A. Code

In this section, the code that has been implemented will be presented, with a brief description of it. The functions that have not been directly implemented, but have been used from previous work will also be described, explaining the input and the output of the function, but the actual code will not be presented.

Main

The main program is divided into three parts: preproces, solution computation and post-process. In the pre-process, the geometry of the problem is obtained and the mesh is generated. The characteristics of the element, such as the number of nodes per element or the degrees of freedom per node are also given by several functions. The levelset function is also determined, which will characterize the interface and the two subdomains. A modification of the mesh must be done in case that any node is close to the interface. When this happens, the node is moved to be on the interface, to avoid numerical errors. The code presented here has no change in the mesh, as it is not necessary.

In the solution computation part, three functions compute all the matrices that form the system of equations described in equation (15). One function computes the matrices corresponding to the Stokes equations, another function the matrices corresponding to the Darcy equations, and a last function computes the matrices relating to the interface. Then the system of equations is solved.

In the postproces part, the solution obtained before is reordered to obtain

the solution for the velocity, pressure and potential. Then the solution is plotted, as well as the errors obtained during the computation process.

```
% This program solves a 2D Stokes/Darcy problem

clear all, close all, home
setpath

global nu;global k;

nu = 1;
k=nu;
disp(' ' )
disp('2D STOKES FLOW. Problem with analytical solution')

% MIXED ELEMENT USED FOR THE SOLUTION (0 quad, 1 tri, 2 mini)

%reference element Stokes
%elementType = 0; nen=9; nenP=4;
elementTypeStokes = 2; nenStokes=4; nenPStokes=3;

%reference element Darcy
elementTypeDarcy=1; nenDarcy=3; grauDarcy=1; %linear triangles

% GEOMETRY
% Number of elements on each direction
%
nx=input('Number of elements in x direction: ');ny=2*nx-1;n=nx;
%nx = 5; ny=9;n=nx;
%nx = 10; ny=19;n=nx;
%nx = 20; ny=39; n=nx;
%nx = 40; ny=79; n=nx;
```

```
% Nodal coordinates matrix and connectivities for velocity
[X,T] = CreateUniformMesh(elementTypeStokes,nenStokes,0,1,0,2,nx+1,ny+1);
% Nodal coordinates matrix and connectivities for pressure
[XP,TP] = CreatePresNod(elementTypeStokes,nenPStokes,X,T,nx+1,ny+1);

% Total number of nodes and degrees of freedom
if elementTypeStokes == 2 && nenStokes == 4     % MINI
    numnp = size(X,1) + size(T,1); nunk = 2*numnp;
else
    numnp = size(X,1); nunk = 2*numnp;
end
ndofP = size(XP,1);

%Move the nodes close to the void

global levelsetTol;
levelsetTol=0.01*2/n;
x = X(:,1); y = X(:,2); R=0.3;
phi=y-1;
phip=y-1;

[nodesVoidStokes] = GetnodesVoid(X,T,phi,levelsetTol);
[bublesVoidStokes]= GetbublesVoid(X,T,phi,levelsetTol);

% Meshes plot
if 1
    figure('Name','Meshes','NumberTitle','off'); clf;
    set(gca, 'FontSize', 12,...
        'XTick', [0:0.25:1], 'YTick', [0:0.25:2]);
    axis([0,1,0,2])
    plotMeshESL(T,X,elementTypeStokes,'r'),     hold on
    plot(X(nodesVoidStokes,1),X(nodesVoidStokes,2),'bo')
    Plotbubles(bublesVoidStokes,T,X);
```

```
    figure('Name','Meshes','NumberTitle','off'); clf;
    plotMeshESL(TP,XP,elementTypeStokes,'g',1),
end

%Reference element
referenceElement=createReferenceElementStokesN(elementTypeStokes,nenStokes);
[pgauss,pesos,N,Nxi,Neta] = ElementoReferencia2D(elementTypeDarcy,grauDarcy);

% COMPUTATION OF MATRICES

[K,G,f,cutElements] = CreMatStokes(X,T,XP,TP,referenceElement,phi,nenDarcy);
phid=-phi;
[D,fp]=CreMatDarcy(X,T,XP,TP,pgauss,pesos,N,Nxi,Neta,phid,nenDarcy,nenStokes);
[ Mt,Bf,Bp ] = CreMatInterface( X,T,TP,phi,cutElements,nenStokes,nenDarcy );
fp=zeros(size(D,1),1);

nunkP = ndofP;

alpha=1;

Atot = [K+alpha*Mt  G' Bf
    G  zeros(nunkP,nunkP) zeros(nunkP,size(Bf,2))
    Bp zeros(size(D,1),nunkP) k*D];
btot = [f ; zeros(nunkP,1);fp];


% BOUNDARY CONDITIONS
%Boudary conditions at the limit of the domain

if 1
    [ unknownsCCD,uCCD ] = BoundaryConditions(X,T,nx,ny);
```

```
%Boundary conditions in the void
nodesVoidStokes = GetnodesVoid(X,T,phi,levelsetTol);
unknownP=nodesVoidStokes;
[bublesVoidStokes]= GetbublesVoid(X,T,phi,levelsetTol);
nodesVoidStokes=[nodesVoidStokes,bublesVoidStokes];
[nodesVoidDarcy]= GetnodesVoid(X,T,phid,levelsetTol);

unknownVoid=[2*nodesVoidStokes-1,2*nodesVoidStokes];
unknownsCCD = [unknownsCCD,unknownVoid];
uCCD = [uCCD ; zeros(length(unknownVoid),1)];

% System reduction (boundary conditions)
unknownP=unknownP+size(K,1);
unknownDarcy=nodesVoidDarcy+size(K,1)+size(G,1);
unknownsCCD=[unknownsCCD, unknownP,unknownDarcy];
uCCD = [uCCD ; zeros(length(unknownP),1);zeros(length(unknownDarcy),1)];
notCCD= setdiff(1:size(Atot,1),unknownsCCD);

bred = btot(notCCD)-Atot(notCCD,unknownsCCD)*uCCD;
Ared=Atot(notCCD,notCCD);

% SYSTEM SOLUTION
solaux = Ared\bred;
% Complete solution (with prescribed values)
sol = zeros(size(Atot,1),1);
sol(notCCD) = solaux; sol(unknownsCCD) = uCCD;


% POSTPROCESS
% Recover velocity field
velo = zeros(numnp,2);
velo = reshape(sol(1:nunk),2,numnp)';
```

```
    % Recover pressure field
    pres= sol(nunk+1:nunk+nunkP);
    % Recover phi Darcy
    PhiDarcy= sol(nunk+nunkP+1:end);
end

clear aux; %pack;

%Plot the solution of the system
% Velocity
if 0
    figure('Name','Velocity field','NumberTitle','off'); clf;
    box on; axis equal tight;
    set(gca, 'FontSize', 12,...
        'XTick', [0:0.25:1], 'YTick', [0:0.25:2]);
    axis([0,1,0,2])
    PlotCutMeshVelo(X,T,velo,phi);
end
% Pressure
if 0
    figure('Name','Pressure field','NumberTitle','off'); clf;
    PlotCutMeshPres(XP,TP,pres,phip)
end
%phiDarcy
if 0
    figure('Name','Pressure field','NumberTitle','off'); clf;
    PlotCutMeshPres(XP,TP,PhiDarcy,phid)
end

load error
[errorL2Stokes]=ErrorL2Stokes(velo,X,T,referenceElement,phi);
[errorL2Darcy]=ErrorL2Darcy(PhiDarcy,X,T,phid);
error=[error;nx errorL2Stokes errorL2Darcy];
```

```
save ('error','error')

% Plot computed velocity and analytical velocity
if 1
    figure('Name','Vx computed','NumberTitle','off'); clf;
        caxis([min(velo(:,1)) max(velo(:,1))]);
    PlotCutMeshPres(XP,TP,velo(:,1),phi),shading interp
    title('v_x')
    figure('Name','Vx analytical','NumberTitle','off'); clf;
        caxis([min(velo(:,1)) max(velo(:,1))]);
    u = analyticalVelocity(XP);
    PlotCutMeshPres(XP,TP,u(:,1),phi),shading interp
    title('v_x')
end

if 1
    figure('Name','Vy computed','NumberTitle','off'); clf;
        caxis([min(velo(:,2)) max(velo(:,2))]);
    PlotCutMeshPres(XP,TP,velo(:,2),phi),shading interp
    title('v_y computed')
    figure('Name','Vy analytical','NumberTitle','off'); clf;
        caxis([min(velo(:,2)) max(velo(:,2))]);
    u = analyticalVelocity(XP);
    PlotCutMeshPres(XP,TP,u(:,2),phi),shading interp
    title('v_y analytical')
end

%Plot computed phi and phi analytical
if 0
    figure('Name','Phi computed','NumberTitle','off'); clf;
        caxis([min(PhiDarcy) max(PhiDarcy)]);
    PlotCutMeshPres(XP,TP,PhiDarcy,phid),shading interp
    title('Phi computed')
```

```
    figure('Name','Phi analytical','NumberTitle','off'); clf;
        caxis([min(PhiDarcy) max(PhiDarcy)]);
    u = analyticalPotential(XP);
    PlotCutMeshPres(XP,TP,u,phid),shading interp
    title('Phi analytical')

end

%Error computation
[errorVelo,errorPhi]=Errors(X,XP,T,velo,PhiDarcy);
% Plot errors
if 0
    figure('Name','error Vx','NumberTitle','off'); clf; colorbar;
    PlotCutMeshPres(XP,TP,errorVelo(:,1),phi),shading interp
    title('error vx')
    figure('Name','error Vy','NumberTitle','off'); clf; colorbar;
    PlotCutMeshPres(XP,TP,errorVelo(:,2),phi),shading interp
    title('error vy')
end
if 0
    figure('Name','error Phi','NumberTitle','off'); clf; colorbar;
    PlotCutMeshPres(XP,TP,errorPhi,phid),shading interp
    title('error Phi')
end



if 0
    figure('Name','Error system Darcy','NumberTitle','off'); clf; colorbar;
    PlotCutMeshPres(XP,TP,ferrorDarcy,phid),shading interp
    title('Error system Darcy')
end
```

CreateUniformMesh

INPUT: Element type for the velocity, number of nodes per element for the velocity, first x coordinate, last x coordinate, first y coordinate, last y coordinate, number of nodes in x direction, number of nodes in y direction.

OUTPUT: Matrix of nodal coordinates X for velocity, matrix of connectivity T for velocity.

CreatePresNod

INPUT: Element type for the velocity, number of nodes per element for the velocity, matrix of nodal coordinates X for velocity, matrix of connectivity T for velocity, number of nodes in x direction, number of nodes in y direction.

OUTPUT: Matrix of nodal coordinates XP for pressure, matrix of connectivity TP for pressure.

GetnodesVoid

INPUT: Matrix of nodal coordinates X for velocity, matrix of connectivity T for velocity, value of levelset function on nodes, levelset tolerance.

OUTPUT: Index of the nodes that are in the void.

```
function [nodesVoid] = GetnodesVoid(X,T,phi,levelsetTol)
% x is the nodal coordinates
% T is connectivity matrix
% phi is the level set element
nodesDomain=[];
```

```
m=size(T,1); % m is the number of elements
for counter=1:m
    Te=T(counter,1:3);
    phie=phi(Te);
    if (any(phie>levelsetTol))
        nodesDomain=union(nodesDomain,Te);
    end
end

nodesVoid = setdiff([1:size(X,1)],nodesDomain);
end
```

GetbublesVoid

INNPUT: Matrix of nodal coordinates X for velocity, matrix of connectivity T for velocity, value of levelset function on nodes, levelset tolerance.

OUTPUT: Index of the interior nodes that are in the void.

```
function [bublesVoid]=GetbublesVoid(X,T,phi,levelsetTol)
% x is the nodal coordinates
% T is connectivity matrix
% phi is the level set element
bublesVoid=[];
m=size(T,1); % m is the number of elements
for counter=1:m
    Te=T(counter,1:3);
    Tee=T(counter,4);
    phie=phi(Te);
    if (all(phie<levelsetTol))
```

```
        bublesVoid=union(bublesVoid,Tee);
    end
end
```

createReferenceElementStokesN

INPUT:Type of element and number of nodes per element.

OUTPUT: Structure that contains all the characteristics for the Stokes domain element, such as the coordinates and weights of the Gauss points, the shape function and its derivatives on the Gauss points, and the spatial coordinates of the nodes.

ElementoReferencia2D

INPUT: Type of element.

OUTPUT: Coordinates and weights for the Gauss points, shape function and its derivatives on Gauss points for the element used in the Darcy domain.

CreMatStokes

INPUT: Matrix of nodal coordinates X and XP, matrix of conectivity T and TP, position of the Gauss points and its weight when the element is not cut by the interface pgauss and pesos, shape functions and its derivatives on the Gauss points N, Nxi and Neta, value of the levelset function on the nodes phi, number of nodes per element for the velocity element nenStokes, number of nodes per element for the potential element nenDarcy.

OUTPUT: Matrices K and G, corresponding to matrix $\boldsymbol{K}$ and $\boldsymbol{G}$ from

