Julia_notebook_week2

January 12, 2024

1 Exercise 2.1

[1]:
```julia
using LinearAlgebra
using SparseArrays
using Plots
using Polynomials
using SymRCM
```

[2]:
```julia
function xy(
    x0::Float64,
    y0::Float64,
    L1::Float64,
    L2::Float64,
    noelms1::Int64,
    noelms2::Int64
)::Tuple{Vector{Float64}, Vector{Float64}}
    VX = repeat(collect(LinRange(x0, x0+L1, noelms1+1)), inner=noelms2+1)
    VY = repeat(collect(LinRange(y0+L2, y0, noelms2+1)), noelms1+1)
    return VX, VY
end

x0 = -2.5
y0 = -4.8
L1 = 7.6
L2 = 5.9
noelms1 = 4
noelms2 = 3

VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
display(VX)
display(VY)
```

```
20-element Vector{Float64}:
 -2.5
 -2.5
 -2.5
 -2.5
 -0.6000000000000001
```

```
 -0.6000000000000001
 -0.6000000000000001
 -0.6000000000000001
  1.2999999999999998
  1.2999999999999998
  1.2999999999999998
  1.2999999999999998
  3.1999999999999997
  3.1999999999999997
  3.1999999999999997
  3.1999999999999997
  5.1
  5.1
  5.1
  5.1
```

```
20-element Vector{Float64}:
  1.1000000000000005
 -0.8666666666666661
 -2.833333333333333
 -4.8
  1.1000000000000005
 -0.8666666666666661
 -2.833333333333333
 -4.8
  1.1000000000000005
 -0.8666666666666661
 -2.833333333333333
 -4.8
  1.1000000000000005
 -0.8666666666666661
 -2.833333333333333
 -4.8
  1.1000000000000005
 -0.8666666666666661
 -2.833333333333333
 -4.8
```

[3]:
```
function conelmtab(noelms1::Int64, noelms2::Int64)::Matrix{Int64}
    k = [i for i in 1:(noelms1*(noelms2+1)) if i % (noelms2+1) != 0]

    return [
        k (2 + noelms2 .+ k) (1 + noelms2 .+ k);
        k (1 .+ k) (2 + noelms2 .+ k)
    ]
end

EToV = conelmtab(noelms1, noelms2)
```

```
display(EToV)
```

```
24×3 Matrix{Int64}:
  1   6   5
  2   7   6
  3   8   7
  5  10   9
  6  11  10
  7  12  11
  9  14  13
 10  15  14
 11  16  15
 13  18  17
 14  19  18
 15  20  19
  1   2   6
  2   3   7
  3   4   8
  5   6  10
  6   7  11
  7   8  12
  9  10  14
 10  11  15
 11  12  16
 13  14  18
 14  15  19
 15  16  20
```

2 Exercise 2.2

[4]:
```
function basfun(n::Int64, VX::Vector{Float64}, VY::Vector{Float64}, EToV::
 ↪Matrix{Int64})::Tuple{Float64, Matrix{Float64}}
    abc = zeros(3,3)

    i1 = EToV[n,1]
    i2 = EToV[n,2]
    i3 = EToV[n,3]

    for (n, (j,k)) in enumerate([[i2,i3], [i3,i1], [i1,i2]])
        abc[n,1] = VX[j]*VY[k] - VX[k]*VY[j]
        abc[n,2] = VY[j] - VY[k]
        abc[n,3] = VX[k] - VX[j]
    end

    delta = sum(abc[:,1])/2

    return delta, abc
```

```
end

function basfun2(VX::Vector{Float64}, VY::Vector{Float64}, EToV::Matrix{Int64}):
    ↪:Tuple{Matrix{Float64}, Matrix{Float64}, Matrix{Float64}}
    i1 = EToV[:,1]
    i2 = EToV[:,2]
    i3 = EToV[:,3]

    as = [(VX[i3].*VY[i1] - VX[i1].*VY[i3]) (VX[i1].*VY[i2] - VX[i2].*VY[i1])␣
    ↪(VX[i2].*VY[i3] - VX[i3].*VY[i2])]
    bs = [(VY[i2] - VY[i3]) (VY[i3] - VY[i1]) (VY[i1] - VY[i2])]
    cs = [(VX[i3] - VX[i2]) (VX[i1] - VX[i3]) (VX[i2] - VX[i1])]

    return as, bs, cs
end

function basfun3(VX, VY, EToV)
    xjs = VX[EToV[:, [2,3,1]]]
    yjs = VY[EToV[:, [2,3,1]]]

    xks = VX[EToV[:, [3,1,2]]]
    yks = VY[EToV[:, [3,1,2]]]

    as = xjs .* yks - xks .* yjs
    bs = yjs - yks
    cs = xks - xjs

    return as, bs, cs
end

n = 4
delta, abc = basfun(n, VX, VY, EToV)
display(delta)
display(abc)

as, bs, cs = basfun2(VX, VY, EToV)
as2, bs2, cs2 = basfun3(VX, VY, EToV)
display(as[n,:])
display(as2[n,:])

display(bs[n,:])
display(bs2[n,:])

display(cs[n,:])
display(cs2[n,:])
```

```
1.8683333333333332
```

```
3×3 Matrix{Float64}:
  2.55667  -1.96667   0.0
  2.09      0.0      -1.9
 -0.91      1.96667   1.9

3-element Vector{Float64}:
  2.0900000000000007
 -0.9100000000000008
  2.5566666666666666

3-element Vector{Float64}:
  2.5566666666666666
  2.0900000000000007
 -0.9100000000000008

3-element Vector{Float64}:
 -1.9666666666666668
  0.0
  1.9666666666666668

3-element Vector{Float64}:
 -1.9666666666666668
  0.0
  1.9666666666666668

3-element Vector{Float64}:
  0.0
 -1.9
  1.9

3-element Vector{Float64}:
  0.0
 -1.9
  1.9
```

[5]:

```
function outernormal(
    n::Int64,
    k::Int64,
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64}
)::Tuple{Float64, Float64}
    i1 = EToV[n, k]
    i2 = EToV[n, k % 3 + 1]

    dx = VX[i2] - VX[i1]
    dy = VY[i2] - VY[i1]

    norm = sqrt(dx^2 + dy^2)
    n1 = dy / norm
```

```
    n2 = -dx / norm

    return n1, n2
end

for k in 1:3
    display(outernormal(n,k,VX,VY,EToV))
end
```

(-0.7191913900847712, -0.6948120209293551)

(1.0, -0.0)

(0.0, 1.0)

