Instructions for my C++ Elliptic PDE Solver Programs using Mixed Methods on General Geometries

Philip T. Keenan

CRPC-TR94393
May, 1994

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Supported in part by a National Science Foundation Post-doctoral Fellowship

Instructions for my C++ Elliptic PDE Solver Programs using Mixed Methods on General Geometries

Philip T. Keenan[*]

April 25, 1994

Contents

1 Introduction

This manual describes the four elliptic solver programs `elliptic_m2d_X`, where X is the name of the numerical method used in that variant. The four programs are otherwise identical and build on a substantial C++ library of tools for general geometry and linear algebra. The four methods are defined in detail in [1]. In particular, X is `saddlepoint` for the MFEM, `hybrid` for the MHFEM, `stencil` for the CCSM, and `enhanced` for the ECCSM. These C++ programs all solve the elliptic partial differential equation

$$-\nabla \cdot (K \nabla p) = f,$$

on some two dimensional polygonal region Ω defined by triangles and rectangles. Here K is a symmetric tensor function of position. The scalar p represents a potential and $u = -K\nabla p$ is called the velocity. If $\hat{n}$ is a unit normal to an edge then $u \cdot \hat{n}$ is called the flux across the edge in the direction of the normal. On each external boundary edge, one of the following two boundary conditions must be supplied: either a scalar boundary condition

[*]Supported in part by a National Science Foundation Postdoctoral Fellowship

$$p = p_0,$$

or a flux boundary condition

$$u \cdot \hat{n} = g_0.$$

The boundary functions p_0 and g_0, the coefficient tensor K, and the source/sink function f must all be specified. The programs also allow "wells" to be specified as delta-function sources and sinks.

The programs use the lowest order Raviart-Thomas spaces on triangles and rectangles to construct a mixed finite element approximation to the PDE. The programs differ in the manner in which they solve the resulting saddle point problem. The saddlepoint variant rewrites it as a positive definite linear operator and applies conjugate gradient. However, evaluation of the operator itself requires a nested application of conjugate gradients, resulting in very slow run times for large problems. The hybrid variant introduces Lagrange multipliers on all mesh edges, yielding a sparse positive definite system, but with substantially more unknowns. The stencil variant uses special quadrature rules to reduce the problem to a sparse positive definite system in the scalar variables, analogous to the finite difference method on rectangles. Finally, the enhanced stencil method combines features of the stencil and hybrid methods to handle certain problems that can arise when using regular, rather than smooth, mesh refinement.

The programs are written in C++ for maximum flexibility. In particular, versions for 3-D domains, parallel architectures and systems of nonlinear time evolution equations are all in preparation, based upon the same libraries.

2 Input Commands

The programs are based on a command interpreter which was built automatically using my C++ code generator tool `cmdGen`, which is further documented in the manual *Instructions for* `cmdGen`.

The programs take an input file which can contain any of the following commands. In addition, the standard commands for command interpreters are accepted. These include commands for online help, include file capability and comments beginning with `#`. See the command interpreter manual for further details on these commands. Also note that this list is subject to change: use the command `help help` to get started with on-line help, which has the completely up-to-date set of commands and options for the version you are running.

2.1 Mesh Description Commands

- `bndy: bndyConds bc` Select the type of boundary condition to impose on subsequent edges. Boundary conditions can be one of
  - `scalar` The scalar solution variable. (Dirichlet)
  - `flux` The normal flux in the solution across a face. (Neumann)
- `prop: double prop` Specify a property value for subsequent edges and elements.

- `v: string name, double x, double y` Define a vertex by its coordinates.
- `e: string name, string id1, string id2` Define an edge by its vertices.
- `tri: string name, string id1, string id2, string id3` Define a triangle by its edges.
- `rect: string name, string id1, string id2, string id3, string id4` Define a rectangle by its edges.
- `setBndy: char* faceName, bndyConds bc, double value` Modify the boundary condition for a given edge.
- `setProp: char* elementName, double propValue` Modify the property value for a given element.

2.2 Compute and Display Commands

- `subdivide: int N` Globally subdivide the mesh N times. This applies regular subdivision to each element.
- `solve:` Solve the partial differential equation on the current mesh.
- `colorRange: ranges r` Select a range for coloring.
  - `auto` The color range is automatically selected to fit the observed range of values.
  - `range min max` The color range is defined to be $[min, max]$. Colors outside this range are shown as black.
- `colorTransform: transforms t` Select a transformation to apply to variables for coloring. Choices include:
  - `identity` Colors are based directly on the value.
  - `absolute-value` Colors are based on the absolute value.
  - `log10-of-absolute-value` Colors are based on the base 10 logarithm of the absolute value.
- `plot: colorVariables var` Append a plot of the specified variable to the plot file. Currently the following variable names may be used:
  - `mesh` This draws just the mesh without reference to the solution.
  - `edgeLines` This draws just the edges of the mesh, in black.
  - `mesh-area` This colors each element based on its area.
  - `mesh-map` This displays the determinant of each element's map to the reference element.
  - `mesh-regularity` For each element, the length of the shortest edge divided by the length of the longest edge.

