MATH 521 - Numerical Analysis of Differential Equations

Christoph Ortner, 03/2024

Assignment 4 : Implementation Practice

Name: Jincong Li

Student ID: 60539939

Notes

This assignment is concerned with implementation and empirical convergence studies. There will be a little bit of theory but it purely formal - no proofs are required. I will not give too detailed instructions or have model codes ready to be completed. But you are of course allowed (in fact encouraged) to use as much of the codes from class as you wish. In fact the assignment requires again relatively minimal adaptation of my model codes.

You may add as many solutions cells as needed to implement your numerical methods and tests. To help with Q2 I advise that you implement it in a structured way based on function calls rather than a single script, so you can then re-use those functions for the convergence study. For the study of singularities, you should be able to import and use the codes from class directly, you should not need to adapt them at all.

In principle you may of course write your own code from scratch using a different (but similarly low-level) finite element package. Unless you already have experience with such a package I don't recommend this since it would be a fairly time-consuming undertaking. If you plan to do this please check with me so I can be sure the code you are using is suitable.

Q1: Inhomogeneous Dirichlet Problem [15]

We consider the boundary value problem

$$\begin{aligned} -\mathrm{div} A(x) \nabla u &= f, \quad \Omega \\ u &= u_D(x), \quad \partial\Omega, \end{aligned}$$

where $\Omega = (0,1)^2$, $f(x) = xy$, $u_D(x) = \sin(\pi(x_1 + x_2))$, and

$$A(x) = \begin{pmatrix} 1 & \frac{1}{2}\sin(\pi x_1) \\ \frac{1}{2}\sin(\pi x_1) & 1 \end{pmatrix}$$

Adapt the code from class to implement a P_k-Finite Element method. Visualize the solution.

In [ ]:
```
using Pkg; Pkg.activate(".")
using LinearAlgebra, Ferrite, LaTeXStrings, FerriteViz, WGLMakie, Makie
Makie.inline!(true);
```

```
 Activating project at `e:\UBC\MEng\Term2\MATH 521\HW4\A4`
```

In [ ]:
```
function setup_fem(N, p)
    grid = generate_grid(Triangle, (N, N),Ferrite.Vec(0.0, 0.0),Ferrite.Vec(1.0, 1.0))
    dim = 2
    ip = Lagrange{dim, RefTetrahedron, p}()
    ip_geo = Lagrange{dim, RefTetrahedron, 1}()
    qr = QuadratureRule{dim, RefTetrahedron}(3*p)
    cellvalues = CellScalarValues(qr, ip, ip_geo)
    dh = DofHandler(grid)
    add!(dh, :u, 1, ip)
    close!(dh)
    K = create_sparsity_pattern(dh)
    ch = ConstraintHandler(dh)
    ∂Ω = union(getfaceset.(Ref(grid), ["left", "right", "top", "bottom"])...)
    dbc = Dirichlet(:u, ∂Ω, (x,t) -> sin(π * (x[1] + x[2])))
    add!(ch, dbc)
    close!(ch)
    return cellvalues, dh, ch, K
end
```

```
setup_fem (generic function with 2 methods)
```

In [ ]:
```
function solve_fem(cellvalues, dh, ch, K, ffun)
    K, f = assemble_global!(cellvalues, dh, K, ξ -> ffun(ξ));
    apply!(K, f, ch)
    u = K \ f;
    return u
end
```

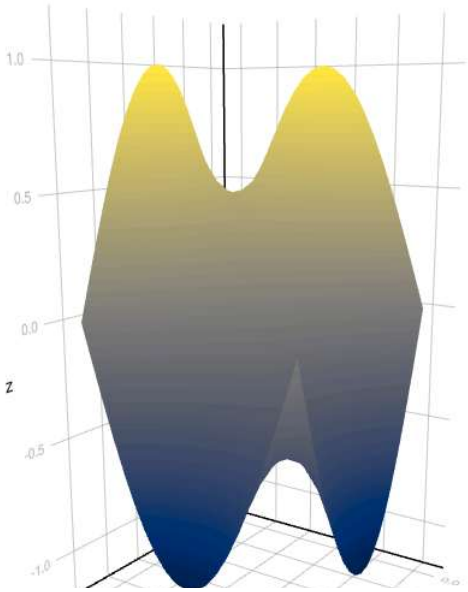
```
solve_fem (generic function with 1 method)
```