3 Exercise 2.3

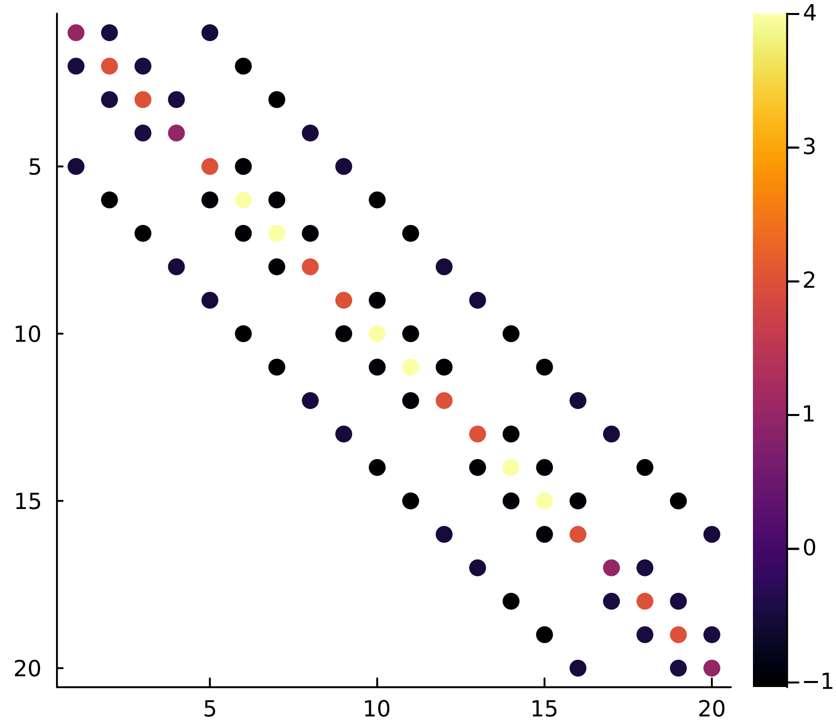
[6]:
```
function assembly(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Vector{Float64}
)::Tuple{Matrix{Float64}, Vector{Float64}}
    N = size(EToV)[1]
    M = length(VX)

    A = spzeros(M, M)
    b = zeros(M)

    for n in 1:N
        delta, abc = basfun(n, VX, VY, EToV)
        q = abs(delta) * sum(qt[EToV[n, :]]) / 9

        for r in 1:3
            i = EToV[n,r]
            b[i] += q

            for s in 1:3
                j = EToV[n,s]
                A[i,j] += (lam1*abc[r, 2]*abc[s, 2] + lam2*abc[r, 3]*abc[s, 3])␣
 ↪/ (4 * abs(delta))
            end
        end
    end

    return A, b
```

```julia
end

lam1 = 1.0
lam2 = 1.0
qt(x, y) = -6*x + 2*y - 2

A, b =  assembly(VX, VY, EToV, lam1, lam2, qt.(VX, VY))
display(A)
display(b)
spy(A, markersize=5)
```

```
20×20 Matrix{Float64}:
  1.00059   -0.483051   0.0        …   0.0         0.0         0.0
 -0.483051   2.00119   -0.483051       0.0         0.0         0.0
  0.0       -0.483051   2.00119        0.0         0.0         0.0
  0.0        0.0       -0.483051       0.0         0.0         0.0
 -0.517544   0.0        0.0            0.0         0.0         0.0
  0.0       -1.03509    0.0        …   0.0         0.0         0.0
  0.0        0.0       -1.03509        0.0         0.0         0.0
  0.0        0.0        0.0            0.0         0.0         0.0
  0.0        0.0        0.0            0.0         0.0         0.0
  0.0        0.0        0.0            0.0         0.0         0.0
  0.0        0.0        0.0        …   0.0         0.0         0.0
  0.0        0.0        0.0            0.0         0.0         0.0
  0.0        0.0        0.0            0.0         0.0         0.0
  0.0        0.0        0.0           -1.03509     0.0         0.0
  0.0        0.0        0.0            0.0        -1.03509     0.0
  0.0        0.0        0.0        …   0.0         0.0        -0.517544
  0.0        0.0        0.0           -0.483051    0.0         0.0
  0.0        0.0        0.0            2.00119    -0.483051    0.0
  0.0        0.0        0.0           -0.483051    2.00119    -0.483051
  0.0        0.0        0.0            0.0        -0.483051    1.00059

20-element Vector{Float64}:
    9.383185185185186
    9.950604938271605
    2.6018271604938272
    0.5674197530864201
   -0.899567901234565
   -0.4982222222222199
  -15.195777777777774
   -6.9474320987654306
  -22.19856790123456
  -43.09622222222222
  -57.79377777777776
  -28.24643209876542
  -43.49756790123456
  -85.69422222222221
```

```
-100.39177777777778
  -49.54543209876542
  -17.382419753086424
  -53.046827160493834
  -60.39560493827161
  -43.01318518518518
```

[6]:



4 Exercise 2.4

[7]:

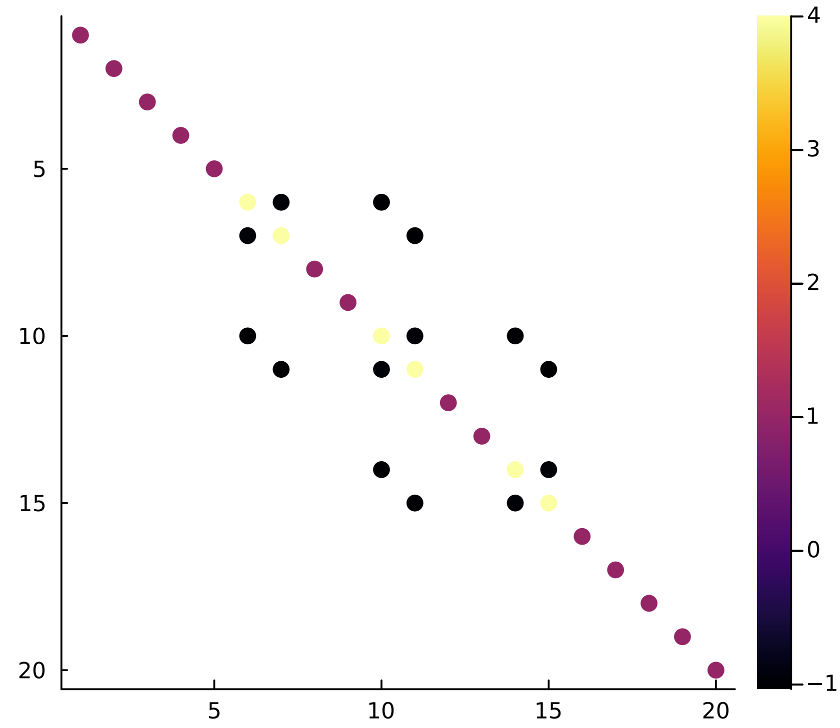
```
function dirbc(
    bnodes::Vector{Int64},
    f::Vector{Float64},
    A::Matrix{Float64},
    b::Vector{Float64}
)::Tuple{Matrix{Float64}, Vector{Float64}}
    A = copy(A)
    b = copy(b)

    for (i, k) in enumerate(bnodes)
        b[k] = f[i]

        A[k, k] = 0
        indices = findall(A[:,k] .!= 0)
```

```julia
        b[indices] -= A[indices, k] .* f[i]

        A[indices, k] .= 0
        A[k, indices] .= 0
        A[k, k] = 1
    end

    return A, b
end

function get_bnodes(noelms1::Int64, noelms2::Int64)
    return sort(vcat(
        1:noelms2 + 1,
        1 + noelms1*(noelms2 + 1) : (noelms1 + 1)*(noelms2 + 1),
        noelms2 + 2 : noelms2 + 1 : (noelms1 - 1)*noelms2 + noelms1,
        2*noelms2 + 2 : noelms2 + 1 : noelms1*(noelms2 + 1)
    ))
end

f(x,y) = x^3 - x^2*y + y^2 -1
bnodes = get_bnodes(noelms1, noelms2)
A, b = dirbc(bnodes, f.(VX[bnodes], VY[bnodes]), A, b)
display(A)
display(b)
spy(A, markersize=5)
```

```
20×20 Matrix{Float64}:
 1.0  0.0  0.0  0.0  0.0   0.0       …   0.0       0.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  1.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   4.00238   …   0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  -0.966102      0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  -1.03509       0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0       …  -1.03509   0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0          -0.966102  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           4.00238   0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0       …   0.0       1.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  1.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  1.0  0.0
 0.0  0.0  0.0  0.0  0.0   0.0           0.0       0.0  0.0  0.0  0.0  1.0
```

```
20-element Vector{Float64}:
 -22.290000000000003
 -10.457222222222226
   9.111111111111109
  36.415
  -0.40199999999999925
 -11.710737403773088
  16.98864856113919
  23.552
   0.5479999999999998
 -42.566798493408655
 -26.5413540489642
  32.349
  21.71399999999999
  95.66445656986153
 220.90473142366272
 103.95999999999998
 104.24999999999997
 154.94411111111106
 213.37377777777775
 279.539
```

[7]:



5 Exercise 2.5

[8]:
```
uhat = A \ b
u(x, y) = x^3 - x^2*y + y^2 - 1
E = maximum(abs.(uhat - u.(VX, VY)))

display(uhat)
display(E)
```

```
20-element Vector{Float64}:
 -22.290000000000003
 -10.457222222222226
   9.111111111111109
  36.415
  -0.40199999999999925
  -0.1528888888888887
   7.831777777777779
  23.552
   0.5479999999999998
   3.4127777777777797
  14.013111111111115
  32.349
  21.713999999999999
  41.39377777777776
  68.8091111111111
 103.95999999999998
 104.24999999999997
 154.94411111111106
 213.37377777777775
 279.539
```

```
7.105427357601002e-15
```

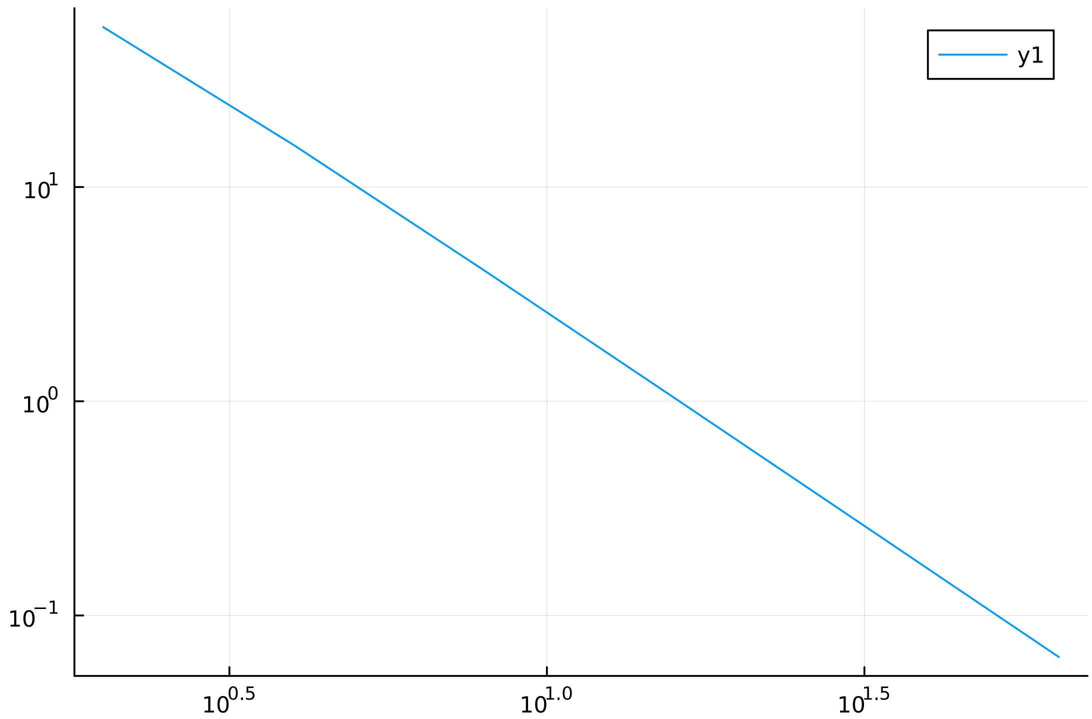
[9]:
```
u(x,y) = x^2 * y^2
qt(x,y) = -2*x^2 - 2*y^2

x0 = -2.5
y0 = -4.8
L1 = 7.6
L2 = 5.9
lam1 = 1.0
lam2 = 1.0

N = 6
dofs = zeros(N)
max_errors = zeros(N)

for p in 1:N
```

```julia
    noelms1 = noelms2 = 2^p

    VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
    EToV = conelmtab(noelms1, noelms2)

    A, b = assembly(VX, VY, EToV, lam1, lam2, qt.(VX, VY))

    bnodes = get_bnodes(noelms1, noelms2)
    A, b = dirbc(bnodes, u.(VX[bnodes], VY[bnodes]), A, b)

    uhat = A \ b

    dofs[p] = noelms1
    max_errors[p] = maximum(abs.(uhat - u.(VX, VY)))
    display("p-values $p, noelms $(dofs[p]), error $(max_errors[p])")
end

display(fit(log.(dofs), log.(max_errors), 1))
plot(dofs, max_errors, xscale=:log10, yscale=:log10)
```

```
"p-values 1, noelms 2.0, error 55.85071111111111"

"p-values 2, noelms 4.0, error 15.652673687141721"

"p-values 3, noelms 8.0, error 4.046801781976418"

"p-values 4, noelms 16.0, error 1.0206836335860272"

"p-values 5, noelms 32.0, error 0.2557434740247917"

"p-values 6, noelms 64.0, error 0.06397183343271617"
```

$5.437814515184364 - 1.9612447301715852 \cdot x$

[9]:



6 Exercise 2.6

[10]:
```
function constructBeds(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    tol::Float64,
    fd::Function,
)::Matrix{Int64}
    xc = (VX[EToV] + VX[EToV[:, [2, 3, 1]]]) ./ 2
    yc = (VY[EToV] + VY[EToV[:, [2, 3, 1]]]) ./ 2
    return getindex.(findall(abs.(fd.(xc, yc)) .<= tol), [1 2])
end

x0 = -2.5
y0 = -4.8
L1 = 7.6
L2 = 5.9
noelms1 = 4
noelms2 = 3

VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
```

```
EToV = conelmtab(noelms1, noelms2)

tol = 0.0001
d(x, y) = min(x - x0, y - y0, x0 + L1 - x, y0 + L2 - y)

beds = constructBeds(VX, VY, EToV, tol, d)
display(beds)
```

```
14×2 Matrix{Int64}:
 13  1
 14  1
 15  1
 10  2
 11  2
 12  2
 15  2
 18  2
 21  2
 24  2
  1  3
  4  3
  7  3
 10  3
```

[11]:

```
function edgeIndices(
    EToV::Matrix{Int64},
    beds::Matrix{Int64},
)::Tuple{Vector{Int64}, Vector{Int64}}
    n = beds[:, 1]
    r = beds[:, 2]
    s = r .% 3 .+ 1

    i = EToV[CartesianIndex.(n, r)]
    j = EToV[CartesianIndex.(n, s)]

    return i,j
end

function neubc(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    beds::Matrix{Int64},
    q::Vector{Float64},
    b::Vector{Float64}
)::Vector{Float64}
    i, j = edgeIndices(EToV, beds)
```

```julia
    q1 = q .* sqrt.((VX[j] - VX[i]).^2 + (VY[j] - VY[i]).^2) ./ 2

    b[i] -= q1
    b[j] -= q1

    return b
end
```

[11]: neubc (generic function with 1 method)

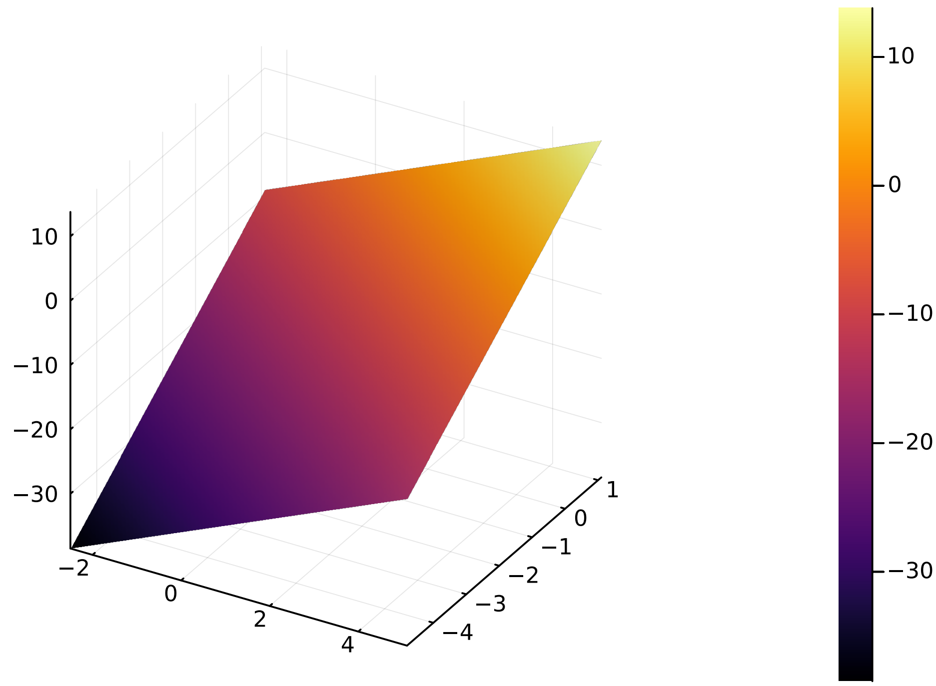
7 Exercise 2.7

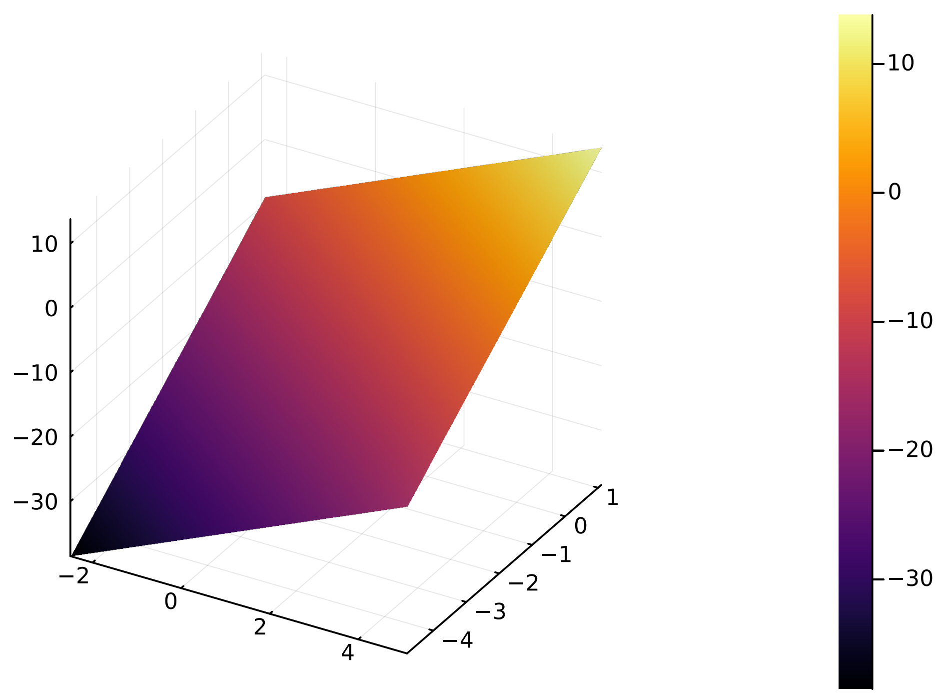
[12]:
```julia
function constructBnodes(
    VX::Vector{Float64},
    VY::Vector{Float64},
    tol::Float64,
    fd::Function,
)::Vector{Int64}
    return findall(abs.(fd.(VX, VY)) .<= tol)
end

function solveNDBVP(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Function,
    q::Function,
    f::Function,
    fd_gamma1::Function,
    fd_gamma2::Function,
    tol::Float64
)::Vector{Float64}
    A, b = assembly(VX, VY, EToV, lam1, lam2, qt.(VX, VY))

    beds = constructBeds(VX, VY, EToV, tol, fd_gamma1)
    i, j = edgeIndices(EToV, beds)
    b = neubc(VX, VY, EToV, beds, q.(VX[i], VY[i], VX[j], VY[j]), b)

    bnodes = constructBnodes(VX, VY, tol, fd_gamma2)
    A, b = dirbc(bnodes, f.(VX[bnodes], VY[bnodes]), A, b)

    uhat = A \ b

    return uhat
end
```

[12]: solveNDBVP (generic function with 1 method)

[13]:
```julia
x0 = -2.5
y0 = -4.8
L1 = 7.6
L2 = 5.9
noelms1 = 4
noelms2 = 3
lam1 = 1.0
lam2 = 1.0
```

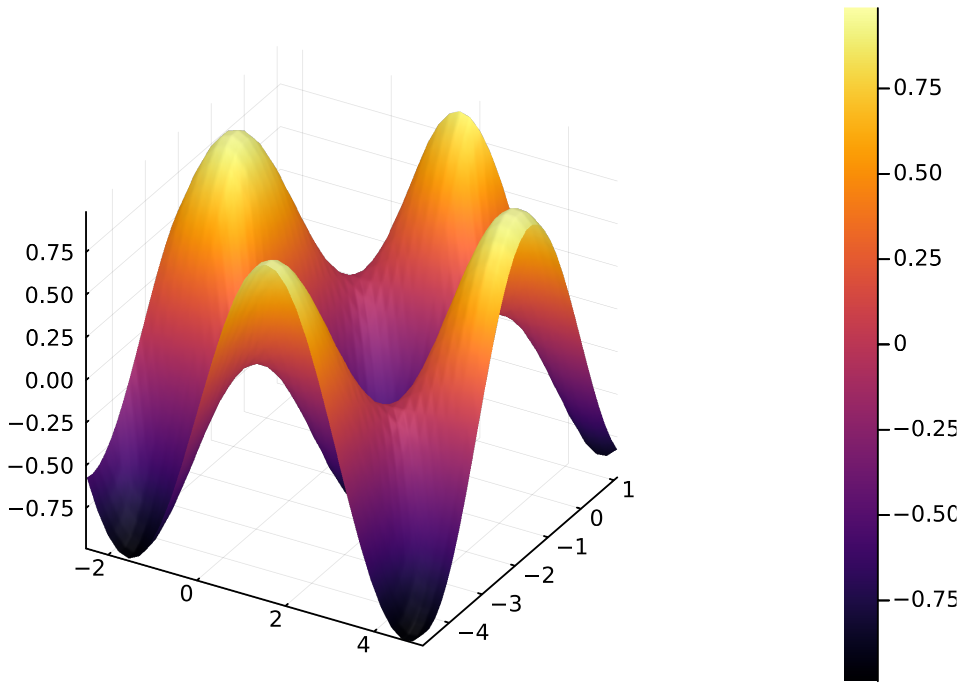
```julia
VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
EToV = conelmtab(noelms1, noelms2)

u(x, y) = 3.0*x + 5.0*y - 7
ux(x, y) = 3.0
uy(x, y) = 5.0
uxx(x, y) = 0.0
uyy(x, y) = 0.0

qt(x, y) = - uxx(x, y) - uyy(x, y)
f(x, y) = u(x, y)
function q(x1, y1, x2, y2)
    dx = x2 - x1
    dy = y2 - y1
    norm = sqrt(dx^2 + dy^2)
    n1 = dy / norm
    n2 = -dx / norm

    xc = (x1 + x2) / 2
    yc = (y1 + y2) / 2

    return - lam1 * ux(xc, yc) * n1 - lam2 * uy(xc, yc) * n2
end

fd_gamma1(x, y) = min(x - x0, y - y0)
fd_gamma2(x, y) = min(x0 + L1 - x, y0 + L2 - y)
tol = 0.0001

uhat = solveNDBVP(VX,VY,EToV,lam1,lam2,qt,q,f,fd_gamma1,fd_gamma2,tol)
display(scatter(VX, VY, uhat, st=:surface))
display(scatter(VX, VY, u.(VX, VY), st=:surface))

E = maximum(uhat - u.(VX, VY))
```

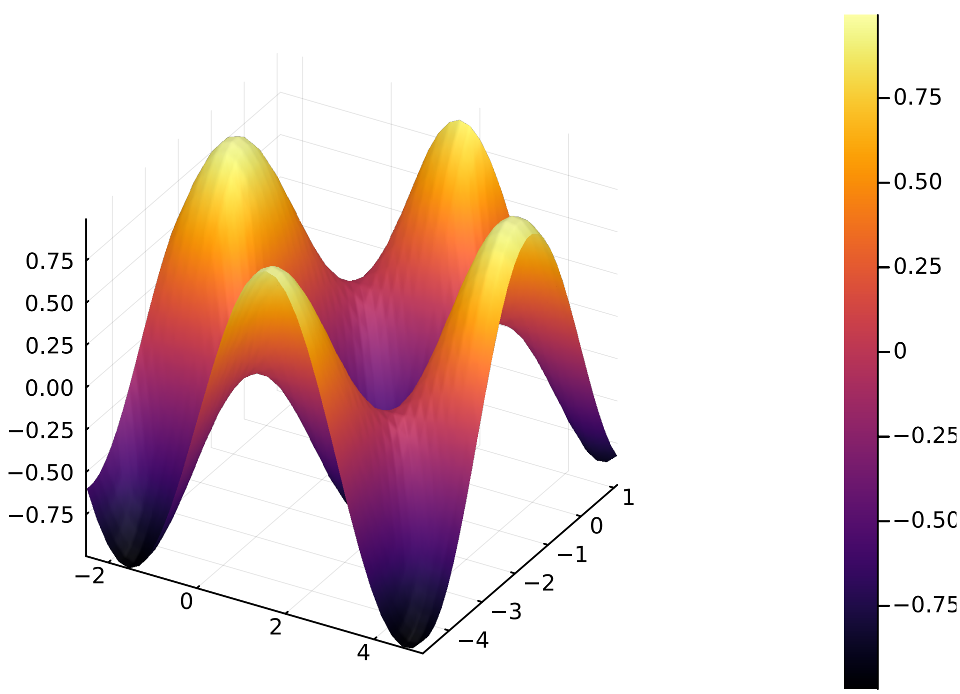

10
0
−10
−20
−30
−2
0
2
4
1
0
−1
−2
−3
−4
10
0
−10
−20
−30


10
0
−10
−20
−30
−2
0
2
4
1
0
−1
−2
−3
−4
10
0
−10
−20
−30

[13]: 0.0

[14]:
```
x0 = -2.5
y0 = -4.8
L1 = 7.6
L2 = 5.9
noelms1 = 32
noelms2 = 32
lam1 = 1.0
lam2 = 1.0

VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
EToV = conelmtab(noelms1, noelms2)

u(x, y) = sin(x)*sin(y)
ux(x, y) = cos(x)*sin(y)
uy(x, y) = sin(x)*cos(y)
uxx(x, y) = -sin(x)*sin(y)
uyy(x, y) = -sin(x)*sin(y)

qt(x, y) = - uxx(x, y) - uyy(x, y)
f(x, y) = u(x, y)
function q(x1, y1, x2, y2)
    dx = x2 - x1
    dy = y2 - y1
    norm = sqrt(dx^2 + dy^2)
    n1 = dy / norm
    n2 = -dx / norm

    xc = (x1 + x2) / 2
    yc = (y1 + y2) / 2

    return - lam1 * ux(xc, yc) * n1 - lam2 * uy(xc, yc) * n2
end

fd_gamma1(x, y) = min(x - x0, y - y0)
fd_gamma2(x, y) = min(x0 + L1 - x, y0 + L2 - y)
tol = 0.0001

uhat = solveNDBVP(VX,VY,EToV,lam1,lam2,qt,q,f,fd_gamma1,fd_gamma2,tol)
display(scatter(VX, VY, uhat, st=:surface))
display(scatter(VX, VY, u.(VX, VY), st=:surface))

E = maximum(uhat - u.(VX, VY))
```

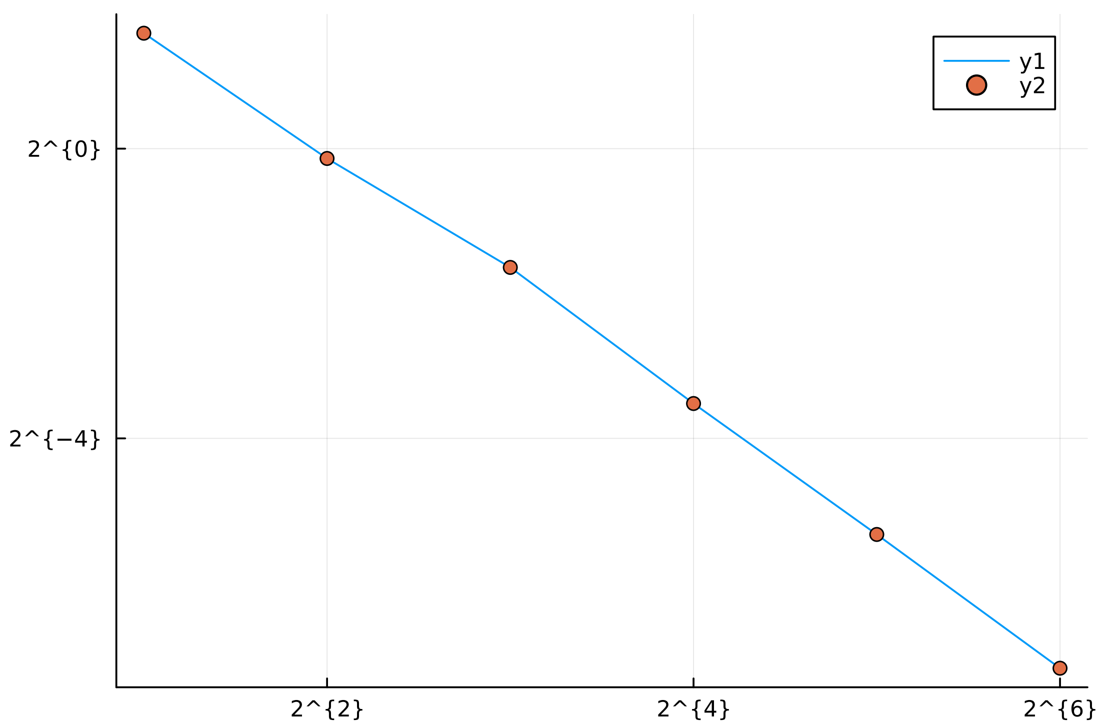

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2
0
2
4
1
0
−1
−2
−3
−4
0.75
0.50
0.25
0
−0.25
−0.50
−0.75


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2
0
2
4
1
0
−1
−2
−3
−4
0.75
0.50
0.25
0
−0.25
−0.50
−0.75

[14]: 0.024919692458461595

[15]:
```julia
N = 6
dof = zeros(N)
max_errors = zeros(N)

for i in 1:N
    noelms = 2^i

    VX, VY = xy(x0, y0, L1, L2, noelms, noelms)
    EToV = conelmtab(noelms, noelms)
    uhat = solveNDBVP(VX,VY,EToV,lam1,lam2,qt,q,f,fd_gamma1,fd_gamma2,tol)

    dof[i] = noelms
    max_errors[i] = maximum(uhat - u.(VX, VY))
end

display(fit(log.(dof[4:end]), log.(max_errors[4:end]), 1))
p = plot(dof, max_errors, xscale=:log2, yscale=:log2)
scatter!(dof, max_errors)
display(p)
```

$2.631696889432387 - 1.8272471381398483 \cdot x$



8 Exercise 2.8

[16]:
```
# Example matrix
A = [1 2 3; 4 5 6; 7 8 9]

# Sum over rows (along dimension 1)
row_sums = sum(A, dims=1)

# Display the result
println("Original Matrix:")
println(A)

println("\nSum over Rows:")
println(row_sums)
```

```
Original Matrix:
[1 2 3; 4 5 6; 7 8 9]

Sum over Rows:
[12 15 18]
```