  - `property` This displays the property code for each element.
  - `boundary-conditions` The boundary conditions on each edge.
  - `sources-sinks` This displays the right hand side of the scalar PDE, ignoring any wells (delta functions) that may also be present.
  - `coef-tensor` The determinant of the coefficient tensor.
  - `scalar` The computed scalar solution.
  - `gradient` The gradient of the computed solution.
  - `flux` The computed flux, converted into a vector velocity at the center of the element.
  - `divergence` The divergence of the computed solution.
  - `reference-solution` The reference scalar solution.
  - `absolute-error` The absolute value of the error in the scalar solution.
  - `relative-error` The relative error (computed-reference)/reference.
  - `reference-gradient` The gradient of the reference solution.
  - `gradient-error` The vector error in the gradient.
  - `gradient-absolute-error` The absolute error in the gradient.
  - `gradient-relative-error` The relative error in the gradient.
  - `reference-flux` The flux from the reference solution.
  - `flux-error` The vector error in the flux.
  - `flux-absolute-error` The absolute error in the flux.
  - `flux-relative-error` The relative error in the flux.
  - `reference-divergence` The divergence of the reference solution.
  - `divergence-absolute-error` The absolute error in the divergence.
  - `divergence-relative-error` The relative error in the divergence.

- `plotCommands: literal {, char* text, literal }` Give low level plot commands.
- `write: colorVariables var` Append transformed values to the log file. This accepts the same list of variables as `plot`.
- `norms: colorVariables var` Compute various error norms for the indicated variable. This accepts the same list of variables as `plot`.
- `label: labels obj` Label the specified objects in the current plot. Objects include:
  - `vertices`
  - `edges`
  - `elements`
  - `wells`
- `vectorScale: double factor` Change the scale factor used in drawing vectors.

- `redirect: outputFiles file, char* newFileName` Redirect any output file. Valid output file names are
  - `plot` The plot file.
  - `log` The log file.

  New output is appended to old when the specified file already exists.

- `refSoln: refSolnKinds kind` Specify a reference solution.
  - `linear`
  - `quadratic`
  - `cubic`
  - `wave`
  - `property`

  The various polynomial reference solutions of total degree $d \in \{1, 2, 3\}$ are given by

  $$\sum_{i+j=0}^{d} c_{ij} x^i y^j,$$

  where the coefficients are specified with the command

  ```
  refSolnParams { c00 c10 c01 c20 c11 c02 ... }
  ```

  The wave solution is

  $$\sin(c_0 \pi x) * \cos(c_1 \pi y),$$

  with

  ```
  refSolnParams { c0 c1 }
  ```

  The `property` based reference solution uses the element and face property values as source/sink terms and boundary values, respectively. It does not attempt to compute an analytic solution (it uses zero instead).

  Users with access to the libraries can extend the list of reference solutions simply by supplying a few functions and relinking the code.

- `refSolnParams: doubleArray params` Specify parameters for the reference solution.

- `tensor: doubleArray tensor` Specify the coefficient tensor. The command

  ```
  tensor { k11 k12 k22 }
  ```

  specifies

  $$K = \begin{pmatrix} k_{11} & k_{12} \\ k_{12} & k_{22} \end{pmatrix}.$$

- `info:` Print information about this program.
- `iterations: double C, double p, double rtol` Iteration parameters for conjugate gradient: allow up to CN^p iterations for N equations, while seeking to reduce the relative error by rtol.
- `well: double x, double y, double flowRate` Add a well at the specified position. The flow rate q is positive for injection wells and negative for production wells. The well acts like q times a delta function in the right hand side of $\nabla \cdot u = f$.

3 Some Mathematical Details

The programs solve the elliptic partial differential equation

$$-\nabla \cdot (K(x)\nabla p(x)) = f(x) \text{ for all } x \in \Omega \subset R^2,$$

with boundary conditions

$$p(x) = p_0(x) \text{ for all } x \in \partial\Omega_D,$$

and

$$-(K(x)\nabla p(x)) \cdot \hat{n}(x) = g_0(x) \text{ for all } x \in \partial\Omega_N,$$

where $\partial\Omega_N = \partial\Omega - \partial\Omega_D$.

Following the standard mixed finite element formulation let

$$u = -K\nabla p,$$

whence

$$\nabla \cdot u = f.$$

Let $(\cdot,\cdot)$ denote the L^2 inner product on Ω, and $<\cdot,\cdot>$ the L^2 inner product on $\partial\Omega$.

Multiplying by suitable test functions, integrating and applying the divergence theorem yields

$$(K^{-1}u, v) = -(\nabla p, v) = (p, \nabla \cdot v) - <p, v>,$$

and

$$(\nabla \cdot u, w) = (f, w).$$

For $u \in H(div)$ and $p \in L^2$, the above equations are equivalent to the original partial differential equation, provided they hold for all $v \in H_0(div)$ and $w \in L^2$. Here $H_0(div) = \{v \in H(div) : v \cdot \hat{n} = 0 \text{ on } \partial\Omega_N\}$. Thus the term $<p, v>$ becomes $<p_0, v>_{\partial\Omega_D}$.

Now let $\{w_j : j \in I_w\}$ be a basis for a finite dimensional subspace of L^2, and $\{v_j : j \in I_v\}$ be a basis for a suitable corresponding finite dimensional subspace of $H(div)$. In these programs the lowest order Raviart-Thomas spaces are used, corresponding to a decomposition of Ω into triangular and rectangular elements, so the w_j are piecewise constants and the

v_j are discontinuous piecewise linear functions with continuous normal components across elements.

Let I_a be the set of indices j for which $v_j \cdot \hat{n} = 0$ on $\partial\Omega_N$, and let $I_b = I_v - I_a$.

We then seek approximate solutions

$$\mathbf{U} = \sum_{j \in I_v} U_j v_j,$$

and

$$\mathbf{P} = \sum_{j \in I_w} P_j w_j.$$

The unknown coefficients U_j and P_j must satisfy

$$\sum_{j \in I_v} U_j(K^{-1} v_j, v_i) - \sum_{j \in I_w} P_j(w_j, \nabla \cdot v_i) = -< p_0, v_i >, \text{ for all } i \in I_a,$$

$$\sum_{j \in I_v} U_j(\nabla \cdot v_j, w_i) = (f, w_i), \text{ for all } i \in I_w,$$

and

$$U_j = g_0(x_j) \text{ for all } j \in I_b,$$

where x_j is the midpoint of the external edge on which $v_j \cdot \hat{n} = 1$.

Let $U = (U_j)^T_{j \in I_v}$ be the vector of unknown flux coefficients, and $P = (P_j)^T_{j \in I_w}$ be the vector of unknown scalar coefficients. Let us use block notation for the I_v index range, writing for example

$$U = \begin{pmatrix} U_a \\ U_b \end{pmatrix}.$$

Then in block matrix form we have

$$\begin{aligned} M_{aa} U_a + M_{ab} U_b - B_a P &= R_a, \\ B_a^T U_a + B_b^T U_b &= R_w, \\ U_b &= G, \end{aligned}$$

where

$$\begin{aligned} M_{ij} &= (K^{-1} v_j, v_i), \\ B_{ij} &= (w_j, \nabla \cdot v_i), \end{aligned}$$

$$R_{ai} = -< p_0, v_i >,$$

and

$$R_{wi} = (f, w_i).$$

This is the symmetric indefinite sparse square linear system which must be solved in the saddlepoint (MFEM) variation. The other variants are similar and are described in greater detail in [1].

We take K, f, p_0 and g_0 to be piecewise constant functions for simplicity in the user interface and in evaluating the above integrals.

4 Running the Programs

4.1 Command Line Arguments

Executing a command like

```
elliptic_m2d_X -usage
```

where X is replaced by for instance `enhanced` will bring up a complete list of the command line options and C-shell environment variables used by the program. In particular, the `-echo` option displays input commands as they are processed, which may help with debugging input files. The standard command line is

```
elliptic_m2d_X inputFileName plotFileName logFileName
```

Using `-` in place of a file name makes the program read from the keyboard or send output to the screen, which also happens if the output files are omitted.

4.2 Sample Input Files

Suppose the file `twistM` contains the following lines:

```
# a pair of triangles stretched along the normal direction

 v a     -1 0
 v b     0 1
 v c     0 -1
 v d     2 1.5

# boundary edges

 e ab     a b
 e ac     a c

 bndy flux

 e db     d b
 e dc     d c

# internal edges

 e bc     b c

 tri t     ab ac bc
 tri tt     db dc bc
```

It defines a domain made from two triangles, as shown in figure 1.

