A Posteriori Error Estimates and Adaptivity

Assignment A1

A Posteriori Error Estimation

Having obtained a finite element approximation u_h to the solution u of a differential equation $Lu = f$, we would like to be able to estimate the error $e = u - u_h$. This can be done with the so-called *a posteriori error estimate*, which gives a computable estimate of the error e by evaluating the residual $r = f - Lu_h$. The idea is that if the residual r is small then (hopefully) this holds for the error e as well. In other words, if u_h fits well into the equation then u_h can not be very different from u. A posteriori error estimates are often used to analyze the accuracy of finite element approximations and to design algorithms which minimize the computational cost of computing these approximations. This is accomplished via adaptive mesh refinement which simply means inserting additional nodes into the mesh to resolve regions where the error is large, for example near singularities or discontinuities. The aim of this assignment is to extend the code `my_first_fem_solver.m`, from Computer Lab 1, to include a posteriori based adaptive mesh refinement.

Let

$$-u'' = f, \quad -1 < x < 1, \tag{1}$$

$$u(-1) = u(1) = 0, \tag{2}$$

and let u_h be a finite element approximation on a mesh $-1 = x_0 < x_1 < \cdots < x_N = 1$. Then the following a posteriori error estimate holds

$$\|e'\| \leq C \left(\sum_{i=1}^{N} \rho_i^2 \right)^{1/2}, \tag{3}$$

where the *element residual* ρ_i is defined by

$$\rho_i = h_i \| f + u_h'' \|_{L^2(I_i)}, \tag{4}$$

and $I_i = [x_i, x_{i+1}]$.

For piecewise linear approximations $u_h'' = 0$, but one can approximate it using a discrete Laplacian as explained in Lab 1:

$$\rho_i = h_i \| f + \Delta_h u_h \|_{L^2(I_i)}. \tag{5}$$

Problem 1. Prove equation (3) with ρ_i defined by equation (4). What is C in this case?

Adaptive Mesh Refinement

The a posteriori error estimate (3) gives a computable bound on the error contribution from element I_i to the total error e, namely the element residual ρ_i. Apparently, the local error depends on the residual $f + u_h''$, and the size of the element h_i. Thus to increase the accuracy of the approximation u_h we can simply split the elements with biggest contribution to the error into smaller ones. In doing so we strive for a uniform distribution of the overall error among the elements. This reasoning leads us to the following algorithm for implementing adaptive mesh refinement.

1. Give a (coarse) mesh with $N + 1$ nodes.
2. While $N + 1$ is not too large
3. Compute the finite element approximation u_h by solving

$$A\xi = b,$$

where A is the stiffness matrix and b is the load vector

4. Evaluate the element residuals ρ_i, $i = 1, 2, \ldots, N$

5. Refine the elements with biggest contribution to the error

6. End while loop

The adaptive algorithm above consists of four main components:

- Computation of the element residuals ρ_i.
- Selection of elements to be refined.
- A refinement procedure.
- A stopping criterion.

We now present the implementation of these four steps.

In practice, we calculate the element residuals ρ_i using numerical quadrature, for instance trapezoidal rule. It is convenient to store them as a vector, say, `rho2`, when approximating ρ_i^2:

```
rho2 = zeros(N,1);
for i = 1:N
  h = x(i+1)-x(i);
  rho2(i) = ????;
end
```

Here `x` is a vector of node coordinates and `N` is the number of elements.

Problem 2. Fill in the questions marks in the algorithm for computing rho2(i), using trapesoidal quadrature rule.

There are different possibilities for selecting the elements to be refined given the element residuals ρ_i. For instance, a popular method is to refine element i if

$$\rho_i > \lambda \max_{i=1,2,\ldots,N} \rho_i, \tag{6}$$

where $0 \leq \lambda \leq 1$ is a parameter which must be chosen by the user. Note that $\lambda = 0$ gives a uniform refinement, while $\lambda = 1$ gives no refinement at all.

The refinement procedure consists of the insertion of new nodes at the center of the elements chosen for refinement. In other words, if we are refining $[x_i, x_{i+1}]$, then we replace it by $[x_i, (x_i + x_{i+1})/2] \cup [(x_i + x_{i+1})/2, x_{i+1}]$. This is easily implemented by adding the coordinate of the midpoint of all the elements to be refined to the vector of nodes `x`, and then to sort it. We list the code:

```
lambda = 0.9 for i = 1:length(rho2)
  if rho2(i) > lambda*max(rho2)
    x = [x (x(i+1)+x(i))/2];
  end
end
x = sort(x);
```

The stopping criterion determines when the adaptive algorithm should stop. It can, for instance, take the form of a maximum bound on the number of nodes, the memory usage, the time of the computation, the total size of the residual, or a combination of these.

Problem 3. Put the pieces together and implement the code outlined above. Solve $-u'' = e^{-1000x^2} + 10^{-3}$, on the interval $-1 < x < 1$, and with the boundary conditions $u(-1) = u(1) = 0$. Start from a uniform grid with say 5 nodes. Choose $\lambda = \{0.05, 0.25, 0.5, .9\}$ and stop the refinement when the number of nodes exceeds 10^3. Illustrate the local mesh sizes h_i of the final meshes using the different values of λ. This can be done by plotting `plot(x(2;end),[1./diff(x)])`, where `x` is the coordinates of the final mesh. What does the pictures show? How does the choice of λ influence final solution u_h?