

Ani2D Version 2.0

**Ani2D-FEM: Flexible Generator of
Finite Elements Systems
on Triangular Meshes**

**User's Guide for libfem2D-2.0.a
February 10, 2007**

1 Introduction

The Fortran package Ani2D-FEM is developed by Konstantin Lipnikov and Yuri Vasilevski. It is designated for generating finite element matrices on triangular meshes. The package allows to build elemental matrices for variety of finite elements, modify these matrices, assemble them, and impose boundary conditions.

The package Ani2D-FEM differs from other similar packages by providing a very flexible interface for incorporating problem coefficients in elemental matrices. In addition, the elemental matrices are understood in a very broad sense. They may involve different types of finite elements.

The library *libfem2D-2.0.a* can be incorporated into other packages.

This document describes the 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

The software is made available for nonprofit use only. You may copy and use this software without any charge, provided that the COPYRIGHT file is attached to all copies. For all other uses please contact one of the authors.

The software is made available “as is” without any assurance that it will work for your purposes. The authors are not responsible for any damages caused by using this software.

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

$$\langle D Op_A(u), Op_B(v) \rangle \quad (1)$$

where D is a tensor, Op_A and Op_B are linear first-order or zero-order differential operators, and u and v are finite element basis 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_P2vector | vector continuous piecewise quadratic, $P_2 \times P_2$. The unknowns are ordered first by vertices, then by edges, and then by the space directions (x and y) |
| 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 uses several quadrature formulae:

| | |
|-----------|--|
| order = 1 | quadrature formula with one center point |
| order = 2 | quadrature formula with 3 points on triangle edges |
| order = 5 | quadrature formula with 7 points inside triangle |

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

```

Integer Function Dcoef(x, y, label, DATA, iSYS, Coef)

C   The function returns type of the tensor Coef (see the table above).
C
C   (x, y) - [input] Real*8 Cartesian coordinates of a 2D
C           point where tensor Coef should be evaluated
C
C   label  - [input] Integer label of a mesh element
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           iSYS(1) [input] triangle number
C           iSYS(2) [input] 1st vertex number
C           iSYS(3) [input] 2nd vertex number
C           iSYS(4) [input] 3rd vertex number
C
C           iSYS(1) = iD [output] number of rows in Coef
C           iSYS(2) = jD [output] number of columns in Coef
C
C   Coef(4,jD) - [output] Real*8 matrix with the leading dimension 4

```

To compute entries of the tensor `Coef`, the user may use the triangle number `iSYS(1)` and array `DATA`. Here are a few examples.

- isotropic diffusion coefficient. The user has to set `iD = jD = 1`, `Dcoef = TENSOR_SCALAR` and to return the diffusion value `Coef(1,1)` at the point (x, y) .
- anisotropic diffusion coefficient. The user has to set `iD = jD = 2`, `Dcoef = TENSOR_SYMMETRIC`, and to return diffusion tensor (2x2 matrix with entries `Coef(1,1)`, `Coef(1,2)`, `Coef(2,1)`, `Coef(2,2)`) at the point (x, y) .
- convection coefficient. The user has to set `iD = 1`, `jD = 2`, `Dcoef = TENSOR_GENERAL`, and to return the velocity transposed vector values `Coef(1,1)`, `Coef(1,2)` at the point (x, y) .

Now we are ready to call routine *fem2Dtri* which computes elemental matrix A:

```

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

C      XYi(2)      - [input] Real*8 Cartesian coordinates of i-th vertex
C      OpA, OpB    - [input] operators in (1), integers
C      FemA, FemB - [input] type of finite elements from (1), integers
C
C      Dcoef       - [input] external integer function using label and DATA
C      order       - [input] order 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 within the elemental matrix:

- First, basis functions associated with vertices (if any) are numerated in the same order as the vertices r_i , $i = 1, 2, 3$ (input parameters XY1, XY2, XY3).
- Second, basis functions associated with edges (if any) are numerated in the order of edges r_{12} , r_{23} and r_{13} .
- Third, basis functions associated with element (if any) are numbered.
- 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 .

In order to compute a linear form representing an elemental right hand side, we can use the following trick:

$$f(v) = \langle D_{rhs} FEM_P0, v \rangle \quad (2)$$

where D_{rhs} represents the right hand side function f :

```

      Call FEM2Dtri(
&      XY1, XY2, XY3,
&      IDEN, FEM_P0, IDEN, FemB,
&      label, Drhs, DATA, iSYS, order,
&      LDA, F, nRow, nCol)

```

3.2 Extended elemental finite element matrix

Now we describe an alternative way to create and assemble elemental matrices. Each elemental matrix may be a combination of a few *fem2Dtri* calls reflecting the fact that the bilinear form (1) may consist of a few simple forms. One of the examples is the Stokes problem. degrees of freedom in the extended elemental matrix are characterized by arrays *templateR* and *templateC*:

```
Subroutine FEM2Dext(  
&      XY1, XY2, XY3,  
&      lbE, lbF, lbP, DATA, iSYS,  
&      LDA, A, F, nRow, nCol,  
&      templateR, templateC)  
  
C      XYi(2) - [input] Real*8 Cartesian coordinates of i-th point  
C  
C      lbE    - [input] Integer ID of the triangle (material label)  
C              inherited from global material labels  
C      lbF(3) - [input] Integer IDs of triangle edges (boundary labels)  
C              lbF(i) = 0 for internal edges, otherwise it is the copy  
C              of IFP(4,k) where k is the global boundary edge  
C      lbP(3) - [input] Integer IDs of triangle nodes inherited  
C              from global nodal labels  
C  
C      DATA  - [input] Real*8 user given data (a number or an array)  
C      iSYS   - [input] integer buffer for providing triangle information:  
C              iSYS(1)  triangle number  
C              iSYS(2)  1st vertex number  
C              iSYS(3)  2nd vertex number  
C              iSYS(4)  3rd vertex number  
C  
C      LDA    - [input] leading dimension of matrix A  
C      A(LDA, *) - [output] Real*8 elemental matrix, degrees of freedom  
C              are ordered according to templateR and templateC  
C      F(nRow) - [input] Real*8 vector of the right-hand side  
C  
C      nRow   - [output] the number of rows in A  
C      nCol   - [output] the number of columns in A  
C  
C      templateR(nRow) - [output] Integer array of degrees of freedom  
C              for rows. We recommend to group them, e.g.  
C              three for points, three for edges, etc.  
C      templateC(nCol) - [output] Integer array of degrees of freedom  
C              for columns.
```

In general, different order of unknowns is allowed. However, in the assembled matrix, they will be grouped according to their geometric location. For instance, the first three unknowns associated with points will go to the first group of point-based unknowns. Next three point-based unknowns will go to the second group. After point-based unknowns we group the edge-based unknowns. The element-based unknowns are grouped the last.

Here are a few examples where this routine may be useful.

- For the diffusion reaction equation we sum elemental matrices corresponding to diffusion and reaction.
- For the diffusion equation written in a mixed form using Lagrange multipliers, we use hybridization algorithm inside `FEM2Dext`.
- We may also incorporate boundary conditions in the elemental matrix `A`.

3.3 Assembling utilities

The package provides a few utilities for assembling elemental matrices and right hand sides. The assemble routine returns a sparse matrix in the format required by the AMG solver (CSR format with diagonal entries in the beginning of each row). Other formats will be supported in the nearest future or by request (see converters in file `algebra.f`). Here is the header of the assembling routine. We describe only the new parameters.

