AMSC 714 - HW6: Final Project

Ekrem Demirboga

May 17, 2025

1 Problem 1: Corner singularity

Let $\Omega = (-1,1) \times (-1,1) \backslash (0,1) \times (-1,0)$ be an L-shaped domain. Let u be the exact solution

$$u(r,\theta) = r^{\frac{2}{3}} \sin\left(\frac{2}{3}\theta\right)$$

for the Poisson equation $-\Delta u = f$ with Dirichlet boundary condition. The information about the domain and initial triangulation is in the subdirectory `L-shape-dirichlet`.

The tasks are:

(a) Find the corresponding right-hand side f and boundary conditions by direct differentiation of the given function u. Update the file `init_data.m`: change the information about f in `prob_data.f`, about the Dirichlet data in `prob_data.gD`, and about the Neumann data in `prob_data.gN`, as well as about the diffusion coefficient in `prob_data.a`. Write MATLAB functions `u_ex.m`, `grdu_ex.m` for each problem containing the exact function and its gradient.

(b) Select the marking strategy in `afem.m` to be either global refinement (GR), maximum strategy (MS), or Dörfler strategy (or bulk-chasing), also called Guaranteed error reduction strategy (GERS). Present a set of relevant pictures for various adaptive cycles showing the solution and mesh, and the error and estimators. Perform these experiments with threshold $\theta = 0.5$ for element marking. Stop either when the number of adaptive iterations is `max_iter` $= 34$ or the energy error is smaller than `tol`$= 3 \times 10^{-2}$. The residual estimators are computed in `estimate.m` with interpolation constants C_1, C_2 that you have to provide in `adapt.C`; these constants should be about 0.2 and are set in the file `init_data.m`.

(c) Experiment with the threshold $\theta = 0.2, 0.8$ and draw conclusions about its effect in the adaptive procedure.

(d) Discuss the regularity of the continuous solution in intermediate Sobolev spaces $H^s(\Omega)$, and the expected rate of convergence for piecewise linear FEM with uniform and adaptive refinements. Compare with the computed results and draw conclusions.

Solution

(a)

To find f, we compute the Laplacian of u. Let $\alpha = 2/3$, so $u(r,\theta) = r^\alpha \sin(\alpha\theta)$. The partial derivatives in polar coordinates are:

$$\begin{aligned}
\frac{\partial u}{\partial r} &= \alpha r^{\alpha-1}\sin(\alpha\theta) \\
\frac{\partial^2 u}{\partial r^2} &= \alpha(\alpha-1)r^{\alpha-2}\sin(\alpha\theta) \\
\frac{\partial u}{\partial \theta} &= \alpha r^{\alpha}\cos(\alpha\theta) \\
\frac{\partial^2 u}{\partial \theta^2} &= -\alpha^2 r^{\alpha}\sin(\alpha\theta)
\end{aligned}$$

The Laplacian in polar coordinates is $\Delta u = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}$. Substituting the derivatives:

$$\begin{aligned}
\Delta u &= \alpha(\alpha-1)r^{\alpha-2}\sin(\alpha\theta) + \frac{1}{r}(\alpha r^{\alpha-1}\sin(\alpha\theta)) + \frac{1}{r^2}(-\alpha^2 r^{\alpha}\sin(\alpha\theta)) \\
&= \alpha(\alpha-1)r^{\alpha-2}\sin(\alpha\theta) + \alpha r^{\alpha-2}\sin(\alpha\theta) - \alpha^2 r^{\alpha-2}\sin(\alpha\theta) \\
&= \left[\alpha(\alpha-1) + \alpha - \alpha^2\right] r^{\alpha-2}\sin(\alpha\theta) \\
&= \left[\alpha^2 - \alpha + \alpha - \alpha^2\right] r^{\alpha-2}\sin(\alpha\theta) = 0 \cdot r^{\alpha-2}\sin(\alpha\theta) = 0
\end{aligned}$$

Thus, the right-hand side is $f = -\Delta u = 0$. The problem is Laplace's equation. The problem specifies Dirichlet boundary conditions. This means $u = g_D$ on the entire boundary $\partial\Omega$, where g_D is the trace of the exact solution $u(r,\theta)$ on $\partial\Omega$.

To solve this problem using the provided `afem.m` codebase, several modifications were made:

- `u_ex1.m`: Implements the exact solution $u(r,\theta) = r^{2/3}\sin(\frac{2}{3}\theta)$. It takes Cartesian coordinates (x,y) as input, converts them to polar coordinates (r,θ) using `cart2pol`, adjusts θ to the range $[0, 2\pi)$ if necessary, and returns u. For $r = 0$, $u = 0$.
- `grdu_ex1.m`: Computes the Cartesian gradient $\nabla u = (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y})^T$. It uses the chain rule with polar derivatives: $\nabla u = \frac{\partial u}{\partial r}\nabla r + \frac{\partial u}{\partial \theta}\nabla\theta$.
- `init_data.m`: This file was updated to reflect Problem 1 specifics. Mainly, the Dirichlet boundary data is defined using newly created functions. The interpolation constants C_1 and C_2 are set to 0.2. Adaptive loop parameters are also added to make sure the loop stops when energy error drops below 3×10^{-3}.
- `afem.m`: The script was modified to correctly call the problem-specific exact solution and gradient functions for error calculation. Minor enhancements for console output and optional final convergence plotting were also included in the version of `afem.m` used for this homework. The updated files are added at the end of this document.

(b)

This part involves conducting numerical experiments using three different mesh refinement strategies: Global Refinement (GR), Maximum Strategy (MS), and Dörfler's Strategy (GERS). For MS and GERS, a marking threshold of $\theta = 0.5$ is used. The adaptive process stops when either the maximum number of iterations (34) is reached or the total error estimator falls below 3×10^{-2}. Estimator constants are $C_1 = C_2 = 0.2$.

The specific strategy is selected by setting 'adapt.strategy' in `init_data.m` to 'GR', 'MS', or 'GERS'. For the rest of this problem, we will stick with 'GERS'. During the execution, plots of the mesh and the solution are generated at each step. Numerical data, including the number of elements (N), H^1-error, L^2-error, and the total error estimator (η), are printed to the console and should be recorded for analysis.

