Courses/FEM/modules/mesh

From Icarus

< Courses | FEM

Contents

Adaptivity/Mesh operations

Precondition

- Function approximation
- Galerkin's method
- Error estimation

Introduction

The a posteriori error estimate indicates the size of the error of an approximation on a given mesh, it is natural to try to use this information to generate a better mesh that gives more accuracy. This is the basis of adaptive error control.

The computational problem that arises once a boundary value problem is specified is to find a mesh such that the finite element approximation achieves a given level of accuracy, or in other words, such that the error of the approximation is bounded by an *error tolerance* TOL. In practice, we are also concerned with efficiency, which means in this case, that we want to determine a mesh with the fewest number of elements that yields an approximation with the desired accuracy. We try to reach this optimal mesh by starting with a coarse mesh and successively refining based on the size of the a posteriori error estimate. By starting with a coarse mesh, we try to keep the number of elements as small as possible.

More precisely, we choose an initial mesh $\mathcal{T}_h$, compute the corresponding cG(1) approximation U for our example equation, and then check whether or not

$$C_i\|h\hat{R}(U)\| \leq \text{TOL}.$$

This is the *stopping criterion*, which guarantees that $\|u-U\|_E \leq \text{TOL}$ by the example a posteriori error estimate, and therefore when it is satisfied, U is sufficiently accurate. If the stopping criterion is not satisfied, we try to construct a new mesh $\mathcal{T}_{\tilde{h}}$ of mesh size $\tilde{h}$ with as few elements as possible such that

$$C_i\|\tilde{h}\hat{R}(U)\| = \text{TOL}.$$

This is the *mesh modification criterion* from which the new mesh size $\tilde{h}$ is computed from the residual error $\hat{R}(U)$ of the approximation on the old mesh. In order to minimize the number of mesh points, it turns out that the mesh size should be chosen to equidistribute the residual error in the sense that the contribution from each element to the integral giving the total residual error is roughly the same. In practice, this means that elements with large residual errors are refined, while elements in intervals where the residual error is small are *coarsened* (combined together to form bigger elements).

We repeat the mesh modification followed by solution on the new mesh until the stopping criterion is satisfied. By the a priori error estimate, we know that if u'' is bounded, then the error tends to zero as the mesh is refined. Hence, the stopping criterion will be satisfied eventually.

Theory

Adaptive error control

We formulate the basic goal of adaptive error control as: for a given tolerance

TOL, find a triangulation $\mathcal{T}_h$ that requires the least amount of computational work to achieve

$$\|u - U\|_E \le \text{TOL},$$

where $U \in V_h$ is the finite element approximation corresponding to $\mathcal{T}_h$. Measuring the computational work in terms of the number of nodes of the triangulation $\mathcal{T}_h$ and estimating the unknown error by the computable a posteriori error bound, we are led to the problem of finding the triangulation $\mathcal{T}_h$ with the least number of nodes such that the corresponding finite element approximation U satisfies the stopping criterion

$$C_i \|h\hat{R}(U)\| \le \text{TOL}.$$

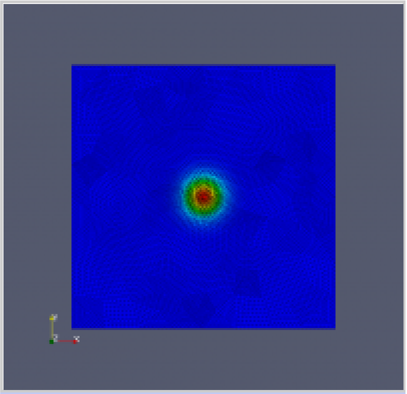
This is a nonlinear constrained minimization problem with U depending on $\mathcal{T}_h$. If the error estimate is a reasonably sharp estimate of the error, then a solution of this optimization problem will meet our original goal.

We cannot expect to be able to solve this minimization problem analytically. Instead, a solution has to be sought by an iterative process in which we start with a coarse initial mesh and then successively modify the mesh by seeking to satisfy the stopping criterion with a minimal number of elements.

