%% TESINA STAZIONARIO GRUPPO 9

%% pulizia del foglio di calcolo

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

%I comandi S.draw permettono di visualizzare il dominio con i nodi e lati

%numerati.

%figure(); S.draw('e')

%figure(); S.draw('n')

%% 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(103+273);

%Condizioni Neumann lato destro

for i = 1:length(S)

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

end

%condizione di Neumann 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; % 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

figure();S.draw('bc') %Visualiziamo il dominio con le relative condizioni al bordo

title('Dominio e Bc');

ylabel('m');

xlabel('km');

%% 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

ro\_fluid = 975; %densità acqua

cp\_fluid = 3950; % cp dell'acqua

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

%% Mesh

Me\_dimention = [0.1 0.01 0.005]; % Diverse dimensioni dell'area massima della mesh

zz = zeros(100,length(Me\_dimention)); %Inizializzazione vettore per calcolo

%convergenza mesh

for i=1:length(Me\_dimention) %Ciclo per ciascuna dimensione dell'area massima

%della mesh

Me\_S = mesh2D(S,Me\_dimention(i)); %Rappresentiamo la mesh

figure(); Me\_S.draw();

title(['Mesh area max : ',num2str(Me\_dimention(i)),]);

ylabel('m');

xlabel('km');

%% Calcolo della velocità: oltre a valutarne il valore si è deciso di

%rappresentare il suo andamento

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 assegnata da testo

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

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

b\_y = zeros(length(y\_b),1); %si definisce la velocità lungo asse y che è

%nulla ed è definita come un vettore colonna di zeri

figure();

axis equal;

quiver(Me\_Tubo.Nodes.X,Me\_Tubo.Nodes.Y,b\_x,b\_y); % si disegna il profilo delle velocità

title(['Profilo di velocità con mesh di area max : ',num2str(Me\_dimention(i)),]);

ylabel('m');

xlabel('km');

%Definizione del vettore velocità nelle sue due componenti

XX=Me\_Tubo.Triangles.CenterOfMass.X;

YY=Me\_Tubo.Triangles.CenterOfMass.Y;

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

%la componente x della velocità per ciuascun centro di massan

%( 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)); %si calcola la componente

%y della velocità

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

%Assegnazione della propriètà di velocità nel condotto

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

S(4).addProperty('beta',[V(:,1) V(:,2)]); %si sovrascive la velocità nel tubo

%% risoluzione del sistema con function

[A,b] = Tesina\_stazionario\_BuildStiff(Me\_S, @(x,y) zeros(size(x))); %si impone forzante nulla

% Risoluzione del sistema

t = A\b;

T = Me\_S.copyToAllNodes(t); %si restituisce il vettore t esteso a tutti i

%nodi e non indicizzato rispetto ai soli gradi

%di libertà

%% Rappresentazione grafica

figure();

axis equal;

draw(Me\_S,T,'h');

ylabel('m');

xlabel('km');

zlabel('K');

title(['Mesh area max :',num2str(Me\_dimention(i))]);

%% Elaborazione della soluzione per calcolo convergenza

yy = 4.4\*ones(100,1);

xx = linspace(0,3,100).';

zz(:,i) = Me\_S.interpolate(T,[xx,yy]);

end

%%Si plotta la convergenza

figure();

axis equal;

plot(xx,zz(:,1:end));

ylabel('m');

xlabel('km');

legend([num2str(Me\_dimention(1,1))],[num2str(Me\_dimention(1,2))],[num2str(Me\_dimention(1,3))]);

title(['Temperatura: convergenza della mesh']);

%% Si valuta il numero di Peclet (non espressamente richiesto dai dati)

Areas=(Me\_S.Triangles.Areas)\*(10^3); %Si inserisce il vettore che continene

%il numero delle aree di ciascun triangolo

Num\_peclet = zeros(length(Areas),1); %si inizializza Peclet a zeri. Questo

%vettore deve essere grande nello stesso

%numero dei triangoli che compongono la mesh

lato = zeros(length(Areas),1); %contiene la lunghezza del lato di ciascun

%triangolo che compone la mesh e lo si

%inizializza a zero

Beta = Me\_S.evaluateProperty('beta'); %contiene la proprietà relativa alla velocità

Lambda = Me\_S.evaluateProperty('lambda'); %contiene l'informazione legata alla conducibilità

for k = 1:length(Areas) %ciclo triangolo per triangolo

lato(k) = ((4.\*Areas(k))/(sqrt(3))).^0.5; %tramite la formula inversa

%dell'area mi calcolo la lunghezza del

%lato con l'ipotesi che i triangoli siano

%tutti equilateri

Num\_peclet(k) = (Beta(k,1)\*ro\_fluid\*cp\_fluid\*lato(k))/(6\*Lambda(k)); %calcolo

%peclet per ciascun triangolo

end

Peclet\_max = max(Num\_peclet) %Peclet rappresenta il numero massimo dei peclet

%che ho nella triangolazione

%% Preconditioning study

%figure; spy(A);

[L,U] = lu(A);

%figure; spy(L);

%figure; spy(U);

% Inizializzazione dei parametri per il ciclo

opts.type = 'ilutp'; % Threshold dropping e pivoting

Nrep = 10; % Numero di ripetizioni su cui mediare il tempo

droptols = 10.^(-2:-0.5:-8); % Vettore delle tolleranze considerate

% Allocazione dei vettori

time\_lu = zeros(size(droptols));

time\_bicg = zeros(size(droptols));

flag = zeros(size(droptols));

res = zeros(size(droptols));

iter = zeros(size(droptols));

for k=1:length(droptols)

opts.droptol = droptols(k);

time\_lu(k) = 0;

time\_bicg(k) = 0;

for i=1:Nrep

% Fattorizzazione LU incompleta

tic();

[L,U] = ilu(A,opts);

time\_lu(k) = time\_lu(k) + toc();

% Gradiente BiConiugato precondizionato

tic();

[x,flag(k),res(k),iter(k)] = bicg(A,b,1e-6,100,L,U);

time\_bicg(k) = time\_bicg(k) + toc();

end

time\_lu(k) = time\_lu(k)/Nrep;

time\_bicg(k) = time\_bicg(k)/Nrep;

end

% Rappresentazione grafica dei tempi parziali e del tempo totale

figure();

semilogx(droptols,time\_lu, droptols,time\_bicg, droptols,time\_lu+time\_bicg);

legend('ilu','bicg','totale');

ylabel('time [s]');

xlabel('droptolerance');

title(['Preconditioning study']);

%%FUNZIONE

function [D,b] = Tesina\_stazionario\_BuildStiff(Me,f)

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

%non-homogeneous Dirichlet B.C.s

%Input:

% Me :a Mesh2D object

% f :MATLAB function of (x,y) which returns the values of the

% external source. Default: constant value=4

%

%Output:

% D :diffusion matrix

% b :constant terms vector

%check inputs

if nargin<2

f=@(x,y)4\*ones(size(x));

end

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

V=Me.Triangles.Vertices;

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

CenterOfMass=Me.Triangles.CenterOfMass;

Nodes=Me.Nodes;

Dof=Me.Nodes.Dof;

Edges=Me.Edges; %condizione necessario per Neumann e 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

b = 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

%we evaluate the external force in the center of mass of this triangle

force = f(CenterOfMass.X,CenterOfMass.Y);

%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;

%%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

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

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

end

end

%build the constant terms vector adding the external

%contribution

b(ii) = b(ii) + Areas(e)\*force(e)/3.0;

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 calcolo la matrice

%del termine noto e dei coeff solo per il primo nodo

b(ii1)=b(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

b(ii2)=b(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

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

b(ii2)=b(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);