

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:

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

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 successive 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 rapidly. 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.

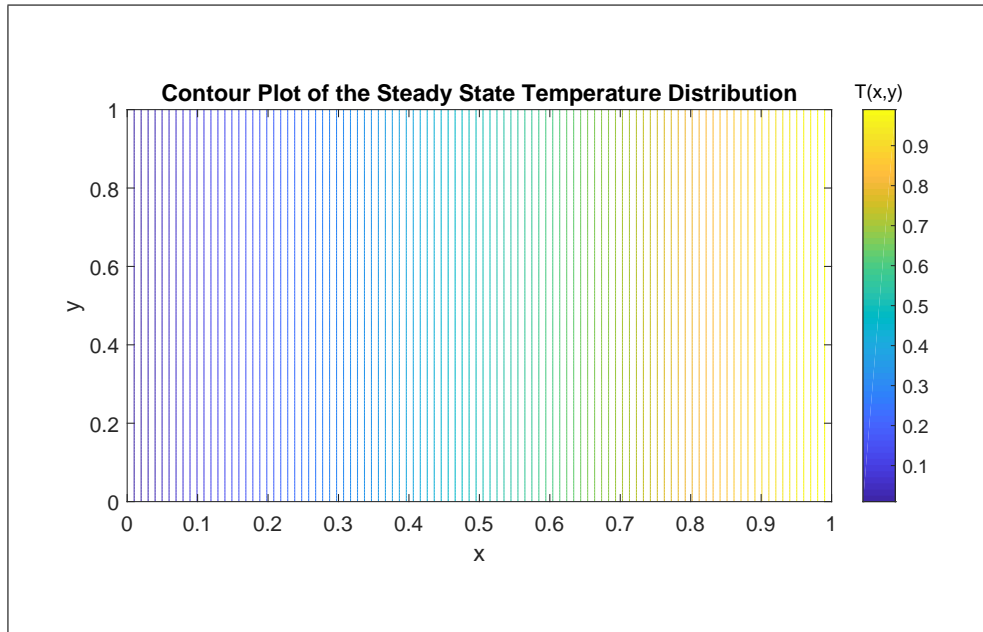


Figure 1: Contour plot of the steady state temperature distribution of $T(x, y)$ for the initial given boundary conditions.

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:

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

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:

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

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:

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

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

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

There is an equivalent central difference approximation¹ 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 $\frac{\partial T}{\partial y} = 0$.

¹https://en.wikipedia.org/wiki/Finite_difference

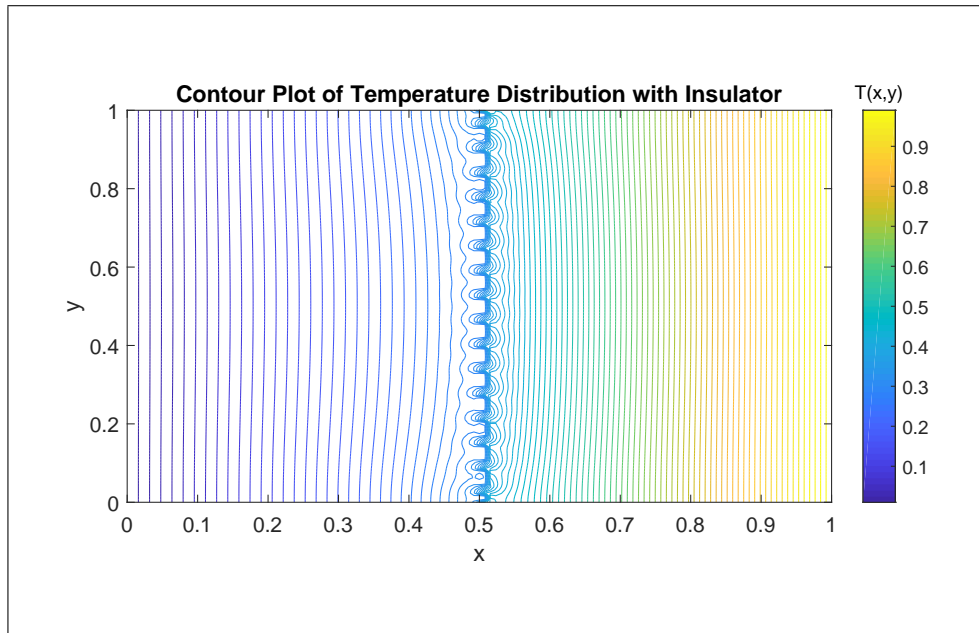


Figure 2: Contour plot of the steady state temperature distribution of $T(x, y)$ with an insulating strip in the middle of the box. In this case, $\epsilon = 4\Delta$, $\delta = 8\Delta$, $N_x = N_y = 128$, $\sigma = 1.9$.

Figure 2 shows the plot of the output of my program². 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

²After 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.

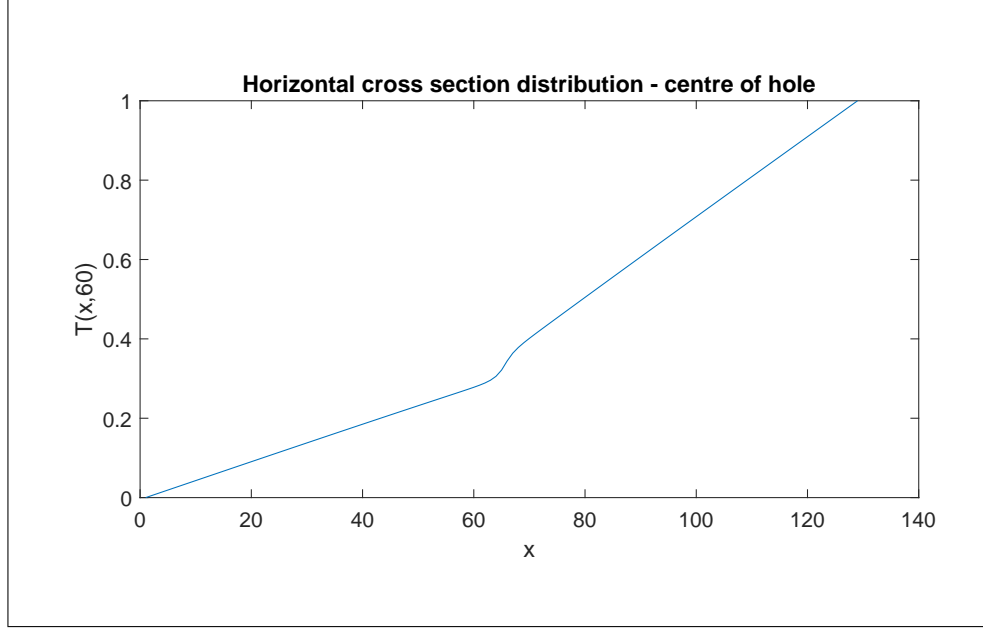


Figure 3: Horizontal cross section at $y_0 = \frac{60}{128}$ in the discretisation of the unit square. This is the case that goes through the centre of a hole in the insulating strip.

