%% TESINA GRUPPO 9 TRANSIENTE

%% pulizia rimanenze

clc

clear all

close all

%% Discretizzazione dominio: lavoriamo con due sistemi di riferimento

%diverso tra asse x e asse y (in un caso km e in un caso metri)

S(1) = regions.rectN([0, 0], [3000/1000, 3]); % terreno low

S(2) = regions.rectN([0, 3], [3000/1000, 3.1]); %isolante low

S(3) = regions.rectN([0, 3.1], [3000/1000, 3.4]); %spessore low

S(4) = regions.rectN([0,3.4], [3000/1000, 5.4]); %tubo con raggio interno 1m

S(5) = regions.rectN([0,5.4], [3000/1000, 5.7]); %spessore up

S(6) = regions.rectN([0,5.7], [3000/1000, 5.8]); %isolante up

S(7) = regions.rectN([0, 5.8], [3000/1000, 7.8]); %terreno up

%% condizioni al contorno

%inseriamo i nodi sul bordo inferiore:

S(1).Borders(1).insertNode(4, [1800/1000 0]);

S(1).Borders(1).insertNode(5, [1200/1000 0]);

%dirichlet bordo inferiore terreno:

S(1).Borders(1).Bc([4 6])=boundaries.dirichlet(12+273);

S(1).Borders(1).Bc(5)=boundaries.dirichlet( @(x,y) ((12-34)/((1200/1000)-...

(1500/1000))^2)\*(x-1500/1000).^2+34+273 ); %dirichlet parabolico

%dirichlet lato sinistro tubo

S(4).Borders(1).Bc(1) = boundaries.dirichlet(1); %impongo 1 per poter

%andare a inserire l'andamento

%di temperatura in funzione del tempo

%Condizioni Neumann lato destro

for i = 1:length(S)

S(i).Borders(1).Bc(3) = boundaries.neumann(0);

end

%condizione di Neuman lato sinistro

for i = 1:length(S)

if i ~= 4

S(i).Borders(1).Bc(1) = boundaries.neumann(0);

end

end

% Dati del problema robin

hc = 5.5; %per coeff convettivo per condizione di robin dell'aria

t\_inf = 20+273; % T infinito di robin

%condizione di Robin

a = hc;

b = hc\*t\_inf;

S(7).Borders(1).Bc(2)=boundaries.robin(a,b);

%condizione di continuità

for i = 1:length(S)-1

S(i).Borders(1).Bc(2)=boundaries.continuity();

end

for i = 2:length(S)

S(i).Borders(1).Bc(4)=boundaries.continuity();

end

%% Aggiunta delle proprietà

lambda\_floor = 2; %valore di lambda del terreno

lambda\_fluid = 0.5; %valore di lambda del fluido

lambda\_isulator = 0.04; %valore di lambda dell'isolante

lambda\_metal = 50; %valore di lambda del tubo metallico

cp\_fluid = 3950; % cp dell'acqua

cp\_metal = 500; %cp dell'acciaio

cp\_isulator = 1300; %cp dell'isolante

cp\_floor = 900; %cp del terreno

ro\_fluid = 975; %densità acqua

ro\_metal = 7800; %densità metallo

ro\_isulator = 250; %densità isolante

ro\_floor = 1100; %densità del terreno

S(4).addProperty('lambda',lambda\_fluid); %diffusibilità termica su dominio del tubo

S(1).addProperty('lambda',lambda\_floor); %proprietà di diffusione del terreno

S(7).addProperty('lambda',lambda\_floor); %proprietà di diffusione del terreno

S(2).addProperty('lambda',lambda\_isulator); %proprietà di diffusione dell'isolante

S(6).addProperty('lambda',lambda\_isulator); %proprietà di diffusione dell'isolante

S(3).addProperty('lambda',lambda\_metal); %proprietà di diffusione del tubo

S(5).addProperty('lambda',lambda\_metal); %proprietà di diffusione del tubo

S(4).addProperty('rho',ro\_fluid\*cp\_fluid); %cp \* ro su dominio del tubo

S(1).addProperty('rho',ro\_floor\*cp\_floor); %cp \* ro del terreno

S(7).addProperty('rho',ro\_floor\*cp\_floor); %cp \* ro del terreno

S(2).addProperty('rho',ro\_isulator\*cp\_isulator); %cp \* ro dell'isolante

S(6).addProperty('rho',ro\_isulator\*cp\_isulator); %cp \* ro dell'isolante

S(3).addProperty('rho',ro\_metal\*cp\_metal); %cp \* ro del tubo

S(5).addProperty('rho',ro\_metal\*cp\_metal); %cp \* ro del tubo

%% Mesh

Me\_dimention = [0.005]; %valore della dimensione della meh

Me\_S = mesh2D(S,Me\_dimention);

%% Calcolo della velocità:

Me\_Tubo = Me\_S.extractMesh(4); % estraggo la mesh che mi serve ossia la parte 4

f\_beta = @(y) -(0.5)\*((y-4.4).^2)+(0.5); % funzione parabolica della velocità

y\_b = Me\_Tubo.Nodes.Y; % calcolo i nodi lungo l'asse y

b\_x = f\_beta(y\_b) ; %definisco la velocità lungo asse x

b\_y = zeros(length(y\_b),1); %definisco velocità lungo asse y che è nula ed è

%definita come un vettore colonna di zeri

XX=Me\_Tubo.Triangles.CenterOfMass.X; % calcolo i centri di massa di x per

%ciarcun triangolo per la cordinata x

YY=Me\_Tubo.Triangles.CenterOfMass.Y; % calcolo i centri di massa di x per

%ciarcun triangolo per la cordinata x

beta\_x=(Me\_Tubo.interpolate(b\_x,[XX,YY],1:length(XX)))\*10^-3; % calcolo la

%componente x della velocità per ciuascun centro di

%massa ( il comando 1:length(XX) mi permette di

%valutare tutti i centri di massa della zona indicata)

beta\_y=Me\_Tubo.interpolate(b\_y,[XX,YY],1:length(YY)); %calcolo la componente y

%della velocità

V=[beta\_x , beta\_y]; %definisco il vettore velocità

%% definisco la proprietà di velocità:

S.addProperty('beta',[0 0]); % definisco su tutto il dominio beta nullo

S(4).addProperty('beta',[V(:,1) V(:,2)]); %sovrascrivo velocità diversa da

%zero nel dominio del tubo tale per cui ho

%V(:,1)=tutti i valori della prima coordinata

%di V e V(:,2)= tutti i valori della seconda

%coordinata di V

%% risoluzione del sistema con function

[A,bconst\_iniziale, bvar\_iniziale] = Tesina\_transitorio\_INIZIALE\_BuildStiff(Me\_S);

%richiamo la funzione per il calcolo della condizione

%iniziale

[D,bconst, bvar] = Tesina\_transitorio\_BuildStiff(Me\_S);%richiamo la funzione per

%il calcolo di tutti gli altri istanti temporali

%% Nodi sul bordo di dirichlet annullando il profilo parabolico

DirichletNodesVar=Me\_S.find(@(x,y) y>=3.4,'d'); %definisco i nodi sul lato

%di dirichlet che varia

%% Gradi di libertà

Dof=Me\_S.Nodes.Dof>0; %definisco i gradi di libertà

%% Soluzione stazionaria

T0=12+273; %temperatura iniziale

uStationary0=A\(bconst\_iniziale+bvar\_iniziale\*T0);

uu=Me\_S.copyToAllNodes(uStationary0);

uu(DirichletNodesVar)=T0;

figure;

Me\_S.draw( uu);

ylabel(colorbar(),'Temperature [K]');

%% time evolution: definition of the function T

%edge 4 temperature

Tend=20000000;

fDirichlet=@(t) 273+12+(91)\*(2/pi)\*atan(t);

[M, mvar] = Tesina\_Massa(Me\_S);

%% implicit Euler method

disp('Implicit Euler method');

u=uStationary0;

dt=25000;

figure;

S=(M+D\*dt);

tic

TPipeOld=fDirichlet(0);

TT=zeros(size(Tend));

tt=zeros(size(Tend));

for k=1:Tend/dt,

t=k\*dt;

TPipe=fDirichlet(t);

%original: u=(M+D\*dt)\(M\*u+b\*dt\*T4);

u=S\(M\*u+dt\*(bconst+bvar\*TPipe)-mvar\*(TPipe-TPipeOld));

uu(Dof)=u;

uu(DirichletNodesVar)=TPipe;

TPipeOld=TPipe;

hold off;

TT(k) = uu(1020);

tt(k) = t;

if rem (k,100)==0

Me\_S.draw(uu,'hidemesh');

zlim([273 380]);

caxis([273 376]);

title(['Andamento termico al tempo t = ' num2str(t) 's']);

xlabel('x dir [m]');

ylabel('y dir [km]');

zlabel('[K]');

ylabel(colorbar(),'Temperature [K]');

drawnow();

end

end

toc

figure();

plot(tt, TT);

title('andamento temperatura nodo 1020');

xlabel('time [s]');

ylabel('Temperature [K]');

%% Rappresentazione dei primi 2000000 secondi

Tend=2000000;

u=uStationary0;

dt=1000;

figure;

S=(M+D\*dt);

tic

TPipeOld=fDirichlet(0);

for k=1:Tend/dt,

t=k\*dt;

TPipe=fDirichlet(t);

%original: u=(M+D\*dt)\(M\*u+b\*dt\*T4);

u=S\(M\*u+dt\*(bconst+bvar\*TPipe)-mvar\*(TPipe-TPipeOld));

uu(Dof)=u;

uu(DirichletNodesVar)=TPipe;

TPipeOld=TPipe;

hold off;

if rem (k,100)==0

Me\_S.draw(uu,'hidemesh');

zlim([273 380]);

caxis([273 376]);

title(['Dettaglio iniziale della temperatura per t = ' num2str(t) 's']);

xlabel('x dir [m]');

ylabel('y dir [km]');

zlabel('[K]');

ylabel(colorbar(),'Temperature [K]');

drawnow();

end

end

toc

%%FUNZIONE MATRICE DI MASSA

function [M, mvar] = buildMassVariableDirichlet(Me)

%Assemble the mass matrix M

%Input:

% Me :a Mesh2D object

%

%Output:

% M :mass matrix

% mvar :vector to include the effect of the non Homogeneous Dirichlet B.C.

%for clarity, call some properties of Me with shorter names

V = Me.Triangles.Vertices;

Dof = Me.Nodes.Dof;

Areas=Me.Triangles.Areas\*10^3;

%number of internal nodes: we know that the N unknown nodes are numbered from

%1 to N in Me.UnknownNodes; the maximum is therefore the number of unknown

%(degrees of freedom)

numDof = max(Dof);

%vectors preallocation: instead of allocating the (sparse) diffusion matrix,

%we save the rows, columns and values corresponding to each contribution;

%at the end, we'll call sparse(...) to obtain the diffusion matrix

row = zeros(Me.MatrixContributions, 1);

col = zeros(Me.MatrixContributions, 1);

m = zeros(Me.MatrixContributions, 1);

mvar = zeros(numDof, 1);

pos = 1; %we start from the element in position 1, we'll increase this index

%everytime we add an entry

%evaluate the value of the coefficient in front of the time derivative

rho = Me.evaluateProperty('rho');

%main loop on each triangle

for e = 1:size(V, 1)

%for each vertex of this triangle

for ni = 1:3

%look at the "unknown" numbering: if the node is positive, it

%corresponds to a degree of freedom of the problem

ii = Dof(V(e, ni));

%is it unknown?

if ii > 0

%yes it is! second loop on the vertices

for nj = 1:3

jj = Dof(V(e, nj));

%%is it unknown as well?

mtmp = 1/12 \* Areas(e) \* rho(e) \* ((ii == jj) + 1);

if jj > 0

%add the contribution to the mass matrix

row(pos) = ii;

col(pos) = jj;

m(pos) = mtmp;

pos = pos + 1;

else

if abs(Me.Nodes.Y(V(e,nj)))>=3.4 %on the inner circle?

mvar(ii)=mvar(ii)+mtmp;

end

end

end

end

end

end

%assemble the mass matrix M

M = sparse(row, col, m, numDof, numDof);

%%FUNZIONE TRANSIENTE ISTANTE INIZIALE

function [D, bconst, bvar] = heatEquationVariableDirichlet\_BuildStiff(Me)

%Assemble the matrix D and the vectors b of the Diffusion problem with

%time-varying BC.s

%Input:

% Me :a Mesh2D object

%

%Output:

% D :diffusion matrix

% b :constant terms vector

%for clarity, call some properties of Me with shorter names

V=Me.Triangles.Vertices;

Areas=Me.Triangles.Areas\*10^3;

Nodes=Me.Nodes;

Dof=Me.Nodes.Dof;

Edges=Me.Edges; %condizione necessario per Robin

Robin=Me.BC.RobinEdges;%condizione necessaria per Robin

%number of internal nodes: we know that the N unknown nodes are numbered from

%1 to N in Me.UnknownNodes; the maximum is therefore the number of unknown

%(degrees of freedom)

numDof = max(Dof);

%vectors preallocation: instead of allocating the (sparse) diffusion matrix,

%we save the rows, columns and values corresponding to each contribution;

%at the end, we'll call sparse(...) to obtain the diffusion matrix

bconst = zeros(numDof,1);

bvar = zeros(numDof,1);

row = zeros(Me.MatrixContributions,1);

col = zeros(Me.MatrixContributions,1);

d = zeros(Me.MatrixContributions,1);

pos=1; %we start from the element in position 1, we'll increase this index

%everytime we add an entry

%evaluate the value of the coefficient in front of the Laplace operator

c = Me.evaluateProperty('lambda');

ro\_fluid = 975; %densità acqua

cp\_fluid = 3950; % cp dell'acqua

beta=((Me.evaluateProperty('beta'))\*ro\_fluid\*cp\_fluid\*0); %indica il termine

%di trasporto che all'istante iniziale viene considerato pari a zero

%main loop on each triangle

for e=1:size(V,1)

Dx(1) = (Nodes.X(V(e,3)) - Nodes.X(V(e,2)))\*10^3; %andiamo a scalare la

%distanza tra i nodi in modo da considerare l'approssimazione

%di 3000m a 3m;

Dx(2) = (Nodes.X(V(e,1)) - Nodes.X(V(e,3)))\*10^3;

Dx(3) = (Nodes.X(V(e,2)) - Nodes.X(V(e,1)))\*10^3;

Dy(1) = Nodes.Y(V(e,3)) - Nodes.Y(V(e,2));

Dy(2) = Nodes.Y(V(e,1)) - Nodes.Y(V(e,3));

Dy(3) = Nodes.Y(V(e,2)) - Nodes.Y(V(e,1));

%for each vertex of this triangle

for ni=1:3

%look at the "unknown" numbering: if the node is positive, it

%corresponds to a degree of freedom of the problem

ii = Dof(V(e,ni));

%is it unknown?

if ii > 0

%yes it is! second loop on the vertices

for nj=1:3

jj = Dof(V(e,nj));

dtmp=c(e)\*(Dy(ni)\*Dy(nj)+Dx(ni)\*Dx(nj))/(4.0\*Areas(e)) + ...

(-beta(e,1)\*Dy(nj)+beta(e,2)\*Dx(nj))\*1/6; %ho aggiunto il

%termine relativo a 'beta'

%%is it unknown as well?

if jj > 0

%add the contribution to the stiffness matrix

row(pos)=ii;

col(pos)=jj;

d(pos)=dtmp;

pos=pos+1;

%Non sparse solution: D(ii,jj)=D(ii,jj) + c\*(Dy(i)\*Dy(j)

%+Dx(i)\*Dx(j))/(4.0\*Area) ;

else %%jj is a node on a Dirichlet edge

val=Me.BC.DirichletNodes(-jj,2);

if abs(Me.Nodes.Y(V(e,nj)))>=3.4 %on the inner circle?

bvar(ii) = bvar(ii) - dtmp\*val ;

else

bconst(ii) = bconst(ii) - dtmp\*val ;

end

end

end

%in this example, no external force contribution

end

end

end

% Introduco condizione di Robin necessaria per il bordo superiore

for k=1:size(Robin,1)