In [ ]:
```
function assemble_global!(cellvalues, dh, K, ffun)
    # we pre-allocate the element stiffness matrix and element force vector
    # these will be passed to the element assembly to avoid many allocations
    n_basefuncs = getnbasefunctions(cellvalues)
    Ke = zeros(n_basefuncs, n_basefuncs)
    fe = zeros(n_basefuncs)

    # Allocate global force vector f
    f = zeros(ndofs(dh))

    # Create an assembler: this object knows how to write
    # the local arrays Ke, fe into the global arrays K, f
    assembler = start_assemble(K, f)

    # Loop over all cells; this is managed by the DOF handler
    # since the `cell` comes with information about local
    # DOFs attached.
    for cell in CellIterator(dh)
        # Reinitialize cellvalues for this cell
        # `cellvalues` has iterators attached that need
        # to be reset. This seems unnecessary and probably
        # just a poor code design decision.
        reinit!(cellvalues, cell)
        # ========================================
        # Compute element contribution;
        # this is where the actual work happens
        assemble_element!(Ke, fe, cell, cellvalues, ffun)
        # ========================================
        # local-to-global assemble Ke and fe into K and f
        assemble!(assembler, celldofs(cell), Ke, fe)
    end
    return K, f
end

function assemble_element!(Ke, fe, cell, cellvalues, ffun)
    # number of local basis functions
    n_basefuncs = getnbasefunctions(cellvalues)
    # Reset the local arrays
    fill!(Ke, 0); fill!(fe, 0)
    # precompute the cell coordinates
    cell_coords = getcoordinates(cell)

    # Loop over quadrature points
    for i_q in 1:getnquadpoints(cellvalues)
        # Get the quadrature weight for the current quad point
        # this includes the det(F) terms. It can be thought of
        # as the volume element (hence dΩ)
        dΩ = getdetJdV(cellvalues, i_q)

        # evaluate f at the quadrature point
        ξ_q = spatial_coordinate(cellvalues, i_q, cell_coords)
        f_q = ffun(ξ_q)

        # Loop over test shape functions (basis functions)
        for i in 1:n_basefuncs
            # get the values v = ψ_i(ξ_q), ∇v = ∇ψ_i(ξ_q)
            v  = shape_value(cellvalues, i_q, i)
            ∇v = shape_gradient(cellvalues, i_q, i)

            # ∫_K f v dx
            # Add contribution to fe
            fe[i] += f_q * v * dΩ

            # Loop over trial shape functions
            for j in 1:n_basefuncs
                ∇u = shape_gradient(cellvalues, i_q, j)
                # Add contribution to Ke
                #  ∫_K A \cdot ∇v · ∇u
                Ke[i, j] += dot(∇v, A(ξ_q), ∇u) * dΩ
            end
        end
```

```
    end
    return Ke, fe
end

function A(x)
    return [1 0.5*sin(π*x[1]);
            0.5*sin(π*x[1]) 1]
end
```

```
A (generic function with 1 method)
```

In [ ]:
```
cellvalues, dh, ch, K = setup_fem(20, 3)

# solve
u = solve_fem(cellvalues, dh, ch, K, ξ -> ξ[1]*ξ[2])  #10*

# visualize : we can't see a difference in the solution
# but this is good of course. We will look at errors next.
plotter = FerriteViz.MakiePlotter(dh, u)
fig = FerriteViz.surface(plotter, field=:u,
                  figure = (resolution = (700,700,),))
```



