Master in Computational Mechanics	Computational Mechanics Tools
Assignment 1	Year 2019 - 2020
Transfinite Interpolation (TFI)	Due to the 11th November

1 Introduction

In this homework you will implement a 2D version of the Transfinite Interpolation (TFI) method. To this end we assume a computational domain (ξ, η) and a physical space (x, y) such that

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

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

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

where $I = 1, 2, \ldots, M+1$ and $J = 1, 2, \ldots, N+1$, being M and N the number of elements in the ξ and η directions respectively, see Figure 1.

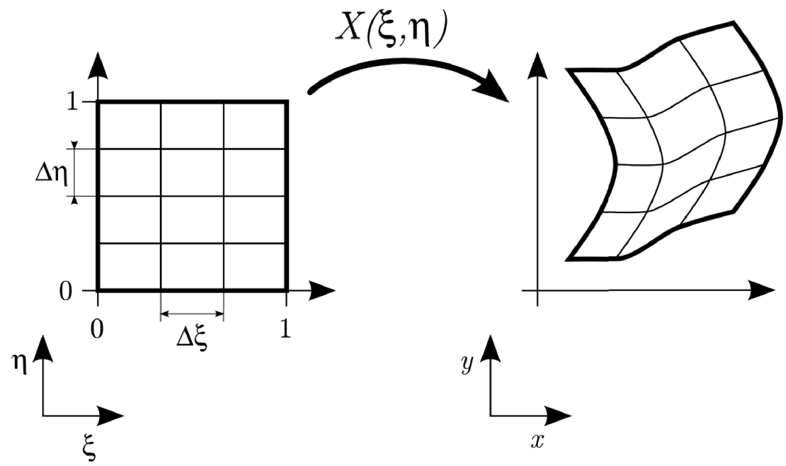

Figure 1: Mapping between computational and physical domain.

TFI uses an univariate interpolation in each direction of the computational space:

$$\mathbf{U}(\xi,\eta) = \sum_{i=1}^{2} \alpha_i(\xi)\,\mathbf{X}(\xi_i,\eta)$$

$$\mathbf{V}(\xi,\eta) = \sum_{j=1}^{2} \beta_j(\eta)\,\mathbf{X}(\xi,\eta_j)$$

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:

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

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

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

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

$$\mathbf{X}(\xi_I, \eta_J) = \mathbf{U}(\xi_I, \eta_J) \oplus \mathbf{V}(\xi_I, \eta_J) = \mathbf{U}(\xi_I, \eta_J) + \mathbf{V}(\xi_I, \eta_J) - \mathbf{UV}(\xi_I, \eta_J) \tag{1}$$

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

$$\mathbf{U}(\xi_I, \eta_J) = (1 - \xi_I)\mathbf{X}(0, \eta_J) + \xi_I \mathbf{X}(1, \eta_J) \tag{2}$$

$$\mathbf{V}(\xi_I, \eta_J) = (1 - \eta_J)\mathbf{X}(\xi_I, 0) + \eta_J \mathbf{X}(\xi_I, 1) \tag{3}$$

$$\begin{aligned} \mathbf{UV}(\xi_I, \eta_J) = {} & (1 - \xi_I)(1 - \eta_J)\mathbf{X}(0, 0) + (1 - \xi_I)\eta_J \mathbf{X}(0, 1) + \\ & \xi_I(1 - \eta_J)\mathbf{X}(1, 0) + \xi_I \eta_J \mathbf{X}(1, 1). \end{aligned} \tag{4}$$

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, see Figure 2:

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

In our implementation we will define the intermediate space (*i.e.* functions $f(\xi, \eta)$ and $g(\xi, \eta)$ using the single-exponential function:

$$r = \frac{e^{A\rho} - 1}{e^A - 1} \tag{5}$$

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 (5) can be approximated by the straight line $r = \rho$.