GERS with $\theta = 0.5$:

GERS marks a minimal set of elements $\mathcal{M}_k$ such that their combined squared error indicators satisfy $\sum_{T \in \mathcal{M}_k} \eta_k^2(T) \geq (1 - 0.5)^2 \sum_{T \in \mathcal{T}_k} \eta_k^2(T)$. This is also expected to focus refinement near the singularity. The results for $\theta = 0.5$ can be seen in Figures 7, 2 and 1

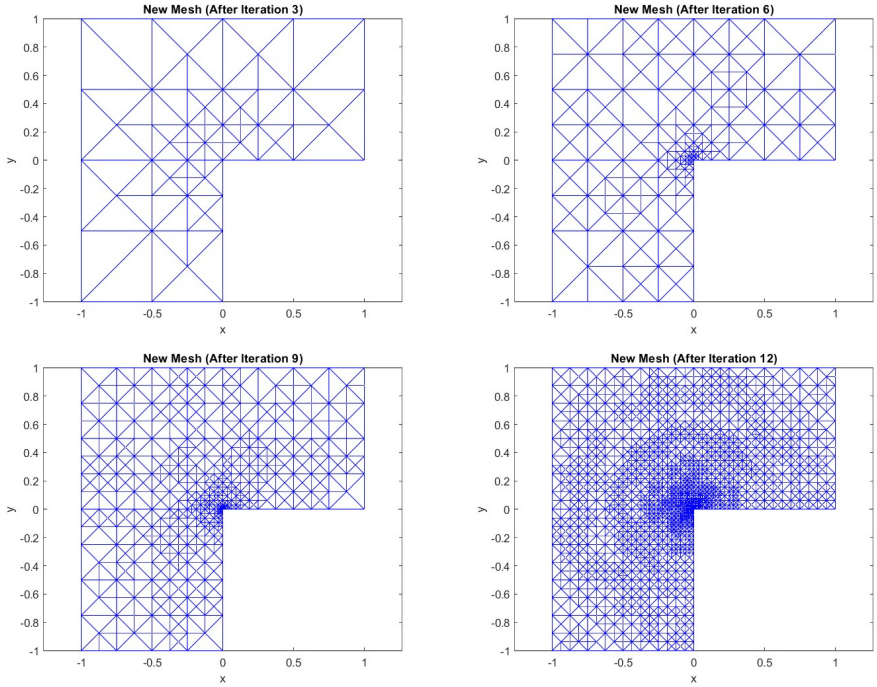

Figure 1: Mesh evolution under Dörfler's Strategy (GERS) with $\theta = 0.5$.

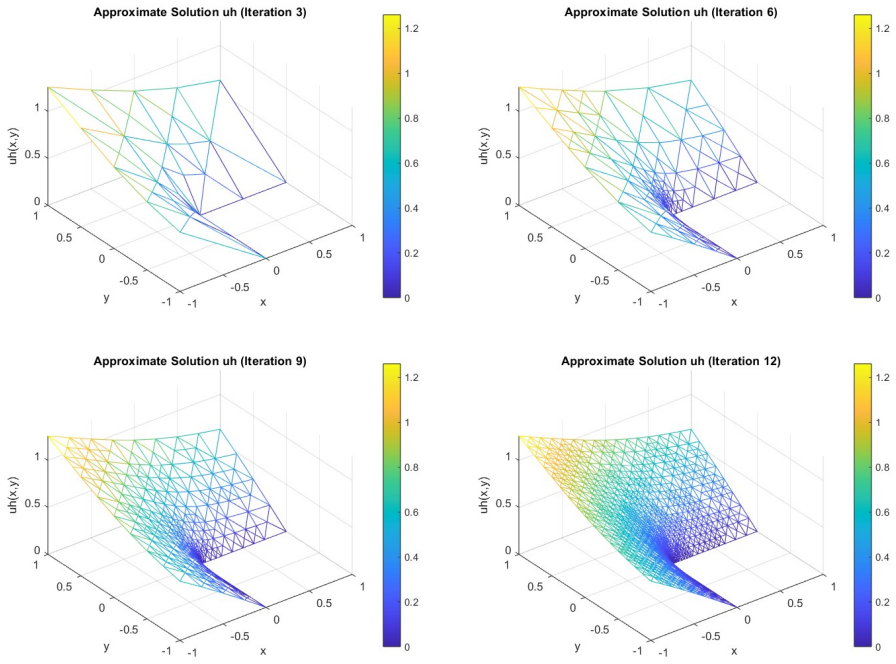

Figure 2: Solution u for each mesh given in Figure 1 under Dörfler's Strategy (GERS) with $\theta = 0.5$.

GERS with $\theta = 0.8$:

The results for $\theta = 0.8$ can be seen in Figures 8, 3 and 4

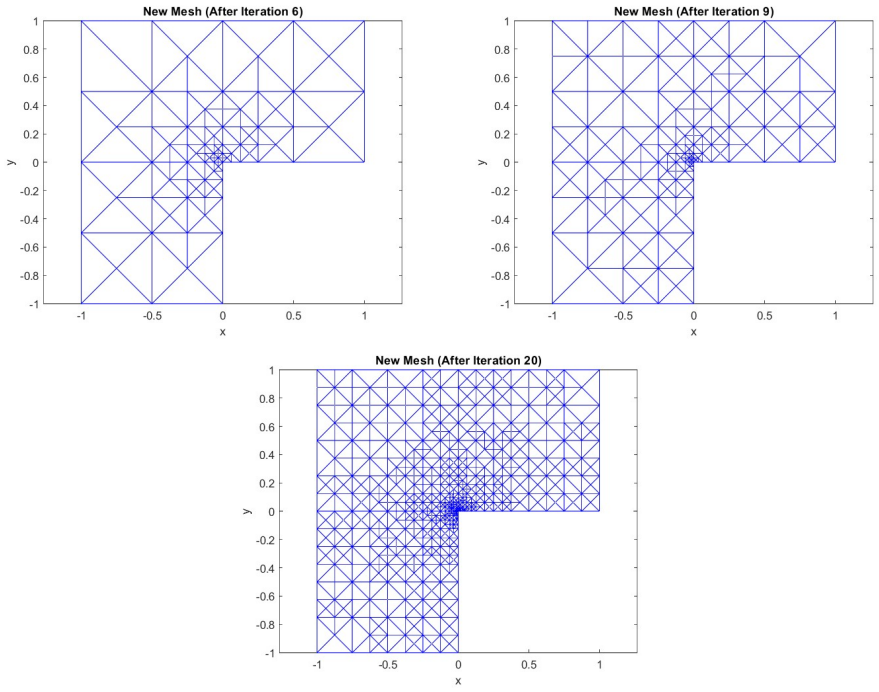

Figure 3: Mesh evolution under Dörfler's Strategy (GERS) with $\theta = 0.8$.

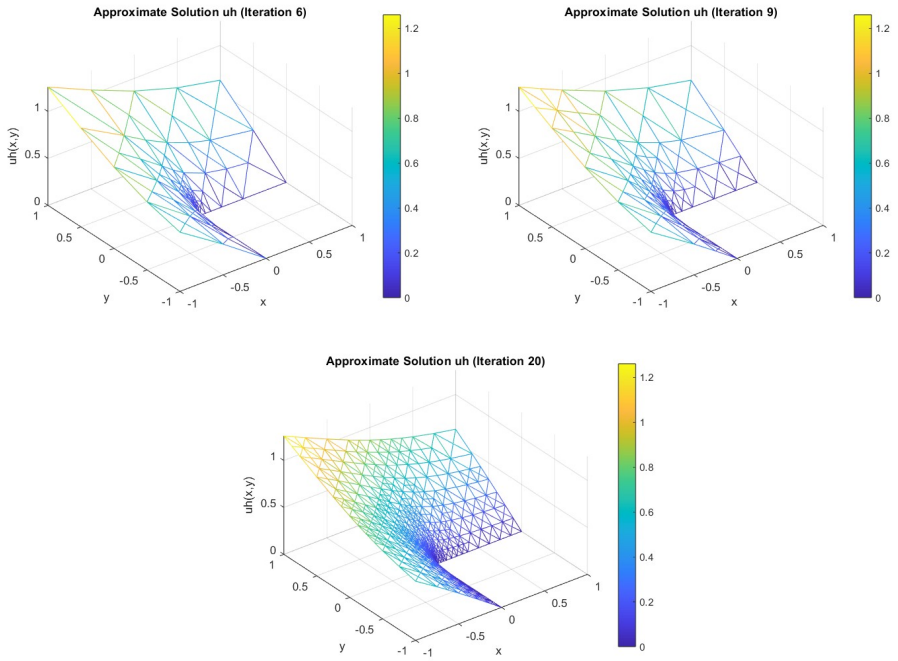

Figure 4: Solution u for each mesh given in Figure 1 under Dörfler's Strategy (GERS) with $\theta = 0.8$.