Node1=Edges(Robin(k,1),1);

Node2=Edges(Robin(k,1),2);

dx=(Nodes.X(Node1)-Nodes.X(Node2))\*10^3;

dy=Nodes.Y(Node1)-Nodes.Y(Node2);

dist=sqrt(dx\*dx+dy\*dy);

ii1=Dof(Node1);

ii2=Dof(Node2);

g=Robin(k,3);

h=Robin(k,2);

if ii1>0 && ii2<0 %ii1 is unknown, ii2 is known

bconst(ii1)=bconst(ii1)+g/2\*dist;

row(pos)=ii1;

col(pos)=ii1;

d(pos)=h\*dist/3;

pos=pos+1;

%D(ii1,ii1)=D(ii1,ii1)+h\*dist/3;

elseif ii1<0 && ii2>0 %ii1 is known, ii2 is unknown

bconst(ii2)=bconst(ii2)+g/2\*dist;

row(pos)=ii2;

col(pos)=ii2;

d(pos)=h\*dist/3;

pos=pos+1;

%D(ii2,ii2)=D(ii2,ii2)+h\*dist/3;

else %both are unknwon

bconst(ii1)=bconst(ii1)+g/2\*dist; %integrale di bordo di Robin

bconst(ii2)=bconst(ii2)+g/2\*dist;

row(pos:pos+3)=[ii1;ii2;ii1;ii2];

col(pos:pos+3)=[ii1;ii2;ii2;ii1];

d(pos:pos+3)=[2;2;1;1]\*h\*dist/6;

pos=pos+4;

%D(ii1,ii1)=D(ii1,ii1)+h\*dist/3;

%D(ii2,ii2)=D(ii2,ii2)+h\*dist/3;

%D(ii1,ii2)=D(ii1,ii2)+h\*dist/6;

%D(ii2,ii1)=D(ii2,ii1)+h\*dist/6;

end

end

%assemble the stiffness matrix D from the

D=sparse(row,col, d, numDof, numDof);

%%FUNZIONE TRANSIENTE

function [D, bconst, bvar] = heatEquationVariableDirichlet\_BuildStiff(Me)

%Assemble the matrix D and the vectors b of the Diffusion problem with

%time-varying BC.s

%Input:

% Me :a Mesh2D object

%

%Output:

% D :diffusion matrix

% b :constant terms vector

%for clarity, call some properties of Me with shorter names

V=Me.Triangles.Vertices;

Areas=Me.Triangles.Areas\*10^3;

Nodes=Me.Nodes;

Dof=Me.Nodes.Dof;

Edges=Me.Edges; %condizione necessario per Robin

Robin=Me.BC.RobinEdges;%condizione necessaria per Robin

%number of internal nodes: we know that the N unknown nodes are numbered from

%1 to N in Me.UnknownNodes; the maximum is therefore the number of unknown

%(degrees of freedom)

numDof = max(Dof);

%vectors preallocation: instead of allocating the (sparse) diffusion matrix,

%we save the rows, columns and values corresponding to each contribution;

%at the end, we'll call sparse(...) to obtain the diffusion matrix

bconst = zeros(numDof,1);

bvar = zeros(numDof,1);

row = zeros(Me.MatrixContributions,1);

col = zeros(Me.MatrixContributions,1);

d = zeros(Me.MatrixContributions,1);

pos=1; %we start from the element in position 1, we'll increase this index

%everytime we add an entry

%evaluate the value of the coefficient in front of the Laplace operator

c = Me.evaluateProperty('lambda');

ro\_fluid = 975; %densità acqua

cp\_fluid = 3950; % cp dell'acqua

beta=((Me.evaluateProperty('beta'))\*ro\_fluid\*cp\_fluid); %indica il termine

%di trasporto che è aggiunto ma avendo velocità solo nella zona del

%tubo posso tenere conto direttamente della densità del fluido del Cp

%main loop on each triangle

for e=1:size(V,1)

