Practical session: Mesh adaptation with the Mmg platform

Charles Dapogny, Algiane Froehly

Janunary 17, 2018

1 Goals of the session:

- 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 of a user-defined function.

2 The Mmg platform in short

The Mmg platform is a suite of three softwares for performing modifications on simplicial meshes (i.e. composed of triangles in 2d, tetrahedra in 3d) :

- mmg2d is dedicated for the treatment of 2d meshes;
- mmgs deals with 3d surface meshes;
- mmg3d considers 3d volume meshes.

These three softwares perform quite similar operations in their respective settings: quality improvement of a user-supplied mesh, mesh adaptation to a size map (isotropic/anisotropic) and level set discretization.

A more exhaustive documentation of the Mmg softwares may be founded at the following addresses:

http://www.mmgtools.org/mmg-remesher-try-mmg/mmg-remesher-tutorials ,

and

http://www.mmgtools.org/mmg-remesher-try-mmg/mmg-remesher-options .

2.1 Installation of Mmg

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 may use 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

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

In the above command line, **$PATH_TO_BIN** has to be replaced by your path through the **Mmgbuild/bin** directory.

2.2 Mesh vizualisation

You can choose to save your meshes either at the native **Mmg** file format (**.mesh**) and then to vizualize your mesh with the **Medit** software (advised) or as a **Gmsh** file format (**.msh**), in which case you must 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 documentation (in French) is available at:

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 automatically installed in the folder **~/bin**, and 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 try out **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 prints some mesh statistics in your terminal among which:

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

You can click on the graphic 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 Getting started with the remeshing mode of `Mmg`

To run the remesher, you simply have to enter the name of the program (by default `mmg2d_O3` for a 2d remeshing), followed by the path and mesh name. For instance, remeshing the 2d mesh **`naca_embedded.mesh`** mesh provided in the **`TP/Data`** directory of this repository is achieved by the following command line:

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

By default, the output mesh lies in the same path and has the same extension as the input mesh (**`.mesh`** here) with the **`.o`** prefix before the extension; in the above case: **`Data/naca_embedded.o.mesh`**.

3.1 Getting help

You can get help by using the **`-h`** argument in the command line:

```
$ mmg2d_O3 -h
```

Man pages are available in the **`doc/man`** directory:

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

If successfully builded, Doxygen documentation can be opened in the folder

`build/doc/$EXEC_NAME/html/index.html`

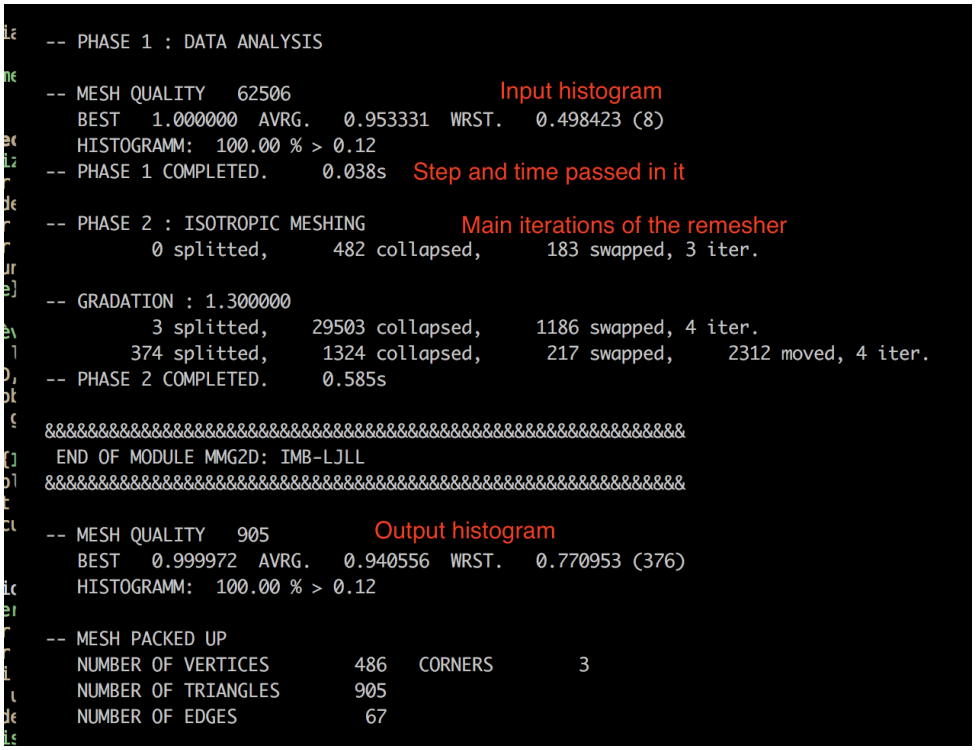


Figure 1: A typical default Mmg output.

where **$EXEC_NAME** stands for **mmg2d**, **mmg3d** or **mmgs** depending on the context.

You may alternatively open this help file in your web browser by supplying the address

file:///$PATH_TO_BUILD/build/doc/$EXEC_NAME/html/index.html,

where **$PATH_TO_BUILD** has to be replaced by your path through the Mmg build directory.

3.2 The output of Mmg runs

The output of the computation is displayed in the terminal; see figure 1 for an example. By default, Mmg prints:

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

You may change the default verbosity of Mmg with the **-v** option. By default, the verbosity value is 1. For instance, turning this verbosity to 5 by using

mmg2d_O3 Data/naca_embedded.mesh -v 5,

allows to display:

- Detailed quality histograms (see figure 2);
- Detailed remeshing steps;
- Edge length histogram (see figure 3).

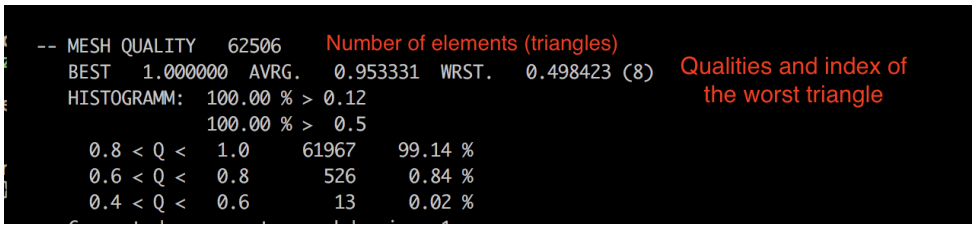


Figure 2: A detailed quality histogram.

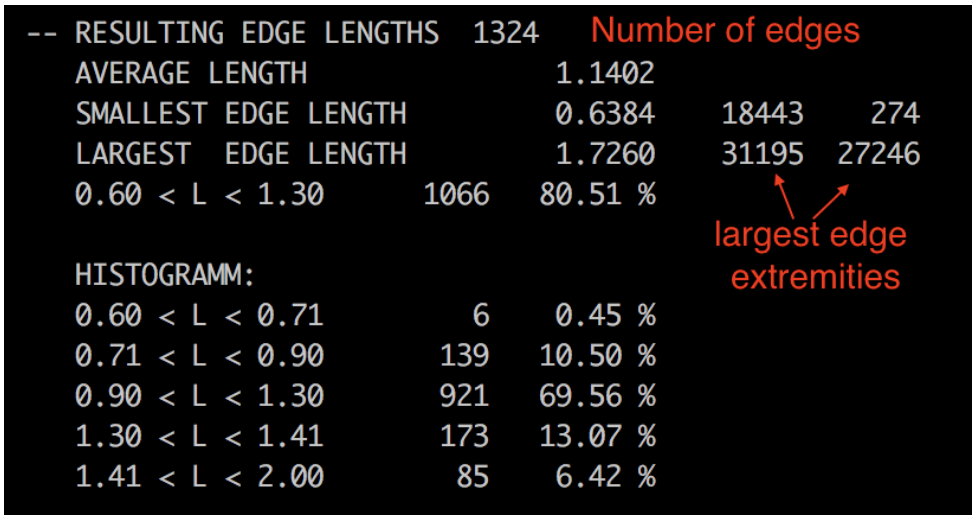


Figure 3: An edge length histogram.

3.3 The four main parameters of Mmg

By default, Mmg creates a mesh that complies with

- The minimum length of an edge in the mesh, controlled by the **-hmin** command line parameter;
- The maximum length of an edge in the mesh, controlled by the **-hmax** command line parameter;
- The required boundary approximation, controlled by the **-hausd** parameter; see Section 3.5 below about this point;
- the maximal ratio between two adjacent edges, controlled by the **-hgrad** parameter: the ration between the length of two adjacent edges e_1, e_2 in the mesh satisfies

$$\frac{1}{\texttt{hgrad}} \leq \frac{|e_1|}{|e_2|} \leq \texttt{hgrad};$$

  this parameter is set to 1.3 by default.

3.4 Mesh improvement with edge length preservation: -optim option

If you want to improve your mesh quality without modifying its initial size (so you wish 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** and **Data/naca_embedded.o.mesh**, and check that the edge lengths are preserved.

You can visualize the metric computed by Mmg according to the edge length in the input mesh (see Section 4 to get a hint of how Mmg uses this information to proceed to remeshing). To this end, using Medit, select the window associated to the output mesh and press **m** (you may need to remove the mesh lines with **l**).

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

3.5 Boundary approximation

In order to better approach the naca geometry, it is desirable to adapt the mesh to the curvature of the boundary - i.e. to impose smaller elements where the boundary of the naca is more curved. To this end,

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 (notably division by 0) and users mistakes (0 length edges asked), Mmg automatically computes a minimal edge size. If a size map is provided (see Section 4), this minimal edge size is smallest than the smallest required length but if the user does not supply a size map, a length information is extracted from the initial mesh: in this case, by default, Mmg sets **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 is too large when a finer boundary approximation is desired.

3.6 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 **mmgs** is used in order to remesh it.

Remark 3.1 *The input mesh is a non conforming mesh (see under the bed plate).* ***Mmg*** *does not detect such patterns 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 the approximation of the non conforming area is pretty bad.

3.6.1 Mesh analysis without any modification

Mmg allows to choose the remeshing operators that are applied. By default, insertion/collapse, edge swapping and vertex relocation 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 these 3 options, Mmg does not modify the input mesh but only performs 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 a Gmsh file into a Mmg one or vice versa.

3.6.2 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 there remain very few ridges. The only remaining ones are:
   - Non-manifold edges: edges at the intersection of a surface and a hanging surface (so the surfaces intersect in a T-shaped pattern);
   - The boundary edges of an open surface.
4. Remesh your mesh with a suitable value for the ridge detection angle.

3.6.3 Play with the hausdorff parameter

1. Open your mesh with Medit to get the size of its bounding box;
2. Evaluate the order of amplitude for the Hausdorff parameter;
3. Try out several values of the hausdorff parameter (**-hausd** *val*).

4 Mesh adaptation to a size map

It is possible to supply Mmg with a size map in a **.sol** file. The role of this file is to prescribe a desired local edge length around each vertex in the mesh.

- In the case of *isotropic* mesh adaptation, 1 scalar data is supplied per node (the wanted edge length around it).

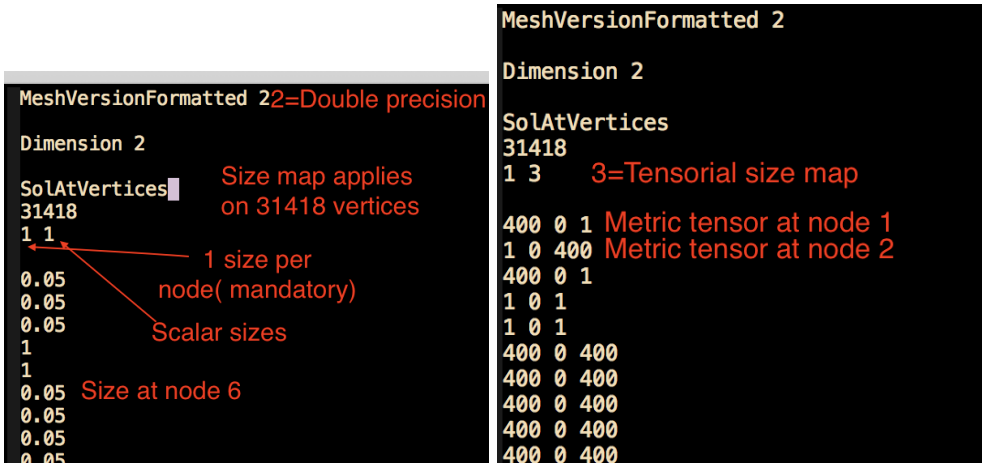


Figure 4: Isotropic (left) and anisotropic (right) size maps at **medit** file format.

- In the case of *anisotropic* mesh adaptation, this file supplies a metric tensor M at each vertex, that can be diagonalized in the eigenbasis:

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

where:

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

A given eigenvalue λ_j and the desired edge length s_j in the direction of this the eigenvector r_j are related as:

$$\lambda_j = \frac{1}{s_j^2}.$$

See Figure 4 for an explanation of the **.sol** file format for both isotropic and anisotropic size maps.

You will find in the **Data** directory two size maps (**naca_iso.sol** and **naca_aniso.sol**) for the **naca_embedded.mesh** 2D mesh. A 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 may play with the gradation parameter **-hgrad**.

5 Calling the Mmg libraries to manually compute a size map

The Mmg library may be called from **C**, **C++** or **Fortran** codes by using its API functions.

In this section, we shall 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.
Now, follow the program:

1. Include the **mmg2d** header file (needed to know the Mmg structures):

```
#include "mmg/mmg2d/libmmg2df.h".
```

2. Initialize the Mmg structures (**MMG2D_Init_mesh** function);

Figure 5: Access to the API functions documentation in Doxygen

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 set, it is not saved.
5. Set the hausdorff parameter to 0.0001 (**MMG2D_Set_dparameter** function);
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 may 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).

A few remarks are in order for **Fortran** users:

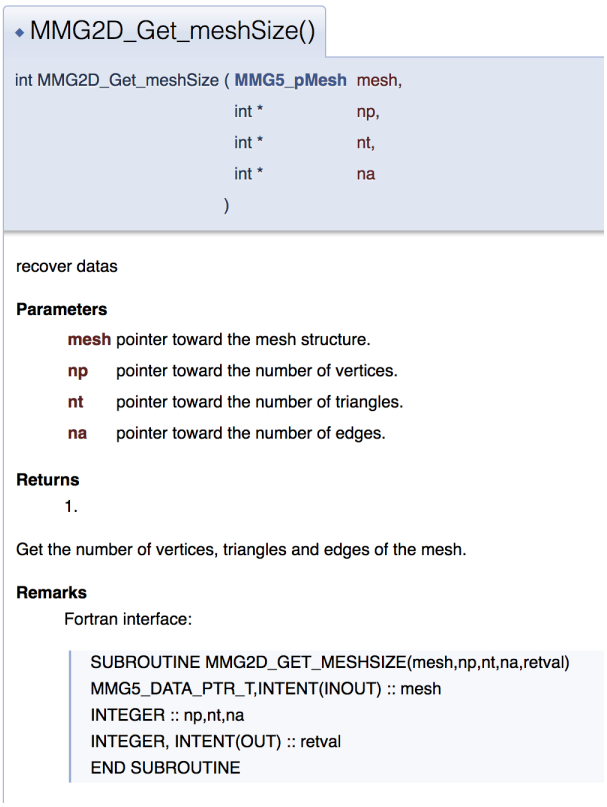
- The **Fortran** prototypes are given in the **Remarks** section of the documentation. In general, **Fortran** arguments are the same than the **C** arguments with an additional integer argument (the last one) to store the return value of the fortran subroutine.
- For **C** variadic function (**Init_mesh** 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.

5.1 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
```



◆ 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 in the Doxygen documentation.

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). Doing so creates the **firstSizeMap** application.

Call this application and look at its outputs.

5.2 Size map computation

We will create a size map associated to the **naca_embedded.mesh** mesh. For example, we can ask for an edge length equal to

- 0.05 inside a ball of center (0,0) and radius 3
- 0.2 outside this ball.

In other terms, a finer mesh is required near the nose of the naca.

To achieve this purpose, 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 with the suitable size;
2. Perform a loop over the mesh nodes and get their coordinates.
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 interpolation error of an analytic function over the mesh

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

1. Choose a function $f : \mathbb{R}^2 \to \mathbb{R}$, for example,
$$f(x, y) = \sin\left(\frac{x}{2} + \frac{y}{2}\right);$$
2. Compute its nodal values at the mesh nodes. You may start from the **Data/createSol.c** and **Data/createSol.F90** files and fill the function **f** ;
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
```
Doing so 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 (in the case of isotropic adaptation), 1 for a matrix one (in the case of anisotropic adaptation). 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 generates 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 makes even the interpolation error.

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 [3] 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).

To achieve this, proceed as follows:

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 node 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.

A correction for the exercices in Sections 5 and 6 is available in the **Correction** folder of this repository.

References

[1] C. Dapogny, C. Dobrzynski and P. Frey, *Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems*, Journal of Computational Physics. 2014;262:358–378.

[2] C. Dobrzynski and P. Frey, *Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations*, Proc. 17th Int. Meshing Roundtable, Pittsburgh, (2008).

[3] F. Alauzet and P. Frey, *Estimateur d'erreur géométrique et métriques anisotropes pour l'adaptation de maillage. Partie I : aspects théoriques*, Rapport de recherche RR-4759, INRIA, (2003).