

Ani2D Version 2.0

**Ani2D-MBA: Flexible Mesh Generator
Using Metric Based Adaptation**

**User's Guide for libmba2D-2.0.a
February 10, 2007**

1997-2007

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 explicitly, via an analytical formula, or implicitly, via a discrete Hessian recovered from a user-supplied mesh function (usually, a problem solution). In the first case, the user may generate a mesh with desirable properties. In the second case, the resulting mesh is adapted to the given mesh function enabling a better resolution of sharp changes in the solution.

The library *libmba2D-2.0.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. It is pertinent to note that Ani2D-MBA was written as a test-bed for Ani3D-MBA, the generator of anisotropic tetrahedral meshes. The package Ani3D-MBA is expected to be freely available in 2007.

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 only available for nonprofit use. 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 Description of Ani2D-MBA

The main objective of 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.

3.1 Structure of the package

The main Fortran 77 subroutines of Ani2D-MBA are *mesh_metric* and *mesh_solution* located in files with the same names. The depending subroutines are contained in the other files in directory `src/aniMBA`. The files

```
main_metric.f    main_solution.f    time.f
```

may be modified by the user. Routines in the first file generate analytic metric. Routines in the second file use a user-supplied solution defined at mesh nodes of the input mesh to generate a metric. In addition to that, these files contain routine *calCrv* describing a parametrization of curved boundaries.

Some of the models do not have curved boundaries. In this case routine *calCrv* is the dummy routine. Similarly, in the case of using *mesh_solution*, routines defining the analytic metric are dummy. File *time.f* is a wrapper for the system call *etime* that computes CPU time. Generally speaking, this routine depends on the operational system. A few examples of files *main_metric.f* and *main_solution.f* can be found in directory *src/aniMBA/examples*.

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

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

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

```
main_metric.f  main_solution.f
```

are examples of programs showing how to call two main routines of the package: *mesh_metric* and *mesh_solution*, respectively. For visualization purposes, a simple service library *libview2D.a* was created. Routine *draw()* from *libview2D.a* is used in *main_metric.f* and *main_solution.f* for generation of PostScript figures. The file

```
Makefile
```

builds the package under Linux. The executable programs are put in directory *bin*. The names for compilers are defined in *src/Rules.make*. A few examples of input meshes may be found in directory *data*. This document and other documentation related to the package Ani2D are located in directory *doc*.

3.2 Basic things the user should know

The package provides two methods to control the mesh generation. The first method is based on a piecewise linear interpolant of the user-defined metric. The second method is based on a piecewise linear metric defined by a discrete Hessian of the user-supplied mesh function. This function 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.

The package is encapsulated in the two basic routines *mesh_metric* and *mesh_solution* corresponding to the above methods. The comments in file *src/aniMBA/mesh_solution.f* are worth to read! After understanding what are input and output data for each of the methods, the user may find more details in files *main_metric*.f* and *main_solution*.f* located in directory *src/aniMBA/examples*.

3.3 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 *loadMone*.

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

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

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

An analytical metric has to be supplied for module *mesh_metric*. The user should change functions *F*, *G* and *H* located in file `src/aniMBA/main_metric.f`. For more detail, we refer to comments in this file.

A routine *calCrv* has to be supplied for both modules *mesh_metric* and *mesh_solution* if the user model has curved boundaries. If the user model does not have curved boundaries, the routine *calCrv* is the dummy routine. *CalCrv* 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_metric.f` and `main_solution.f`. These files are in directory `src/aniMBA`. The input control parameters are the following variables:

```

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

3.3.1 Mesh representation

Understanding 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 indentificator
                  (example: unit square has 4 boundaries which
                  may have different indentificators)

```

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

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.

4 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 `src/Rules.make` and then to execute the following commands:

```
$make libs  
$cd src/aniMBA  
$make exe
```

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

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

```
Loading mesh ../data/wing.ani
```

```
STONE FLOWER! (1997-2007), version 2.0
```

```
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 ../data/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 `ini.ps` and `fin.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` and `make 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 loadMone(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      nPv, MaxPV, nFv, MaxFV, nEv, MaxEV,
&      XYP, IPF, IPE, IPV, IFV, IEV, lbE,
&      ParCrv, iFnc,
&      "../data/square")
```

or to use one the examples from directory `src/animBA/examples`. The user may play with the input control parameters in file `src/animBA/main_metric.f` and with functions F, G , and H located in the same file. 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.

5 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. 1.

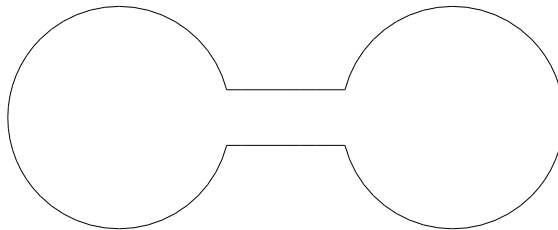


Figure 1: The domain to be meshed.

The user has to write routine *calCrv*. If the user wishes to use mesh loader *loadMone*, 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 *calCrv* may be as follows:

```

      Subroutine calCrv(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 `src/aniMBA/example/main.metric_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. 2.

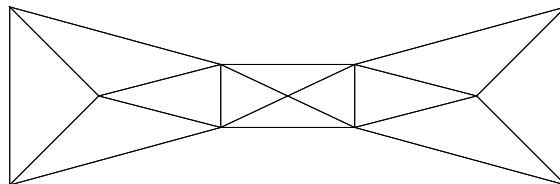


Figure 2: The initial coarse mesh.

The file `data/sport.ani` has a header (9 lines), followed by the list of points (11 points), list of edges (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
0.5000000000000000 0.5000000000000000
0.8000000000000000 0.5000000000000000
0.2000000000000000 0.5000000000000000
0.606350832689630 0.5500000000000000
0.941421356237310 0.641421356237310
0.941421356237310 0.358578643762690
0.606350832689630 0.4500000000000000
0.393649167310370 0.4500000000000000
5.857864376269000E-002 0.358578643762690
5.857864376269000E-002 0.641421356237310
0.393649167310370 0.5500000000000000
```

```
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 *calCrv*. It is very important to guarantee that evaluation of *calCrv* 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, routine *calCrv* 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 mesh to be generated is quasi-uniform. In other words, we have to define functions F , G and H . We choose the identity matrix, i.e.

$$F = G = 1 \quad \text{and} \quad H = 0.$$

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 file `src/aniMBA/main_metric.f`, compile the package and execute the code (`# make exe run-met`). We get the mesh shown in Fig. 3.

6 Two more examples

The first geometric model is shown in Fig.4 (left picture). The left side of the model is partly curved. This part is parametrized as follows:

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

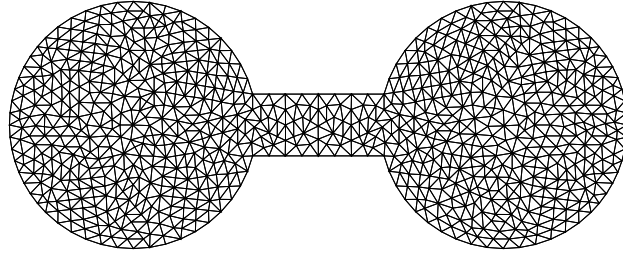


Figure 3: The final mesh.

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

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

The file `src/aniMBA/main_solution.f` demonstrates how to modify the input mesh data `data/square.ani` according to the solution `data/square.sol`.

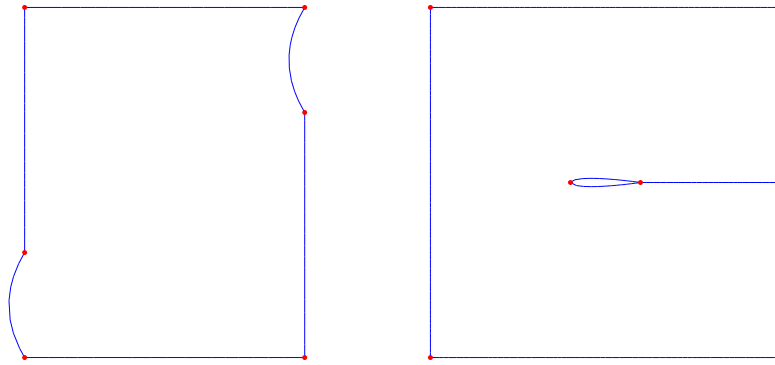


Figure 4: Two models: the square with curved sides and the wing.

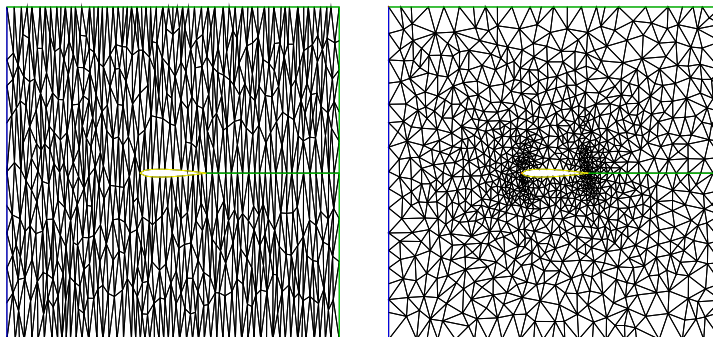


Figure 5: The initial and final meshes of the model `data/wing.ani`.

The second model is shown in Fig.4 (right picture). We use one parametrization for the wing and the other parametrization for the slit behind the tail. The file `src/aniMBA/main_metric.f` defines a metric such that the final mesh refines isotropically towards the leading and trailing edges of the wing, see Fig.5.

7 Useful features of Ani2D-MBA , version 2.0

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. It is possible to produce meshes minimizing different L_p -norms of the interpolation. The input non-negative parameter `Lp` defines this norm. In the special case `Lp = 0`, the maximum norm is used.
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.6 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.

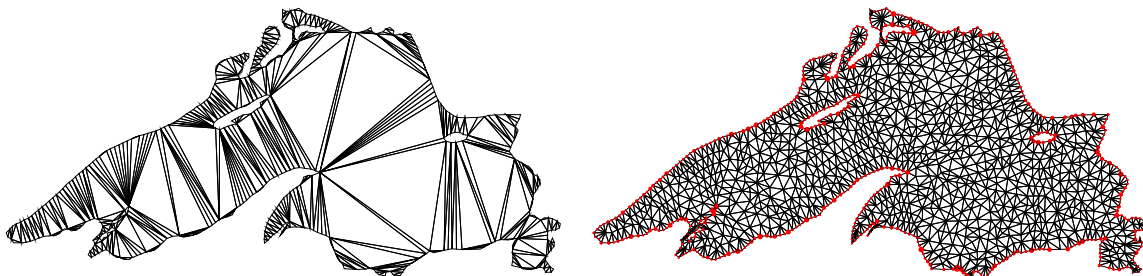


Figure 6: The initial and final meshes of the model `data/country.ani`.

4. The library `libmba2D-2.0.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 routines `listX2Y` which create connectivity lists $X \rightarrow Y$ for mesh objects X and Y such as elements, edges, boundary edges, and points (see `src/aniMBA/utlis.f` for more detail).

- Miscellaneous 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 necessary 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.

8 How to use library libmba2D-2.0

Here we describe one of the main modules, *mesh_solution*, from the library `libmba2D-2.0.a`. The other module, *mesh_metric*, has exactly one parameter (`Sol`) less than *mesh_solution*.

```

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

```

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

```

I      MaxP - [integer] maximal number of points
N      MaxF - [integer] maximal number of boundary and interface 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 controlling 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

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

```


second one. In other words `IPF(1, *)` and `IPF(2, *)` are flipped if necessary. The size of working integer array `iW` is $3 \text{ nE} + 2 \text{ nF} + \text{nP}$.

```
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 node by at most two mesh edges. The size of working integer array `iW` is $3 \text{ nE} + 3 \text{ nP}$. 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/utills.f` for more detail).

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

- `listR2R` creates connectivity lists `nRR` and `IRR` for mesh edges. The routine counts the number of mesh edges, `nR`. Then, `nRR(i) - nRR(i-1)` (`nRR(1)` when `i=1`) gives the total number of edges in 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/utills.f` for more detail).

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

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

10 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 `mesh_solution` and `mesh_metric` have so many input parameters?
A. Next release of the package will have routines `mesh_metric_short` and `mesh_solution_short` with functionality close to that of main routines `mesh_metric` and `mesh_solution`, 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-2.0.a` in an adaptive loop?
A. Yes. Use `make libs` to generate the library `libmba2D-2.0.a` which may be linked with other codes. Depending on the user goals, he or she may call either `mesh_metric` or `mesh_solution`. The package contains a few examples of solving partial differential equations on adaptive grids (see `src/TutorialIntermediate`, `src/TutorialAdvanced` for more detail).
- Q. Why does `libmba2D-2.0.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-2.0.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_metric.f` or `main_solution.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.