Assignment 1: Transfinit Interpolation (TFI)

1) In file linearTFi.m write the code corresponding to functions:

- createInnerNodes

```
function [phi]=createInnerNodes(phi)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to create the inner nodes of the domain
%   nOfChiElems => Number of elements in the chi direction
%   nOfEtaElems => Number of elements in the eta direction
%   phi         => Temporary multi-array to store the coordinates of grid
%                  points of dimension: nOfChiNodes x nOfEtaNodes x 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nOfChiNodes=size(phi,1);
nOfEtaNodes=size(phi,2);

% We compute the computational coordinates
chi=linspace(0,1,nOfChiNodes);
eta=linspace(0,1,nOfEtaNodes);

for i=2:nOfChiNodes-1
    for j=2:nOfEtaNodes-1
    % First, we create the intermediate coordinates
    [u,v]=gridControlSpacing(chi(i),eta(j));
    % Second, we compute the physical coordinates
    phi(i,j,:)=U(u,v)+V(u,v)-UV(u,v);
    end;
end;
end
```

- U

```
function [p]=U(u,v)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to compute the univariate blending function U for a linear TFI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

p=(1-u)*boundary(0,v)+u*boundary(1,v);

end
```

- V

```
function [p]=V(u,v)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to compute the univariate blending function V for a linear TFI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

p=(1-v)*boundary(u,0)+v*boundary(u,1);

end
```

- UV

```
function [p]=UV(u,v)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to compute the tensor product function UV of the
% univariate blending function U and V for a linear TFI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

p=(1-u)*(1-v)*boundary(0,0)+(1-u)*v*boundary(0,1)+u*(1-
v)*boundary(1,0)+u*v*boundary(1,1);

end
```

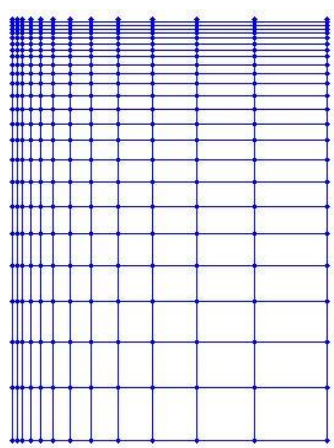
2) In file gridControlSpacing.m write the code corresponding to function singleExp:

```
function psi=singleExp(psi_ini, A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to distribute points in the computational domain
% according to the single exponential spacing control function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%My code
psi=(exp(psi_ini*A)-1)/(exp(A)-1);
end
```

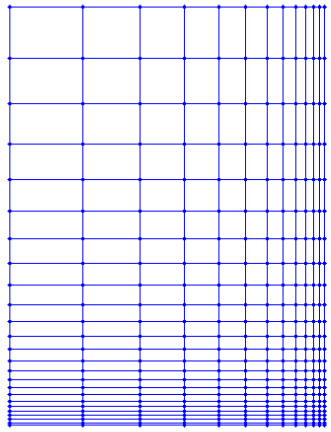
3) Generate a structued mesh using your application for:

- A rectangular domain of height equals 4 and width equals 3 (example 1 in boundary.m file)

Using A=3 and 12 divisions in ξ direction and A=-3 and 24 divisions in η direction:

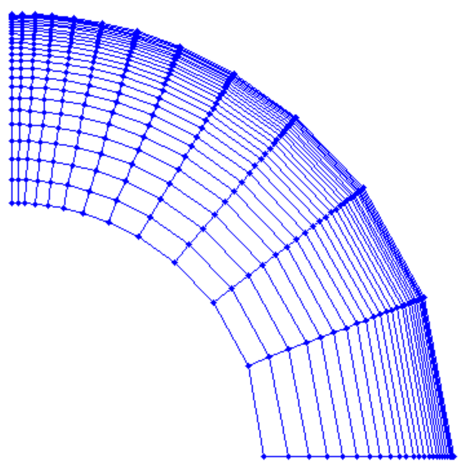


Using A=-3 and 12 divisions in ξ direction and A=3 and 24 divisions in η direction:

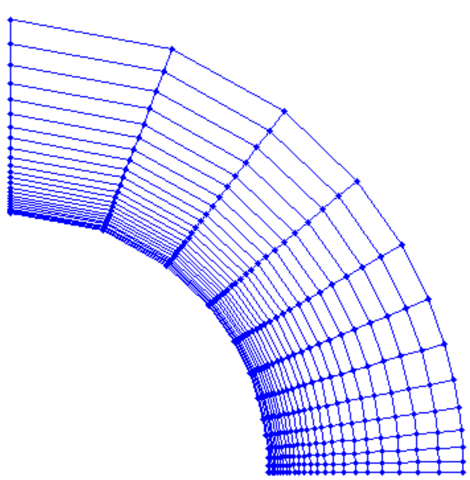


- A quarter of circular ring of inner radii equals 4, outer radii equals 7 and angle $\frac{\pi}{2}$ (example 2 in boundary.m file)

Using A=3 and 12 divisions in ξ direction and A=-3 and 24 divisions in η direction:



Using A=-3 and 12 divisions in ξ direction and A=3 and 24 divisions in η direction:



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

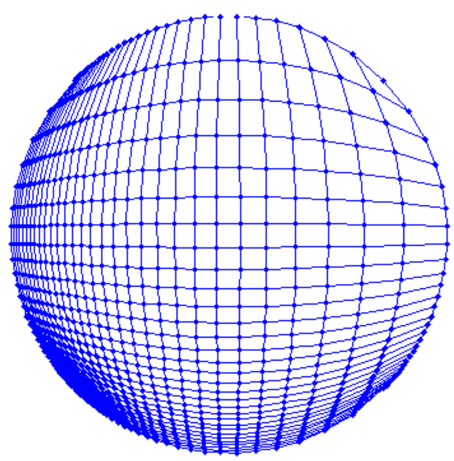
The new geometry created is a circle with radii 10. It may seem to not have four edges, but the TFI method can be used dividing the circle in 4 different edges. The main counterback using this method to mesh a circle is the 4 vertex nodes that will appear at the contour of the circle and the distortion of the mesh around these artificial vertexes.

The m function used to define the boundaries is:

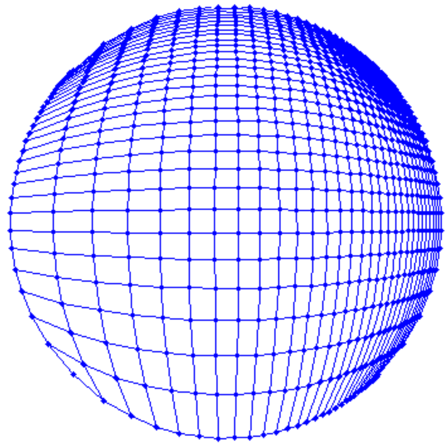
```
function [p]=boundaryCircle(chi,eta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Function to create the geometry (boundary) of the domain for the four
% sides of the representation in the intermediate space:
%        chi=0  (Chi0)
%        chi=1  (Chi1)
%        eta=0  (Eta0)
%        eta=1  (Eta1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
if chi==0
    p=boundaryChi0(eta);
elseif chi==1
    p=boundaryChi1(eta);
elseif eta==0
    p=boundaryEta0(chi);
elseif eta==1
    p=boundaryEta1(chi);
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CIRCLE Geometry: 4 edges with vertexes at chi and eta equal to +- %
&sqrt(R^2/2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [p]=boundaryChi0(eta)
   if eta<0.5
    p=[-x1+eta*2*x1,-sqrt(radi^2-(-x1+eta*2*x1)^2)];
    else
       p=[(eta-0.5)*2*x1,-sqrt(radi^2-((eta-0.5)*2*x1)^2)];
   end
end

function  [p]=boundaryChi1(eta)
    if eta<0.5
    p=[-x1+eta*2*x1,sqrt(radi^2-(-x1+eta*2*x1)^2)];
    else
       p=[(eta-0.5)*2*x1,sqrt(radi^2-((eta-0.5)*2*x1)^2)];
   end
end
```

```
function [p]=boundaryEta0(chi)
    if chi<0.5
    p=[-sqrt(radi^2-(-y1+chi*2*y1)^2),-y1+chi*2*y1];
    else
       p=[-sqrt(radi^2-((chi-0.5)*2*y1)^2),(chi-0.5)*2*y1];
   end
end

function [p]=boundaryEta1(chi)
    if chi<0.5
    p=[sqrt(radi^2-(-y1+chi*2*y1)^2),-y1+chi*2*y1];
    else
       p=[sqrt(radi^2-((chi-0.5)*2*y1)^2),(chi-0.5)*2*y1];
   end
end

function [value]=radi()
    value=10;
end

function [value]=x1()
    value=sqrt(radi^2/2);
end

function [value]=y1()
    value=x1;
end
```

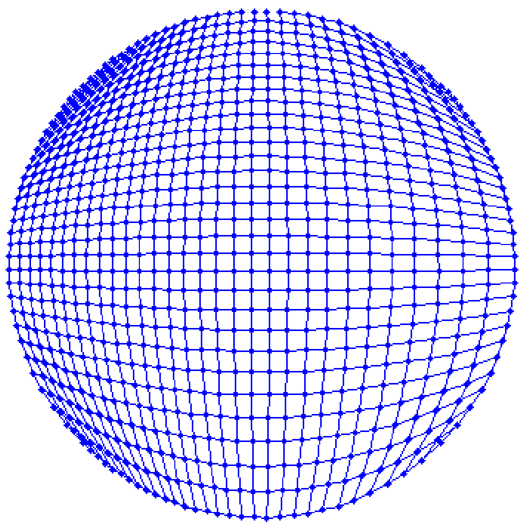
Using A=3 and 30 divisions in both ξ and η directions:



Using A=-3 and 30 divisions in both ξ and η directions:



Using 30 divisions in both ξ and η directions and A=-1 in ξ direction and A=1 in η direction:



Using A=0.01 and 30 divisions in both ξ and η directions: