Project: 7.6 Insulation

May 1, 2019

To begin, I will work out the analytic solution to (2), by separation of variables, subject to the boundary conditions stated in the question.

Analytic Solution to (2):

The boundary conditions on $T(x,y) = X(x)Y(y)$ give us conditions on $X(x)$ and $Y(y)$:

$$\begin{aligned}
T(0,y) = 0 &\implies X(0)Y(y) = 0 \implies X(0) = 0 \\
T(1,y) = 1 &\implies X(1)Y(y) = 1 \implies X(1) = \frac{1}{Y(y)} \Rightarrow Y(y) \text{ is Constant} \\
\frac{\partial T}{\partial y}(x,0) = 0 &\implies X(x)Y'(0) = 0 \implies Y'(0) = 0 \\
\frac{\partial T}{\partial y}(x,1) = 0 &\implies X(x)Y'(1) = 0 \implies Y'(1) = 0
\end{aligned}$$

Hence, solving (2) gives us:

$$\begin{aligned}
\boldsymbol{\nabla}^2 T(x,y) &= 0 \\
\boldsymbol{\nabla}^2 X(x)Y(y) &= 0 \\
X''(x)Y(y) &= 0 \\
X(x) &= Ax + B, \text{ where } A, B \in \mathbb{R}
\end{aligned}$$

Applying the boundary conditions, we can work out A and B:

$$\begin{aligned}
X(0) = 0 &\Rightarrow B = 0 \\
X(1) = \frac{1}{Y(y)} &\Rightarrow A = \frac{1}{Y(y)}
\end{aligned}$$

Therefore the solution to (2) is:

$$\begin{aligned}
T(x,y) &= \frac{x}{Y(y)} Y(y) \\
T(x,y) &= x
\end{aligned}$$

1. **Describe how you decided whether the T array had converged. Experiment with values of σ and comment on the effect of changing σ on the number of iterations required for convergence. Include in your write-up a contour plot of the steady-state temperature distribution of $T(x,y)$.**

   To decide whether the T array had converged, I compared two sucessive iterates of the T array, the old T array and the new T array. I chose to minimise the root mean square (RMS) error between these two arrays which gave a condition for convergence. My program minimised the RMS error to within 10^{-10}, which gave the total sum of the error in the T array correct to at least 5 significant figures.

   I have chosen $N_x = N_y = 100$ with the contour plot showing 50 contours. This allows for good accuracy of what the temperature distribution is as well as being able to resolve the contour lines.

   The table below shows a selection of results on the number of iterations required for convergence for different values of σ. Here, one iteration is defined to be updating all elements in the T array to the next value.

   As σ approaches 2 from below, the number of iterations required for convergences increases rapdily. Eventually for $\sigma \geq 2$, the algorithm does not converge at all.

σ	**Iteration**
0.3	40,891
0.6	21,048
0.9	12,639
1.0	10,752
1.3	6,476
1.6	3,406
1.9	877
1.95	411
1.99	1024
1.999	10,613
≥ 2	Na

Table 1: Table showing how the number of iterations required to obtain convergence differs for different values of σ.

Below is a contour plot of the steady state solution that all of the algorithms for different σ's converge to, to 3 significant figures.

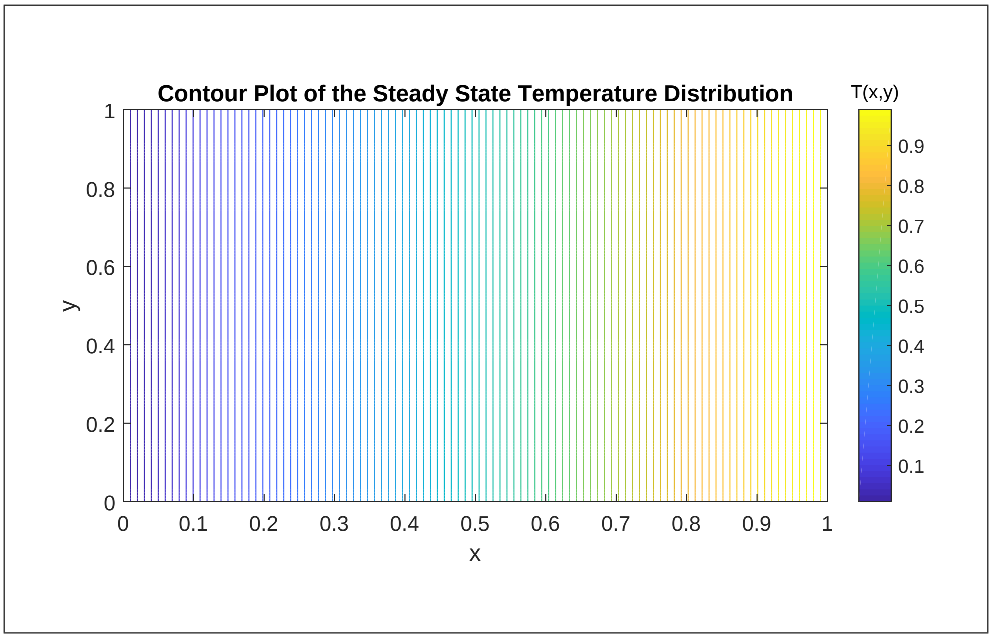

Figure 1: Contour plot of the steady state temperature distribution of $T(x, y)$ for the initial given boundary conditions.

My modified program `RelaxationMethodWithInsulator.m` works by first finding out how many insulating pieces can fit between the top and the bottom of the box, spaced at a constant ϵ apart with an ϵ length of space above the first and below the last insulating piece. In most cases, there will be a leftover length which the program then divides by two and adds to both the top and bottom ends of this strip of insulating pieces and holes. This then means we have symmetrically placed vertical pieces of insulation with holes at a symmetrically spaced, with a length of insulator at the top and bottom attached to the box.

In the special case that you can fit an exact number of insulating pieces between the top and the bottom of the box, the program removes one of the insulating pieces and an ϵ length hole, divides this length by two (equal to δ), and uses these two equal lengths to create the top and bottom pieces of insulation. This is also perfectly symmetric about a horizontal axis through the midpoint of the box.

In terms of symmetry about a vertical axis through the midpoint of the box, this is not always the case. When N is odd, there is perfect symmetry since the insulating strip can be placed between $j = \frac{N-1}{2}$ and $j = \frac{N-1}{2} + 1$. When N is even, perfect symmetry cannot be obtained, hence the program chooses to set the insulating strip slightly to the left of the true symmetric position.

2. (a) **Write down and explain the formulae you used to compute $T_{i,j}^{\text{new}}$ at the new types of boundary grid point A-J. Include in your write-up a contour plot of the steady-state temperature distribution of $T(x, y)$ for the case $\epsilon = 4\Delta, \delta = 8\Delta, N_x = N_y = 128$. What effect does putting holes in the wall have on the temperature distribution in the unit square?**

On the vertical boundaries of the insulating wall, points C and D, the appropriate boundary condition is:

$$\frac{\partial T}{\partial x}(x_i, y_j) = 0, \tag{1}$$

where (x_i, y_j) is a point on the edge of the vertical insulating wall. On the corner boundaries, such as points A, B, E, F, G, H, I and J, we want the the following boundary condition:

$$\begin{aligned}
&\frac{\partial T}{\partial r}(x_i, y_i) = 0 \text{ where } r \text{ is standard polar coordinate relative to } (x_i, y_i). \\
\Rightarrow\ &\frac{\partial T}{\partial r}(x_i, y_i) = \frac{\partial T}{\partial x}(x_i, y_i)\frac{\partial x}{\partial r} + \frac{\partial T}{\partial y}(x_i, y_i)\frac{\partial y}{\partial r} \\
&\qquad\qquad\quad = \frac{\partial T}{\partial x}(x_i, y_i)\cos\theta + \frac{\partial T}{\partial y}(x_i, y_i)\sin\theta = 0.
\end{aligned}$$

The angle θ represents direction that the r coordinate points which will depend on which boundary point we are considering. As before, (x_i, y_i) is at the point where this boundary condition is relevant.