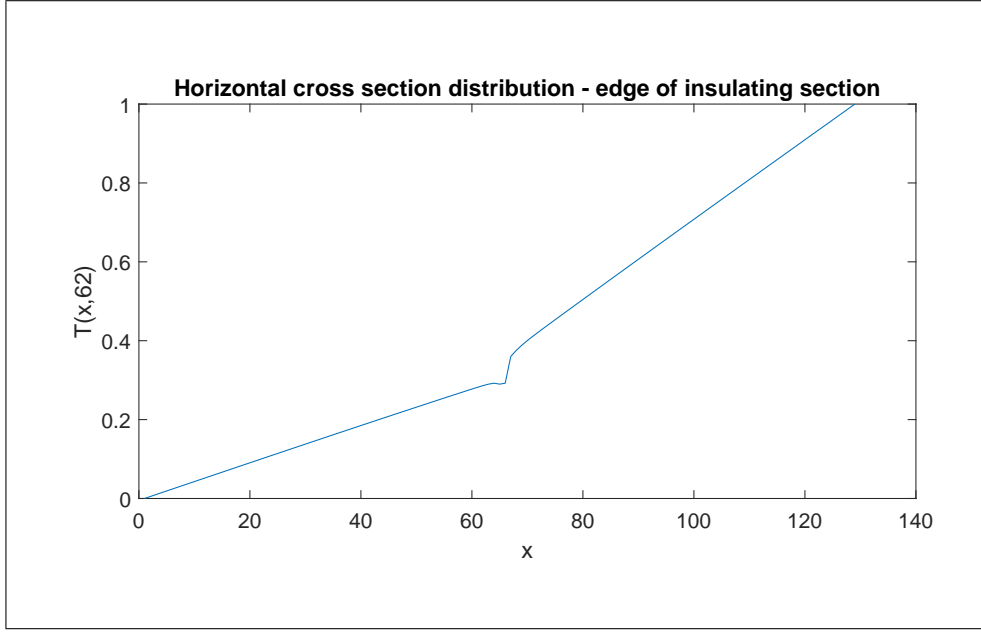


Figure 4: Horizontal cross section at $y_0 = \frac{62}{128}$ in the discretisation of the unit square. This is the case that goes through an edge of an insulating piece.

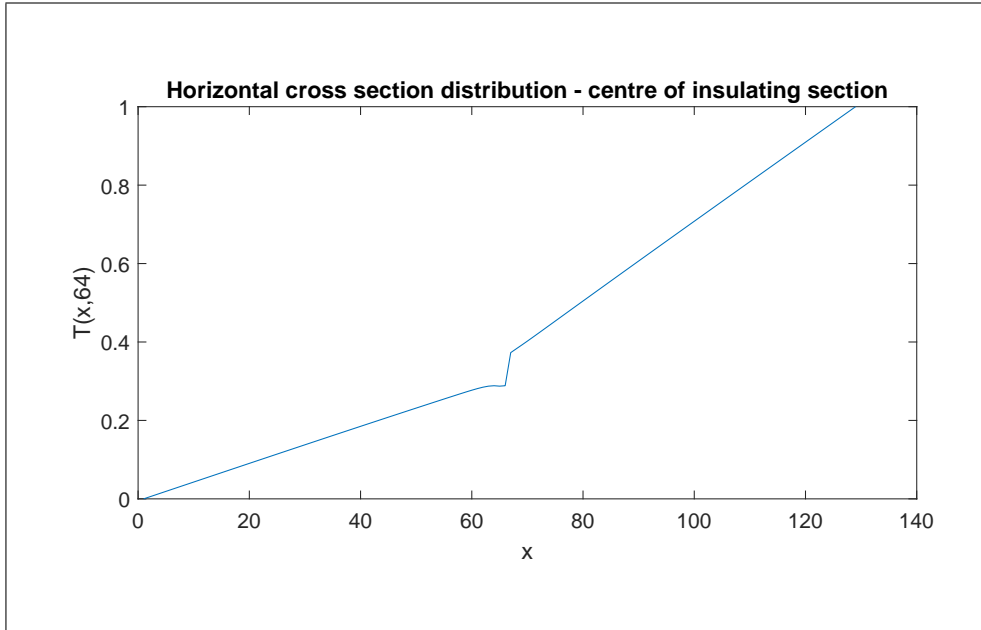


Figure 5: Horizontal cross section at $y_0 = \frac{64}{128}$ in the discretisation of the unit square. This is the case that goes through the centre of an insulating piece.

- (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 represents 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 insulation 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 .