[17]:
```
using LinearAlgebra
using SparseArrays
using Plots
using Polynomials
using SymRCM
using AMD

function xy(
    x0::Float64,
    y0::Float64,
    L1::Float64,
    L2::Float64,
    noelms1::Int64,
    noelms2::Int64
)::Tuple{Vector{Float64}, Vector{Float64}}
    VX = repeat(collect(LinRange(x0, x0+L1, noelms1+1)), inner=noelms2+1)
    VY = repeat(collect(LinRange(y0+L2, y0, noelms2+1)), noelms1+1)
    return VX, VY
end

function conelmtab(noelms1::Int64, noelms2::Int64)::Matrix{Int64}
    k = [i for i in 1:(noelms1*(noelms2+1)) if i % (noelms2+1) != 0]

    return [
        k (2 + noelms2 .+ k) (1 + noelms2 .+ k);
        k (1 .+ k) (2 + noelms2 .+ k)
    ]
```

```julia
end

function basfun(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64}
)::Tuple{Matrix{Float64}, Matrix{Float64}, Matrix{Float64}}
    xjs = VX[EToV[:, [2,3,1]]]
    yjs = VY[EToV[:, [2,3,1]]]

    xks = VX[EToV[:, [3,1,2]]]
    yks = VY[EToV[:, [3,1,2]]]

    as = xjs .* yks - xks .* yjs
    bs = yjs - yks
    cs = xks - xjs

    return as, bs, cs
end

function constructBeds(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    tol::Float64,
    fd::Function,
)::Matrix{Int64}
    xc = (VX[EToV] + VX[EToV[:, [2, 3, 1]]]) ./ 2
    yc = (VY[EToV] + VY[EToV[:, [2, 3, 1]]]) ./ 2
    return getindex.(findall(abs.(fd.(xc, yc)) .<= tol), [1 2])
end

function constructBnodes(
    VX::Vector{Float64},
    VY::Vector{Float64},
    tol::Float64,
    fd::Function,
)::Vector{Int64}
    return findall(abs.(fd.(VX, VY)) .<= tol)
end

function dirbc(
    bnodes::Vector{Int64},
    f::Vector{Float64},
    A::SparseMatrixCSC{Float64, Int64},
    b::Vector{Float64}
)::Tuple{SparseMatrixCSC{Float64, Int64}, Vector{Float64}}
```

```julia
    for (i, k) in enumerate(bnodes)
        b[k] = f[i]

        indices1 = findall(A[1:k-1, k] .!= 0)
        indices2 = findall(A[k, 1+k:end] .!= 0) .+ k

        b[indices1] -= A[indices1,k] .* f[i]
        b[indices2] -= A[k,indices2] .* f[i]

        A[indices1, k] .= 0
        A[k, indices2] .= 0
        A[k, k] = 1
    end

    return A, b
end

function edgeIndices(
    EToV::Matrix{Int64},
    beds::Matrix{Int64},
)::Tuple{Vector{Int64}, Vector{Int64}}
    n = beds[:, 1]
    r = beds[:, 2]
    s = r .% 3 .+ 1

    i = EToV[CartesianIndex.(n, r)]
    j = EToV[CartesianIndex.(n, s)]

    return i,j
end

function neubc(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    beds::Matrix{Int64},
    q::Vector{Float64},
    b::Vector{Float64}
)::Vector{Float64}
    i, j = edgeIndices(EToV, beds)
    q1 = q .* sqrt.((VX[j] - VX[i]).^2 + (VY[j] - VY[i]).^2) ./ 2

    b[i] -= q1
    b[j] -= q1

    return b
end
```

```
function assembly(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Vector{Float64}
)::Tuple{SparseMatrixCSC{Float64, Int64}, Vector{Float64}}
    N = size(EToV)[1]
    M = length(VX)

    A = spzeros(M, M)
    B = zeros(M)

    as, bs, cs = basfun(VX, VY, EToV)
    deltas = sum(as, dims=2) ./ 2
    qs = abs.(deltas) .* sum(qt[EToV], dims=2) / 9

    for n in 1:N
        delta = deltas[n]
        q = qs[n]
        b = bs[n, :]
        c = cs[n, :]

        for r in 1:3
            i = EToV[n,r]
            B[i] += q

            for s in r:3
                j = EToV[n,s]
                kn = (lam1*b[r]*b[s] + lam2*c[r]*c[s]) / (4 * abs(delta))
                A[min(i, j), max(i, j)] += kn
            end
        end
    end

    return A, B
end

function assembly2(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Vector{Float64}
```

```
)::Tuple{SparseMatrixCSC{Float64, Int64}, Vector{Float64}}
    N = size(EToV)[1]
    M = length(VX)
    A = spzeros(M, M)
    b = zeros(M)

    as, bs, cs = basfun(VX, VY, EToV)
    deltas = sum(as, dims=2) ./ 2
    qs = abs.(deltas) .* sum(qt[EToV], dims=2) / 9

    for r in 1:3
        i = EToV[:,r]

        for n in 1:N
            b[i[n]] += qs[n]
        end

        for s in r:3
            j = EToV[:,s]
            ks = (lam1 .* bs[:,r] .* bs[:,s] + lam2 .* cs[:,r] .* cs[:,s]) ./
 ↪(4 .* abs.(deltas))
            idx = CartesianIndex.(min.(i, j), max.(i, j))

            for n in 1:N
                A[idx[n]] += ks[n]
            end
        end
    end

    return A, b
end

function assembly3(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Vector{Float64}
)::Tuple{SparseMatrixCSC{Float64, Int64}, Vector{Float64}}
    N = size(EToV)[1]
    M = length(VX)
    A = spzeros(M, M)
    b = zeros(M)

    as, bs, cs = basfun(VX, VY, EToV)
    deltas = sum(as, dims=2) ./ 2
```

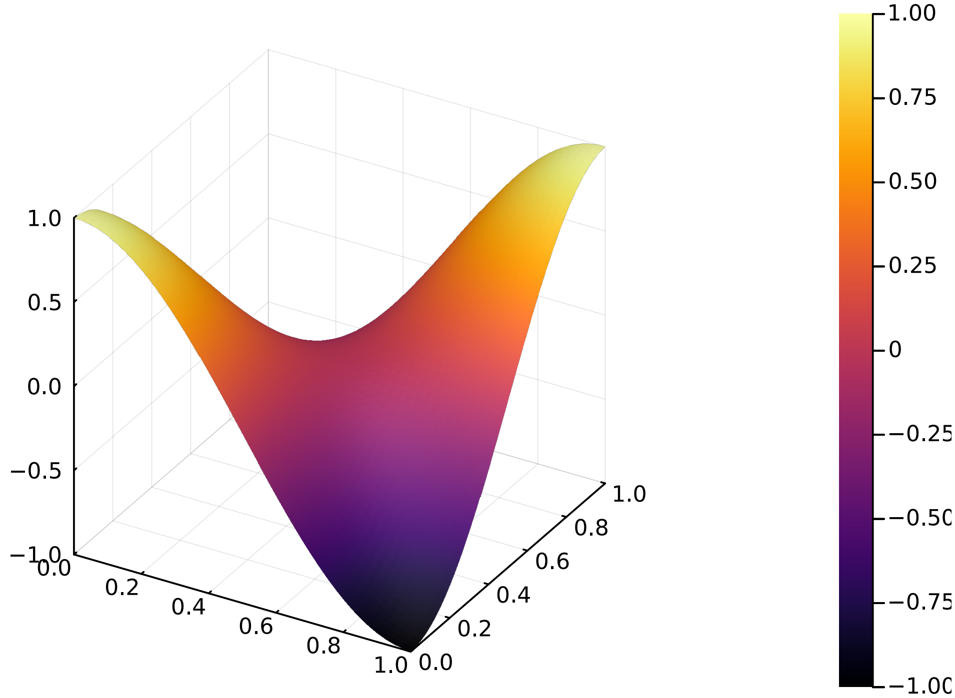
```julia
    qs = abs.(deltas) .* sum(qt[EToV], dims=2) / 9

    r = [1,1,1,2,2,3]
    s = [1,2,3,2,3,3]
    i = EToV[:,r]
    j = EToV[:,s]
    ks = (lam1 .* bs[:,r] .* bs[:,s] .+ lam2 .* cs[:,r] .* cs[:,s]) ./ (4 .*
 ↪abs.(deltas))
    idx = CartesianIndex.(min.(i, j), max.(i, j))

    for n in 1:N
        A[idx[n, :]] += ks[n, :]
        b[EToV[n, :]] .+= qs[n]
    end

    return A, b
end

function solveNDBVP(
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Function,
    q::Function,
    f::Function,
    fd_gamma1::Function,
    fd_gamma2::Function,
    tol::Float64
)::Vector{Float64}
    A, b = assembly(VX, VY, EToV, lam1, lam2, qt.(VX, VY))

    beds = constructBeds(VX, VY, EToV, tol, fd_gamma1)
    i, j = edgeIndices(EToV, beds)
    b = neubc(VX, VY, EToV, beds, q.(VX[i], VY[i], VX[j], VY[j]), b)

    bnodes = constructBnodes(VX, VY, tol, fd_gamma2)
    A, b = dirbc(bnodes, f.(VX[bnodes], VY[bnodes]), A, b)

    A = Symmetric(A)
    uhat = A \ b

    return uhat
end

function solveDBVP(
```

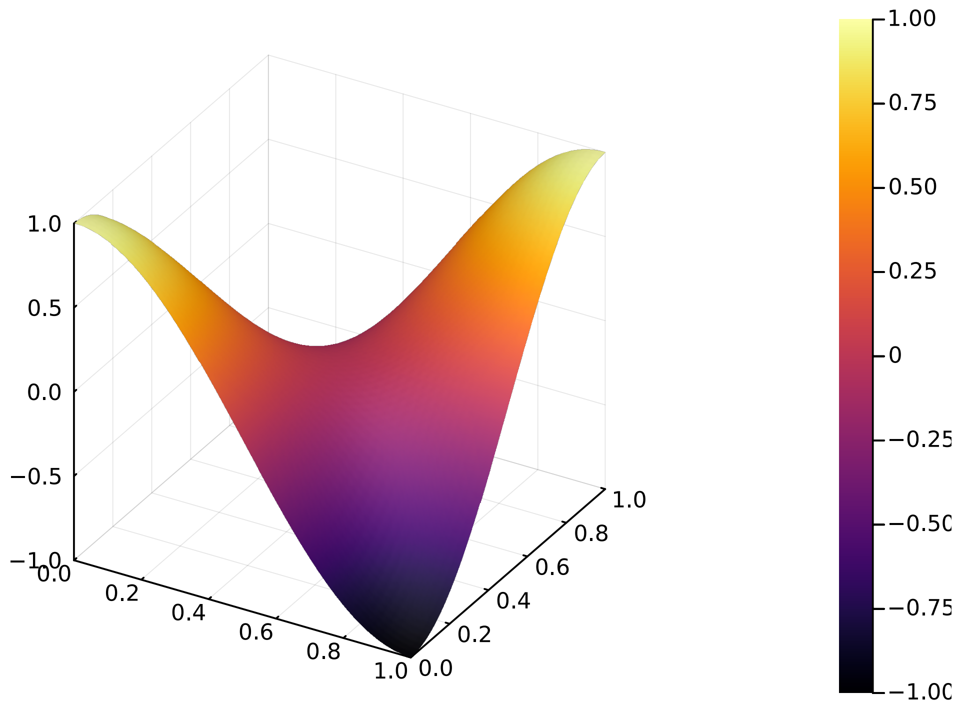
```julia
    VX::Vector{Float64},
    VY::Vector{Float64},
    EToV::Matrix{Int64},
    lam1::Float64,
    lam2::Float64,
    qt::Function,
    f::Function,
    fd_gamma::Function,
    tol::Float64
)::Vector{Float64}
    A, b = assembly(VX, VY, EToV, lam1, lam2, qt.(VX, VY))

    bnodes = constructBnodes(VX, VY, tol, fd_gamma)
    A, b = dirbc(bnodes, f.(VX[bnodes], VY[bnodes]), A, b)

    A = Symmetric(A)
    uhat = A \ b

    return uhat
end

function Driver28b(
    x0::Float64,
    y0::Float64,
    L1::Float64,
    L2::Float64,
    noelms1::Int64,
    noelms2::Int64,
    lam1::Float64,
    lam2::Float64,
    f::Function,
    qt::Function
)::Tuple{Vector{Float64}, Vector{Float64}, Matrix{Int64}, Vector{Float64}}
    fd_gamma(x, y) = min(x0 + L1 - x, y0 + L2 - y)
    tol = 0.0001

    VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
    EToV = conelmtab(noelms1, noelms2)
    uhat = solveDBVP(VX, VY, EToV, lam1, lam2, qt, f, fd_gamma, tol)

    return VX, VY, EToV, uhat
end

function Driver28c(
    x0::Float64,
    y0::Float64,
    L1::Float64,
```

```
    L2::Float64,
    noelms1::Int64,
    noelms2::Int64,
    lam1::Float64,
    lam2::Float64,
    f::Function,
    qt::Function
)::Tuple{Vector{Float64}, Vector{Float64}, Matrix{Int64}, Vector{Float64}}
    fd_gamma(x, y) = min(x - x0, y - y0, x0 + L1 - x, y0 + L2 - y)
    tol = 0.0001

    VX, VY = xy(x0, y0, L1, L2, noelms1, noelms2)
    EToV = conelmtab(noelms1, noelms2)
    uhat = solveDBVP(VX, VY, EToV, lam1, lam2, qt, f, fd_gamma, tol)

    return VX, VY, EToV, uhat
end
```

[17]: Driver28c (generic function with 1 method)

[18]:
```
x0 = 0.0
y0 = 0.0
L1 = 1.0
L2 = 1.0
noelms1 = 40
noelms2 = 50
lam1 = 1.0
lam2 = 1.0
f(x, y) = cos(  * x) * cos(  * y)
qt(x, y) = 2 *  ^2 * cos(  * x) * cos(  * y)

VX, VY, EToV, uhat = Driver28b(x0, y0, L1, L2, noelms1, noelms2, lam1, lam2, f,␣
 ↪qt)

u(x, y) = cos(pi*x)*cos(pi*y)

display(scatter(VX, VY, uhat, st=:surface))
display(scatter(VX, VY, u.(VX, VY), st=:surface))
maximum(abs.(uhat - u.(VX, VY)))
```

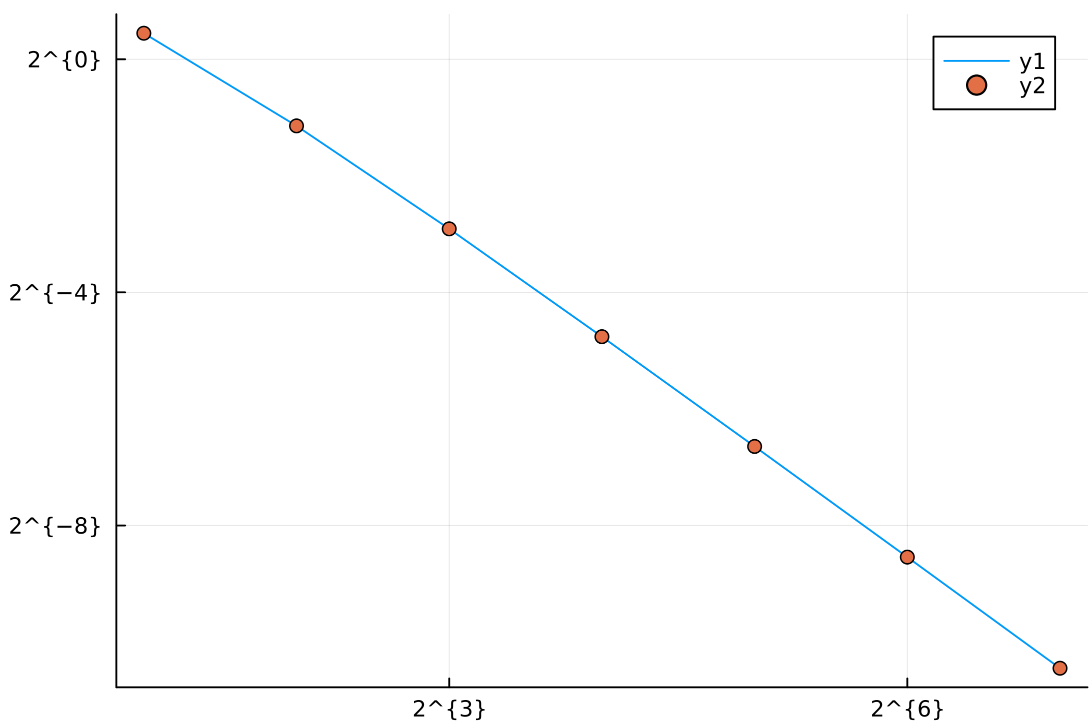

1.00
0.75
0.50
0.25
0
−0.25
−0.50
−0.75
−1.00
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


1.00
0.75
0.50
0.25
0
−0.25
−0.50
−0.75
−1.00
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

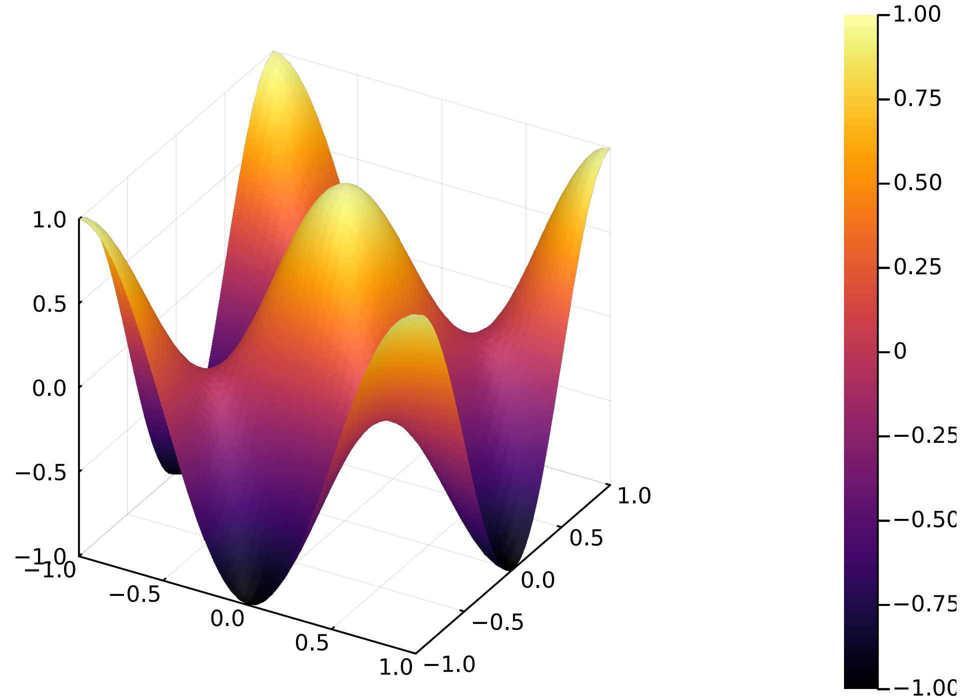
[18]: 0.005355378345388018

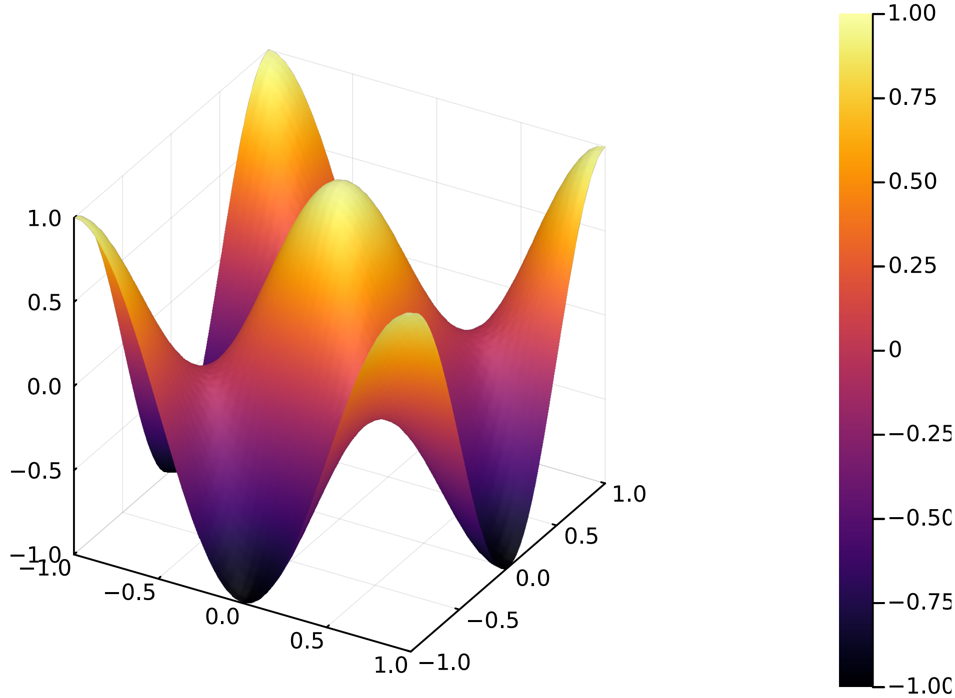
[19]:
```
N = 7
dofs = zeros(N)
max_errors = zeros(N)

for p in 1:N
    noelms = 2^p

    VX, VY, EToV, uhat = Driver28b(x0, y0, L1, L2, noelms, noelms, lam1, lam2,
 ↪f, qt)

    dofs[p] = noelms
    max_errors[p] = maximum(abs.(uhat - u.(VX, VY)))
end

display(fit(log.(dofs[end-3:end]), log.(max_errors[end-3:end]), 1))
p = plot(dofs, max_errors, xscale=:log2, yscale=:log2)
scatter!(dofs, max_errors)
display(p)
```

$1.963173729059884 - 1.8966032046811219 \cdot x$



[20]:
```julia
x0 = -1.0
y0 = -1.0
L1 = 2.0
L2 = 2.0
noelms1 = 40
noelms2 = 50
lam1 = 1.0
lam2 = 1.0
f(x, y) = cos(  * x) * cos(  * y)
qt(x, y) = 2 *  ^2 * cos(  * x) * cos(  * y)

VX, VY, EToV, uhat = Driver28c(x0, y0, L1, L2, noelms1, noelms2, lam1, lam2, f, qt)

u(x, y) = cos(pi*x)*cos(pi*y)

display(scatter(VX, VY, uhat, st=:surface))
display(scatter(VX, VY, u.(VX, VY), st=:surface))
maximum(abs.(uhat - u.(VX, VY)))
```

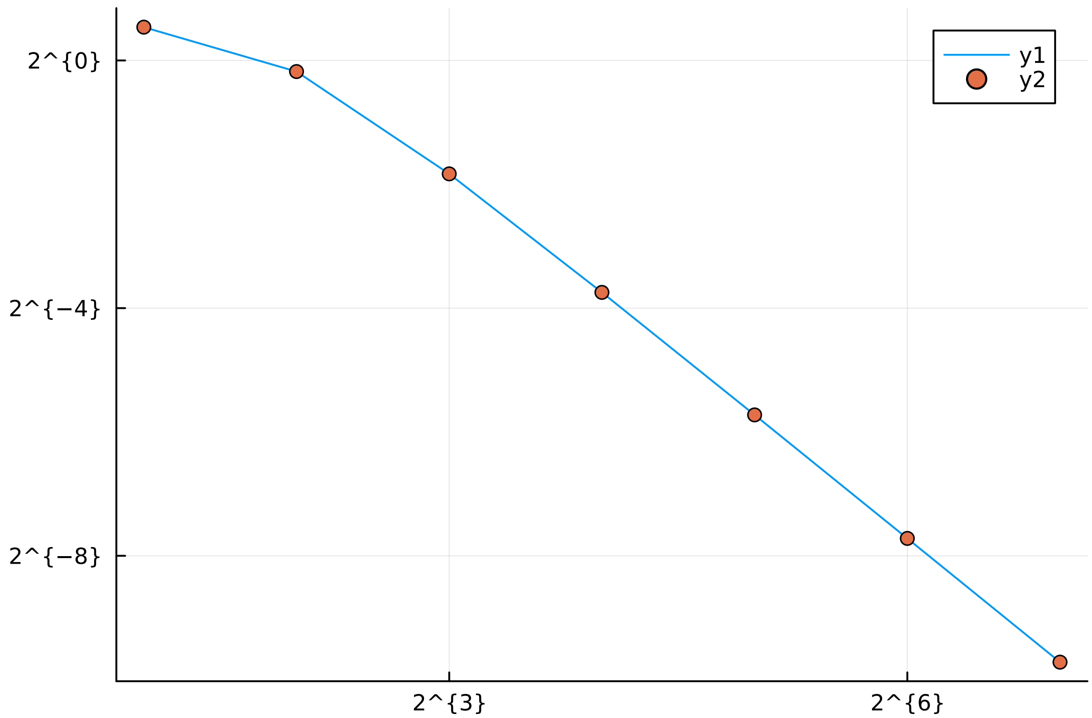




[20]: 0.00994984676417987

[21]:
```
N = 7
dofs = zeros(N)
max_errors = zeros(N)

for p in 1:N
    noelms = 2^p

    VX, VY, EToV, uhat = Driver28c(x0, y0, L1, L2, noelms, noelms, lam1, lam2,␣
 ↪f, qt)

    dofs[p] = noelms
    max_errors[p] = maximum(abs.(uhat - u.(VX, VY)))
end

display(fit(log.(dofs[end-3:end]), log.(max_errors[end-3:end]), 1))
p = plot(dofs, max_errors, xscale=:log2, yscale=:log2)
scatter!(dofs, max_errors)
display(p)
```

$2.927864540150888 - 1.9910496649218166 \cdot x$


2^{0}
2^{-4}
2^{-8}
2^{3}
2^{6}
y1
y2