Q2: Convergence Study [15]

Implement a numerical convergence study of you method from Q1. Using the method of manufactured solutions, demonstrate that - provided your solution is sufficiently smooth - your method achieves the optimal rate of convergence in the **energy norm**,

$$\|u - u_h\|_a := \left(\int_\Omega \nabla(u - u_h)^T A(x) \nabla(u - u_h)\, dx \right)^{1/2}.$$

Provide a similar visualization of the optimal convergence rates (in this norm) as in reference implementation from class for the P_k-FEM, $k = 1, 2, 3$.

In [ ]:
```
using ForwardDiff

# generate a smooth solution consistent with the boundary condition

u_ex = x -> sin(π * (x[1]+ x[2]))# choose a model "manufactured solution"
∇u_ex = x -> ForwardDiff.gradient(u_ex, x)
ffun = x ->  - tr( ForwardDiff.jacobian(x -> A(x) * ∇u_ex(x), x) )
```

```
#212 (generic function with 1 method)
```

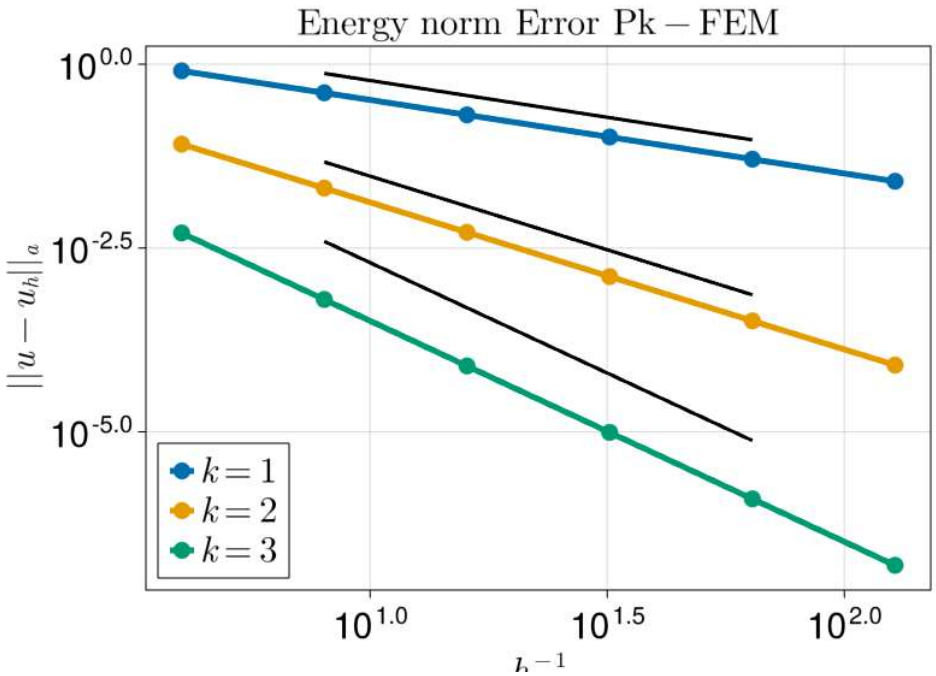
In [ ]:
```
function fem_errors(N, k, ffun)
    cellvalues, dh, ch, K = setup_fem(N, k)
    u = solve_fem(cellvalues, dh, ch, K,ffun)
    return compute_errors(cellvalues, dh, u, u_ex)
end
```

```
function compute_errors(cellvalues, dh, u, u_ex)
    n_basefuncs = getnbasefunctions(cellvalues)
    err_energy = 0.0

    # loop over cells (= elements)
    for cell in CellIterator(dh)
        reinit!(cellvalues, cell)
        n_basefuncs = getnbasefunctions(cellvalues)
        cell_coords = getcoordinates(cell)

        # we also need the local degrees of freedom
        u_cell = u[cell.dofs]

        vK = 0.0
        for i_q in 1:getnquadpoints(cellvalues)
            dΩ = getdetJdV(cellvalues, i_q)
            ξ_q = spatial_coordinate(cellvalues, i_q, cell_coords)

            u_q   = u_ex(ξ_q)
            ∇u_q  = ForwardDiff.gradient(u_ex, ξ_q)
            uh_q  = function_value(cellvalues, i_q, u_cell)
            ∇uh_q = function_gradient(cellvalues, i_q, u_cell)

            err_energy += dΩ *norm((∇u_q - ∇uh_q)' * A(ξ_q) * (∇u_q - ∇uh_q))
        end
    end
    return sqrt(err_energy)
end
```

```
compute_errors (generic function with 1 method)
```

