Jose A. Millan
April 4, 2019

ME 254 Computational Fluid Dynamics
Homework 5

Due on 04/04/19 at 11:59 pm (through Catcourses)
Maximum points: 200

1. (**100 points**) Consider the heat equation in two-dimensions ($0 \leq x \leq 1$) and ($0 \leq y \leq 1$).

$$\frac{\partial u}{\partial t} = 0.01\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{1}$$

The initial conditions are

$$u(x, y, 0) = \sin(2\pi x)\sin(2\pi y) \tag{2}$$

Assume the boundary conditions are

$$u(0, y, t) = 0 \tag{3}$$
$$u(1, y, t) = 0 \tag{4}$$
$$u(x, 0, t) = 0 \tag{5}$$
$$u(x, 1, t) = 0 \tag{6}$$

The exact solution is given by

$$u(x, y, t) = \exp(-0.08\pi^2 t)\sin(2\pi x)\sin(2\pi y) \tag{7}$$

   (a) (**65 points**) Write a code to solve the above problem numerically using the ADI method. You should use the Thomas algorithm for solving the resulting tridiagonal system of equations.

   (b) (**10 points**) Run your code for $r_x = r_y = 0.5$ and $\Delta x = \Delta y = 0.1$. Show contour plots of your results at $t = 0.4$ s and $t = 1$ s. Determine the L_2 norm for your numerical solution using the exact solution as reference.

   (c) (**10 points**) Run your code for same r_x,r_y as above but with a $\Delta x = \Delta y = 0.05$. Show contour plots of your results at $t = 0.4$ s and $t = 1$ s. Determine the L_2 norm for your numerical solution using the exact solution as reference.

   (d) (**10 points**) Run your code for same r_x,r_y and $\Delta x = 0.025$ and $\Delta y = 0.025$. Show contour plots of your results at $t = 0.4$ s and $t = 1$ s. Determine the L_2 norm for your numerical solution using the exact solution as reference.

   (e) (**5 points**) Determine the order of convergence of the ADI scheme using your L_2 norms.

2. (**100 points**) Consider the 2-D steady state heat conduction equation in the unit square, $0 \leq x \leq 1$, $0 \leq y \leq 1$, with boundary conditions given by

$$T(0, y) = 0 \tag{8}$$
$$T(1, y) = 0 \tag{9}$$
$$\frac{\partial T}{\partial y}(x, 0) = 0 \tag{10}$$
$$T(x, 1) = \sin(\pi x) \tag{11}$$

   (a) (**55 points**) Write a function with the capability to use either the Gauss-Jacobi or Gauss-Seidel iterative solution for the Laplace's equation. The function should be able to accept user-defined parameters to set a maximum number of iterations, a

tolerance to use for the convergence criterion and a relaxation parameter ω. One possible convergence criterion is

$$\max\left(\left|\frac{T_{i,j}^{k+1}-T_{i,j}^{k}}{T_{\max}}\right|\right)\leq\epsilon \tag{12}$$

where k refers to the iteration index and ϵ is some tolerance. Note that the $T_{\max}$ value could occur at the boundary.

(b) (**5 points**) Use the Gauss-Seidel method (with no relaxation) using a 65×65 mesh to solve the above PDE. Note down the number of iterations to converge to $\epsilon = 10^{-6}$. Also record the CPU time elapsed.

(c) (**5 points**) Now use the Gauss-Jacobi method (with no relaxation) with same parameters to determine the temperature distribution. Note down the number of iterations to converge to $\epsilon = 10^{-6}$. Also record the CPU time elapsed.

(d) (**20 points**) Now use the Gauss-Seidel method with $\omega = 0.6$, $\omega = 0.9$, $\omega = 1.1$ and $\omega = 1.4$. Note down the number of iterations to converge to $\epsilon = 10^{-6}$. Also record the CPU time elapsed.

(e) (**5 points**) Which value of ω leads to fastest convergence?

