COMPUTATIONAL MECHANICS TOOLS

Homework 1
Transfinite Interpolation

Submitted by
Karthik Neerala Suresh
M.Sc. Computational Mechanics
Universitat Politècnica de Catalunya, BarcelonaTECH

Submitted to
Dr. Josep Sarrate
Associate Professor
Universitat Politècnica de Catalunya, BarcelonaTECH

28 November 2016

Contents

List of Figures

1. Assignment

1.1 Transfinite interpolation

The aim of this work is to implement a 2D version of the Transfinite Interpolation (TFI) method. To this end a computational domain (ξ, η) and a physical space (x, y) is assumed such that

$$\boldsymbol{X}(\xi, \eta) = \begin{bmatrix} x(\xi, \eta) \\ y(\xi, \eta) \end{bmatrix}$$

with $0 \leq \xi \leq 1$ and $0 \leq \eta \leq 1$. Moreover, a discretized version of the computational domain is assumed such that $X(\xi_I, \eta_J)$ is a structured grid for:

$$\begin{cases} 0 \leq \xi_I = \dfrac{I-1}{M} \leq 1 \\ 0 \leq \eta_J = \dfrac{J-1}{M} \leq 1, \end{cases}$$

where $I = 1, 2, ..., M$ and $J = 1, 2, ..., N$, being M and N the number of elements in the ξ and η directions respectively. TFI uses an univariate interpolation in each direction of the computational space

$$\boldsymbol{U}(\xi, \eta) = \sum_{i=1}^{2} \alpha_i(\xi) \boldsymbol{X}(\xi_i, \eta)$$
$$\boldsymbol{V}(\xi, \eta) = \sum_{j=1}^{2} \beta_j(\eta) \boldsymbol{X}(\xi, \eta_j),$$

where $\xi_1 = \eta_1 = 0$ and $\xi_2 = \eta_2 = 1$ are the computational domain limits, and $\alpha_i(\xi)$ and $\beta_j(\eta)$ are called blending functions. The blending functions for the linear TFI are defined as:

$$\begin{cases} \alpha_1(\xi) = 1 - \xi \\ \alpha_2(\xi) = xi \\ \beta_1(\eta) = 1 - \eta \\ \beta_2(\eta) = \eta \end{cases}$$

TFI also considers the tensor product of these univariate interpolation as

$$\boldsymbol{UV}(\xi,\eta) = \sum_{i=1}^{2}\sum_{j=1}^{2} \alpha_i(\xi)\beta_j(\eta)\boldsymbol{X}(\xi_i,\eta_j).$$

Finally, the transfinite mapping is defined as the Boolean sum of the two interpolation as

$$\boldsymbol{X}(\xi,\eta) = \boldsymbol{U}(\xi,\eta) \oplus \boldsymbol{V}(\xi,\eta) = \boldsymbol{U}(\xi,\eta) + \boldsymbol{V}(\xi,\eta) - \boldsymbol{UV}(\xi,\eta).$$

Therefore, the structured mesh in the physical space is computed as

$$\boldsymbol{X}(\xi_I,\eta_J) = \boldsymbol{U}(\xi_I,\eta_J) \oplus \boldsymbol{V}(\xi_I,\eta_J) = \boldsymbol{U}(\xi_I,\eta_J) + \boldsymbol{V}(\xi_I,\eta_J) - \boldsymbol{UV}(\xi_I,\eta_J). \tag{1.1}$$

for $I = 1, 2, ..., M$ and $J = 1, 2, ..., N$, being

$$\boldsymbol{U}(\xi_I,\eta_J) = (1-\xi_I)\boldsymbol{X}(0,\eta_J) + \xi_I\boldsymbol{X}(1,\eta_J) \tag{1.2}$$

$$\boldsymbol{V}(\xi_I,\eta_J) = (1-\eta_J)\boldsymbol{X}(\xi_I,0) + \eta_J\boldsymbol{X}(\xi_I,1) \tag{1.3}$$

$$\begin{aligned}\boldsymbol{UV}(\xi_I,\eta_J) = {} & (1-\xi_I)(1-\eta_J)\boldsymbol{X}(0,0) + (1-\xi_I)\eta_J\boldsymbol{X}(0,1) \\ & + \xi_I(1-\eta_J)\boldsymbol{X}(0,1) + \xi_I\eta_J\boldsymbol{X}(1,1)\end{aligned} \tag{1.4}$$

