

Ani3D

1997-2008

The package ani3D is designated for generating unstructured tetrahedral meshes, adapting them isotropically and anisotropically, discretizing systems of PDEs, solving linear systems, visualizing meshes and associated solutions. It is a set of independent libraries with different tasks. All libraries may be combined in a single program. Extensive tutorials represent powerful capabilities of the package.

The package ani3D was developed by a team of researchers headed by the two principle investigators:

- Konstantin Lipnikov¹
- Yuri Vassilevski².

Ideas and technologies, as well as packages ani3D-MBA , ani3D-FEM , and ani3D-VIEW have been developed by the principal investigators.

The package ani3D-AFT was developed by students of the Moscow State University:

- Alexander Danilov
- Kirill Nikitin
- Anatoly Vershinin
- Andrey Plenkin

under the supervision of the principal investigators.

The package ani3D-RCB was developed by Vadim Chugunov² and Yuri Vassilevski². The package ani3D-ILU was developed by

- Sergei Goreinov²
- Vadim Chugunov²
- Yuri Vassilevski².

Besides the original software, the package ani3D incorporates a number of public libraries such as BLAS, LAPACK, UMFPACK, AMD, GLUT, OPENCASCADE, and CGM.

The authors are grateful to Rao Garimella for mentoring Alexander Danilov and helping him with the CGM and OPENCASCADE packages.

¹Los Alamos National Laboratory Theoretical Division, MS-284 Los Alamos, NM 87545, USA.

²Institute of Numerical Mathematics RAS 8 Gubkina St, 119333 Moscow, RUSSIA.

Contents

Package ani3D-AFT	4
Package ani3D-RCB	11
Package ani3D-MBA	16
Package ani3D-FEM	26
Package ani3D-ILU	34

Ani3D-AFT version 2.1 ”*Forget-me-not*”

**Flexible Tetrahedral Mesh Generator
Using Advanced Front Technique**

**User’s Guide for libaft3D-2.1.a,
libfrtprm-2.1.a, libfrtmdf-2.1.a,
libfrtscg-2.1.a, libfrtcad-2.1.a**

1 Introduction

The C package ani3D-AFT is an independent part of the package ani3D . ani3D-AFT was developed by Alexander Danilov, Kirill Nikitin, Anatoly Vershinin and Andrey Plenkin under the supervision of Yuri Vassilevski and Konstantin Lipnikov. It is designated for generating tetrahedral meshes in arbitrary 3D domains. The libraries of the package ani3D-AFT are split into two subsets by their objectiveis: to generate an initial front as a boundary discretization (*libfrtprm-2.1.a*, *libfrtmdf-2.1.a*, *libfrtscg-2.1.a*, *libfrtcad-2.1.a*) and to generate a tetrahedrization with the given boundary trace (*libaft3D-2.1.a*).

The libraries 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

This software is release under the GNU LGPL Licence. You may copy and use this software without any charge, provided that the COPYRIGHT and LICENSE files are attached to all copies.

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

3 Description of Ani3D-AFT

The basic features of package ani3D-AFT are listed below.

Domain type : single-/multi-component and simply-/ multi-connected bounded domains

Boundary type : piecewise smooth or polyhedral (piecewise linear)

Domain data input : set of 2D patches representing the boundary, or the front (triangulation of the boundary)

Number of mesh elements : non-limited

Data format : double precision and integer arrays. Enumeration starts with 1 (default).

The library *libaft3D-2.1.a* uses the Advanced Front Technique for 3D meshing. The technique presumes that an initial front defining the boundary is given. The initial front is a surface shape-regular triangulation with a prescribed orientation of normals for all triangles. The orientation defines the direction of the front advancing. The initial front may be composed of multiple components. The local mesh size may be controlled by the user defined function **fsize**.

The libraries *libfrtprm-2.1.a*, *libfrtmdf-2.1.a*, *libfrtscg-2.1.a*, *libfrtcad-2.1.a* use different technologies for the generation of the initial front. These technologies are listed below:

1. Representation of the boundary as a union of smooth parameterized patches with explicit parameterizations of their boundaries (if patches are flat polygons then parameterization of their boundaries is not needed). An automated triangulation of the boundary yields the initial front. The algorithms are in the library *libfrtprm-2.1.a*.
2. Representation of the initial front as unions and intersections of prescribed surface meshes of the given primitives. The primitives may be defined by the user via templated routines. The surface meshes are produced by the automated surface triangulation using algorithms from library *libfrtscg-2.1.a*.
3. Representation of the boundary as a conformal triangulation approximating the actual boundary with a given accuracy (a CAD mesh). The triangles (facets) may be anisotropic or severely distorted. An automated remeshing of the CAD mesh such that the new mesh nodes belong to the CAD surface, yields the initial front. The algorithms are in the library *libfrtmdf-2.1.a*.
4. Representation of the boundary using a public CAD system (OpenCASCADE) and the public interface library CGM. The algorithms in library *libfrtcad-2.1.a* provides the interface to technology 1.

The package is designed so that no graphical interface is needed for the front and 3D mesh generation. The mesh and the front may be saved in files with the GMV-format (`write_mesh_gmv`) and then visualized and analyzed with the GMV package.

However, for user convenience, one graphical user interface (GUIs) is added for the second tool (`src/aniAFT/src/aniFRT/GUI/SCG`). It visualizes operations with primitives using the *freelut* package. An example of using the GUI is given in `src/aniAFT/src/aniFRT/GUI/SCG/gui_glut.cpp`. A description of GUI parameters is given in `src/aniAFT/src/aniFRT/GUI/SCG/README`.

If the user installed OpenCASCADE library, he can use its GUI executable `DRAWEXE`. A description of GUI parameters is given in `src/aniAFT/src/aniFRT/GUI/CAD/README`.

4 Generation of the initial front

4.1 Boundary as a union of smooth parameterized patches (library *libfrtprm-2.1.a*)

The library contains three routines which may be used for the initial front generation:

```
set_param_functions(surface_param, v_u_param);
set_surface_size_function(fsize);
i = surface_boundary_( ... );
```

Routine `set_param_functions` adopts the user given parameterizations of the patches `surface_param()`, and their boundaries `v_u_param()`. Routine `set_surface_size_function` adopts the user given mesh size function `fsize()`. Routine `surface_boundary_` produces

a shape regular triangulation of the boundary with a mesh size controlled by `fsize()`. An example using these routines is given in file `src/Tutorials/PackageAFT/main_prm.c`.

If the boundary is a union of flat polygons, the user is advised to design a wrapper `make_front` to `surface_boundary_` which uses the polygons data. An example of the wrapper is given in file `src/Tutorials/PackageAFT/main_prm_flat.c`. The user is advised to replace `main_prm.c` with `main_prm_flat.c` and recompile the library:

```
set_surface_size_function(fsize);
make_front(&nV, vertex, nL, nE, edge, c1, c2,
           &nF, face, facematerial, nnV, nnF);
```

4.2 Boundary as a combination of given front primitives (library `libfrtscg-2.1.a`)

The library contains routines which operate with initial fronts. They generate surface meshes for primitives (parallelepiped, sphere or cylinder), move, rotate and scale them, and generates a surface mesh for the intersection/union/difference of the primitives.

```
/* Surface mesh generation for the sphere with radius 1.45;
   mesh step is 0.3 */
scg_make_sphere (&nV_in1, &nF_in1, Vertex_in1, Index_in1, 1.45, 0.3);
```

```
/* Surface mesh generation for the cylinder with radius 0.5,
   height 5; mesh step is 0.3 */
scg_make_cylinder (&nV_in2, &nF_in2, Vertex_in2, Index_in2, 0.5, 5, 0.3);
```

```
/* Translation of an object along vector (0.1, 0.5, -2.3) */
scg_translate (nV_in2, Vertex_in2, 0.15, 0.5, -2.3);
```

```
/* Rotation of an object by angles pitch=0.2, yaw=0.1 and roll=0 */
scg_rotate (nV_in2, Vertex_in2, 0.2, 0.1, 0);
```

```
/* Intersection of meshes for two objects: the last integer
   parameter is an operation:
   0 - union (in1 + in2)
   1 - difference (in1 - in2)
   2 - another difference (in2 - in1)
   3 - intersection (in1 * in2) */
scg_intersect_mesh (nV_in1, nF_in1, Vertex_in1, Index_in1,
                   nV_in2, nF_in2, Vertex_in2, Index_in2,
                   &nV_out, &nF_out, Vertex_out, Index_out, 2);
```

An example of using these routines is in file `src/Tutorials/PackageAFT/main_scg.cpp`.

The user can visualize and control the process of meshing. To this end, a Graphical User Interface is provided in `src/aniAFT/src/aniFRT/GUI/SCG`. It is based on library `libfrtscg-2.1.a` and the public library `glut` located in `src/glut`. The user can operate with the primitives using keyboard and input string and load and save the mesh front. An example of use of the GUI is given in file `src/aniAFT/src/aniFRT/GUI/SCG/gui_glut.cpp`. In order to create a 0.3-mesh for sphere with radius 1.6, centered at (0, 0, 1) and rotated by angles 0, 0, 0 and a 0.3-mesh for cylinder with radius 0.6, height 5, centered at (0, 0, -1) and rotated by angles 0.3, 0 and 0.5, type

```
bin/gui_glut.exe +s 1.6 0 0 1 0 0 0 0.3 +c 0.6 5 0 0 -1 0.3 0 0.5 0.3
```

Then use the keyboard to generate the surface mesh of a combination of the two primitives.

4.3 Boundary as a CAD mesh with triangular facets (library `libfrtmdf-2.1.a`)

The library contains three routines which convert the input CAD mesh to a shape-regular mesh:

```
check_surface_topology_fix_(&nV, vertex, &nF, face);
surface_refine_setup_poly(0.10, 0.10, 1e-6, 1e-6);
surface_refine_(&nV, vertex, &nF, face, facematerial, &nV, &nF);
```

Routine `check_surface_topology_fix_` checks the topology of the CAD mesh and merges possible duplicate vertices. Routine `surface_refine_setup_poly` sets control parameters for `surface_refine`. Their meaning is described in `src/Tutorials/PackageAFT/main_mdf.c`. Routine `surface_refine_` refines the CAD mesh by generating a shape regular triangulation which approximates the boundary with the same accuracy as the CAD mesh. The nodes of the shape-regular triangulation are on the CAD mesh.

An example using these routines is in file `src/Tutorials/PackageAFT/main_mdf.c`.

4.4 Boundary representation through a CAD system (library `libfrtcad-2.1.a`)

The library generates the initial front using the public CAD system OPENCASCADE (<http://www.opencascade.org>) through the interface library CGM. The library may be used only if Open CASCADE is properly installed before the compilation of Ani3D-AFT. The interface between AFT kernel and CAD system is based on CGM (Common Geometry Module), the interface library. Original version of CGM can be found at <http://cubit.sandia.gov/cgm.html>. ani3D-AFT uses a modified version of CGM with

basic Open CASCADE support. This version is located in `src/cgm`. It was tested with Open CASCADE versions 6.1 and 6.2.

In order to compile CGM and use Open CASCADE support in `ani3D-AFT`, environment variable `$CASROOT` should be defined. It should point to the path where Open CASCADE is installed. CGM package expects include files to be in `$CASROOT/inc`, and libraries in `$CASROOT/Linux/lib`. If `$CASROOT` variable is unset, CGM will not be compiled, and Open CASCADE support will be disabled. If `$CASROOT` variable is set, CGM will be compiled, and library `libfrtcad-$(version).a` (source code is in `src/aniAFT/src/aniFRT/CAD`) will be built.

Routine `CGM_Init` initializes the CGM interface. Routine `CGM_LoadModelFromFile` loads the CAD input data (*.brep) and results in OPENCASCADE object 'model'. Routine `surface_CGM_model_(model, ...)` produces a shape regular triangulation of the boundary of 'model'.

```
// initialize CGM
CGM_Init();

// load CGM model from file
model = CGM_LoadModelFromFile(argv[1]);

// allocate memory
vertex      = (double*)malloc(sizeof(double) * 3 * nnV);
face        = (int*)   malloc(sizeof(int)   * 3 * nnF);
facedup     = (int*)   malloc(sizeof(int)   * 3 * nnF);
facematerial = (int*)   malloc(sizeof(int)   * nnF);
facematdup  = (int*)   malloc(sizeof(int)   * nnF);
tetra       = (int*)   malloc(sizeof(int)   * 4 * nnT);
tetramaterial = (int*)   malloc(sizeof(int)   * nnT);

// this is the relative mesh size for the front
surfacesizeratio = 0.1;

// create initial front
surface_CGM_model_(model, &nnV, vertex, &nnF, face, facematerial, &nnV, &nnF);
```

An example is given in file `src/Tutorials/PackageAFT/main.cad.c`.

There exists the GUI for the OPENCASCADE library. The executable `bin/DRAWEXE` allows the user to visualize the process of production of the CAD input data (*.brep). A brief tutorial for this GUI may be found in `src/aniAFT/src/aniFRT/GUI/CAD/README`.

5 Generation of a tetrahedral mesh

After an initial front is defined, the user may call one of two 3D meshing routines

```
i = mesh_3d_aft_func_( ..., fsize );
i = mesh_3d_aft_cf_( ..., &cf );
```

Routine `mesh_3d_aft_func_` generates a shape regular tetrahedrization with a local mesh size controlled by the user defined function `fsize`. Routine `mesh_3d_aft_cf_` generates a shape regular tetrahedrization which is coarsened towards the domain interior with the geometric factor `cf`. The second routine is useful when the local mesh size is unknown, (e.g., if the input is a CAD mesh). Both routines return the error code: zero value means the mesh is generated successfully, other values imply generation failure.

Examples using these routines are given in files `src/Tutorials/PackageAFT/main_aft.c`, `src/Tutorials/PackageAFT/main_prm.c`, `src/Tutorials/PackageAFT/main_mdf.c`, and `src/Tutorials/PackageAFT/main_scg.cpp`.

6 Output

Two library routines allow to save mesh:

```
write_mesh    ("mesh.out", ... );
write_mesh_gmv("mesh.gmv", ... );
```

Routine `write_mesh_gmv` generates the GMV-file `mesh.gmv`. The General Mesh Viewer (GMV) is a public visualizing tool available at www-xdiv.lanl.gov/XCM/gmv/GMVHome.html.

Routine `write_mesh` save the mesh in file `mesh.out` using a simple format:

```
nV
x_1 y_1 z_1
...
x_nV y_nV z_nV
nT
i1_1 i2_1 i3_1 i4_1 tetramaterial_1
...
i1_nT i2_nT i3_nT i4_nT tetramaterial_nT
nF
j1_1 j2_1 j3_1 facematerial_1
...
j1_nF j2_nF j3_nF facematerial_nF
```

ani3D-RCB version 2.1 “*Windflower*”

**Flexible Mesh Refining/Coarsening Tool
Using Marked Edge Bisection**

User’s Guide for librcb3D-2.1.a

1 Basic features of the library

The FORTRAN77 package ani3D-RCB is a part of the package ani3D . ani3D-RCB was developed by Vadim Chugunov and Yuri Vassilevski. It is designated for hierarchical refining and coarsening of arbitrary tetrahedral meshes. Basic restriction: prior coarsening the mesh must be refined; no coarsening is applied to an unrefined mesh.

The library contains an initialization tool, a refinement tool, and a coarsening tool.

Mesh data

The mesh output is produced in place of the mesh input. A mesh is represented by the following data. The number of mesh nodes is `nv`, their Cartesian coordinates are stored in the array `vrt(3,*)`. The number of mesh tetrahedra is `nt`, the connectivity list of tetrahedra is stored in the array `tet(4,*)`, the tetrahedron materials (labels) are `material(*)`. The number of mesh boundary faces is `nb`. The columns of `bnd(3,*)` are node indexes of the boundary faces. Face material/label is stored in `labelF(*)`.

2 Initialization

The initialization tool (`aux.f`) prepares auxiliary structure which defines how to bisect the tetrahedra. In `InitializationRCB` all input tetrahedra are marked for bisection according to specific rule. Also, auxiliary data structure is generated in this routine. In actual refinement, the user is free to mark for refinement any subset of tetrahedra.

```
iERR      = 0
call InitializeRCB (nt, ntmax, nv, nvmax, vrt, tet,
&      MaxWi, iW, listtet, tetpmax, iERR)
If(iERR.GT.0) Call errMes(iERR, 'main', 'error in InitializeRCB')
```

The size of work memory `iW` for `InitializeRCB`, `LocalRefine`, `LocalCoarse` should be at least $15*ntmax+nvmax+41$. The size of data array `liststet` is $(tetpmax+1)*ntmax$ where `tetpmax=50`.

3 Refinement

The refinement tool `LocalRefine` (`refine.f`) refines the input tetrahedrization according to the user defined routine `RefineRule`. The name of the latter routine is the input parameter of `LocalRefine`. The output tetrahedrization is in place of the input tetrahedrization. The by-product of `LocalRefine` is the logical data array `history(4,maxlevel,ntmax)`. It will be used in later coarsening. The input current index of the refinement level `ilevel` is passed to `RefineRule`.

```
c ... user defined procedures
external RefineRule
```

```

...
nlevel = 5
Do ilevel = 1, nlevel
  call LocalRefine (
&      nv, nvmax, nb, nbmax, nt, ntmax,
&      vrt, tet, bnd, material,labelF,
&      RefineRule, ilevel,
&      maxlevel, history,
&      listtet, tetpmax, RefineRuleData,
&      MaxWi, iW,
&      iPrint, iERR)
      If(iERR.GT.0) Call errMes(iERR, 'main',
&          'error in LocalRefine')
  End do

```

The key control of the refinement process is the user defined routine `RefineRule`. Here, the user defines which tetrahedra have to be refined and how they must be refined, depending on each tetrahedra data and the current level of refinement. The array `RefineRuleData` is served for passing external data to the routine. The control for refinement is the marker `verf(i)`, where `i` runs from 1 to `nt`. If the marker is 0, then there is no need to refine tetrahedron `i`; if the marker is 1, then the user wants to refine tetrahedron by single bisection; if the marker is 2, then the user wants to refine tetrahedron by two levels of bisection; if the marker is 3, then the user wants to refine tetrahedron by three levels of bisection into eight similar subtetrahedra.

```

Subroutine RefineRule (nt, tet, vrt, verf, ilevel, RefineRuleData)

```

```

...
If (ilevel .le. 3) Then
  Do i = 1, nt
    verf(i) = 1 ! one level of bisection
  End do
Else ! refine towards the plane y=0
  Do i = 1, nt
    xy = vrt(3,tet(1,i))* vrt(3,tet(2,i))*
&      vrt(3,tet(3,i))* vrt(3,tet(4,i))

    If (xy .eq. 0) Then
      verf(i) = 3 ! three levels of bisection (keep the shape)
    Else
      verf(i) = 0 ! no need to refine
    End if
  End do
End if
End

```

4 Coarsening

The coarsening tool `LocalCoarse` (`coarse.f`) coarses the input tetrahedrization according to the user defined routine `CoarseRule`. The name of the latter routine is the input parameter of `LocalCoarse`. The output tetrahedrization is in place of the input tetrahedrization. The array `CoarseRuleData` is served for passing external data to the routine. The by-product of `LocalCoarse` is the logical data array `history(4,maxlevel,ntmax)`. It will be used in later coarsening/refinement. The input current index of the refinement level `ilevel` is passed to `CoarseRule`.

```
c ... user defined procedures
  external CoarseRule
  ...
  nlevel = 5
  Do ilevel = nlevel, 1, -1
    call LocalCoarse (
&      nv, nvmax, nb, nbmax, nt, ntmax,
&      vrt, tet, bnd, material,labelF,
&      CoarseRule, ilevel,
&      maxlevel, history,
&      listtet, tetpmax, CoarseRuleData,
&      MaxWi, iW,
&      iPrint, iERR)
      If(iERR.GT.0) Call errMes(iERR, 'main',
&      'error in LocalCoarse')
  End do
```

The key control of the coarsening process is the user defined routine `CoarseRule`. Here, the user defines which tetrahedra have to be merged and how they must be merged, depending on each tetrahedron data and the current level of coarsening. The control for coarsening is the marker `verf(i)`, where `i` runs from 1 to `nt`. If the marker is 0, then there is no need to coarsen tetrahedron `i`; if the marker is 1, then the user wants to merge tetrahedron with its neighbor; if the marker is 2, then the user wants to merge tetrahedron with its neighbor and then merge the result one more time; if the marker is 3, then the user wants to merge tetrahedron with its neighbor and then merge the result to more times so that the result be similar to tetrahedron `i`.

```
  Subroutine CoarseRule (nt, tet, vrt, verf, ilevel, CoarseRuleData)
  ...
c Coarsening conjugate to refinement rule
  If (ilevel .le. 3) Then
    Do i = 1, nt
      verf(i) = 1 ! one level of merging
    End do
  Else ! coarse tets touching plane y=0
    Do i = 1, nt
```

```
xy = vrt(3,tet(1,i))* vrt(3,tet(2,i))*  
& vrt(3,tet(3,i))* vrt(3,tet(4,i))  
  
If (xy .eq. 0) Then  
  verf(i) = 3 ! three levels of merging (keep the shape)  
else  
  verf(i) = 0 ! no need to coarse  
End if  
End do  
End if  
End
```

Ani3D-MBA version 2.1 ”*Stone Flower*”

**Flexible Mesh Generator Using
Metric Based Adaptation**

User’s Guide for libmba3D-2.1.a

1 Introduction

The Fortran package ani3D-MBA (3D metric based adaptation) is a part of the package ani3D . It has been developed by Konstantin Lipnikov and Yuri Vassilevski. Package ani3D-MBA generates conformal tetrahedral meshes which are quasi-uniform in a given metric. The metric may be defined either explicitly, via an analytical formula, or implicitly, via a discrete Hessian recovered from a user-supplied mesh function (usually, a mesh solution). In the first case, the user may generate a mesh with desirable properties. In the second case, the resulting mesh is adapted to the given mesh function enabling a better resolution of sharp changes in the solution.

The library *libmba3D-2.1.a* can be incorporated into other packages.

The input data for our generator is an initial conformal triangulation. It may be a very coarse mesh consisting of a few tetrahedra (for example, made by hands), or a very fine mesh produced by another mesh generator. ani3D-MBA *changes* the initial mesh through a sequence of local modifications. This approach provides a stable algorithm for generating strongly anisotropic grids.

This document describes the structure of the package, input data, and user-supplied (optional) routines. It explains how the user can control the mesh generation process.

2 Copyright and Usage Restrictions

This software is released under the GNU LGPL Licence. You may copy and use this software without any charge, provided that the COPYRIGHT and LICENSE files are attached to all copies.

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

3 For existing users

To accomodate new capabilities, we had to make a few critical changes described below.

- Routine *metric3D* generating a nodal metric has been moved to a separate library *liblmr3D-2.1.a*. As the result, the list of input parameters of the main routines has been modified. The discrete solution (parameter SOL) has been replaced by the **Metric** and parameter **Lp** has been removed.
- The names of main routines have been changed to reflect these modifications. *mbaSolution* has been renamed to *mbaNodal* to highlight the fact that a nodal metric (parameter **Metric**) has to be provided by the user. *mbaMetric* has been renamed to *mbaAnalytic* to highlight the fact that an analytic metric function (parameter **MetricFunction**) has to be provided by the user.

4 Description of ani3D-MBA

The main objective of ani3D-MBA is to produce a mesh with a prescribed number of tetrahedra which is as much quasi-uniform in a given tensor metric as possible. For example, when the metric is isotropic and constant, ani3D-MBA may generate a mesh consisting of equilateral tetrahedra if the model geometry allows this. A measure of quasi-uniformity is a positive number less or equal to 1 which is called the *mesh quality*. The mesh with a prescribed number of equilateral tetrahedra of the same size (measured in the given metric) has quality 1.

4.1 Structure of the package

The main Fortran 77 subroutines of ani3D-MBA are *mbaAnalytic* and *mbaNodal* located in files `mba_metric.f` and `mba_solution.f`, respectively. The depending subroutines are contained in the other files in directory `src/aniMBA`. The examples using these subroutines are in directory `src/Tutorials/PackageMBA`. The files

```
main_analytic.f    main_nodal.f    time.c
```

may be modified by the user. The program in the first file generates a mesh using an analytic (isotropic) metric. The program in the second file uses a user-supplied solution defined at mesh nodes to generate an adapted mesh. File `time.c` is a wrapper for the system call `times` that returns the CPU time. Generally speaking, this routine depends on the operational system. A few examples of files `main_analytic.f` and `main_nodal.f` can be found in directory `src/Tutorials/PackageMBA/examples`.

For user convenience, package ani3D-MBA is equipped with auxiliary files

```
loadM.f      saveM.f
```

Their purpose is to facilitate loading and saving meshes. The files

```
main_analytic.f  main_nodal.f
```

show how to call two main routines of the package: `mbaAnalytic` and `mbaNodal`. For visualization purposes, a simple service library `libview3D-2.1.a` was created. Routine `drawMesh` from `libview3D.a` draws five cuts of the mesh by *XY*-planes. This provides a quick overview of the mesh. For more details, we recommend to use routine `drawGMV` which saves the mesh in the GMV format. To view the `*.gmv` file, the user has to install the General Mesh Viewer from www-xdiv.lanl.gov/XCM/gmv/GMVHome.html. The file

```
Makefile
```

builds the package under Linux. The executable programs are put in directory `bin`. The names for compilers are defined in `src/Rules.make`. A few examples of input meshes may be found in directory `data`. This document and other documentation related to the package ani3D are located in directory `doc`.

4.2 Basic things the user should know

The package provides two major methods for mesh generation. The first method is based on a piecewise linear interpolant of the user-defined metric (see Sec. 7). The second method is based on a piecewise linear metric recovered automatically from the user-supplied mesh function. This mesh function is defined at mesh nodes. If such a mesh function is not available, the package contains a few routines for accurate interpolation of mesh functions defined on edges or triangles to nodes (see Sec. 8).

The package is encapsulated in the two basic routines `mbaAnalytic` and `mbaNodal` corresponding to the above methods. The comments in file `mba_solution.f` are worth to read! After understanding what are input and output data for each of the methods, the user may find more details in files `main_analytic*.f` and `main_nodal*.f` located in directory `src/Tutorials/PackageMBA/examples`.

4.3 Input data

The input data may be split into three types: data files, user input routines (optional) and control parameters.

- The *data files* are the files containing coordinates of mesh nodes, connectivity tables for tetrahedra and boundary faces, and lists of fixed nodes, faces and elements. The lists of fixed nodes, faces and elements may be empty. The list of boundary faces may be also empty. In this case, the boundary faces are recovered by package routines. A good example illustrating format of the data file is `data/cube.ani`. A data file can be accessed via routine `loadMani`.

The mesh loader `loadMani` understands the format of input data files `*.ani` located in directory `data`. For other formats, a new mesh loader has to be written by the user.

Depending of the mesh generation method, in addition to mesh data, nodal values of a mesh function have to be provided. It can be done by the function loader `loadS` located in file `src/aniMBA/loadM.f`.

- The *user input routines* are the Fortran 77 routines describing the analytical space metric. Example of such a routine is in the file `forlibmba.f`. Note that the analytical metric is required only for routine `mbaAnalytic`. For more detail, we refer to comments in the file `forlibmba.f`.

- The *control parameters* are the input parameters that control the mesh generation. They are defined in files `main_analytic.f` and `main_nodal.f`. These files are in directory `src/Tutorials/PackageMBA`. The input control parameters are the following variables:

```

nEstar   - [integer] the desired number of tetrahedra
Lp       - [real*8]  the norm in that the error is minimized
MaxQItr  - [integer] the maximal number of local grid modifications
Quality  - [real*8]  the desired quality for the final grid
              (a positive number between 0 and 1)
MaxSkipE - [integer] the maximal number of skipped tetrahedra

```

The mesh generation is an iterative process every step of which is a local modification of the current mesh. The stopping criteria for the iterative process is either the final mesh quality (`Quality`) or the allowed number of local modifications (`MaxQItr`). We recommend to set `Quality` to a value between 0.3 and 0.5 and to choose `MaxQItr` to be several times bigger than `nEstar`. We also recommend to set `MaxSkipE` (an interior parameter for the iterative process) to the default value which is about 500.

4.3.1 Mesh format

Understanding details of the mesh format is one of the first steps in discovering capabilities of package `ani3D-MBA`. The mesh presentation includes:

```

nP - [integer] the number of points
nF - [integer] the number of boundary and interface faces
nE - [integer] the number of tetrahedra

XYP(2, *) - [real*8]  the Cartesian coordinates of mesh points
IPE(4, *) - [integer] connectivity list of tetrahedra
lbE(*)    - [integer] material indentificator (a positive number)

IPF(3, *) - [integer] connectivity list of boundary and interior faces
lbF(*)    - [integer] boundary indentificator (a positive number)

nPv - [integer] the number of fixed points
nFv - [integer] the number of fixed faces
nEv - [integer] the number of fixed tetrahedra

IPV(*) - [integer] list of fixed points
IFV(*) - [integer] list of fixed faces
IEV(*) - [integer] list of fixed tetrahedra

```

Since some of the mesh data may be empty lists, the minimal mesh representation may contain only `nP`, `nE`, `XYP`, `IPE` and `lbE`.

5 Getting started

After package installation, the user will get the following subdirectories

```
bin/ data/ doc/ include/ lib/ src/
```

By default, the executable files are stored in `bin/`. A few example of input files are located in `data/`. A documentation for the package may be found in `doc/`. The source code is stored in `src/aniMBA/`. The examples are in directory `src/Tutorials/PackageMBA/`. In order to compile the code, the user has to set up the compilers names in `scr/Rules.make` and then to execute the following commands:

```

$make libs
$cd src/Tutorials/PackageMBA
$make help exe

```

The user may change the names and options for compilers in file `src/Rules.make`. After the successful compilation, the user may run one of the executables in `bin/`. The same task can be accomplished with `make run-met` or `make run-sol`. The output may look like:

```

$ cd bin; ./mbaAnalytic.exe

Loading mesh ../data/ramp.ani
Saving GMV image ramp.gmv

STONE FLOWER! (1997-2008), version 2.1
Target: Quality 0.40 with 20000 tetrahedra for at most 50000 iterations

Avg Quality is 0.3157E+00, Maximal R/r = 0.9044E+01, status.fd: 11
ITRs:      28 Q=0.2620E-03 #P,F,E: 1558      956      7340 tm= 0.0s
ITRs: 21574 Q=0.2426E+00 #P,F,E: 5153      2424      24912 tm= 11.0s
Total: 21574 Q=0.2426E+00 #V,F,E: 5153      2424      24912 tm= 11.1s
Avg Quality is 0.5745E+00, Maximal R/r = 0.3704E+01, status.fd: 11

Saving GMV image save.gmv
Saving mesh save.ani

```

First, some of the control parameters and average mesh quality are printed. Then, the quality of the current mesh and the numbers of points, faces and tetrahedra are printed. Additional output goes into GMV files `ramp.gmv` and `save.gmv` containing initial and final meshes, respectively. The files are located in directory `bin`. One way to check the contents of these files is to execute `gmw -i ramp.gmv`.

The program loads the input file `../data/ramp.ani`. To run a different example, the user may either to change the name of the input file in the mesh loader:

```

Call loadMone(
&      MaxP, MaxF, MaxE, nP, nF, nE,
&      XYP, IPF, IPE, lbF, lbE,
&      nPv, nFv, nEv, IPV, IFV, IEV,
&      iW, iW, "../data/cube.ani")

```

or to use one the examples from directory `src/Tutorials/PackageMBA/examples`. The user may play with the input control parameters in file `main_analytic.f` and the analytical metric in file `forlibmba.f`. For instance, changing the metric the user will learn how to control the shape of tetrahedra.

6 Useful features of ani3D-MBA

We continuously improve robustness and efficiency of the code, make it more user friendly and add new features in each release. The most important features are listed below:

1. It is possible to produce meshes minimizing different L_p -norms of the interpolation. The input non-negative parameter L_p defines this norm. In the special case $L_p = 0$, the maximum norm is used.
2. The complete list of available features is in file `src/aniMBA/status.fd`. Here are the most important features:
 - The user may freeze boundary points. This allows to preserve important geometric features for both isotropic and anisotropic metrics.

- The user may freeze boundary faces and/or mesh elements. This allows to preserve mesh structure in important regions (e.g., in boundary layers).
 - The interfaces between materials with different labels (lbE) are recovered and preserved automatically.
 - The initial mesh may be tangled. In this case, the user may add `ANITangleMesh` defined in `src/aniMBA/status.fd` to the input parameter `status` to untangle the input mesh.
3. The library *libmba3D-2.1.a* contains routine *P02P1* which maps a discontinuous piece-wise constant function defined on elements onto a continuous piece-wise linear function defined at mesh points (see `src/aniMBA/ZZ.f` for more detail).

The same library contains a few routines *listX2Y* which create connectivity lists $X \rightarrow Y$ for mesh objects X and Y such as elements, faces, boundary faces, edges, and points (see `src/aniMBA/utils.f` for more detail).

7 How to use library libmba3D-2.1

Here we describe one of the main subroutines, *mbaNodal*, from the library *libmba3D-2.1.a*. Both subroutines have the same number of parameters and differ in only one parameter (`MetricFunction` vs `Metric`).

```
Call mbaAnalytic(
&    nP, MaxP, nF, MaxF, nE, MaxE,
&    XYP, IPF, IPE, lbF, lbE,
&    nEStar,
&    nPv, nFv, nEv, IPV, IFV, IEV,
&    flagAuto, status,
&    MaxSkipE, MaxQItr,
&    MetricFunction, Quality, rQuality,
&    MaxWr, MaxWi, rW, iW,
&    iPrint, iERR)
```

```
Call mbaNodal(
&    nP, MaxP, nF, MaxF, nE, MaxE,
&    XYP, IPF, IPE, lbF, lbE,
&    nEStar,
&    nPv, nFv, nEv, IPV, IFV, IEV,
&    flagAuto, status,
&    MaxSkipE, MaxQItr,
&    Metric, Quality, rQuality,
&    MaxWr, MaxWi, rW, iW,
&    iPrint, iERR)
```

Some of the parameters were described in Section 3 (see file `src/aniMBA/mba_solution.f` for more detail). The details on the other input parameters are below:

```
I    MaxP - [integer] maximal number of points
N    MaxF - [integer] maximal number of boundary and interface faces
P    MaxE - [integer] maximal number of tetrahedra
U
T    nEstar - [integer] the desired number of tetrahedra

    nFv - [integer] the number of fixed faces
P    nEv - [integer] the number of fixed tetrahedra
A
```

R
A
M
E
T
E
R
s

IFV(nFv) - [integer] list of fixed faces
 IEV(nEv) - [integer] list of fixed tetrahedra

flagAuto - [logical] flag controlling mesh generation:
 TRUE - recover missing mesh elements
 FALSE - check that input data are complete

MaxSkipE - [integer] maximal number of skipped tetrahedra
 MaxQItr - [integer] maximal number of mesh modifications

Metric(6, nP) - metric defined at mesh nodes. The metric is a
 3x3 symmetric matrix M_{ij} . Each column of this
 array stores the upper diagonal entries in the
 following order: M_{11} , M_{22} , M_{33} , M_{12} , M_{23} , M_{13} .

Metricfunction - [integer] function created by the user:

Integer Function MetricFunction(x,y,z, Metric)

It provides a 3x3 symmetric metric at the given
 point (x,y,z). Only the upper triangular part of
 array Metric must be defined.

Quality - [real*8] target quality for the final mesh

Lp - [real*8] the norm in which the final mesh be optimal
 Lp > 0 means the L_p norm
 Lp = 0 means the maximum norm
 Lp < 0 means the H_1 norm (not implemented)

MaxWr - [integer] maximal memory allocation for rW
 MaxWi - [integer] maximal memory allocation for iW

iPrint - [integer] level of output information (0 - nothing)

Here we collect parameters which are both input and output:

I
N
P
U
T
/
O
U
T
P
U
T
P
A
R
A

nP - [integer] the number of points
 nF - [integer] the number of boundary and interface faces
 nE - [integer] the number of tetrahedra

XYP(3, MaxP) - [integer] list of points
 IPE(4, MaxE) - [integer] list of tetrahedra
 lbE(MaxE) - [integer] material indentificator

IPF(3, MaxF) - [integer] list of boundary and interface faces
 ParCrv(2, MaxF) - [real*8] parametrizations of curved edges
 iFnc(MaxF) - [integer] list of corresponding functions

nPv - [integer] the number of fixed points
 IPV(nPv) - [integer] list of fixed points

rQuality - [real*8] quality of the final mesh

status - [integer] sum of positive numbers corresponding

```

M                               to desired mesh properties (see status.fd)
E
T      rW(MaxWr) - [real*8]  working array
E      iW(MaxWi) - [integer] another working array
R
s      iERR    - error code:
           0 - the correct program termination

```

8 Useful routines

The library *libmba3D-2.1.a* has a few subroutines that can be useful in other projects. Most of the input parameters in these routines are explained above.

- Uniform and local mesh refinement (partitioning of a tetrahedron into 8 subtetrahedra). The mesh shape quality is not degrading during this refinements. Local refinement is robust if refinement regions are not fractal. Before refinement, initialization step has to be performed that fills partially arrays *MapMtr(3,3,MaxE)* and *Ref2MapMtr(MaxE)*.

```

Subroutine initializeRefinement(
&      nP, nE, XYP, IPE,
&      MapMtr, Ref2MapMtr)

```

Then, uniform refinement subroutine can be called (the size of working integer array *iW* is at least $14\ nE + nP$ where *nP, nE* are the input values)

```

Subroutine uniformRefinement(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      XYP, IPF, IPE, lbF, lbE,
&      MapMtr, Ref2MapMtr,
&      iW, MaxWi)

```

A local refinement subroutine refines elements marked *.TRUE.* in the logical array *SplitFlag(nE)*. The size of working integer array *iW* is at least $14\ nE + nP + 3\ nR + 4\ \min(\text{MaxF}, 4\ nF)$ where *nP, nE, nF, MaxF* are the input values and *nR* is the estimated number of edges in the input mesh:

```

Subroutine localRefinement(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      XYP, IPF, IPE, lbF, lbE,
&      MapMtr, Ref2MapMtr, SplitFlag,
&      iW, MaxWi)

```

- *P02P1* maps a discontinuous piece-wise constant function defined on tetrahedra onto a continuous piece-wise linear function defined at points. We use the *ZZ* method for interpolation. We assume that each boundary node can be connected with an interior node by at most two mesh edges. The size of working integer array *iW* is $4\ nE + 3\ nP$. The size of working double precision array *rW* is *nP*.

```

Subroutine P02P1(
&      nP, nF, nR, nE, XYP, IPF, IPE,
&      fP0, fP1,
&      MaxWr, MaxWi, rW, iW)

```

- `listE2R` creates a connectivity list `IRE` for mesh edges. The routine counts mesh edges. For an element `E`, `IRE([1:6], E)` give indexes of six edges in the following order: 12, 13, 14, 23, 24, and 34. For example, the second edge is `[IPE(1,E), IPE(3,E)]`. The working integer arrays are `nEP(nP)` and `IEP(4 nE)` (see `src/aniMBA/utills.f` for more detail):

```
Subroutine listE2R(
&          nP, nR, nE, IPE, IRE, nEP, IEP)
```

- `listR2R` creates connectivity lists `nRR` and `IRR` for mesh edges. The routine counts the number of mesh edges, `nR`. Then, `nRR(i) - nRR(i-1)` (`nRR(1)` when `i=1`) gives the total number of edges in tetrahedra sharing the edge `i`. The corresponding edge numbers are saved in array `IRR` in positions `nRR(i-1) + 1` to `nRR(i)`. The size of working integer array `iW` is `12 nE + nR` (see `src/aniMBA/utills.f` for more detail):

```
Subroutine listR2R(
&          nP, nR, nE, MaxL, IPE, nRR, IRR, iW, iERR)
```

- File `src/aniMBA/utills.f` contains more routines for creating other connectivity lists such as edges to points, points to points, elements to boundary edges, etc. The routine `listConv` colvolutes two given connectivity lists. The routines `backReferences` and `reverseMap` create reverse connectivity lists for a given structured and unstructured connectivity list, respectively. For instance, `backReferences` takes the structured connectivity list `IPE` from elements to points and creates the unstructured connectivity lists `nEP` and `IEP` from points to elements.

9 FAQ

- Q. The mesh generator does not refine the input mesh.
A. There are two cases when the code may do nothing. First, the number of mesh elements whose quality is limited by model geometry (e.g. thin layers) is bigger then the control parameter `MaxSkipE`. The remedy is to increase this parameter. Second, a severe anisotropic input metric does not allow to insert new mesh points in a very coarse mesh. The simple remedy is to refine mesh using an isotropic metric and then switch to the anisotropic metric.
- Q. The mesh generator uses the same input data but produces different grids on different computers.
A. The output of the mesh generator may depend on a computer arithmetics. The order of local mesh modifications depends on round-off errors and may be computer-dependent.
- Q. The final mesh quality is very small.
A. The mesh quality equals to quality of the worst tetrahedron in the mesh. In some cases, the shape of near-boundary tetrahedra is driven mainly by the geometry. A possible remedy is either to increase the number `nEStar` of desired tetrahedra or to fix a possible contradiction between the boundary and the metric. Another reason for the low mesh quality is strong jumps in the metric. If the metric is isotropic, the optimal tetrahedra are equilateral ones. This size is changed strongly across lines of metric discontinuity.
- Q. The number of tetrahedra in the final mesh is never equal to `nEStar`.
A. `nEStar` is achieved exactly if and only if `Quality = 1` and the computational domain may be covered by equilateral (in the user given metric) tetrahedra. Apparently, it is possible only in very special cases.
- Q. Why `mbaNodal` and `mbaAnalytic` have so many input parameters?
A. The package has routines `mbaAnalyticShort` and `mbaNodalShort` with functionality close to that of main subroutines `mbaAnalytic` and `mbaNodal`, respectively, but with smaller number of input parameters. For example, lists of fix points and boundary tetrahedra are omitted.

- Q. Is it possible to use *libmba3D-2.1.a* in an adaptive loop?
A. Yes. Use `make libs` to generate the library *libmba3D-2.1.a* which may be linked with other codes. Depending on the user goals, he or she may call either *mbaAnalytic* or *mbaNodal*. The package contains a few examples of solving partial differential equations on adaptive grids (see `src/Tutorials/MultiPackage` for more detail).
- Q. Why does *libmba3D-2.1.a* fail to untangle the mesh?
A. This may happen when the initial mesh is either topologically incorrect or extremely tangled. The second case is curable. Try to run the code with the identity metric or/and change significantly the desired number of mesh elements.
- Q. I do not understand why *libmba3D-2.1.a* fails to generate a mesh.
A. The developers are interested in any feedback from users. To report a problem, please email to either `lipnikov@hotmail.com` or `vasilevs@dodo.inm.ras.ru`. To help us to fix the problem, please attach file `main_analytic.f` or `main_nodal.f` and files containing the input mesh.

References

1. Yu.Vassilevski and K.Lipnikov, An adaptive algorithm for quasioptimal mesh generation, *Computational Mathematics and Mathematical Physics* (1999) **39**, No.9, 1468–1486.
2. A.Agouzal, K.Lipnikov, Yu.Vassilevski, Adaptive Generation of Quasi-optimal Tetrahedral Meshes, *East-West Journal* (1999) **7**, No.4, 223–244.
3. K.Lipnikov, Y.Vassilevski, Parallel adaptive solution of 3D boundary value problems by Hessian recovery, *Comput. Methods Appl. Mech. Engrg.* (2003) **192**, 1495–1513.
4. K.Lipnikov, Yu. Vassilevski, Optimal triangulations: existence, approximation and double differentiation of P_1 finite element functions, *Computational Mathematics and Mathematical Physics* (2003) **43**, No.6, 827–835.
5. K.Lipnikov, Yu.Vassilevski, On a parallel algorithm for controlled Hessian-based mesh adaptation. Proceedings of 3rd Conf. Appl. Geometry, Mesh Generation and High Performance Computing, Moscow, June 28 - July 1, Comp. Center RAS, V.1, 2004, 154–166.
6. K.Lipnikov, Yu.Vassilevski, On control of adaptation in parallel mesh generation. *Engrg. Computers* (2004) **20**, 193–201.
7. K.Lipnikov, Yu.Vassilevski, Error bounds for controllable adaptive algorithms based on a Hessian recovery. *Computational Mathematics and Mathematical Physics* (2005) **45**, 1374–1384.
8. K.Lipnikov, Yu.Vassilevski, Analysis of Hessian recovery methods for generating adaptive meshes. *Proceedings of 15th International Meshing Roundtable*, P.Pebay (Editor), Springer, Berlin, Heidelberg, New York, 2006, pp.163–171.

Ani3D-FEM version 2.1 ”*Sunflower*”

**Flexible Generator of Finite Elements
Systems on Tetrahedral Meshes**

User’s Guide for libfem3D-2.1.a

1 Introduction

The Fortran package ani3D-FEM is developed by Konstantin Lipnikov and Yuri Vassilevski. It is designated for generating finite element matrices on tetrahedral meshes. The package allows to build elemental matrices for variety of finite elements, modify these matrices, assemble them, and impose boundary conditions.

The package ani3D-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. For instance, the elemental matrix for the Stokes problem is a saddle point matrix involving both the pressure and velocity unknowns.

The library *libfem3D-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

This software is release under the GNU LGPL Licence. You may copy and use this software without any charge, provided that the COPYRIGHT and LICENSE files are attached to all copies.

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

3 Description of Ani3D-FEM

3.1 Elemental finite element matrix

The core of the package is routine *fem3Dtet* which computes elemental matrix corresponding to the bilinear form

$$\langle DOP_A(u^h), OP_B(v^h) \rangle \quad (1)$$

where D is a tensor, OP_A and OP_B are linear first-order or zero-order differential operators, and u^h and v^h are finite element functions. Here is the list of implemented finite elements (see file *fem3Dtet.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 unknown are ordered first by vertices, then by edges, and then by the space directions (x and y)
FEM_ND1	the lowest order Nedelec (edge) finite element
FEM_RT1	the lowest order Raviart-Thomas (face) finite element
FEM_CR1	the Crouzeix-Raviart finite element

Here is the list of available operators (see file *fem3Dtet.f* for more detail):

IDEN	identity operator
GRAD	gradient operator
DIV	divergence operator
CURL	rotor operator

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 tensor
TENSOR_GENERAL	general tensor

The package uses several quadrature formulae:

order = 1	quadrature formula with one center point
order = 2	quadrature formula with 4 points inside tetrahedron
order = 3	quadrature formula with 8 points
order = 5	quadrature formula with 15 points inside tetrahedron

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, z, label, DATA, iSYS, Coef)

C   The function returns type of the tensor Coef (see the table above).
C
C   (x, y, z) - [input] Real*8 Cartesian coordinates of a 3D
C               point where tensor Coef should be evaluated
C
C   label      - [input] integer label of a either element or face
C
C   DATA      - [input] Real*8 user given data array
C
C   iSYS       - [input/output] integer buffer for information exchange:
C               iSYS(1) [input] tetrahedron number
C               iSYS(2) [input] 1st vertex number
C               iSYS(3) [input] 2nd vertex number
C               iSYS(4) [input] 3rd vertex number
C               iSYS(5) [input] 4th 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(9,jD) - [output] Real*8 matrix with the leading dimension 9!

```

To compute entries of the tensor $Coef$, the user may use the tetrahedron 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, z) .
- anisotropic diffusion coefficient. The user has to set $iD = jD = 3$, $Dcoef = TENSOR_SYMMETRIC$, and to return diffusion tensor (3×3 matrix with entries $Coef(i,j)$, $i,j=1,2,3$) at the point (x, y, z) .
- convection coefficient. The user has to set $iD = 1$, $jD = 3$, $Dcoef = TENSOR_GENERAL$, and to return the velocity vector values $Coef(1,1)$, $Coef(1,2)$, $Coef(1,3)$ at the point (x, y, z) .

Now we are ready to call routine *fem3Dtet* which computes an elemental matrix A :

```

Call FEM3Dtet(
&   XY1, XY2, XY3, XY4,
&   OpA, FemA, OpB, FemB,
&   label, Dcoef, DATA, iSYS, order,

```

```

&    LDA, A, nRow, nCol)

C    XYi(3)    - [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 , $i = 1, 2, 3, 4$ (input parameters XY1, XY2, XY3, XY4).
- Second, basis functions associated with edges (if any) are numerated in the order of edges r_{12} , r_{13} , r_{14} , r_{23} , r_{24} and r_{34} .
- Third, basis function associated with faces are numerated in the same order as faces r_{123} , r_{234} , r_{341} and r_{412} .
- Fourth, basis functions associated with element (if any) are numbered.
- The vector basis functions with 3 degrees of freedom per a mesh object (vertex, edge, face) are enumerated first by the corresponding mesh objects and then by the space coordinates, first x , then y , and finally z .

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

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

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

```

Call FEM3Dtet(
&    XY1, XY2, XY3, XY4,
&    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 *fem3Dtet* 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 FEM3Dext(
&    XY1, XY2, XY3, XY4,
&    lbE, lbF, lbR, lbP, DATA, iSYS,
&    LDA, A, F, nRow, nCol,
&    templateR, templateC)

C    XYi(3) - [input] Real*8 Cartesian coordinates of the i-th point
C
C    lbE    - [input] Integer label of the tetrahedron (material label)

```

```

C     lbF(4) - [input] Integer labels of its faces (boundary labels)
C                   lbFloc(i) = 0 for internal faces
C     lbR(6) - [input] Integer labels of its edges (copied from global lbR)
C     lbP(3) - [input] Integer labels of its vertices (copied from global lbP)
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) tetrahedron number
C           iSYS(2) 1st vertex number
C           iSYS(3) 2nd vertex number
C           iSYS(4) 3rd vertex number
C           iSYS(5) 4th 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 for rows. We
C                   recommend to group them: four for points, six for edges, etc.
C     templateC(nCol) - [output] Integer array of degrees of freedom 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 four unknowns associated with points will go to the first group of point-based unknowns. Next four point-based unknowns will go to the second group. After counting all point-based unknowns, we group the face-based unknowns, then the edge-based unknowns, and finally the element-based unknowns.

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

- For a diffusion reaction equation, we sum elemental matrices corresponding to the diffusion and reaction terms.
- For a diffusion equation written in a mixed form, we use hybridization algorithm inside FEM3Dext and return the elemental matrix for Lagrange multipliers.
- 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 CSR (compressed sparse row) format with diagonal entries at 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(3, nP) - [input] Real*8 Cartesian coordinates of mesh points
C      IPF(3, nF) - [input] connectivity list of boundary faces (see
C                  documentation for package Ani3D-MBA)
C      IPE(4, nE) - [input/output] connectivity list of elements.
C                  On output, each column is ordered by increasing.
C
C      lbF(nF)    - [input] boundary labels
C      lbE(nE)    - [input] element (material) labels
C
C      assembleStatus - [input] a priori 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 the AMG solver (CSR with
C                  rows starting with the 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 will be passed to the user-written routine *FEM3Dext* and may be used to assign Dirichlet boundary conditions.

```

Subroutine BilinearFormTemplate(
&      nP, nF, nE, XYP, lbP, IPF, lbF, IPE, lbE,

```

```
&      FEM3Dext, 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 $\Omega = T(x, y) \times (0, 1)$ is a prism with base T . This base is the triangle with vertices $(0, 0)$, $(0, 1)$ and $(1, 0)$. The Neumann boundary condition is imposed on only one side of the prism, $\partial\Omega_N = \{(x, y, z) \in \partial\Omega : y = 0\}$. The Dirichlet boundary condition is imposed on the rest of the boundary $\partial\Omega_D = \partial\Omega \setminus \partial\Omega_N$. The diffusion coefficient D is a scalar function:

$$D(x, y, z) = 1 + x^2.$$

First, the program refines an initial coarse mesh consisting of three tetrahedra. This is accomplished with routines `initializeRefinement` and `uniformRefinement` from the library `libmba3D-2.1.a`. Second, the program generates the finite element system using subroutines `BilinearFormVolume`, `LinearFormVolume` and `BoundaryConditions` from the library `libfem3D-2.1.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:

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

where $\Omega = T(x, y) \times (0, 1)$ is a prism with base T . This base is the triangle with vertices $(0, 0)$, $(0, 1)$ and $(1, 0)$. The boundaries $\partial\Omega_i$ are defined as follows:

$$\begin{aligned} \partial\Omega_1 &= \{(x, y, z) \in \partial\Omega : z = 0\}, \\ \partial\Omega_3 &= \{(x, y, z) \in \partial\Omega : z = 1\}, \\ \partial\Omega_2 &= \partial\Omega \setminus (\partial\Omega_1 \cup \partial\Omega_3). \end{aligned}$$

The non-homogeneous boundary condition is

$$\mathbf{u}_0 = (\mathbf{0}, \mathbf{0}, 4\mathbf{y}(1 - \mathbf{y}))^T.$$

First, the program refines an initial coarse mesh consisting of three tetrahedra and creates a quasi-uniform mesh with 460 elements. This is accomplished with subroutine `mbaMetric` from the library `libmba3D-2.1.a`. Second, the program generates the finite element system using subroutine `BilinearFormTemplate` from the library `libfem3D-2.1.a`. The elemental matrices are generated in subroutine `FEM3Dext` which is in file `aniFEM/mainTemplate.f`.

ani3D-ILU Version 2.1 ”Bellflower”

**Flexible Iterative Solver Using
Incomplete LU Factorization**

User’s Guide for libilu-2.1.a

1 Basic features of the library

The FORTRAN package ani3D-ILU is an independent part of the package ani3D . ani3D-ILU was developed by Yuri Vassilevski, Sergey Goreinov and Vadim Chugunov. It is designated for the iterative solution of sparse linear systems. Ani3D-ILU may be easily incorporated into any other software, for instance, package Ani2D.

The basic features of library *libilu.a* are listed below.

Iterative method : BiConjugate Gradient Stabilized, BiCGStab, and Conjugate Gradient, PCG

Preconditioners : ILU0 and ILU2, the second order accurate ILU

Matrix storage format : Compressed Sparse Row-wise, CSR

Data format : double precision or integer arrays. Enumeration starts from 1.

Typical memory requests : for systems with N equations and NZ non-zero matrix elements, BiCGStab (resp., PCG) needs 8 (resp., 4) work vectors of dimension N , right-hand side and solution vectors. ILU0 requires the same storage as the CSR matrix representation. ILU2 requires upto 2-5-fold memory for the CSR matrix representation.

2 Iterative solution

The default iterative solver is BiConjugate Gradient Stabilized method (BiCGStab). This is the Krylov subspace method applicable to non-singular non-symmetric matrices. Therefore, it requires two procedures: matrix-vector multiplication and precondition-vector evaluation. If the user is not confident that the matrix is symmetric positive definite, he or she is advised to choose the default method. The call of the method is

```
call slpbcgs(  
& prevec, IPREVEC, iW,rW,  
& matvec, IMATVEC, ia,ja,a,  
& WORK, MW, NW,  
& N, RHS, SOL,  
& ITER, RESID,  
& INFO, NUNIT )
```

- **prevec** is the name of a precondition-vector multiply routine and **IPREVEC** is an integer array of user's data which may be passed to **prevec** and used there. In the presented examples of preconditioners **IPREVEC** contains single entry equal to the system order. The format of **prevec** is:

```
Subroutine prevec(IPREVEC, ICHANGE, X, Y, iW, rW)  
c Input  
Integer IPREVEC(*), ICHANGE, iW(*)  
Real*8 X(*), rW(*)  
c Output  
Real*8 Y(*)
```

This routine solves the system $(LU)Y = X$ with L low triangular and U upper triangular factors stored in arrays **iW,rW**. **ICHANGE** is the flag controlling the change of the preconditioner (useful when convergence stagnation occurs). The user is given two examples of **prevec** corresponding to two preconditioners, **prevec0** (ilu0.f) and **prevec2** (ilu00.f).

- **iW, rW** are Integer and Real*8 arrays which store the preconditioner.
- **matvec** is the name of generalized matrix-vector multiply routine and **IMATVEC** is an integer array of user's data which may be passed to **matvec** and used there. In the presented example **IMATVEC** contains single entry equal to the system order. The format of **matvec** is:

```

        Subroutine matvec(IMATVEC, ALPHA, X, BETA, Y, ia, ja, a)
c Input
        Integer IMATVEC(*), ia(*), ja(*)
        Real*8 X(*), Y(*), a(*), ALPHA, BETA
c Output
        Real*8 Y(*)

```

This routine calculates matrix-vector product AX and adds the vector βY :

$$Y := \alpha AX + \beta Y.$$

For example, for $\alpha = 1, \beta = 0$ `matvec` returns $Y = AX$. The example of a `matvec` routine is in file `bcg.f`. It uses the compressed sparse row (CSR) representation of matrix A stored in arrays `ia`, `ja`, `a`.

- `ia, ja, a` are two Integer and one Real*8 arrays containing matrix in the CSR format.
- `WORK(MW,NW)` is Real*8 working two-dimensional array which stores at least 8 Krylov vectors.
- `MW*NW` the total length of `WORK` which must be not less than `8N`.
- `N` is order of system and length of vectors.
- `RHS` is the right hand side vector (Real*8).
- `SOL` is the initial guess and the iterated solution (Real*8).
- `ITER` is the maximal number of iterations on input and the actual number of iterations on output.
- `RESID` is the convergence criterion on input and norm of the final residual on output.
- `INFO` is the performance information, 0 - converged, 1 - did not converge, etc.
- `NUNIT` is the channel number for output (0 - no output).

If the user is confident that the matrix is symmetric and positive definite, he or she can save 4 work vectors and probably 10-30% of the CPU time by calling the Preconditioned Conjugate Gradient method (PCG):

```

call slpcg(
& prevec, IPREVEC, iw,rw,
& matvec, IMATVEC, ia,ja,a,
& WORK, MW, NW,
& N, RHS, SOL,
& ITER, RESID,
& INFO, NUNIT )

```

The parameters of this routine are the same, except that `MW*NW` must be not less than $4N$.

3 ILU0 preconditioner

ILU0 preconditioner is the simplest and the most popular ILU preconditioner. It is characterized by very fast and economical factorization. The drawbacks of the method are slow convergence and danger to get zero pivot. Nevertheless, for simple non-stiff problems it works well. The application of the preconditioner has two stages: initialization and evaluation. The evaluation must be performed at each step of the iterative method. It is provided by the routine `prevec0` accompanying the initialization routine `ilu0`. The user should only put the name `prevec0` as the input parameter in the iterative solver:

```

    external prevec0
    ....

    call slpbcgs(
& prevec0, IPREVEC, iW,rW,
& matvec, IMATVEC, ia,ja,a,
& WORK, MW, NW,
& N, RHS, SOL,
& ITER, RESID,
& INFO, NUNIT )

```

The initialization routine has the following parameters

```

    call ilu0(n, a, ja, ia, alu, jlu, ju, iw, ierr)

```

where

- `n` is matrix order,
- `ja,ia,a` are two Integer and one Real*8 arrays containing the matrix in the CSR format,
- `alu,jlu,ju` are one Real*8 and two Integer arrays containing the L and U factors together,
- `ierr` is the integer error code (0 - successful factorization, k - zero pivot at step k),
- `iw` is the integer working array of length n .

Below we present the basic blocks of a program solving a system with matrix `a`, `ia`, `ja` and a right hand side vector `f` by the BiCGstab method with the ILU0 preconditioner.

First we define all necessary arrays and variables:

C Arrays for matrix kept in CSR format

```

    Integer ia(maxn+1), ja(maxnz)
    Real*8  a(maxnz), f(maxn), u(maxn)

```

C Work arrays keeping ILU factors and 8 BCG vectors

```

    Integer  MaxWr,MaxWi
    Parameter(MaxWr=maxnz+8*maxn, MaxWi=maxnz+2*maxn+1)
    Real*8  rW(MaxWr)
    Integer iW(MaxWi)

```

C BiCGStab data

```

    External matvec, prevec0
    Integer  ITER, INFO, NUNIT
    Real*8   RESID

```

C ILU0 data

```

    Integer  ierr, ipaLU, ipjLU, ipjU, ipiw

```

C Local variables

```

    Integer  ipBCG

```

When the matrix is stored in the CSR format, we initialize the preconditioner by computing L and U factors and saving them in `rW`, `iW`:

```

ipaLU=1
ipBCG=ipaLU+nz
ipjU =1
ipjLU=ipjU+n+1
ipiw =ipjLU+nz !work array of length n

call ilu0(n,a,ja,ia, rW(ipaLU),iW(ipjLU),iW(ipjU),iW(ipiw),ierr)

if (ierr.ne.0) then
  write(*,*)'initialization of ilu0 failed, zero pivot=',ierr
  stop
end if

```

c Keep data in rW and iW up to rW(nz) and iW(nz+n+1) !

Once the preconditioner is initialized, we can call the iterative solution:

```

ITER = 1000           ! max number of iterations
RESID = 1d-8         ! threshold for \|RESID\|
INFO = 0             ! no troubles on input
NUNIT = 6            ! output channel

```

```

call slpbcgs(
> prevec0, n, iW,rW,
> matvec, n, ia,ja,a,
> rW(ipBCG), n, 8,
> n, f, u,
> ITER, RESID,
> INFO, NUNIT )

```

```

if (INFO.ne.0) stop 'BiCGStab failed'

```

An example of calling program is in file `src/Tutorials/PackageILU/main_ilu0.f`.

4 ILU2 preconditioner

The ILU2 preconditioner is an ILU factorization with two thresholds proposed by I.Kaporin in 1998. For symmetric positive definite stiff systems it is shown to be robust and to give better convergence rates compared to other factorizations. It can be applied for non-symmetric matrices as well. The factorization of the input matrix A satisfies the formula

$$A = LU + TU + LR - S$$

where L, U are the first order factors, T, R are the second order factors (kept and used in calculation, neglected after calculation), S is the residual matrix (neglected during the calculation). The method seems to be a very flexible and powerful tool to construct efficient preconditioners for stiff matrices. The application of the preconditioner has two stages: initialization and evaluation. The evaluation must be performed at each step of an iterative method. It is provided by the routine `prevec2` accompanying the initialization routine `ilu00`. The user should only put the name `prevec2` as an input parameter in the iterative solver:

```

    external prevec2
    ....

    call slpbcgs(
& prevec2, IPREVEC, iW,rW,
& matvec, IMATVEC, ia,ja,a,
& WORK, MW, NW,
& N, RHS, SOL,
& ITER, RESID,
& INFO, NUNIT )

```

The initialization routine has the following parameters

```

    call iluoo (n, ia, ja, a, tau1, tau2, verb,
& work, iwork, lendwork, leniwork,
& partlur, partlurout,
& lendworkout, leniworkout, ierr)

```

- `n` is the order of the square matrix A ;
- `ia,ja,a` are two Integer and one Real*8 arrays containing matrix in the CSR format;
- `tau1` is the absolute threshold for entries of L and U (elements of L and U greater than τ_1 will enter L and U ; recommended values lie in the interval $[0.01; 0.1]$);
- `tau2` is the absolute threshold for entries of T and R (elements not included in L and U but greater than τ_2 will enter T and R ; recommended value lie in the interval τ_1^2 or $5\tau_1^2 - 0.1\tau_1$);
- `verb` sets up the verbosity level: 0 means no output, positive means verbose output;
- `work,iwork,lendwork,leniwork` are working Real*8 and Integer arrays and their sizes;
- `partlur` is user defined partition of the available memory `work,iwork`, L,U occupy $(1-\text{partlur}) * \text{lendwork}$ and R occupies `partlur*lendwork`);
- `partlurout` is the optimal partition computed during factorization (may be useful for the next factorization);
- `lendworkout,leniworkout` are the actual memory demands;
- `ierr` is the integer error code (0 - successful factorization).

Below we present the basic blocks of a program solving a system with matrix `a`, `ia`, `ja` and a right hand side vector `f` by the BiCGstab method with the ILU2 preconditioner. First we define all necessary arrays and variables:

```

C Arrays for matrix kept in CSR format
  Integer ia(maxn+1), ja(maxnz)
  Real*8  a(maxnz), f(maxn), u(maxn)

```

```

C Work arrays
  Integer  MaxWr,MaxWi
  Parameter(MaxWr=5*maxnz, MaxWi=6*maxnz)
  Real*8  rW(MaxWr)
  Integer iW(MaxWi)

```

```

C BiCGStab data

```

```

External  matvec, prevec2
Integer   ITER, INFO, NUNIT
Real*8    RESID

```

C ILU data

```

Real*8    tau1,tau2,partlur,partlurout
Integer   verb, ierr, UsedWr, UsedWi

```

C Local variables

```

Integer   ipBCG, ipIFREE

```

When the matrix is stored in the CSR format, we initialize the preconditioner by computing L and U factors and saving them in rW , iW :

```

verb      = 0      ! verbose no
tau1      = 1d-2
tau2      = 1d-3
partlur   = 0.5
ierr      = 0

call iluoo (n, ia, ja, a, tau1, tau2, verb,
&  rW, iW, MaxWr, MaxWi, partlur, partlurout,
&  UsedWr, UsedWi, ierr)

if (ierr.ne.0) then
  write(*,*)'initialization of iluoo failed, ierr=',ierr
  stop
end if

if (UsedWr+8*n.gt.MaxWr) then
  write(*,*) 'Increase MaxWr to ',UsedWr+8*n
  stop
end if
ipBCG = UsedWr + 1

```

Once the preconditioner is initialized, we can call the iterative solution:

```

ITER = 1000          ! max number of iterations
RESID = 1d-8        ! threshold for \RESID\
INFO = 0            ! no troubles on input
NUNIT = 6           ! output channel

call slpbcgs(
> prevec2, n, iW,rW,
> matvec, n, ia,ja,a,
> rW(ipBCG), n, 8,
> n, f, u,
> ITER, RESID,
> INFO, NUNIT )

if (INFO.ne.0) stop 'BiCGStab failed'

```

An example is given in file `src/Tutorials/PackageILU/main_ilu2.f`.