(f) (**10 points**) Plot contours of the steady-state temperature distribution (this should be the same for all the methods and so one plot is sufficient).

```
clc
clear all
close all

% The following code gives you the temperature gradient of a 2D mesh
% using the ADI Method

%constants
alpha = 0.01;
r_x = 0.5;
r_y = 0.5;

% setting up the mesh
% differential to be used
dx = 0.025;
dy = 0.025;
dt = (r_x*dx^2)/alpha;

% dimensions of the mesh in question (x and y)
dimension = length(0:dx:1);

% since for the 1st case we will use a time of t = 1, this will
 determine the number of
% time steps, position vector in the x and y direction
time = 0:dt:1;
x_direction = 0:dx:1;
y_direction = 0:dy:1;

% mesh parameters
T = zeros(dimension,dimension,length(time));

% general mesh parameters for position
x = x_direction.*ones(dimension,dimension);
y = y_direction'.*ones(dimension,dimension);

for i=2:1:dimension-1
 for w=2:1:dimension-1
  T(i,w,1) = sin(2*pi*x_direction(w))*sin(2*pi*y_direction(i));
 end
end

% for the half part of the algorithm
T_half1 = zeros(dimension,dimension);

a_un = (2*(dx^2+alpha*dt))/(dt*dx^2);
b_un = -alpha/dx^2;
c_un = -alpha/dx^2;

a_sol = (2*(dy^2-alpha*dt))/(dt*dy^2);
b_sol = alpha/dy^2;
c_sol = alpha/dy^2;
```

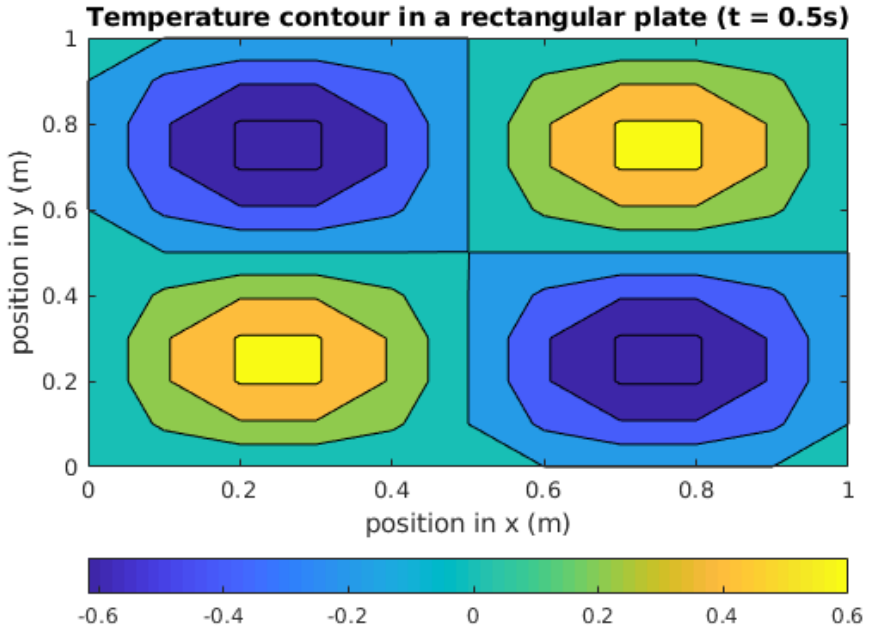
```
% we really need to repeat this for all the internal points in
% order to get the 1/2 answer. How do we do that?

for tt=2:1:length(time)
 for w=2:1:dimension-1
  % this is for the a_sol
  for i=3:1:dimension-1
   if i==3
    a_un_M(i-2,1) = a_un-(c_un*b_un)/(a_un);
   elseif i>3
    a_un_M(i-2,1) = a_un-(c_un*b_un)/(a_un_M(i-3,1));
   end
  end

  for i=2:1:dimension-1
   k(i-1,1) = b_sol*T(i,w
+1,tt-1)+a_sol*T(i,w,tt-1)+c_sol*T(i,w-1,tt-1);
  end

  %this is for the k_sol
  for i=3:1:dimension-1
   if i==3
    k_M(i-2,1) = k(i-1,1)-((c_un)*k(i-2,1))/a_un;
   elseif i>3
    k_M(i-2,1) = k(i-1,1)-((c_un)*k_M(i-3,1))/a_un_M(i-3,1);
     end
  end

  % now for setting up the values I just obtained
  for i=length(k_M)+1:-1:1
      if i==length(k_M)+1
          u_k(i,1) = k_M(i-1,1)/a_un_M(i-1,1);
      elseif i>1
          u_k(i,1) = (k_M(i-1,1)-(b_un*u_k(i+1,1)))/a_un_M(i-1,1);
      else
          u_k(i,1) = (k(1,1)-(b_un*u_k(i+1,1)))/a_un;
      end
  end

  T_half1(w,2:dimension-1) = u_k';

  clear a_un_M;
  clear k_M;
  clear k;

  for w=2:1:dimension-1
   % this is for the a_sol
   for i=3:1:dimension-1
    if i==3
     a_un_M(i-2,1) = a_un-(c_un*b_un)/(a_un);
    elseif i>3
     a_un_M(i-2,1) = a_un-(c_un*b_un)/(a_un_M(i-3,1));
    end
   end
```