For points A, E, H and J, θ is either $\frac{\pi}{4}$ or $\frac{5\pi}{4}$, which has the result that $\cos\theta = \sin\theta = \pm\frac{\sqrt{2}}{2}$. This then gives us the boundary condition:

$$\frac{\partial T}{\partial x}(x_i, y_i) + \frac{\partial T}{\partial y}(x_i, y_i) = 0 \tag{2}$$

in either case. Similarly, the corresponding boundary condition for B, F, G and I is:

$$\frac{\partial T}{\partial x}(x_i, y_i) - \frac{\partial T}{\partial y}(x_i, y_i) = 0. \tag{3}$$

There is an equivalent central difference approximation[1] to $\frac{\partial T}{\partial x} = 0$,

$$\frac{\partial T}{\partial x} = \frac{T_{i+1,j} - T_{i-1,j}}{2\Delta} = 0.$$

This enabled me to be able to derive the necessary boundary conditions appropriate for my program at each point in the grid by using the formula above and the formula provided for the central difference approximation to $\frac{\partial T}{\partial y} = 0$.

[1] `https://en.wikipedia.org/wiki/Finite_difference`

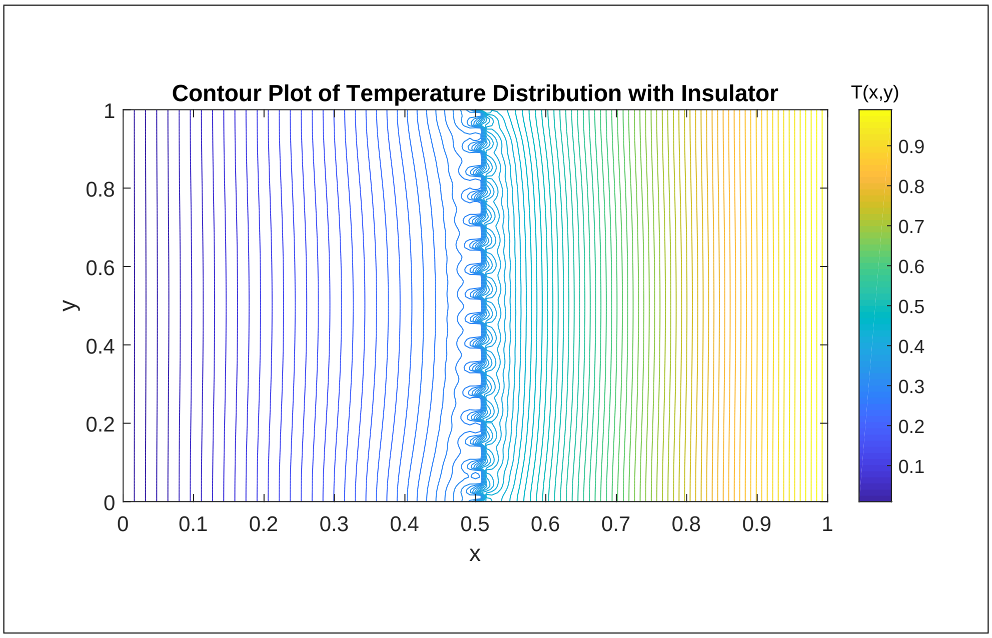

Figure 2: Contour plot of the steady state temperature distribution of $T(x, y)$ with an insulating strip in the middle of the box. In this case, $\epsilon = 4\Delta, \delta = 8\Delta, N_x = N_y = 128, \sigma = 1.9$.

Figure 2 shows the plot of the output of my program[2]. Convergence is obtained by comparing one previous iteration of T with the next iteration of T. The difference between these successive iterates gets smaller for convergence. As before, the program minimises the root mean square error to be less than 10^{-10} before terminating.

The affect on the temperature distribution due to the insulator is that heat is retained more closer to the right hand side of the box where the temperature is at a constant $T(x, y) = 1$. There is larger spacing between contours on the left hand side of the box which indicates a shallower decrease in temperature going towards the part the box held at a constant $T(x, y) = 0$. The holes allow for some heat to pass through the (perfect) insulator, and that is why there is still some gradient of heat on the LHS of the box.

(b) **Examine what happens to temperature along horizontal cross-sections through the temperature distribution by plotting $T(x, y_0)$ against x for a few values of y_0. Choose values of y_0 that are near the centre of the box and make sure you include cases that go through the centre of a hole and through the centre of an insulating section.**

After examining the different horizontal cross sections for different y_0 values that were kept constant, we can see a clear general trend. As the heat dissipates from the side of constant temperature = 1, and propagates across the box to the side of constant temperature = 0; the gradient of each graph is much more steeper before the heat reaches the insulator. It then gets even more steeper as it passes through the insulator and then becomes most shallow after leaving the insulator.

Figure 3 represents the value of y_0 passing through the centre of a hole. We can see from the graph the gradient change at the insulator is much more shallow compared

[2] After reviewing my project recently, I realise there is an error in my output. I would expect in fact to have a symmetrical distribution between the left and right sides of the box. This is because each side is held at a constant temperature, and so we can view warm air as diffusing across in the same way the cold air diffuses across. The steady state temperature distribution should hence be symmetric. I tried to amend my program to represent these results, however I have had no luck and have had to just keep my output as it is.

to the other cases in Figure 4 and Figure 5. This makes sense as we expect that heat will pass through with little resistance as it is passing through a gap in the insulator. In the other case, Figure 4, the heat passes through an edge of an insulating piece. This has a steeper gradient since it is passing along the perfect insulator. Figure 5 shows the example of heat passing straight onto a piece of the perfect insulator. This has the largest gradient at the insulator, since it is hitting a perfect insulator. There is a small point where the heat increases momentarily which suggests that heat can get trapped just behind the insulator. However more heat gets through the insulator and hence causes that slight increase when we consider one y_0 value.

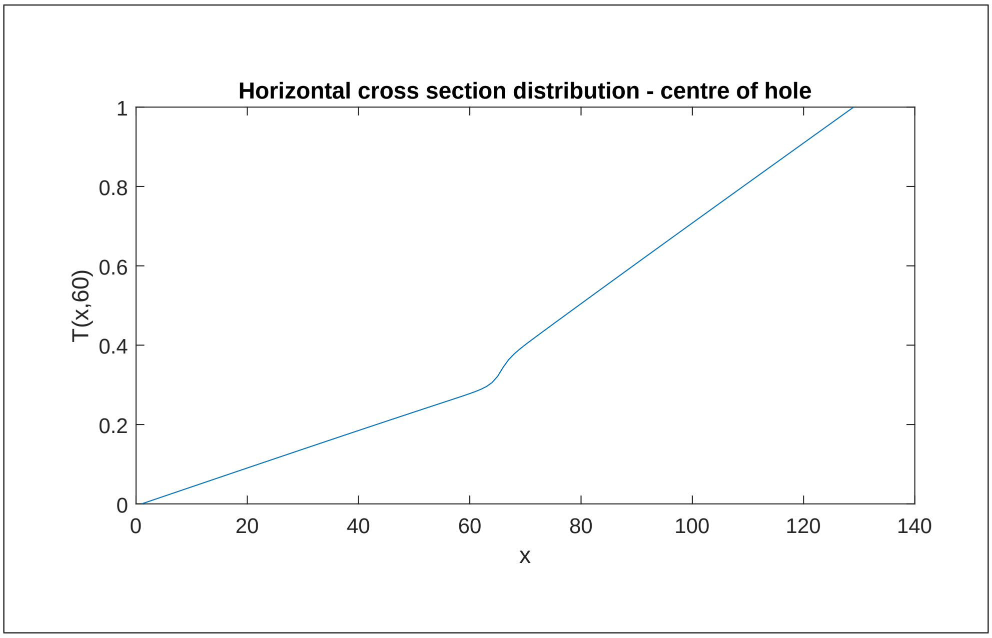

Figure 3: Horizontal cross section at $y_0 = \frac{60}{128}$ in the discretisation of the unit square. This is the case that goes through the centre of a hole in the insulating strip.