```

Subroutine BilinearFormVolume(
&      nP, nE, XYP, IPE, lbE,
&      OpA, FemA, OpB, FemB,
&      Dcoef, DATA, order,
&      assembleStatus, 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 (see
C                  documentation for package Ani2D-MBA)
C      IPE(3, nE) - [input/output] connectivity list of elements.
C                  On output, the nodal indexes in each triangle are reordered
C                  by index increasing.
C
C      lbF(nF)    - [input] boundary label
C      lbE(nE)    - [input] element label

```

```

C
C   assembleStatus - [input] a priory information about matrix A.
C           The 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 (CSR with
C                       rows starting by diagonal entry)
C           FORMAT_ROW - diagonal of A is saved only in DA
C
C   MaxIA - [input] the maximal number of equations plus one
C   MaxA  - [input] the maximal number of nonzero entries in A
C   IA,JA,DA,A - [output] sparsity structure of matrix A:
C
C           IA(nRow+1)- number 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 - [input] size of the working integer array
C
C   iW(MaxWi) - 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,
&      Drhs, DATA, order,
&      F, nRow,
&      MaxWi, iW)

```

The following assembling routine must be used for the extended elemental matrices described in Section 3.2. All but one parameters were described above. The new parameter `lbP` is optional labels of mesh points. They may be useful for assigning Dirichlet boundary conditions.

```

Subroutine BilinearFormTemplate(
&          nP, nF, nE, XYP, lbP, IPF, IPE, lbE,
&          FEM2Dext, DATA, assembleStatus,
&          MaxIA, MaxA, IA, JA, A, F, nRow, nCol,
&          MaxWi, iW)

```

The matrix A is in one of the sparse row formats. The unknowns are ordered in groups as explained in Section 3.2.

4 Examples

4.1 Diffusion problem

The program `aniFEM/mainSimple.f` demonstrates the approximate solution of the boundary value problem with continuous piecewise linear finite elements P_1 :

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

where $\Omega = (0, 1)^2$, $\partial\Omega_N = \{(x, y) : x = 1, 0 < y < 1\}$, $\partial\Omega_D = \partial\Omega \setminus \partial\Omega_N$. The diffusion coefficient D is the diagonal piecewise constant tensor given by

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

First, the program refines an initial coarse triangulation consisting of two triangles. and creates a quasi-uniform mesh with 4000 elements. This is accomplished with routine `mesh_metric` from the library *libmba2D-2.0.a*. Second, the program generates the finite element system using the routines `BilinearFormVolume`, `LinearFormVolume` and `BoundaryConditions` from the library *libfem2D-2.0.a*.

4.2 Stokes problem

The program `aniFEM/mainTemplate.f` demonstrates the approximate solution of the Stokes problem with $P_2 \times P_1$ pair of finite elements:

$$\begin{aligned}
-\operatorname{div} \operatorname{grad} \mathbf{u} + \nabla p &= 0 && \text{in } \Omega, \\
-\operatorname{div} \mathbf{u} &= 0 && \text{in } \Omega, \\
\mathbf{u} &= \mathbf{u}_0 && \text{on } \partial\Omega_1, \\
\mathbf{u} &= 0 && \text{on } \partial\Omega_2, \\
\frac{\partial \mathbf{u}}{\partial n} - p &= 0 && \text{on } \partial\Omega_3,
\end{aligned}$$

where $\Omega = (0, 1)^2$, $\partial\Omega_1 = \{(x, y) : x = 0, 0 < y < 1\}$, $\partial\Omega_3 = \{(x, y) : x = 1, 0 < y < 1\}$, $\partial\Omega_2 = \partial\Omega \setminus (\partial\Omega_1 \cup \partial\Omega_3)$, and $\mathbf{u}_0 = (4y(1 - y), 0)^T$.

First, the program refines an initial coarse triangulation (consisting of two triangles) and creates a quasi-uniform mesh with 4000 elements. This is accomplished with routine `mesh_metric` from the library *libmba2D-2.0.a*. Second, the program generates the finite element system using routine `BilinearFormTemplate` from the library *libfem2D-2.0.a*.