GERS with $\theta = 0.2$:

The results for $\theta = 0.2$ can be seen in Figures 9, 5 and 6

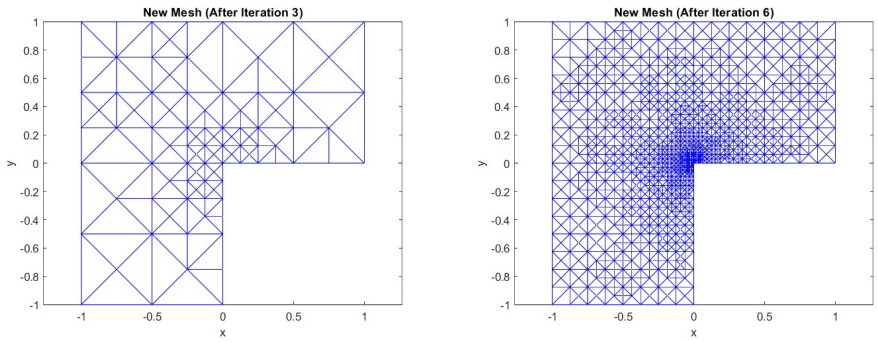

Figure 5: Mesh evolution under Dörfler's Strategy (GERS) with $\theta = 0.2$.

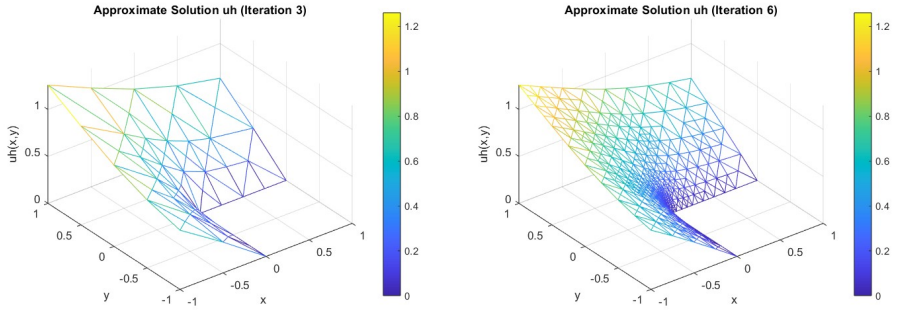

Figure 6: Mesh evolution under Dörfler's Strategy (GERS) with $\theta = 0.2$.

(c)

WE investigate how varying the marking threshold θ affects the adaptive procedure. Experiments were run with $\theta \in \{0.2, 0.8\}$, in addition to the $\theta = 0.5$ case from part (b).

We have observed that while it took 13 iterations for the system to converge when $\theta = 0.5$, it took 30 iterations when $\theta = 0.8$; and it took 7 iterations when $\theta = 0.2$. The convergence rates and number of iterations can be clearly seen by comparing the figure 7,8 and 9

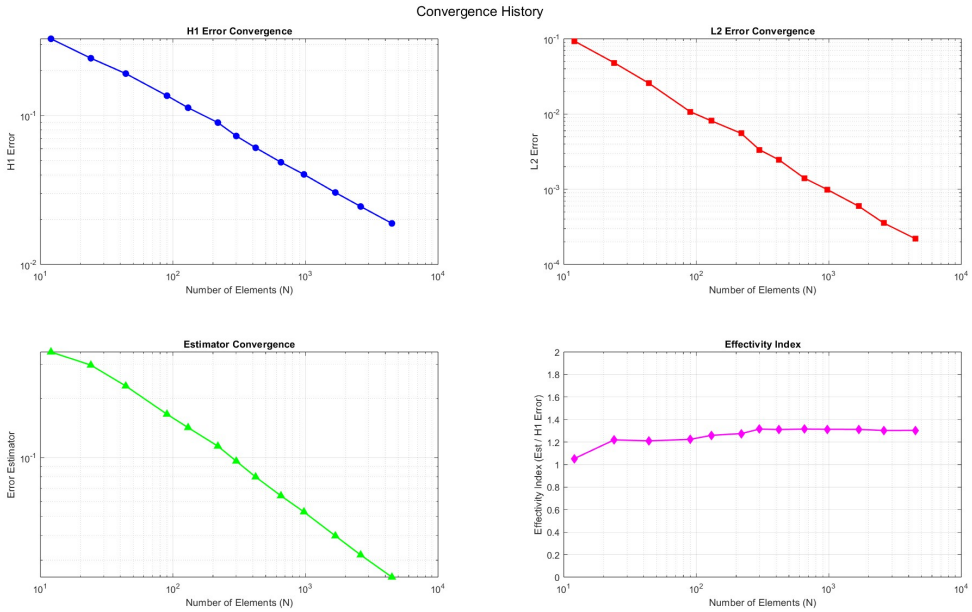

Figure 7: Convergence history for the GERS strategy with $\theta = 0.5$. The plots show (top-left) H^1-error vs. N, (top-right) L^2-error vs. N, (bottom-left) Error Estimator vs. N, and (bottom-right) Effectivity Index (Estimator / H^1-Error) vs. N. The adaptive process converged in 13 iterations.