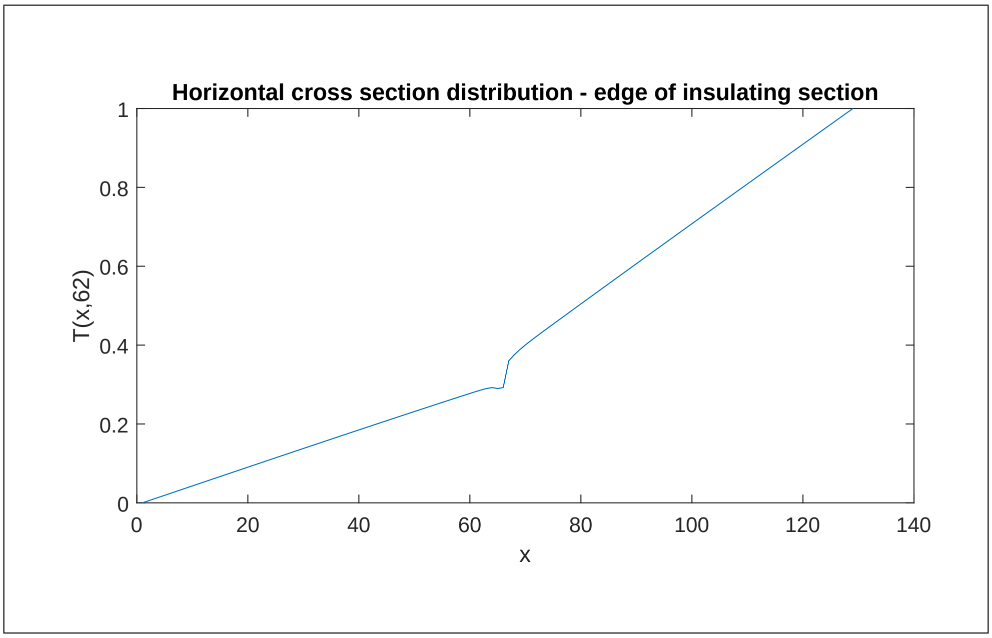

Figure 4: Horizontal cross section at $y_0 = \frac{62}{128}$ in the discretisation of the unit square. This is the case that goes through an edge of an insulating piece.

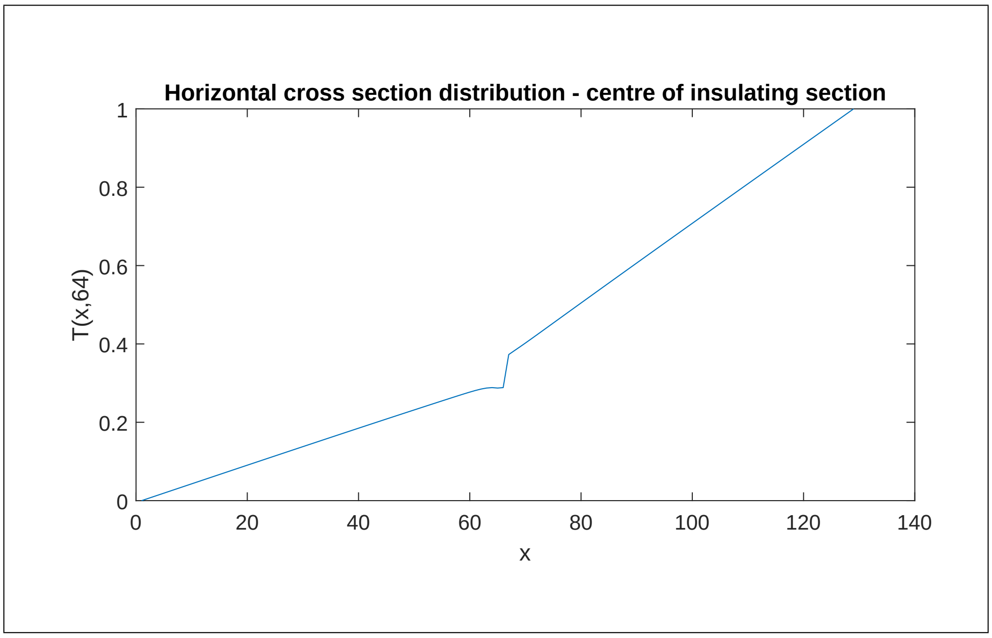

Figure 5: Horizontal cross section at $y_0 = \frac{64}{128}$ in the discretisation of the unit square. This is the case that goes through the centre of an insulating piece.

(c) **The total heat flux across the boundary $x = 1$ is a gauge of the quality of the insulating layer. Given that the heat flux at any point (x, y) is given by $-\kappa\nabla T(x, y)$, where $-\kappa$ is the thermal conductivity from equation (1), define a suitable measure Q of the insulator's quality. How would Q differ for a good insulator versus a good conductor? Comment on the insulating properties of the wall with $\epsilon = 4\Delta, \delta = 8\Delta$.**

A suitable measure Q of the insulator's quality will be the flux across the boundary near $x = 1$. We are only interested in the heat flow perpendicular to the boundary so we must consider $\frac{\partial T}{\partial x}$. For the discrete case of the distribution, I will define Q as follows:

$$Q = \frac{-\kappa}{N-1} \sum_{j=1}^{N-1} \left(\frac{T_{N,j} - T_{N-2,j}}{2\Delta} \right), \text{ where } \Delta = \frac{1}{N}.$$

This is the average heat flux perpendicular to the line $x = x_{N-1} = (N-1)\Delta$, equivalent to $-\kappa\frac{\partial T}{\partial x}$. Since κ is a constant depending on certain conditions about the thermal conductivity, in this project we can, WLOG, say that $\kappa = -1$. Hence our Q is just equal to $\frac{\partial T}{\partial x}$ in the discretised version as above. I have not included the endpoints in the analysis since it represnets the corner of the box and will not be representative of the flux across the boundary.

A good insulator will have a higher Q since the heat flux will be larger near the boundary that is at a constant temperature of 1. This is because the heat is contained in a smaller space due to the insulation and the other side of the box is still maintained at a constant temperature of 0. Examples of Q values for different insulation scenarios are:

$Q = 1.000612$ for the steady state solution with no inuslation strip.

$Q = 1.727693$ for the steady state solution with a complete insulation strip.

$Q = 1.279195$ for the steady state solution for the case of $\epsilon = 4\Delta, \delta = 8\Delta$.

All of these values were obtained using a $N = 128$. Discussing the case for $\epsilon = 4\Delta, \delta = 8\Delta$, we can see that the insulator is effective despite having several holes in it. It is closer to having a Q value of a complete insulating strip than no strip at all.

3. **Investigate what happens to the insulator quality Q when you vary ϵ and δ (but keep $N_x = N_y$ constant) according to**

$$\epsilon = k\delta^{\alpha} \tag{7}$$

where k and α are suitable real constants. In our discrete model, since ϵ and δ must be multiples of Δ, you would need to choose the nearest integer multiple of Δ for ϵ for a given δ or vice-versa. Include in your write-up a few plots that illustrate what happens to Q as you decrease δ: choose relationships that show interesting behaviour. Comment on your plots and on the physical significance of (7).

We know that $0 \leq \epsilon < 1$ and $0 < \delta < 1$. To make sure that $\epsilon \leq \delta$ and satisfies equation (7), we can set $\alpha > 1$, and $k < 1$. Here are some plots for different values of k and α that shows different behaviour of Q for decreasing δ.

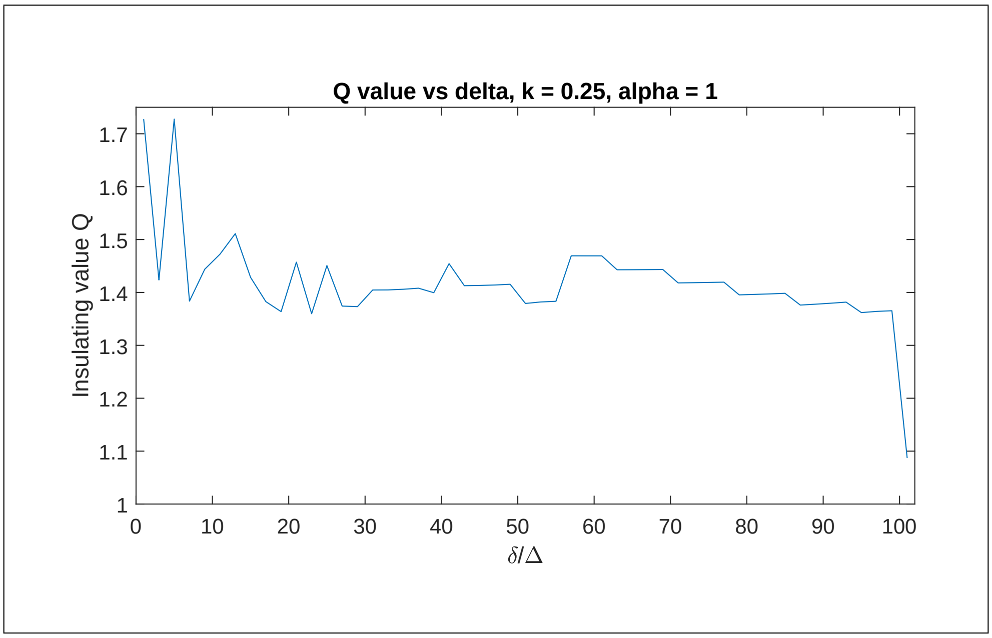

Figure 6: This is the case for which $\epsilon = \frac{\delta}{4}$

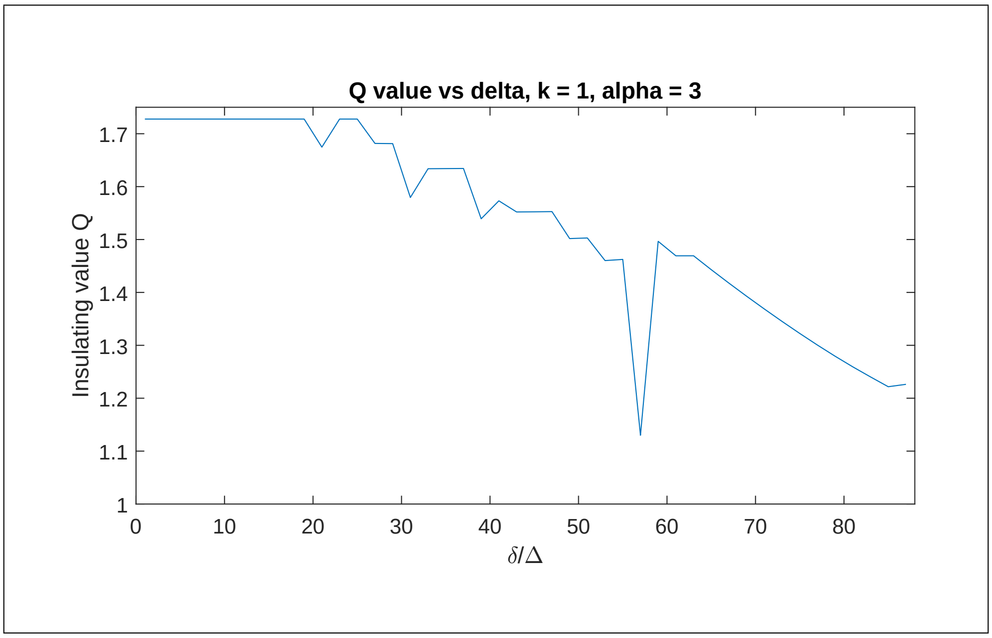

Figure 7: This is the case for which $\epsilon = \delta^3$

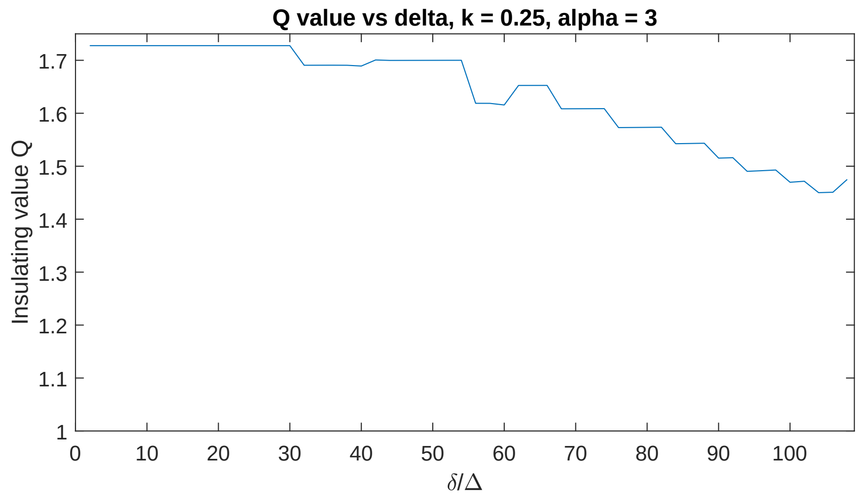

Figure 8: This is the case for which $\epsilon = \frac{\delta^3}{4}$

As we can see from the graphs, as epsilon gets much smaller than delta, the value of Q maintains a higher level even as delta is increased (evident from Figure 8). This implies a larger number of smaller holes within the insulator is much better for insulation. The physical significance of (7) is that it relates the hole size to the spacing between them in an exponential equation. This is useful since it is easier to analyse the effect of changing one variable according to the equation as opposed to having two variables to manipulate. We can then have one independent variable to test.

4. **Investigate what happens to Q when you vary ϵ and δ (but keep N_x constant) in this new model. Include in your write-up a couple of illustrative plots. Is there any need to change your definition of the quality Q? How does this periodic boundary condition model relate to the model in Question 3?**

   As ϵ and δ are varied, the Q value varies in a similar way to the first model. For example, increasing the hole size, ϵ, leads to a smaller Q value since more heat is allowed to pass through which means there is a smaller flux of heat near the boundary at $x = 1$. Increasing δ will increase the value of Q since the holes are more spaced out and so there is more insulation stopping heat from passing through.

   To remain consistent, I will use $N_x = 128$ with $\sigma = 1.9$. Below is a selection of the output, starting with $\epsilon = 4\Delta, \delta = 8\Delta$, with $\Delta = \Delta x = \frac{1}{N_x}$. We can see the for the cases shown, the value of Q increases as you decrease ϵ or increase δ. The definition of my Q value had to be changed slightly since we were only averaging over a smaller range, so it was important to take this into consideration. We also can set $\kappa = -1$ again, WLOG.

$$Q_{\text{New}} = \frac{1}{\delta - 1} \sum_{j=1}^{\delta-1} \left(\frac{T_{N_x,j} - T_{N_x-2,j}}{2\Delta} \right), \text{ where } \Delta = \frac{1}{N_x}.$$

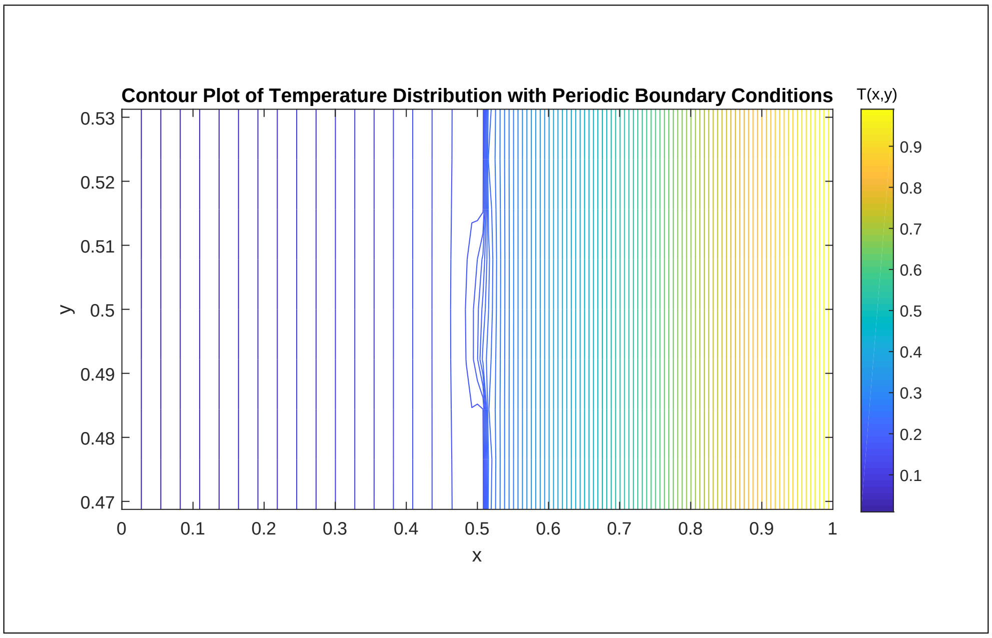

Figure 9: Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 1.587720$.

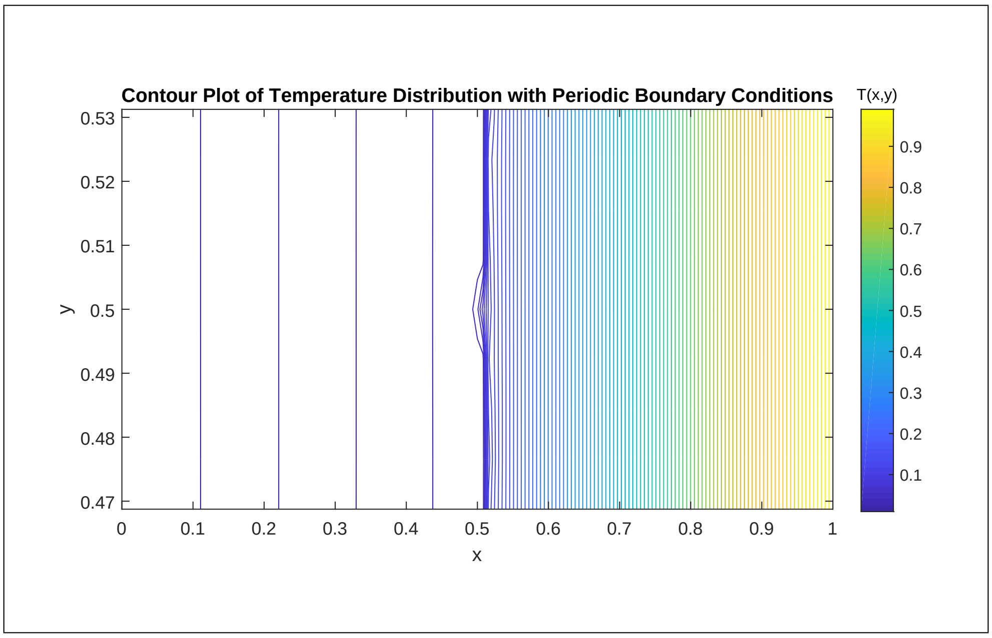

Figure 10: Temperature distribution for $\epsilon = 2, \delta = 8$, which has a value of $Q = 1.829834$

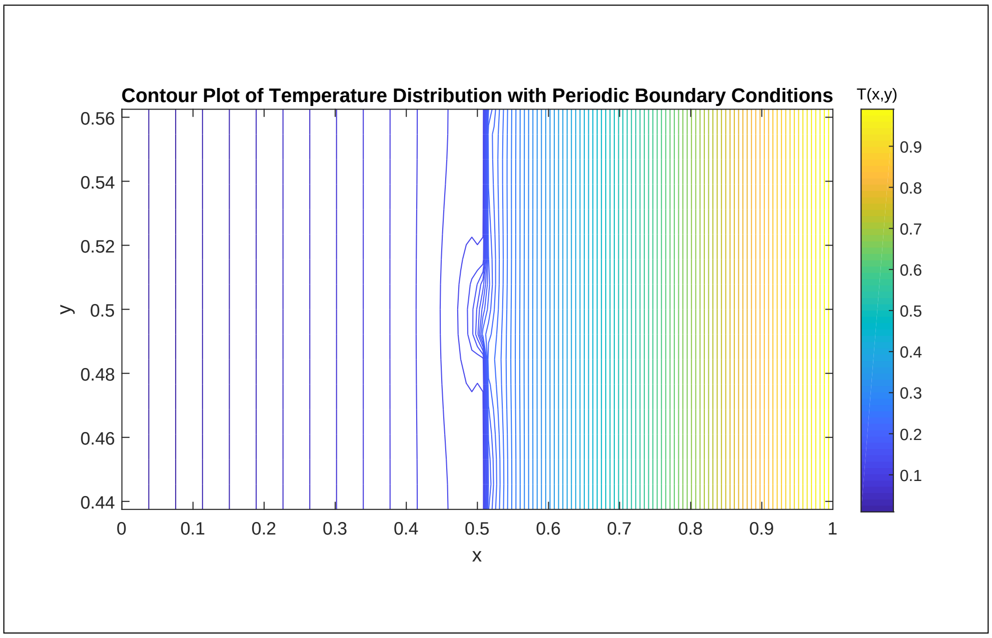

Figure 11: Temperature distribution for $\epsilon = 4, \delta = 16$, which has a value of $Q = 1.642438$.

This new model is equivalent to just considering a small section of the insulator in question 3. In the model in question 3, we have a regular repeating arrangement in the middle, hence it is appropriate to consider one hole, and applying periodic boundary conditions to achieve a similar analysis. However, the values of Q that we get are quite different since in the previous model, we have a boundary condition at the top which affects the flux across the boundary at $x = x_{N-1} = (N-1)\Delta$. However the relative values of the Q for different values of ϵ and δ in the new model are consistent with the first model.

5. **Try one of the following and comment on your results:**

   1. **Vary the width of the box without changing the wall thickness (i.e., vary N_x without changing Δx).**
   2. ~~**Vary the wall thickness for fixed ϵ, δ and N_x.**~~

   I will consider case 1, varying the width of the box without changing the wall thickness. Again to remain consistent, I will consider $N = N_x = 128$ and fix $\Delta = \Delta x = \frac{1}{128}$.

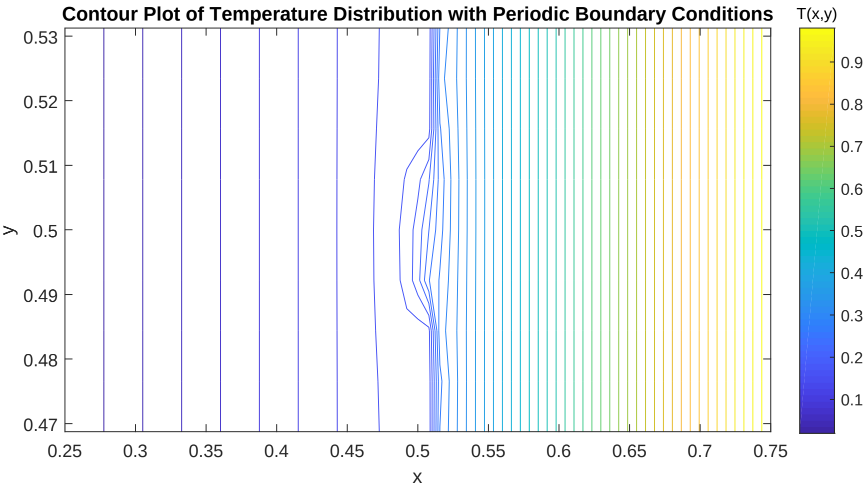

Figure 12: Shrinking N_x to $N_x = 64$ while keeping $\Delta = \frac{1}{128}$ constant. Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 3.095102$.

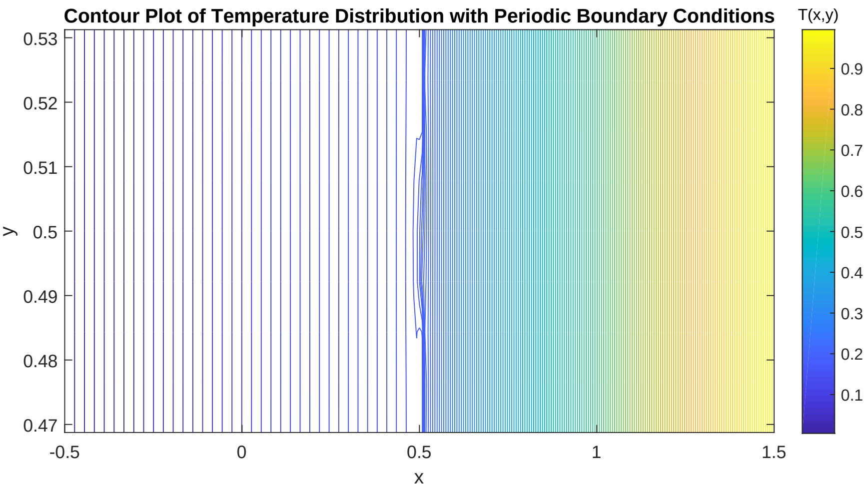

Figure 13: Increasing N_x to $N_x = 256$ while keeping $\Delta = \frac{1}{128}$ constant. Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 0.807903$.

I have chosen the number of contour lines to be equal to 100 within the range $[0, 1]$ so that it is visually consistent with all other images in this project. As we can see both values of Q are quite extreme, and this is due to the change in dimensions of the box, while keeping the insulator width constant. A smaller box, with effectively a thicker insulator relative to the size of the box, has a large Q value indicating that this is better for insulation. However a large box will result in a much lower Q value and hence the effectiveness of the insulator is not as evident since it may be too far away from the source of heat to have an affect.

6. **The original task was to investigate, for insulating sheets, the balance between the need to minimise the loss of heat and the need to include (non-insulating) holes to allow gases, especially water vapour, to pass. In the light of what you have learned from this model, what advice would you give a manufacturer of insulating sheets? What are the limitations of the model and what steps could be taken towards greater realism?**

   For a manufacturer, I would advise that it is a good idea to have small holes that are spaced out generously. This is based on the graph shown in Figure 8, where we can see that the Q value remains fairly high even as δ is increased to close to the length of the entire box. We can see that as ϵ increases as $\frac{\delta^2}{4}$, the value of Q stays above 1.4 and so the difference between ϵ and δ is the most important thing for maintaining a good insulator that also allows gasses and water vapour to pass through.

