MATH 521 - Numerical Analysis of Differential Equations

Christoph Ortner, 03/2024

Assignment 4 : Implementation Practice

Name:

Student ID:

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 [7]:
```
using Pkg; Pkg.activate(".")
using LinearAlgebra, Ferrite, LaTeXStrings, FerriteViz, WGLMakie, Makie
Makie.inline!(true);
```

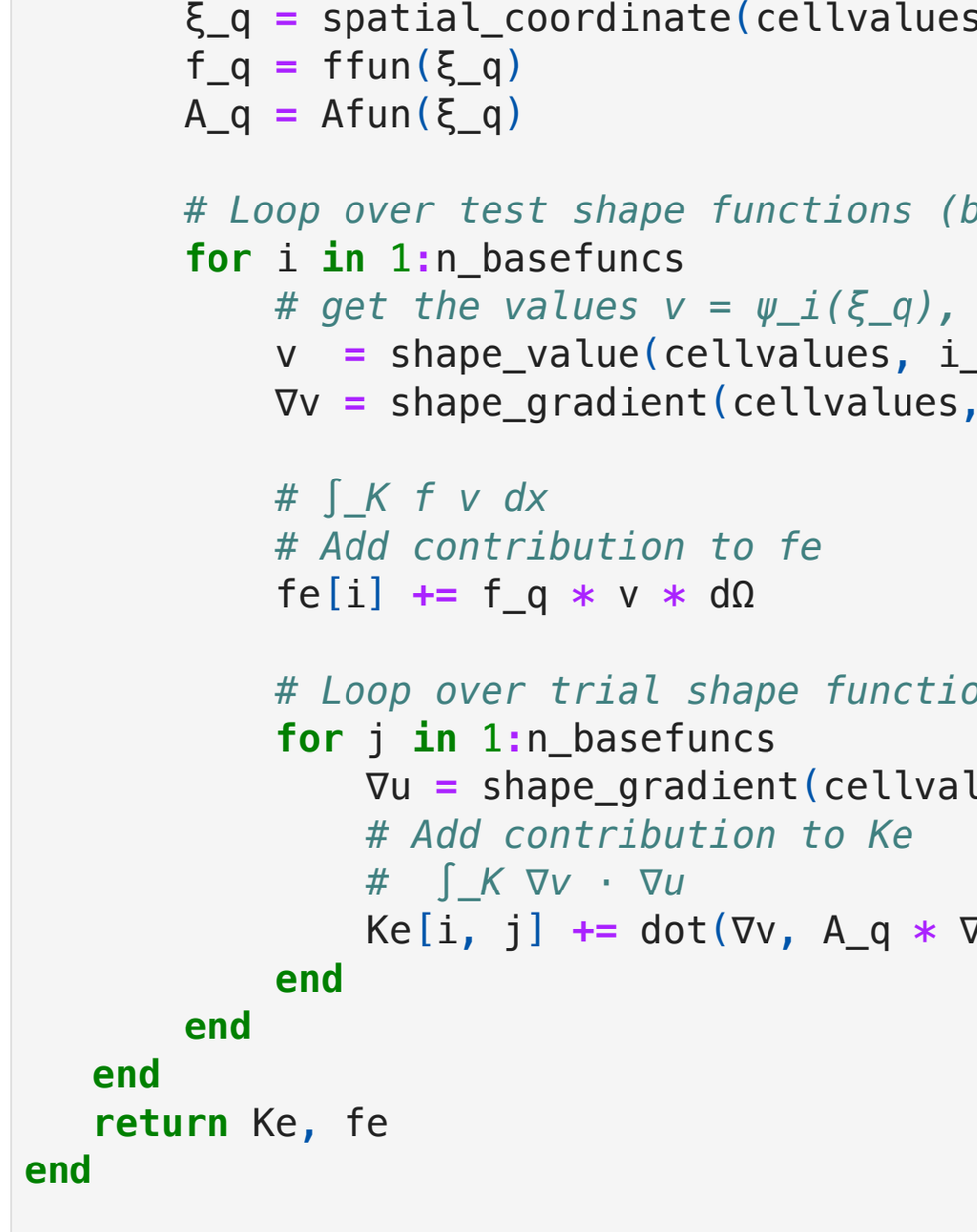
```
  Activating project at `~/Desktop/math521/assignments/A4_solution`
```

