        return 6*x[1]^2*(4-3*x[1]^2)*cos(3*x[2])
    end

    # create spatial grid to fit the desired resolution
    if xmax[1]-xmin[1]<xmax[2]-xmin[2]
        dx=(xmax[2]-xmin[2])/(mres-1)
    else
        dx=(xmax[1]-xmin[1])/(mres-1)
    end
    nu=Int.(round.((xmax-xmin)/dx).+1)
    image=zeros(Int,nu...)
    xmax=xmin+dx*(nu.-1)
    x1s=[xmin[1]+(i1-1)*dx for i1=1:size(image)[1]]
    x2s=[xmin[2]+(i2-1)*dx for i2=1:size(image)[2]]
    function xi(i)
        return [x1s[i[1]],x2s[i[2]]]
    end

    # specify the domain
    function domain(x)
        a=[1.5,2.0]
        b=[6.5,3.0]
        l=2.6
        c=(a+b)/2
        w=norm(b-a)
        cs=(b-c)/norm(b-c)
        S=[2/w 0.0; 0.0 2/l]
        R=[cs[1] cs[2]; -cs[2] cs[1]]
        return 2-(x-c)'*R'*S*S*R*(x-c)
    end

    # count the grid points inside the domain
    function enumerate()
        count=0
        for i1=2:size(image)[1]-1
            for i2=2:size(image)[2]-1
                if domain(xi([i1,i2]))>1.0
                    count+=1
                    image[i1,i2]=count
                end
            end
        end
        return count
    end

    # create A and F using the 4-point stencil
    function stencil(count)
        xs=[]; ys=[]; as=[]
        fv=Vector{Float64}(undef,count)
        for i1=1:size(image)[1]
            for i2=1:size(image)[2]
                function look(j1,j2)
                    if image[j1,j2]>0
```

```
                        push!(xs,image[i1,i2])
                        push!(ys,image[j1,j2])
                        push!(as,1.0)
                    else
                        fv[image[i1,i2]]-=psi(xi([j1,j2]))
                    end
                end
                if image[i1,i2]>0
                    fv[image[i1,i2]]=dx^2*f(xi([i1,i2]))
                    push!(xs,image[i1,i2])
                    push!(ys,image[i1,i2])
                    push!(as,-4.0)
                    look(i1,i2-1)
                    look(i1-1,i2)
                    look(i1+1,i2)
                    look(i1,i2+1)
                end
            end
        end
        return (sparse(Int.(xs),Int.(ys),Float64.(as)),fv)
    end

    # by construction the exact solution is psi
    function exact(count)
        xv=zeros(count)
        for i1=1:size(image)[1]
            for i2=1:size(image)[2]
                if image[i1,i2]>0
                    xv[image[i1,i2]]=psi(xi([i1,i2]))
                end
            end
        end
        return xv
    end

    # solve the problem and print the error
    count=enumerate()
    Uex=exact(count)
    A,F=stencil(count)
    Uap=A\F
    nerr=norm(Uap-Uex)*dx
    return nerr
end

@printf("%6s %22s %22s\n","n","E(n)","E(n)/E(2n)")
for mr in [100,200,400,800]
    @printf("%6d %22.15e %22.15f\n",mr,en(mr),en(mr)/en(2*mr))
end
```

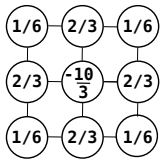
with output

```
     n                   E(n)             E(n)/E(2n)
   100  6.521397154376893e+00      4.126494235784221
   200  1.580372292253434e+00      4.058703661426537
```

```
400  3.893785858950766e-01      4.027990973798121
800  9.666818729931737e-02      4.014930015395115
```

We note that $E_n/E_{2n} \approx 4$ for the values of n requested. Even though the boundary was resolved to only first order, the values on the boundary were taken directory from the solution u_{ex} as given by ψ. Thus, the values on the shifted boundary remained exact for the solution being approximated.

5. [Extra Credit and Math 667] Modify your program to use the the 9-point stencil



Compute E_n/E_{2n} for $n = 100$, 200, 400 and 800. What is the apparent order of the method? Is this consistent with the theoretical result you expected? Replace f by $g = f + \frac{1}{12}h^2\Delta f$. Now what happens?

The Mathematica script