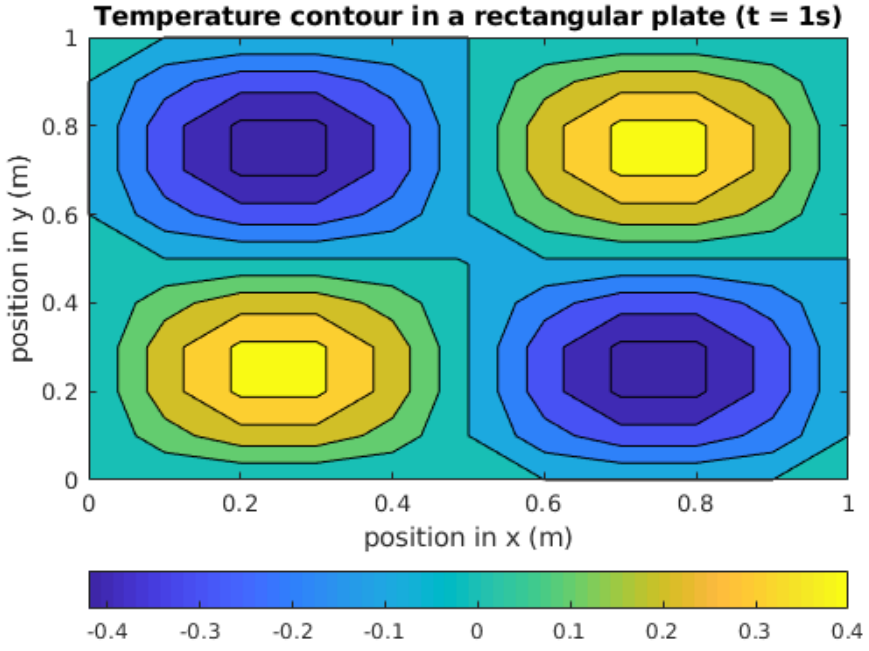
```
   for i=2:1:dimension-1
    k(i-1,1) = b_sol*T_half1(w
+1,i)+a_sol*T_half1(w,i)+c_sol*T_half1(w-1,i);
           end

   %this is for the k_sol
   for i=3:1:dimension-1
    if i==3
     k_M(i-2,1) = k(i-1,1)-((c_un)*k(i-2,1))/a_un;
    elseif i>3
     k_M(i-2,1) = k(i-1,1)-((c_un)*k_M(i-3,1))/a_un_M(i-3,1);
      end
   end

   % now for setting up the values I just obtained
   for i=length(k_M)+1:-1:1
       if i==length(k_M)+1
           u_k(i,1) = k_M(i-1,1)/a_un_M(i-1,1);
       elseif i>1
           u_k(i,1) = (k_M(i-1,1)-(b_un*u_k(i+1,1)))/a_un_M(i-1,1);
       else
           u_k(i,1) = (k(1,1)-(b_un*u_k(i+1,1)))/a_un;
       end
   end

   T(2:dimension-1,w,tt) = u_k;
  end
    end
    T_half1 = zeros(dimension,dimension);
end

% Now lets do this for the exact solution
T_exact = zeros(dimension,dimension,length(time));

for tt=1:1:length(time)
    for i=2:1:(dimension-1)
        for w=2:1:(dimension-1)
            T_exact(i,w,tt) =
 exp(-0.08*pi^2*time(tt))*sin(2*pi*x_direction(w))*sin(2*pi*y_direction(i));
        end
    end
end

% so how many points the matrix in question has?
M = dimension*dimension;

err = 0;

for i=1:1:dimension
    for w=1:1:dimension
        err = err+(T(i,w,length(time))-T_exact(i,w,length(time)))^2;
    end
end
```

```
L_2 = sqrt(err/M);

% figure(1)
% contourf(x_direction,y_direction,T(:,:,33));
% title('Temperature contour in a rectangular plate (t = 1s)');
% xlabel('position in x (m)');
% ylabel('position in y (m)');
% colorbar('southoutside')
%
% figure(2)
% contourf(x_direction,y_direction,T(:,:,18));
% title('Temperature contour in a rectangular plate (t = 0.5s)');
% xlabel('position in x (m)');
% ylabel('position in y (m)');
% colorbar('southoutside')
```