In [8]:
```
function assemble_global!(cellvalues, dh, K, ffun, Afun)
   n_basefuncs = getnbasefunctions(cellvalues)
   Ke = zeros(n_basefuncs, n_basefuncs)
   fe = zeros(n_basefuncs)
   f = zeros(ndofs(dh))
   assembler = start_assemble(K, f)
   for cell in CellIterator(dh)
       reinit!(cellvalues, cell)
       assemble_element!(Ke, fe, cell, cellvalues, ffun, Afun)
       assemble!(assembler, celldofs(cell), Ke, fe)
   end
   return K, f
end

function assemble_element!(Ke, fe, cell, cellvalues, ffun, Afun)
   n_basefuncs = getnbasefunctions(cellvalues)
   fill!(Ke, 0); fill!(fe, 0)
   cell_coords = getcoordinates(cell)

   for i_q in 1:getnquadpoints(cellvalues)
       dΩ = getdetJdV(cellvalues, i_q)

       ξ_q = spatial_coordinate(cellvalues, i_q, cell_coords)
       f_q = ffun(ξ_q)
       A_q = Afun(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 ∇v · ∇u
               Ke[i, j] += dot(∇v, A_q * ∇u) * dΩ
           end
       end
   end
   return Ke, fe
end

function setup_fem_square(N, p, uD_fun)
    grid = generate_grid(Triangle, (N, N))
    dim = 2
    ip = Lagrange{dim, RefTetrahedron, p}()
    ip_geo = Lagrange{dim, RefTetrahedron, 1}()
    qr = QuadratureRule{dim, RefTetrahedron}(2*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) -> uD_fun(x))
    add!(ch, dbc)
    close!(ch)
    return cellvalues, dh, ch, K
end


function solve_fem_square(N, p, f_fun, A_fun, uD_fun)
    cellvalues, dh, ch, K = setup_fem_square(N, p, uD_fun)
    K, f = assemble_global!(cellvalues, dh, K, f_fun, A_fun);
    apply!(K, f, ch)
    u = K \ f;
    return u, dh, cellvalues
end
```

Out[8]:
```
solve_fem_square (generic function with 1 method)
```

In [9]:
```
uD_fun = x -> sin(π*(x[1]+x[2]))
f_fun = x -> x[1]*x[2]
A_fun = x -> [1.0                  0.5 * sin(π*x[1]);
              0.5 * sin(π*x[1])  1.0 ]

u, dh, _ = solve_fem_square(20, 3, f_fun, A_fun, uD_fun);
```

In [15]:
```
plotter = FerriteViz.MakiePlotter(dh, u)
fig = FerriteViz.surface(plotter, field=:u,
                figure = (resolution = (700,700,),));
Makie.save("Q1.png", fig)
```