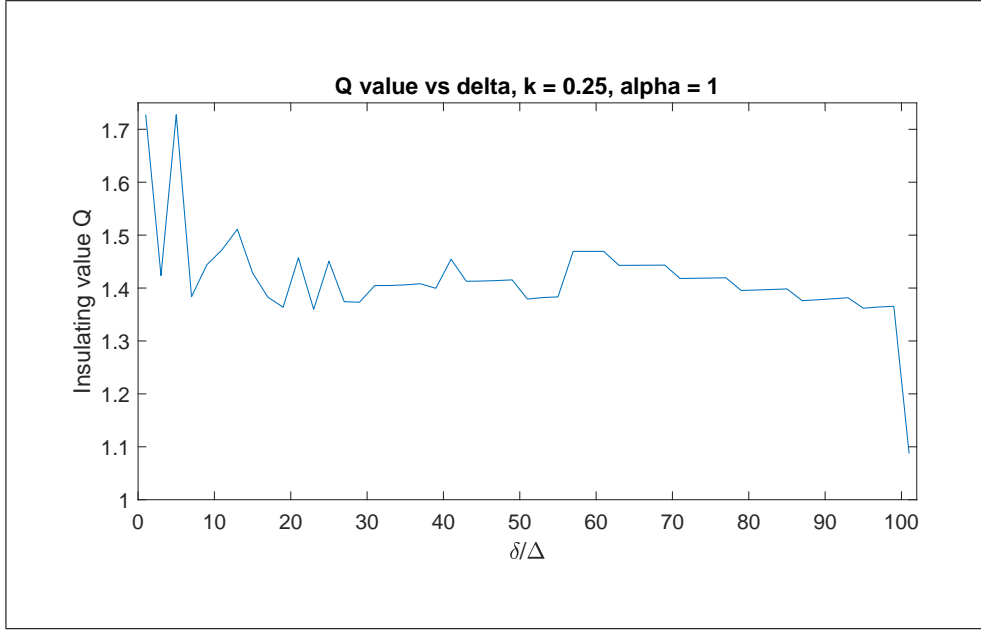


Figure 6: This is the case for which $\epsilon = \frac{\delta}{4}$

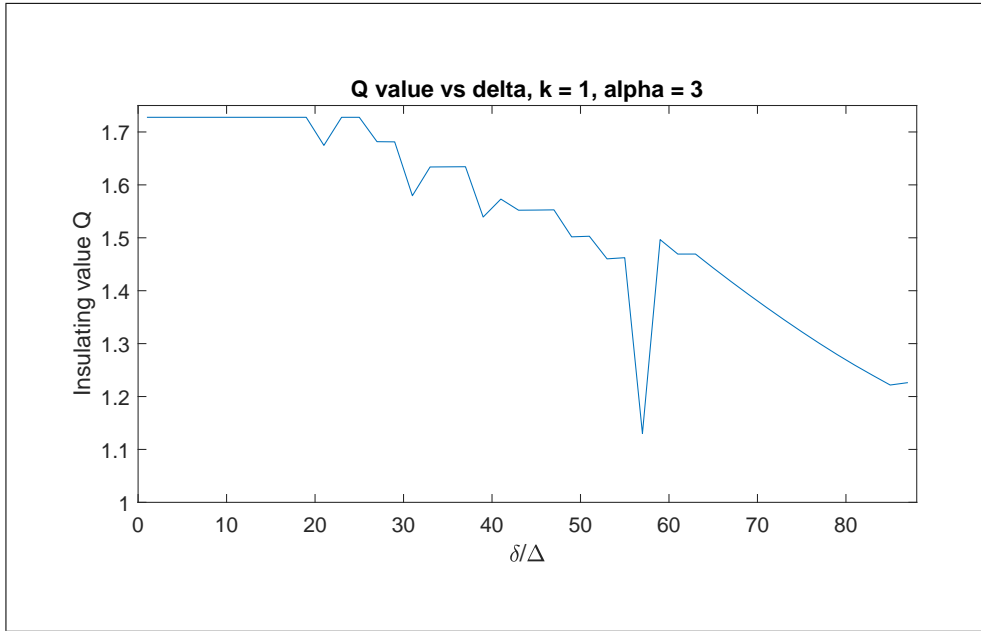


Figure 7: This is the case for which $\epsilon = \delta^3$

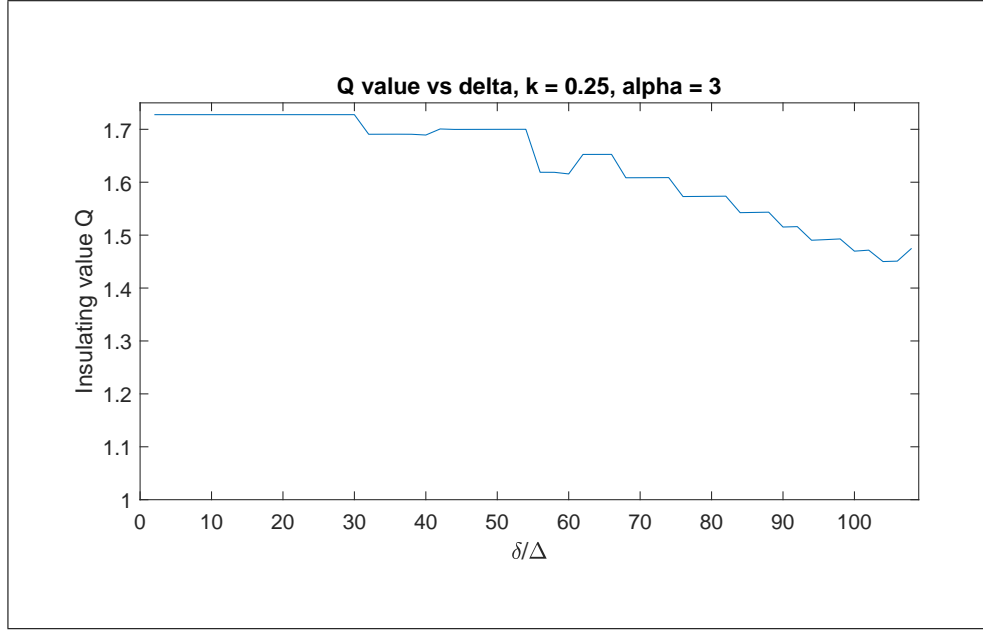