In [ ]:
```
KK = 1:3
NN = [ [4, 8, 16, 32, 64, 128],
       [4, 8, 16, 32, 64, 128],
       [4, 8, 16, 32, 64, 128] ]

errs_energy = [ Float64[] for _ = 1:length(KK) ]

for k in KK, N in NN[k]
    err_energy = fem_errors(N, k,  ffun)
    push!(errs_energy[k], err_energy)
end
```

In [ ]:
```
# ... and visualize them
fig = Figure(size = (400, 400); fontsize=30)
ax = Axis(fig[1, 1], xlabel = L"h^{-1}", ylabel = L" || u - u_h ||_a ",
          xscale = log10, yscale = log10,
          title = L"\text{Energy norm Error Pk-FEM}")
for k in KK
    scatterlines!(NN[k], errs_energy[k]; linewidth=5, markersize=20,
                  label = latexstring("k = $k"))
    NN1 = (k < 3) ? NN[k][2:5] : NN[k][2:5]
    lines!(NN1, 6/k ./ NN1.^(k); color=:black, linewidth=3)

end
axislegend(ax, position = :lb)
fig
```



The next two questions concern semi-empirical and theoretical convergence studies on domains with corners. No rigorous proofs need to be given, but please do justify your arguments. You can re-use the code from the class on corner-singularities almost verbatim, very few changes need to be made.

In [ ]:
```
include("ferrite_tools.jl")
include("aitken.jl")

function compute_energy(k, N, generate_dom, ffun)
    grid, ∂Ω = generate_dom(N)
    cellvalues, dh, ch, K = setup_fem(k, grid, ∂Ω)
    K, f = assemble_global!(cellvalues, dh, K, ξ -> ffun(ξ));
    apply!(K, f, ch)
    u = K \ f;
    return 0.5 * dot(u, K * u) - dot(u, f)
end
```

```
compute_energy (generic function with 1 method)
```

Q3: Regularity in Polygonal Domains - Part 1 [10]

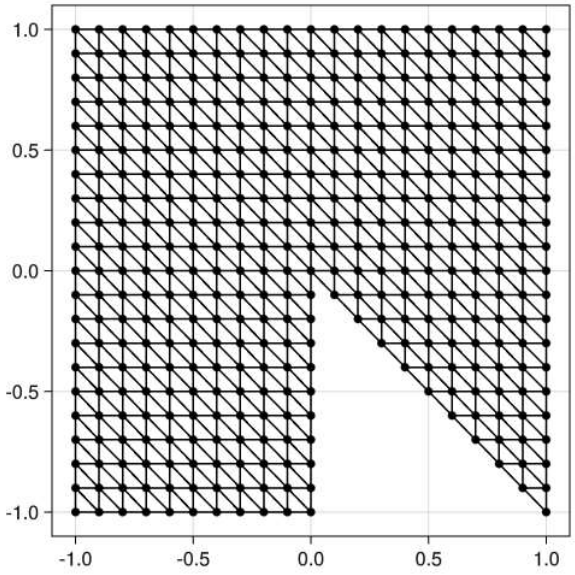
We consider the boundary value problem

$$\begin{aligned} -\Delta u &= 10, \quad x \in \Omega \\ u &= 0, \quad x \in \partial\Omega \end{aligned}$$

where Ω is a domain with a triangle cut out with re-entrant corner of angle $\pi/4$.