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**,

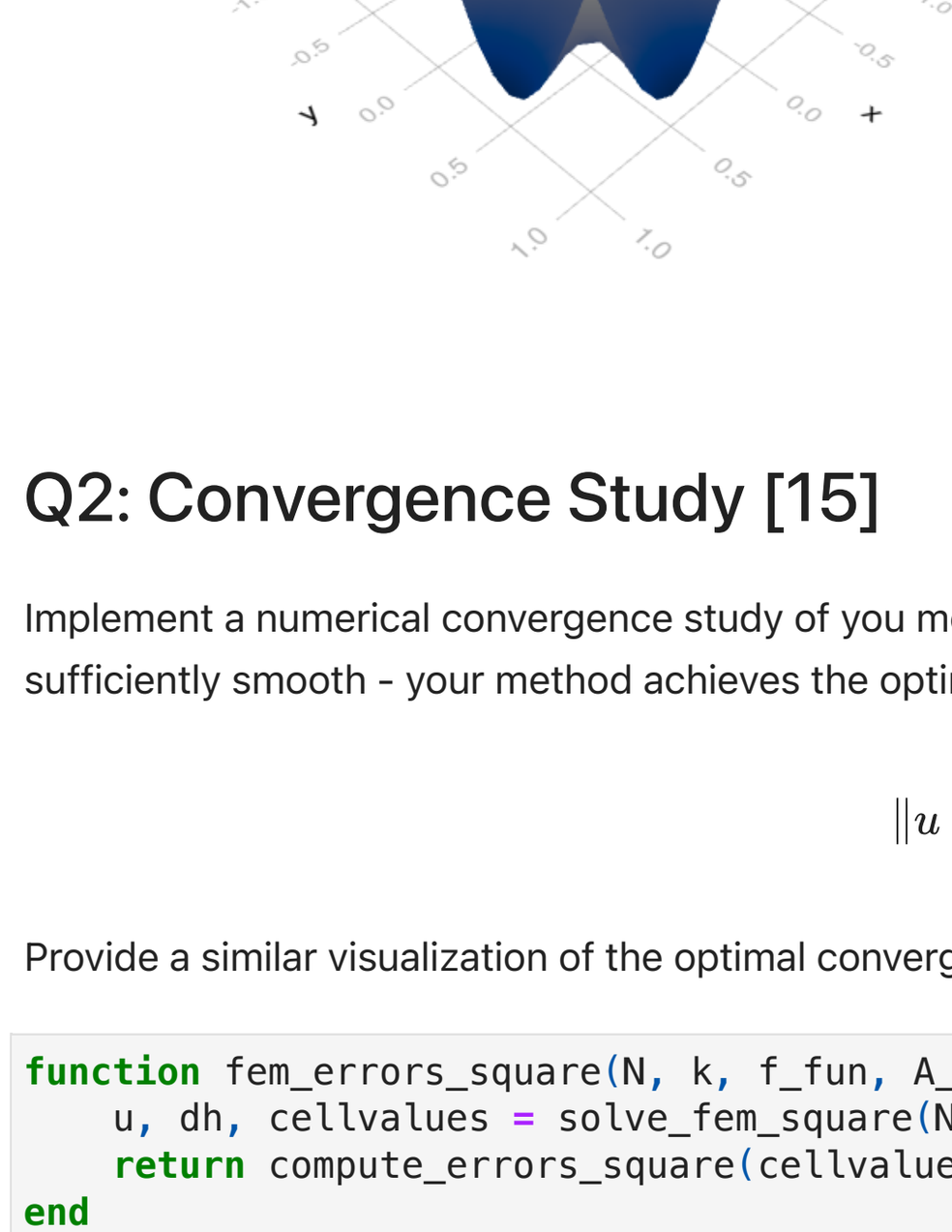
$$\|u - u_h\|_a := \left(\int_\Omega \nabla(u - u_h)^T A(x) \nabla(u - u_h)\, dx \right)^{1/2}.$$

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 [16]:
```
function fem_errors_square(N, k, f_fun, A_fun, uD_fun, u_ex)
    u, dh, cellvalues = solve_fem_square(N, k, f_fun, A_fun, uD_fun)
    return compute_errors_square(cellvalues, dh, u, u_ex, A_fun)
end

function compute_errors_square(cellvalues, dh, u, u_ex, A_fun)
    n_basefuncs = getnbasefunctions(cellvalues)
    err_a = 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)
            A_q = A_fun(ξ_q)
            ∇u_q  = ForwardDiff.gradient(u_ex, ξ_q)
            ∇uh_q = function_gradient(cellvalues, i_q, u_cell)
            err_a += dΩ * dot(∇u_q - ∇uh_q, A_q * (∇u_q - ∇uh_q))
        end
    end
    return sqrt(err_a)
end
```

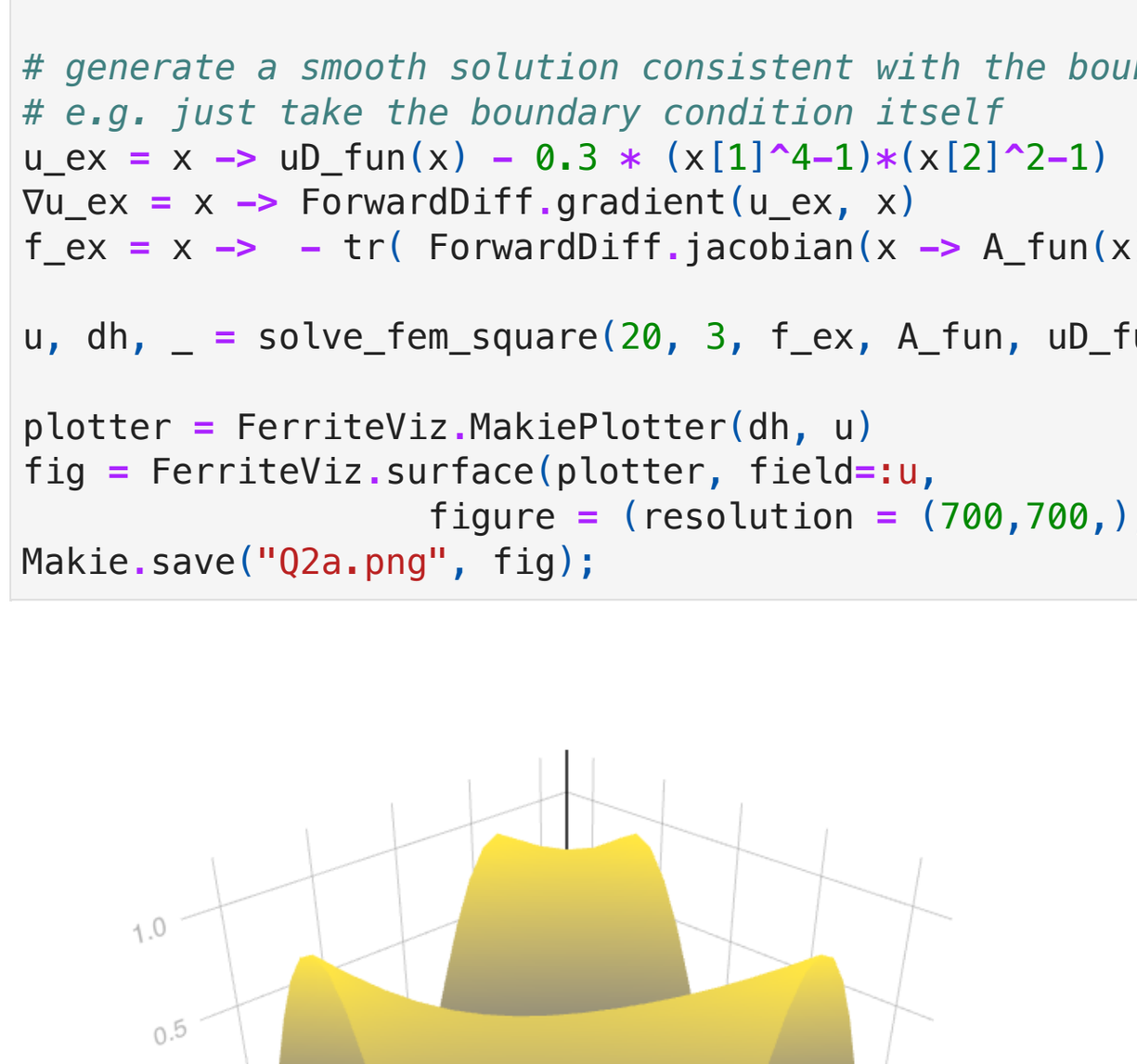
Out[16]:
```
compute_errors_square (generic function with 1 method)
```

In [19]:
```
using ForwardDiff

# generate a smooth solution consistent with the boundary condition
# e.g. just take the boundary condition itself
u_ex = x -> uD_fun(x) - 0.3 * (x[1]^4-1)*(x[2]^2-1)
∇u_ex = x -> ForwardDiff.gradient(u_ex, x)
f_ex = x ->  - tr( ForwardDiff.jacobian(x -> A_fun(x) * ∇u_ex(x), x) )

u, dh, _ = solve_fem_square(20, 3, f_ex, A_fun, uD_fun)

plotter = FerriteViz.MakiePlotter(dh, u)
fig = FerriteViz.surface(plotter, field=:u,
                figure = (resolution = (700,700,),));
Makie.save("Q2a.png", fig);
```



In [20]:
```
NN = [4, 8, 16, 32, 64, 128]
errs = [ Float64[]  for k = 1:3]
for k = 1:3, N in NN
    push!(errs[k], fem_errors_square(N, k, f_ex, A_fun, uD_fun, u_ex))
end
```

In [22]:
```
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{Error Pk-FEM}")
for k in 1:3
    scatterlines!(NN, errs[k]; linewidth=5, markersize=20,
                  label = latexstring("k = $k"))
    NN1 = NN[3:5]
    lines!(NN1, 6/k ./ NN1.^k; color=:black, linewidth=3)
end
axislegend(ax, position = :lb)
Makie.save("Q2b.png", 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 [23]:
```
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
```

Out[23]:
```
compute_energy (generic function with 1 method)
```

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

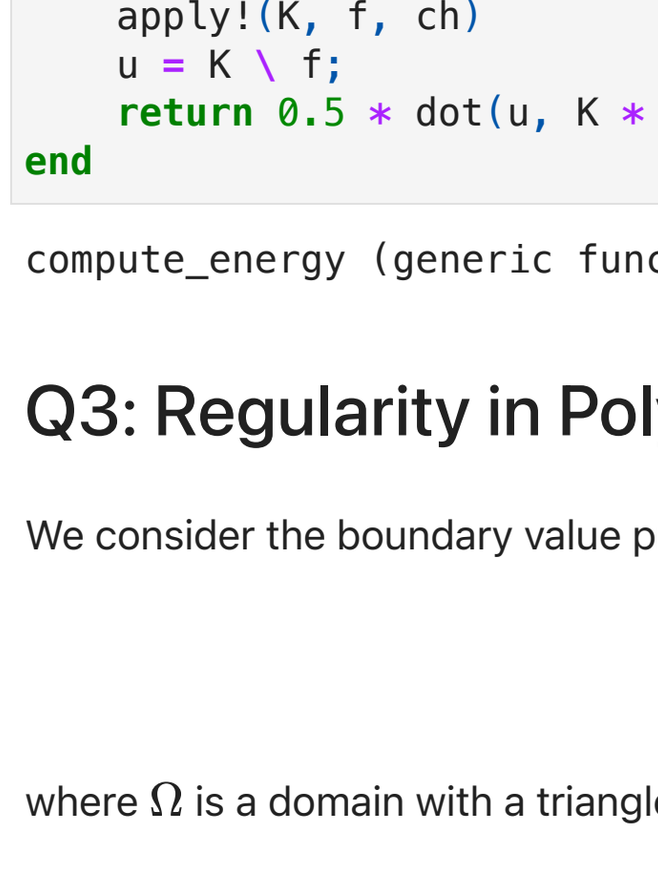
We consider the boundary value problem

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

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 [25]:
```
grid, ∂Ω = generate_wedge(20)

