Math 292C (Spring 1998, Instructor: M. Holst)

Final Project (*A posteriori* error estimates and multigrid)

Handed out: 3 June 1998
Due in class: 12 June 1998

You have been provided with the MATLAB finite element code FEMBIF which can treat the following class of problems numerically (for $d = 2$):

$$\begin{aligned}
-\nabla \cdot (a(x)\nabla u(x)) + b(x,u) &= f(x), \quad \text{in } \Omega \subset \mathbb{R}^d, \\
n(x) \cdot (a(x)\nabla u(x)) + c(x,u) &= h(x), \quad \text{on } \partial_N\Omega, \\
u(x) &= g(x), \quad \text{on } \partial_D\Omega, \\
\partial\Omega &= \partial_D\Omega \cup \partial_N\Omega, \\
\{\} &= \partial_D\Omega \cap \partial_N\Omega.
\end{aligned}$$

The MATLAB code FEMBIF discretizes this problem with Petrov-Galerkin finite element methods, based on piecewise-linear basis functions with local support. The resulting implicit nonlinear algebraic equations are then solved with a Newton iteration. Each Newton iteration requires the solution of a large, sparse, unstructured linear algebraic system of equations (the linearization or Jacobian system). These equations are solved using MATLAB's builtin sparse direct solver (some variant of sparse Gaussian elimination), or any of the iterative methods (classical, CG, or multigrid) that you implemented in the previous homeworks. In the previous homework, you extended the refinement routine to handle selective refinement, in that you can now mark a subset of the total set of elements, and then refine (via bisection) this subset, independent of the remaining unmarked elements. (You also implemented a conformity generator, which then does a small amount of additional refinement around the marked region just to ensure that the final mesh is conforming.)

Our tasks in this homework are to implement the residual-based error estimator that we derived this quarter, and to modify the multigrid routine you wrote last quarter to be more suitable for the new adaptive nature of the code.

- **Problem 1.** (*A posteriori* error estimation.)
  Implement the residual-based error estimator to use as a marking procedure, just before calling the refinement routine as you go through the solve-mark-refine loop in the code. The error estimator satisfied the following approximation theorem, for a piecewise-linear galerkin discretization of the problem above, for the case of: $d = 2$, $a = 1$, $b = c = g = h = 0$, and $\partial_D\Omega \equiv \partial\Omega$.
  THEOREM 0.1. *Let T_h be a shape-regular triangulation with shape parameter κ. Then there exists a constant $C = C(\kappa, \Omega)$ such that*

$$\|u - u_h\|_{H^1(\Omega)} \le C \left\{ \sum_{\tau \in T_h} \eta_{\tau,R}^2 \right\}^{1/2},$$

  and

$$\eta_{\tau,R} \le C \left\{ \|u - u_h\|_{H^1(\omega_\tau)}^2 + \sum_{\tau' \subset \omega_\tau} h_\tau^2 \|f - f_h\|_{L^2(\tau')}^2 \right\}^{1/2},$$

  where

$$\eta_{\tau,R} = \left\{ h_\tau^2 \|R_\tau\|_{L^2(T)}^2 + \frac{1}{2} \sum_{e \in \partial\tau} h_e \|R_e\|_{L^2(e)}^2 \right\}^{1/2}, \quad \tau \in T_h.$$

$$R_\tau = R_\tau(u_h) = \nabla^2 u_h + f = f - (-\nabla^2 u), \qquad R_e = R_e(u_h) = \left[\frac{\partial u_h}{\partial n}\right]_e = [\nabla u_h]_e \cdot n_e,$$

$$w_\tau = \bigcup \left\{ \tau' \in T_h | \tau \equiv \tau', \text{ or the two have a common edge} \right\}, \qquad w_e = \bigcup \left\{ \tau' \in T_h | e \in \partial\tau' \right\}.$$

Hints: You need to write a (very) short routine that traverses the elements one time, and produces a single real number, call it ERR, for each element. Assuming you want the error (in the H^1-form) in each element to fall under some tolerance TOL, you then place any element when fails the test (ERR < TOL) into the refinement queue.

To produce the indicator ERR, you just need to computer $\eta_{\tau,R}$ for each element, using numerical quadrature, as described in class. Note that you have everything available in the routine `assem.m` to evaluate the indicator

$$\text{ERR} = \eta_{\tau,R} = \left\{ h_\tau^2 \|R_\tau\|_{L^2(T)}^2 + \frac{1}{2} \sum_{e \in \partial\tau} h_e \|R_e\|_{L^2(e)}^2 \right\}^{1/2}$$

on an arbitrary element τ. In particular, you can use the quadrature rules for the volume and edge integrals that we use to produce the stiffness matrix. In fact, you should probably start with `cp assem.m estim.m`, and then start editing `estim.m`...

- **Problem 2.** (Multigrid – extra credit.)

Implement the algebraic representation of the hierarchical basis multigrid method discussed in class, as a replacement for the multigrid iteration we wrote for the uniform refinement case.

Note that you have everything already in place, since the key matrix S, the basis-transformation matrix, is formed from the prolongation matrix produced in your refinement routine in a very simple way, as follows. If the prolongation matrix $\hat{P}$ that you are using now has the form:

$$\hat{P} = \begin{pmatrix} I \\ P \end{pmatrix},$$

then the new matrix S takes the form:

$$\hat{S} = \begin{pmatrix} [\hat{P}] & \begin{matrix} 0 \\ I \end{matrix} \end{pmatrix} = \begin{pmatrix} I & 0 \\ P & I \end{pmatrix}.$$

You can easily block the nodal stiffness matrix as follows:

$$A^N = \begin{pmatrix} A_{11}^N & A_{12}^N \\ A_{21}^N & A_{22}^N \end{pmatrix},$$

since you know exactly how many unknowns there were before you refined the mesh. You could (but wouldn't want to do this!) form the hierarchical system as follows:

$$A^H = S^T A^N S == \begin{pmatrix} I & P \\ 0 & I \end{pmatrix} \begin{pmatrix} A_{11}^N & A_{12}^N \\ A_{21}^N & A_{22}^N \end{pmatrix} \begin{pmatrix} I & 0 \\ P & I \end{pmatrix} = \begin{pmatrix} A_{11}^H & A_{12}^H \\ A_{21}^H & A_{22}^H \end{pmatrix},$$

where

$$A_{11}^H = A_{11}^N + P^T A_{21}^N + A_{12}^N P + P^T A_{22}^N P$$

$$A_{12}^H = A_{12}^N + P^T A_{22}^N$$

$$A_{21}^H = A_{21}^N + A_{22}^N P$$

$$A_{22}^H = A_{22}^N$$