What is the expected rate of convergence in energy-norm for P1-FEM and P2-FEM? Briefly justify your answer, then demonstrate this numerically.

In [ ]:
```
grid, ∂Ω = generate_wedge(20)

FerriteViz.wireframe(grid, markersize = 10, strokewidth = 2,
                     figure = (resolution=(500,500),))
```



Solution: theory

Consider the singular harmonic functions:

$$s_\alpha = r^\alpha \sin(\alpha\theta),$$

In this case, the re-entrant corner has an interior angle $\theta_0 = 7\pi/4 = \pi/\alpha$.

Following the same procedure as dicussed in class:

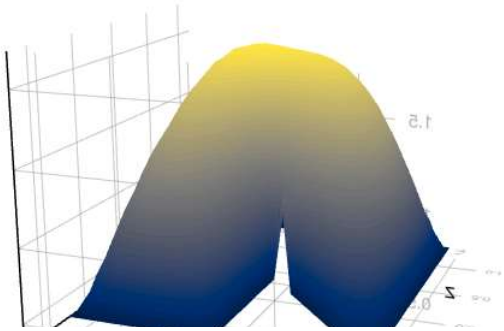
$$||\nabla u - \nabla u_h||_{L^2} \leq C||\nabla u - \nabla I_h u||_{L^2}(*)$$

And by separating the FEM domain into two parts: the first layer around the re-entrant corner and the rest of the domain, one can get:

$$(*) = C(||\nabla u - \nabla I_h u||_{L^2((B\setminus B_\epsilon)\cap\Omega)} + ||\nabla u - \nabla I_h u||_{L^2(B_\epsilon\cap\Omega)})$$

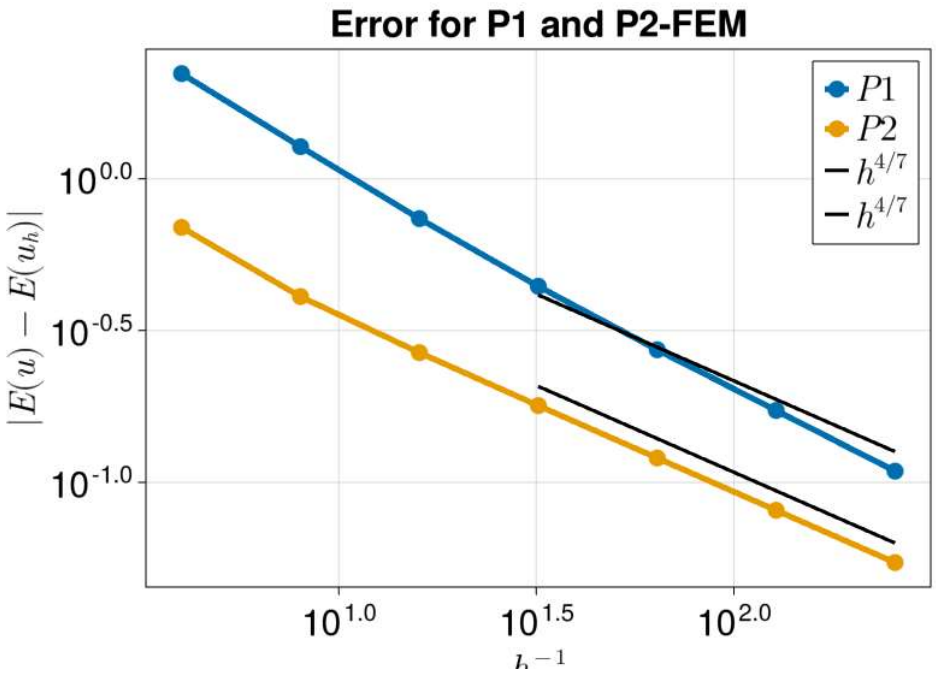
with $\alpha = 4/7$, the rate of convergence in energy-norm is determmined to be $O(h^{4/7})$ for both P1 and P2-fem. For convergence in energy, one could get $O(h^{8/7})$.

