4 April 2017 - Heat Plate Simulation using 900 Watt total boundary condition heat flux input

Contents

4 April 2017 - Heat Plate Simulation using 900 Watt internal heat source

17feb2017 - Robert E. Nee

This matlab script is an adaptation of the MathWorks example "Nonlinear Heat Transfer In a Thin Plate". The goal of the adaptation is to model a 2ft square 1/4 inch aluminum plate with various heat resistors as external heat sources. The results of the simulations will be transient and steady state temperature solutions. The plate is for a 3D Printer I'm building and which can be found on the Openbuilds.org website. The target temperature is a nominal 400 degress Kelvin or 100 degrees Celsius which is the temperature the plate should be for ABS.

```
% A number of simulations will be run and documented increasing in complexity.
%
% heat resistor specs:
% % Arcol HS100 47R J   47 ohms, 100 watts, 47.5mm by 65.2mm footprint
% to arrive at a g value for
% applyBoundaryCondition(model,'neumann','Edge',14,'g',hin,'q',[0]);
% resistor hole (appr) = perimeter side = 0.05635
% perimeter side * plate thickness .00635 = heatflux face = 0.000357

%
% Simulations:
%
% |No | heat_resistors_enabled | total resistor_power (W) | convection | radiation |
```

```
%  1          0                       0                       0          0
%  2          1                       100                     0          0
%  3          1                       100                     1          0
%  4          1                       100                     1          1
%  5          1-2                     200                     1          1
%  6          1-3                     300                     1          1
%  7          1-5                     500                     1          1
%  8          1-7                     700                     1          1
%  9          1-10                    1000                    1          1
%
%
```

Nonlinear Heat Transfer In a Thin Plate

This example shows how to perform a heat transfer analysis of a thin plate using the Partial Differential Equation Toolbox™.

The plate is square and the temperature is fixed along the bottom edge. No heat is transferred from the other three edges (i.e. they are insulated). Heat is transferred from both the top and bottom faces of the plate by convection and radiation. Because radiation is included, the problem is nonlinear. One of the purposes of this example is to show how to handle nonlinearities in PDE problems.

Both a steady state and a transient analysis are performed. In a steady state analysis we are interested in the final temperature at different points in the plate after it has reached an equilibrium state. In a transient analysis we are interested in the temperature in the plate as a function of time. One question that can be answered by this transient analysis is how long does it take for the plate to reach an equilibrium temperature.

Heat Transfer Equations for the Plate

The plate has planar dimensions one meter by one meter and is 1 cm thick. Because the plate is relatively thin compared with the planar dimensions, the temperature can be assumed constant in the thickness direction; the resulting problem is 2D.

Convection and radiation heat transfer are assumed to take place between the two faces of the plate and a specified ambient temperature.

The amount of heat transferred from each plate face per unit area due to convection is defined as

$$Q_c = h_c(T - T_a)$$

where T_a is the ambient temperature, T is the temperature at a particular x and y location on the plate surface, and h_c is a specified convection coefficient.

The amount of heat transferred from each plate face per unit area due to radiation is defined as

$$Q_r = \epsilon\sigma(T^4 - T_a^4)$$

where ϵ is the emissivity of the face and σ is the Stefan-Boltzmann constant. Because the heat transferred due to radiation is proportional to the fourth power of the surface temperature, the problem is nonlinear.

The PDE describing the temperature in this thin plate is

$$\rho C_p t_z \frac{\partial T}{\partial t} - k t_z \nabla^2 T + 2Q_c + 2Q_r = 0$$

where ρ is the material density, C_p is the specific heat, t_z is the plate thickness, and the factors of two account for the heat transfer from both plate faces.

It is convenient to rewrite this equation in the form expected by PDE Toolbox

$$\rho C_p t_z \frac{\partial T}{\partial t} - k t_z \nabla^2 T + 2h_c T + 2\epsilon\sigma T^4 = 2h_c T_a + 2\epsilon\sigma T_a^4$$

```
%resistor_area = 0.00335 % m^2
pause('on');


enableConvection = 1;
enableRadiation  = 1;
enableHeatIn     = 0;

number_of_resistors = 9;

heatIn = number_of_resistors * (100); % (W)

%heatIn = 500;
```

Problem Parameters

The plate is composed of aluminum which has the following properties

```
k = 204; % thermal conductivity of Aluminum, W/(m-K)
rho = 2720; % density of aluminum, kg/m^3
specificHeat = 910; % specific heat of Aluminum, J/(kg-K)
          %thick = .01; % plate thickness in meters


thick = .00635; % 0.25 inch (6.35 mm =0.00636m) plate thickness in meters
%thick = 0.1*thick; % 0.25 inch (6.35 mm =0.00636m) plate thickness in meters

stefanBoltz = 5.670373e-8; % Stefan-Boltzmann constant, W/(m^2-K^4)

hCoeff = 1; % Convection coefficient, W/(m^2-K)

% The ambient temperature is assumed to be 300 degrees-Kelvin.
ta = 300;

% for Al, emmiss = 0.08 for extruded,unfinished plate; but 0.85 for
% anodized. Non-anodized Al conducts (via ohmmeter).

emiss = 0.08; % emissivity of the non-anodized Al conducting plate surface
%emiss = 0.85; anodized Al
```

Create the PDE Model with a single dependent variable

```
numberOfPDE = 1;
model = createpde(numberOfPDE);
```

Geometry

Define the square by giving the 4 x-locations followed by the 4 y-locations of the corners.

```
%heatIN = 1000  R2-R1
% Geometry description:



%geometry is wrong plate is 608 mm sq ==> .3.. ok but resistor are too big


sq = 1*0.025   % small square % 388 deg
% sq = 2*0.025   % small square %  388 deg
%heatflux face area = 0.000357 implies for 25W/side 0.0089W
%
% sqm = (w+l)/2; 47.5mm by 65.2mm
sqm = (0.047 + 0.0652)/2;
sqm = sqm/2;
%sq = 0.75*0.025   % small square %  388 deg
%sq = 0.50*0.025   % small square %  388 deg
%sq = 0.25*0.025   % small square % 388 deg




            % sq = 3*0.025   % small square %  388 deg
            % sq = 8*0.025   % small square %  388 deg
            % sq = 10*0.025  % small square % 388  deg
%sqm = 0.003087;

sq =

    0.0250
```

collapse sqm to effectively not use 9 cutouts as only using external plate

boundary for Neumann boundary heat flux input

```
sqm = .005


sq = sqm;
% heatIN = 1000  R2+R1

% sq = 1*0.025   % small square % 388 deg
%  sq = 2*0.025   % small square %  388 deg
%  sq = 4*0.025   % small square %  388 deg
%  sq = 8*0.025   % small square %  388 deg
%  sq = 10*0.025  % small square % 388  deg
% %
%
% Note above results imply heatIn of 1000 W gives 388 degree plate temp
% independent of R2+R1  or R2-R1 or size of inner square

%sq = 10*0.0295   % small square
%sq = 0.1
lq = 0.3048;
% lq = 0.3048/2;
% lq = 0.3048/4;
% lq = 0.3048/6;
```

```
% Create a square geometry centered at |x = 0| and |y = 0| with sides of
% length 0.3048 meters. Include a circle of radius 0.4 concentric with the square.
OF3 = 0.125; %offset
OF3 = 0.150; %offset
OF3 = 0.175; %offset
OF3 = 0.2;   %offset

R1 = [3;4;  -lq; -lq; lq; lq;  -lq; lq; lq; -lq];
% C1 = [1; 0; 0; 0.3048*0.4];
%C1 = [C1;zeros(length(R1) - length(C1),1)];
%R2 = [3;4;  -0.1; -0.1; 0.1; 0.1;   -0.1; 0.1; 0.1; -0.1];

%  middle,center


R2 = [3;4;  -sqm; -sqm; sqm; sqm;   -sqm; sqm; sqm; -sqm ];
R2 = [R2;zeros(length(R1) - length(R2),1)];

% middle right +OF
R3 = [3;4;  -sqm + OF3; -sqm + OF3; sqm + OF3 ; sqm + OF3;   -sqm; sqm; sqm; -sqm];
R3 = [R3;zeros(length(R2) - length(R3),1)];

%middle left -OF
R4 = [3;4;  -sqm - OF3; -sqm - OF3; sqm - OF3 ; sqm - OF3;   -sqm; sqm; sqm; -sqm];
R4 = [R4;zeros(length(R3) - length(R4),1)];

% above middle,center +OF
R5 = [3;4;  -sqm; -sqm; sqm; sqm;   -sqm+OF3; sqm+OF3; sqm+OF3; -sqm+OF3];
R5 = [R5;zeros(length(R1) - length(R5),1)];
% above middle,right +OF
R6 = [3;4;  -sqm + OF3; -sqm + OF3; sqm + OF3 ; sqm + OF3;   -sqm+OF3; sqm+OF3;
sqm+OF3; -sqm+OF3];
R6 = [R6;zeros(length(R1) - length(R6),1)];
% above middle,left -OF
R7 = [3;4;  -sqm - OF3; -sqm - OF3; sqm - OF3 ; sqm - OF3;   -sqm+OF3; sqm+OF3;
sqm+OF3; -sqm+OF3];
R7 = [R7;zeros(length(R1) - length(R7),1)];



% below middle,center

R8 = [3;4;  -sqm; -sqm; sqm; sqm;   -sqm-OF3; sqm-OF3; sqm-OF3; -sqm-OF3];
R8 = [R8;zeros(length(R1) - length(R8),1)];
% below middle,right +OF
R9 = [3;4;  -sqm + OF3; -sqm + OF3; sqm + OF3 ; sqm + OF3;   -sqm-OF3; sqm-OF3; sqm-
OF3; -sqm-OF3];
R9 = [R9;zeros(length(R1) - length(R9),1)];
% below middle,left -OF
R10 = [3;4;  -sqm - OF3; -sqm - OF3; sqm - OF3 ; sqm - OF3;   -sqm-OF3; sqm-OF3; sqm-
OF3; -sqm-OF3];
R10 = [R10;zeros(length(R1) - length(R10),1)];




%gd = [R1,R2];
%sf = 'R1-R2-R3-R4-R5-R6-R7-R8-R9-R10';
%gd = [R1,R3,R2];
sf = 'R1-R2-R3-R4-R5-R6-R7-R8-R9-R10';
%sf = 'R1-R2-R3-R4-R5';
% if enableHeatIn  == 1
%    sf = 'R1+R2+R3+R4+R5+R6+R7+R8+R9+R10';
```

```
% end;
%

gd = [R1,R2,R3,R4,R5,R6,R7,R8,R9,R10];
%sf = 'R1+R2+R3+R4-R5-R6-R7-R8-R9-R10';
%sf = 'R1+R2+R3+R4+R5+R6+R7+R8+R9+R10';
%sf = 'R1-R2';
%sf = 'R1+R2';


%ns = char('R1','R2')';
ns = char('R1','R2','R3','R4','R5','R6','R7','R8','R9','R10')';
g = decsg(gd,sf,ns);

sqm =

    0.0050
```

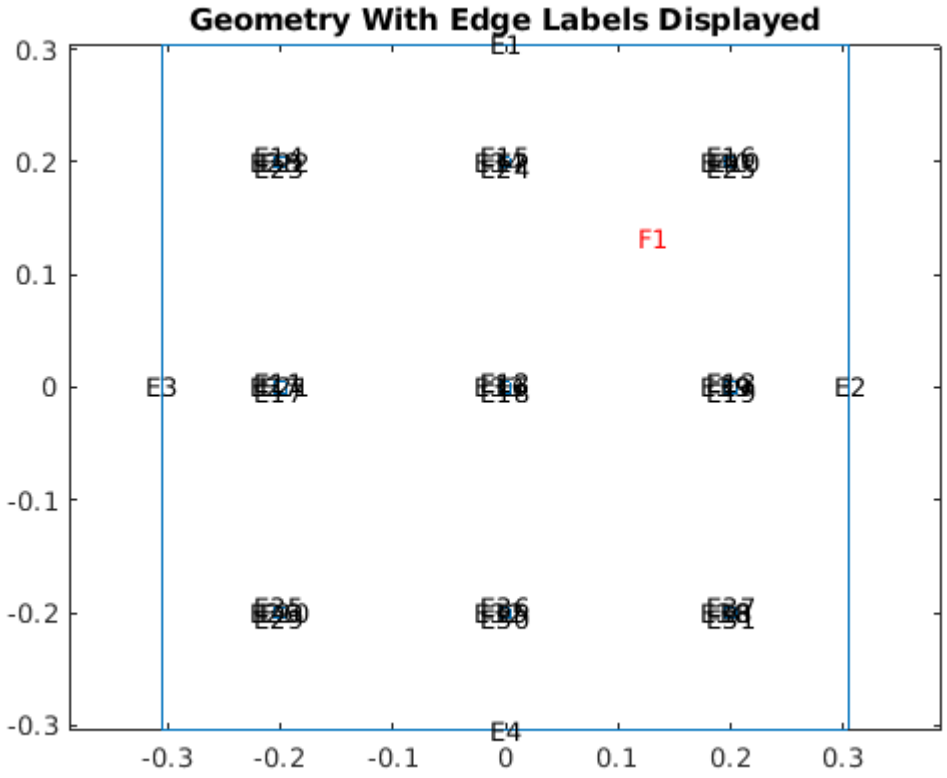
Convert the DECSG geometry into a geometry object

on doing so it is appended to the PDEModel

```
geometryFromEdges(model,g);
```

Plot the geometry and display the edge labels for use in the boundary condition definition.

```
figure;
pdegplot(model,'EdgeLabels','on','FaceLabels','on');
axis auto;
title 'Geometry With Edge Labels Displayed';
% Specify internal heat source.
    %        internalHeatSource(model,1000,'Face',1);
         %   thermalBC(model,"Face",2,'HeatFlux',1000);
```



Definition of PDE Coefficients

The expressions for the coefficients required by PDE Toolbox can easily be identified by comparing the equation above with the scalar parabolic equation in the PDE Toolbox documentation.

```
c = thick*k;
```

Because of the radiation boundary condition, the "a" coefficient is a function of the temperature, u. It is defined as a MATLAB expression so it can be evaluated for different values of u during the analysis.

```
if (enableConvection == 1) & (enableRadiation == 1) & (enableHeatIn == 0)
    f =  2*hCoeff*ta  + 2*emiss*stefanBoltz*ta^4    ;
    a = @(~,state) 2*hCoeff + 2*emiss*stefanBoltz*state.u.^3;
end

if (enableConvection == 1) & (enableRadiation == 1) & (enableHeatIn == 1)
    f =  2*hCoeff*ta  + 2*emiss*stefanBoltz*ta^4    + heatIn  ;
    a = @(~,state) 2*hCoeff + 2*emiss*stefanBoltz*state.u.^3;

end

if (enableConvection == 0) & (enableRadiation == 1) & (enableHeatIn == 1)
    f =                 2*emiss*stefanBoltz*ta^4    + heatIn  ;
    a = @(~,state)      2*emiss*stefanBoltz*state.u.^3;
end

if (enableConvection == 1) & (enableRadiation == 0) & (enableHeatIn == 1)
    f = 2*hCoeff*ta                                 + heatIn  ;
    %a = @(~,state) 2*hCoeff + 2*emiss*stefanBoltz*state.u.^3;
    a =  2*hCoeff ;
end
```

```
if (enableConvection == 1) & (enableRadiation == 0) & (enableHeatIn == 0)
    f = 2*hCoeff*ta                                  ;
    %a = @(~,state) 2*hCoeff + 2*emiss*stefanBoltz*state.u.^3;
    a =  2*hCoeff ;
end


if (enableConvection == 0) & (enableRadiation == 0) & (enableHeatIn == 1)
    f =   heatIn ;
    a = 0;
end

if (enableConvection == 0) & (enableRadiation == 0) & (enableHeatIn == 0)
   % f =  2*hCoeff*ta  + 2*emiss*stefanBoltz*ta^4    + heatIn  ;
    f = 0;
    a = 0;
end

%f = 2*hCoeff*ta + 2*emiss*stefanBoltz*ta^4             + heatIn  ;
d = thick*rho*specificHeat;
%m = -k*thick;
%specifyCoefficients(model,'m',0,'d',0,'c',c,'a',a,'f',f);
```

Boundary Conditions

The boundary conditions are defined below. Three of the plate edges are insulated. Because a Neumann boundary condition equal zero is the default in the finite element formulation, the boundary conditions on these edges do not need to be set explicitly. A Dirichlet condition is set on all nodes on the bottom edge, edge 1,

```
%applyBoundaryCondition(model,'neumann','Edge',5:8,'u',1000);

%applyBoundaryCondition(model,'dirichlet','Edge',1:model.Geometry.NumEdges,'u',300);
%applyBoundaryCondition(model,'dirichlet','Edge',1:4,'u',300);

%applyBoundaryCondition(model,'neumann','edge',[ 1:2 6:7],'g',250, 'q', 0);
%applyBoundaryCondition(model,'neumann','Edge',[3,4,5,8],'g',[],'q',[0]);

% applyBoundaryCondition
```

only if enableHeatIn == 0 , applyBoundaryCondition to plate external edges

```
if enableHeatIn  == 0

%hin = 5.0849e6 /4
hin = 25;
hin=100;
hin=900/4;

% outside perimeter

                applyBoundaryCondition(model,'neumann','Edge',1,'g',hin,'q',[0]);
                applyBoundaryCondition(model,'neumann','Edge',2,'g',hin,'q',[0]);
                applyBoundaryCondition(model,'neumann','Edge',3,'g',hin,'q',[0]);
                applyBoundaryCondition(model,'neumann','Edge',4,'g',hin,'q',[0]);
```

```
% % top left
%                 applyBoundaryCondition(model,'neumann','Edge',14,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',22,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',28,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',23,'g',hin,'q',[0]);
% % top middle
%                 applyBoundaryCondition(model,'neumann','Edge',15,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',7,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',24,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',34,'g',hin,'q',[0]);
% % top right
%                 applyBoundaryCondition(model,'neumann','Edge',16,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',10,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',25,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',40,'g',hin,'q',[0]);
%


%
% % middle left
%                 applyBoundaryCondition(model,'neumann','Edge',11,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',21,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',17,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',27,'g',hin,'q',[0]);
% middle middle
    %                 applyBoundaryCondition(model,'neumann','Edge',6,'g',hin,'q',
[0]);
    %                 applyBoundaryCondition(model,'neumann','Edge',12,'g',hin,'q',
[0]);
    %                 applyBoundaryCondition(model,'neumann','Edge',18,'g',hin,'q',
[0]);
    %                 applyBoundaryCondition(model,'neumann','Edge',33,'g',hin,'q',
[0]);
% % middle right
%                 applyBoundaryCondition(model,'neumann','Edge',9,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',13,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',19,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',39,'g',hin,'q',[0]);
%

% middle middle
               %  applyBoundaryCondition(model,'neumann','Face',2,'g',100,'q',[0]);
               %  applyBoundaryCondition(model,'neumann','Face',3,'g',100,'q',[0]);
                % applyBoundaryCondition(model,'neumann','Face',4,'g',100,'q',[0]);
                % applyBoundaryCondition(model,'neumann','Face',5,'g',100,'q',[0]);


%
% % bottom left
%                 applyBoundaryCondition(model,'neumann','Edge',20,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',29,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',26,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',35,'g',hin,'q',[0]);
% % bottom middle
%                 applyBoundaryCondition(model,'neumann','Edge',5,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',30,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',32,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',36,'g',hin,'q',[0]);
% % bottom right
%                 applyBoundaryCondition(model,'neumann','Edge',8,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',31,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',37,'g',hin,'q',[0]);
%                 applyBoundaryCondition(model,'neumann','Edge',38,'g',hin,'q',[0]);
```

```
%
end;

         % note the above set of neumann boundary conditions seem to be more
        % effective as the perimeter of the innner square is increased.
        % Perhaps its' 1000 W/m^2 * the inner square perimeter area
        %
        % sq = 1*0.025   % small square % 348 deg
        % sq = 2*0.025   % small square % 380 deg
        % sq = 4*0.025   % small square % 438 deg
        % sq = 8*0.025   % small square % 575  deg
        % sq = 10*0.025  % small square %  700 deg


%      applyBoundaryCondition(model,'neumann','Face',1,'g',1000);


      %    applyBoundaryCondition(model,'neumann','Edge',7,'g',[500],'q',[0]);
      %  applyBoundaryCondition(model,'neumann','Edge',5,'g',[550],'q',[0]);

specifyCoefficients(model,'m',0,'d',0,'c',c,'a',a,'f',f);
%F2 = findBoundaryConditions(model.BoundaryConditions,'Edge',1);
%F2 = findBoundaryConditions(model.BoundaryConditions,'Face',2);
```

Initial guess

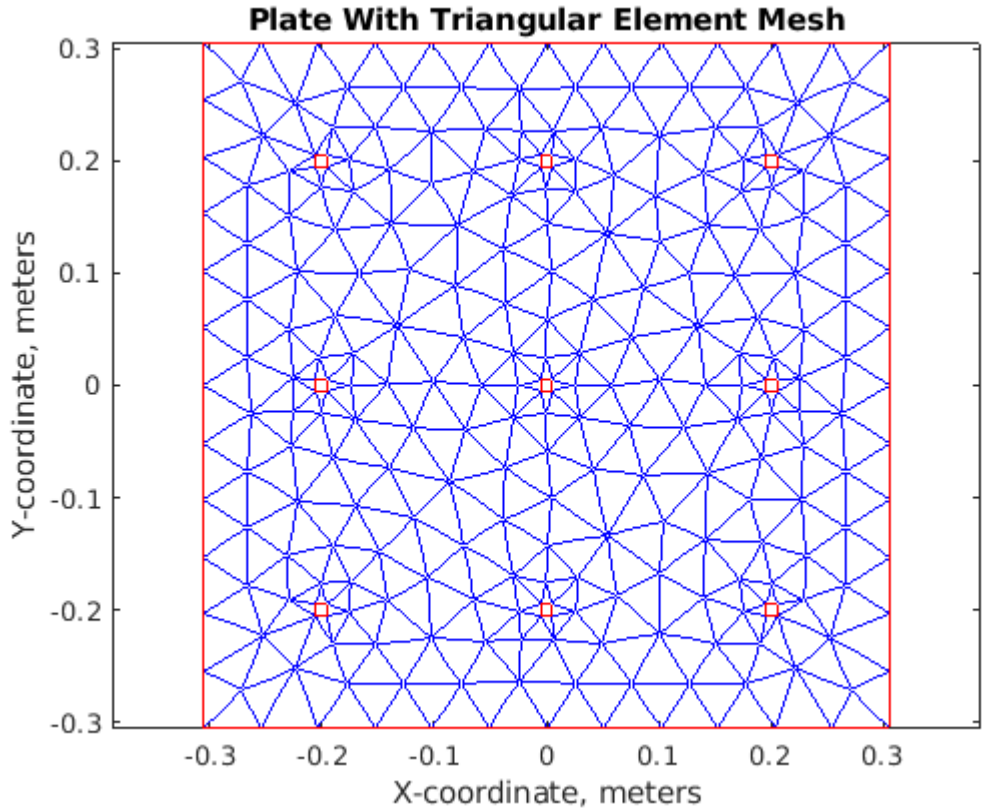
The initial guess is defined below.

```
setInitialConditions(model,300);


%setInitialConditions(model,5,'Face',2);



% setInitialConditions(model,350,'Face',2);
```

Mesh

Create the triangular mesh on the square with approximately ten elements in each direction.

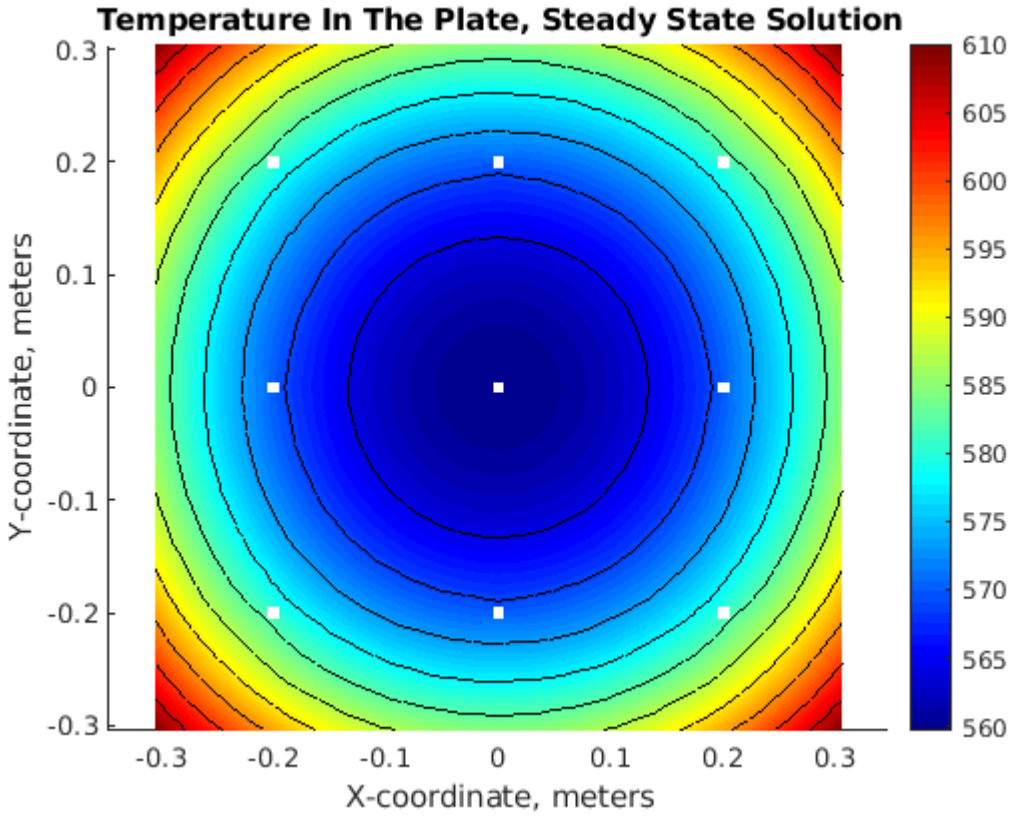
```
hmax = .05; % element size

msh = generateMesh(model,'Hmax',hmax);
figure;
pdeplot(model);
axis auto
title 'Plate With Triangular Element Mesh'
xlabel 'X-coordinate, meters'
ylabel 'Y-coordinate, meters'
```



Steady State Solution

Because the a and f coefficients are functions of temperature (due to the radiation boundary conditions), `solvepde` automatically picks the nonlinear solver to obtain the solution.

```
R = solvepde(model);
u = R.NodalSolution;
figure;
pdeplot(model,'XYData',u(:,end), 'Contour','on','ColorMap','jet');
%pdeplot(model,'XYData',u,'ZData',u);

%     pdeplot(model,'XYData',u,'ZData',u);


%pdeplot(model,'XYData',u,'Contour','on','ColorMap','jet');


title 'Temperature In The Plate, Steady State Solution'
xlabel 'X-coordinate, meters'
  ylabel 'Y-coordinate, meters'
axis equal

%
p = msh.Nodes;
% plotAlongY(p,u,0);
% title 'Temperature As a Function of the Y-Coordinate'
% xlabel 'Y-coordinate, meters'
% ylabel 'Temperature, degrees-Kelvin'
% fprintf('Temperature at the top edge of the plate = %5.1f degrees-K\n', ...
%   u(4));
%
```



Transient Solution

Include the "d" coefficient.

```
specifyCoefficients(model,'m',0,'d',d,'c',c,'a',a,'f',f);
endTime = 10000;
tlist = 0:50:endTime;
numNodes = size(p,2);
```

Set the initial temperature of all nodes to ambient, 300 K.

```
u0(1:numNodes) = 300;

                %setInitialConditions(model,300,'face',2);
```

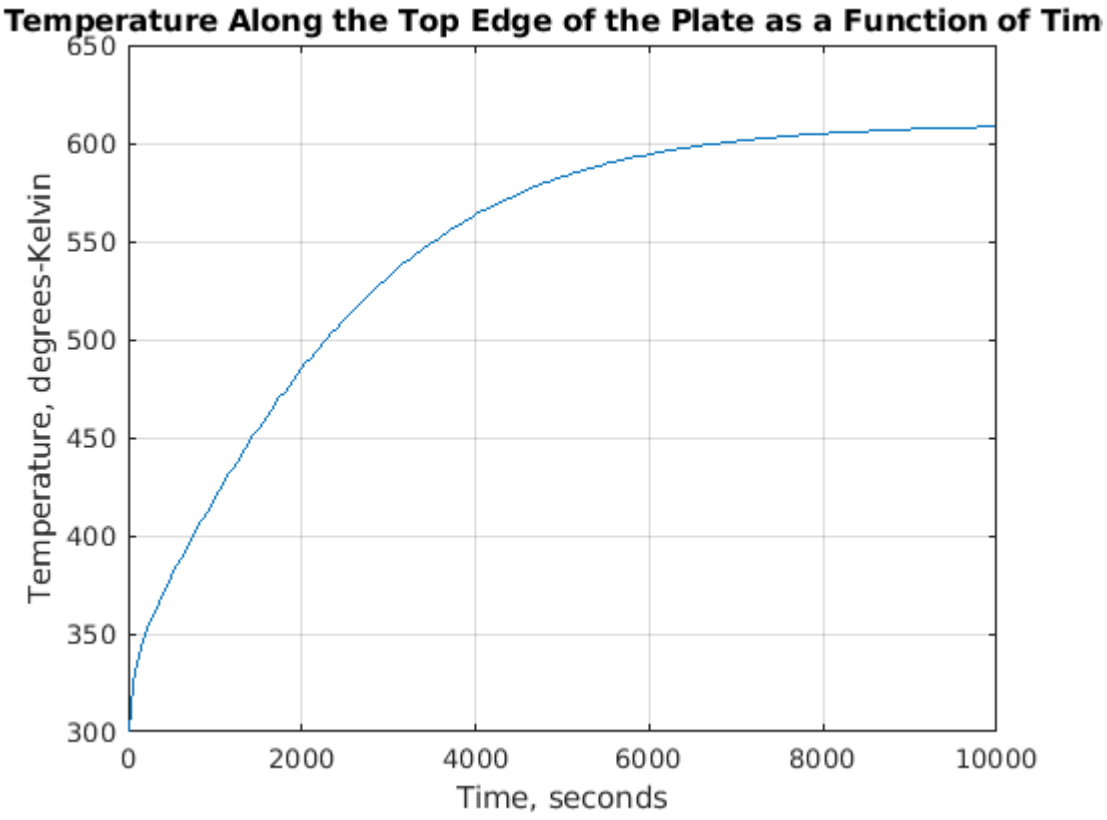
Set solver options.

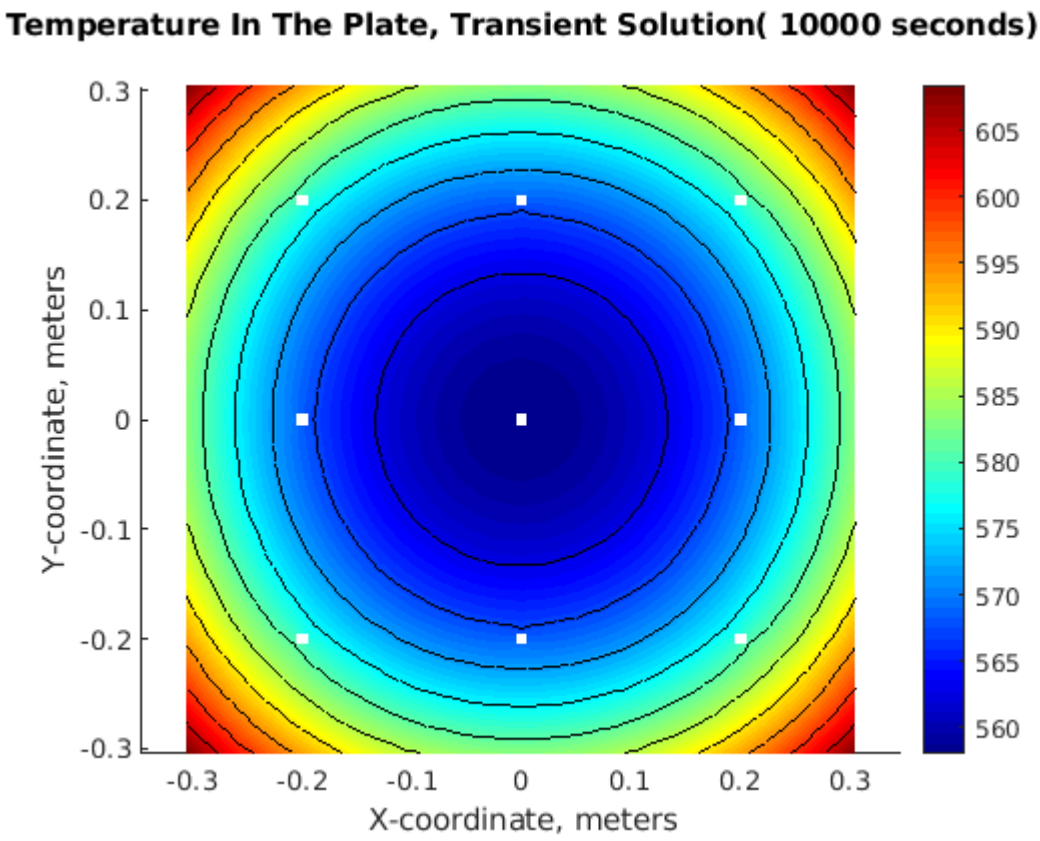
```
model.SolverOptions.RelativeTolerance = 1.0e-3;
model.SolverOptions.AbsoluteTolerance = 1.0e-4;
```

`solvepde` automatically picks the parabolic solver to obtain the solution.

```
R = solvepde(model,tlist);
u = R.NodalSolution;
                    %

figure;
plot(tlist,u(3, :));

model.SolverOptions.ReportStatistics ='on';
```

```
grid on
title 'Temperature Along the Top Edge of the Plate as a Function of Time'
xlabel 'Time, seconds'
ylabel 'Temperature, degrees-Kelvin'

figure;
% pdeplot(model,'XYData',u(:,end),'Contour','on','ColorMap','jet');
pdeplot(model,'XYData',u(:,end),'Contour','on','ColorMap','jet');
title(sprintf('Temperature In The Plate, Transient Solution( %d seconds)\n', ...
  tlist(1,end)));
xlabel 'X-coordinate, meters'
ylabel 'Y-coordinate, meters'
axis equal;


%
% fprintf('\nTemperature at the top edge(t=%5.1f secs)=%5.1f degrees-K\n', ...
%   tlist(1,end), u(4,end));
%
```





Summary

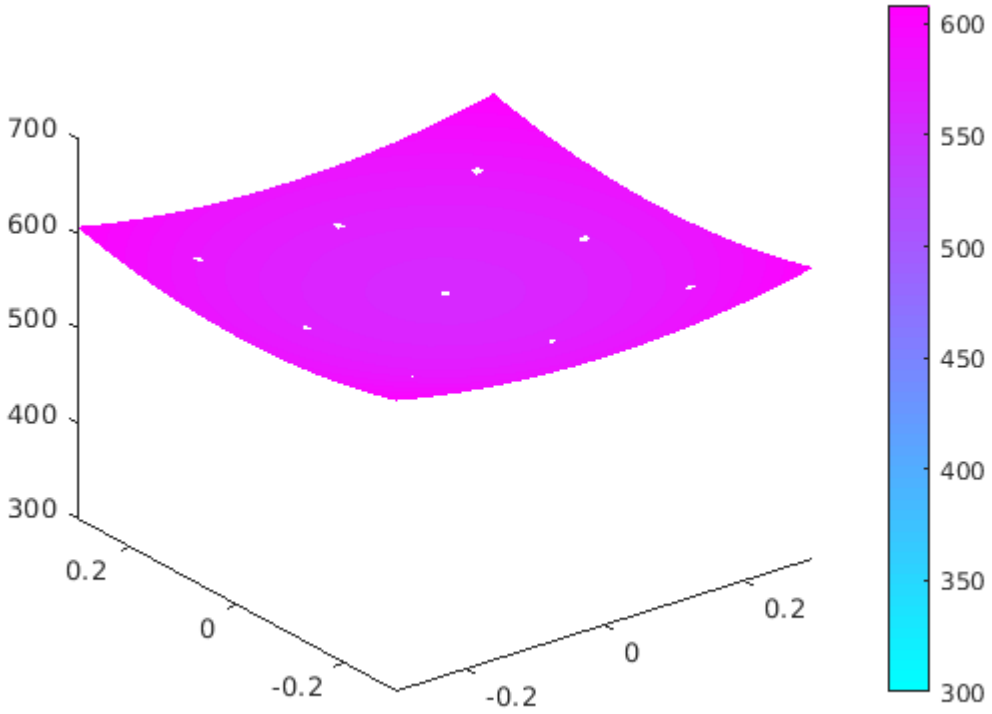
As can be seen, the plots of temperature in the plate from the steady state and transient solution at the ending time are very close. That is, after around 5000 seconds, the transient solution has reached the steady state values. The temperatures from the two solutions at the top edge of the plate agree to within one percent. Plot the solution.

Find and Plot Solution

Find the solution at 20 points in time between 0 and 0.1.

```
nframes = 201;
% tlist = linspace(0,0.1,nframes);
%
% thermalmodel.SolverOptions.ReportStatistics ='on';
% result = solve(thermalmodel,tlist);
% T = result.Temperature;

figure

 T = u;
%
Tmax = max(max(T));
Tmin = min(min(T));
%Tmin = min(T);
for j = 1:nframes
        % pdeplot(model,'XYData',u(:,j),'Contour','on','ColorMap','jet');
      %  pdeplot(model,'XYData',u(:,j),'ColorMap','jet');
%
        pdeplot(model,'XYData',u(:,j),'ZData',u(:,j));
```

```
    % pdeplot(model,'XYData',u,'ZData',u);
  %   pdeplot(model,'XYData',u(:,j));
    caxis([Tmin Tmax]');
 %   caxis('auto');
  %axis([-1 1 -1 1 0 1]);

  axis([-0.3 0.3 -0.3 0.3    300 800])
  axis([-0.3 0.3 -0.3 0.3    300 400])
  axis([-0.3 0.3 -0.3 0.3    300 500])
  axis([-0.3 0.3 -0.3 0.3    300 500])
  axis([-0.3 0.3 -0.3 0.3    300 700])



 % axis auto;
     Mv(j) = getframe;
%   pause(1);
 end
 %movie(Mv,1);
```



Published with MATLAB® R2018a