   The limitations of this project lie in the fact that we have modeled the space in two dimensions. A manufacturer would need to consider a three dimensional model to have a better understanding of the temperature distribution in any standard room that they would be looking to insulate. We have also not taken into account objects that may be inside a room as, well as assuming that the walls above and below (and the insulator itself) are perfectly insulating, when in reality they are not. There may also be more than one heat source in a room, or heat sink, which may be necessary to include in the model.

```
function [T, Iteration, Xlength, Ylength] = RelaxationMethod(N, sigma)
%This function solves equation (2) in the project 7pt6 using a relaxation
%method
A = linspace(0,1,N+1);
TActual = repmat(A',1,N+1);
T = zeros(N+1,N+1);
Iteration = 0;
TOld = ones(size(T));
while (sum(sum(TOld-T).^2)/(N+1)^2) > 10^-10     %%Root mean square error
    TOld = T;
    T(1,:) = 0;
    T(N+1,:) = 1;
        for i = 2:N;
            for j = 2:N;
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,j-1));
            end
            for j = 1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j+1));
            end
            for j = N+1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j-1));
            end
        end
        Iteration = Iteration + 1;
end
contour(A,A,T',100);
title('Contour Plot of the Steady State Temperature Distribution')
xlabel('x')
ylabel('y')
title(colorbar,'T(x,y)')
Xlength = N;
Ylength = N;
end
```

```
function [T, Iteration, Xlength, N_y] = RelaxationMethodWithInsulatory(N, sigma, epsilon, delta)
%This function solves equation (2) in the project 7pt6 using a relaxation
%method
A = linspace(0,1,N+1);
T = zeros(N+1,N+1);
Iteration = 0; %counts iterations

%%%%%%%%%%%%%%%%%%

% 0 is No BC
% 1 is dT/dx = 0
% 2 is dT/dx + dT/dy = 0
% 3 is dT/dx - dT/dy = 0
% 4 is dT/dx = dT/dy = 0

endlength = 0.5*(mod(N-epsilon,delta)); %calculates the length of the insulators at the end

if endlength == 0
    endlength = delta/2; %This deals with the case where there is no end boundary, it creates one
end

M = (N - epsilon - 2*endlength)/delta; %number of insulators in the middle of a common length

endlength=endlength+1; %Because we are working with the number of points it hits on the grid

if mod(2*endlength,2)==1
    endlengthoriginal = endlength; %stores
    endlength = floor(endlength);
else
    endlengthoriginal = endlength; %This makes the endlength an integer number
end

length = delta - epsilon; %length of one piece of insulator in wall between adjacent holes

%Defines the endlength, M the number of full lengths of insulation in the
%middle, and also the length of a piece of insulation in the middle

Boundary1 = [0];
for i=1:endlength
    if i == 1
        Boundary1(i) = 4;
    elseif i == endlength
        Boundary1(i) = 2;
    else
        Boundary1(i) = 1;
    end
end
if i~=1
    if mod(2*endlengthoriginal,2)==1
            Boundary1(i) = 1;
    end
else
end
if  mod(2*endlengthoriginal,2)==1
```

```
    for i=endlength+1:endlength+epsilon
        Boundary1(i)=0;
    end
else
    for i=endlength+1:endlength+(epsilon-1)
        Boundary1(i)=0;
    end
end

%T defines the boundary values along the piece of insualator, and this
%first bit sets up the boundary values up to the end of the first endpiece
%and then one epsilon length

if mod(2*endlengthoriginal,2)==1
    for j = 0:M-1
        for i = j*(delta)+endlength+epsilon+1:j*(delta)+endlength+epsilon+length
            Boundary1(i) = 1;
        end
        for i = j*(delta)+endlength+epsilon+length+1:j*(delta)+endlength+epsilon+length+epsilon
            Boundary1(i)=0;
        end
    end
else
    for j = 0:M-1
        if epsilon ~= delta
            for i=j*(delta)+endlength+(epsilon-1)+1
                Boundary1(i)=3;
            end
        else
            for i=j*(delta)+endlength+(epsilon-1)+1
                Boundary1(i)=0;
            end
        end
        for i=j*(delta)+endlength+(epsilon-1)+1+1:j*(delta)+endlength+(epsilon-1)+1+length
            if i == j*(delta)+endlength+(epsilon-1)+1+length
                    Boundary1(i)=2;
            else
                Boundary1(i)=1;
            end
        end
        for i=j*(delta)+endlength+(epsilon-1)+1+length+1:j*(delta)+endlength+(epsilon-1)+1+length+(epsilon-1)
            Boundary1(i) = 0;
        end
    end
end

%Defines all the middle insulation lengths and the boundary condiitons

if mod(2*endlengthoriginal,2)==1
    for i = (M-1)*(delta)+endlength+epsilon+length+epsilon+1:(M-1)*(delta)+endlength+epsilon+length+epsilon+endlength-1
        Boundary1(i) = 1;
    end
    for i = (M-1)*(delta)+endlength+epsilon+length+epsilon+endlength
        Boundary1(i) = 4;
    end
else
```

```
    for i=(M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1
        Boundary1(i)=3;
    end
    for i=(M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1+1:(M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1+endlength-1
        if i == (M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1+endlength-1
            Boundary1(i) = 4;
        else
            Boundary1(i)=1;
        end
    end
end
%Defines the last endlength boundary conditions

Boundary2=[0];

for i=1:size(Boundary1,2)
    if Boundary1(i) == 2
        Boundary2(i) = 3;
    elseif Boundary1(i) == 3
        Boundary2(i) = 2;
    else
        Boundary2(i) = Boundary1(i);
    end
end

%%%%%%%%%%%%%

BoundaryInsulator = [[fliplr(Boundary1)];[fliplr(Boundary2)]];
if epsilon == 0
      Temp = [4,repmat(1,1,N-1),4];
      BoundaryInsulator = repmat(Temp,2,1);
end

%Boundary conditions on the insulator

if mod(N,2) == 1
    centre_beginning = (N+1)/2; %Perfect symmetry
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
else
    centre_beginning = (N)/2; %NO perfect symmetry so wlog it is slightly to the left of the true centre
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
    BoundaryInsulator = padarray(BoundaryInsulator,1,0,'post');
end

%%%%%%%%%%

%%%Running the iteration algorithm
TOld = ones(size(T));
while (sum(sum(TOld-T).^2)/(N+1)^2) > 10^-10 %Root mean square dependent on the previous iteration of the program
    TOld = T;
    T(1,:) = 0; %Boundary condition for x=0 side
    T(N+1,:) = 1; %Boundary condition for x=N side
        for i = 2:N
            for j = 2:N
```

```
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,j-1));
            end
            for j = 1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j+1));
                %Boundary condition for dT/dy = 0 at y = 0
            end
            for j = N+1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j-1));
                %Boundary condition for dT/dy = 0 at y = N
            end
        end
        if epsilon ~= 0
        for i = centre_beginning:centre_end
            for j = 1:N+1
                if BoundaryInsulator(i,j)==1
                    %%%%%%%%%%%%%%%
                    T(i+1,j)=T(i-1,j);
                elseif BoundaryInsulator(i,j)==2
                    T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1));
                elseif BoundaryInsulator(i,j)==3
                    T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
                elseif BoundaryInsulator(i,j)==4
                    if i == centre_beginning
                        for j = 1
                            T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)+2*T(i,j+1));
                            %Boundary 2
                        end
                        for j = N+1
                            T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)+2*T(i,j-1));
                            %Boundary 3
                        end
                    else
                        for j = 1
                            T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i-1,j)+2*T(i,j+1));
                            %Boundary 3
                        end
                        for j = N+1
                            T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i,j-1)+2*T(i-1,j));
                            %Boundary 2
                        end
                    end
                elseif BoundaryInsulator(i,j)==0
                else