Figure 8: This is the case for which $\epsilon = \frac{\delta^3}{4}$

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.

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

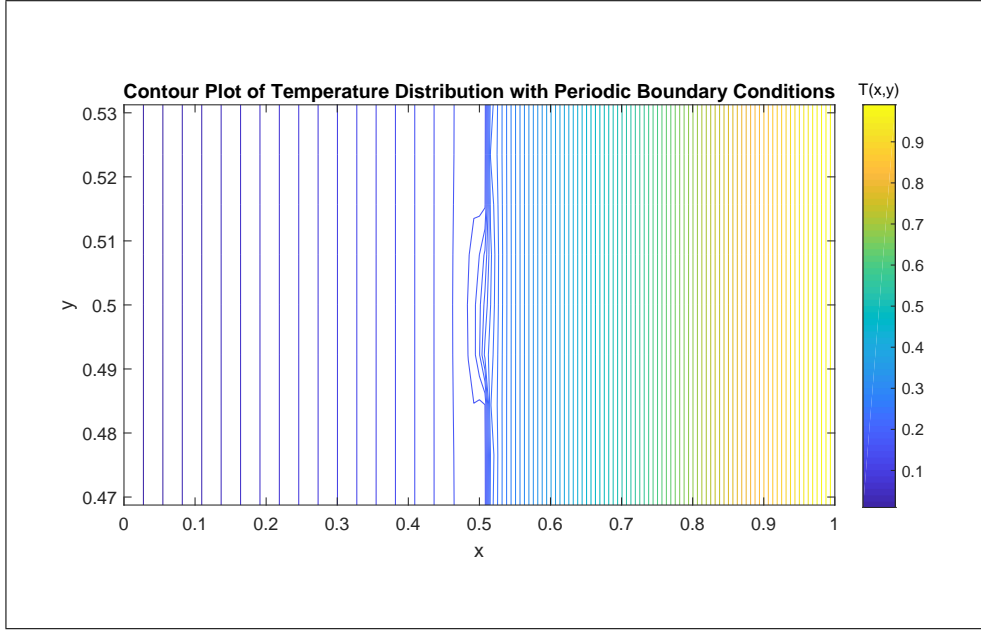


Figure 9: Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 1.587720$.