In order to control the desired spacing between grid points in the physical space we introduce an intermediate control domain between the computational and physical domains according to,

$$(u,v) = \boldsymbol{F}(\xi,\eta), \qquad \Longrightarrow \qquad \begin{cases} u = f(\xi,\eta) \\ v = g(\xi,\eta) \end{cases}$$

In the implementation an intermediate space (*i.e.* functions $f(\xi,\eta)$ and $g(\xi,\eta)$ is defined using the single-exponential function

$$r = \frac{e^{A\rho} - 1}{e^{A} - 1} \tag{1.5}$$

that maps $0 \le \rho \le 1$ into $0 \le r \le 1$. Note that A is a parameter selected by the user. The sign and magnitude of the parameter A allows to concentrate nodes near the desired position. Equation (5) becomes singular for A = 0. However for small values of $|A|$ function (1.5) can be approximated by the straight line $r = \rho$.

1.2 Implementation

The Matlab code `mainMesher.m` is the main function. It calls the function `linearTFI.m` which generates a structured quadrilateral mesh using linear TFI method. This code is divided into three parts. The first part creates boundary nodes, the second is for the creation of interior nodes and finally the third part creates the mesh. The task in the assignment is to code the missing parts in this function.

In the function `createBoundaryNodes`, in order to map the reference coordinates to the intermediate control domain, the function `gridControlSpacing.m` is used. In this function we are required to define the function `singleExp` based on Equation (1.5). This is coded as follows.

```
function psi = singleExp(psi_ini, A)
if A == 0
        psi = psi_ini;
else
        psi = (exp(A * psi_ini)-1)/(exp(A)-1);
end
```

In order to create the inner nodes, the function `createInnerNodes` is coded as

```
function [phi]=createInnerNodes(phi)
nOfChiNodes=size(phi,1);
nOfEtaNodes=size(phi,2);
chi=linspace(0,1,nOfChiNodes);
eta=linspace(0,1,nOfEtaNodes);

for j = 2:nOfEtaNodes-1
for i = 2: nOfChiNodes-1
[u0,v0] = gridControlSpacing(chi(i),eta(j));
u = U(u0,phi,j);
v = V(v0,phi,i);
uv = UV(u0,v0,phi);
phi(i,j,:) = u + v - uv;
end
end
```

Here, the functions U, V and UV are coded in accordance with Equations (1.2)-(1.4) as

```
function [p]=U(u,phi,j)
p = (1-u)*phi(1,j,:) + u*phi(size(phi,1),j,:);

function [p]=V(v,phi,i)
```

```
p = (1-v)*phi(i,1,:) + v*phi(i,size(phi,2),:);

function [p]=UV(u,v,phi)
p = (1-u)*(1-v)*phi(1,1,:) + (1-u)*v*phi(1,size(phi,2),:)...
          + u*(1-v)*phi(size(phi,1),1,:)...
          + u*v*phi(size(phi,1),size(phi,2),:);
```

The rest of the code is not modified and this code is run for different domains. The definition of the boundaries is done in the function `boundary.m`. Plotting of the mesh is done by the function `plotMesh.m`

1.3 Results

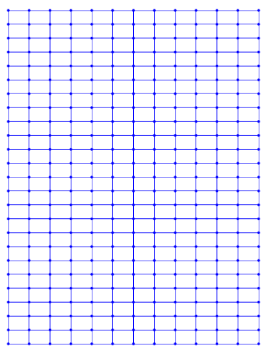
The code is used to mesh three different domains, the results for which are shown in Figures 1.1, 1.2 and 1.3. In all the meshes shown, there are 12 elements along the ξ direction and 24 elements along the η direction.

1.3.1 Structured mesh for a rectangular domain

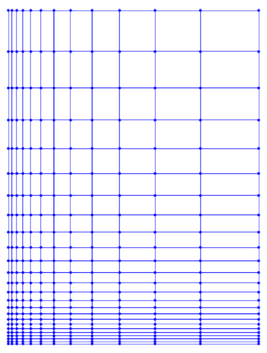
The rectangular domain is has height equal to 4 units and width equal to 3 units. The boundary is defined in the code as

```
function [p]=boundaryChi0Example1(eta)
p=[0,eta*height];
function [p]=boundaryChi1Example1(eta)
p=[width,eta*height];
function [p]=boundaryEta0Example1(chi)
p=[chi*width,0];
function [p]=boundaryEta1Example1(chi)
p=[chi*width,height];

function [value]=height()
value=4;

function [value]=width()
value=3;
```

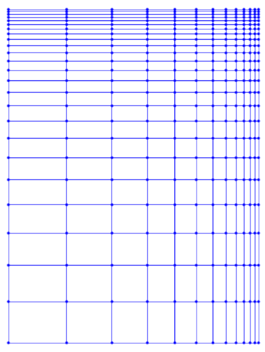
where `boundaryChi0Example1` corresponds to the physical space mapped from the line $\xi = 0$ in the reference space. Similarly `boundaryChi1Example1`, `boundaryEta0Example1` and `boundaryEta1Example1` correspond to the physical space mapped from the lines $\xi = 1$, $\eta = 0$ and $\eta = 1$, respectively, in the reference space.



(a) $\rho = 0$



(b) $\rho = 3$



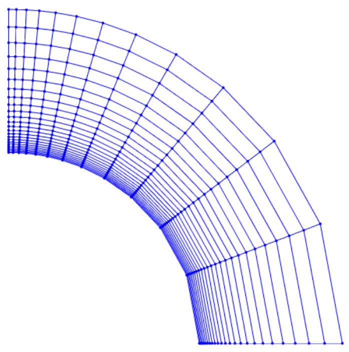
(c) $\rho = -3$

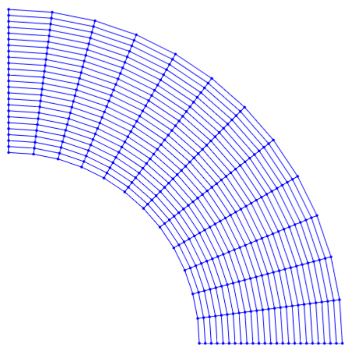
Figure 1.1: Structured mesh for a rectangular domain with different values of ρ.

1.3.2 Structured mesh for a domain of a quarter of a ring

This domain is one with inner radius equal to 4 units and outer radius equal to 3 units. The boundary is defined in the code as

```
function [p]=boundaryChi0Example2(eta)
p=[0,radiIn+eta*(radiOut-radiIn)];
function [p]=boundaryChi1Example2(eta)
p=[radiIn+eta*(radiOut-radiIn) 0];
function [p]=boundaryEta0Example2(chi)
p=[radiIn*cos((1-chi)*sector) radiIn*sin((1-chi)*sector)];
function [p]=boundaryEta1Example2(chi)
p=[radiOut*cos((1-chi)*sector) radiOut*sin((1-chi)*sector)];
```


(a) $\rho = 3$


(b) $\rho = 0$

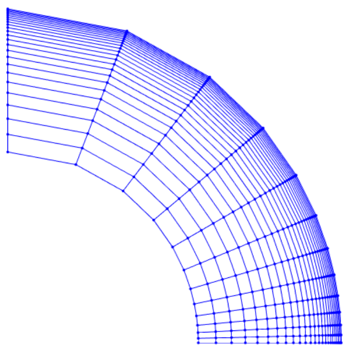
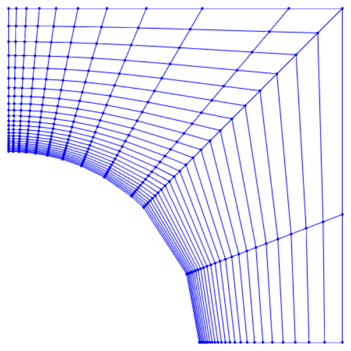
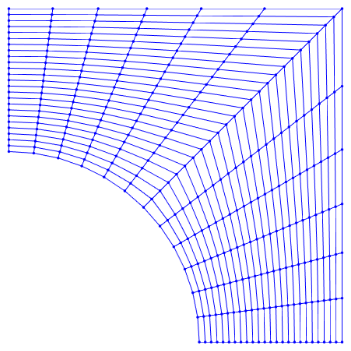

(c) $\rho = -3$

Figure 1.2: Structured mesh for a quarter of circular ring with different values of ρ.

1.3.3 Structured mesh for domain of a quarter of square with a circular central hole


(a) $\rho = 3$


(b) $\rho = 0$

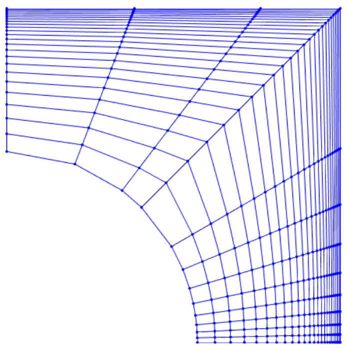

(c) $\rho = -3$

Figure 1.3: Structured mesh for a quarter of a square with a central hole with different values of ρ.

The side of the square is 7 units while the radius of the hole is 4 units. This boundary is defined as

```
function [p]=boundaryChi0Example3(eta)
p=[0,radiIn+eta*(radiOut-radiIn)];
function [p]=boundaryChi1Example3(eta)
p=[radiIn+eta*(radiOut-radiIn) 0];
function [p]=boundaryEta0Example3(chi)
p=[radiIn*cos((1-chi)*sector) radiIn*sin((1-chi)*sector)];
function [p]=boundaryEta1Example3(chi)
```

```
f = 1/max(abs(cos((1-chi)*sector)),abs(sin((1-chi)*sector)));
p=[radiOut*f*cos((1-chi)*sector) radiOut*f*sin((1-chi)*sector)];
```

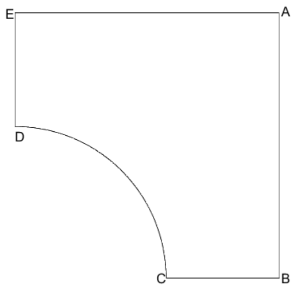


Figure 1.4: Domain of a quarter of a square with a central hole

In this case special case must be taken in order to place a node at the edge of the square marked 'A' in Figure 1.4. In the reference space this edge is located at coordinated $(\xi, \eta) = (0.5, 1)$. In order to manually place this node for any given spacing of the nodes the following routine is inserted into the code `createBoundaryNodes`

```
[u1,v1]=gridControlSpacing(chi(i),1);
if example == 3
        if test
                if u1>=0.5
                        u1 = 0.5;
                        u0 = 0.5;
                        test = 0;
                end
        end
end
```

This is done to ensure that the node corresponding to $(\xi, \eta) = (0.5, 1)$ is a node for any given spacing of nodes. As a result of this one can see in that Figures 1.3a and 1.3c, the spacing does not increase or decrease smoothly near the node at 'A'. Hence, this is not a very good fix. This can be improved, but is out of the scope of this assignment.