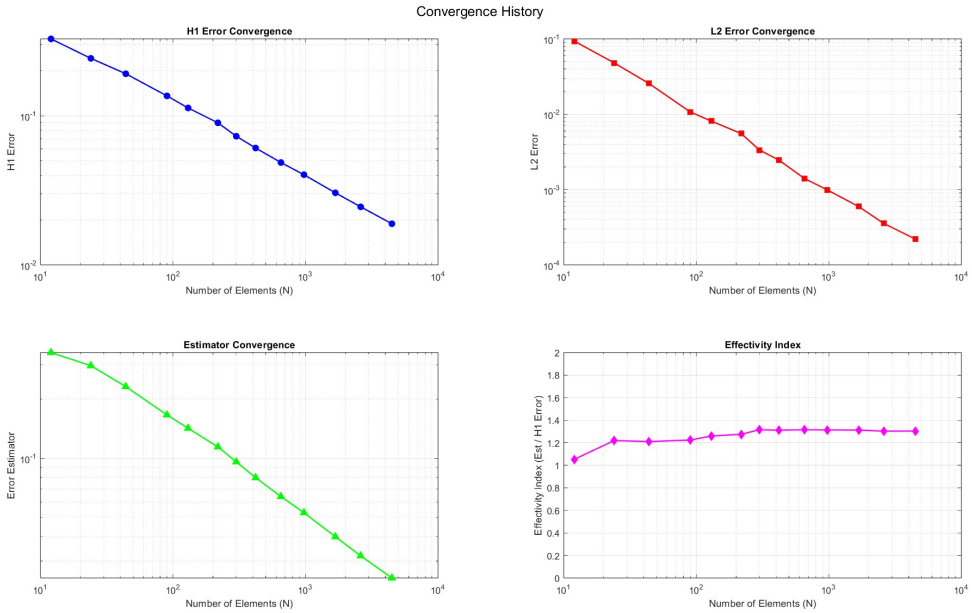

Figure 8: Same as figure 7 for $\theta = 0.8$

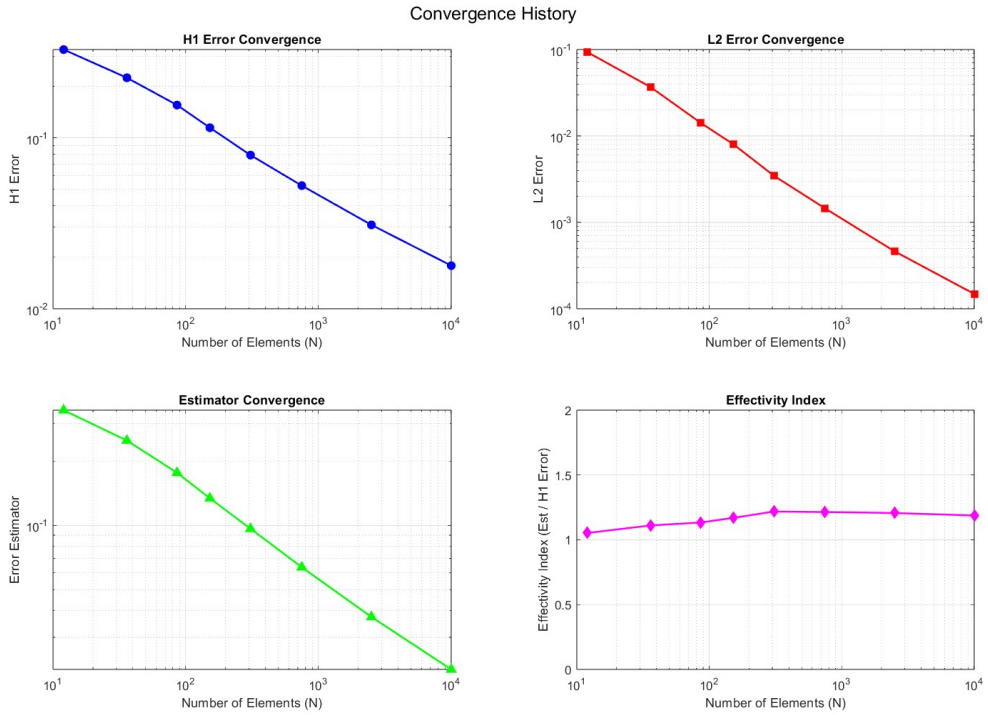

Figure 9: Same as figure 7 for $\theta = 0.2$

For the Dörfler's Strategy (GERS), the marking threshold θ significantly influences the adaptive procedure. The strategy marks a minimal set of elements $\mathcal{M}_k$ such that their contribution to the total squared error estimator, $\sum_{T\in\mathcal{M}_k} \eta_k^2(T)$, is at least $(1-\theta)^2 \sum_{T\in\mathcal{T}_k} \eta_k^2(T)$. Experimental results indicate a clear relationship between θ and the number of iterations required for convergence to the specified tolerance.

When $\theta = 0.2$, the algorithm targets a substantial portion of the error, $(1-0.2)^2 = 0.64$ (or 64%), for reduction in each step. This more aggressive approach, which refines a larger set of critical elements per iteration, led to convergence in only 7 iterations. In contrast, when $\theta = 0.8$, the target error portion is much smaller, $(1-0.8)^2 = 0.04$ (or 4%). This more conservative strategy refines fewer elements per iteration—only those with the very highest error indicators. Consequently, the error reduction per iteration is less pronounced, and the algorithm required 30 iterations to converge. The baseline case with $\theta = 0.5$ targeted $(1-0.5)^2 = 0.25$ (or 25%) of the error and converged in an intermediate 13 iterations. These observations suggest that a smaller θ in GERS results in a more aggressive refinement, leading to convergence in fewer iterations. However, this typically comes at the cost of refining more elements in each of those iterations. The choice of θ thus represents a trade-off between the number of iterations and the computational work performed per iteration, as well as the overall evolution of the mesh size.

(d)

The exact solution is $u(r,\theta) = r^{2/3}\sin\left(\frac{2}{3}\theta\right)$. The L-shaped domain has a re-entrant corner at the origin with an internal angle $\omega = \frac{3\pi}{2}$. The solution's singular behavior near $r = 0$ is characterized by the term $r^{\pi/\omega} = r^{2/3}$. Standard elliptic regularity theory states that $u \in H^{1+s}(\Omega)$ if $s < \pi/\omega$. In this case, $s < 2/3$. Thus, $u \in H^{1+2/3-\epsilon}(\Omega)$ for any $\epsilon > 0$, but $u \notin H^{1+2/3}(\Omega)$. This means $u \in H^1(\Omega)$, but its second derivatives are not in $L^2(\Omega)$ due to the singularity. The gradient behaves as $|\nabla u| \sim r^{-1/3}$ near the origin.

For a solution $u \in H^{1+s}(\Omega)$, the P_1 FEM error in the H^1-norm behaves as $||u - u_h||_{H^1(\Omega)} \approx CN^{-s/2}$ in 2D, where N is the number of elements.

Adaptive methods are designed to achieve optimal computational complexity. For singular solutions, they aim to achieve a convergence rate similar to that for smooth solutions on uniform meshes, which for P_1 elements in 2D is typically $O(N^{-1/2})$ for the energy error (and thus H^1-error). It is expected that adaptive refinement will significantly outperform uniform refinement.

In conclusion, the experimental results are anticipated to clearly show the superiority of adaptive refinement strategies in achieving higher convergence rates than uniform refinement for this singular problem, validating theoretical AFEM expectations.

2 Problem 5

Consider the model problem $-\Delta u = f$ with zero Dirichlet condition, and polynomial degree $k = 1$. Show that, up to higher order terms, the jump residual

$$\eta_{\mathcal{T}}(U) = \left(\sum_{S \in \mathcal{S}} \left\| h_S^{1/2} J_S \right\|_{L^2(S)}^2 \right)^{1/2}$$

bounds $\|\mathcal{R}\|_{H^{-1}(\Omega)}$. This entails that the residual estimator $\mathcal{E}_{\mathcal{T}}(U)$ is dominated by $\eta_{\mathcal{T}}(U)$. Hint: to estimate $\|\mathcal{R}\|_{H^{-1}(\Omega)}$ start with $\mathcal{R}(v)$ for any $v \in H_0^1(\Omega)$ and proceed as follows. First, use the partition of unity property $1 = \sum_{z \in \mathcal{N}} \phi_z$ in the error-residual relation, where $\{\phi_z\}_{z \in N}$ is the set of hat functions. Second, employ Galerkin orthogonality $\mathcal{R}(\phi_z) = 0$ to subtract the constant $v_z = \frac{1}{\int_{\omega_z} \phi_z} \int_{\omega_z} v \phi_z$ from v, where ω_z is the support of ϕ_z. Finally, rewrite $\int_{\omega_z} f(v - v_z)\phi_z$ exploiting the built-in weighted L^2-orthogonality.