Dx(1) = (Nodes.X(V(e,3)) - Nodes.X(V(e,2)))\*10^3; %andiamo a scalare la

%distanza tra i nodi in modo da considerare l'approssimazione

%di 3000m a 3m;

Dx(2) = (Nodes.X(V(e,1)) - Nodes.X(V(e,3)))\*10^3;

Dx(3) = (Nodes.X(V(e,2)) - Nodes.X(V(e,1)))\*10^3;

Dy(1) = Nodes.Y(V(e,3)) - Nodes.Y(V(e,2));

Dy(2) = Nodes.Y(V(e,1)) - Nodes.Y(V(e,3));

Dy(3) = Nodes.Y(V(e,2)) - Nodes.Y(V(e,1));

%for each vertex of this triangle

for ni=1:3

%look at the "unknown" numbering: if the node is positive, it

%corresponds to a degree of freedom of the problem

ii = Dof(V(e,ni));

%is it unknown?

if ii > 0

%yes it is! second loop on the vertices

for nj=1:3

jj = Dof(V(e,nj));

dtmp=c(e)\*(Dy(ni)\*Dy(nj)+Dx(ni)\*Dx(nj))/(4.0\*Areas(e)) + ...

(-beta(e,1)\*Dy(nj)+beta(e,2)\*Dx(nj))\*1/6; %ho aggiunto

%il termine relativo a 'beta'

%%is it unknown as well?

if jj > 0

%add the contribution to the stiffness matrix

row(pos)=ii;

col(pos)=jj;

d(pos)=dtmp;

pos=pos+1;

%Non sparse solution: D(ii,jj)=D(ii,jj) + c\*(Dy(i)\*Dy(j)

%+Dx(i)\*Dx(j))/(4.0\*Area) ;

else %%jj is a node on a Dirichlet edge

val=Me.BC.DirichletNodes(-jj,2);

if abs(Me.Nodes.Y(V(e,nj)))>=3.4 %on the inner circle?

bvar(ii) = bvar(ii) - dtmp\*val ;

else

bconst(ii) = bconst(ii) - dtmp\*val ;

end

end

end

%in this example, no external force contribution

end

end

end

% Introduco condizione di Robin necessaria per il bordo superiore

for k=1:size(Robin,1)

Node1=Edges(Robin(k,1),1);

Node2=Edges(Robin(k,1),2);

dx=(Nodes.X(Node1)-Nodes.X(Node2))\*10^3;

dy=Nodes.Y(Node1)-Nodes.Y(Node2);

dist=sqrt(dx\*dx+dy\*dy);

ii1=Dof(Node1);

ii2=Dof(Node2);

g=Robin(k,3);

h=Robin(k,2);

if ii1>0 && ii2<0 %ii1 is unknown, ii2 is known

bconst(ii1)=bconst(ii1)+g/2\*dist;

row(pos)=ii1;

col(pos)=ii1;

d(pos)=h\*dist/3;

pos=pos+1;

%D(ii1,ii1)=D(ii1,ii1)+h\*dist/3;

elseif ii1<0 && ii2>0 %ii1 is known, ii2 is unknown

bconst(ii2)=bconst(ii2)+g/2\*dist;

row(pos)=ii2;

col(pos)=ii2;

d(pos)=h\*dist/3;

pos=pos+1;

%D(ii2,ii2)=D(ii2,ii2)+h\*dist/3;

else %both are unknwon

bconst(ii1)=bconst(ii1)+g/2\*dist; %integrale di bordo di Robin

bconst(ii2)=bconst(ii2)+g/2\*dist;

row(pos:pos+3)=[ii1;ii2;ii1;ii2];

col(pos:pos+3)=[ii1;ii2;ii2;ii1];

d(pos:pos+3)=[2;2;1;1]\*h\*dist/6;

pos=pos+4;

%D(ii1,ii1)=D(ii1,ii1)+h\*dist/3;

%D(ii2,ii2)=D(ii2,ii2)+h\*dist/3;

%D(ii1,ii2)=D(ii1,ii2)+h\*dist/6;

%D(ii2,ii1)=D(ii2,ii1)+h\*dist/6;

end

end

%assemble the stiffness matrix D from the

D=sparse(row,col, d, numDof, numDof);