Published with MATLAB® R2018b

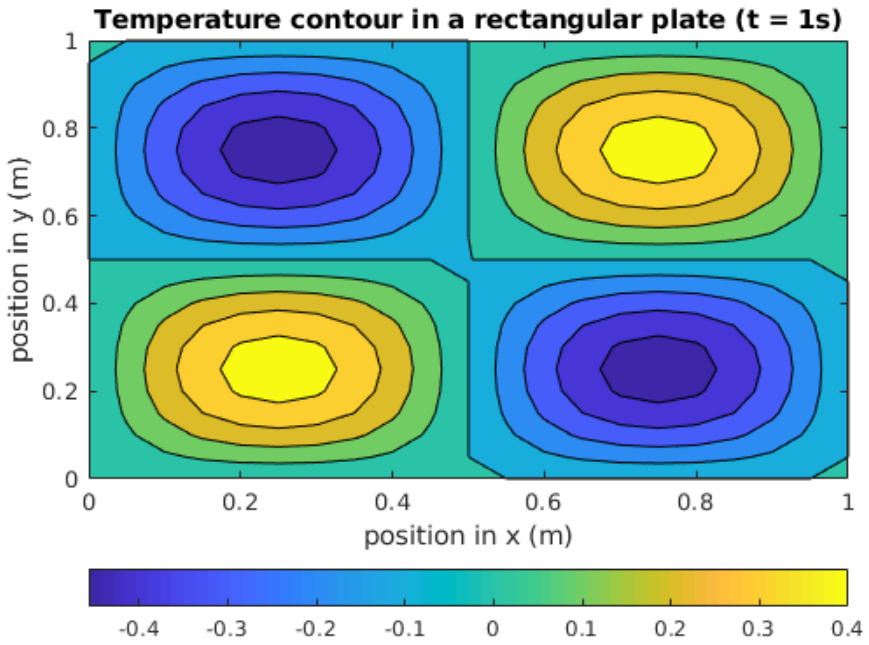
(b) for $r_x=r_y=0.5$; $\Delta x = \Delta y = 0.1$; t = 0.5s, t = 1s