In [ ]:
```
# solve the dirichlet problem
ffun = ξ -> 10
k = 1
cellvalues, dh, ch, K = setup_fem(k, grid, ∂Ω)
u = solve_fem(cellvalues, dh, ch, K, ffun);
# visualize the solution
plotter = FerriteViz.MakiePlotter(dh, u)
fig = FerriteViz.surface(plotter, field=:u,
                   figure = (resolution = (700,700,),))
```



In [ ]:
```
ffun = ξ -> 10
NN = [4, 8, 16, 32, 64, 128, 256]
E = [ [ compute_energy(k, N, generate_wedge, ffun) for N in NN ]
      for k = 1:2 ]
Elim = aitken.(E)
err = [ sqrt.(abs.(E[i] .- Elim[i])) for i = 1:2 ]
;
```

In [ ]:
```
# show the convergence of total energy
fig = Figure(size = (400, 400); fontsize=30)
ax = Axis(fig[1, 1], xlabel = L"h^{-1}", ylabel = L"|E(u) - E(u_h)|",
          xscale = log10, yscale = log10,
          title = "Error for P1 and P2-FEM")
NN1 = NN[4:7]
scatterlines!(NN, err[1]; linewidth=5, markersize=20, label=L"P1")
scatterlines!(NN, err[2]; linewidth=5, markersize=20, label=L"P2")
lines!(NN1, 3 ./ NN1.^(4/7); color=:black, linewidth=3, label=L"h^{4/7}")
lines!(NN1, 1.5 ./ NN1.^(4/7); color=:black, linewidth=3, label=L"h^{4/7}")

axislegend(ax)
fig
```



Thus, one could conclude the numerical result agrees with the theoretical result for P2 FEM, but slightly off for the P1 FEM. This might cause by the effect of mesh size h in P1 FEM, the slope should be equal when h is really small.

Q4: Regularity in Polygonal Domains - Part 2 [10]

We consider the boundary value problem

$$\begin{aligned} -\Delta u &= 5, \quad x \in \Omega \\ u &= 0, \quad x \in \partial\Omega \end{aligned}$$

where Ω is now again the unit square, namely a triangle with angles $\pi/4, \pi/4, \pi/2$. (visualized in the first cell below)

What is the expected rate of convergence in energy-norm for P1-FEM, P2-FEM, P3-FEM?

Even though the square should be the simplest of all cases, it is actually not trivial to adapt our arguments for re-entrant corners to this setting, even formally. Try to work it out, but if you can't see how to go about it, then do a purely empirical study and try to explain more intuitively what you observe. I will give close to full points for that.

(Because of the high symmetry, there are alternative techniques to understand the regularity of solutions in a square, e.g. Fourier. You could think about that too. But this takes a lot of time and I don't expect it.)

Solution: theory

The domain is considered to be convex so $u \in H^2$, this means the rate for P1-FEM is h.

Consider all corners in the square domain are right angles： $\theta_0 = \pi/2 = \pi/\alpha$, i.e. $\alpha = 2$. By the same singular harmonic functions: $s_2(x) = r^2 \sin(2\theta)$.

