Ani2D Version 1.3

Generator of Adaptive Anisotropic Meshes based on the Hessian Recovery

User's Guide

December 1, 2005

1997-2005

1 Introduction

The Fortran package Ani2D is developed by Konstantin Lipnikov and Yuri Vassilevski. It is designated for generation conformal triangular meshes which are quasi-uniform in a given metric. The metric may be defined either explicitly, via analytical formulae, or implicitly, via a discrete Hessian recovered from a user-supplied mesh function. 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 *libani2D-1.2.a* can be built to incorporate our code 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 *modifies* the initial mesh through a sequence of local modifications. This approach provides a stable algorithm for generating strongly anisotropic grids. A generalization of the algorithm to tetrahedral meshes has been successfully implemented by authors. It is pertinent to note that Ani2D was written as a test-bed for Ani3D, the generator of anisotropic tetrahedral meshes. The latter is available by contacting the authors (`lipnikov@hotmail.com` or `vasilevs@dodo.inm.ras.ru`).

This document is a quick guide to the user. It describes a structure of the package, input data, and user-supplied (optional) routines. Furthermore, it explains how the user can control the mesh generation. Finally, it presents a synthetic example showing mesh generation process in detail.

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

The main objective of Ani2D is to produce a mesh with a prescribed number of triangles which is as much quasi-uniform in a given metric as possible. For example, when the metric is isotropic and constant, Ani2D will generate a mesh consisting of equilateral triangles. 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 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/aniGEN`. 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 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. The file `time.f` is a wrapper for the system call *etime* computing the 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/aniGEN/examples`.

For user convenience, package Ani2D is equipped with auxiliary files

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

Their purpose is to facilitate loading, saving and visualizing meshes. The file `bin/ps.lib` contains a library of PostScript routines used for the mesh visualization. The files

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

are examples of main programs showing how to call two main routines of the package: *mesh_metric* and *mesh_solution*, respectively. The file

```
Makefile
```

is an example of building the package under Linux. The executable programs are put in directory `bin`. A few examples of input meshes may be found in directory `data`. This document and other documentation related to the package are located in directory `doc`.

3.2 Basic things the user should know

The package provides two methods for controling the mesh generation. The first method is based on a piece-wise linear interpolant of the user-defined metric. The second method is based on a piece-wise linear metric defined by a discrete Hessian of the user-supplied mesh function. The package is encapsulated in the two basic routines *mesh_metric* and *mesh_solution* corresponding to the above methods. The comments in file `src/aniGEN/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 detail in files `main_metric_*.f` and `main_solution_*.f` located in directory `src/aniGEN/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.

  The current version of Ani2D supports the old format for data files. Each piece of the mesh data (nodes, elements, edges, etc.) is put in a separate file. One example of this format is located in directory `data/old-data`. These mesh data can be accessed via the (obsolete) routine *loadM*. The user may use program *Main_Convert* in file `src/aniGEN/main_convert.f` to convert old data format into the new one (see also the list of new features in Ani2D-1.3 in Section 6).

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

- The input *Fortran 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/aniGEN/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's model has curved boundaries. If the user's 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 use new features of Ani2D to fix the boundary points (see Section 6). Then, the final mesh will approximate curved boundaries with the same accuracy as the initial mesh is.

- The input *control parameters* are the numbers used to control the mesh generation. They are defined in files `main_metric.f` and `main_solution.f`. These files are in directory `src/aniGEN`. 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 requested 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 achieving the requested quality (`Quality`) or performing 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 recommend to use for `MaxSkipE` the value set in the main programs (about 100).

3.3.1 Mesh representation

The understanding of a mesh representation is important for the correct usage of modules *mesh_metric* and *mesh_solution*:

```
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

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

               parameters for inner points are computed by the
               linear interpolation between two given parameters

               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 decompressing the distributive, 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/` and `data/old-data`. A documentation for the package may be found in `doc/`. The source code is stored in `src/aniGEN/`. In order to produce and compile the package, the user has to run

```
$cd src/aniGEN
$make exe
```

The user may change the names and options for compilers in file `src/Rules.make`. After the successful compilation, the user is invited to 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! (1997-2005), version 1.2
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 ../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/aniGEN/examples`. The user may play with the input control parameters in file `src/aniGEN/main_metric.f` and with functions *F,G*, and *H* ilocated in the same file. For instance, changing the metric (*F,G,H*), the user can understand 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 which 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 (0.2,0.45), (0.2,0.55), (0.8,0.55), (0.8,0.45). The domain is shown in Fig. 1.

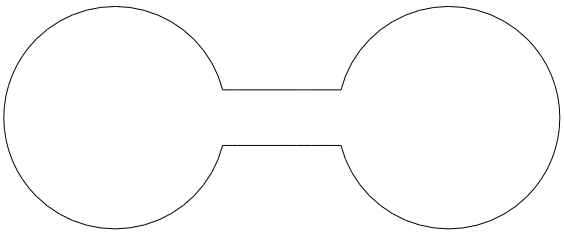

Figure 1: The domain to be meshed.

The user has to write routine *calCrv*. If the user wishes to use mesh loader *loadMone* and mesh saver *saveMone*, he has to create an input data file. Below, we explain how to produce these data from scratch.

Step 1. First, we chose a model of parameterization. The shape of the domain dictates a natural choice for the parameterization of curvilinear parts of the boundary: each circle is parametrized by a trigonometric function. 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/aniGEN/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.

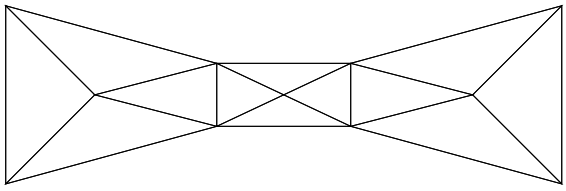

Figure 2: The initial coarse mesh.

The file `data/sport.ani` has a header (9 lines), followed by the list of points (11 points), list of 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 and even 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. However, this list may be replaced by proper coloring of boundary edges. If a point is shared by two edges colored differently, 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 forth 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 third row, i.e. `tc = 0.2526802551420` and `iFnc = 1`. Then, routine *calCrv* should give the Cartesian coordinates of the 4th mesh node.

Step 3. Third, we have to choose an analytic metric. In other words, we have to define functions F, G and H. We choose the identity matrix, i.e.

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

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

6 What is new in Ani2D-1.X

We improved robustness and efficiency of the code, made it more user friendly and added a few new features. The most important features are listed below:

1. We revomed restriction that the initial model should be in the unit square.
2. The initial mesh may be tangled. The user may add `ANIUntangleMesh` to the parameter `status` to untangle the mesh.
3. A few interesting features have been implemented. The full list of available options is in file `src/aniGEN/status.fd`. Here are two most important options:

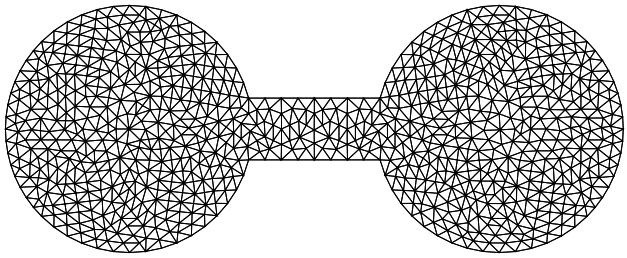

Figure 3: The final mesh.

- The user may freeze boundary points. This feature allows to preserve important geometric features which is important for both isotropic and anisotropic metrics. Fig. 5 in Appendix illustrates this feature. The fixed boundary points (red dots) prevents sharp boundary from smearing. (*The initial mesh has been found on the webcite of Jonathan Shewchuk.*)
- The user may freeze boundary edges and mesh elements. This feature allows to preserve mesh structure in important regions (e.g., in boundary layers).
- The interfaces between materials with different labels (lbE) are automatically recovered and preserved.

4. We added a single-file data format. The data can be manipulated with routines *loadMone* and *saveMone*.
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 put in directory `src/Lapack` and can be binded into a short library `liblapack-3.0.a` using `make lapack`.

7 How to use library libani2D-1.3

Here we describe one of the main modules, *mesh_solution*, of the library `libani2D-1.3.a`. The other module, *mesh_metric*, has exactly one parameter 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,
     &      MaxWr, MaxWi, rW, iW,
     &      iPrint, iERR)
```

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


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       rQuality - [real*8] quality of the final mesh
R
A       status   - [integer] sum of positive numbers corresponding
M                          to desired mesh properties (see status.fd)
E
T       Sol(MaxP) - [real*8] mesh function defined at points;
E                            linearly interpolated function on output
R
s       rW(MaxWr) - [real*8]  working array
        iW(MaxWi) - [integer] another working array
```

8 FAQ

- Q. The mesh generator does not refine 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 is 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 strongly changed across lines of a 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 Ani2D has so many input parameters?
  A. Next version of Ani2D 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 Ani2D in an adaptive loop?
  A. Yes. Use `make lib` to generate a library `libani2D-1.3.a` which may be linked with other codes. Depending on the user goals, he or she may call either *mesh_metric* or *mesh_solution*.

- Q. Ani2D fails to untangle the mesh. Why?
  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 Ani2D fails to generate a mesh.
  A. The authors are very interesting in any feedback from the 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.

9 Two more models

The first model is shown in Fig.5 (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:

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

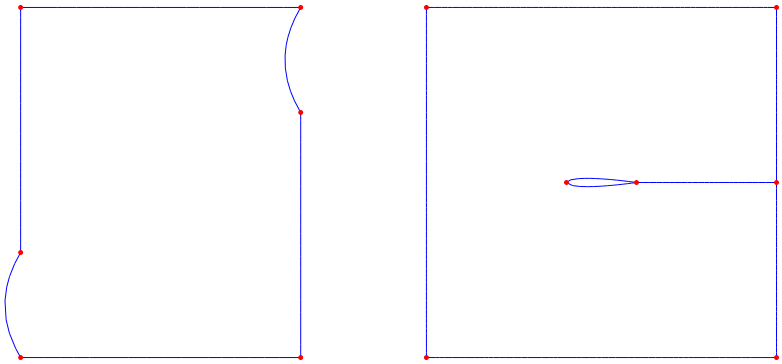

Figure 4: Two models: the square with curved sides and the wing.

The second model is shown in Fig.4 (right picture). We use one parametrization for the wing and the other parametrization for the tail line (see file `src/aniGEN/examples/main_metric_wing.f` for more detail).

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.

10 Appendix A

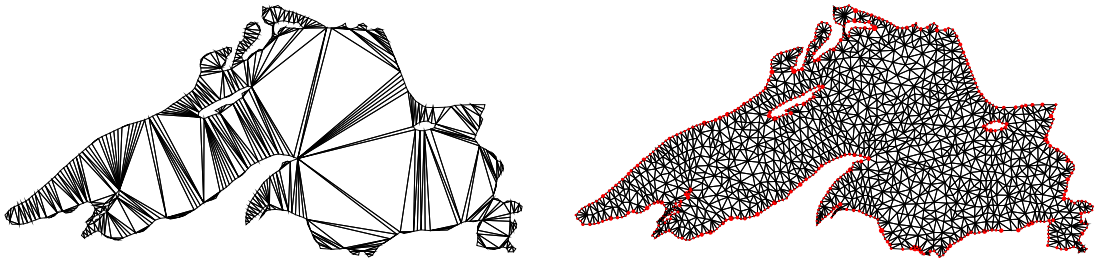

Figure 5: The initial and final meshes of a model `country`.