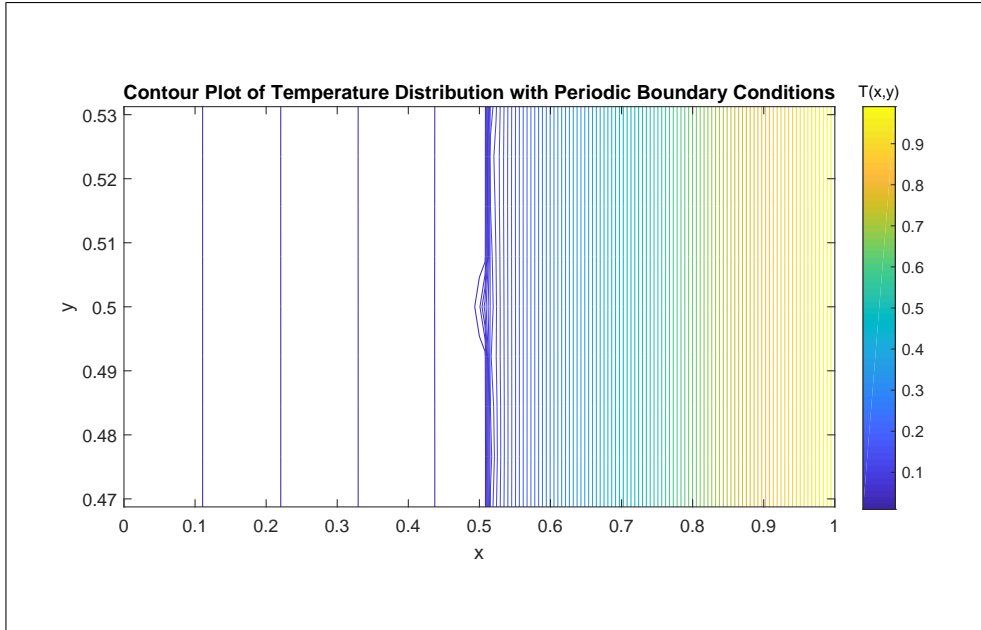


Figure 10: Temperature distribution for $\epsilon = 2, \delta = 8$, which has a value of $Q = 1.829834$

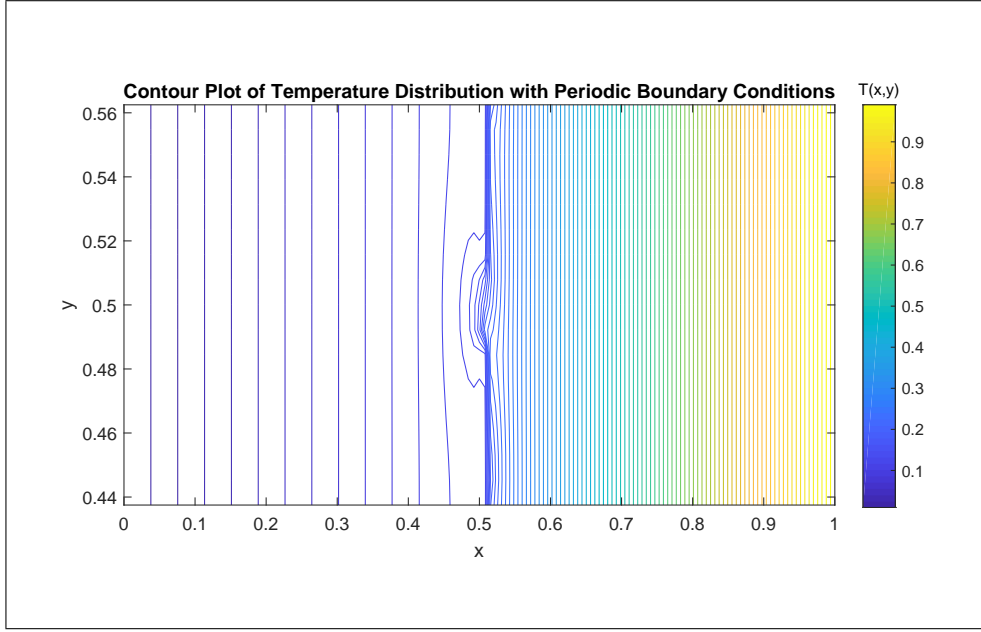


Figure 11: Temperature distribution for $\epsilon = 4, \delta = 16$, which has a value of $Q = 1.642438$.

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 $\Delta = \Delta x = \frac{1}{128}$.