**Because higher r-derivatives just vanish, we should look at the θ-derivatives instead. ** (The rest of the theroy part of the solution is not included in my initial answer, so I just leave them here)

Recall that the derivatives in euclidean coordinates scale roughly as $\nabla_{xy} \sim (\partial_r, r^{-1}\partial_\theta)$. For the third derivative we get

$$r^{-3}\partial_\theta^3 s_2 = -8r^{-1}\cos(2\theta)$$

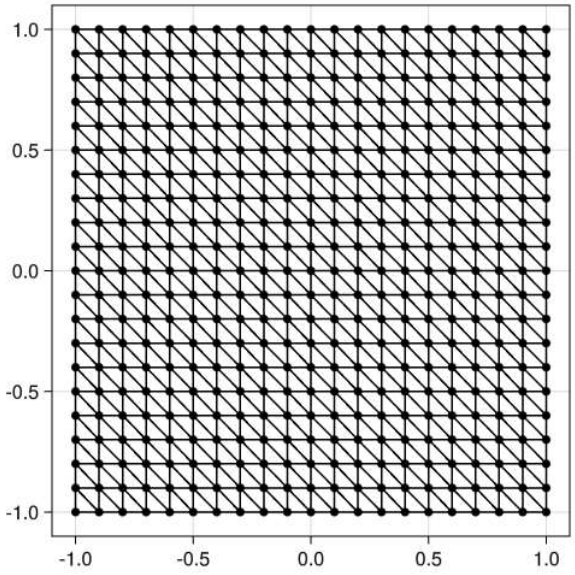
This suggests that $|\nabla^3 s_2|^2 \approx r^{-2}$ near the singularity, which would mean that it is "almost but not quite integrable", i.e. $u \notin H^3$, but "almost".

We can speculate that the rate of convergence for P2 will be h^t for all $t < k$, i.e. almost optimal. This is really hard to see numerically so we will test against h^2. For P3 to get a better rate we would need strictly better than H^3-regularity, which we clearlly cannot have, so the rate for P3 will be the same as for P2, i.e. almost h^2. (or, h^4 for the convergence of the energy)

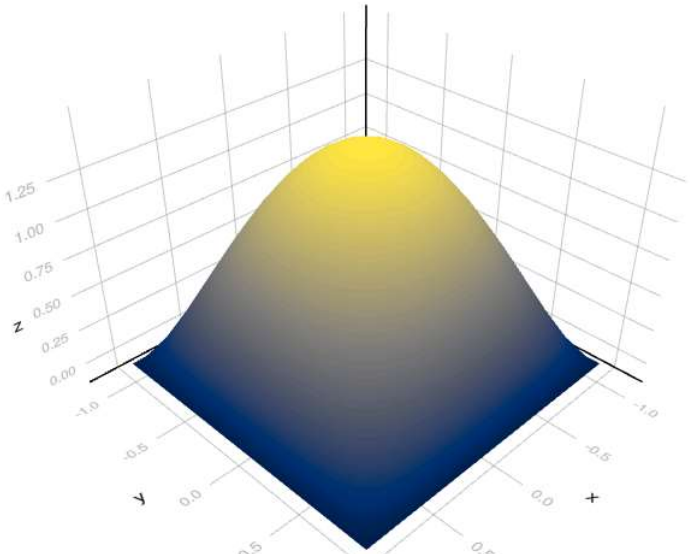
In [ ]:
```
include("ferrite_tools.jl")
include("aitken.jl")

function compute_energy(k, N, generate_dom, ffun)
    grid, ∂Ω = generate_dom(N)
    cellvalues, dh, ch, K = setup_fem(k, grid, ∂Ω)
    K, f = assemble_global!(cellvalues, dh, K, ξ -> ffun(ξ));
    apply!(K, f, ch)
    u = K \ f;
    return 0.5 * dot(u, K * u) - dot(u, f)
end
```

```
compute_energy (generic function with 1 method)
```

In [ ]:
```
grid = generate_grid(Triangle, (20, 20))
∂Ω = union(getfaceset.(Ref(grid), ["left", "right", "top", "bottom"])...)
FerriteViz.wireframe(grid, markersize = 10, strokewidth = 2,
                     figure = (resolution=(500,500),))
```