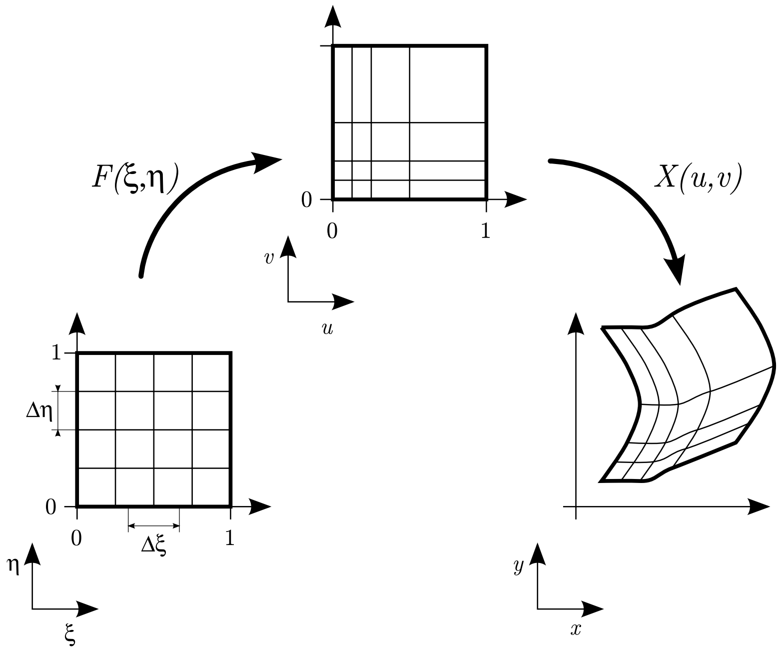

Figure 2: Intermediate control domain between the computational and physical domains.

2 Implementation details

Our implementation is composed by five files:

- `mainMesher.m` It is the main function and controls the execution flow of our application.
- `linearTFI.m` It implements the linear TFI method.
- `girdControlSpacing.m` It implements the definition of the intermediate space to control the spacing between points.
- `boundary.m` It defines the boundary of the geometry to be meshed.
- `plotMesh.m` It plots the final mesh on the screen.

Function `mainMesher` controls the flow of our application:

```
function [] = mainMesher( )
clear all;

[X,T]=linearTFI(12,24);

plotMesh(X,T,'qua',0)
```

where:

- function `linearTFI` generates a structured quadrilateral mesh using the linear TFI method (you will implement several parts of this method).
- function `plotMesh` plots a mesh on the screen (we provide a complete version of this function)

Function `linearTFI` is implemented as:

```
function [X,T] = linearTFI(nOfChiElems,nOfEtaElems)


nOfChiNodes=nOfChiElems+1;
nOfEtaNodes=nOfEtaElems+1;

phi=createBoundaryNodes(nOfChiNodes,nOfEtaNodes);
phi=createInnerNodes(phi);
[X,T]=createMesh(phi);
```

where

- Function `createBoundaryNodes` generates boundary nodes following the three steps depicted in Figure 2:
  - First, it generates a equidistributed set of points in the computational space (the (ξ, η)-space).
  - Second, it maps this set of points to the intermediate space (the (u, v)-space) using function `gridControlSpacing`. This function calls function `singleExp` that performs the mapping according to equation (5). You will code this function.
  - Third, it maps the intermediate coordinates to the physical space (the (x, y)-space) using function `boundary`. Function `boundary` defines the contour of the geometry for two cases: a rectangular domain (example 1 in the provided code), and a quarter of circular ring (example 2 in the provided code). We provide a complete version of this function. To use each example comment and uncomment the corresponding lines. Note that we implement this function because Matlab does not provide a graphical interface to define geometries.
- Function `createInnerNodes` generates points in the inner part of the geometry. You will code this function according to the code of function `createBoundaryNodes`. That is, for each inner node:
  - First, you compute its computational coordinates, (ξ, η).
  - Second, you compute its intermediate coordinates, (u, v), using (5).
  - Third, you compute its physical coordinates, (x, y), using equation (1). Hence, you will need to code the inivariate interpolants $\mathbf{U}$ and $\mathbf{V}$, and the tensor product $\mathbf{UV}$, see equations (2), (3) and (4) respectivelly.
- Function `createMesh` generates a standard representation of the mesh. That is, it generates the coordinate matrix `X` and the connectivity matrix `T` from an internal representation stored in the multi-array `Phi`. We provide a complete version of this function.

3 Tasks

1. In file `linearTFI.m` write the code corresponding to functions:

- `createInnerNodes`
- `U`
- `V`
- `UV`

2. In file `gridControlSpacing.m` write the code corresponding to function `singleExp`.

3. Generate a structured mesh using your application for:

- a rectangular domain of height equals 4 and width equals 3 (example 1 in `boundary.m` file).
- a quarter of circular ring of inner radii equals 4, outer radii equals 7 and angle equals $\pi/2$ (example 2 in `boundary.m` file).

For both examples present the obtained mesh using $A = 3$ and $A = -3$ when function `singleExp` is used to concentrate nodes in the ξ and η directions.

4. Apply the developed application to a new geometry. To this end modify file `boundary.m` and create a new domain. Present three meshes concentrating nodes near different boundaries