TP: Mesh adaptation with the Mmg platform

Algiane Froehly

16 janvier 2018

1 Goals :

- Learn to run the Mmg applications ;
- learn to call the Mmg libraries ;
- understand the Mmg outputs ;
- adapt a mesh to a given size map ;
- compute an isotropic/anisotropic size map based on the interpolation error.

2 Mmg platform in short

The Mmg platform gathers software dedicated to simplicial mesh modifications :

- mmg2d : for 2D meshes ;
- mmgs : for 3D surface meshes ;
- mmg3d : for 3D volume meshes.

All this software allow quality improvement, mesh adaptation to a size map (isotropic/anisotropic) and level-set discretization.

Additional documentation can be founded here :
http://www.mmgtools.org/mmg-remesher-try-mmg/mmg-remesher-tutorials
and here :
http://www.mmgtools.org/mmg-remesher-try-mmg/mmg-remesher-options

2.1 Installation

1. Clone the Mmg repository and build the applications, libraries and Doxygen documentation :

```
$ git clone https://github.com/MmgTools/mmg.git
$ cd mmg
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make doc
```

2. if your are root, you can run the make install command too.
3. otherwise, add the path of the **build/bin** folder to your **PATH** variable to be able to run the applications from your terminal without adding the full binary path (in the command line, **$PATH_TO_BIN** must be replaced by your path through the Mmg **build/bin** directory :

```
$ echo "PATH=$PATH_TO_BIN:$PATH" >> ~/.bashrc
$ source ~/.bashrc
```

2.2 Mesh vizualisation

You can choose to save your meshes at the native `Mmg` file format (`.mesh`) and to vizualize your mesh with the `Medit` software (advised) or at `Gmsh` file format (`.msh`, you must then use `Gmsh` to vizualize your mesh).

2.2.1 Medit installation

`Medit` is an OpenGL-based scientific visualization software that can be downloaded on Github : https://github.com/ISCDtoolbox/Medit.
A french documentation is available :
https://www.ljll.math.upmc.fr/frey/logiciels/Docmedit.dir/index.html.

To build Medit, you need to have git and CMake on your PC. Then :

```
$ git clone https://github.com/ISCDtoolbox/Medit.git
$ cd Medit
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install
```

`Medit` is installed in **`~/bin`**, you can add this path to your **`PATH`** variable :

```
$ echo "PATH=~/bin:$PATH" >> ~/.bashrc
$ source ~/.bashrc
```

Graphic packages for linux

- GLUT : `apt-get install -y freeglut3-dev`
- `GLUT-Xi: apt-get install -y libxi-dev`
- `GLUT-Xmu: apt-get install -y libxmu-dev`

2.2.2 Medit main commands

To test `Medit`, you can use the **`linkrods.mesh`** file provided in the **`TP/Data`** directory of this repository. To vizualize your mesh, simply run :

```
$ medit $MESH_PATH/linkrods.mesh
```

where **`$MESH_PATH`** is the path toward the **`TP/Data`** folder. Medit print some mesh statistics in your terminal among which :

- The number of each entity of the mesh (vertices, triangles...) ;
- the size of the mesh bounding box ;
- if a solution file (**`.sol`**) file as been readed.

You can click on the graphical window and :

- rotate the object by maintaining **`left click`** and moving the mouse ;
- translate it with **`alt+left click`** ;
- zoom with the **`z`** key and unzoom with **`Z`** ;
- print/remove the mesh lines with **`l`** ;
- print colors by clicking on **`c`** ;
- print colors associated to the entities reference with **`e`** ;
- print the mesh singularities with **`g`**. Required points appears in green, corners and ridges in red and edges in different colors depending on their references (orange for a 0 ref) ;
- inspect the inside mesh (**`clipping mode`**, 3D only) :
  - cut/uncut the mesh along a plane with **`F1`** ;
  - edit/unedit this plane (**`F2`**) and rotate it (**`left click`**) or translate it (**`alt+left click`**) ;
  - print/unprint volume mesh with **`F4`** ;
- delete (resp. undelete) elements of a given color : **`shift-click`** on one element of this color, then press **`r`** (resp. **`R`**) ;
- quit `Medit` with **`q`**.

3 A first run of the remesher

To run the remesher, you must give the application name followed by the path and mesh name (you can use the **naca_embedded.mesh** 2D mesh provided **TP/Data** directory of this repository) :

```
$ cd TP
$ mmg2d_O3 Data/naca_embedded.mesh
```

By default, Mmg creates a mesh in the same path and of the same extension than the input mesh (**.mesh** here) with the **.o** prefix before the extension, so here : **Data/naca_embedded.o.mesh**.

3.1 Get help

You can get help by running **Mmg** with **-h** argument :

```
$ mmg2d_O3 -h
```

Man pages are available under the **doc/man** directory :

```
$ man ../doc/man/mmg2d.1.gz
```

If successfully builded, Doxygen documentation can be oppened in **build/doc/$EXEC_NAME/html/index.html** with **$EXEC_NAME**=mmg2d,mmg3d or mmgs. Thus, you can open it in your web browser (address : **file:///$PATH_TO_BUILD/build/doc/$EXEC_NAME/html/index.html**, where **$PATH_TO_BUILD** must be replaced by your path through the Mmg build directory).

3.2 The Mmg output

You can see the Mmg output in your terminal. By default, Mmg prints (see figure 1) :

- The different phases of the algorithm (analysis step, remeshing step...) and the time spent in each of this steps ;
- some info about the input/output qualities histogram ;
- the final mesh statistics (number of nodes, elements and edges).

You can change the default verbosity of Mmg with the **-v** option. By default, the verbosity value is setted to 1. If you set the verbosity to 5, **mmg2d_O3 Data/naca_embedded.mesh -v 5**, you will obtain :

- detailed quality histograms (see figure 2) ;
- detailed remeshing steps ;
- edge length histogram (see figure 3).

3.3 Mesh improvement with edge length preservation : -optim option

Open your output mesh in Gmsh : by default, Mmg tries to create a mesh that respect the asked boundary approximation (**-hausd** option, 0.01 by default), the maximal ratio between two adjacent edges (**-hgrad** option, 1.3 by default) and that contains the smallest possible number of points.
If you want to preserve the edge length of the input mesh, you can run Mmg with the **-optim** option :

```
$ mmg2d_O3 Data/naca_embedded.mesh -optim -v 5
```

Compare the input and output quality/lengths histograms and check that your ouput mesh is of the same size than the input one.

Open the input and output meshes (**medit Data/naca_embedded.mesh Data/naca_embedded.o.mesh**) and check that the edge lengths are preserved. You can visualize the metric computed by **Mmg** by selecting the window associated to the output mesh and by pressing **m** (you may need to remove the mesh lines with **l**).

If you want, you can play with other **Mmg** options : try for example to run **Mmg** without the -optim option and to disable the gradation (**-hgrad -1**).

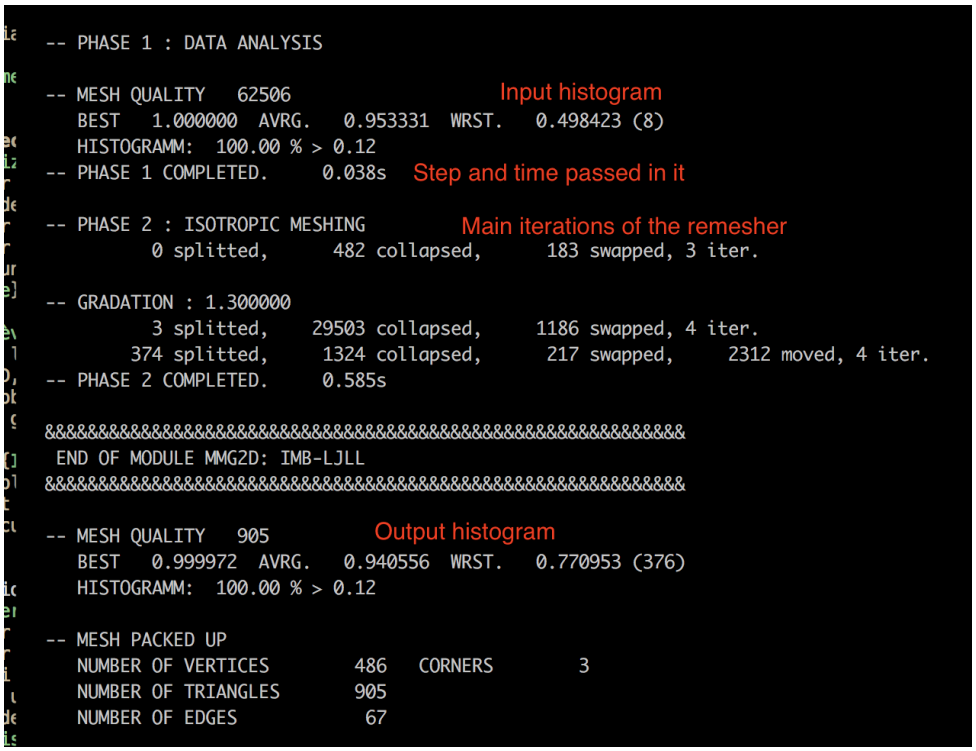


FIGURE 1 – Default Mmg output.

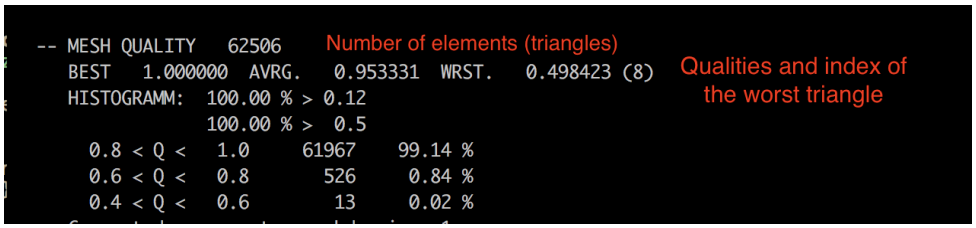


FIGURE 2 – Detailed quality histogram.

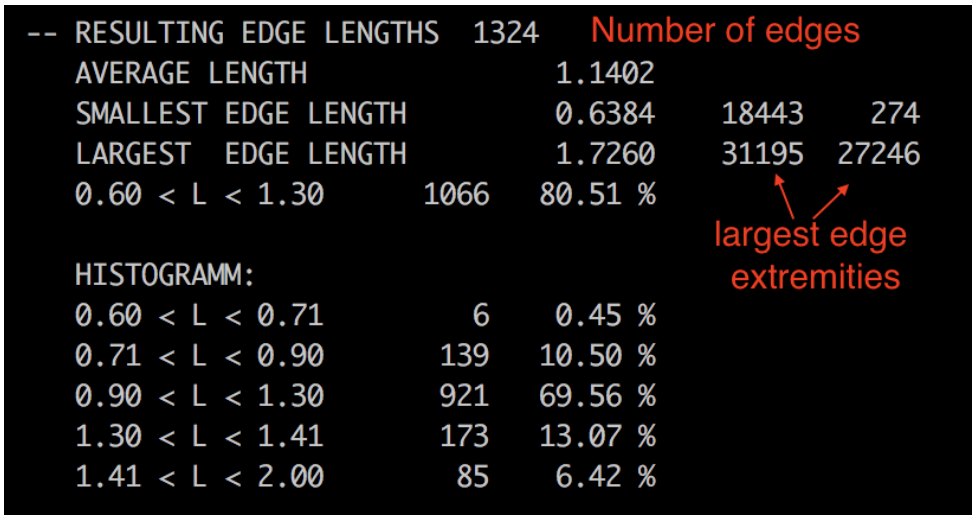


FIGURE 3 – Edge length histogram.

3.4 Boundary approximation

You can try to better approach the naca geometry.

1. look at the size of the naca airfoil (in `Medit`, zoom over the naca and select a vertex near the top of the naca (alt+shift+left click) and another near the bottom. The point coordinates are printed in the terminal so you can evaluate the naca thickness) ;
2. try a hausdorff value related to this length and decrease the minimal edge size in consequence (for example **`-hausd 0.0001 -hmin 0.000001`**)

Why do I need to specify `hmin` in addition to the hausdorff parameter ?
To avoid numerical errors (division by 0) and users mistakes (0 length edges asked), Mmg automatically computes a minimal edge size. If a size map is provided, this minimal edge size is smallest than the smallest asked length but if the user doesn't provide a size map, we must extract a length information from the initial mesh : in this case, by default, Mmg set **`hmin`** to 0.1 times the mesh bounding box size. In our case, because the naca is a very small object in an infinite box, the default **`hmin`** value became too large when we ask for a finer boundary approximation.

3.5 3D boundary approximation

In this subsection, we will use the **`thinker.mesh`** mesh provided in the **`Data`** folder. It is a 3D surface mesh so we will use **`mmgs`** to remesh it.

Remark : The input mesh is a non-conformal mesh (under the bed plate). `Mmg` doesn't detect such meshes and is not supposed to work on it. In the **`thinker`** case, it just leads to a warning :
`## Warning: anaelt: flattened angle around ridge. Unable to split it.`
and to have a bad approximation of the non conformal area.

3.5.1 Mesh analysis without any modification

`Mmg` allows to choose the remeshing operator that can be performed. By default, insertion/collapse, edge swapping and vertex moving are authorized. You can manually disabled each one of this operator :

- no point insertion/collapse : **`-noinsert`** command line argument ;
- no edge swapping : **`-noswap`** ;
- no point relocation : **`-nomove`**.

If you combine this 3 options, `Mmg` doesn't modify your mesh but perform the analysis. Thus, the output mesh contains the singularities detected by the remesher on the initial mesh. This combination can also be used to convert `Gmsh` file into `Mmg` one or vice versa.

3.5.2 Edge detection

1. Analyze your mesh with the default edge detection value. Use `Medit` to vizualize the detected singularities.
2. increase/decrease this value (**`-ar`** ***`val`***) to see the effect on the sharp angle (ridge) detection (analysis only) ;
3. analyze your mesh without the detection of sharp angle(**`-nr`** option). You can see that it remains very few ridges. This ridges are setted by Mmg to mark :
   - non-manifold edges : edges at the intersection of a surface and a hanging surface (so the surfaces intersect in a T-shape) ;
   - the boundary edges of an open surface.
4. remesh your mesh with a suitable value for the ridge detection.

3.5.3 Play with the hausdorff parameter

1. open your mesh with medit to get its bounding box ;
2. evaluate the order of amplitude for the Hausdorff parameter ;
3. play with some hausdorff values (-hausd *val*).

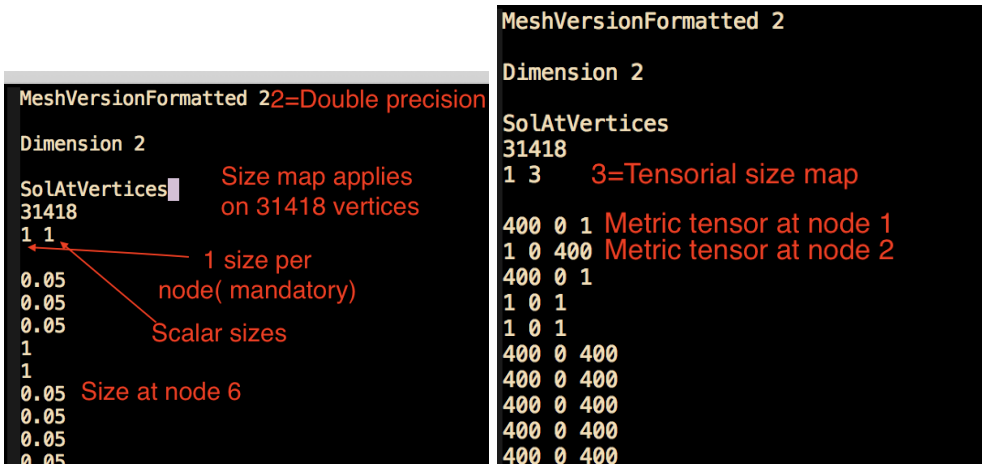


FIGURE 4 – Isotropic (left) and anisotropic (right) size maps at **Medit** file format.

4 Mesh adaptation to a size map

You can provide to Mmg a size map in a **.sol** file. This file lists, for each node, the prescribed edge length. In isotropic case, the file contains 1 scalar data per node (the wanted edge length). In anisotropic case, it contains a metric tensor M that can be diagonalized in the basis of the eigenvectors :

$$M = R\,\Lambda\,R^{-1}$$

With :

- $R = (r_{ij})_{ij}$, the matrix of the eigenvectors ;
- $\Lambda = (\lambda_j)_j$ the diagonal matrix of the eigenvalues.

A given eigenvalue and the associated eigenvector are related to the wanted edge length : $\lambda_j = \frac{1}{s_j^2}$ with s_j the wanted length in the direction r_j.

See figure 4 to see an explanation of the .sol file format for an isotropic and an anisotropic size map.

You will find in the **Data** directory two size maps (**naca_iso.sol** and **naca_aniso.sol**) for the **naca_embedded.mesh** 2D mesh. Try to adapt your mesh to each map. For example, for the isotropic map :

```
$ mmg2d_O3 Data/naca_embedded.mesh -sol naca_iso.sol -hausd 0.001 -v 5
```

Again, you can play with the gradation parameter.

5 Call the Mmg libraries to manually compute a size map

Mmg can be called from **C**, **C++** or **Fortran** code using its API functions.

In this section, we will create a size map for the **naca_embeddded.mesh** mesh and call the Mmg library.
You can start either from the **Data/firstSizeMap.c** or **Data/firstSizeMap.F90** file.
For now, this program :

1. include the mmg2d header file (needed to know the Mmg stuctures) :
   #include "mmg/mmg2d/libmmg2df.h"
2. initialize the Mmg structures (**MMG2D_Init_mesh** function) ;
3. load the mesh (**MMG2D_loadMesh** function) ;
4. save the initial mesh and metric in the **init.mesh** and **init.sol** files (**MMG2D_saveMesh** and **MMG2D_saveSol** functions). Note that if the solution has not been setted, it is not saved.
5. set the hausdorff parameter to 0.0001 (**MMG2D_Set_dparameter** function) ;

FIGURE 5 – Access to the API functions documentation in Doxygen

6. set the minimal edge size parameter to 0.00001 (**MMG2D_Set_dparameter** function) ;
7. call the **mmg2d** library (**MMG2D_mmg2dlib** function) ;
8. save the final mesh and metric ;
9. free the **mmg** structures (**MMG2D_Free_all** function) ;

You can find informations about the prototypes and the role of the API functions in the **mmg2d** Doxygen documentation. In the left panel :

- unroll the **mmg2d**→**Files**→**File list** panel (see image 5) and click on the **libmmg2d.h** item.
- Go into the list of functions and click on the function for which you need informations (for example, the picture 6 shows the role and prototype of the **MMG2D_Get_meshSize** function).

Few remarks for fortran users :

- the fortran prototypes are given in the **Remarks** section of the doc. In general, Fortran arguments are the same than the C arguments with an additional integer argument (the last one) to store the return value of the fortran subroutine.
- for C variadic function (**Init_mesh** and **Free_all**), it is not possible to provide a Fortran interface, thus, wrong arguments can be passed without error at build time.

You can open the program file and try to understand what is done.

5.0.1 Build an application that calls the mmg2d library

You can build the application with the following command :

```
$ gcc firstSizeMap.c -o firstSizeMap -L $MMG_PATH/build/lib/ -lmmg2d
  -I $MMG_PATH/build/include/ -lm
```

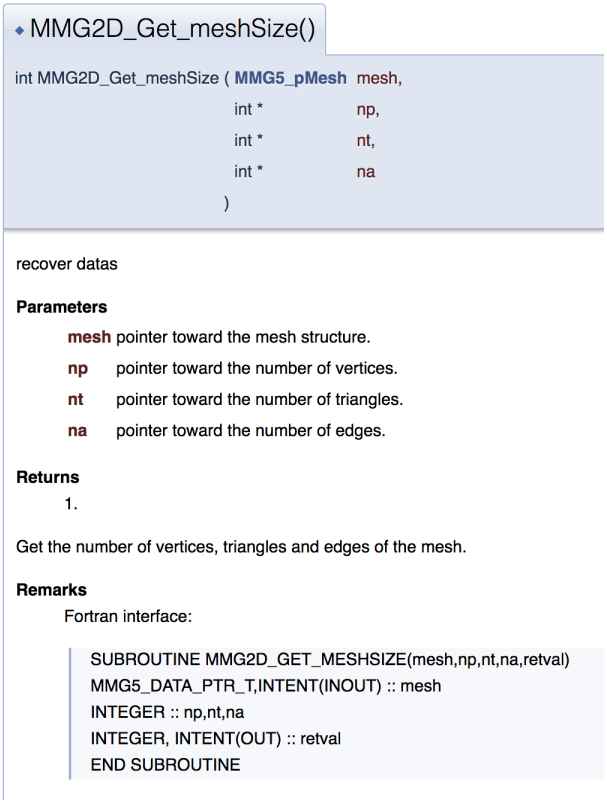
where the **$MMG_PATH** variable must be replaced by your path through the Mmg directory (Fortran users just need to use a fortran compiler instead of a C one and to replace the **firstSizeMap.c** file by the **firstSizeMap.F90** one). It creates the **firstSizeMap** application.

Call this application and look at its outputs.

5.0.2 Size map computation

We will create a size map that asks for edges of size 0.05 inside a circle of center (0,0) and radius 3 and for edges of size 0.2 outside (we create a finer mesh near the naca nose).
For that, we need to specify to Mmg the size and type of the solution and the solution value at each mesh node :

1. Once the mesh is stored, get its size (number of nodes, elements...) and create a scalar solution of the suitable size ;
2. loop over the mesh nodes and get their coordinates.



◆ MMG2D_Get_meshSize()

```
int MMG2D_Get_meshSize ( MMG5_pMesh  mesh,
                         int *       np,
                         int *       nt,
                         int *       na
                       )
```

recover datas

Parameters

mesh pointer toward the mesh structure.

np pointer toward the number of vertices.

nt pointer toward the number of triangles.

na pointer toward the number of edges.

Returns

1.

Get the number of vertices, triangles and edges of the mesh.

Remarks

Fortran interface:

```
SUBROUTINE MMG2D_GET_MESHSIZE(mesh,np,nt,na,retval)
MMG5_DATA_PTR_T,INTENT(INOUT) :: mesh
INTEGER :: np,nt,na
INTEGER, INTENT(OUT) :: retval
END SUBROUTINE
```

FIGURE 6 – MMG2D_Get_meshSize function Doxygen documentation/

3. uncomment the call to the **scalar_size** function and fill this function : given the (x,y) coordinates of a vertex, it must compute the wanted edge size at this vertex ;
4. set the computed size in the size map with the **Set_scalarSol** function.

Run your program, check your initial size map (**init.mesh** file) and the final mesh (**firstSizeMap.mesh**).

6 Size map computation to control the error of interpolation of an analytic function over the mesh

6.1 Computation of the nodal values of a 2D analytic function

1. Choose a function, for example, the following sinus function $f(x, y) = sin(x/2 + y/2)$;
2. Compute its nodal values at the mesh nodes. You can start from the **Data/createSol.c** and **Data/createSol.F90** files and fill the **f** function ;
3. Build the application (the **$MMG_PATH** variable must be replaced by your path through the Mmg directory) :

```
$ gcc createSol.c -o createSol -L $MMG_PATH/build/lib/ -lmmg2d
  -I $MMG_PATH/build/include/ -lm
```

It creates the **createSol** application.

4. This application takes 3 arguments : your inital mesh, the wanted maximal error of interpolation (ϵ) and the type of metric that you want to compute : 0 for a scalar metric, 1 for a tensorial one. For example, to create the anisotropic metric that prescribes edge lengths allowing to have a maximal error of 0.01 over the **naca_embedded.mesh** file :

```
$ ./createSol naca_embedded.mesh 0.01 1
```

This command will creates 3 files :

- **vizuSolution.mesh** that allows to vizualize the analytic function ;
- **vizuMet.mesh** that allows to vizualize the computed metric ;
- **adaptedMesh.mesh**, the final mesh that equirepartite the error of interpolation.

Note that at this step, the metric computation is not yet implemented thus **vizuMet.mesh** contains uninitialized values and the remeshing step must not been performed.

6.2 Computation of a size map to control the interpolation error over the mesh

In this section, we note :

- u the exact solution of a problem ;
- H_u the Hessian of this solution ;
- K a mesh element ;
- V_i the vertices of the element K ;
- $\vec{e}$ the largest edge of the element K ;
- $\langle \vec{e}, M\vec{e} \rangle$ the length of the edge $\vec{e}$ in the metric M.

In the case of a $P1$ finite element method, we can impose an error of interpolation of ϵ over an element K by choosing $\vec{e}$, such as :

$$\epsilon = \frac{2}{9} \left\langle \vec{e}, \max_{v \in V_i} |H_u(v)| \, \vec{e} \right\rangle$$

wich means that we want :

$$1 = \left\langle \vec{e}, \frac{2}{9\epsilon} \max_{v \in V_i} |H_u(v)| \, \vec{e} \right\rangle$$

If we note $M = \frac{2}{9\epsilon} \max_{v \in V_i} |H_u(v)|$, we can see that we want edges of length 1 in the metric M. (See https://www.ljll.math.upmc.fr/frey/publications/mim2.pdf for the proof).

In practice, we will give to the remesher the matric M at the mesh nodes. Thus, at a mesh node V, we want to compute the tensor $M(V)$ such as :

$$M(V) = \frac{2}{9\epsilon} \, |H_u(V)|$$

M is a symetric definite positive tensor so it can be diagonalized in the eigenvector basis :

$$M(V) = \frac{2}{9\epsilon} \, R \, |\Lambda| \, R^{-1}.$$

6.2.1 Anisotropic size map

You can compute the tensor metric $M(V)$ inside the **tensor_size** function of the **createSol.c** file. Use the **siz** array (of size 3) to store m_{11}, m_{12} and m_{22} ($m_{21} = m_{12}$ so it is useless to store it).

For this :

1. Compute $H(V)$, the Hessian of the previous analytical function at a node V. This matrix is symetric definite positive, thus, it is possible to store only 3 of the 4 tensor data inside a 1D array : h_{11}, h_{12}, h_{22}. (you can implement this inside the **hessian** function of the **createSol.c** file) ;
2. compute $\bar{H}(V) = \frac{2}{9\epsilon} H(V)$;
3. compute the eigenvectors and the absolute value of the eigenvalues of $\bar{H}(V)$ (you can use the given **eigenvals** function that computes the eigenvectors and eigenvalues of a symetric matrix) ;

4. a null eigenvalue (which physically means that we want an infinite edge) will create numerical issues (division by 0), thus, we need to truncate the maximal edge length. Truncate the maximal edge length by a suitable value (for example, 10. is a suitable value for the **naca_embedded.mesh** mesh).
5. compute $M(V) = R\bar{\Lambda}R^{-1}$, with $\bar{\Lambda}$ the diagonal matrix of the truncated absolute values of the eigenvalues of $\bar{H}(V)$.
6. uncomment the call to the **mmg2d** library

Open the **vizuMet.mesh** file to vizualize your anisotropic metric field. You can click over a node to print the ellipse associated to the prescribed metric.

Open the **adaptedMesh.mesh** file to see the final result.

6.2.2 Isotropic size map

You can implement the computation of the isotropic edge length at a nod V in the **scalar_size** function of the **createSol.c** file :

1. Perform the 4 steps of the previous section ;
2. find $\bar{\lambda}$, the maximum value of the truncated absolute values of the eigenvalues of $\bar{H}(V)$ and compute $s(V) = \frac{1}{\sqrt{\bar{\lambda}}}$.

Run the application and check your isotropic metric field as well as the adapted mesh.

You can find a correction of the 5 and 6 sections in the **Correction** folder of this repository.