                    disp('Error with insulator boundary')
                end
            end
        end
        else
           %If the insulator is completely along the box with no
           %holes?
           %
           %%%
            for i = centre_beginning:centre_end
```

```
                for j = 1:N+1
                    if BoundaryInsulator(i,j)==1
                        T(i+1,j)=T(i-1,j);
                    elseif BoundaryInsulator(i,j)==4
                        if i == centre_beginning
                            for j = 1
                                T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)+2*T(i,j+1));
                                %Boundary 2
                            end
                            for j = N+1
                                T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)+2*T(i,j-1));
                                %Boundary 3
                            end
                        else
                            for j = 1
                                T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i-1,j)+2*T(i,j+1));
                                %Boundary 3
                            end
                            for j = N+1
                                T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i,j-1)+2*T(i-1,j));
                                %Boundary 2
                            end
                        end
                    end
                end
            end
        end
    Iteration = Iteration + 1;
end
contour(A,A,T',100);
title('Contour Plot of Temperature Distribution with Insulator')
xlabel('x')
ylabel('y')
title(colorbar,'T(x,y)')
N_y = N;
Xlength=N;
end
```

```
[T, Iteration] = RelaxationMethodWithInsulator(128,1.9,4,8);
x=1:129;
figure(2)
plot(x,T(x,61)) %Through centre of hole
title('Horizontal cross section distribution - centre of hole')
xlabel('x')
ylabel('T(x,60)')
figure(3)
plot(x,T(x,65)) %Through centre of insulating section
title('Horizontal cross section distribution - centre of insulating section')
xlabel('x')
ylabel('T(x,64)')
figure(4)
plot(x,T(x,63)) %edge of insulating section
title('Horizontal cross section distribution - edge of insulating section')
xlabel('x')
ylabel('T(x,62)')
figure(5)
plot(x,T(x,2))  %one extreme
title('Horizontal cross section distribution - near bottom of insulating section
    ')
xlabel('x')
ylabel('T(x,1)')
figure(6)
plot(x,T(x,128)) %other extreme
title('Horizontal cross section distribution - near top of insulating section')
xlabel('x')
ylabel('T(x,127)')
```

```
%This script defines the Q for which we measure the flux as close to the
%x=1 boundary.
clear QMatrix Q
N_x=Xlength;
N_y=Ylength;
for j=2:N_y
QMatrix(j-1) = (T(N_x+1,j)-T(N_x-1,j))/(2*(1/N_x));
end
Q = sum(QMatrix)/(N_y-1)
```

```
clear
N = 128;
k = 0.25;
alpha = 1;
delta = (floor(N-sqrt(N)))*(1/N);
epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
while delta*N+epsilon*N+1>N
    delta = delta - (1/N);
    epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
end
xlimit = delta*N+1;
count = 1;

    while delta>=epsilon
        [T, Iteration, Xlength, Ylength] = RelaxationMethodWithInsulator(N, 1.9,
            N*epsilon, N*delta);
        CalculatingQfromT;
        Qvector(count) = Q;
        Deltavector(count) = delta*N;
        Epsilonvector(count) = epsilon*N;
        count = count + 1;
        delta = delta - 2*(1/N);
        epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
            if delta <= 0
                break
            end
    end
figure(2)
plot(Deltavector, Qvector)
title(sprintf('Q value vs delta, k = %1g, alpha = %1g',k,alpha))
xlabel('\delta/\Delta')
ylabel('Insulating value Q')
ylim([1, 1.75])
xlim([0, xlimit])
```

```
function [T, Iteration, Xlength, N_y] = RelaxationMethodPeriodicQ4(N_x, sigma, epsilon, delta)
%This function solves equation (2) in the project 7pt6 using a relaxation
%method
Ax = linspace(0,1,N_x+1);
T = zeros(N_x+1,delta+1);
Iteration = 0; %counts iterations

%%%%%%%%%%%%%%%%%%

% 0 is No BC
% 1 is dT/dx = 0            FOR ZERO GRADIENT
% 2 is dT/dx + dT/dy = 0    FOR POSITIVE GRADIENT
% 3 is dT/dx - dT/dy = 0    FOR NEGATIVE GRADIENT

Boundary1 = [0];
for i=1:delta+1
    if mod(delta,2)==0 %delta even
        if mod(epsilon,2)==0 %epsilon even
            if i < (delta/2)+1-(epsilon/2)
                Boundary1(i) = 1;
            elseif i == (delta/2)+1-(epsilon/2)
                Boundary1(i) = 2;
            elseif i > (delta/2)+1-(epsilon/2) && i < (delta/2)+1+(epsilon/2)
                Boundary1(i) = 0;
            elseif i == (delta/2)+1+(epsilon/2)
                Boundary1(i) = 3;
            else
                Boundary1(i) = 1;
            end
        else %epsilon odd
            if i < (delta/2)+1-((epsilon+1)/2)
                Boundary1(i) = 1;
            elseif i > (delta/2)+1-((epsilon+1)/2) && i < (delta/2)+1+((epsilon+1)/2)
                Boundary1(i) = 0;
            else
                Boundary1(i) = 1;
            end
        end
    else %delta odd
        if mod(epsilon,2)==1 %epsilon odd
            if i < ((delta+1)/2)-((epsilon-1)/2)
                Boundary1(i) = 1;
            elseif i == ((delta+1)/2)-((epsilon-1)/2)
                Boundary1(i) = 2;
            elseif i > ((delta+1)/2)-((epsilon-1)/2) && i < ((delta+1)/2)+1+((epsilon-1)/2)
                Boundary1(i) = 0;
            elseif i == ((delta+1)/2)+1+((epsilon-1)/2)
                Boundary1(i) = 3;
            else
                Boundary1(i) = 1;
            end
        else %epsilon even
            if i <= ((delta+1)/2)-(epsilon/2)
                Boundary1(i) = 1;
            elseif i > ((delta+1)/2)-(epsilon/2) && i < ((delta+1)/2)+1+(epsilon/2)
                Boundary1(i) = 0;
```

```
            else
                Boundary1(i) = 1;
            end
        end
    end

end


%Above defines the strip of length delta that we will apply periodic
%boundary conditions to.


%Defines all the middle insulation lengths and the boundary condiitons

Boundary2=[0];

for i=1:size(Boundary1,2)
    if Boundary1(i) == 2
        Boundary2(i) = 3;
    elseif Boundary1(i) == 3
        Boundary2(i) = 2;
    else
        Boundary2(i) = Boundary1(i);
    end
end

%%%%%%%%%%%%%%

BoundaryInsulator = [[fliplr(Boundary1)];[fliplr(Boundary2)]];
%Defines the matrix that represents the boundary conditions
if epsilon == 0
      Temp = [repmat(1,1,delta+1)];
      BoundaryInsulator = repmat(Temp,2,1);
end


%Boundary conditions on the insulator are set in the matrix
%BoundaryInsulator

if mod(N_x,2) == 1
    centre_beginning = (N_x+1)/2; %Perfect symmetry
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
else
    centre_beginning = (N_x)/2; %NO perfect symmetry so wlog it is slightly to the left of the true centre
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
    BoundaryInsulator = padarray(BoundaryInsulator,1,0,'post');
end

%%%%%%%%%%%

%%%Running the iteration algorithm
TOld = ones(size(T));
while (sum(sum(TOld-T).^2)/(N_x+1)^2) > 10^-10   %Root mean square dependent on the previous iteration of the program
    TOld = T;
    T(1,:) = 0; %Boundary condition for x=0 side
```

```
    T(N_x+1,:) = 1; %Boundary condition for x=N side
        for i = 2:N_x
            for j = 2:delta-1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,j-1));
            end
            for j = 1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,delta));
                %Boundary condition for dT/dy = 0 at y = 0
            end
            for j = delta
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,1)+T(i,delta-1));
                %Periodic Boundary COndition at j=delta
            end
            for j = delta+1
                T(i,j) = T(i,1);
                %Periodic Boundary Condition for j=delta+1 and j=1
            end
        end
        if epsilon ~= 0 && epsilon ~= delta
        for i = centre_beginning:centre_end
            for j = 1:delta+1
                if BoundaryInsulator(i,j)==1
                    %%%%%%%%%%%%%%
                    T(i+1,j)=T(i-1,j); %No flux in the x direction (dT/dx = 0)
                elseif BoundaryInsulator(i,j)==2
                    T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1)); %dT/dx + dT/dy = 0
                elseif BoundaryInsulator(i,j)==3
                    T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
                elseif BoundaryInsulator(i,j)==0
                else
                     disp('Error with insulator boundary')
                end
            end
        end
        elseif epsilon == delta

        elseif epsilon == 0
%              If the insulator is completely along the box with no
%              holes?
%
%             %
            for i = centre_beginning:centre_end
                for j = 1:delta+1
                    if BoundaryInsulator(i,j)==1
                        T(i+1,j)=T(i-1,j);
                    end
                end
            end
        else
            disp('Error with epsilon and delta')
            return
        end
     Iteration = Iteration + 1;
end
if mod(delta,2)==0
    if mod(N_x,2)==0
        Ay = linspace((((N_x/2)-(delta/2)))/N_x,((N_x/2)+(delta/2))/N_x,delta+1)
```

```
;
    else
        Ay = linspace((((N_x-1)/2)-(delta/2))/N_x,(((N_x-1)/2)+(delta/2))/N_x,delta+1);
    end
else
    if mod(N_x,2)==0
        Ay = linspace(((N_x/2)-(delta/2))/N_x,((N_x/2)+(delta/2))/N_x,delta+1);
    else
        Ay = linspace(((N_x/2)-(delta/2))/N_x,((N_x/2)+(delta/2))/N_x,delta+1);
    end
end
contour(Ax,Ay,T',100);
title('Contour Plot of Temperature Distribution with Periodic Boundary Conditions')
xlabel('x')
ylabel('y')
title(colorbar,'T(x,y)')
N_y=delta;
Xlength = N_x;
end
```

```matlab
function [T, Iteration, Xlength, N_y] = RelaxationMethodPeriodicQ5(N_x, sigma, epsilon, delta)
%This function solves equation (2) in the project 7pt6 using a relaxation
%method
N=128;
T = zeros((N_x)+1,delta+1);
Iteration = 0; %counts iterations

%%%%%%%%%%%%%%%%%%

% 0 is No BC
% 1 is dT/dx = 0            FOR ZERO GRADIENT
% 2 is dT/dx + dT/dy = 0    FOR POSITIVE GRADIENT
% 3 is dT/dx - dT/dy = 0    FOR NEGATIVE GRADIENT


length = delta - epsilon; %length of one piece of insulator in wall between adjacent holes

Boundary1 = [0];
for i=1:delta+1
    if mod(delta,2)==0 %delta even
        if mod(epsilon,2)==0 %epsilon even
            if i < (delta/2)+1-(epsilon/2)
                Boundary1(i) = 1;
            elseif i == (delta/2)+1-(epsilon/2)
                Boundary1(i) = 2;
            elseif i > (delta/2)+1-(epsilon/2) && i < (delta/2)+1+(epsilon/2)
                Boundary1(i) = 0;
            elseif i == (delta/2)+1+(epsilon/2)
                Boundary1(i) = 3;
            else
                Boundary1(i) = 1;
            end
        else %epsilon odd
            if i < (delta/2)+1-((epsilon+1)/2)
                Boundary1(i) = 1;
            elseif i > (delta/2)+1-((epsilon+1)/2) && i < (delta/2)+1+((epsilon+1)/2)
                Boundary1(i) = 0;
            else
                Boundary1(i) = 1;
            end
        end
    else %delta odd
        if mod(epsilon,2)==1 %epsilon odd
            if i < ((delta+1)/2)-((epsilon-1)/2)
                Boundary1(i) = 1;
            elseif i == ((delta+1)/2)-((epsilon-1)/2)
                Boundary1(i) = 2;
            elseif i > ((delta+1)/2)-((epsilon-1)/2) && i < ((delta+1)/2)+1+((epsilon-1)/2)
                Boundary1(i) = 0;
            elseif i == ((delta+1)/2)+1+((epsilon-1)/2)
                Boundary1(i) = 3;
            else
                Boundary1(i) = 1;
            end
        else %epsilon even
            if i <= ((delta+1)/2)-(epsilon/2)
```

```
                Boundary1(i) = 1;
            elseif i > ((delta+1)/2)-(epsilon/2) && i < ((delta+1)/2)+1+(epsilon/2)
                Boundary1(i) = 0;
            else
                Boundary1(i) = 1;
            end
        end
    end

end


%Above defines the strip of length delta that we will apply periodic
%boundary conditions to.


%Defines all the middle insulation lengths and the boundary condiitons

Boundary2=[0];

for i=1:size(Boundary1,2)
    if Boundary1(i) == 2
        Boundary2(i) = 3;
    elseif Boundary1(i) == 3
        Boundary2(i) = 2;
    else
        Boundary2(i) = Boundary1(i);
    end
end

%%%%%%%%%%%%%%%%

BoundaryInsulator = [[fliplr(Boundary1)];[fliplr(Boundary2)]];
%Defines the matrix that represents the boundary conditions
if epsilon == 0
      Temp = [repmat(1,1,delta+1)];
      BoundaryInsulator = repmat(Temp,2,1);
end


%Boundary conditions on the insulator are set in the matrix
%BoundaryInsulator

if mod(N,2) == 1
    centre_beginning = (N+1)/2; %Perfect symmetry
    centre_end = centre_beginning+1;
else
    centre_beginning = (N)/2; %NO perfect symmetry so wlog it is slightly to the left of the true centre
    centre_end = centre_beginning+1;
end

if mod(N_x,2)==0
    x_1 = (centre_beginning*(1/N))-(((N_x)/2)*(1/N));
    x_2 = (centre_beginning*(1/N))+((N_x/2)*(1/N));
else
    x_1 = (centre_beginning*(1/(N)))-(((N_x+1)/2)*(1/(N)));
    x_2 = (centre_beginning*(1/(N)))+(((N_x+1)/2)*(1/(N)));
end
```

```

Ax = linspace(x_1,x_2,(N_x)+1);

if mod(N_x,2) == 1
    centre_beginning = (N_x+1)/2; %Perfect symmetry
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
else
    centre_beginning = (N_x)/2; %NO perfect symmetry so wlog it is slightly to the left of the true centre
    centre_end = centre_beginning+1;
    BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
    BoundaryInsulator = padarray(BoundaryInsulator,1,0,'post');
end

%%%%%%%%%%%%

%%%Running the iteration algorithm

TOld = ones(size(T));
while (sum(sum(TOld-T).^2)/(N_x+1)^2) > 10^-10   %Root mean square dependent on the previous iteration of the program
    TOld = T;
    T(1,:) = 0; %Boundary condition for x=0 side
    T(N_x+1,:) = 1; %Boundary condition for x=N side
        for i = 2:N_x
            for j = 2:delta-1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,j-1));
            end
            for j = 1
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j+1)+T(i,delta));
                %Boundary condition for dT/dy = 0 at y = 0
            end
            for j = delta
                T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,1)+T(i,delta-1));
                %Periodic Boundary COndition at j=delta
            end
            for j = delta+1
                T(i,j) = T(i,1);
                %Periodic Boundary Condition for j=delta+1 and j=1
            end
        end
        if epsilon ~= 0 && epsilon ~= delta
        for i = centre_beginning:centre_end
            for j = 1:delta+1
                if BoundaryInsulator(i,j)==1
                    %%%%%%%%%%%%%%%%
                    T(i+1,j)=T(i-1,j); %No flux in the x direction (dT/dx = 0)
                elseif BoundaryInsulator(i,j)==2
                    T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1)); %dT/dx + dT/dy = 0
                elseif BoundaryInsulator(i,j)==3
                    T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
                elseif BoundaryInsulator(i,j)==0
                else
                    disp('Error with insulator boundary')
                end
            end
```

```
        end
        elseif epsilon == delta

        elseif epslion == 0
%              If the insulator is completely along the box with no
%              holes?
%
%              %
            for i = centre_beginning:centre_end
                for j = 1:delta+1
                    if BoundaryInsulator(i,j)==1
                        T(i+1,j)=T(i-1,j);
                    end
                end
            end
        else
            disp('Error with epsilon and delta')
            return
        end
     Iteration = Iteration + 1;
end
if mod(delta,2)==0
    if mod(N,2)==0
        Ay = linspace((((N/2)-(delta/2)))/N,((N/2)+(delta/2))/N,delta+1);
    else
        Ay = linspace((((N-1)/2)-(delta/2))/N,(((N-1)/2)+(delta/2))/N,delta+1);
    end
else
    if mod(N,2)==0
        Ay = linspace(((N/2)-(delta/2))/N,((N/2)+(delta/2))/N,delta+1);
    else
        Ay = linspace(((N/2)-(delta/2))/N,((N/2)+(delta/2))/N,delta+1);
    end
end
contour(Ax,Ay,T',abs(x_1-x_2)*100);
%normalise the number of contour lines so that between 0 and 1 there is alwyas 100
title('Contour Plot of Temperature Distribution with Periodic Boundary Conditions')
xlabel('x')
ylabel('y')
title(colorbar,'T(x,y)')
N_y=delta;
Xlength = N_x;

clear QMatrix Q
for j=2:N_y
QMatrix(j-1) = (T(N_x+1,j)-T(N_x-1,j))/(2*(1/N));
end
Q = sum(QMatrix)/(N_y-1)
end
```