Next, suppose the file `demo` contains the following lines:

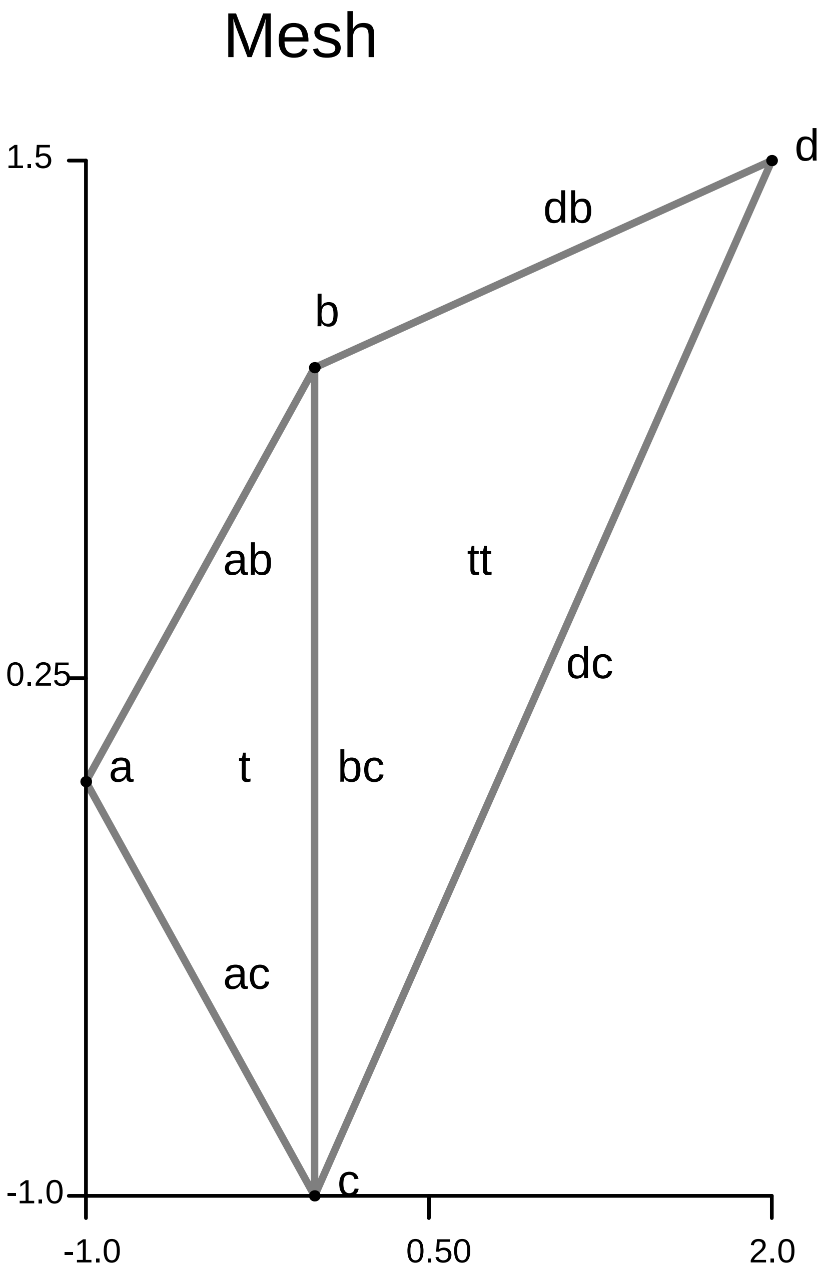

Figure 1: Sample coarse mesh

```
# a sample driver file

include twistM # this reads in the above mesh description

plot mesh
plotCommands { new }

refSoln wave
refSolnParams 2 { 2.5 2.2 }
vectorScale 0.01

plot edgeLines
plot boundary-conditions
plotCommands { new }

iterations 1.1 1 1e-12
subdivide 3
solve

plot scalar
plot edgeLines
plotCommands { new }

plot reference-solution
plot edgeLines
plotCommands { new }

plot absolute-error
plot edgeLines
plotCommands { new }

norms absolute-error
norms gradient-absolute-error
norms flux-absolute-error
norms divergence-absolute-error
```

This subdivides the mesh and solves the PDE using the `wave` test problem. It produces several informative plots as well as norms for the error. If run via a command like

```
elliptic_m2d_enhanced demo demo.plot demo.log
```

the plots will be in the file `demo.plot`, while the norms and other convergence information will be in `demo.log`. The plot file can be displayed with my `plot` program, or converted to PostScript for printing using my `plot2ps` program.

References

[1] Arbogast, T., Dawson, C., and Keenan, P. T., *Mixed Finite Element Methods as Finite Difference Methods for Solving Elliptic Equations on Triangular Elements*, Dept. of Computational and Applied Mathematics Tech. Report #93–53, Rice University, 1993.