3 Solution

We want to show that, up to higher-order terms, the jump residual $\eta_{\mathcal{T}}(U) = \left(\sum_{S \in \mathcal{S}} \left\| h_S^{1/2} J_S \right\|_{L^2(S)}^2 \right)^{1/2}$ bounds $\|\mathcal{R}\|_{H^{-1}(\Omega)}$.

The model problem is Poisson's equation with homogeneous Dirichlet boundary conditions:

$$-\Delta u = f \quad \text{in } \Omega \tag{1}$$
$$u = 0 \quad \text{on } \partial\Omega \tag{2}$$

Let $U \in V_h \subset H_0^1(\Omega)$ be the P_1 (piecewise linear, $k = 1$) finite element approximation satisfying

$$a(U, v_h) = (f, v_h) \quad \forall v_h \in V_h$$

where $a(w, z) = \int_\Omega \nabla w \cdot \nabla z \, dx$ and $(g, v) = \int_\Omega g v \, dx$.

The residual functional $\mathcal{R} : H_0^1(\Omega) \to \mathbb{R}$ is defined by:

$$\mathcal{R}(v) = (f, v) - a(U, v) \quad \forall v \in H_0^1(\Omega)$$

The $H^{-1}(\Omega)$ norm of the residual is:

$$\|\mathcal{R}\|_{H^{-1}(\Omega)} = \sup_{v \in H_0^1(\Omega), \|\nabla v\|_{L^2(\Omega)} = 1} |\mathcal{R}(v)|$$

Note that $\|v\|_{H_0^1(\Omega)} = \|\nabla v\|_{L^2(\Omega)}$ due to the Poincaré inequality for $v \in H_0^1(\Omega)$.

By Galerkin orthogonality, $\mathcal{R}(v_h) = 0$ for all $v_h \in V_h$. Let $P_C : H_0^1(\Omega) \to V_h$ be a Clément-type interpolation operator. This operator has the following key properties for $v \in H_0^1(\Omega)$:

- Approximation properties: For an element $T \in \mathcal{T}$ and an edge $S \in \mathcal{S}$,

$$\|v - P_C v\|_{L^2(T)} \leq C h_T \|\nabla v\|_{L^2(\omega_T)} \tag{3}$$
$$\|v - P_C v\|_{L^2(S)} \leq C h_S^{1/2} \|\nabla v\|_{L^2(\omega_S)} \tag{4}$$

  where ω_T and ω_S are patches of elements around T and S respectively, and h_T, h_S are their characteristic sizes. The constant C is independent of h_T, h_S, and v.

- For P_1 elements ($k = 1$), $P_C v$ can be constructed such that for each element T:

$$\int_T (v - P_C v) q \, dx = 0 \quad \text{for all } q \in P_0(T) \text{ (constants on T)}$$

  In particular, $\int_T (v - P_C v) \, dx = 0$.

Using Galerkin orthogonality, for any $v \in H_0^1(\Omega)$:

$$\mathcal{R}(v) = \mathcal{R}(v - P_C v)$$

Let $w = v - P_C v$. Then

$$\mathcal{R}(w) = (f, w) - a(U, w) = \sum_{T \in \mathcal{T}} \int_T f w \, dx - \sum_{T \in \mathcal{T}} \int_T \nabla U \cdot \nabla w \, dx$$

We integrate the second term by parts element by element:

$$-\int_T \nabla U \cdot \nabla w \, dx = \int_T (\Delta U) w \, dx - \int_{\partial T} (\nabla U \cdot n_T) w \, ds$$

Since U is piecewise linear (P_1), its Laplacian $\Delta U = 0$ on the interior of each element T. So,

$$\mathcal{R}(w) = \sum_{T \in \mathcal{T}} \int_T f w \, dx - \sum_{T \in \mathcal{T}} \int_{\partial T} (\nabla U \cdot n_T) w \, ds$$

The sum over element boundaries can be reorganized by edges $S \in \mathcal{S}$ (where $\mathcal{S} = \mathcal{S}_I \cup \mathcal{S}_B$ is the set of all interior and boundary edges). For an interior edge $S = T_1 \cap T_2$, let n_1 and n_2 be the outward unit normals to T_1 and T_2 respectively. Then $n_1 = -n_2 =: n_S$. The jump in the normal derivative flux across S is $J_S = [\![\nabla U \cdot n]\!]_S = \nabla U|_{T_1} \cdot n_S - \nabla U|_{T_2} \cdot n_S$. For a boundary edge $S \in \mathcal{S}_B$ (where $u = 0$, so the true normal flux is 0), the residual flux is $J_S = -\nabla U \cdot n_S$ (where n_S is the outward normal to Ω). Thus, defining J_S appropriately for all edges $S \in \mathcal{S}$:

$$-\sum_{T \in \mathcal{T}} \int_{\partial T} (\nabla U \cdot n_T) w \, ds = \sum_{S \in \mathcal{S}} \int_S J_S w \, ds$$

Let $R_T = f|_T$ be the element residual (since $\Delta U = 0$ on T). Then

$$\mathcal{R}(v) = \mathcal{R}(w) = \sum_{T \in \mathcal{T}} \int_T R_T w \, dx + \sum_{S \in \mathcal{S}} \int_S J_S w \, ds$$

Now we bound each term. First, the jump term contribution:

$$\left| \sum_{S \in \mathcal{S}} \int_S J_S w \, ds \right| \leq \sum_{S \in \mathcal{S}} \|J_S\|_{L^2(S)} \|w\|_{L^2(S)} \quad \text{(by Cauchy-Schwarz on each integral)} \tag{5}$$

$$\leq \sum_{S \in \mathcal{S}} \left\| h_S^{1/2} J_S \right\|_{L^2(S)} \left\| h_S^{-1/2} w \right\|_{L^2(S)} \tag{6}$$

$$\leq \left(\sum_{S \in \mathcal{S}} \left\| h_S^{1/2} J_S \right\|_{L^2(S)}^2 \right)^{1/2} \left(\sum_{S \in \mathcal{S}} \left\| h_S^{-1/2} w \right\|_{L^2(S)}^2 \right)^{1/2} \quad \text{(by Cauchy-Schwarz on the sum)} \tag{7}$$

The first term is the definition of $\eta_{\mathcal{T}}(U)$. For the second term, using the approximation property of $P_C v$ ($w = v - P_C v$):

$$\left\| h_S^{-1/2} w \right\|_{L^2(S)} = h_S^{-1/2} \|v - P_C v\|_{L^2(S)} \leq C h_S^{-1/2} h_S^{1/2} \|\nabla v\|_{L^2(\omega_S)} = C \|\nabla v\|_{L^2(\omega_S)}$$

So, due to the finite overlap of the patches ω_S (each element belongs to a bounded number of such patches):

$$\left(\sum_{S \in \mathcal{S}} \left\| h_S^{-1/2} w \right\|_{L^2(S)}^2 \right)^{1/2} \leq C \left(\sum_{S \in \mathcal{S}} \|\nabla v\|_{L^2(\omega_S)}^2 \right)^{1/2} \leq C' \|\nabla v\|_{L^2(\Omega)}$$

Therefore,

$$\left| \sum_{S \in \mathcal{S}} \int_S J_S w \, ds \right| \leq C \eta_{\mathcal{T}}(U) \|\nabla v\|_{L^2(\Omega)}$$

Next, we bound the element term contribution (data oscillation). Consider $\sum_{T\in\mathcal{T}}\int_T R_T w\,dx = \sum_{T\in\mathcal{T}}\int_T f w\,dx$. Let $\Pi_0 f$ be the L^2-projection of f onto piecewise constant functions on each element T, i.e., $(\Pi_0 f)|_T = \frac{1}{|T|}\int_T f\,dx$. Since $w = v - P_C v$ and $\int_T w\,dx = \int_T (v - P_C v)\,dx = 0$ (property of Clément interpolant for P_0 functions):

$$\int_T f w\,dx = \int_T (f - (\Pi_0 f)|_T) w\,dx + \int_T (\Pi_0 f)|_T w\,dx = \int_T (f - \Pi_0 f) w\,dx$$

So,

$$\sum_{T\in\mathcal{T}}\int_T f w\,dx = \sum_{T\in\mathcal{T}}\int_T (f - \Pi_0 f) w\,dx$$

Then,

$$\left|\sum_{T\in\mathcal{T}}\int_T (f - \Pi_0 f) w\,dx\right| \le \sum_{T\in\mathcal{T}} \|f - \Pi_0 f\|_{L^2(T)} \|w\|_{L^2(T)} \tag{8}$$

$$\le \left(\sum_{T\in\mathcal{T}} h_T^2 \|f - \Pi_0 f\|_{L^2(T)}^2\right)^{1/2} \left(\sum_{T\in\mathcal{T}} h_T^{-2} \|w\|_{L^2(T)}^2\right)^{1/2} \quad \text{(by Cauchy-Schwarz)} \tag{9}$$

Define the local data oscillation $\mathrm{osc}(f,h,T) = \|f - \Pi_0 f\|_{L^2(T)}$. The global data oscillation term is $\mathrm{Osc}(f,h) = \left(\sum_{T\in\mathcal{T}} h_T^2 \mathrm{osc}(f,h,T)^2\right)^{1/2}$. For the second term, using the approximation property of $P_C v$:

$$h_T^{-1} \|w\|_{L^2(T)} = h_T^{-1} \|v - P_C v\|_{L^2(T)} \le C h_T^{-1} h_T \|\nabla v\|_{L^2(\omega_T)} = C \|\nabla v\|_{L^2(\omega_T)}$$

So, due to finite overlap of the patches ω_T:

$$\left(\sum_{T\in\mathcal{T}} (h_T^{-1} \|w\|_{L^2(T)})^2\right)^{1/2} \le C \|\nabla v\|_{L^2(\Omega)}$$

Therefore,

$$\left|\sum_{T\in\mathcal{T}}\int_T f w\,dx\right| \le C \mathrm{Osc}(f,h) \|\nabla v\|_{L^2(\Omega)}$$

The term $\mathrm{Osc}(f,h)$ is a higher-order term if f is smooth. For example, if $f \in H^s(T)$ for $s > 0$, then by Bramble-Hilbert lemma, $\|f - \Pi_0 f\|_{L^2(T)} \le C h_T^s \|f\|_{H^s(T)}$. If $s = 1$, then $\mathrm{Osc}(f,h) \approx \left(\sum C^2 h_T^4 \|\nabla f\|_{L^2(T)}^2\right)^{1/2}$.

Combining the bounds for the jump and element terms:

$$|\mathcal{R}(v)| = |\mathcal{R}(w)| \le \left|\sum_{T\in\mathcal{T}}\int_T f w\,dx\right| + \left|\sum_{S\in\mathcal{S}}\int_S J_S w\,ds\right|$$

$$|\mathcal{R}(v)| \le C\left(\eta_{\mathcal{T}}(U) + \mathrm{Osc}(f,h)\right) \|\nabla v\|_{L^2(\Omega)}$$

Dividing by $\|\nabla v\|_{L^2(\Omega)}$ and taking the supremum over $v \in H_0^1(\Omega)$ with $\|\nabla v\|_{L^2(\Omega)} = 1$:

$$\|\mathcal{R}\|_{H^{-1}(\Omega)} = \sup_{v\in H_0^1(\Omega), \|\nabla v\|_{L^2(\Omega)}=1} \frac{|\mathcal{R}(v)|}{\|\nabla v\|_{L^2(\Omega)}} \le C\left(\eta_{\mathcal{T}}(U) + \mathrm{Osc}(f,h)\right)$$

This result shows that $\|\mathcal{R}\|_{H^{-1}(\Omega)}$ is bounded by the sum of the jump residual $\eta_{\mathcal{T}}(U)$ and the data oscillation term $\mathrm{Osc}(f,h)$. The term "up to higher order terms" refers to $\mathrm{Osc}(f,h)$. If f is sufficiently smooth (e.g., $f \in H^1(\Omega)$ such that $\mathrm{Osc}(f,h)$ is $O(h^2)$) or if f is approximated well by piecewise constants (for instance if $f \in P_0(\mathcal{T})$ then $\mathrm{Osc}(f,h) = 0$), then $\mathrm{Osc}(f,h)$ converges to zero at a faster rate than potentially $\eta_{\mathcal{T}}(U)$ (which is typically $O(h)$ for P_1 elements if the solution is not smooth enough) or is of a smaller order. In such cases, $\eta_{\mathcal{T}}(U)$ is the dominant term bounding $\|\mathcal{R}\|_{H^{-1}(\Omega)}$.

The standard a posteriori error estimator $\mathcal{E}_{\mathcal{T}}(U)$ is typically composed of terms related to element residuals and jump residuals. For P_1 elements, where $\Delta U = 0$ inside elements, the element residual is f. The estimator often takes the form $\mathcal{E}_{\mathcal{T}}(U)^2 \approx \sum_{T\in\mathcal{T}} h_T^2 \|f\|_{L^2(T)}^2 + \eta_{\mathcal{T}}(U)^2$. The derivation shows that the H^{-1} norm of the true residual $\mathcal{R}$ is controlled by $\eta_{\mathcal{T}}(U)$ and $\mathrm{Osc}(f,h)$. Since $\mathrm{Osc}(f,h)$ involves $h_T^2 \|f - \Pi_0 f\|_{L^2(T)}^2$, it is related to the element residual part but specifically measures the non-approximability of f by constants. If this data oscillation is higher order, then the jump residual $\eta_{\mathcal{T}}(U)$ is the dominant term controlling $\|\mathcal{R}\|_{H^{-1}(\Omega)}$. This, in turn, implies that $\eta_{\mathcal{T}}(U)$ is a dominant part of estimators that control the actual error $u - U$ in the energy norm, as $\|u - U\|_{H^1(\Omega)} \approx \|\mathcal{R}\|_{H^{-1}(\Omega)}$ (up to constants, via Riesz representation).