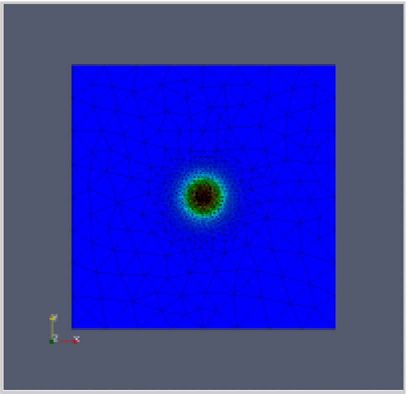
We skip coarsening for now and follow the following simple *adaptive* algorithm *to satisfy the tolerance:*

1. Choose an (arbitrary) initial triangulation T_h^0
2. Given the jth triangulation T_h^j compute FEM solution U
3. Compute residual $\hat{R}(U)$ and evaluate error bound $C\|h\hat{R}(U)\|$
4. If $C\|h\hat{R}(U)\| \leq TOL$ stop, else
5. Refine a percentage of cells K where the error contribution $\int_K (h\hat{R}(U))^2 dx$ is largest.
6. Continue with 2

The rationale is that refining an element with large contribution to the error norm gives a large pay-off in terms of error reduction per new degree of freedom. To achieve an optimal mesh we also need to add *coarsening* (combining several smaller cells to few bigger cells).



Example of uniform mesh refinement



Example of adaptive mesh refinement

Mesh refinement algorithm - Bisection

Step 5 of the adaptive algorithm requires a *refinement* algorithm for splitting mesh cells (we limit the discussion to triangles and tetrahedra). Here refinement means reducing h_K for a cell K.

The following algorithm bisects (splits) the longest edge of a cell, thus replacing the cell with two new cells, and uses successive bisection to eliminate non-conforming cells with *hanging nodes*. A non-conforming cell K_1 has a neighbor cell K_2 that has a vertex on an edge of cell K_1.

Recursive bisection algorithm (Rivara):

```
function bisect($K$):

    Split longest edge $e$
    While $K_i(e)$ is non-conforming
        bisect($K_i$)
```

where $K_i(e)$ is cell incident on edge e.

The same algorithm holds in both 2D/3D (triangles/tetrahedra).

It can be shown that there is a bound on cell quality in 2D (smallest angle) also a bound in 3D with a variation on the edge selection.

Software

Cell-wise integral contribution

We can formulate integration on each cell as testing against a piecewise constant test-function (1 on cell K_i and 0 everywhere else). To compute the error indicator on each cell, one can write something like this:

```
celement = FiniteElement("Discontinuous Lagrange", "triangle", 0)
v = TestFunction(celement)
h = MeshSize("triangle", mesh)
Lei = (h*R)*(h*R)*v*dx
eix = assemble(Lei, mesh)
```

eix is then a vector which holds the error indicators for the cells.

Mesh refinement interface

To mark cells for mesh refinement, we need to construct a *mesh function* on cells:

```
cell_markers = MeshFunction("bool", mesh, mesh.topology().dim())
cell_markers.fill(False)
```

We can iterate over all cells and choose values for the mesh function:

```
for c in cells(mesh):
    if(condition):
        cell_markers.set(c, True)
```

To refine the marked cells, we simply pass the mesh function:

```
mesh.refine(cell_markers)
```

We can save mesh functions for visualization, here is an example how to output the error indicator function as a mesh function on cells:

```
eimf = MeshFunction("real", mesh, mesh.topology().dim())
for c in cells(mesh):
    eimf.set(c, eix[c.index()])
file = File("ei.pvd")
file << eimf
```

Residual computation

A function for computing the "jump" (facet integral) part of the residual is provided: http://www.icarusmath.com/icarus/images/Rjump.py

Usage is like so:

```
from Rjump import *

...

Rj = rjump(U, mesh)

R = Rj + modulus(f)
```

Then assemble an error estimate or indicator based on R.

Postcondition

You should now be familiar with:

- A tolerance on the error ϵ
- The stopping criterion for an adaptive algorithm
- A simple adaptive algorithm based on local refinement
- Local mesh refinement algorithm: the edge bisection algorithm (Rivara)

Exercises

Examination

1.1

Implement the simple adaptive algorithm described above for Poisson's equation and the energy norm. Use the following data as test:

Initial mesh: http://www.icarusmath.com/icarus/images/Square2.xml

$$f(x) = \frac{4}{\pi}\alpha^2(1 - \alpha(x_1^2 + x_2^2))exp(-\alpha(x_1^2 + x_2^2))$$
$$u(x) = \frac{\alpha}{\pi}exp(-\alpha(x_1^2 + x_2^2))$$
$$\alpha = 200$$

Homogenous Dirichlet BC.

Choose a tolerance and compare the performance (number of nodes in the final mesh) for the adaptive algorithm and uniform refinement to satisfy the stopping criterion.

[TODO]

Retrieved from "http://www.icarusmath.com/icarus/index.php?title=Courses/FEM/modules/mesh"

- This page was last modified 09:45, 16 October 2008.