fig = FerriteViz.wireframe(grid, markersize = 10, strokewidth = 2,
                     figure = (resolution=(500,500),))
Makie.save("Q3a.png", fig);
```



Solution: theory

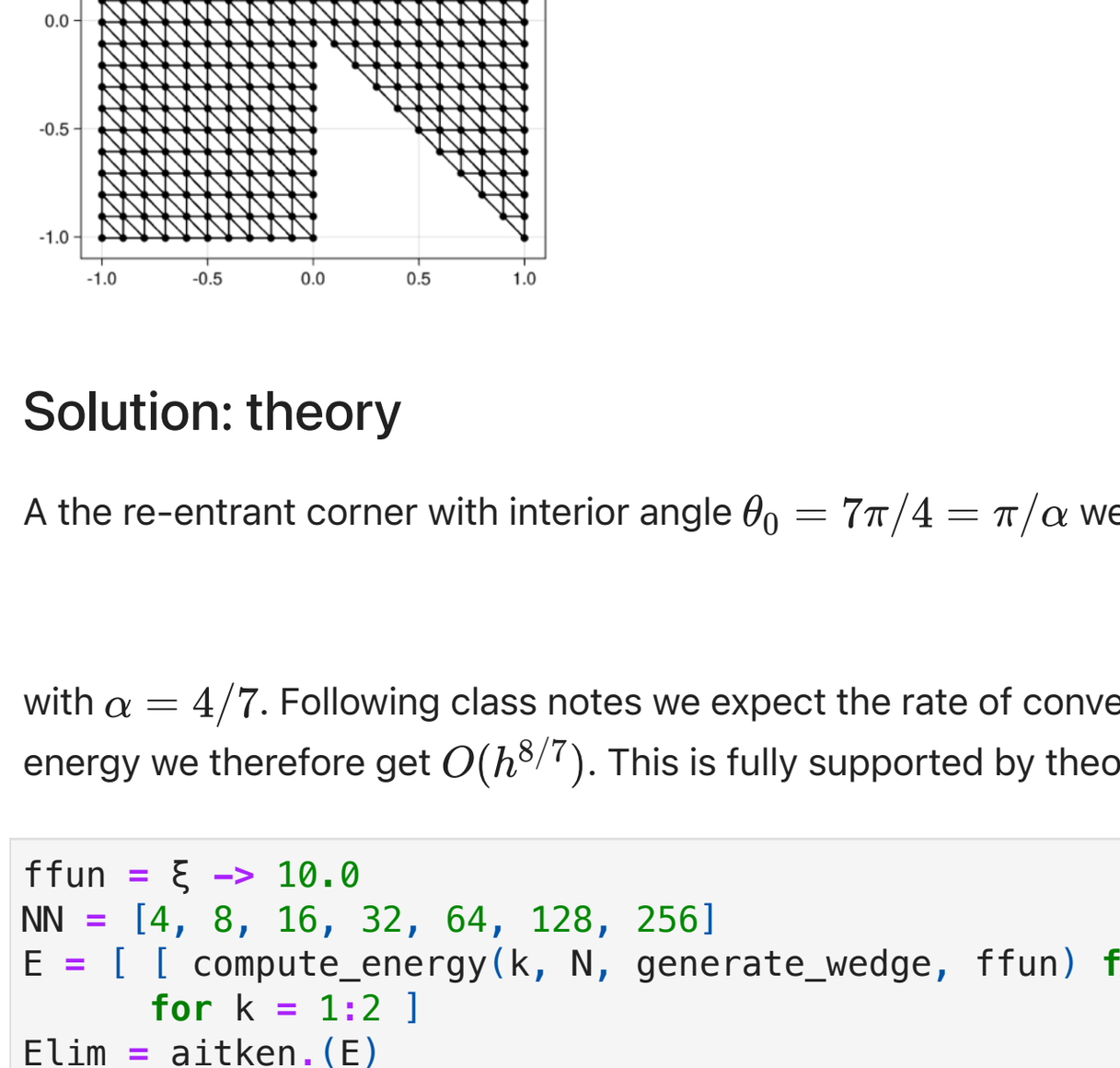
A the re-entrant corner with interior angle $\theta_0 = 7\pi/4 = \pi/\alpha$ we have the leading-order singularity

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

with $\alpha = 4/7$. Following class notes we expect the rate of convergence in energy-norm to be only $O(h^{4/7})$ for both P1 and P2-fem. For convergence in energy we therefore get $O(h^{8/7})$. This is fully supported by theory.

In [27]:
```
ffun = ξ -> 10.0
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 = [ abs.(E[i] .- Elim[i]) for i = 1:2 ];
```

In [28]:
```
# 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-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, 2.5 ./ NN1.^(8/7); color=:black, linewidth=3, label=L"h^{8/7}")