A Appendix: MATLAB Code Listings

This appendix includes listings of key MATLAB scripts modified or created for this problem.

afem.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AFEM in 2d for the elliptic problem
%  -div(a grad u) + b . grad u + c u = f    in Omega
%        u = gD   on Gamma_D
%   du/dn = gN   on Gamma_N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear previous figures and variables (optional, but good practice)
close all;
clearvars -except % keep breakpoints if any, or specify variables to keep
format compact; % More compact command window output

% --- Initialization Phase ---
% 1. Load problem-specific data and adaptive strategy parameters
%    This is the primary configuration file you will modify for each problem.
init_data;
% After this call, the global structs 'prob_data' and 'adapt' are populated,
% and variables like 'domain' and 'global_refinements' are set.

% 2. Declare global variables for mesh, solution (uh), and rhs (fh)
%    These are used by various functions to avoid passing large structures repeatedly.
global mesh uh fh

% 3. Initialize mesh and finite element function storage
fprintf('Loading initial mesh from domain: %s\n', domain);
mesh.elem_vertices      = load([domain '/elem_vertices.txt']);
mesh.elem_neighbours    = load([domain '/elem_neighbours.txt']);
mesh.elem_boundaries    = load([domain '/elem_boundaries.txt']);
mesh.vertex_coordinates = load([domain '/vertex_coordinates.txt']);
mesh.n_elem             = size(mesh.elem_vertices, 1);
mesh.n_vertices         = size(mesh.vertex_coordinates, 1);

% Initialize solution vector (uh) and right-hand side vector (fh)
% These will be resized if necessary during refinement.
uh = zeros(mesh.n_vertices, 1);
fh = zeros(mesh.n_vertices, 1);

% 4. Define exact solution and its gradient for error computation
%    !!! THIS IS THE MAIN PART OF AFEM.M YOU CHANGE FOR EACH PROBLEM !!!
%    For Problem 1, we use u_ex1.m and grdu_ex1.m
u_exact     = inline('u_ex1(x)', 'x');    % Exact solution function
grd_u_exact = inline('grdu_ex1(x)', 'x');  % Gradient of exact solution function
%    For a different problem, say Problem N, you would change these to:
%    u_exact     = inline('u_exN(x)', 'x');
%    grd_u_exact = inline('grdu_exN(x)', 'x');

% 5. Plot initial mesh
figure(1); % Figure for mesh plots
clf();     % Clear the figure
triplot(mesh.elem_vertices, ...
        mesh.vertex_coordinates(:,1), mesh.vertex_coordinates(:,2));
axis equal;
title('Initial Mesh');
xlabel('x'); ylabel('y');
drawnow;
fprintf('Initial mesh loaded. Press any key to continue...\n');
pause;

% 6. Perform initial global refinements if specified in init_data.m
if (global_refinements > 0)
```

```
    fprintf('Performing %d initial global refinement(s)...\n', global_refinements);
    mesh.mark = global_refinements * ones(mesh.n_elem, 1); % Mark all elements
    refine_mesh; % This function modifies global 'mesh', 'uh', 'fh'

    figure(1); % Update mesh plot
    clf();
    triplot(mesh.elem_vertices, ...
            mesh.vertex_coordinates(:,1), mesh.vertex_coordinates(:,2));
    axis equal;
    title(['Mesh after ', num2str(global_refinements), ' global refinement(s)']);
    xlabel('x'); ylabel('y');
    drawnow;
    fprintf('Global refinement(s) done. Press any key to start adaptive loop...\n');
    pause;
end

% --- Adaptive Loop (SOLVE -> ESTIMATE -> MARK -> REFINE) ---
iter_counter = 1;
all_n_elem = []; % To store number of elements per iteration
all_H1_err = []; % To store H1 error per iteration
all_L2_err = []; % To store L2 error per iteration
all_est    = []; % To store estimator value per iteration

fprintf('\nStarting adaptive loop...\n');
fprintf('Max iterations: %d, Tolerance for estimator: %.2e\n', adapt.max_iterations,
    adapt.tolerance);
fprintf('
    ----------------------------------------------------------------------------------------\
    n');
fprintf('| Iter | Elements | H1 Error   | L2 Error   | Estimator  | Max Estimator |
    Theta (GERS) |\n');
fprintf('
    |------|----------|------------|------------|------------|---------------|--------------|\
    n');

figure(2); % Figure for solution plots
clf();

while (1)
    % 1. SOLVE: Assemble the system and solve for uh
    assemble_and_solve; % Modifies global 'uh' and 'fh'

    % Plot the approximate solution uh
    figure(2);
    clf();
    trimesh(mesh.elem_vertices, ...
            mesh.vertex_coordinates(:,1), mesh.vertex_coordinates(:,2), ...
            uh);
    axis equal; view(3); % Use a 3D view for surface plot
    title(['Approximate Solution uh (Iteration ', num2str(iter_counter), ')']);
    xlabel('x'); ylabel('y'); zlabel('uh(x,y)');
    colorbar;
    drawnow;

    % 2. ESTIMATE: Compute the a posteriori error estimator
    est = estimate(prob_data, adapt); % Returns global estimator, updates mesh.
        estimator

    % Compute actual errors (if exact solution is known)
    current_H1_err = H1_err(mesh.elem_vertices, mesh.vertex_coordinates, uh,
        grd_u_exact);
    current_L2_err = L2_err(mesh.elem_vertices, mesh.vertex_coordinates, uh, u_exact);

    % Store data for plotting later
    all_n_elem(iter_counter) = mesh.n_elem;
    all_H1_err(iter_counter) = current_H1_err;
    all_L2_err(iter_counter) = current_L2_err;
```

```
    all_est(iter_counter)    = est;

    % Display iteration statistics
    % For GERS, it might be insightful to see the gamma value, but mark_elements doesn't return it.
    % We can display mesh.max_est which is used by MS and GERS.
    fprintf('| %4d | %8d | %.6e | %.6e | %.6e | %.6e |\n',...
           iter_counter, mesh.n_elem, current_H1_err, current_L2_err, est, mesh.max_est);

    % 3. Check stopping criteria
    if ((iter_counter >= adapt.max_iterations) || (est < adapt.tolerance))
        fprintf('-------------------------------------------------------------------------------------\n');
        if (iter_counter >= adapt.max_iterations)
            fprintf('Stopping: Maximum number of iterations (%d) reached.\n', adapt.max_iterations);
        end
        if (est < adapt.tolerance)
            fprintf('Stopping: Estimator (%.2e) is below tolerance (%.2e).\n', est, adapt.tolerance);
        end
        break; % Exit the adaptive loop
    else
        % 4. MARK: Mark elements for refinement
        mark_elements(adapt); % Modifies mesh.mark based on adapt.strategy

        % 5. REFINE: Refine the mesh
        % refine_mesh uses mesh.mark and updates global 'mesh', 'uh', 'fh'
        fprintf('Refining mesh for iteration %d...\n', iter_counter + 1);
        n_refined_total_in_step = refine_mesh; % n_refined_total_in_step is the number of bisections performed
        fprintf('Number of bisections performed in this refinement step: %d\n', n_refined_total_in_step);


        % Plot the new mesh
        figure(1);
        clf();
        triplot(mesh.elem_vertices, ...
                mesh.vertex_coordinates(:,1), mesh.vertex_coordinates(:,2) );
        axis equal;
        title(['New Mesh (After Iteration ', num2str(iter_counter), ')']);
        xlabel('x'); ylabel('y');
        drawnow;

        fprintf('Refinement done. Press any key for next iteration...\n');
        pause;
    end
    iter_counter = iter_counter + 1;
end

fprintf('\nAdaptive process finished.\n');

% --- Optional: Plot convergence history ---
if iter_counter > 1
    figure(3);
    clf;
    subplot(2,2,1);
    loglog(all_n_elem, all_H1_err, 'b-o', 'LineWidth', 1.5, 'MarkerFaceColor', 'b');
    xlabel('Number of Elements (N)');
    ylabel('H1 Error');
    title('H1 Error Convergence');
    grid on;

    subplot(2,2,2);
```

```
    loglog(all_n_elem, all_L2_err, 'r-s', 'LineWidth', 1.5, 'MarkerFaceColor', 'r');
    xlabel('Number of Elements (N)');
    ylabel('L2 Error');
    title('L2 Error Convergence');
    grid on;

    subplot(2,2,3);
    loglog(all_n_elem, all_est, 'g-^', 'LineWidth', 1.5, 'MarkerFaceColor', 'g');
    xlabel('Number of Elements (N)');
    ylabel('Error Estimator');
    title('Estimator Convergence');
    grid on;

    subplot(2,2,4);
    if ~isempty(all_H1_err) && all_H1_err(1) > 0 % Avoid division by zero if error is
        already zero
        effectivity_index = all_est ./ all_H1_err;
        semilogx(all_n_elem, effectivity_index, 'm-d', 'LineWidth', 1.5, '
            MarkerFaceColor', 'm');
        xlabel('Number of Elements (N)');
        ylabel('Effectivity Index (Est / H1 Error)');
        title('Effectivity Index');
        grid on;
        ylim([0 max(2, max(effectivity_index))]); % Adjust y-axis for better
            visualization
    end
    sgtitle('Convergence History'); % Super title for the subplots
    drawnow;
end
```

Listing 1: Exact solution function `u_ex1.m` for Problem 1.

`init_data.m`

```
% File: init_data.m
% Updated for Problem 1: Corner Singularity

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  problem  data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Folder or directory where the domain mesh is described
% For Problem 1, this should be the L-shaped domain.
domain = 'L_shape_dirichlet'; % This should already be set if you are using the L-shape
    files

% Initial global refinements (can be adjusted based on experiments)
global_refinements = 1; % Default value from the provided file

global prob_data

% PDE: -div(a grad u) + b . grad u + c u = f

% Diffusion coefficient (a) of the equation
% For Poisson equation -Delta u = f, which is -div(1 * grad u) = f, so a=1.
prob_data.a = 1;

% Convection coefficient (b) of the equation (row vector)
% For Poisson equation, there is no convection term.
prob_data.b = [0.0  0.0];

% Reaction coefficient (c) of the equation
% For Poisson equation, there is no reaction term.
prob_data.c = 0.0;

```

```
% Right-hand side function f
% For Problem 1, we found f = 0.
prob_data.f = inline('0', 'x');

% Dirichlet data, function g_D
% This is the exact solution evaluated on the boundary.
% We will use the u_ex1.m function created in Step 2.
prob_data.gD = inline('u_ex1(x)','x'); % Changed from u_ex3

% Neumann data, function g_N
% Problem 1 specifies Dirichlet boundary conditions on the entire boundary.
% So, gN is not actively used if the boundary is correctly marked as Dirichlet.
prob_data.gN = inline('0', 'x'); % Placeholder, as it's a full Dirichlet problem

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  data for a posteriori estimators and adaptive strategy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global adapt


adapt.C(1) = 0.2;
adapt.C(2) = 0.2;

% Stop when energy error is smaller than tol = 3e-2
adapt.tolerance = 3e-2;

% Stop either when iterations = max_iter = 34 or tolerance is met.
adapt.max_iterations = 34;

% Marking_strategy, possible options are
% GR: global (uniform) refinement,
% MS: maximum strategy,
% GERS: guaranteed error reduction strategy (D rfler's)
adapt.strategy = 'GERS'; % Default is GERS.

adapt.n_refine = 2; % Default from provided file

% Parameters of the different marking strategies
% Perform these experiments with threshold theta=0.5 for element marking.
adapt.MS_gamma = 0.5;

% GERS: guaranteed error reduction strategy (D rfler's)
% If using GERS, theta=0.5 should apply here too.
adapt.GERS_theta_star = 0.2; % Changed from 0.8 to match problem spec theta=0.5
adapt.GERS_nu = 0.1; % Default from provided file
```

Listing 2: Exact solution function `u_ex1.m` for Problem 1.

`u_ex1.m`

```
% File: u_ex1.m
function u = u_ex1(x_coords)


% Convert Cartesian coordinates to polar coordinates
[t, r] = cart2pol(x_coords(1), x_coords(2)); % t is angle (theta), r is radius

% Handle the case r=0 separately
% The solution u is 0 at the origin (r=0).
if r == 0
    u = 0;
    return;
end

% cart2pol returns theta in the interval [-pi, pi].
```

```
% For consistency in problems involving specific angular domains (like the L-shape),
% it's often useful to adjust theta to be in [0, 2*pi).
if (t < 0)
  t = t + 2*pi;
end

% Calculate the exact solution
u = r^(2/3) * sin(2*t/3);

end
```

Listing 3: Exact solution function `u_ex1.m` for Problem 1.

`grdu_ex1.m`

```
% File: grdu_ex1.m
function g = grdu_ex1(x_coords)
% Gradient of the exact solution u(r,theta) = r^(2/3)*sin(2*theta/3) for Problem 1
% x_coords is a 2x1 vector [x_coord; y_coord]

% Extract Cartesian coordinates
x = x_coords(1);
y = x_coords(2);

% Convert Cartesian coordinates to polar coordinates
[t, r] = cart2pol(x, y); % t is angle (theta), r is radius

% Handle r=0: the gradient is singular (undefined) at r=0.
% The function u is in H^1, but its classical gradient blows up at the origin.
if r == 0
    g = [NaN; NaN]; % Represent undefined gradient as NaN (Not-a-Number)
    return;
end

% cart2pol returns theta in the interval [-pi, pi].
% Adjust theta to be in [0, 2*pi) for consistency.
if (t < 0)
  t = t + 2*pi;
end

% Partial derivative of u with respect to r
dudr = (2/3) * r^(-1/3) * sin(2*t/3);

% Gradient of r in Cartesian coordinates: grad(r) = [cos(theta); sin(theta)] = [x/r; y/
    r]
grad_r = [x/r; y/r]; % Note: this is x_coords/r

% Partial derivative of u with respect to theta
dudt = r^(2/3) * cos(2*t/3) * (2/3);

% Gradient of theta in Cartesian coordinates: grad(theta) = [-sin(theta)/r; cos(theta)/
    r] = [-y/r^2; x/r^2]
grad_theta = [-y/r^2; x/r^2];

% Chain rule: grad(u) = dudr * grad(r) + dudt * grad(theta)
g = dudr * grad_r + dudt * grad_theta;

end
```

Listing 4: Gradient of exact solution function `grdu_ex1.m` for Problem 1.