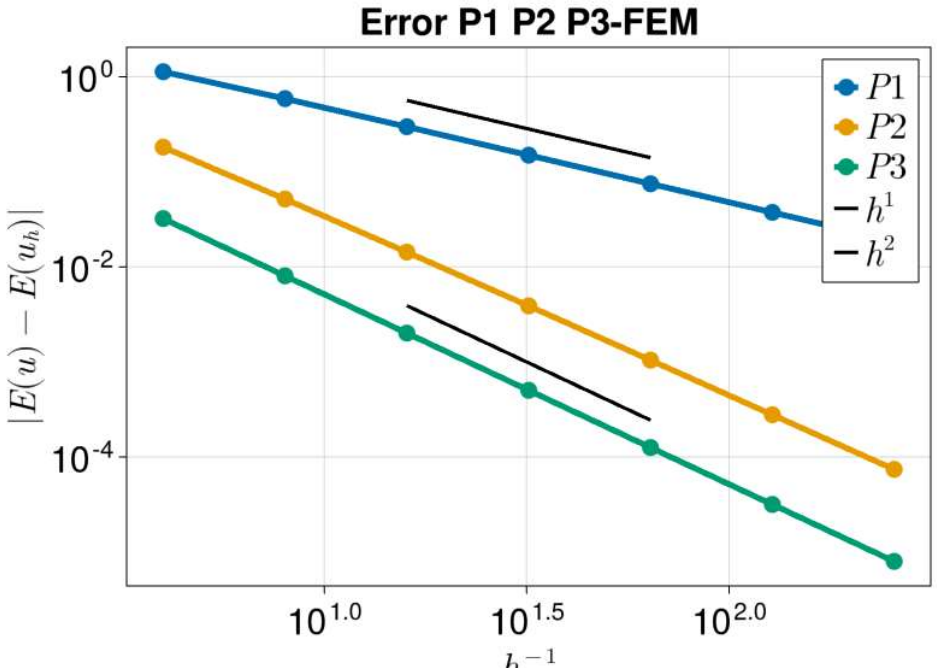
In [ ]:
```
# solve the dirichlet problem
ffun = ξ -> 5
k = 1
cellvalues, dh, ch, K = setup_fem(k, grid, ∂Ω)
u = solve_fem(cellvalues, dh, ch, K, ffun);
# visualize the solution
plotter = FerriteViz.MakiePlotter(dh, u)
fig = FerriteViz.surface(plotter, field=:u,
                  figure = (resolution = (700,700,),))
```



In [ ]:
```
ffun = ξ -> 5
NN = [4, 8, 16, 32, 64, 128, 256]
E = [ [ compute_energy(k, N, generate_square, ffun) for N in NN ]
      for k = 1:3 ]
Elim = aitken.(E)
err = [ sqrt.(abs.(E[i] .- Elim[i])) for i = 1:3 ]
;
```

In [ ]:
```
# show the convergence of total energy
fig = Figure(size = (400, 400); fontsize=30)
ax = Axis(fig[1, 1], xlabel = L"h^{-1}", ylabel = L"|E(u) - E(u_h)|",
          xscale = log10, yscale = log10,
          title = "Error P1 P2 P3-FEM")
```

```
NN1 = NN[3:5]
scatterlines!(NN, err[1]; linewidth=5, markersize=20, label=L"P1")
scatterlines!(NN, err[2]; linewidth=5, markersize=20, label=L"P2")
scatterlines!(NN, err[3]; linewidth=5, markersize=20, label=L"P3")
lines!(NN1, 9 ./ NN1.^(1); color=:black, linewidth=3, label=L"h^1")
lines!(NN1, 1 ./ NN1.^(2); color=:black, linewidth=3, label=L"h^2")

axislegend(ax)
fig
```



Thus, one could conclude the numerical result agrees with the theoretical result for P1 P2 and P3 FEM.