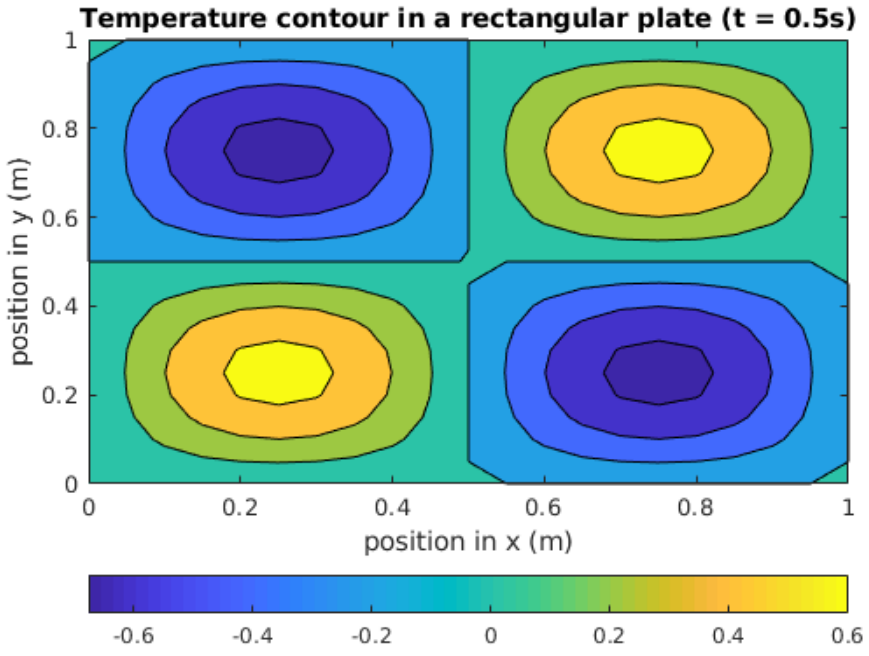




Time (s)	L^2
0.5	0.003589316189784
1	0.00286443005219

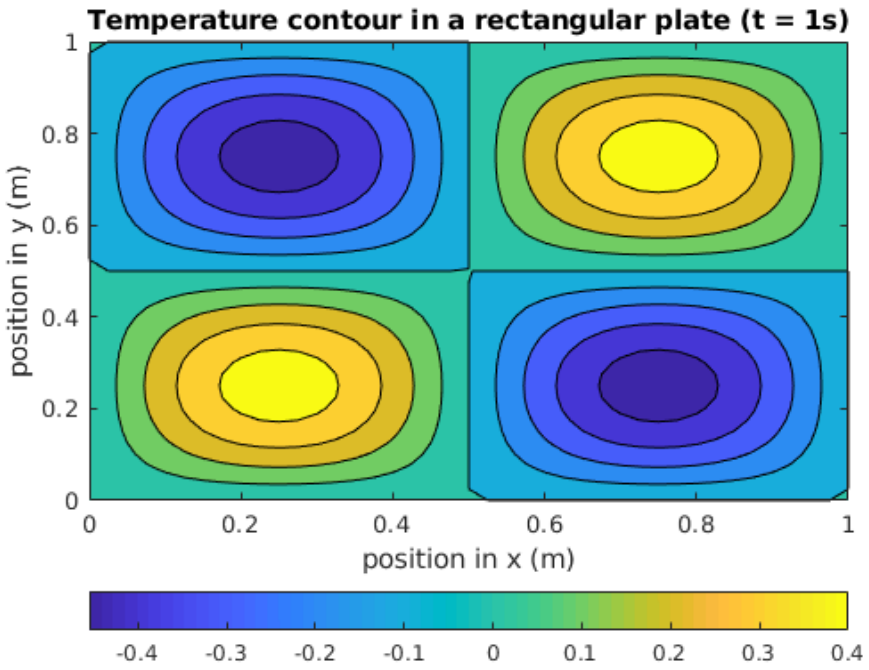
(c) for $r_x=r_y=0.5$; $\Delta x = \Delta y = 0.05$; t = 0.5s, t = 1s

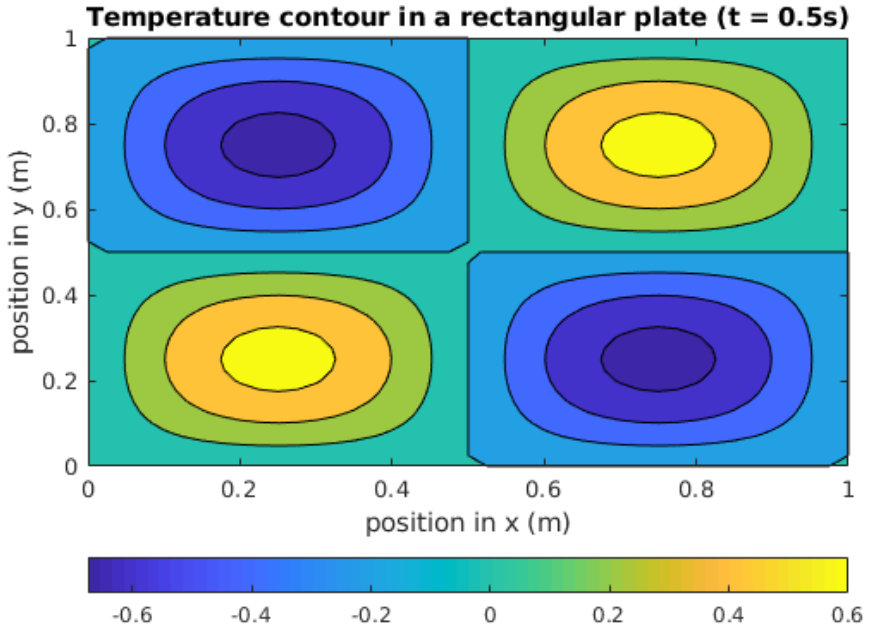




Time (s)	L^2
0.5	0.001014944395506
1	0.000827657276233

(d) for $r_x=r_y=0.5$; $\Delta x = \Delta y = 0.025$; t = 0.5s, t = 1s

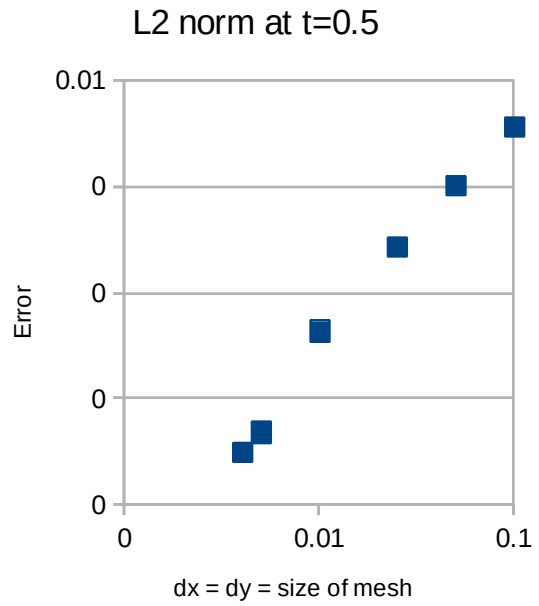


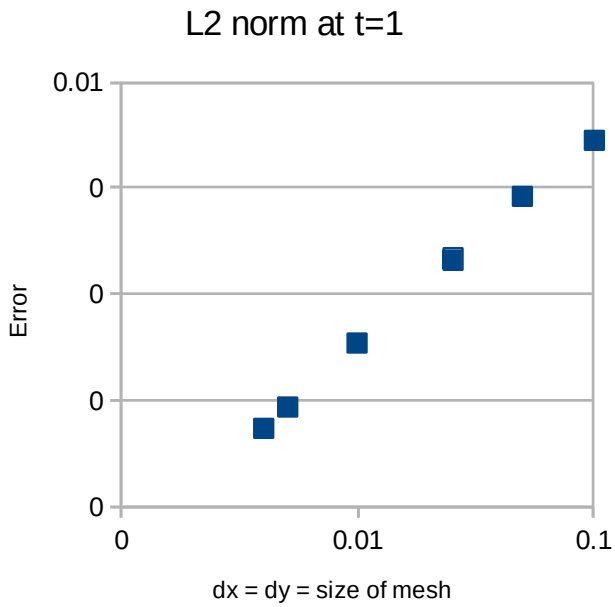


`

Time (s)	L^2
0.5	0.00026506765560376
1	0.000214677315075907

(e) Determine the order of convergence of the ADI scheme using your L^2 norms.





Time (s)	m
0.5	2.19532087432088
1	1.93984686941209

```
function T = iterativeMethod(x,y,relaxation000,algorithm)

if strcmp(algorithm,'jacobi')
    tic
    % The following program will use the iterative method in order to
    % to get a solution regarding a steady state heat system

    % setting up the mesh
    dx = 1/x;
    dy = 1/y;

    % dimensions of the mesh to be used
    % x and y. The iterations will be dealt apart
    dimension = length(0:dx:1);

    %position in the mesh
    x_direction = 0:dx:1;
    y_direction = 0:dy:1;

    % mesh at initial conditions
    % lets assume that all the unknowns are 0 and from there part
    T_initial = zeros(dimension,dimension);

    for i=1:1:dimension-1
        T_initial(dimension,i) = sin(pi*x_direction(i));
    end

    % since we need a T_max which is global, I will ask the program to
    % give me the largest number possible in the initial matrix
    T_max = max(T_initial(:));

    % now we need to set up the loop. The error will be set up in a
    % arbitrary way just for initialization purporses

    e = 1;                                % initialization of error
    k = 1;                                % initialization of time steps
    T = zeros(dimension,dimension,1);     % Temperature iteration matrix
    T(:,:,1) = T_initial;                 % Initial condition
    T_difference = zeros(dimension,dimension,1);

    %This is for the jacobi method
    while e > (1E-6)
        k = k+1;
        for w = dimension-1:-1:2
            for i = 2:1:dimension-1
                T(w,i,k) = (T(w,i+1,k-1)+T(w,i-1,k-1)+T(w
+1,i,k-1)+T(w-1,i,k-1))/4;
            end
        end

        % for the derivative boundary condition
        T(1,2:dimension-1,k) = T(2,2:dimension-1,k);
```

```
            % for the boundary condition when y=1
            for i=1:1:dimension-1
                T(dimension,i,k) = sin(pi*x_direction(i));
            end

            T_difference = abs(T(:,:,k)-T(:,:,k-1));

            %now which is the largest absolute difference
            T_max_diff = max(T_difference(:));

            e = abs(T_max_diff/T_max);
        end
        toc

        figure(1)
        contourf(x_direction,y_direction,T(:,:,k));
        title('Steady State Temperature');
        xlabel('position in x (m)');
        ylabel('position in y (m)');
        colorbar('southoutside')

    elseif strcmp(algorithm,'seidel')
        tic
        % the following program will use use a iterative method in order
     to
        % get a solution regarding a steady state heat system. For this
     case
        % we use the gauss-seidel method

        % setting up the mesh
        dx = 1/x;
        dy = 1/y;

        % dimension of the mesh to be used. x and y. The iteration will be
        % dealt apart
        dimension = length(0:dx:1);

        % position in the mesh
        x_direction = 0:dx:1;
        y_direction = 0:dy:1;

        % mesh at initial conditions
        % lets assume that all unknows are 0 for this part
        T_initial = zeros(dimension,dimension);

        for i=1:1:dimension-1
            T_initial(dimension,i) = sin(pi*x_direction(i));
        end

        % since we need a T_max which is global, I will ask the program to
        % give me the largest nummber possible in the initial matrix
        T_max = max(T_initial(:));
```

```
    % now we need to set up the loop. I will ask the program to give
 me the
    % largest number possible in the initial matrix

    e = 1;
    k = 1;
    T = zeros(dimension,dimension,1);
    T(:,:,1) = T_initial;
    T_difference = zeros(dimension, dimension,1);

    % this is for gauss-seidal method
    while e > (1E-6)
        k = k+1;
        T(:,:,k) = zeros(dimension,dimension,1);
        % this is for the derivative boundary condition
        for i=2:1:dimension-1
            T(2,i,k) = (T(2,i+1,k-1)+T(2,i-1,k)+T(3,i,k-1))/3;
        end

        % for the derivative boundary condition
        T(1,2:dimension-1,k) = T(2,2:dimension-1,k);

        % now lets get into filling everything
        for w = 3:1:dimension-1
            for i=2:1:dimension-1
                T(w,i,k) = (T(w,i+1,k-1)+T(w,i-1,k)+T(w
+1,i,k-1)+T(w-1,i,k))/4;
            end
        end

        % for the boundary condition when y=1
        for i=1:1:dimension-1
            T(dimension,i,k) = sin(pi*x_direction(i));
        end

        T_difference = abs(T(:,:,k)-T(:,:,k-1));

        %now which is the largest absolute difference
        T_max_diff = max(T_difference(:));

        e = abs(T_max_diff/T_max);
    end
    toc

    figure(1)
    contourf(x_direction,y_direction,T(:,:,k));
    title('Steady State Temperature');
    xlabel('position in x (m)');
    ylabel('position in y (m)');
    colorbar('southoutside')

elseif strcmp(algorithm,'relax')
    tic
    % setting up the mesh
```

```
    dx = 1/x;
    dy = 1/y;

    % dimension of the mesh to be used. x and y. The iteration will be
    % dealt apart
    dimension = length(0:dx:1);

    % position in the mesh
    x_direction = 0:dx:1;
    y_direction = 0:dy:1;

    % mesh at initial conditions
    % lets assume that all unknows are 0 for this part
    T_initial = zeros(dimension,dimension);

    for i=1:1:dimension-1
        T_initial(dimension,i) = sin(pi*x_direction(i));
    end

    % since we need a T_max which is global, I will ask the program to
    % give me the largest nummber possible in the initial matrix
    T_max = max(T_initial(:));

    % now we need to set up the loop. I will ask the program to give
 me the
    % largest number possible in the initial matrix

    e = 1;
    k = 1;

    T_seidel_new = zeros(dimension,dimension);
    T_seidel_old(:,:) = T_initial;

    T_new = zeros(dimension,dimension);
    T_old = T_initial.*ones(dimension,dimension);

    T_difference = zeros(dimension, dimension);

    ww = relaxation000;

    % this is for gauss-seidal methods
    while e > (1E-6)

        k = k+1;

        % this is for the derivative boundary condition
        for i=2:1:dimension-1
            T_seidel_new(2,i) = (T_seidel_old(2,i
+1)+T_seidel_new(2,i-1)+T_seidel_old(3,i))/3;
            T_new(2,i) = ww.*T_seidel_new(2,i) + (1-ww).*T_old(2,i);
        end

        % for the derivative boundary condition
        T_seidel_new(1,2:dimension-1) = T_seidel_new(2,2:dimension-1);
```

```
        T_new(1,2:dimension-1) = T_new(2,2:dimension-1);

        % now lets get into filling everything
        for w = 3:1:dimension-1
            for i=2:1:dimension-1
                T_seidel_new(w,i) = (T_seidel_old(w,i
+1)+T_seidel_new(w,i-1)+T_seidel_old(w+1,i)+T_seidel_new(w-1,i))/4;
                T_new(w,i) = ww.*T_seidel_new(w,i) + (1-
ww).*T_old(w,i);
            end
        end

        % for the boundary condition when y=1
        for i=1:1:dimension-1
            T_seidel_new(dimension,i) = sin(pi*x_direction(i));
            T_new(dimension,i) = sin(pi*x_direction(i));
        end

        T_difference = abs(T_new(:,:)-T_old(:,:));

        %now which is the largest absolute difference
        T_max_diff = max(T_difference(:));

        T_seidel_old = T_seidel_new;
        T_old = T_new;

        e = abs(T_max_diff/T_max);

    end
    toc

    figure(1)
    contourf(x_direction,y_direction,T_new);
    title('Steady State Temperature');
    xlabel('position in x (m)');
    ylabel('position in y (m)');
    colorbar('southoutside')
end

PRINT_THIS = ['Number of iterations: ',num2str(k)];
disp(PRINT_THIS);
end
```

```
Not enough input arguments.

Error in iterativeMethod (line 3)
if strcmp(algorithm,'jacobi')
```

Published with MATLAB® R2018b

(b) CPU time elapsed = Elapsed time is 0.651543 seconds. 6979 iterations.
(c) CPU time elapsed = Elapsed time is 0.306079 seconds. 3955 iterations.

(d) CPU time elapsed by relaxation factor (w)

Relaxation factor (w)	CPU time elapsed	# of iterations
0.6	0.229825	3956
0.9	0.247120	3955
1.1	0.231245	3955
1.4	0.216365	3955

(e) 1.4 leads the fastes convergence

(f) Plot of steady state condition