equation 11, force term f, corresponding to $\boldsymbol{f}_f$ from equation 11 and a list with the element number of the cut element.

```
function [K,G,f,cutElements] =
=CreMatStokes(X,IEN,XP,IENP,referenceElement,phi,nenDarcy)
% [K,G,f] = CreMat(X,IEN,XP,IENP,referenceElement)
% This function computes viscosity matrix K, gradient matrix G
% and vector f (source term) obtained by discretizing the Galerkin
% weak form of the Stokes problem in 2D using finite elements.
%
% Input:
%
%   X:      nodal coordinates (velocity)
%   IEN:    connectivities (velocity)
%   XP:     nodal coordinates (pressure)
%   IENP:   connectivities (pressure)
%
%


cutElements = [];
elem = referenceElement.elementType;
global levelsetTol;
global dl;
% Total number of elements and element node's number
% for velocity and pressure
[numel,nen] = size(IEN);
[numelP,nenP] = size(IENP);
% Total number of nodes
if elem == 2 & nen == 4 % MINI
    numnp = size(X,1) + numel;
```

```
else
    numnp = size(X,1);
end
% Degrees of freedom
nunk  = 2*numnp; ndofn = 2*nen; ndofnp=nenDarcy;
nunkP = size(XP,1);

wpg=referenceElement.IPweights;
pospg=referenceElement.IPcoordinates;
N=referenceElement.N;
Nxi=referenceElement.Nxi;
Neta=referenceElement.Neta;
NP=referenceElement.NP;
ngeom = referenceElement.nOfElementNodesGeometry;

% Allocates storage for matrices
if nen <= 4
    K = spalloc(nunk,nunk,10*nunk);
    G = spalloc(nunkP,nunk,10*nunk);
else
    K = spalloc(nunk,nunk,20*nunk);
    G = spalloc(nunkP,nunk,20*nunk);
end
f = zeros(nunk,1);


% Loop on elements
for ielem = 1:numel
    % Te: current velocity element
    Te = reshape([2*IEN(ielem,:)-1; 2*IEN(ielem,:)],1,ndofn);
    % TeP: current pressure element
    TeP = IENP(ielem,:);
    % Xe: element nodes' coordinates
```

```
    Xe = X(IEN(ielem,1:ngeom),:);

    Tee=IEN(ielem,1:3); %notes the index of the item
    phie=phi(Tee);
    % if cut element
    if(any(phie>levelsetTol) && any(phie<=levelsetTol))
        cutElements = [cutElements, ielem];
       [www,XX,NPc,Nc,Nxic,Netac,condition,nOfIntPoints,coord]=
            =cutReferenceElementStokesMini(phie);
        wpgc=www(1:nOfIntPoints);% weight
        pospgc=XX(1:nOfIntPoints,:);% coordinates of gauss points
        Nc=Nc(1:nOfIntPoints,:);
        Ncxi=Nxic(1:nOfIntPoints,:);
        Nceta=Netac(1:nOfIntPoints,:);
       %[s1e,sxe,sye,Me]=calcul_elemental_mass(Xe,pgaussc,pesosc,Nc,Ncxi,Nceta);
       [Ke,Ge,fe] = EleMat(Xe,ngeom,ndofn,pospgc,wpgc,Nc,Ncxi,Nceta,nenP,NPc);
    elseif all(phie>=levelsetTol)
        %[s1e,sxe,sye,Me]=calcul_elemental_mass(Xe,pgauss,pesos,N,Nxi,Neta);
        [Ke,Ge,fe] = EleMat(Xe,ngeom,ndofn,pospg,wpg,N,Nxi,Neta,nenP,NP);
    end
    if any(phie>levelsetTol)
    K(Te,Te) = K(Te,Te) + Ke;
    G(TeP,Te)= G(TeP,Te)+ Ge;
    f(Te) = f(Te) + fe;
    %clear Ke Ge fe;
        %s1 = s1 + s1e;
        %sx = sx + sxe;
        %sy = sy + sye;
    end
end

K = sparse(K);
G = sparse(G);
```

```
end

%-------------------------------------------------------------------------
function [Ke,Ge,fe] = EleMat(Xe,ngeom,ndofn,pospg,wpg,N,Nxi,Neta,nenP,NP)

% [Ke,Ge,fe] = EleMat(Xe,ngeom,ndofn,pospg,wpg,N,Nxi,Neta,nenP,NP)
% This function computes element matrices obtained for the 2D Stokes proble
%
% Input:
%   Xe:           Element nodes' coordinates
%   ngeom:        Number of nodes describing element geometry
%                 (it is the number of element nodes except in PI+,
%                                   which has 4 nodes and ngeom=3)
%   ndofn:        Number of degrees of freedom for velocity
%   pospg, wpg:   Gauss points and weights int he reference element
%   N,Nxi,Neta:   Shape functions for velocity and its derivatives on
%                                   Gauss points (local coordinates)
%   nenP:         Number of element nodes for the pressure field
%   NP:           Shape functions for the pressure field
%

global nu

% Allocation
Ke = zeros(ndofn,ndofn);
Ge = zeros(nenP,ndofn);
fe = zeros(ndofn,1);

% Number of Gauss points
ngaus = size(pospg,1);

% Loop on Gauss points
for igaus = 1:ngaus
```

```
    % Shape functions on Gauss point igaus
    N_igaus    = N(igaus,:);
    Nxi_igaus  = Nxi(igaus,:);
    Neta_igaus = Neta(igaus,:);
    NP_ig      = NP(igaus,:);
    % Jacobian matrix on the Gauss point
    Jacob =
        =[Nxi_igaus(1:ngeom)*(Xe(1:ngeom,1))     Nxi_igaus(1:ngeom)*(Xe(1:ngeom,2))
      Neta_igaus(1:ngeom)*(Xe(1:ngeom,1)) Neta_igaus(1:ngeom)*(Xe(1:ngeom,2))];
    %
    dvolu=wpg(igaus)*det(Jacob);
    % Shape functions' derivatives in global coordinates
    res = Jacob\[Nxi_igaus;Neta_igaus];
    % Gradient
    Nx = [reshape([1;0]*res(1,:),1,ndofn); reshape([0;1]*res(1,:),1,ndofn)];
    Ny = [reshape([1;0]*res(2,:),1,ndofn); reshape([0;1]*res(2,:),1,ndofn)];
    % Divergence
    dN = reshape(res,1,ndofn);
    % Contribution to element matrices
    Ke = Ke + nu*(Nx'*Nx+Ny'*Ny)*dvolu;
    Ge = Ge - NP_ig'*dN*dvolu;
    aux = N_igaus(1:ngeom)*Xe;
    f_igaus = SourceTerm(aux);
    Ngp = [reshape([1;0]*N_igaus,1,ndofn); reshape([0;1]*N_igaus,1,ndofn)];
    fe = fe + Ngp'*f_igaus*dvolu;
end
end
```

cutReferenceElementStokesMini

INPUT: Value of the levelset function on the nodes of the element.

OUTPUT: Weight and position of the Gauss points in the cut element, shape function values and its derivatives on the Gauss points, condition number that describes how the interface cuts the element, number of integration points needed to compute the integration, and coordinates of the points where the interface cuts the element.

SourceTerm

INPUT: list of points xc.

OUTPUT:value of the source term in such points s.

```
function s = SourceTerm(xc)
% s = SourceTerm(xc)
%

x = xc(1); y = xc(2);
s = zeros(2,1);

%Solucin 1
%s(1)=0;s(2)=0;


%Solucin 2
s(1)=1;s(2)=0;
```

CreMatDarcy

INPUT: Matrix of nodal coordinates X and XP, matrix of conectivity T and TP, position of the Gauss points and its weight when the element is not

cut by the interface pgauss and pesos, shape functions and its derivatives on the Gauss points N, Nxi and Neta, value of the levelset function on the nodes phi, number of nodes per element for the velocity element nenStokes, number of nodes per element for the potential element nenDarcy.

OUTPUT: Matrix K, corresponding to matrix $\boldsymbol{K}$ from equation 13, and force term f, that in this case $f = 0$.

```
function [K,f]=CreMatDarcy(X,T,XP,TP,pgauss,pesos,N,Nxi,Neta,phi,nenDarcy,nenStokes

global levelsetTol;

nnodos = size(XP,1);
nelem = size(TP,1);

K=spalloc(nnodos,nnodos,7*nnodos);
f=zeros(nnodos,1);


%Loop elements
for i=1:nelem
    TeP=T(i,1:3); %notes the index of the item
    Xe=X(TeP,:); %coordinates of the element nodes
    Te = reshape([2*T(i,:)-1; 2*T(i,:)],1,2*nenStokes);
    phie=phi(TeP);
    % if cut element
    if(any(phie>levelsetTol) & any(phie<-levelsetTol))
       [www,XX,Nc,Nxic,Netac,condition,nOfIntPoints,coord]=
            =CutReferenceElement(phie);
        pesosc=www(1:nOfIntPoints);% weight
```

```
        pgaussc=XX(1:nOfIntPoints,:);% coordinates of gauss points
        Nc=Nc(1:nOfIntPoints,:);
        Ncxi=Nxic(1:nOfIntPoints,:);
        Nceta=Netac(1:nOfIntPoints,:);
        [Ke,fe]=calcul_elemental(Xe,pgaussc,pesosc,Nc,Ncxi,Nceta);
    elseif all(phie>=0)
        [Ke,fe]=calcul_elemental(Xe,pgauss,pesos,N,Nxi,Neta);
    end
    if any(phie>levelsetTol)
        K(TeP,TeP)=K(TeP,TeP)+Ke; %assembly
        f(TeP) = f(TeP) + fe;
    end
end

%----------------------------------------
%Calculate the matrix and vector elementary
function [Ke,fe]=calcul_elemental(Xe,pgauss,pesos,N,Nxi,Neta)

nnodes = size(Xe,1);
Ke=zeros(nnodes);
fe=zeros(nnodes,1);
xe = Xe(:,1); ye = Xe(:,2);
%Loop integration points
for k=1:length(pesos)
    Nk=N(k,:);
    Nkxi=Nxi(k,:);
    Nketa=Neta(k,:);
    xk = Nk*Xe;
    %Jacobia
    J = [Nkxi*xe Nkxi*ye;Nketa*xe Nketa*ye];
    %Derivatives of shape functions with respect to (x, y)
    res = J\[Nkxi;Nketa];
    Nkx = res(1,:); Nky = res(2,:);
```

```
    %differential volume
    dxy=pesos(k)*det(J);
    Ke = Ke + (Nkx'*Nkx + Nky'*Nky)*dxy;
end
end
end
```

CutReferenceElement

INPUT: Value of the levelset function on the nodes of the element.

OUTPUT: Weight and position of the Gauss points in the cut element, shape function values and its derivatives on the Gauss points, condition number that describes how the interface cuts the element, number of integration points needed to compute the integration, and coordinates of the points where the interface cuts the element.

CreMatInterface

INPUT: Matrix of nodal coordinates X, matrix of connecivity T and TP, values of the levelset function on the nodes phi, list with the index of the cut elements cutElements, number of nodes per element for the velocity element nenStokes, number of nodes per element for the potential element nenDarcy.

OUTPUT: Matrices Mt, Bf and Bp, corresponding to the matrices $\boldsymbol{M_t^I}$, $\boldsymbol{B_f^I}$ and $\boldsymbol{B_p^I}$ respectively from equation 15

```
function [ Mt,Bf,Bp ] =
= CreMatInterface( X,T,TP,phi,cutElements,nenStokes,nenDarcy )
```

```
%CREMATINTERFACE Summary of this function goes here
%   Detailed explanation goes here
numelements=size(T,1);
ndofv=2*nenStokes;
ndofd=nenDarcy;
n=2*(size(X,1)+size(T,1));
Mt=spalloc(n,n,3*n);
Bf=spalloc(n,size(X,1),3*n);
Bp=spalloc(size(X,1),n,3*n);
for i=1:numelements
    Te = reshape([2*T(i,:)-1; 2*T(i,:)],1,ndofv);
    Tep = TP(i,:);
    Xe=X(Tep,:);
    if any(i==cutElements)
        Tee=T(i,1:3);
        phie=phi(Tee);
        [www,XX,N,Nxi,Neta,condition,nOfIntPoints,coord]=
                =CutReferenceElement(phie);
        xi=[];
        eta=[];

        pcut1=coord(1,:);xi=[xi; pcut1(1)];eta=[eta; pcut1(2)];
        pcut2=coord(2,:);xi=[xi; pcut2(1)];eta=[eta; pcut2(2)];

        Nn= [1-(xi+eta),xi,eta];

        pcut1global= [Nn(1,:)*Xe(:,1),Nn(1,:)*Xe(:,2)];
        pcut2global= [Nn(2,:)*Xe(:,1),Nn(2,:)*Xe(:,2)];

        t=pcut2global-pcut1global;
        modt=sqrt(t*t');
        t=(1/modt)*t;
        n=[t(2),-t(1)];
```

```
        pg=[-sqrt(1/3);sqrt(1/3)];
        wg=[1;1];

        xi=((pcut2(1)+pcut1(1))/2)+((pcut2(1)-pcut1(1))/2)*pg;
        eta=((pcut2(2)+pcut1(2))/2)+((pcut2(2)-pcut1(2))/2)*pg;

        phiprima=(pcut2global-pcut1global)/2;
        mod=sqrt(phiprima*phiprima');

        Nn= [1-(xi+eta),xi,eta,27*xi.*eta.*(1-xi-eta)];
        Nt=[Nn(:,1)*t(1),Nn(:,1)*t(2),Nn(:,2)*t(1),Nn(:,2)*t(2),Nn(:,3)*t(1),Nn(:,3
        NN=[Nn(:,1)*n(1),Nn(:,1)*n(2),Nn(:,2)*n(1),Nn(:,2)*n(2),Nn(:,3)*n(1),Nn(:,3
        Nd = [1-xi-eta,   xi,   eta];

        Mte=CalcMt(ndofv,Nt,mod,pg,wg);
        Bfe = CalcBf(ndofv,ndofd,NN,Nd,mod,pg,wg);
        Bpe = CalcBp(ndofv,ndofd,Nd,NN,mod,pg,wg);

        Mt(Te,Te)=Mt(Te,Te)+Mte;
        Bf(Te,Tep)=Bf(Te,Tep)+Bfe;
        Bp(Tep,Te)=Bp(Tep,Te)+Bpe;
        %Bp(Tep,Te)=Bp(Tep,Te)-Bpe;
    end
end

end
function [ Mte ] = CalcMt(ndofn,Nt,mod,pg,wg)
%CALCMT Summary of this function goes here
%   Mte= matrix of the equation [K+alpha*Mt]*u+G*p-Bf*phi=f
%   coord=coordinates of the cut points coord =[x1 y1; x2 y2; ...]

Mte=zeros(ndofn,ndofn);
```

```
for i=1:length(pg)
    Mte= Mte + Nt(i,:)'*Nt(i,:)*wg(i)*mod;
end

end

function [ Bfe ] = CalcBf(ndofnf,ndofnp,NN,Nd,mod,pg,wg)
%CALCBF Calcula la matriz elemental Bf
%   Detailed explanation goes here

Bfe=zeros(ndofnf,ndofnp);

for i=1:length(pg)
    Bfe= Bfe + NN(i,:)'*Nd(i,:)*wg(i)*mod;
end
end
function [ Bpe ] = CalcBp(ndofnf,ndofnp,Nd,NN,mod,pg,wg)
%CALCBP Summary of this function goes here
%   xi,eta= valores de xi y eta en los puntos de Gauss

Bpe=zeros(ndofnp,ndofnf);

for k=1:size(pg,1)
    Bpe= Bpe + Nd(k,:)'*NN(k,:)*wg(k)*mod;
end
end
```

BoundaryConditions

INPUT: Matrix of nodal coordinates X, matrix of connectivity T, number of elements in x direction nx, number of elements in y direction y.

OUTPUT: Index of the unknowns that have a boundary condition unknownsCCD, vaule of the boundary condition in such nodes uCCD.

```
function [ unknownsCCD,uCCD ] = BoundaryConditions(X,T,nx,ny)
% Funtion that imposes the Dirichlet boundary conditions

[unknownsCCDstokes,uCCDstokes]=setDirichletBCStokes(X,ny);
[unknownsCCDdarcy,uCCDdarcy]=setDirichletBCDarcy(X,ny);
l=3*size(X,1)+2*size(T,1);
unknownsCCD=[unknownsCCDstokes,unknownsCCDdarcy+l];
uCCD=[uCCDstokes; uCCDdarcy];
end
function [ind,val]=setDirichletBCStokes(X,ny)

x = X(:,1); y=X(:,2);
tol = 1.e-16;
hy=2/ny;
%Nodes on the edges of the domain
nodesCD = find( (abs(x)<tol | abs(x-1)<tol | abs(y-2)<tol) & (y-(1-hy))>tol )';

ind = [2*nodesCD-1,2*nodesCD];
u = analyticalVelocity(X(nodesCD,:));
val = [u(:,1) ; u(:,2)];
end
function [ind,val]=setDirichletBCDarcy(X,ny)

x = X(:,1); y=X(:,2);
tol = 1.e-16;
hy=2/ny;
%Nodes on the edges of the domain
nodesCD = find( (abs(x)<tol | abs(x-1)<tol | abs(y)<tol)& (y-(1+hy))<tol )';
```

```
ind = nodesCD;
val = analyticalPotential(X(nodesCD,:));

end
```

ErrorL2Stokes

INPUT: Velocity solution u, matrix of nodal coordinates X, matrix of connectivity T, structure with the characteristics of the Stokes reference element referenceElement, values of the levelset function on the nodes phi.

OUTPUT: Error in L_2 norm for the velocity in the Stokes domain.

```
function [errorL2]=ErrorL2Stokes(u,X,T,referenceElement,phi)
global levelsetTol
errorL2=0;
normL2=0;

wpg=referenceElement.IPweights;
pospg=referenceElement.IPcoordinates;
N=referenceElement.N;
Nxi=referenceElement.Nxi;
Neta=referenceElement.Neta;
NP=referenceElement.NP;
ngeom = referenceElement.nOfElementNodesGeometry;

%Element loop
for i=1:size(T,1)
    Te=T(i,:);
```

```
Tee=T(i,1:3);
phie=phi(Tee);
Xe=X(Tee,:);
xe=Xe(:,1);ye=Xe(:,2);
ue=u(Te,:);
if all(phie>levelsetTol)
    Xg=[N(:,1:3)*xe, N(:,1:3)*ye];
    u_g=analyticalVelocity(Xg);
    %Gauss point loop
    for g=1:length(wpg)
        %Shape funtion and derivatives on Gauss point
        N_g = N(g,:);
        uh_g=N_g*ue;
        Nxi_g = Nxi(g,1:3);
        Neta_g = Neta(g,1:3);
        %Jacobi
        J = [Nxi_g*xe     Nxi_g*ye
            Neta_g*xe  Neta_g*ye];
        %Weight on Gauss point
        dvolu=wpg(g)*det(J);

        errorL2 =errorL2 +(uh_g-u_g(g,:))*(uh_g-u_g(g,:))'*dvolu;
        normL2 =normL2 +(u_g(g,:))*(u_g(g,:))'*dvolu;
    end
elseif(any(phie>levelsetTol) && any(phie<-levelsetTol))
    [www,XX,NPc,Nc,Nxic,Netac,condition,nOfIntPoints,coord]=
            =cutReferenceElementStokesMini(phie);
    wpgc=www(1:nOfIntPoints);% weight
    pospgc=XX(1:nOfIntPoints,:);% coordinates of gauss points
    Nc=Nc(1:nOfIntPoints,:);
    Ncxi=Nxic(1:nOfIntPoints,:);
    Nceta=Netac(1:nOfIntPoints,:);
    Xg=[N(:,1:3)*xe, N(:,1:3)*ye];
```

```
        u_g=analyticalVelocity(Xg);
        %Gauss point loop
        for g=1:length(wpgc)
            %Shape funtion and derivatives on Gauss point
            N_g = Nc(g,:);
            uh_g=N_g*ue;
            Nxi_g = Ncxi(g,1:3);
            Neta_g = Nceta(g,1:3);
            %Jacobi
            J = [Nxi_g*xe     Nxi_g*ye
                Neta_g*xe  Neta_g*ye];
            %Weight on Gauss point
            dvolu=wpgc(g)*det(J);
            errorL2 =errorL2 +(uh_g-u_g(g,:))*(uh_g-u_g(g,:))'*dvolu;
            normL2 =normL2 +(u_g(g,:))*(u_g(g,:))'*dvolu;
        end
    end
end

errorL2=sqrt(errorL2)/sqrt(normL2);
end
```

ErrorL2Darcy

INPUT: Potential solution u, matrix of nodal coordinates X, matrix of connectivity T, values of the levelset function on the nodes phi.

OUTPUT: Error in L_2 norm for the potential in the Darcy domain.

```
function [errorL2]=ErrorL2Darcy(u,X,T,phi)
```

```
global levelsetTol;
[pospg,pespg,N,Nxi,Neta] = ElementoReferencia2D(3,1);

nelem = size(T,1);
errorL2 = 0; normL2 = 0;
%Element loop
for i=1:nelem
    Te=T(i,:);%Nodal index of the element
    Tee=T(i,1:3);
    Xe=X(Tee,:); %Nodal coordinates of the element
    phie=phi(Tee);
    xe = Xe(:,1); ye=Xe(:,2);
    Ue=u(Tee); %Nodal values of the element
    if(any(phie>levelsetTol) & any(phie<-levelsetTol))
        [www,XX,Nc,Nxic,Netac,condition,nOfIntPoints,coord]=
                =CutReferenceElement(phie);
        pesosc=www(1:nOfIntPoints);% weight
        pgaussc=XX(1:nOfIntPoints,:);% coordinates of gauss points
        Nc=Nc(1:nOfIntPoints,:);
        Ncxi=Nxic(1:nOfIntPoints,:);
        Nceta=Netac(1:nOfIntPoints,:);
        %Shape functions and derivatives on Gauss pints
        for g=1:length(pesosc)
            Nc_g = Nc(g,:);
            Ncxi_g = Ncxi(g,:);
            Nceta_g = Nceta(g,:);
            %Jacobi
            J = [Ncxi_g*xe    Ncxi_g*ye
                Nceta_g*xe  Nceta_g*ye];
            %Weight on Gauss point
            dvolu=pesosc(g)*det(J);
            xg = Nc_g*Xe(:,1);yg=Nc_g*Xe(:,2);Xg=[xg, yg];
            ug = analyticalPotential(Xg);
```

```
            errorL2 = errorL2 + (ug-Nc_g*Ue)^2*dvolu;
            normL2 = normL2 + (ug)^2*dvolu;
        end
    elseif all(phie>=0)
        for g=1:length(pespg)
            %Shape functions and derivatives on Gauss pints
            N_g = N(g,:);
            Nxi_g = Nxi(g,:);
            Neta_g = Neta(g,:);
            %Jacobi
            J = [Nxi_g*xe     Nxi_g*ye
                Neta_g*xe  Neta_g*ye];
            %Weight on Gauss point
            dvolu=pespg(g)*det(J);
            xg = N_g*Xe(:,1);yg=N_g*Xe(:,2);Xg=[xg, yg];
            ug = analyticalPotential(Xg);
            errorL2 = errorL2 + (ug-N_g*Ue)^2*dvolu;
            normL2 = normL2 + (ug)^2*dvolu;
        end
    end
end
errorL2 = sqrt(errorL2)/sqrt(normL2);
```

PlotCutMeshPres

INPUT: Matrix of nodal coordinates X, matrix of connectivity T, pressure values (or any 1D variable) pres, levelset values on the nodes phi.

OUTPUT: Figure that plots the values of the variable pres in a mesh with cut elements.

```
function  PlotCutMeshPres(X,T,pres,phi)
%PLOTCUTMESHVELO Dibuja la malla cortada
%   Plots the mesh with cut elements

nnodes=size(X,1);   % number of nodes
nelements=size(T,1);    % number of elements

%tri = delaunay(XP(:,1),XP(:,2));
%trisurf(tri,XP(:,1),XP(:,2),pres);
%view(3); grid on;
%set(gca, 'FontSize', 12,...
 %       'XTick', [0:0.25:1], 'YTick', [0:0.25:1]);
hold on
%R=0.3;
%aux=0:0.1:2*pi;
%plot(R*cos(aux)+0.5,R*sin(aux)+0.5,'g-','LineWidth',2);
for e = 1:nelements

    Te = T(e,1:3); %Obtains the three nodes of the trinagle,
                                          %without considering the node in the middle
    Xe = X(Te,:);  %Coordinates of the three nodes of the element
    phie = phi(Te); %Obtains the levelset value on the nodes


    if(all(phie>0)) %all nodes in the domain
        %disp('interior element')
        prese = pres(Te);
        tri=delaunay(Xe(:,1),Xe(:,2));
        trisurf(tri,Xe(:,1),Xe(:,2),prese);

    elseif (any(phie>0)) %cut element
        [cond,coord,nOfIntPoints]=Choosecond1(phie);
```

```
%disp(sprintf('cond=%d',cond))
if(cond==1 | cond==2 | cond==3) %the domain area is one triangle
    xi = coord(:,1); eta = coord(:,2);
    N = [1-xi-eta,   xi,   eta];
    Xcut = [Xe(cond,:);N*Xe]; %N*Xe= N(2x3) * Xe(3x2)= (2x2)
    prese = pres(Te,:);
    prescut = [prese(cond,:);N*prese];
    if ~ (all(Xcut(1,:)==Xcut(2,:))| all(Xcut(2,:)==Xcut(3,:))
                |all( Xcut(1,:)==Xcut(3,:)))
    tricut=delaunay(Xcut(:,1),Xcut(:,2));
    trisurf(tricut,Xcut(:,1),Xcut(:,2),prescut);
    end

else %the domain area is a quadrilateral (divide the quad into
    %two triangles, only for plot)
    coord=coord([2,1],:);
    switch(cond)
        case 4, aux = [2,3];
        case 5, aux = [3,1];
        case 6, aux = [1,2];
    end

    xi = coord(:,1); eta = coord(:,2);
    N = [1-xi-eta,   xi,   eta];
    Xcut = [Xe(aux,:);N*Xe];
    Xcut1=Xcut(1:3,:);
    Xcut2=Xcut([3,4,1],:);
    prese = pres(Te,:);
    prescut = [prese(aux,:);N*prese];
    prescut1=prescut(1:3);
    prescut2=prescut([3,4,1]);
    if ~ (all(Xcut1(1,:)==Xcut1(2,:))| all(Xcut1(2,:)==Xcut1(3,:))
```

```
                    |all( Xcut1(1,:)==Xcut1(3,:)))
            tricut1=delaunay(Xcut1(:,1),Xcut1(:,2));
            trisurf(tricut1,Xcut1(:,1),Xcut1(:,2),prescut1);
            end
            if ~ (all(Xcut2(1,:)==Xcut2(2,:))| all(Xcut2(2,:)==Xcut2(3,:))
                    |all( Xcut2(1,:)==Xcut2(3,:)))
            tricut2=delaunay(Xcut2(:,1),Xcut2(:,2));
            trisurf(tricut2,Xcut2(:,1),Xcut2(:,2),prescut2);
            end
        end

    end
end
hold off

end
```

analyticalVelocity

INPUT: List of points X.

OUTPUT: Analytical value of the velocity in such points u.

```
function u = analyticalVelocity(X)
x = X(:,1); y = X(:,2);


%solution 1
u = [ zeros(size(X,1),1) ,ones(size(X,1), 1) ];

%solution 2
```

```
%u=[y, x];
end
```

analyticalPotential

INPUT: List of points X.

OUTPUT: Analytical value of the potential in such points phi.

```
function phi=analyticalPotential(X)
global k nu

x=X(:,1); y=X(:,2);

%solution 1
phi = y;

%solution 3
%phi=x.*y;
end
```

Errors

INPUT: Matrix of nodal coordinates X and XP, matrix of connectivity T, vector with the velocity velo and potential solution phi.

OUTPUT:Error in each node, computed as the solution minus the analytical value.

```
function [errorVelo,errorPhi]=Errors(X,XP,T,velo,phi)

uan=analyticalVelocity(XP);
%Calcula la velocidad analitica en los nodos en el centro del elemento
uext=zeros(size(T,1),2);

for i=1:size(T,1)
 %   Te=T(i,1:3);
   % coord=(1/3)*sum(X(Te,:));
   % uext(i,:)=analyticalVelocity(coord);
end
uan=[uan;uext];
phian=analyticalPotential(XP);

errorVelo=velo-uan;
errorPhi=phi-phian;
end
```

B. Tests to the program

In this section the different tests that have been performed to the program in order to check that it computes the right solution will be explained. There have been two types of test: comparing the solution given by the program with an analytical solution, and computing the residuals of the linear systems. The first type of test checks that the program solves properly the systems of equations, while the second kind of test helps to determine if the program is computing properly the integrals, and if the matrices are well assembled.

B.1. Analytical solutions

Two analytical solutions will be used to check that the program works correctly. The first one will have all their components linear or less, so the program will be capable of computing it exactly, while the second one will be more complex, and because the program cannot compute exactly solutions of order two or higher, a convergence analysis will be done.

The first analytical solution used will be :

$$\begin{aligned} u(x,y) &= (0,1) \\ p(x,y) &= 1 \\ \varphi(x,y) &= y \end{aligned} \tag{B.1}$$

For this solution, the error in the horizontal and vertical component of the velocity in the Stokes problem, and the potential φ in the Darcy problem is shown in figures B.16, B.17 and B.18. In this case, the error is taken as

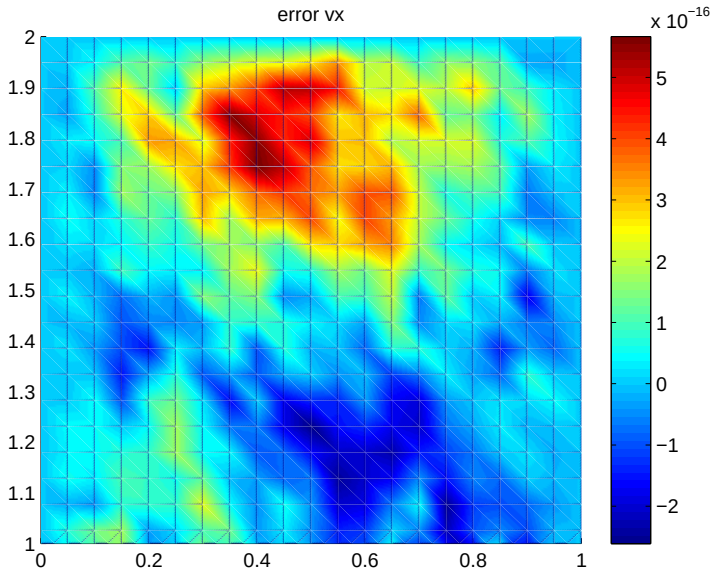

Figure B.16: Error of the horizontal component of the velocity.

the difference between the computed value and the analytical value on each node.

The figures show an error value close to zero (actually, close to 10^{-16}, that is the zero for MATLAB). This means that the program computes exactly the solution, and only errors of rounding occur.

The second analytical solution will be the following:

$$\begin{aligned} u(x,y) &= (y,x) \\ p(x,y) &= x \\ \varphi(x,y) &= xy \end{aligned} \tag{B.2}$$

Now the φ term is not linear, so the program cannot compute it exactly.

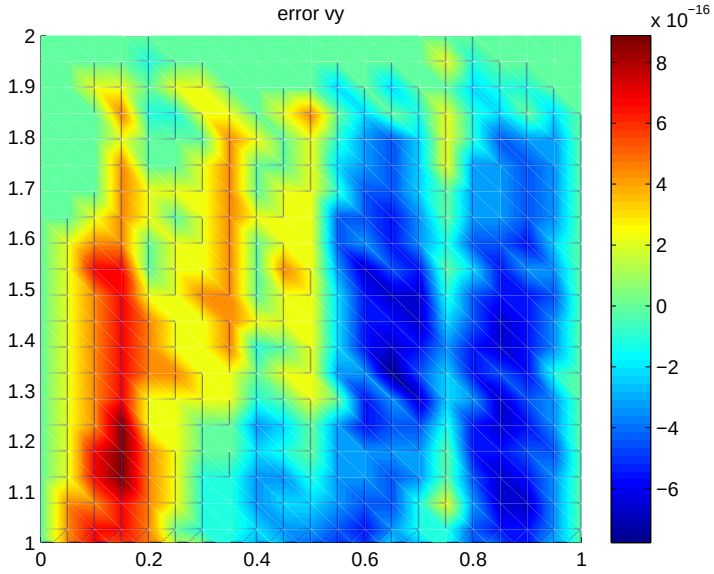

Figure B.17: Error of the vertical component of the velocity.

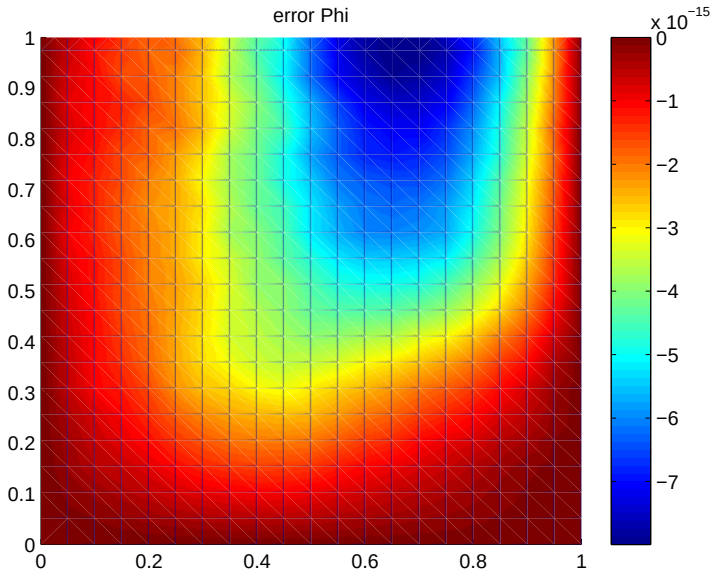

Figure B.18: Error of the φ variable.

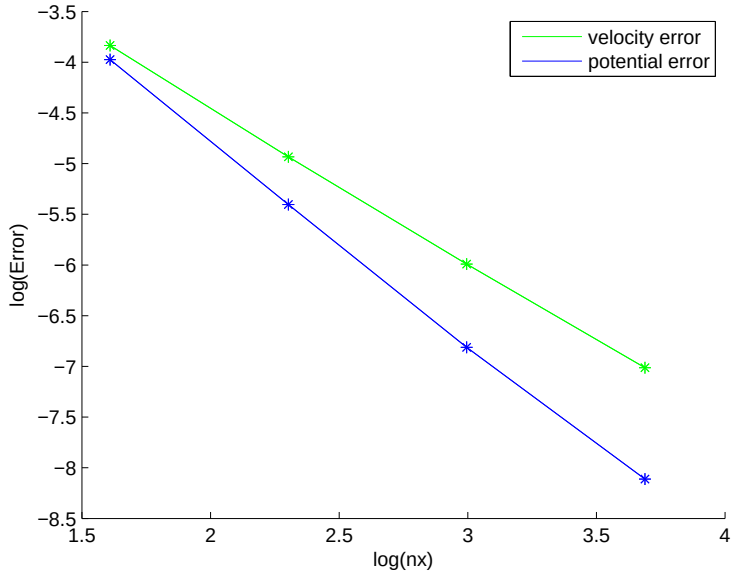

Figure B.19: Graphic of the error in L_2 norm compared with the number of elements in x direction

The graphic of convergence of the error in L_2 norm is shown in B.19, in logarithmic scale:

From the graphic we can conclude that the error decreases in each refinement of the mesh, and that the numerical solution converges to the analytical solution, so the program solves correctly the problem.

B.2. "Decoupled" system computation

Another way to ensure that the code computes the right solution is solving a decoupled system using an analytical solution, as is shown in equation (B.3).

$$\begin{aligned}\left[\boldsymbol{K}+\alpha\boldsymbol{M}_{\boldsymbol{t}}^{\boldsymbol{I}}\right]\mathbf{u}+\boldsymbol{G}\mathbf{p}&=-\boldsymbol{B}_{\boldsymbol{f}}^{\boldsymbol{I}}\varphi^{an}+\boldsymbol{f}\\ \boldsymbol{D}\varphi&=-\boldsymbol{B}_{\boldsymbol{p}}^{\boldsymbol{I}}\mathbf{u}^{an}\end{aligned}\tag{B.3}$$

This way, the Stokes problem is solved based on the analytical solution for the Darcy problem, and the Darcy problem is solved using the Stokes analytical solution. This is done because in this way, the errors of the computation of one problem don't affect the other problem, so it is easy to know if there is any wrong computation and in which of the two problems occur.

B.3. Residuals computation

In order tot check that the integrations are correctly done, and that the matrices are correctly assembled, the residual calculation will be done in this section. The residual calculation derive from equations (11) and (13), where instead of leaving $\mathbf{u}$, $\mathbf{p}$ and φ as unknowns to be solved, the analytical solution is substituted, as it is shown here:

$$\begin{aligned}\left[\boldsymbol{K}+\alpha\boldsymbol{M}_{\boldsymbol{t}}^{\boldsymbol{I}}\right]\mathbf{u}^{an}+\boldsymbol{G}\mathbf{p}^{an}-\boldsymbol{B}_{\boldsymbol{f}}^{\boldsymbol{I}}\varphi^{an}-\boldsymbol{f}&=\mathbf{r}_f\\ \boldsymbol{D}\varphi^{an}+\boldsymbol{B}_{\boldsymbol{p}}^{\boldsymbol{I}}\mathbf{u}^{an}&=\mathbf{r}_p\end{aligned}\tag{B.4}$$

Figures B.20,B.21 and B.22 correspond to the residual for the analytical solution

$$\begin{aligned}u(x,y)&=(0,1)\\ p(x,y)&=1\\ \varphi(x,y)&=y\end{aligned}\tag{B.5}$$

Because all the terms of this analytical solution are linear, and the approximation that it is being used is linear, the residual should be zero (in fact, it is of order 10^{-16}, which is the zero for MATLAB).

Figures B.23, B.24 and B.25 correspond to the analytical solution

$$\begin{aligned} u(x,y) &= (y,x) \\ p(x,y) &= x \\ \varphi(x,y) &= xy \end{aligned} \tag{B.6}$$

Now the φ term is not linear, and the linear approximation can not "capture" it properly, so the residual is not zero, although it is still very low.

These two tests help to conclude that the numerical integration is well performed, and the matrices are correctly assembled.

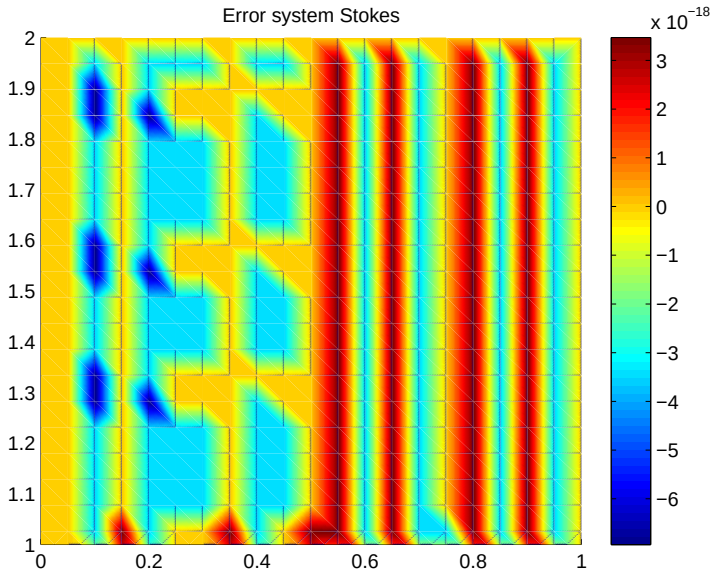

Figure B.20: Residual corresponding to the horizontal component of the velocity.

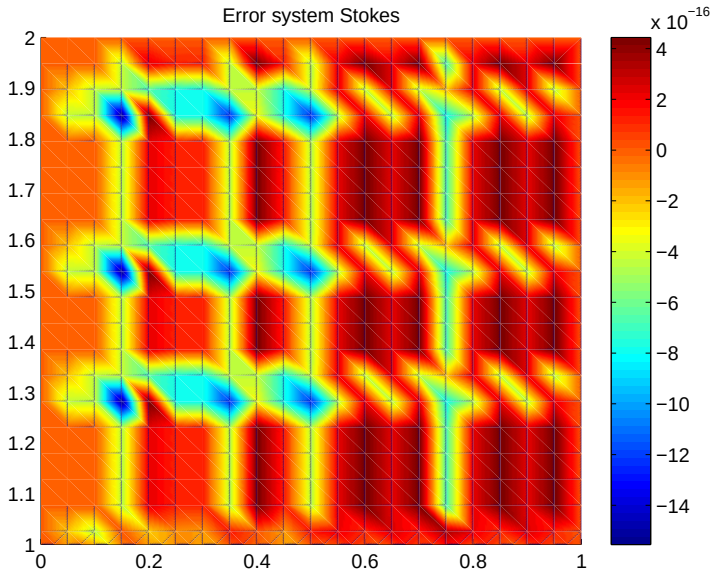

Figure B.21: Residual corresponding to the vertical component of the velocity.

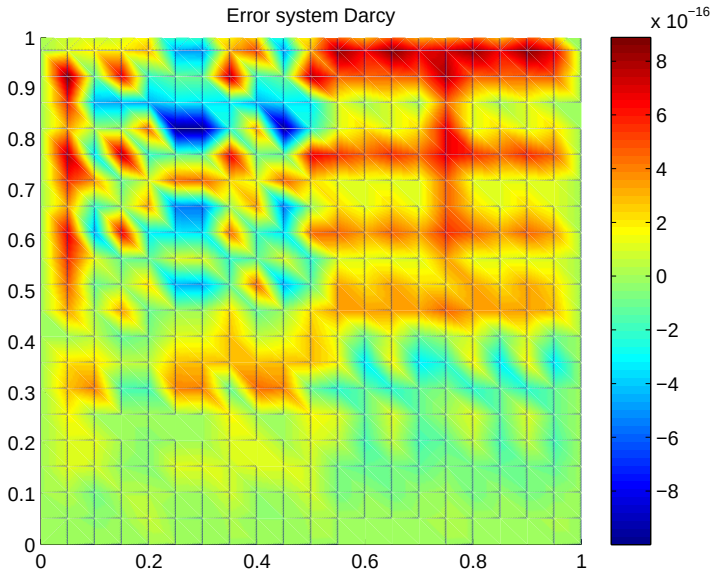

Figure B.22: Residual corresponding to the potential φ.

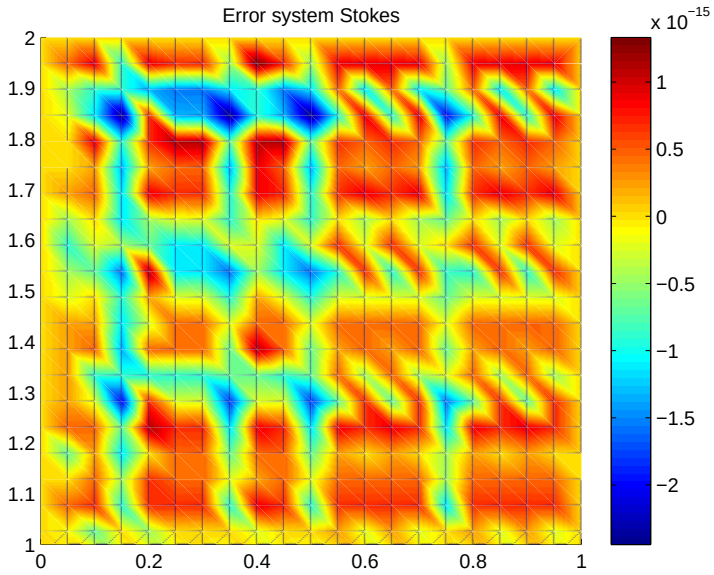

Figure B.23: Residual corresponding to the horizontal component of the velocity.

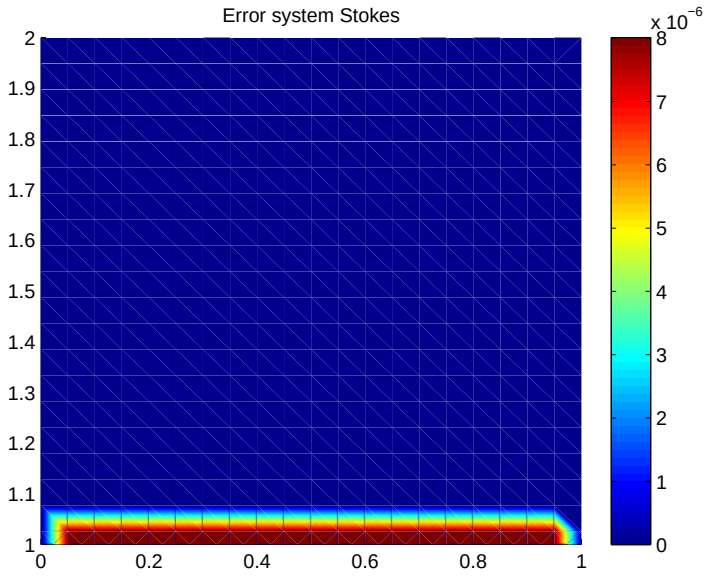

Figure B.24: Residual corresponding to the vertical component of the velocity.

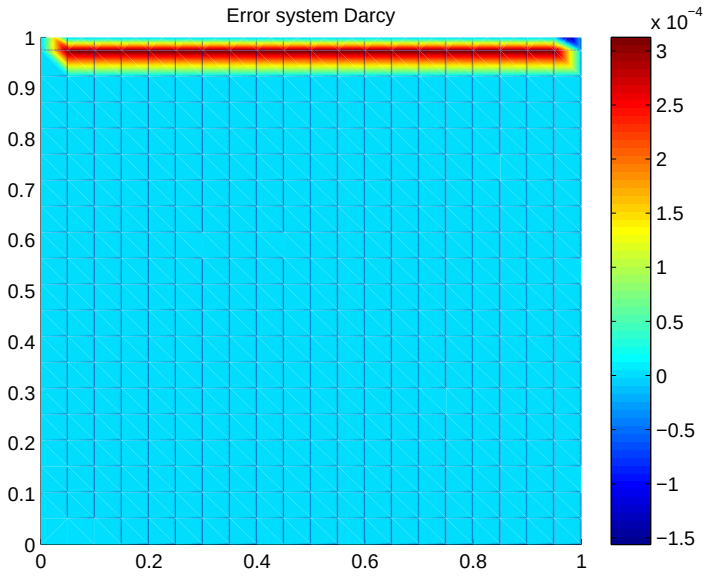

Figure B.25: Residual corresponding to the potential φ.