Ani2D-FEM Version 1.3

Generator of Finite Elements Matrices on Triangular Meshes

User's Guide

December 1, 2005

1 Introduction

The Fortran package Ani2D-FEM is developed by Konstantin Lipnikov and Yuri Vassilevski. It is designated for generating finite element matrices on triangular meshes. The package allows to build elemental matrices for variety of finite elements, assemble these matrices, and impose boundary conditions. The package differ from other similar packages by providing a very flexible interface for incorporating problem coefficients in elemental matrices.

The library *libfem2D-1.3.a* must be built to incorporate our code into other packages. The package Ani2D-FEM can be used only for generating elemental matrices (no matrix assembling) which increase its usefulness for the user.

This document describes a structure of the package, input data, and user-supplied routines. It presents a few examples illustrating details of the package.

2 Copyright and Usage Restrictions

3 Description of Ani2D-FEM

3.1 Elemental finite element matrix

The core of the package is routine *fem2Dtri* which computes elemental matrix corresponding to the bilinear form

$$< D\, Op_A(u), Op_B(v) > \tag{1}$$

where D is a tensor, Op_A and Op_B are linear operators, and u and v are finite element functions. Here is the list of implemented finite elements (see file `fem2Dtri.f` for more detail):

FEM_P0	piecewise constant, P_0
FEM_P1	continuous piecewise linear, P_1
FEM_P2	continuous piecewise quadratic, P_2
FEM_P1vector	vector continuous piecewise linear, $P_1 \times P_1$. The unknowns are ordered first by vertices and then by the space directions (x and y)
FEM_4P1vector	piecewise linear on the uniform partition onto 4 triangles
FEM_RT0	the lowest order Raviart-Thomas finite elements

Here is the list of available operators (see file `fem2Dtri.f` for more detail):

IDEN	identity operator
GRAD	gradient operator
DIV	divergence operator
CURL	rotor operator
DUDX	partial derivative d/dx

The package allows a few types of tensor D to make computations more efficient. Here is the list of supported tensors.

TENSOR_NULL	identity tensor
TENSOR_SCALAR	scalar tensor
TENSOR_SYMMETRIC	symmetric 2x2 tensor
TENSOR_GENERAL	general 2x2 tensor

The package has a few quadrature formulae:

order = 1	quadrature formula with one center point
order = 2	quadrature formula with 3 points on triangle edges

A solution of non-linear problems is usually based on a Newton-type iterative method. In this case D may depend on a discrete function (e.g. approximation from the previous iterative step). If this is the case, evaluation of D may be quite complex and may require additional data. We provide the user the flexible mechanism for incorporating additional data via external function. Let *Dcoef* be the name of this function. It has the following format

```
      Subroutine Dcoef(x, y, DATA, iSYS, Diff)

C     (x, y)  - [input] Real*8 Cartesian coordinates of a 2D
C               point where tensor Diff should be evaluated
C
C     DATA    - [input] Real*8 user given data (a number or an array)
C
C     iSYS    - [input/output] integer buffer for information exchange:
C                 iD  = iSYS(1)  [output] number of rows in Diff
C                 jD  = iSYS(2)  [output] number of columns in Diff
C                 lbE = iSYS(3)  [input]  label of a mesh element
C                 bc  = iSYS(4)  [output] type of the boundary condition
C                                (not required for assembling routines)
C                                1=BC_DIRICHLET, 2=BC_NEUMANN, 4=BC_ROBIN
C     Diff(iD,jD) - [output] Real*8 matrix, the tensor
```

To compute tensor entries, the user may use information from array *DATA* and the element label *iSYS(3)*. Here are the few examples.

- isotropic diffusion problem. The user has to set $iD = jD = 1$ and return the diffusion value at the point (x, y).
- anisotropic diffusion problem. The user has to set $iD = jD = 2$ and return diffusion tensor (2x2 matrix) at the point (x, y).
- convection problem. The user has to set $iD = 1, jD = 2$ and return the velocity vector value at the point (x, y).

In order to compute a linear form, we can use the following trick:

$$f(v) =< D\, v, FEM\_P0 > \tag{2}$$

where D represents the function f.

Now we are ready to call function *fem2Dtri*.

```
      Call FEM2Dtri(
     &     XY1, XY2, XY3,
     &     OpA, FemA, OpB, FemB,
     &     Dcoef, DATA, iSYS, tensor, order,
     &     LDA, A, nRow, nCol)

C     XYi(2)     - [input] real*8 Cartesian coordinates of i-th vetrex
C     OpA, OpB   - [input] operators from (1), integers
C     FemA, FemB - [input] type of finite elements from (1), integers
C
C     Dcoef      - [input] external function using DATA and iSYS
C     tensor     - [input] type of the tensor, integer
C     order      - [input] accuracy of the numeric quadrature, integer
C
C     LDA        - [input] leading dimension of matrix A(LDA, LDA)
C     A(LDA,LDA) - [output] real*8 finite element matrix A
C     nRow       - [output] the number of rows of A
C     nCol       - [output] the number of columns of A
```

The following rules are applied for numbering unknowns:

- First, basis function associated with vertices are numerated in the same order as the vertices r_i, $i = 1, 2$ and 3.
- Second, basis function associated with edges are numerated in the same order as edges r_{12}, r_{13} and r_{23}.
- The vector basis functions with 2 degrees of freedom per a mesh object (vertex, edge) are enumerated first by the corresponding mesh objects and then by the space coordinates, first x and then y.

3.2 Assembling utilities

The package provides a few utilities for assembling elemental matrices and right hand sides. The assemble routine returns a sparse matrix is the format required by the AMG solver. Other formats will be supported in the nearest future or by request. The converters are in file `algebra.f`. Here is example of calling the assemble routine. We describe only the new parameters.

```
      Subroutine BilinearFormVolume(
     &           nP, nE, XYP, IPE, lbE,
     &           OpA, FemA, OpB, FemB,
     &           Dcoef, DATA, tensor, order,
     &           status, MaxIA, MaxA, IA, JA, DA, A, nRow, nCol,
     &           MaxWi, iW)

C     nP  - [input] the number of points (P)
C     nF  - [input] the number of edges (F)
C     nE  - [input] the number of elements (E)
C
C     XYP(2, nP) - [input] real*8 Cartesian coordinates of mesh points
C     IPF(4, nF) - [input] connectivity list of boundary faces
C     IPE(3, nE) - [input/output] connectivity list of elements.
C                  On output, each column is ordered by increasing.
C
C     lbF(nF)    - [input] boundary identificator
C     lbE(nE)    - [input] element  identificator
C
C     status - some a priori information about the matrix A. The
C              logical sum of constants defined in assemble.fd.
C                   MATRIX_SYMMETRIC - symmetric matrix
C                   MATRIX_GENERAL   - general matrix
C
C                   FORMAT_AMG       - format used in AMG
C                   FORMAT_ROW       - diagonal of A is saved only in DA
C
C     MaxA   - the maximal number of nonzero entries in A
C     IA,JA,DA,A - sparcity structure of matrix A:
C
C           IA(nRow + 1) - IA(k + 1) equals to the number of
C                     nonzero entries in first k rows  plus 1
C           JA(M)    - column indexes of non-zero entries ordered
C                     by rows, M = IA(nRow + 1) - 1
C
C           A(M)     - non-zero entries ordered as in JA
```

```
C           DA(nRow) - main diagonal of A
C
C     nRow - [output] the number of rows in A
C     nCol - [output] the number of columns in A
C
C     MaxWi - the size of the working integer array
C
C     iW(MaxWi) - the integer working array.
```

Here is an example of assembling the right-hand side vector `F(nRow)` for the linear form (2).

```
      Subroutine LinearFormVolume(
     &           nP, nE, XYP, IPE, lbE,
     &           FemA,
     &           Dcoef, DATA, order,
     &           F, nRow,
     &           MaxWi, iW)
```

4 Examples

4.1 Elliptic problems

The package Demo2 (type `make demo2` in the root directory or `make exe run gs` in directory `src/Demo2`) demonstrates the iterative adaptive solution of the boundary value problem:

$$\begin{aligned} -\mathrm{div}(D\,\mathrm{grad}u) = 1 \quad &\text{in} \quad \Omega, \\ u = 0 \quad &\text{on} \quad \partial\Omega_D, \\ \tfrac{\partial u}{\partial n} = 0 \quad &\text{on} \quad \partial\Omega_N, \end{aligned}$$

where Ω is the segment of the unit disk,

$$\Omega = \{(x,y):\ x^2+y^2<1,\ x<0 \text{ or } y>0\}.$$

The boundary of Ω consists of two pieces $\partial\Omega_D$ and $\partial\Omega_N$ where

$$\partial\Omega_N = \{(x,y):\ x=0,\ -1<y<0\}.$$

The diffusion coefficient D is the diagonal piecewise constant tensor given by

$$\begin{aligned} D = \mathrm{diag}\{10,10\} \quad &\text{for} \quad x<0, \\ D = \mathrm{diag}\{1,100\} \quad &\text{for} \quad x>0. \end{aligned}$$

The package Demo2 generates an adaptive mesh and saves it in file `hba.ps`.