Finite Differences for Poisson-Problem: Rectangle and Convergence

TODO: Rewrite for rectangle The *Finite Difference Method* (FDM) is a numerical method to solve *Partial Differential Equations* (PDEs) approximately.

Poisson Problem

We want to solve the Poisson equation

$$\begin{aligned} -\Delta u(x,y) &= f(x,y), & (x,y) &\in [0,L]^2 \\ u(x,0) &= b(x), & x &\in [0,L] \\ u(x,L) &= t(x), & x &\in [0,L] \\ u(L,y) &= r(y), & y &\in [0,L] \\ u(0,y) &= l(y), & y &\in [0,L] \end{aligned}$$

on a unit-suqare with $L = 1$. The domain is discreitize uniformly with N grid-points per dimensions leading to $h = \frac{1}{N-1}$. We discreitze the Laplacian using central finite differences with second order as

$$\Delta u(x,y) = \frac{-u(x+h,y) - u(x-h,y) + 4u(x,y) - u(x,y+h) - u(x,y-h)}{h^2} + \mathcal{O}(h^2)$$

such that the stencil for node values reads

$$\begin{aligned} u_{i,j} &\approx \frac{-u_{i+1,j} - u_{i-1,j} + 4u_{i,j} - u_{i,j+1} - u_{i,j-1}}{h^2} \\ \Leftrightarrow [-\Delta u_h]_\xi &= \frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad \xi \in \Omega_h \end{aligned}$$

In [1]:
```
using LinearAlgebra
using SparseArrays
using Plots
```

Domain and Boundary Conditions

The domain $\bar{\Omega}$ $[0,L]^2$ is split into the nodes $\{(x_i, y_j)\}_{i,j=0\ldots N} \in \bar{\Omega}_h$ with

$$x_i = \frac{L}{N-1}, \qquad y_j = \frac{L}{N-1}$$

The interior nodes are $\{(x_i, y_j)\}_{i,j=1\ldots N-1} \in \Omega_h$. We further need boundary conditions for all four sides of the unit square.

In [2]:

```
struct Rectangle
    Lx::Float64
    Ly::Float64
    Nx::Int64
    Ny::Int64
    hx::Float64
    hy::Float64
    xh::Array{Float64,1}
    yh::Array{Float64,1}
    function Rectangle(Lx, Ly, Nx, Ny)
        hx = Lx/(Nx-1)
        hy = Ly/(Ny-1)
        xh = range(0, Lx, step=hx)
        yh = range(0, Ly, step=hy)
        N = new(Lx, Ly, Nx, Ny, hx, hy, xh, yh)
    end
end

struct UnitSquareUniform
    function UnitSquareUniform(N)
        h = 1/(N-1)
        xh = range(0, 1, step=h)
        yh = range(0, 1, step=h)
        N = Rectangle(1., 1., N, N)
    end
end

test = UnitSquareUniform(10)
display(test)

struct RectangleBCs
    bot
    right
    top
    left
end
```

```
Rectangle(1.0, 1.0, 10, 10, 0.1111111111111111, 0.1111111111111111, [0.0, 0.11
11111111111111, 0.2222222222222222, 0.3333333333333333, 0.4444444444444444, 0.
5555555555555556, 0.6666666666666666, 0.7777777777777778, 0.8888888888888888,
1.0], [0.0, 0.1111111111111111, 0.2222222222222222, 0.3333333333333333, 0.4444
444444444444, 0.5555555555555556, 0.6666666666666666, 0.7777777777777778, 0.88
88888888888888, 1.0])
```

Discretization of Poisson Problem

TODO

- Explain Matrix structure and RHS
- Define Kronecker product and Kronecker sum

In [3]:
```
function Δₕ(Ω::Rectangle)
    ⊗ = kron
    dxx = spdiagm(-1=>ones(Ω.Nx-3), 0=>-2ones(Ω.Nx-2), 1=>ones(Ω.Nx-3))
    dyy = spdiagm(-1=>ones(Ω.Ny-3), 0=>-2ones(Ω.Ny-2), 1=>ones(Ω.Ny-3))
    return 1/Ω.hx^2 * I(Ω.Ny-2) ⊗ dxx + 1/Ω.hy^2 * dyy ⊗ I(Ω.Nx-2)
end

function bₕ(Ω::Rectangle, f, bcs::RectangleBCs)
    Nx = Ω.Nx
    Ny = Ω.Ny
    xInt = Ω.xh[2:end-1]
    yInt = Ω.yh[2:end-1]

    fh = vec(f.(xInt,yInt'))
    bh = 1/Ω.hy^2 .* vec(bcs.bot.(xInt))
    rh = 1/Ω.hx^2 .* vec(bcs.right.(yInt))
    th = 1/Ω.hy^2 .* vec(bcs.top.(xInt))
    lh = 1/Ω.hx^2 .* vec(bcs.left.(yInt))

    bvec = zeros((Nx-2)*(Ny-2))
    bvec += fh
    bvec[1                    : 1      : Nx-2] += bh
    bvec[(Nx-2)*(Ny-2-1)+1   : 1      : end] += th
    bvec[(Nx-2)              : (Nx-2) : end] += rh
    bvec[1                    : (Nx-2) : end] += lh

    return bvec
end

function solvePoisson(Ω::Rectangle, f, bcs::RectangleBCs)
    A = -Δₕ(Ω)
    b = bₕ(Ω, f, bcs)
    return (A) \ b
end

function plotSol(Ω::Rectangle, u, bcs::RectangleBCs, edgeAvg=true)
    pyplot()
    Nx = Ω.Nx
    Ny = Ω.Ny
    uMat = zeros(Nx,Ny)
    uMat[2:end-1,2:end-1] = reshape(u, (Nx-2, Ny-2))
    uMat[2:Nx-1,1] = vec(bcs.bot.(Ω.xh[2:end-1]))
    uMat[2:Nx-1,Ny] = vec(bcs.top.(Ω.xh[2:end-1]))
    uMat[Nx,2:Ny-1] = vec(bcs.right.(Ω.yh[2:end-1]))
    uMat[1,2:Ny-1] = vec(bcs.left.(Ω.yh[2:end-1]))
    if edgeAvg
        uMat[1,1] = 0.5 * (uMat[1,2] + uMat[2,1])
        uMat[1,Ny] = 0.5 * (uMat[1,Ny-1] + uMat[2,Ny])
        uMat[Nx,1] = 0.5 * (uMat[Nx-1,1] + uMat[Nx,2])
        uMat[Nx,Ny] = 0.5 * (uMat[Nx-1,Ny] + uMat[Nx-1,Ny])
    else
        uMat[1,1] = uMat[1,Ny] = uMat[Nx,1] = uMat[Nx,Ny] = 0
    end
    Plots.surface(Ω.xh, Ω.yh, uMat', camera=(35, 35), title="Plot")
end
```

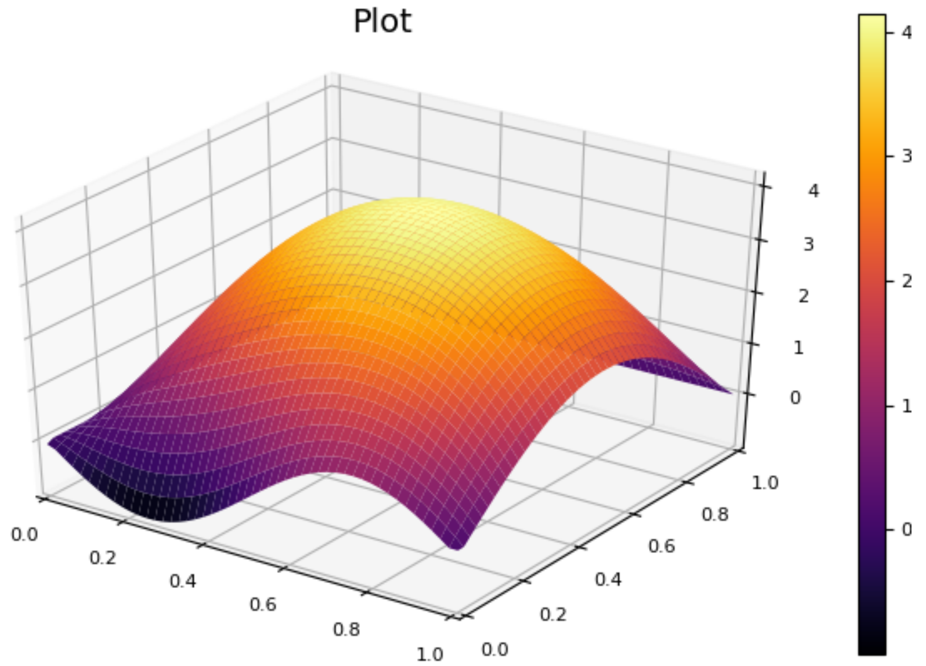
Out[3]:
```
plotSol (generic function with 2 methods)
```

Experiments

- Show unit square from last week
- Show unit square with non-uniform discretization
- Show rectangle with non-uniform discretization

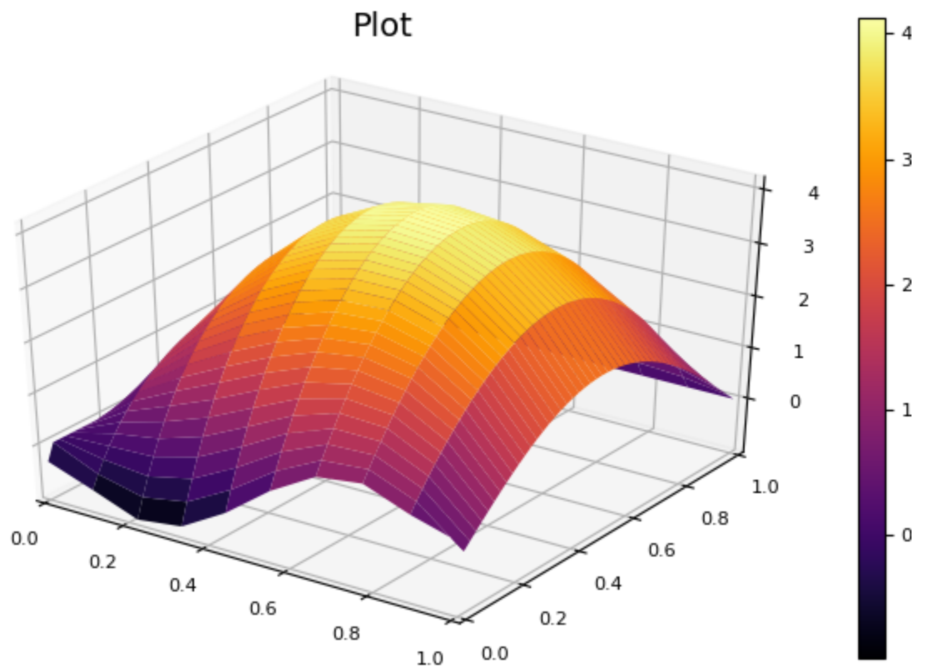
In [4]:
```
Ω = UnitSquareUniform(50)
bcs = RectangleBCs(x -> -sin(2pi*x), y -> 2sin(pi*y), x -> 0, y -> 0)
f(x,y) = 50
u = solvePoisson(Ω, f, bcs)
plotSol(Ω, u, bcs)
```

Out[4]:



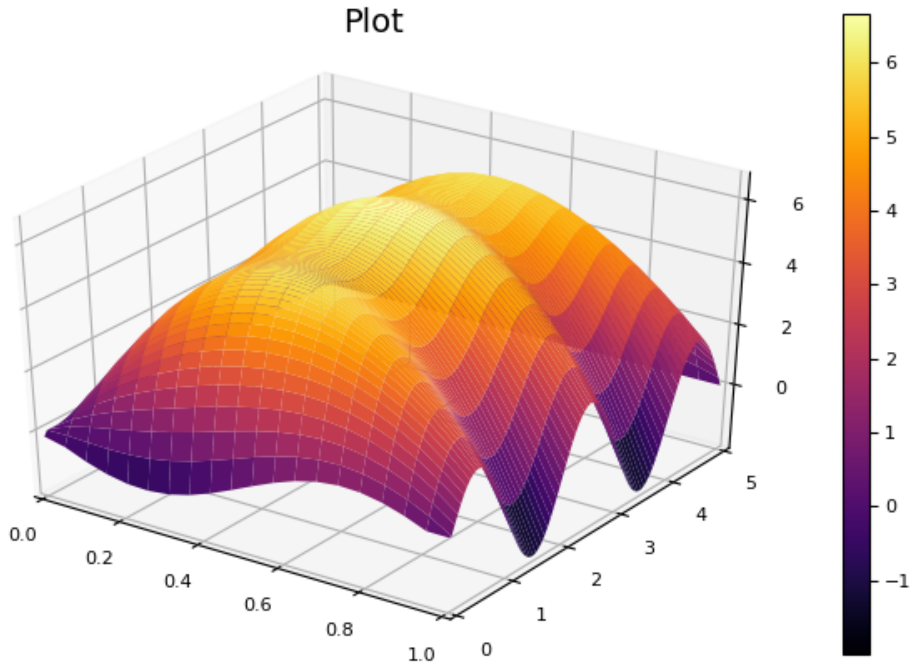
In [5]:
```
Ω = Rectangle(1, 1, 10, 50)
bcs = RectangleBCs(x -> -sin(2pi*x), y -> 2sin(pi*y), x -> 0, y -> 0)
f(x,y) = 50
u = solvePoisson(Ω, f, bcs)
plotSol(Ω, u, bcs)
```

Out[5]:



In [6]:
```
Ω = Rectangle(1, 5, 20, 100)
bcs = RectangleBCs(x -> -sin(2pi*x), y -> 2sin(pi*y), x -> 0, y -> 0)
f(x,y) = 50
u = solvePoisson(Ω, f, bcs)
plotSol(Ω, u, bcs)
```

Out[6]:



Convergence Properties

For convergence, we need a vanishing error $e_h = u|_{\bar{\Omega}_h} - u_h$

$$\|e_h\| \overset{h\to 0}{\to} 0$$

This can be proven by showing stability ($\|\Delta_h^{-1}\| \overset{!}{\leq} C \;\forall h > 0$) and consistency ($\| -\Delta_h u|_{\bar{\Omega}_h} - f|_{\Omega_h}\| \overset{h\to 0}{\to} 0$) due to

$$\|e_h\| \leq \|\Delta_h^{-1}\| \cdot \| -\Delta_h u|_{\bar{\Omega}_h} - f|_{\Omega_h}\|$$

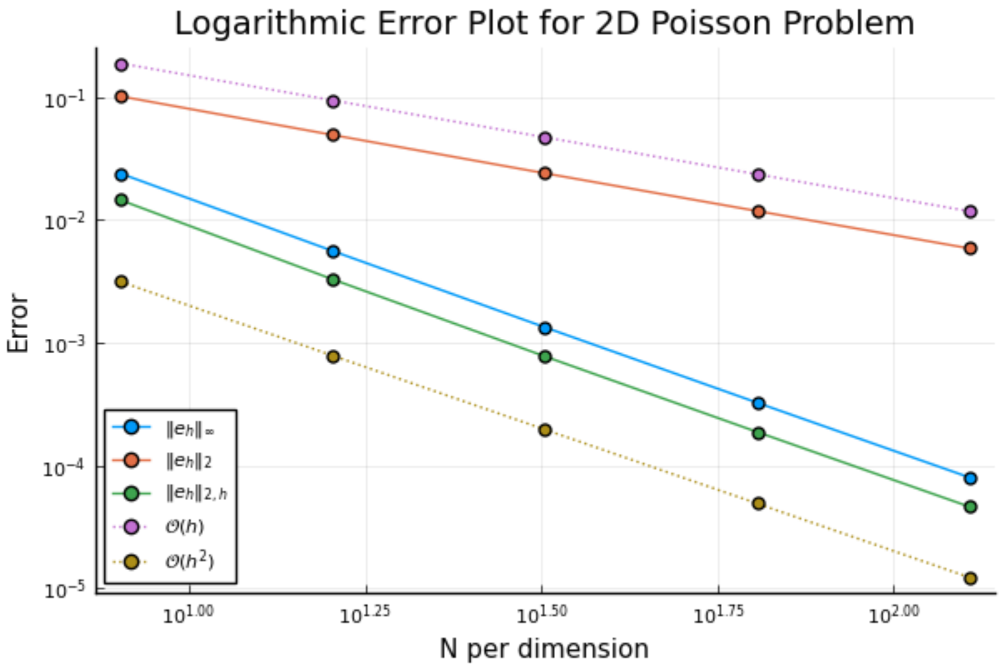
From the lecture, we know that (four our setup):

- $\|e_h\|_\infty \leq \frac{1}{8} \cdot \frac{C}{6} h^2$
- $\|e_h\|_2 \leq \frac{1}{8} \cdot \frac{C}{6} h$
- $\|e_h\|_{2,h} \leq \frac{1}{8} \cdot \frac{C}{6} h^2$

where $\|e_h\|_{2,h} = h\|e_h\|_2$. We observe this numerically.

In [7]:
```
using Printf

errorLInfArray = Vector{Float64}()

uVecDict = Dict{Int64, Vector{Float64}}()
uRealVecDict = Dict{Int64, Vector{Float64}}()
errorVecDict = Dict{Int64, Vector{Float64}}()
ΩDict = Dict{Int64, Rectangle}()

eInfVector = Vector{Float64}()
e2Vector = Vector{Float64}()
e2hVector = Vector{Float64}()

uReal(x,y) = (cosh(2*pi*y) + ((1-cosh(2*pi))/sinh(2*pi))*sinh(2*pi*y) ) * sin(2
*pi*x)

normInf(vec) = norm(vec, Inf)
norm2(vec) = norm(vec, 2)
norm2h(vec, h) = h * norm(vec, 2)

NRange = 2 .^(3:7)

for N = NRange
    Ω = UnitSquareUniform(N)
    ΩDict[N] = Ω
    bcs = RectangleBCs(x -> sin(2pi*x), y -> 0, x -> sin(2pi*x), y -> 0)
    f(x,y) = 0

    uVec = solvePoisson(Ω, f, bcs)
    uRealVec = vec(uReal.(Ω.xh[2:end-1],Ω.yh[2:end-1]'))

    uVecDict[N] = uVec
    uRealVecDict[N] = uRealVec

    errorVec = uVec - uRealVec
    errorVecDict[N] = errorVec

    append!(eInfVector,normInf(errorVec))
    append!(e2Vector,norm2(errorVec))
    append!(e2hVector,norm2h(errorVec, Ω.hx))
end

using LaTeXStrings
plot(NRange, [eInfVector, e2Vector, e2hVector, 1.5 ./NRange, 0.2 ./NRange.^2],
     label=[ L"{\Vert e_h \Vert}_{\infty}" L"{\Vert e_h \Vert}_{2}" L"{\Vert e_
h \Vert}_{2,h}" L"\mathcal{O}(h)" L"\mathcal{O}(h^2)"],
     title="Logarithmic Error Plot for 2D Poisson Problem",
     xaxis=("N per dimension", :log),
     yaxis=("Error", :log),
     marker = (:circle, 5),
     line = [:solid :solid :solid :dot :dot]
 )
```

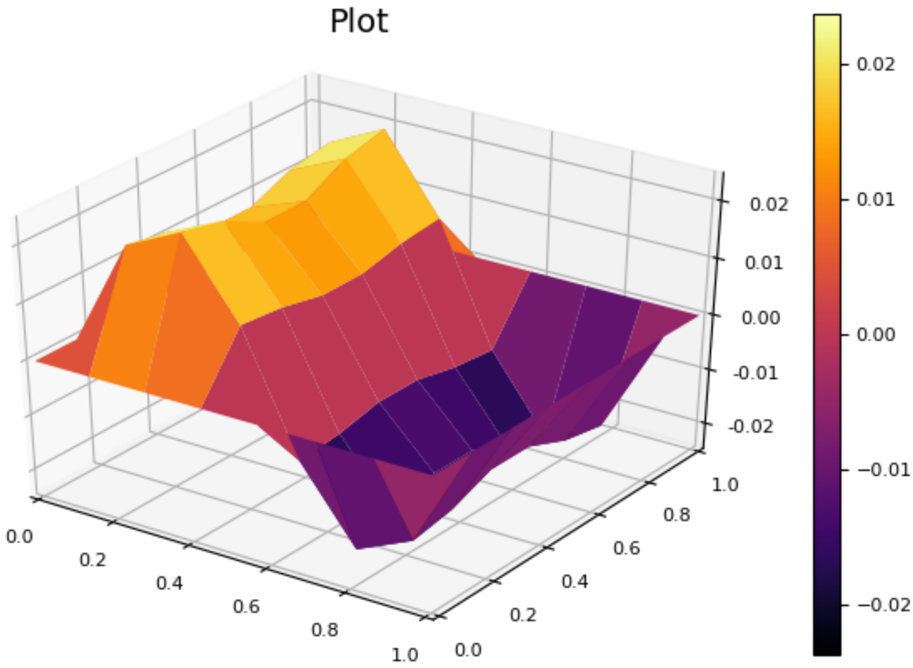
Out[7]:

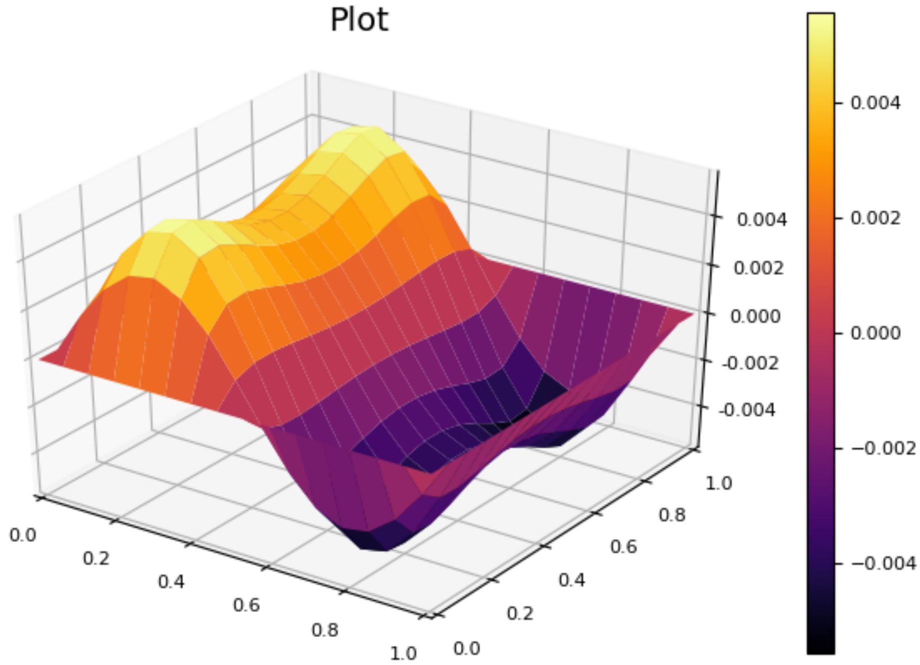


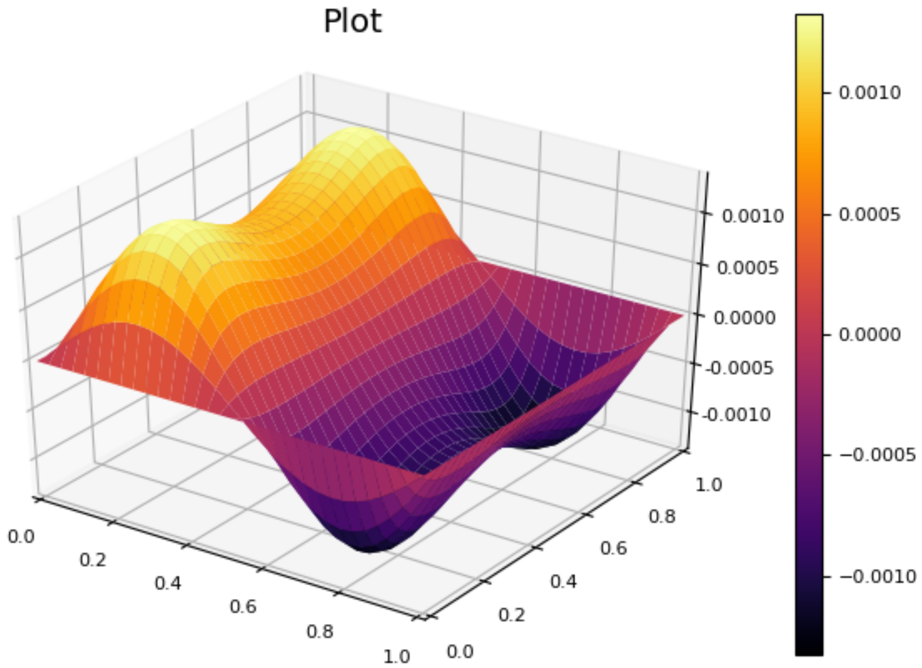
Spatial Distribution of Error

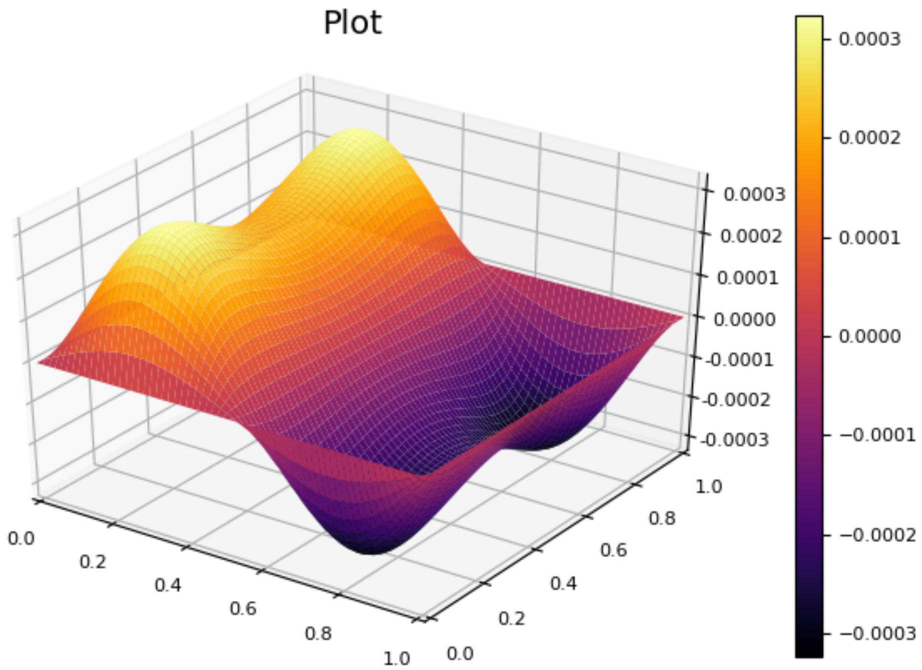
Related to eigenfunctions, see later in lecture...

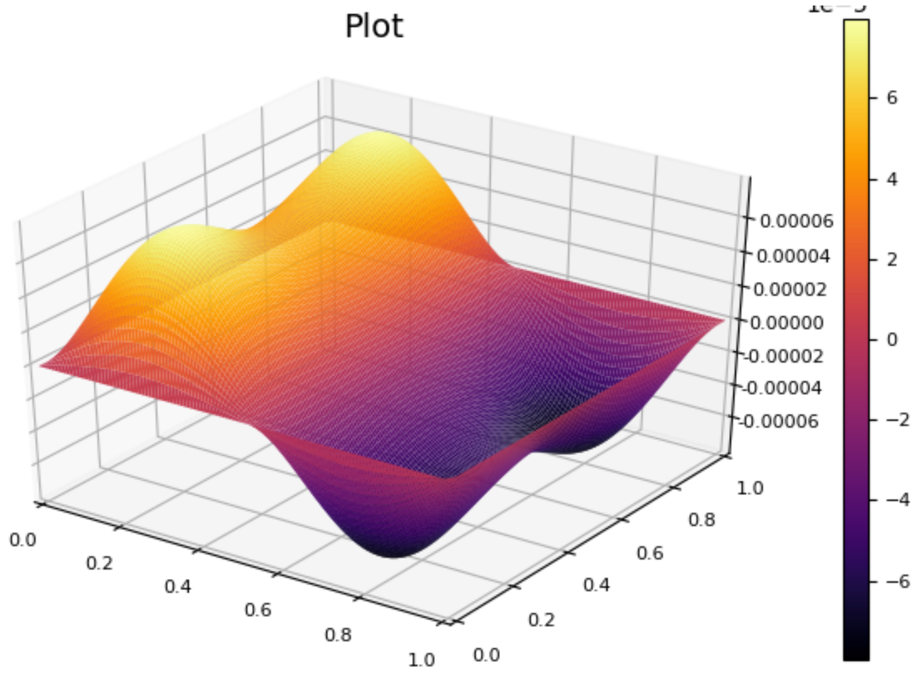
In [8]:
```
for N = NRange
    display(plotSol(ΩDict[N], errorVecDict[N], RectangleBCs(x->0, y->0, x->0, y
->0), false))
end
```

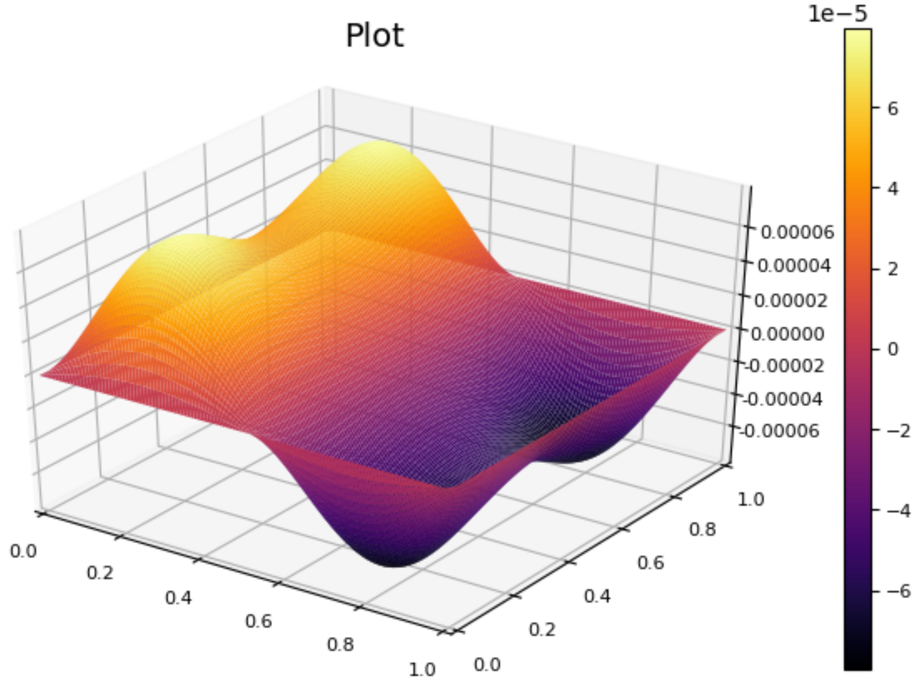

Plot
0.0
0.2
0.4
0.6
0.8
1.0
0.02
0.01
0.00
-0.01
-0.02
−0.01
−0.02


Plot
0.0
0.2
0.4
0.6
0.8
1.0
0.004
0.002
0.000
-0.002
-0.004
−0.002
−0.004


Plot
0.0010
0.0005
0.0000
-0.0005
-0.0010
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.0010
0.0005
0.0000
−0.0005
−0.0010


Plot
0.0003
0.0002
0.0001
0.0000
-0.0001
-0.0002
-0.0003
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.0003
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.0003


Plot
1e−5
6
4
2
0
−2
−4
−6
0.00006
0.00004
0.00002
0.00000
-0.00002
-0.00004
-0.00006
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


Plot
1e−5
6
4
2
0
−2
−4
−6
0.00006
0.00004
0.00002
0.00000
-0.00002
-0.00004
-0.00006
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0