```
psi = 2*x^4*Cos[3*y] + Exp[2*y]*Sin[2*x]
f = Simplify[D[psi, {x, 2}] + D[psi, {y, 2}]]
laplacef = Simplify[D[f, {x, 2}] + D[f, {y, 2}]]
```

found Δf as

```
Wolfram Language 12.1.1 Engine for Linux ARM (32-bit)
Copyright 1988-2020 Wolfram Research, Inc.

In[1]:= psi = 2*x^4*Cos[3*y] + Exp[2*y]*Sin[2*x]

           4             2 y
Out[1]= 2 x  Cos[3 y] + E    Sin[2 x]

In[2]:= f = Simplify[D[psi, {x, 2}] + D[psi, {y, 2}]]

           2         2
Out[2]= 6 x  (4 - 3 x ) Cos[3 y]

In[3]:= laplacef = Simplify[D[f, {x, 2}] + D[f, {y, 2}]]

                   2       4
Out[3]= 6 (8 - 72 x  + 27 x ) Cos[3 y]

In[4]:=
```

Therefore the modified distribution g is

$$g(x) = f(x) + \frac{1}{2}h^2\big(8 - (72 - 27x_1^2)x_1^2\big)\cos 3x_2$$

This function and changes to add the 9-point stencil were incorporated into the previous code to obtain

```
#  poisson.jl -- Solve the Poisson Equation
#
#  del^2 y = f for x such that domain(x)>c
#  y = psi for x such that domain(x)=c

```

```
using SparseArrays, LinearAlgebra, Plots, Printf

# specify the distribution
function f(x,dx)
    return 6*x[1]^2*(4-3*x[1]^2)*cos(3*x[2])
end

# specify the modified distribution
function laplacef(x)
    return 6*(8-(72-27*x[1]^2)*x[1]^2)*cos(3*x[2])
end
function g(x,dx)
    return f(x,dx)+1/12*dx^2*laplacef(x)
end

function en(mres,f)
    xmin=[1,1]
    xmax=[7,4]

    # specify the boundary condition
    function psi(x)
        return 2*x[1]^4*cos(3*x[2])+exp(2*x[2])*sin(2*x[1])
    end

    # create spatial grid to fit the desired resolution
    if xmax[1]-xmin[1]<xmax[2]-xmin[2]
        dx=(xmax[2]-xmin[2])/(mres-1)
    else
        dx=(xmax[1]-xmin[1])/(mres-1)
    end
    nu=Int.(round.((xmax-xmin)/dx).+1)
    image=zeros(Int,nu...)
    xmax=xmin+dx*(nu.-1)
    x1s=[xmin[1]+(i1-1)*dx for i1=1:size(image)[1]]
    x2s=[xmin[2]+(i2-1)*dx for i2=1:size(image)[2]]
    function xi(i)
        return [x1s[i[1]],x2s[i[2]]]
    end

    # specify the domain
    function domain(x)
        a=[1.5,2.0]
        b=[6.5,3.0]
        l=2.6
        c=(a+b)/2
        w=norm(b-a)
        cs=(b-c)/norm(b-c)
        S=[2/w 0.0; 0.0 2/l]
        R=[cs[1] cs[2]; -cs[2] cs[1]]
        return 2-(x-c)'*R'*S*S*R*(x-c)
    end

    # count the grid points inside the domain
```

```
    function enumerate()
        count=0
        for i1=2:size(image)[1]-1
            for i2=2:size(image)[2]-1
                if domain(xi([i1,i2]))>1.0
                    count+=1
                    image[i1,i2]=count
                end
            end
        end
        return count
    end

    # create A and F using the 4-point stencil
    function stencil(count)
        xs=[]; ys=[]; as=[]
        fv=Vector{Float64}(undef,count)
        for i1=1:size(image)[1]
            for i2=1:size(image)[2]
                function look(j1,j2,v)
                    if image[j1,j2]>0
                        push!(xs,image[i1,i2])
                        push!(ys,image[j1,j2])
                        push!(as,v)
                    else
                        fv[image[i1,i2]]-=psi(xi([j1,j2]))*v
                    end
                end
                if image[i1,i2]>0
                    fv[image[i1,i2]]=dx^2*f(xi([i1,i2]),dx)
                    push!(xs,image[i1,i2])
                    push!(ys,image[i1,i2])
                    push!(as,-10/3)
                    look(i1,i2-1,2/3)
                    look(i1-1,i2,2/3)
                    look(i1+1,i2,2/3)
                    look(i1,i2+1,2/3)
                    look(i1-1,i2-1,1/6)
                    look(i1-1,i2+1,1/6)
                    look(i1+1,i2-1,1/6)
                    look(i1+1,i2+1,1/6)
                end
            end
        end
        return (sparse(Int.(xs),Int.(ys),Float64.(as)),fv)
    end

    # by construction the exact solution is psi
    function exact(count)
        xv=zeros(count)
        for i1=1:size(image)[1]
            for i2=1:size(image)[2]
                if image[i1,i2]>0
```

```
                    xv[image[i1,i2]]=psi(xi([i1,i2]))
                end
            end
        end
        return xv
    end

    # solve the problem and print the error
    count=enumerate()
    Uex=exact(count)
    A,F=stencil(count)
    Uap=A\F
    nerr=norm(Uap-Uex)*dx
    return nerr
end

@printf("Using the original f:\n\n")
@printf("%6s %22s %22s\n","n","E(n)","E(n)/E(2n)")
e1=en(100,f)
for mr in [100,200,400,800]
    global e1,e2
    e2=en(mr*2,f)
    @printf("%6d %22.15e %22.15f\n",mr,e1,e1/e2)
    e1=e2
end

@printf("\nUsing the modified f+1/12*dx^2*Delta[f]:\n\n")
@printf("%6s %22s %22s\n","n","E(n)","E(n)/E(2n)")
e1=en(100,g)
for mr in [100,200,400,800]
    global e1,e2
    e2=en(mr*2,g)
    @printf("%6d %22.15e %22.15f\n",mr,e1,e1/e2)
    e1=e2
end
```

with output

```
Using the original f:

     n                   E(n)             E(n)/E(2n)
   100  5.506725039597601e+00      4.137098782611386
   200  1.331059597305939e+00      4.063778334049538
   400  3.275423726125197e-01      4.029621541734892
   800  8.128365634840769e-02      4.015998497012846

Using the modified f+1/12*dx^2*Delta[f]:

     n                   E(n)             E(n)/E(2n)
   100  5.057645770848039e-03     16.736901026987354
   200  3.021853186974612e-04     16.344990773280824
   400  1.848794672869708e-05     16.133065758602214
   800  1.145966117372282e-06     14.261569231727091
```

The apparent order of the method when using the original distribution f is again 2 even though the 9-point stencil is used. This is consistent with theory as it is necessary to modify the distribution as given by g to obtain higher order.

The apparent order of the method when using the modified distribution g is 4 since $2^4 = 16$ and $E_n/E_{2n} \approx 16$. Note that the last ratio $E_{800}/E_{1600} \approx 14$. This may indicate accumation of rounding error or other inaccuracies in the sparse matrix solver used to find the solution. Indeed, as `mres` $= 1600$ corresponds to `count` $= 739537$, the computer must invert a 739537×739537 matrix to perform this calculation. It is not surprising that some rounding errors would accumulate when employing such large matrices.