Ani2D

The package Ani2D is designated for generating unstructured triangular meshes, adapting them isotropically and anisotropically, discretizing systems of PDEs, solving linear and nonlinear systems, visualizing meshes and associated solutions. It is a set of independent libraries with different tasks. The libraries may be combined to solve a complex problem. Extensive tutorials represent powerful capabilities of the package.

The package Ani2D has been developed by a team of researchers since 1997. The team is headed by the two principle investigators:

- Konstantin Lipnikov[1]
- Yuri Vassilevski[2].

Ideas and technologies, as well as packages Ani2D-MBA , Ani2D-FEM , Ani2D-LMR and Ani2D-VIEW , have been developed by the principal investigators.

The package Ani2D-AFT has been developed by

- Alexander Danilov[2]

under the supervision of the principal investigators.

The packages Ani2D-ILU and Ani2D-RCB have been developed by

- Sergei Goreinov[2]
- Vadim Chugunov[2]
- Yuri Vassilevski[2].

The package Ani2D-INB has been developed by

- Alexey Chernyshenko[2]

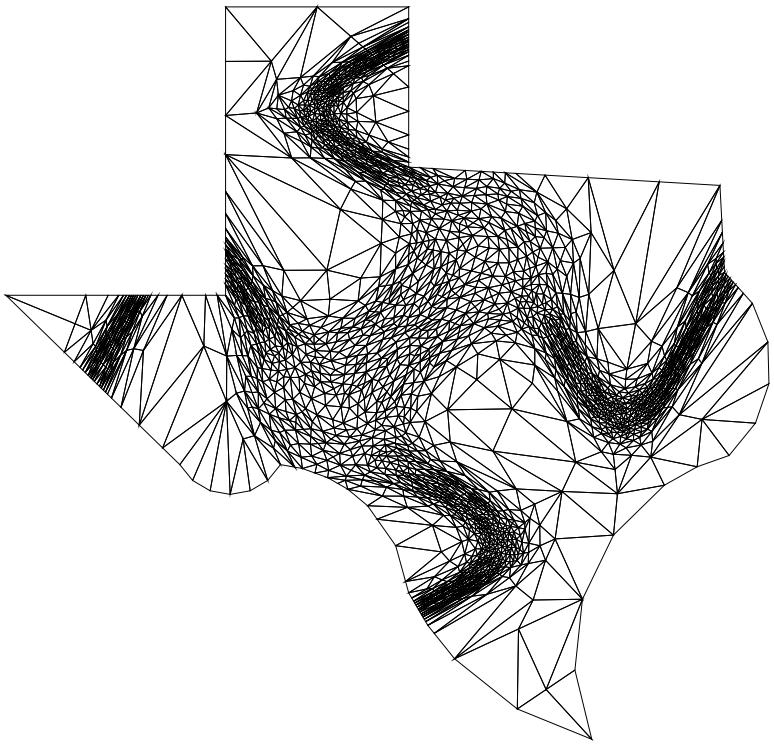
under the supervision of the principal investigators.

Besides the original software, the package Ani2D incorporates a number of public libraries such as BLAS, LAPACK, UMFPACK and AMD.

[1]Los Alamos National Laboratory, Theoretical Division, MS-284, Los Alamos, NM 87545, USA.
[2]Institute of Numerical Mathematics RAS, 8 Gubkina St., 119333 Moscow, RUSSIA.

Contents

MESHING PACKAGEs



Ani2D-AFT version 2.2 "*Forget-me-not*"

Flexible Triangular Mesh Generator Using Advanced Front Technique

User's Guide for libaft2D-2.2.a

1 Basic features of the library

The C package Ani2D-AFT is a part of the package Ani2D. Ani2D-AFT was developed by Alexander Danilov under the supervision of Yuri Vassilevski. It generates triangular meshes in arbitrary 2D domains.

The library *libaft2D-2.2.a* can be used in other packages. Its basic features are listed below.

Domain type : single or multiple component, simply or multiply connected finite domains

Boundary type : piecewise smooth

Domain data input : set of linear/curvilinear intervals representing the boundary, or the boundary mesh

Number of mesh elements : non-limited

Data format : double precision (for real values) or integer (for integer numbers) arrays. Enumeration starts from 1.

2 Analytical representation of the boundary

If the domain boundary is given analytically, the user should provide additional routine describing this boundary.

```
 external userboundary
 call registeruserfn( userboundary )
 ...
 call aft2dboundary( Nbv, bv, Nbl, bl, bltail,  h,
&           nv, vrt,
&           nt, tri, material,
&           nb, bnd,
&           nc, crv, iFNC )
```

2.1 Input

The piecewise smooth boundary is represented in terms of the union of a finite number of intervals. Each interval is a smooth curve. It may be split into several subcurves. End points of the intervals are called the V-points. If the domain is multi-connected, each simply connected subdomain has a boundary composed of the given intervals. The input domain is specified by three arrays. Array `bv` describes all V-points, whereas arrays `bl` and `bltail` describe the intervals. `Nbv` is the number of V-points, i-th column in the array `bv(2,Nbv)` has coordinates of the i-th V-point, $i = 1, \ldots,$ `Nbv`. `Nbl` is the number of intervals, i-th column of the integer array `bl(7,Nbl)` describes the i-th interval, $i = 1, \ldots,$ `Nbl` using 7 parameters. The integers from the 7-parameter are explained below.

1. The index of the V-point at which the interval begins.

2. The index of the V-point at which the interval terminates.

3. If the interval is linear, it is zero. Otherwise it is a positive number defining the type of parameterization in the user defined function `userboundary` which is in the file `crv_model.c`.

4. A dummy integer.

5. The label of the interval. It does not affect the mesh generation. All the mesh edges from the interval will inherit this number.

6. The index of the subdomain, for which the interval is a boundary part.

7. The slit marker. If the interval is the outer part of the domain boundary, it is zero. Otherwise, it is the index of a subdomain that shares the interval with the subdomain indicated by the 6th parameter.

The i-th column of the array `bltail(2,Nbl)` has two zeros when the interval is linear. Otherwise this column contains two parameters corresponding the starting and the terminal points of the interval. These parameters defined the Cartesian coordinates of the V-point.

The rules for interval specification are as follows:

1. When moving along the interval from the starting point to the terminal point, the subdomain is located on the right. In other words, the intervals are given clock-wise.

2. For slit intervals, the subdomain indicated by the 6th parameter must be located on the right.

3. For slit intervals shared by two subdomains, the order of the V-points is arbitrary.

4. Coordinates of V-points defined via parametric functions using the data in array `bltail(2,Nbl)` may be different from the corresponding entries in the array `bv`. The latter entries are not used in this case.

Boundary parameterization is to be defined by the user in the file `crv_model.c`. The name of the user routine must be registered in the library before the call of `aft2dboundary`:

```
external userboundary
call registeruserfn( userboundary )
...
call aft2dboundary( ... )
```

In this example, function `userboundary` is registered and is used in meshing the boundary. An example of the user defined function describing the complement of a wing to the unit square is in file `crv_model.c`.

The generator produces a quasi-uniform mesh of the given mesh size `h`.

2.2 Output

The number of mesh nodes is `nv`, their Cartesian coordinates are stored in the array `vrt(2,*)`. The number of mesh triangles is `nt`, the connectivity list of triangles is stored in the array `tri(3,*)`, the triangle materials (labels) are `material(*)`. The number of mesh boundary edges is `nb`. The first two numbers in each column of `bnd(4,*)` are node indexes of the boundary edges. The third number is the parameterization identificator: 0 for non-parameterized edge (linear segment), positive for parameterized edge with parameterization data in arrays `crv(*,n)` and `iFNC(n)`. The fourth number is a boundary label/identificator. The number of curved (parameterized) boundary edges is `nc`, their parameterization is stored in the corresponding column of `crv(2,*)` and the corresponding entry of `iFNC(*)`. For example, the j-th parameterized edge uses parameter `crv(1,j)` for its starting point and parameter `crv(2,j)` for its terminal point, as well as the identificator `iFNC(j)` of the function to be used in the computation of the Cartesian coordinates of interior points of the edge.

3 Grid representation of the boundary

If the domain boundary is given by a set of mesh edges, the user must call the following routine. There is no need in a user defined function for the boundary parameterization. No dummy function need to be written and be registered in the library.

```
 call aft2dfront(
&           Nbr, brd, Nvr, vbr,
&           nv, vrt,
&           nt, tri, material,
&           nb, bnd)
```

3.1 Input

The domain boundary is described by mesh edges. The total number of the boundary edges is `Nbr`, the number of boundary nodes is `Nvr`. The i-th column of the array `vbr(2,Nvr)` contains Cartesian coordinates of the i-th boundary node. The i-th column of the array `brd(2,Nbr)` contains the node indexes of the starting and terminal points of the edge.

There are two methods for representing the boundary mesh.

The first method assumes that `Nbr > 0`. The boundary nodes and edges are stored in an arbitrary order. While moving along an edge from the starting point to the terminal point, the subdomain must be located on the right.

The second method assumes that `Nbr = 0`. Only boundary nodes are used to represent the boundary, but their order is important. In this case, `brd` is not used.

The rules for the boundary mesh specification are as follows:

1. The boundary is a union of loops. The loops are stored in `vbr` in the sequential order.

2. In each loop, the first node and the last node must be identical. This fact is used to distinguish different loops.

3. When moving to the next node within one loop, the subdomain must be located on the right.

4. Loops can overlap in any way, in this case the shared node(s) must be presented in each loop.

The resulting mesh will have the trace at the boundary matching to the boundary grid `vbr, brd`. The local mesh size depends on the distance to the boundary: the farther from the boundary, the coarser the mesh is.

Two illustrative examples of using the initial front data may be found in directory `PackageAFT/examples`.

3.2 Output

The number of mesh nodes is `nv`, their Cartesian coordinates are stored in the array `vrt(2,*)`. The number of mesh triangles is `nt`, the connectivity list of triangles is stored in the array `tri(3,*)`, the triangle materials (labels) are `material(*)`. The number of mesh boundary edges is `nb`. The first two numbers in each column of `bnd(4,*)` are node indexes of the boundary edges. The third number is not used. The fourth number is the boundary label/identificator.

4 Two examples

In this section we present two meshes generated by the package as well as data specifying the domain.

The first example uses analytical representation of the boundary. We present the piece of FORTRAN code `src/Tutorials/PackageAFT/main_boundary_wing.f` producing the mesh shown in Fig.1.

```
C complement of a wing NACA0012 to the unit square
      double precision bv(2,7),bltail(2,8)
      integer          Nbv,Nbl,bl(7,8)
C number of boundary nodes and number of boundary edges
      data             Nbv/7/,Nbl/8/
C boundary nodes
      data             bv/0,0, 0,1, 1,1, 1,0, .4,.5, .6,.5, 1,.5/
C outer boundary edges
      data             bl/1,2,0,-1,-1,1,0, 4,1,0,-1,-1,1,0,
     &                    2,3,0,-1,1,1,0,  7,4,0,-1,1,1,0,
     &                    3,7,0,-1,1,1,0,
C slit  boundary edges
     &                    6,7,2,0,11,1,1,
C wing  boundary edges
     &                    6,5,1,-1,2,1,0,  5,6,1,-1,2,1,0/
C curved data for each outer boundary edge
      data             bltail/0,0, 0,0, 0,0, 0,0, 0,0, 0,1, 0,.5, .5,1/
```

```
      integer nv,nt,nb,nc
      double precision crv(2,nbmax), vrt(2,nvmax)
      integer          iFNC(nbmax),material(ntmax),
     &                 tri(3,ntmax),bnd(4,nbmax)
      double precision h

c to pass the name of the user function userBoundary (crv_model.c)  to the library
c function in crv_model.c
      external userboundary
c register the name to be used in the library
      call registeruserfn( userboundary )

C  mesh step of the quasi-uniform mesh to be generated
      h = 0.02
C Generate quasiuniform mesh with meshstep h
      call aft2dboundary(
C geomtric data
     &           Nbv, bv, Nbl, bl, bltail,  h,
C mesh data on output
     &           nv, vrt, nt, tri, material, nb, bnd, nc, crv, iFNC )
```

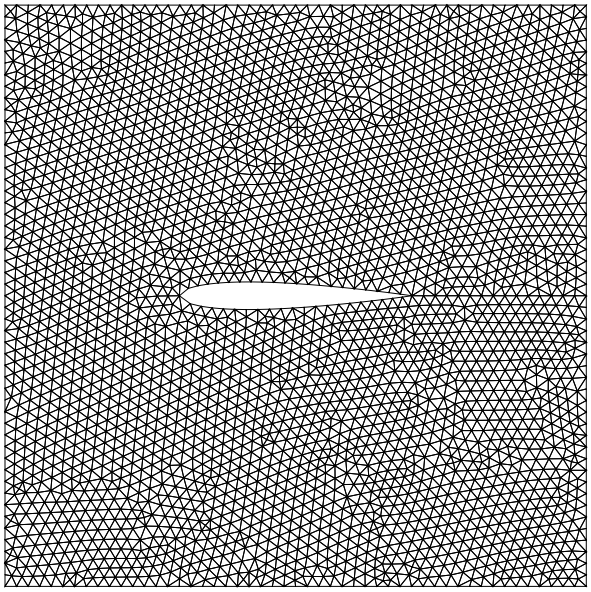


Figure 1: Mesh around the wing.

The second example uses discrete representation of the boundary. We present the piece of FORTRAN code `src/Tutorials/PackageAFT/examples/main_front1.f` and the data file `src/Tutorials/PackageAFT/examples/front1` producing the mesh shown in Fig.2.

```
c mesh generator data specifying domain via in the segment format
      double precision vbr(2,nbmax)
      integer          Nbr,Nvr,brd(2,nbmax)

      integer          nv,nt,nb
      double precision vrt(2,nvmax)
      integer          material(ntmax),tri(3,ntmax),bnd(4,nbmax)

C Read input file that contains coordinates of boundary points
      open(1,file='../src/Tutorials/PackageAFT/examples/front1')
      read(1,*) Nvr, Nbr
      do i = 1, Nvr
        read(1,*) (vbr(j,i),j=1,2)
      end do
      do i = 1, Nbr
        read(1,*) (brd(3-j,i),j=1,2)
      end do
      close(1)

C Generate a mesh  starting  from boundary mesh
      call aft2dfront(
C segment data
     &           Nbr, brd, Nbr, vbr,
C mesh data on output
     &           nv, vrt,
     &           nt, tri, material,
     &           nb, bnd)
```

The contents of file `front1` is

```
 518 518
  0.  0.064933
  0.002293  0.059187
  0.007467  0.055733
  0.01092  0.050573
  ....
  0.648853  0.1954
 3 4
 4 5
 5 6
  ....
 235 236
```

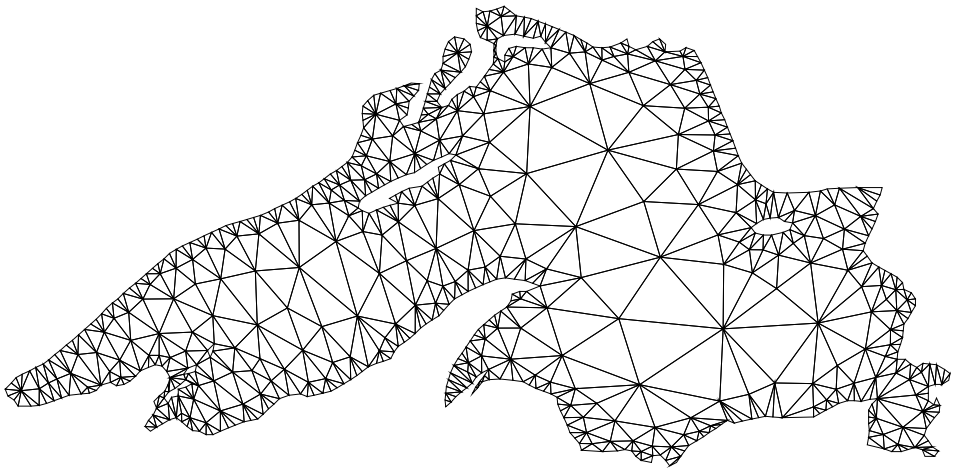


Figure 2: Mesh in complex domain.

Ani2D-RCB version 2.2 "*Windflower*"

Flexible Mesh Refining/Coarsening Tool Using Marked Edge Bisection

User's Guide for librcb2D-2.2.a

1 Basic features of the library

The FORTRAN77 package Ani2D-RCB is a part of the package Ani2D. Ani2D-RCB was developed by Vadim Chugunov and Yuri Vassilevski. It is designated for hierarchical refining and coarsening of arbitrary triangular 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. An example of calling program is given in `Tutorials/PackageRCB/main.f`.

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(2,*)`. The number of mesh triangles is `nt`, the connectivity list of triangles is stored in the array `tri(3,*)`, the triangle materials (labels) are `material(*)`. The number of mesh boundary edges is `nb`. The first two columns of `bnd(4,*)` are node indexes of the boundary edges. The third column of `bnd(4,*)` is dummy. It may be used for curve-linear edges. The fourth column of `bnd(4,*)` is a boundary label/identificator.

2 Initialization

The initialization tool (`aux.f`) prepares auxiliary structure which defines how to bisect the triangles. In `InitializationRCB` all input triangles are marked for bisection according to specific rule. The rule is based on bisecting the longest edge of each triangle. No care of mesh conformity should be taken within this rule. The user may change the rule in `InitializeMeshData`. In actual refinement, the user is free to mark for refinement any subset of triangles.

```
iERR     = 0
call InitializeRCB (nt, ntmax, vrt, tri, MaxWi, iW, iERR)
If(iERR.GT.0) stop 'size of iW is too small'
```

The size of work memory `iW` for `InitializeRCB, LocalRefine, LocalCoarse` should be at least `11*ntmax+7`.

3 Refinement

The refinement tool `LocalRefine` (`refine.f`) refines the input triangulation according to the user defined routine `RefineRule`. The name of the latter routine is the input parameter of `LocalRefine`. The output triangulation is in place of the input triangulation. The by-product of `LocalRefine` is the logical data array `history(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, tri, bnd, material,
     &        RefineRule, ilevel,
     &        maxlevel, history,
     &        MaxWi, iW,
     &        iPrint, iERR)
         If(iERR.GT.0) stop 'iERR.gt.0 in LocalRefine'
      End do
```

The key control of the refinement process is the user defined routine `RefineRule`. Here, the user defines which triangles have to be refined and how they must be refined, depending on each triangle data and the current level of refinement. 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 triangle `i`; if the marker is 1, then the user wants to refine triangle by single bisection; if the marker is 2, then the user wants to refine triangle by two levels of bisection into four similar subtriangles.

```
      Subroutine RefineRule (nt, tri, vrt, verf, ilevel)
      ...
      If (ilevel .le. 0) then
        Do i = 1, nt
          verf(i) =  2 ! two levels of bisection (keep the shape)
        End do
      Else  ! refine towards the diagonal y=x
        Do i = 1, nt
          xy1 = vrt(2,IPE(1,i)) - vrt(1,IPE(1,i))
          xy2 = vrt(2,IPE(2,i)) - vrt(1,IPE(2,i))
          xy3 = vrt(2,IPE(3,i)) - vrt(1,IPE(3,i))
          xy  = (xy1 **2 + xy2 **2) *
     &          (xy1 **2 + xy3 **2) *
     &          (xy2 **2 + xy3 **2)
          If (xy .eq. 0) then ! at least one vertex belongs to y=x
            verf(i) =  2 ! two 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 triangulation according to the user defined routine `CoarseRule`. The name of the latter routine is the input parameter of `LocalCoarse`. The output triangulation is in place of the input triangulation. The by-product of `LocalCoarse` is the logical data array `history(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, tri, bnd, material,
     &        CoarseRule, ilevel,
     &        maxlevel, history,
     &        MaxWi, iW,
     &        iPrint, iERR)
         If(iERR.GT.0) stop 'iERR.gt.0 in LocalCoarse'
      End do
```

The key control of the coarsening process is the user defined routine `CoarseRule`. Here, the user defines which triangles have to be merged and how they must be merged, depending on each triangle 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 coarse triangle `i`; if the marker is 1, then the user wants to merge triangle with its neighbor; if the marker is 2, then the user wants to merge triangle with its neighbor and then merge the result one more time so that the result be similar to triangle `i`.

```
      Subroutine CoarseRule (nE, IPE, XYP, verf, ilevel)
      ...
      If (ilevel .le. 0) then
        Do i = 1, nt
          verf(i) =  2 ! two levels of merging (keep the shape)
        End do
      Else ! coarse towards the diagonal y=x
        Do i = 1, nt
          xy1 = vrt(2,IPE(1,i)) - vrt(1,IPE(1,i))
          xy2 = vrt(2,IPE(2,i)) - vrt(1,IPE(2,i))
          xy3 = vrt(2,IPE(3,i)) - vrt(1,IPE(3,i))
          xy  = (xy1 **2 + xy2 **2) *
     &           (xy1 **2 + xy3 **2) *
     &           (xy2 **2 + xy3 **2)
```

```
   If (xy .eq. 0) then
     verf(i) =  2 ! two levels of merging (keep the shape)
    else
     verf(i) =  0 ! no need to coarse
   End if
 End do
End if
End
```

Ani2D-MBA version 2.2 "*Stone Flower*"[3]

Flexible Mesh Generator Using Metric Based Adaptation

User's Guide for libmba2D-2.2.a

[3]This is our first package. It is hard to carve a flower from a stone.

1 Introduction

The Fortran package Ani2D-MBA (2D metric based adaptation) is a part of the package Ani2D developed by Konstantin Lipnikov and Yuri Vassilevski. Ani2D is designated for the approximate solution of 2D boundary value problems on adaptive anisotropic triangular meshes. Ani2D-MBA package generates conformal triangular meshes which are quasi-uniform in a given metric. The metric may be defined either at every point via an analytical formula or only at mesh nodes. In the first case, the user may generate a mesh with desirable properties. In the second case, the given metric is assumed to be piecewise linear and the resulting mesh is adapted to it.

The library *libmba2D-2.2.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 triangles (made by hands), or a very fine mesh produced by another mesh generator. Ani2D-MBA *changes* the initial mesh through a sequence of local modifications. This approach provides a stable algorithm for generating strongly anisotropic grids. Generalization of this approach to tetrahedral meshes has been successfully implemented by us. The package Ani3D-MBA is freely available at `sourceforge.net/projects/ani3d`.

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. It also presents a synthetic example showing the mesh generation process in detail.

2 Copyright and Usage Restrictions

This software is released under the GNU GPL Licence. 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.

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 *metric2D* generating a nodal metric has been moved to a separate library *liblmr2D-2.2.a*. As the result, the list of input parameters in the main routines has been modified. The discrere solution (parameter `SOL`) has been replaced by the `Metric` and parameter `Lp` has been removed.
- The user supplied routine *calCrv* has been included in the list of input paratemers, see the dummy parameter `CrvFunction`. The library constains an empty routine *CrvFunction_ani* for convience.

4 Description of Ani2D-MBA

The main goal of package Ani2D-MBA is to produce a mesh with a prescribed number of triangles which is as much quasi-uniform in a given tensor metric as possible. For example, when the metric is isotropic and constant, Ani2D-MBA may generate a mesh consisting of equilateral triangles unless the domain boundary has very small angles. 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 triangles of the same size (measured in the given metric) has quality 1.

4.1 Structure of the package

The main Fortran 77 subroutines of Ani2D-MBA are *mbaAnalytic* and *mbaNodal* located in files `mba_analytic.f` and `mba_nodal.f`, respectively. The depending subroutines are contained in the other files in directory `src/aniMBA`. The examples using these subrotuines are in directory `src/Tutorials/PackageMBA`. The files

```
main_analytic.f    main_nodal.f    main_solution.f    time.f
```

may be modified by the user. The program in the first file generates a mesh using an analytic metric. The program in the second file does the same job using a user-defined metric at mesh nodes. The program in the third file uses a user-supplied solution defined at mesh nodes to generate first a metric and then an adapted mesh. A few examples of files `main_analytic.f` and `main_solution.f` can be found in directory `src/Tutorials/PackageMBA/examples`.

In addition to that, these files contain routine *CrvFunction* describing a parametrization of curved boundaries and routine *MetricFunction* describing the metric. Some of the models do not have curved boundaries. In this case package routine *CrvFunction_ani* is used.

File `time.f` is a wrapper for the system call *etime* that computes CPU time. Generally speaking, this routine depends on the operational system.

For user convenience, package Ani2D-MBA is equipped with auxiliary files

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

Their purpose is to facilitate loading and saving of meshes. For visualization purposes, a simple service library *libview2D-2.2.a* was created. Routine `draw()` from *libview2D-2.2.a* is used in `main_analytic.f` and `main_nodal.f` for generation of PostScript figures. The files

```
aniMBA/Makefile   PackageMBA/Makefile
```

build the library and examples, respectively, 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 Ani2D-MBA are located in directory `doc`.

4.2 Basic things the user should know

The package provides two methods to control the mesh generation. The first method is based on an analytic metric. The second method is based on a piecewise linear interpolant of the user-defined metric. This discrete metric is defined at mesh nodes. The package contains a few routines for accurate interpolation of functions defined on edges or over triangles to mesh nodes (see Sec. 10).

The package is encapsulated in the two basic routines *mbaAnalytic* and *mbaNodal* corresponding to the above methods. The comments in file `src/aniMBA/mbaNodal.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, Fortran routines and control parameters.

- The input *data files* are the files containing coordinates of mesh nodes, connectivity tables for triangles and boundary edges, a parametrization of curved boundary edges, a list of fixed mesh nodes, a list of fixed mesh edges, and a list of fixed elements. The lists of fixed points, edges and elements may be empty. The list of boundary edges may be also empty. In this case, the boundary edges will be recovered by package routines. A good example illustrating format of the data file is `data/star.ani` (see Section 5 for a more complicated example). A data file can be accessed via routine *loadMani*.

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

- The input *routines* are the Fortran 77 routines used by the package in the process of mesh generation. They are located in files `main_analytic.f` and `main_nodal.f`.

  An analytical metric has to be supplied for routine *mbaAnalytic*. The user should change function *MetricFunction_user* located in file `PackageMBA/main_analytic.f`. For more detail, we refer to comments in this file.

A routine *CrvFunction* has to be supplied for both routines *mbaAnalytic* and *mbaNodal* if the user model has curved boundaries. If the user model does not have curved boundaries, the empty routine *CrvFunction_ani* may be used. *CrvFunction* describes parameterizations of curved boundaries. There is a way to avoid writing this routine. The user may fix the boundary points of the initial mesh provided that they give accurate representation of the boundary. Then, the final mesh approximates curved boundaries with the same accuracy as the initial mesh does.

- The input *control parameters* are the numbers that control the mesh generation. They are defined in files `main_analytic.f` and `main_nodal.f`. The input control parameters are the following variables:

```
nEstar   - [integer] the desired number of triangles
MaxQItr  - [integer] the maximal number of local grid modifications
Quality  - [real*8]  the target quality for the final grid
                     (a positive number between 0 and 1)
MaxSkipE - [integer] the maximal number of skipped triangles
```

  The mesh generation is an iterative process every step of which is a local modification of the current mesh. The stopping criterion for the iterative process is either the user requested final mesh quality (`Quality`) or the allowed number of local modifications (`MaxQItr`). We recommend to set `Quality` to a value between 0.5 and 0.8 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 100.

4.3.1 Mesh representation

Understaning details of the mesh format is one of the first steps in discovering capabilities of Ani2D-MBA . The mesh presentation includes:

```
nP - [integer] the number of points
nF - [integer] the number of boundary and interface edges
nE - [integer] the number of triangles

XYP(2, *) - [real*8]  the Cartesian coordinates of mesh points
IPE(3, *) - [integer] connectivity list of triangles
lbE(*)    - [integer] material indentificator (a positive number)

IPF(4, *) - [integer] column 1 & 2 - connectivity list of boundary edges
                      column 3     - number in the parametrization list ParCrv:
                                        0   : this edge is a linear segment
                                        n>0 : ParCrv(*, n) gives a parametrization
                                              of this edge and iFnc(n) gives
                                              a function number for computing the
                                              Cartesian coordinates (see calCRv())
                      column 4     - boundary identificator
                                     (example: unit square has 4 boundaries which
                                               may have different identificators)

nPv - [integer] the number of fixed points
nFv - [integer] the number of fixed edges
nEv - [integer] the number of fixed triangles

IPV(*) - [integer] list of fixed points
IFV(*) - [integer] list of fixed edges
IEV(*) - [integer] list of fixed triangles

CrvFunction(tc, xyc, iFnc) - user-created routine:
   tc     - [input]  parametric coordinate of point xyc
```

```
      xyc(2) - [output] Cartesian coordinate of the same point
      iFnc   - [input]  function number associated with a boundary

   ParCrv(2, *) - [real*8] linear parameterization of curvilinear edges
                           column 1 - parameter for the starting point
                           column 2 - parameter for the terminal point

                  parameters for interior points of the edge are computed by
                  linear interpolation between parameters at edge ends

                  Cartesian coordinates are computed by user-given
                  formulas defined in calCrv().

   iFnc(*) - [integer] function number for computing the Cartesian coordinates
```

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/  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/`. 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
```

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-ana` or `make run-nod`. The output may look like:

```
$ cd bin; ./mbaAnalytic.exe

Loading mesh ../data/wing.ani

STONE FLOWER! (1997-2009), version 2.2
Target: Quality 0.70 with    2000 triangles for at most   15000 iterations

status.fd: +1    [ANIForbidBoundaryElements] [user]
status.fd: +2    [ANIUse2ArmRule]            [system]
status.fd: +8    [ANIDeleteTemporaryEdges]   [system]

Maximal R/r = 0.193E+03  (R/r = 2 for equilateral triangle), status.fd:  11
ITRs:     0  Q=0.7635E-03   #P #F #E:     596     178    1037   tm=   0.01s
ITRs:  3486  Q=0.7000E+00   #P #F #E:     989     120    1874   tm=   0.41s
Maximal R/r = 0.397E+01  (R/r = 2 for equilateral triangle), status.fd:  11

Saving  mesh save.ani
```

First, some of the input control parameters are printed out. Then, the quality of the current mesh and the numbers of vertices, edges and triangles are printed. Additional output goes into Postscript files `mesh_initial.ps` and `mesh_final.ps` containing figures of initial and final meshes, respectively. The files are located in directory `bin`. One way to check the contents of these files is to run

```
$ make gs-ini gs-fin
```

The program loads the input file `../data/wing.ani`. The user may either to change the name of the input file in the mesh loader:

```
 Call loadMani(
&      nP,  MaxP,  nF,  MaxF,  nE,  MaxE,
&      nPv, MaxPV, nFv, MaxFV, nEv, MaxEV,
&      XYP, IPF, IPE, IPV, IFV, IEV, lbE,
&      ParCrv, iFnc,
&      "../data/wing")
```

or to use one the examples from directory `PackageMBA/examples`. The user may play with the input control parameters in file `PackageMBA/main_analytic.f` and with the metric defined in routine *MetricFunction_user*. For instance, changing the metric

$$\mathcal{M}(x,y) \equiv \begin{bmatrix} F(x,y) & H(x,y) \\ H(x,y) & G(x,y) \end{bmatrix}$$

the user will learn how to control the shape of triangles.

6 A synthetic example

In this section, we describe in detail a process of creating a new model and generating a quasi-uniform mesh.

Let us assume that the user wishes to generate a quasi-uniform triangulation of a domain that is the union of two circles with the radius 0.2 and centers (0.2,0.5), (0.8,0.5), respectively, and the rectangle defined by vertices (0.2,0.45), (0.2,0.55), (0.8,0.55), and (0.8,0.45). The domain is shown in Fig. 3.

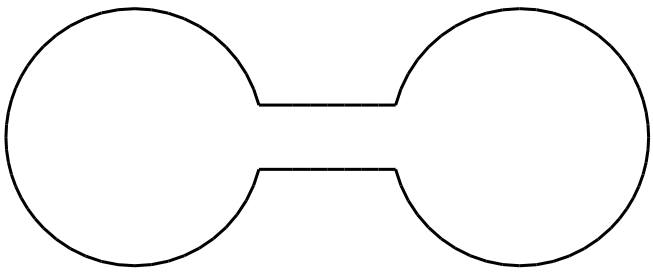


Figure 3: The domain to be meshed.

The user has to write routine *CrvFunction_user*. If the user wishes to use mesh loader *loadMani*, he has to create an input data file. Below, we explain how to produce all these data from scratch.

Step 1. First, we chose a parameterization model. The shape of the domain dictates a natural choice for the parameterization of curvilinear parts of the boundary: each circle is parametrized by trigonometric functions. The input routine *CrvFunction_user* may be as follows:

```
      Subroutine CrvFunction_user(tc, xyc, iFnc)
C =============================================================
C  The routine computes the Cartesian coordinates of point
C  xyc from its parametric coordinate tc.
C
C  tc     - the given parametric coordinate of point
C  xyc(2) - the Cartesian coordinate of the same point
C  iFnc   - the function number for computing
C
C  On input :  tc, iFnc
C  On output:  xyc(2)
C =============================================================
      Real*8  tc, xyc(2), L, H, R

      L = 0.3D0
      H = 0.1D0
      R = 0.2D0
      If(iFnc.EQ.1) Then
         xyc(1) = 5D-1 + L - R * dcos(tc)
         xyc(2) = 5D-1 + R * dsin(tc)
      Else If(iFnc.EQ.2) Then
         xyc(1) = 5D-1 - L + R * dcos(tc)
         xyc(2) = 5D-1 - R * dsin(tc)
      Else
         Write(*,'(A,I5)') 'Undefined function =', iFnc
         Stop
      End if
      Return
      End
```

This code can be found in `PackageMBA/example/main_analytic_sport.f`.

Step 2. Second, we create input data file containing an initial coarse mesh. It is easy to observe that a simple mesh consisting of 12 triangles will be sufficient, see Fig. 4.

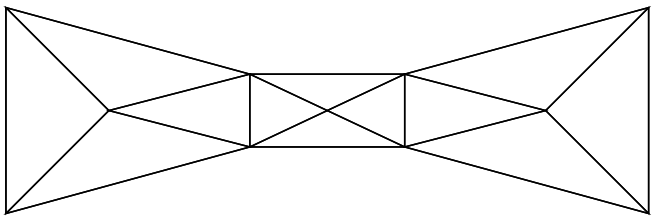


Figure 4: The initial coarse mesh.

The file `data/sport.ani` has a header (9 lines), followed by the list of points (11 points), list of egdes (8 edges), list of triangles (12 edges) and the list of curved edges (6 edges):

```
T points:             11 (lines  10 - 20)
T edges:               8 (lines  23 - 30)
T elements:           12 (lines  33 - 44)
T curved edges:        6 (lines  47 - 52)
```

```
T fixed points:        0
T fixed edges:         0
T fixed elements:      0

 11 # of points
 0.500000000000000      0.500000000000000
 0.800000000000000      0.500000000000000
 0.200000000000000      0.500000000000000
 0.606350832689630      0.550000000000000
 0.941421356237310      0.641421356237310
 0.941421356237310      0.358578643762690
 0.606350832689630      0.450000000000000
 0.393649167310370      0.450000000000000
 5.857864376269000E-002 0.358578643762690
 5.857864376269000E-002 0.641421356237310
 0.393649167310370      0.550000000000000

 8 # of edges
  4  5 1 0 1
  5  6 2 0 1
  6  7 3 0 1
  7  8 0 0 2
 11  4 0 0 2
  8  9 4 0 3
  9 10 5 0 3
 10 11 6 0 3

 12 # of elements
 2  4  5  1
 2  5  6  1
 2  6  7  1
 2  7  4  1
 1  7  8  1
 1  8 11  1
 1 11  4  1
 1  4  7  1
 3  8 11  1
 3 11 10  1
 3 10  9  1
 3  9  8  1

 6 # of curved edges
 0.252680255142080 2.35619449019230  1
 2.35619449019230  3.92699081698720  1
 3.92699081698720  6.03050505203750  1
 0.252680255142080 2.35619449019230  2
 2.35619449019230  3.92699081698720  2
 3.92699081698720  6.03050505203750  2

 0 # number of fixed points

 0 # number of fixed edges

 0 # number of fixed elements
```

- Some of the mesh nodes may be relocated or destroyed in a process of the mesh generation. However,

the domain boundary requires that four nodes (intersections of the rectangle with the circles) remain untouched. In order to provide this information, we need the list of fixed points. This list may be replaced by proper coloring of boundary edges. If a point is shared by two edges with different color, it will be automatically added to the list of fixed points.

- It is clear that there are eight boundary edges, six of them are part of the curvilinear boundary. It is reasonable to mark the edges with three labels associated with the rectangle and two circles. In each row, the first two entries are the node indices, the third entry is a reference to a list of curved edges, the fourth is dummy, and the fifth is a label (color) of the edge.
- The list of curved edges contains the starting and ending parameter values and a positive number corresponding to a function in routine *CrvFunction_user*. It is very important to guarantee that evaluation of *CrvFunction_user* gives exactly the same mesh coordinates as in the input file. For example, let us take `tc` and `iFnc` from the first row, i.e. `tc = 0.252680255142080` and `iFnc = 1`. Then, the routine should give the Cartesian coordinates of the 4th mesh node. The acceptable error is 10^{-8}.

Step 3. Third, we have to choose an analytic metric in which the final mesh be quasi-uniform. In other words, we have to write routine *MetricFunction_user*.

Step 4. Fourth, we set up the control parameters:

```
Integer   nEStar
Parameter(nEStar = 1000)

Real*8    Quality
Parameter(Quality = 8D-1)
```

Thus, we plan to generate a mesh with approximately 1000 triangles. Each of the triangles will be very close to an equilateral triangle.

Step 5. The final step is to collect all routines in a single file `PackageMBA/example/main_analytic_sport.f`, copy it to file `PackageMBA/main_analytic.f`, compile the package and execute the code (`# make exe run-ana`). We get the mesh shown in Fig. 5.

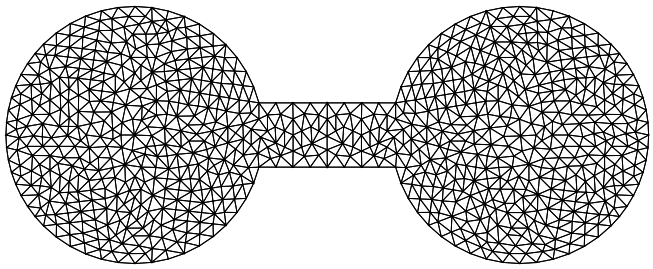


Figure 5: The final mesh.

7 Two more examples

The first geometric model is shown in Fig.6 (left picture). The left side of the model is partly curved. This part is parametrized as follows:

$$x = 0.2 - 2\,t\,(0.3 - t), \quad y = t, \qquad t \in [0, 0.3].$$

The curved part of the right side of the model is parametrized in a similar way:

$$x = 1 - 2\,(1-t)\,(t-0.7), \quad y = t, \qquad t \in [0.7, 1].$$

The file `PackageMBA/examples/main_analytic_square.f` demonstrates how to modify the input mesh data `data/square.ani` according to the solution `data/square.sol`.

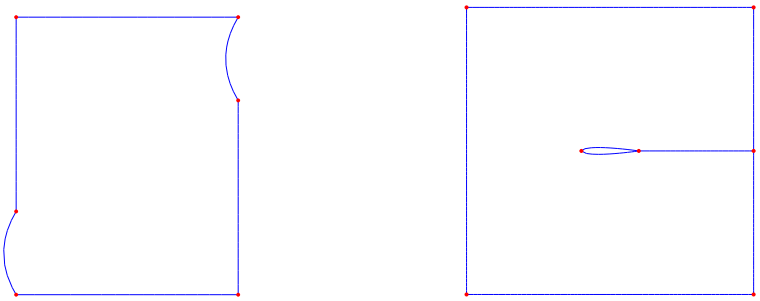


Figure 6: Two models: the square with curved sides and the wing.

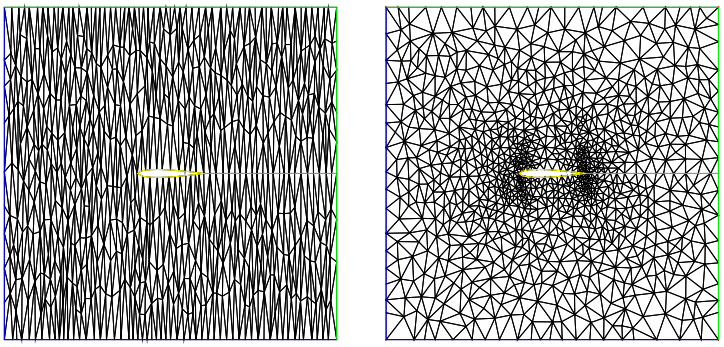


Figure 7: The initial and final meshes of the model `data/wing.ani`.

The second model is shown in Fig.6 (right picture). We use one parametrization for the wing and the other parametrization for the slit behind the tail. The file `PackageMBA/main_analytic_wing.f` defines a metric such that the final mesh refines isotropically towards the leading and trailing edges of the wing, see Fig.7.

8 Useful features of Ani2D-MBA , version 2.2

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

1. The initial mesh may be tangled. In this case, the user may add `ANIUntangleMesh` defined in `src/aniMBA/status.fd` to the input parameter `status` to untangle the input mesh.
2. Using two packages Ani2D-MBA and Ani2D-LMR , it is possible to produce meshes minimizing different maximum and L^p-norms of the interpolation error, $p > 0$.
3. 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. Fig.8 illustrates this feature. The fixed boundary points (red dots) prevent sharp boundary from smearing. (*The initial mesh was found on the webcite of Jonathan Shewchuk.*)
- The user may freeze boundary edges 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 vertices of corners smaller than 30° are marked automatically as fixed points.

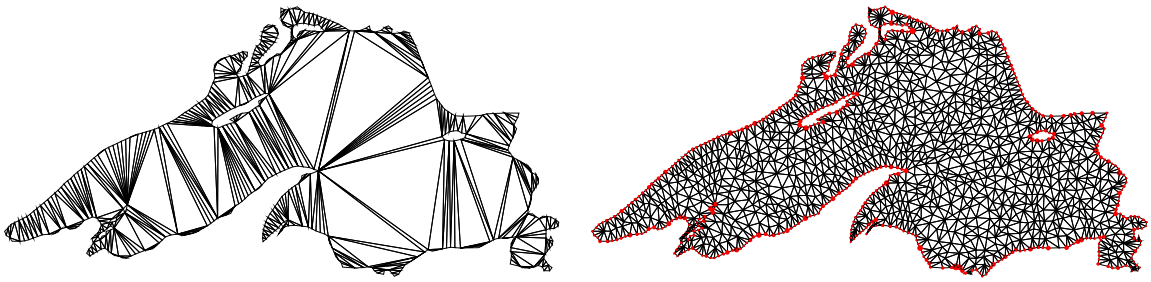


Figure 8: The initial and final meshes of the model `data/country.ani`.

4. The library *libmba2D-2.2.a* contains routine *DG2P1* which maps a discontinuous piece-wise linear function defined on mesh edges 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 rotines *listX2Y* which create connectivity lists $X \to Y$ for mesh objects X and Y such as elements, edges, bounary edges, and points (see `src/aniMBA/utils.f` for more detail).

5. Miscalenious code cleaning, documenting and improving. For example, we replaced Linpack routines by similar routines from package Lapack which is now a part of most Linux distributions. If the user have not installed this package, the neccessary routines are in directories `src/lapack` and `src/blas`. Double precision libraries `liblapack-3.0.a, libblas-3.0.a` are generated with the command "`make lib`" typed in `src/lapack` and `src/blas`, respectively.

9 How to use library libmba2D-2.2

Here we describe one of the main modules, *mbaNodal*, from the library `libmba2D-2.2.a`. The other module, *mbaAnalytic*, uses parameter `MetricFunction_user` (analytic metric) in place of parameter `Metric_user` in routine *mbaNodal*.

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

Most of the parameters were described in Section 3 (see file `src/aniMBA/mba_nodal.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 edges
P       MaxE - [integer] maximal number of triangles
U
T       nFv  - [integer] the number of fixed edges
        nEv  - [integer] the number of fixed triangles

P       IFV(nFv) - [integer] list of fixed edges
A       IEV(nEv) - [integer] list of fixed triangles
R
A       nEstar - [integer] the desired number of triangles
M
E       flagAuto - [logical] flag controling mesh generation:
T                            TRUE  - recover missing mesh elements
E                            FALSE - check that input data are complete
R
s       MaxSkipE - [integer] maximal number of skipped triangles
        MaxQItr  - [integer] maximal number of mesh modifications

        Quality  - [real*8]  desired quality of the final mesh

        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       nP - [integer] the number of points
N       nF - [integer] the number of boundary and interface edges
P       nE - [integer] the number of triangles
U
T       XYP(2, MaxP) - [integer] list of points
/       IPE(3, MaxE) - [integer] list of triangles
O       lbE(MaxE)    - [integer] material indentificator
U
T       IPF(4, MaxF)    - [integer] list of boundary and interface edges
P       ParCrv(2, MaxF) - [real*8]  parametrizations of curved edges
U       iFnc(MaxF)      - [integer] list of corresponding functions
T
        nPv      - [integer] the number of fixed points
        IPV(nPv) - [integer] list of fixed points
P
A       Metric_user(3,MaxP) - Real*8 array containing the metric defined
R              at mesh points. The metric is a 2x2 positive definite
A              symmetric tensor:
M
E                   M11   M12
T                   M12   M22
E
R               Each column of this array stores the upper triangular
s               entries in the following order: M11, M22, and M12.
```

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

status   - [integer] sum of positive numbers corresponding
                   to desired mesh properties (see status.fd)

Sol(MaxP) - [real*8] mesh function defined at points;
                     on output it is interpolated linearly
                     at new mesh points.

rW(MaxWr) - [real*8]  working array
iW(MaxWi) - [integer] another working array
```

10 Useful routines

The library *libmba2D-2.2.a* has a few routines that can be useful in many other projects. Most of the input parameters in these routines are explained above.

- Uniform mesh refinement and linear interpolation of nodal function `F(LDF, *)`. The size of working integer array `iW` is at least `3 nE + nP` where `nP,nE` are input values.

```
 Subroutine uniformRefinement(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      XYP, IPF, IPE, lbE,
&      CrvFunction_user, ParCrv, iFnc, IFE,
&      F, LDF, iW, MaxWi)
```

- `Delaunay` builds the Delaunay triangulation from the existing triangution by swapping edges in pairs of triangles. The size of working integer array `iW` is `6 nE + nP`. The working double precision array `rW` is not used at the moment.

```
 Subroutine Delaunay(
&           nP, nE, XYP, IPE,
&           MaxWr, MaxWi, rW, iW)
```

- `orientBoundary` orients the external boundary of the input mesh in such a way that the computational domain is located on the left when we move from the first edge point to the second one. In other words `IPF(1, *)` and `IPF(2, *)` are flipped if neccessary. The size of working integer array `iW` is `3 nE + 2 nF + nP`.

```
 Subroutine orientBoundary(
&      nP, nF, nE, XYP, IPF, IPE, iW, MaxWi)
```

- `DG2P1` maps a discontinuous piece-wise linear function defined on edges onto a continuous piece-wise linear function defined at vertices. We use the ZZ method for interpolation. We assume that each boundary node can be connected with an interior note by at most two mesh edges. The size of working integer array `iW` is `3 nE + 3 nP`. The size of working double precision array `rW` is `nP`.

```
 Subroutine DG2P1(
&           nP, nF, nR, nE, XYP, IPF, IPE, IRE,
&           fDG, fP1,
&           MaxWr, MaxWi, rW, iW)
```

- `listE2R` creates a connectivity list `IRE` for mesh edges. The routine counts mesh edges. For an element `E`, `IRE([1:3], E)` give indexes of three edges in the order defined by `IPE`. For example, the first edge is `[IPE(1,E), IPE(2,E)]`. The working integer arrays are `nEP(nP)` and `IEP(3 nE)` (see `src/aniMBA/utils.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 triangles 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 `9 nE` (see `src/aniMBA/utils.f` for more detail).

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

- File `src/aniMBA/utils.f` contains more routines for creating other connectivity lists such as edges to points, points to points, elements to boundary edges, elements to elements, etc. The routine *listConv* colvolutes two given connectivity lists. The routines *backReferences* and *reverseMap* create reverse connetivity 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.
- `smoothingMesh` applies the Laplacian smoothing to the mesh. For each mesh vertex, a new position is chosen based on local information (the position of its neighbors) and the vertex is moved there. The size of the working integer array `iW` is `2 nP + 3 nE + 90`.

```
      Subroutine smoothingMesh(
     &           nP, nE, XYP, IPE, MaxWi, iW)
```

11 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 a quality of the worst triangle in the mesh. In some cases, the shape of near-boundary triangles is driven mainly by the geometry. A possible remedy is either to increase the number `nEStar` of desired triangles or to fix a possible contradiction between the boundary and the metric. An example of such a contradiction is a quasi-uniform mesh in `data/Dam.*`. Another reason for low mesh quality may be strong jumps in the metric. If the metric is isotropic, the optimal triangles are equilateral ones. The triangle size is defined by the metric value. Therefore, the optimal size is changed strongly across lines of metric discontinuity. This property is hardly can be satisfied on a conformal mesh.

- Q. The mesh generator is stopped immediately with diagnostics saying that the parametrization is wrong.
  A. There is a contradiction between input data in arrays `ParCrv, iFnc` and `XYP`.
- Q. The number of triangles in the final mesh is never equal to `nEStar`.
  A. The equality is achieved if and only if `Quality = 1` and the computational domain may be covered by equilateral (in the user given metric) triangles. Apparently, it is possible only in very special cases.
- Q. Why *mba_analytic* and *mba_nodal* have so many input parameters?
  A. Next release of the package will have routines *mba_analytic_short* and *mba_nodal_short* with functionality close to that of main routines *mba_analytic* and *mba_nodal*, respectively, but with smaller number of input parameters. For example, lists of fix points and boundary triangles will be omitted.
- Q. Is it possible to use *libmba2D-2.2.a* in an adaptive loop?
  A. Yes. Use `make libs` to generate the library *libmba2D-2.2.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/*` for more detail).
- Q. Why does *libmba2D-2.2.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 *libmba2D-2.2.a* fails to generate a mesh.
  A. The authors 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.

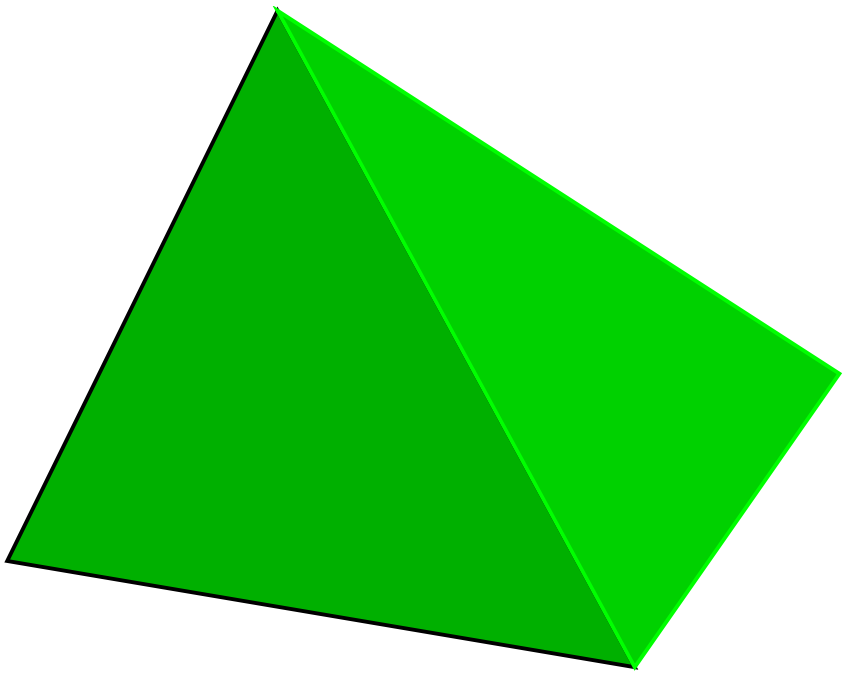
5. K.Lipnikov, Yu.Vassilevski, On a parallel algorithm for controlled Hessian-based mesh adaptation. Proceedings of 3rd Conf. Appl. Geometry, Mesh Generation and High Performance Computing, Moscow, June 28 - July 1, Comp. Center RAS, V.1, 2004, 154–166.

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.

DISCRETIZATION PACKAGEs



Ani2D-FEM version 2.2 "*Sunflower*"

Flexible Generator of Finite Elements Systems on Triangular Meshes

User's Guide for libfem2D-2.2.a

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, 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-2.2.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

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

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 unknown 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
FEM_CR1	the Crouzeix-Raviart finite element

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:

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

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.

Admissible values for arrays `templateR` and `templateC` are defined in file *fem2Dtri.fd*. Including this header file, the user may indicate a velocity degree of freedom as follow

```
templateR(i) = Vdof
```

The degrees of freedom on edges are indicated either by `Rdof` or `RdofOrient`. The former corresponds to a scalar unknown (e.g. a Lagrange multiplier in a hybrid mixed finite element) that has no orientation. The latter correponds to a vector unknown (e.g. the Raviart-Thomas finite element basis function) that has orientation. Finally, s degree of freedom associated with a mesh element is indicated by `Edof`.

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 is 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 traingle 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)
```

In the next version, the above routines will be replaced by a single routine described below. This new 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 `Tutorials/PackageFEM/mainSimple.f` demonstrates the approximate solution of the boundary value problem with continuous piecewise linear finite elements P_1:

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

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

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

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 *mbaAnalytic* from the library *libmba2D-2.2.a*. Second, the program generates the finite element system using the routines `BilinearFormVolume, LinearFormVolume` and `BoundaryConditions` from the library *libfem2D-2.2.a*.

4.2 Stokes problem

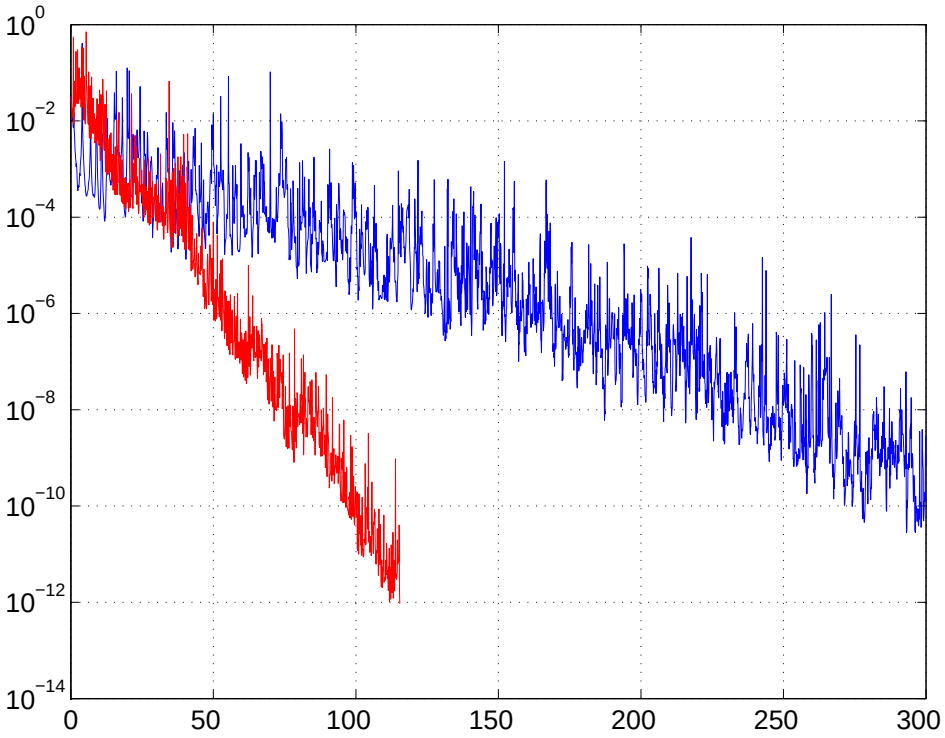
The program `Tutorials/PackageFEM/mainTemplate.f` demonstrates the approximate solution of the Stokes problem with $P_2 \times P_1$ pair of finite elements:

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

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 *mbaAnalytic* from the library *libmba2D-2.2.a*. Second, the program generates the finite element system using routine `BilinearFormTemplate` from the library *libfem2D-2.2.a*.

SOLUTION PACKAGEs



Ani2D-LU "*Twinflower*"

LU Factorization Solver for Sparse Systems

User's Guide for liblu-2.2.a

The C package Ani2D-LU is a simplified version of UMFPACK-4.1 and AMD packages developed by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. It is designated for the direct solution of sparse linear systems. Ani2D-LU is an independent part of the package Ani2D .

Examples of using Ani2D-LU in FORTRAN programs are given in files `src/aniLU/main.f`, `src/Tutorials/PackageLU/mainSolFemSys.f`. For detailed documentation, see `doc/lu_guide.pdf`.

Ani2D-ILU Version 2.2 "*Bellflower*"

Flexible Iterative Solver Using Incomplete LU Factorization

User's Guide for libilu-2.2.a

1 Basic features of the library

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

The basic features of library *libilu-2.2.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 precondtioners `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` (iluoo.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 `iluoo`. 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 verbocity 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-partlur)*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 imput
      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`.

Ani2D-INB Version 2.2 "*Starflower*"

Flexible Iterative Solver Using Inexact Newton-Krylov Backtracking

User's Guide for libinb-2.2.a

1 Basic features of the library

The FORTRAN package Ani2D-INB is an independent part of the package Ani2D . Ani2D-INB was developed by Alexey Chernyshenko under the supervision of Yuri Vassilevski. It is designated for the iterative solution of nonlinear systems. Ani2D-INB may be easily incorporated into any other software. The package interfaces the ILU precondtioners provided by the Ani2D-ILU package. The package Ani2D-INB is a deeply processed and essentially simplified version of the NITSOL package by Homer F. Walker.

The basic features of library *libinb-2.2.a* are listed below.

Iterative method : Inexact Newton-Krylov Backtracking (INB), with BiConjugate Gradient Stabilized (BiCGStab) iteration as the interior Krylov subspace solver

Preconditioners : Common interface with ILU0 and ILU2, the second order accurate ILU (provided by the Ani2D-ILU package).

Problem setting : User defined routine computing the nonlinear residual.

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

Typical memory requests : for systems with N equations INB needs 11 work vectors of dimension N, one solution vector and a room for preconditioner data. If the preconditioner is built by the Ani2D-ILU package, ILU0 requires the same storage as the CSR representation of the jacobian, ILU2 requires 2-5-fold storage.

2 Iterative solution

The iterative solver is Inexact Newton-Krylov Backtracking (INB) method with inner linear solve BiConjugate Gradient Stabilized method (BiCGStab). This is the Newton type method applicable to non-singular nonlinear systems. It requires two procedures: evaluation of the nonlinear residual function and (optional) precondition-vector evaluation. The preconditioner should be an approximation of the inverse jacobian matrix. The jacobian matrix is not required explicitly due to the finite-difference evaluation of the jacobian-vector product. The call of the method is

```
      external prevec, funvec
      ....

      call slInexactNewton(
     &     prevec, IPREVEC, iWprevec, rWprevec,
     &     funvec, rpar, ipar,
     &     N, SOL,
     &     RESID, STPTOL,
     &     rWORK, LenrWORK,
     &     INFO)
```

- `prevec` is the name of a precondition-vector multiplication routine. `IPREVEC, iWprevec, rWprevec` are arrays (Integer, Integer, Real*8, respectively) of user's data which may be passed to `prevec` and used there. Arrays `iWprevec, rWprevec` are recommended to keep the preconditioner bulk data (triangular factors, for instance). Array `IPREVEC` may contain control parameters or basic user data such as the system order and useful pointers. In the presented example, `IPREVEC` contains a single entry equal to the system order. The format of `prevec` coincides with that from the package Ani2D-ILU :

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

```
c Output
      Real*8  Y(*)
```

where X is the input vector and Y is the output vector. `ICHANGE` is the flag controlling the change of the preconditioner. It is useful when convergence stagnation occurs. `iW, rW` are Integer and Real*8 arrays, respectively, which store the preconditioner data. The package Ani2D-ILU provides two examples of routine `prevec` corresponding to two ILU preconditioners, `prevec0` (ilu0.f) and `prevec2` (iluoo.f). The details may be found in the user guide for Ani2D-ILU .

- `funvec` is the name of the user routine computing the nonlinear residual vector function and `ipar, rpar` are Integer and Real*8 arrays of user's data which may be passed to `funvec` and used there. The format of `funvec` is as follows:

```
      Subroutine funvec(n, xcur, fcur, rpar, ipar, itrmf)
c INPUT:
      Integer  n       ! dimension of vectors
      Real*8   xcur(*) ! current vector
      Real*8   rpar(*) ! double precision user-supplied parameters
      Integer  ipar(*) ! integer user-supplied parameters
c OUTPUT:
      Integer  fcur(*) ! nonlinear residual vector (zero for the solution)
      Integer  itrmf   ! flag for successful termination of the routine
```

  This routine calculates the nonlinear residual $F(X)$.

- `N` is order of system and length of vectors.
- `SOL` is the initial guess and the iterated solution (Real*8).
- `RESID` is the convergence criterion for the nonlinear residual.
- `STPTOL` is the stopping tolerance on the Newton's steplength
- `rWORK(LenrWORK)` is Real*8 working array which stores at least 11 vectors of size N.
- `LenrWORK` is the total length of `rWORK` which must be not less than `11*N`.
- `INFO` is the array of control parameters. On input: `INFO(1)` sets initial value for successful termination flag, `INFO(2)` sets the maximal number of linear iterations per Newton step, `INFO(3)` sets the maximal number of nonlinear iterations, `INFO(4)` sets the maximal number of backtracks, `INFO(5)` sets the printing level (0 none, 1 nonlinear residuals, 2 linear residuals) On output: `INFO(1)` is the value of the termination flag (successful termination corresponds to 0), `INFO(2)` is the number of performed linear iterations, `INFO(3)` is the number of performed nonlinear iterations, `INFO(4)` is the number of actual backtracks, `INFO(5)` is the number of performed function evaluations.

Examples of calling programs are in files `src/Tutorials/PackageINB/main_simple.f`, `src/Tutorials/PackageINB/main_bratu.f`, `src/Tutorials/MultiPackage/StokesNavier/main.f`.

SERVICE PACKAGEs

Ani2D-LMR version 2.2 "*Cornflower*"

Local Metric Recovery

User's Guide for liblmr2D-2.2.a

1 Introduction

The FORTRAN package Ani2D-LMR is developed by Konstantin Lipnikov and Yuri Vassilevski. It is designated for generating continuous tensor metrics. The tensor components are piecewise linear functions defined on nodes of a given triangular mesh. The generated metric may be used further in Metric Based Adaptation package Ani2D-MBA .

The input data for metric generation is either a discrete solution defined at mesh nodes, or cell-based or edge-based errors estimates.

The library *liblmr2D-2.2.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

3 Description of Ani2D-LMR

3.1 General structure of package

The package Ani2D-LMR consists of five FORTRAN files and one include file

```
CellEst2Metric.f  EdgeEst2Metric.f  Func2Metric.f  Func2MetricZZ.f  Lp_norm.f  metric.fd
```

The routines in this files implement one of the following basic tasks:

1. Recovery of a nodal metric from a discrete nodal function;
2. Recovery of a nodal metric from an edge-based error estimator;
3. Recovery of a nodal metric from a cell-based error estimator;
4. Modification of a metric for error minimization in the L_p norm.

These features will be discussed in subsequent sections.

In addition to the library Ani2D-LMR , package Ani2D contains a tutorial directory discribed in the last section.

3.2 Local metric recovery from discrete function

A nodal tensor metric may be recovered from the discrete function defined at nodes of the mesh. The metric is the spectral module of the discrete Hessian of this mesh function. A mesh that is quasi-uniform in this metric minimizes the maximum norm of the approximation error of an underlying continuous function. Two methods for the Hessian recovery are implemented in files `Func2Metric.f` and `Func2MetricZZ.f`.

```
      Subroutine Func2Metric(  u,
     &                         Vrt,Nvrt, Tri,Ntri, Bnd,Nbnd, measure,
     &                         Nrmem,rmem, Nimem,imem)
```

```
      Subroutine Func2MetricZZ(u,
     &                         Vrt,Nvrt, Tri,Ntri, Bnd,Nbnd, measure,
     &                         Nrmem,rmem, Nimem,imem)

C  Input: u -    function, the basis for the metric
C         Nvrt - the number of nodes
C         Vrt  - coords of the nodes
C         Ntri - the number of triangles
C         Tri  - the connecticity table
C         Nbnd - the number of boundary edges
C         Bnd  - the list of boundary edges
C  Output: measure - metric to be defined. Is the discrete
C                   Hessian reduced to elliptic form
C  Work arrays: rmem - d.p., of length  Nrmem
C               imem - integer, of length Nimem
```

The input mesh has to satisfy the following condition. Every boundary can be connected to an interior node with at most *two* mesh edges.

3.3 Local metric recovery from edge-based error estimator

Nodal tensor metric may be recovered from edge-based error estimates η_{e_k}. The metric may be anisotropic in this case. Two methods of nodal metric recovery are implemented.

The first method is the Least Squares solution of the local system

$$(M(a_i)e_k, e_k) = \eta_{e_k}.$$

Here $M(a_i)$ is the tensor metric to be recovered at a mesh node a_i, e_k are all mesh edges incident to a_i, η_{e_k} are edge-based error estimates.

```
      Subroutine EdgeEst2MetricLS(nP, nF, nE, XYP, IPE, IPF,
     &                            error, metric,
     &                            MaxWr, MaxWi, rW, iW)

c Input:
      Real*8  error(3, *)  ! edge error estimates data
      Integer nP, nF, nE   ! numbers of nodes, cells, boundary edges
      Real*8  XYP(2, *)    ! coordinates of mesh nodes
      Integer IPE(3, *)    ! connectivity table
      Integer IPF(4, *)    ! boundary edges data

c Output:
      Real*8  metric(3, *) ! node-based metric

c Working arrays:
      Integer MaxWr, MaxWi
      Integer iW(MaxWi)
      Real*8  rW(MaxWr)
```

The second method recovers cell-based tensor metric cell-wise and then for each node a_i it picks the cell metric with the maximum determinant among all within a_i-superelement.

```
      Subroutine EdgeEst2MetricMAX(nP, nF, nE, XYP, IPE, IPF,
```

```
     &                            error, metric,
     &                            MaxWr, MaxWi, rW, iW)

c Input:
      Integer nP, nF, nE   ! numbers of nodes, cells, boundary edges
      Real*8  XYP(2, *)    ! coordinates of mesh nodes
      Integer IPE(3, *)    ! connectivity table
      Integer IPF(4, *)    ! boundary edges data
      Real*8  error(3, *)  ! edge error estimates data

c Output:
      Real*8  metric(3, *) ! node-based metric

c Working arrays:
      Integer MaxWr, MaxWi
      Integer iW(MaxWi)
      Real*8  rW(MaxWr)
```

Both methods are implemented in file `EdgeEst2Metric.f`. Being involved in an adaptive loop, the recovered metric minimizes the estimated norm of the error.

3.4 Local metric recovery from cell-based error estimator

Nodal tensor metric may be recovered from cell-based error estimates η_{Δ_k}:

$$M(\Delta_k) = \eta_{\Delta_k}.$$

The metric is isotropic (scalar tensor) in this case. The nodal metric is generated by ZZ recovery to scalar cell-based metric.

```
      Subroutine CellEst2Metric(nP, nF, nE, XYP, IPE, IPF,
     &                     error, metric,
     &                     MaxWr, MaxWi, rW, iW)

c Input:
      Integer nP, nF, nE   ! numbers of nodes, element, boundary edges
      Real*8  XYP(2, *)    ! coordinates of mesh nodes
      Integer IPE(3, *)    ! connectivity table for elements
      Integer IPF(4, *)    ! boundary edges data
      Real*8  error(*)     ! element-based error estimates

c Output:
      Real*8  metric(3, *) ! node-based metric

c Working arrays:
      Integer MaxWr, MaxWi
      Integer iW(MaxWi)
      Real*8  rW(MaxWr)
```

The method is implemented in file `CellEst2Metric.f`. Being involved in an adaptive loop, the recovered metric minimizes the estimated norm of the error provided the solution is isotropic.

3.5 Metric modification for error minimization in L_p

The local metric recovery from discrete function targets minimization of the maximum (L_∞) norm of the error in discrete solution. If the user wants to minimize L_p norm, he should modify the metric in accordance with value of p.

```
      Subroutine Lp_norm(nP, Lp, Metric)

C  Routine computes the metric for L_p norm using the metric
C  generated for the maximum norm.
C
C     Lp    - norm for which the metric is to be adjusted:
C               Lp > 0  means  L_p     norm
C               Lp = 0  means  maximum norm (L_infinity)
C               Lp < 0  means  H_1     norm (not implemented yet)

      Real*8  Metric(3, *), Lp
```

The method is implemented in file `Lp_norm.f`. The routine must follow the call of `Func2Metric`. Being involved in an adaptive loop, the recovered and modified metric minimizes L_p norm of the error.

4 Examples

Examples of usage of the package Ani2D-LMR are located in `src/Tutorials/PackageLMR`.

The program `mainFunc2Metric.f` demonstrates the local metric recovery from discrete solution defined at nodes of the mesh. The metric is recovered from evaluation of the discrete Hessian of the solution. The metric depends on what Lp norm of the error the user wants to minimize in adaptive mesh generation.

The program `mainEst2Metric.f` builds a metric from errors defined at centers of mesh elements (cells or edges). The errors may be substituted by user given error estimates. The current release calculates the maximum norm of the interpolation error on cells or edges for user-defined function `Func(x,y)`.

Ani2D-VIEW Version 2.2 "*Coneflower*"

Visualization Toolkit

User's Guide for libview-2.2.a

Ani2D-VIEW is a simple visualizing library producing PostScript-files of a mesh and isolines of a discrete solution.

Self-instructive examples of using Ani2D-VIEW are given in `src/Tutorials/PackageVIEW/main.f`.

Ani2D-C2F Version 2.2 "*Fleeceflower*"

C-wrapper for FORTRAN Packages

User's Guide for libc2f-2.2.a

The C package Ani2D-C2F is a simple C-wrapper to call mesh generation routines from package Ani2D-MBA in a C program. In future releases Ani2D-C2F will be extended by C-wrappers to Ani2D-RCB , Ani2D-FEM , Ani2D-ILU .

Examples of using Ani2D-C2F in C programs are given in files `src/Tutorials/PackageC2F/main_nodal.f`, `src/Tutorials/PackageC2F/main_analytic.f`.