axislegend(ax)
Makie.save("Q3b.png", fig)
```



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

But it is not clear whether $u \in H^3$ or even H^4 to get the optimal rates for P2 and P3? We look at the possible corner singularities: Here the interior angle at all corners is $\theta_0 = \pi/2 = \pi/\alpha$, i.e. $\alpha = 2$. This suggests the leading order corner singularity $s_2(x) = r^2 \sin(2\theta)$.

Because higher r-derivatives just vanish, we look at the θ-derivatives instead. 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

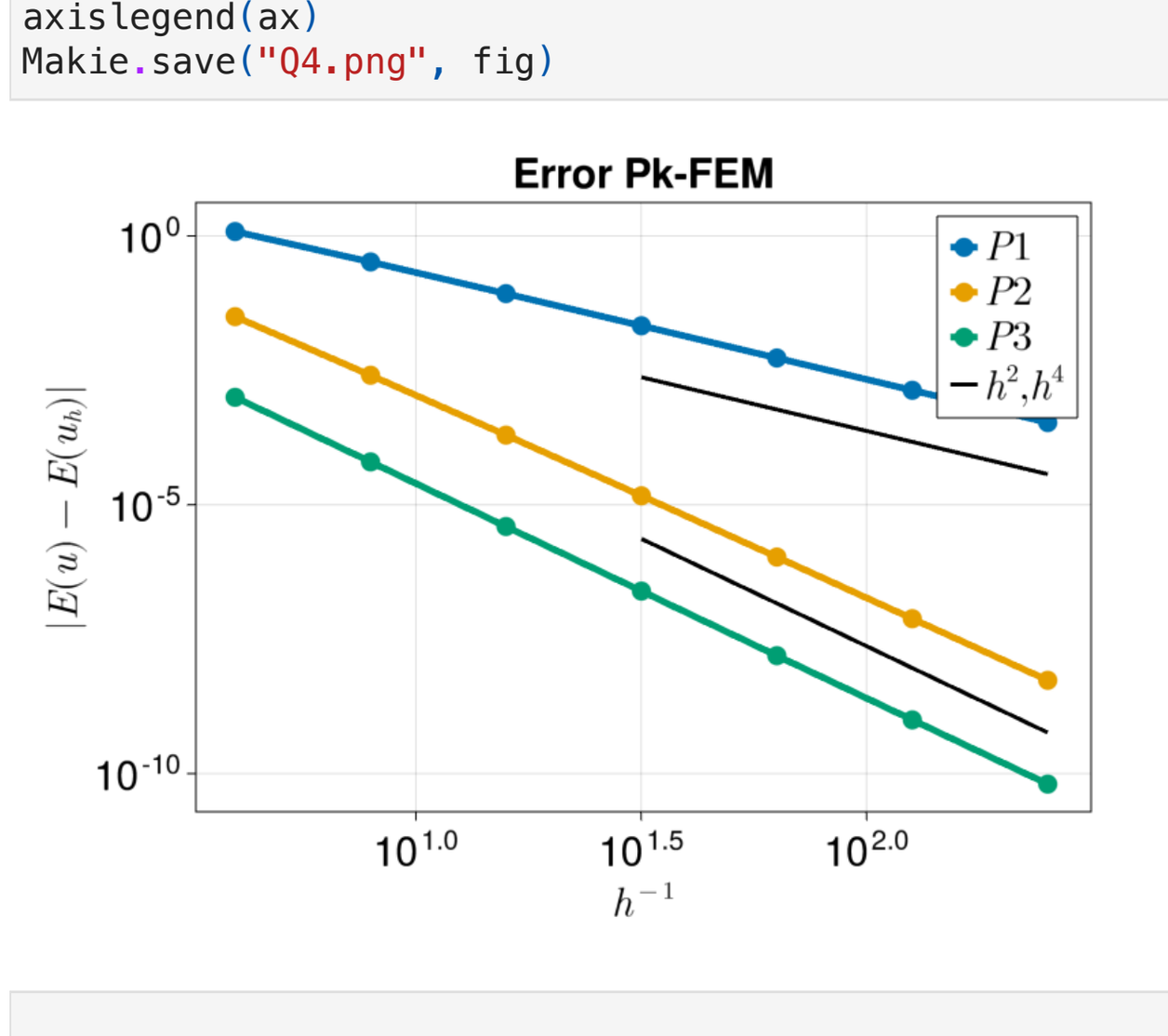
$$r^{-3}\partial_\theta^3 s_2 = -8r^{-1}\cos(2\theta)$$

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 clearly 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 [30]:
```
# Solution: implementation

ffun = ξ -> 5.0
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 = [ abs.(E[i] .- Elim[i]) for i = 1:3 ];
```

In [31]:
```
# show the convergence of total energy : 7/4 π ->
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 Pk-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")
scatterlines!(NN, err[3]; linewidth=5, markersize=20, label=L"P3")

lines!(NN1, 2.5 ./ NN1.^2; color=:black, linewidth=3, label=L"h^2, h^4")
lines!(NN1, 2.5 ./ NN1.^4; color=:black, linewidth=3)

axislegend(ax)
Makie.save("Q4.png", fig)
```



In [ ]: