Mesh Smoothing in Nek5000

Ketan Mittal*

A mesh smoother has been deployed for smoothing meshes in Nek5000. This smoother is a combination of Laplacian smoothing and mesh optimization, which makes the mesh more uniform while preserving the boundary layer resolution of the original mesh. This document explains the parameters that are important for understanding the functionality of the smoother and using it effectively. The example here demonstrates how to use the mesh smoother on a 2D mesh for a turbine blade.

1 Smoother Parameters

The user must follow the following steps to run the mesh smoother:

1. include *smoothlx.f* in the *.usr* file. The file *smoothlx.f* includes the subroutine *smoothmesh* and other dependencies.
2. Specify the metric type (*mtyp*) for optimization - Currently only $mtyp = 1$, which optimizes based on the condition number of the Jacobian matrix, is available. More metrics will be added soon.
3. Specify the number of smoothing iterations:
   nouter - The number of loops for Laplacian and optimization smoothing
   nlap - number of iterations of Laplacian smoothing in each loop
   nopt - number of iterations of optimization based smoothing in each loop

   By default $nouter = 20$, $nlap = 30$, and $nopt = 20$, but the user can change these values based on their mesh.
4. Input parameters *nbc*, *dcbc*, *idftyp*, α, and β required to calculate the *weight function* (w):

   The weight function is used to preserve the boundary layer resolution during the mesh smoothing process. This function is based on the minimum distance (d) of all Gauss-Lobatto Legendre (GLL) points from the boundary conditions specified by the user.

   The user starts by specifying the boundary conditions near which boundary layer resolution must be preserved using parameter *nbc* and character array *dcbc*. The parameter *nbc* is set to 2 by default to indicate two boundary conditions will be specified in *dcbc*, and *dcbc* is set to `/'W  ','P  '/` to indicated that boundary layer resolution will be preserved near *wall* and *periodic* surfaces. The user can modify these values based on their case. For example, the user can set $nbc = 1$ and $dcbc =$ `/'W  '/` to indicate that boundary layer resolution will be preserved only near the *wall* boundary condition.

   Once d is calculated from the boundary conditions specified by the user, it is normalized using the maximum value of d over the computational domain and then the weight function is calculated as:

*Email: kmittal2@illinois.edu, Web: www.k10mittal.com

$$w = 1 - exp(-d/\beta) \text{ if } funtype = 0$$

$$w = 0.5(tanh(\alpha(d - \beta)) + 1) \text{ if } funtype = 1.$$

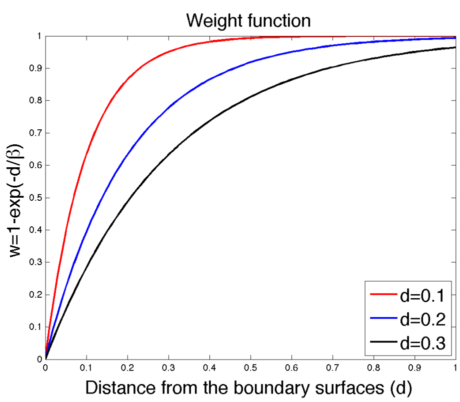
By default $idftyp = 0$, $\alpha = 15$ and $\beta = 0.2$ i.e. the exponential based function is used with $\beta = 0.2$. The user can modify these values of $funtype$, α, and β or implement a function of their own choice by modifying the subroutine *disfun*.

w is used during and after the mesh smoothing process to weight the movement of the nodes. This has the effect of reducing the movement of GLL points near the specified boundary conditions, thus preserving boundary layer resolution, while allowing the GLL points in the far-field to move in order to maximize mesh quality.

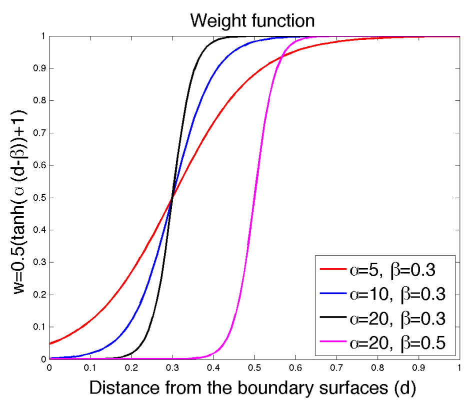
5. Run the smoother from *usrdat2* as:
call smoothmesh(mtyp,nouter,nlap,nopt,nbc,dcbc,idftyp,alpha,beta)

2 Additional notes

1. The values of α and β can be changed depending on how much movement the user wants to allow near the boundary surfaces. Figure 1(a) and (b) shows how w changes for different values of α and β.



(a) Exponential function



(b) Tanh function

Figure 1: Weight function with different parameters

2. Laplacian smoother is computationally cheaper than optimization based smoothing. However, optimization based smoothing is more effective for complicated geometries because it tries to ensure that the mesh stays valid during the smoothing process.

3. The value of the global objective function is printed out by the smoother at the end of each outer iteration. This value should generally be decreasing as the smoother optimizes the mesh quality.

4. Future versions will incorporate surface smoothing.

5. Please send any comments or feedback to kmittal2@illinois.edu

3 Examples

3.1 2D mesh - Low pressure turbine blade

For the 2D case, we take a look at the mesh of a low pressure turbine blade. The domain has *periodic* boundary condition in the pitchwise direction, and *wall* boundary condition at the blade surface. The mesh smoother parameters are set as:

```
c     -------------------
c     -------------------
c     -------------------
      parameter(nbc=2)         !number of boundary conditions
      character*3 dcbc(nbc)
      save        dcbc
      data        dcbc /'W  ','P  '/  !BCs listed here

      idftyp = 0      !distance function - 0 -> exponential, 1-> tanh
      alpha = 15.     !Input for wall distance function
      beta  = 0.2     !Input for wall distance function

      nouter = 20      !total loops around laplacian and optimizer smoothing
      nlap = 30        !number of laplacian iterations in each loop
      nopt = 20        !number of optimization iterations in each loop

      mtyp = 1         !metric type

      call smoothmesh(mtyp,nouter,nlap,nopt,nbc,dcbc,idftyp,alpha,beta)
c     -------------------
c     -------------------
c     -------------------
```

Figure 2 shows a comparison of the mesh before and after smoothing. As evident, mesh smoothing makes the mesh more uniform and transition between different mesh regions smoother.

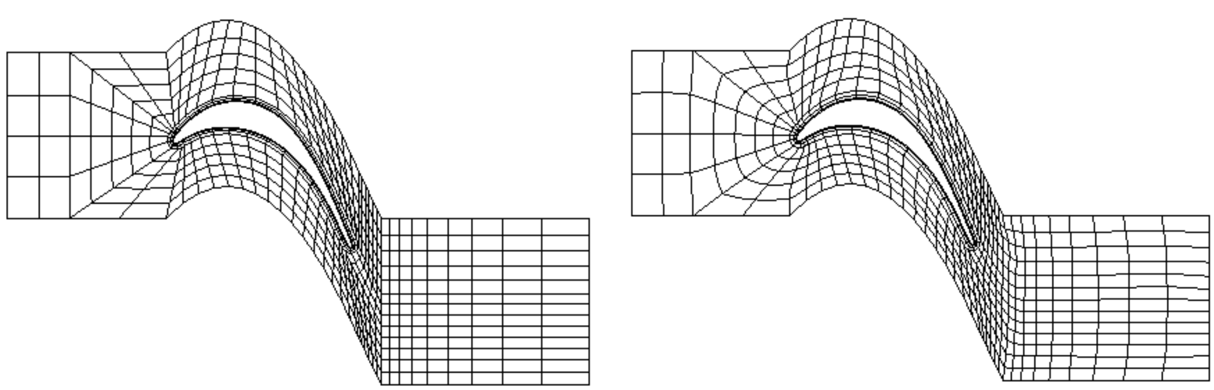


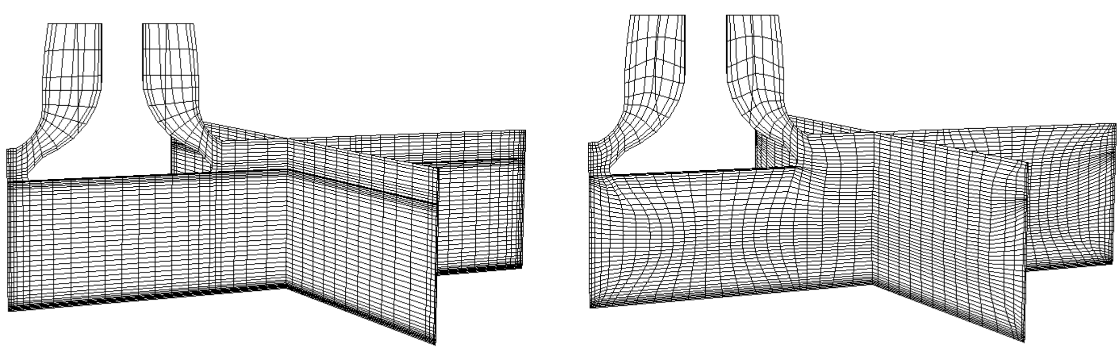
Figure 2: LPT-106 mesh before and after smoothing

3.2 3D mesh - Internal combustion engine's cylinder

For the 3D mesh, we take a look at the mesh for an internal combustion engine's (ICE) cylinder. The mesh has *symmetry* and *wall* boundary conditions. The smoothing parameters are:

```
c     -------------------
c     -------------------
c     -------------------
      parameter(nbc=2)         !number of boundary conditions
      character*3 dcbc(nbc)
      save        dcbc
      data        dcbc /'W  ','SYM'/  !BCs listed here

      idftyp = 0      !distance function - 0 -> exponential, 1-> tanh
      alpha = 15.     !Input for wall distance function
      beta  = 0.2     !Input for wall distance function

      nouter = 35      !total loops around laplacian and optimizer smoothing
      nlap = 10        !number of laplacian iterations in each loop
      nopt = 20        !number of optimization iterations in each loop

      mtyp = 1         !metric type

      call smoothmesh(mtyp,nouter,nlap,nopt,nbc,dcbc,idftyp,alpha,beta)
c     -------------------
c     -------------------
c     -------------------
```

As evident in Fig. 3(a), the original mesh has very thin elements in the far-field because of boundary layer resolving elements below the valve stem. Mesh smoothing makes the mesh more uniform away from the boundary surfaces specified by the user.

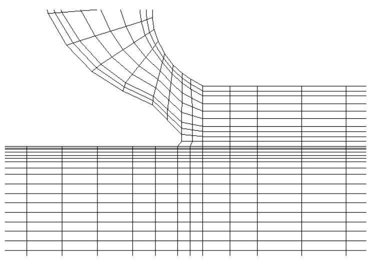


(a) Original mesh

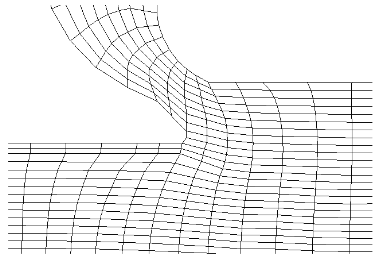
(b) Smoothed mesh

Figure 3: ICE cylinder's mesh before and after smoothing

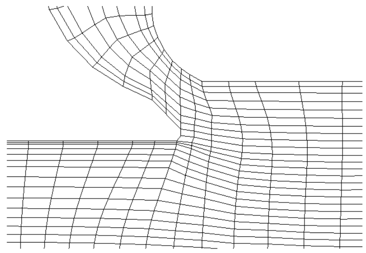
Figure 4 shows a close-up of how mesh smoothing helps make the mesh more uniform away from walls, and the weight function helps in restoring the original boundary layer resolution of the original mesh.



(a) Original Mesh



(b) Smoothed mesh



(c) (w)(smoothed mesh) + (1-w)(original mesh) to restore boundary layer

Figure 4: Mesh at the ICE cylinder's valve stem before and after smoothing