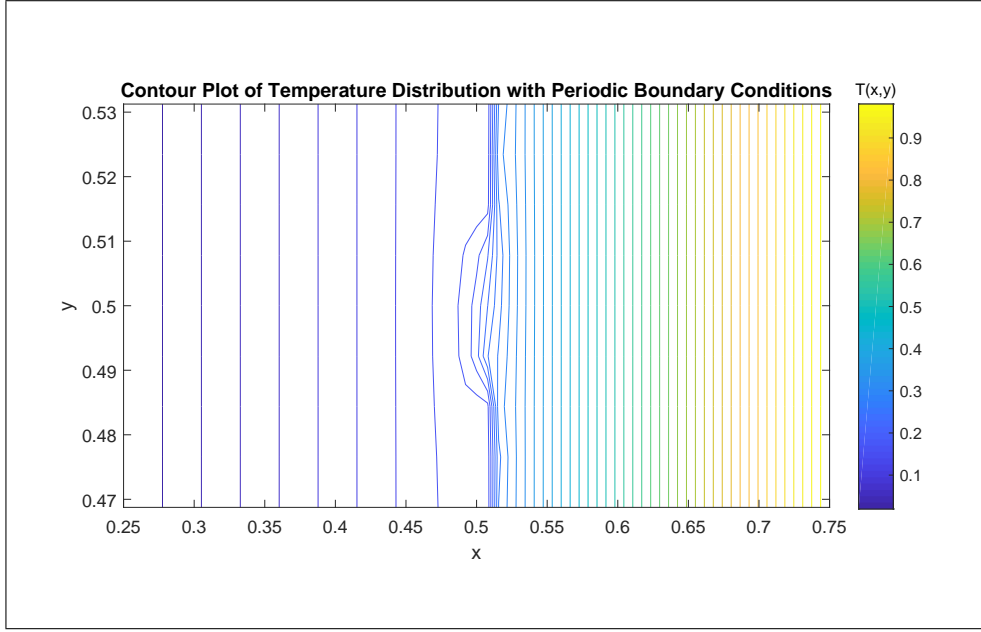


Figure 12: Shrinking N_x to $N_x = 64$ while keeping $\Delta = \frac{1}{128}$ constant. Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 3.095102$.

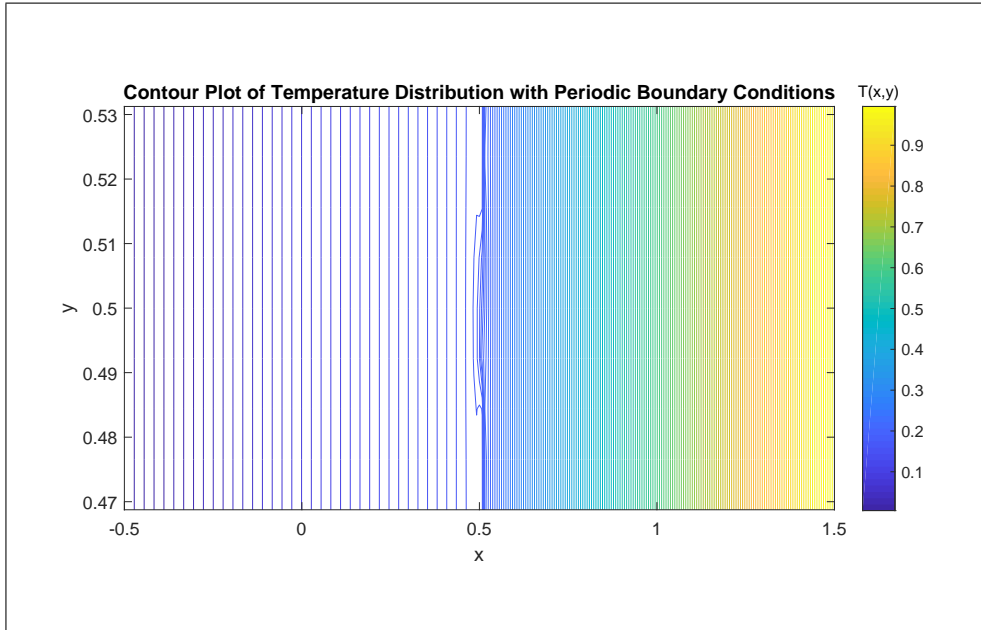


Figure 13: Increasing N_x to $N_x = 256$ while keeping $\Delta = \frac{1}{128}$ constant. Temperature distribution for $\epsilon = 4, \delta = 8$, which has a value of $Q = 0.807903$.

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.

```

1 function [T, Iteration, Xlength, Ylength] = RelaxationMethod(N, sigma)
2 %This function solves equation (2) in the project 7pt6 using a relaxation
3 %method
4 A = linspace(0,1,N+1);
5 TActual = repmat(A',1,N+1);
6 T = zeros(N+1,N+1);
7 Iteration = 0;
8 TOld = ones(size(T));
9 while (sum(sum(TOld-T).^2)/(N+1)^2) > 10^-10    %%Root mean square error
10     TOld = T;
11     T(1,:) = 0;
12     T(N+1,:) = 1;
13     for i = 2:N;
14         for j = 2:N;
15             T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+T(i,j
16                 +1)+T(i,j-1));
17         end
18         for j = 1
19             T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j
20                 +1));
21         end
22         for j = N+1
23             T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j
24                 -1));
25         end
26     end
27     Iteration = Iteration + 1;
28 end
29 contour(A,A,T',100);
30 title('Contour Plot of the Steady State Temperature Distribution')
31 xlabel('x')
32 ylabel('y')
33 title(colorbar,'T(x,y)')
34 Xlength = N;
35 Ylength = N;
36 end

```

```

1 function [T, Iteration, Xlength, N_y] = RelaxationMethodWithInsulatory(N, sigma,
    epsilon, delta)
2 %This function solves equation (2) in the project 7pt6 using a relaxation
3 %method
4 A = linspace(0,1,N+1);
5 T = zeros(N+1,N+1);
6 Iteration = 0; %counts iterations
7
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9
10 % 0 is No BC
11 % 1 is  $dT/dx = 0$ 
12 % 2 is  $dT/dx + dT/dy = 0$ 
13 % 3 is  $dT/dx - dT/dy = 0$ 
14 % 4 is  $dT/dx = dT/dy = 0$ 
15
16 endlength = 0.5*(mod(N-epsilon,delta)); %calculates the length of the insulators
    at the end
17
18 if endlength == 0
19     endlength = delta/2; %This deals with the case where there is no end
        boundary, it creates one
20 end
21
22 M = (N - epsilon - 2*endlength)/delta; %number of insulators in the middle of a
    common length
23
24 endlength=endlength+1; %Because we are working with the number of points it hits
    on the grid
25
26 if mod(2*endlength,2)==1
27     endlengthoriginal = endlength; %stores
28     endlength = floor(endlength);
29 else
30     endlengthoriginal = endlength; %This makes the endlength an integer number
31 end
32
33 length = delta - epsilon; %length of one piece of insulator in wall between
    adjacent holes
34
35 %Defines the endlength, M the number of full lengths of insulation in the
36 %middle, and also the length of a piece of insulation in the middle
37
38 Boundary1 = [0];
39 for i=1:endlength
40     if i == 1
41         Boundary1(i) = 4;
42     elseif i == endlength
43         Boundary1(i) = 2;
44     else
45         Boundary1(i) = 1;
46     end
47 end
48 if i~=1
49     if mod(2*endlengthoriginal,2)==1
50         Boundary1(i) = 1;
51     end
52 else
53 end
54 if mod(2*endlengthoriginal,2)==1

```



```

55     for i=endlength+1:endlength+epsilon
56         Boundary1(i)=0;
57     end
58 else
59     for i=endlength+1:endlength+(epsilon-1)
60         Boundary1(i)=0;
61     end
62 end
63
64 %T defines the boundary values along the piece of insulator, and this
65 %first bit sets up the boundary values up to the end of the first endpiece
66 %and then one epsilon length
67
68 if mod(2*endlengthoriginal,2)==1
69     for j = 0:M-1
70         for i = j*(delta)+endlength+epsilon+1:j*(delta)+endlength+epsilon+length
71             Boundary1(i) = 1;
72         end
73         for i = j*(delta)+endlength+epsilon+length+1:j*(delta)+endlength+epsilon
74             +length+epsilon
75             Boundary1(i)=0;
76         end
77     end
78 else
79     for j = 0:M-1
80         if epsilon ~= delta
81             for i=j*(delta)+endlength+(epsilon-1)+1
82                 Boundary1(i)=3;
83             end
84         else
85             for i=j*(delta)+endlength+(epsilon-1)+1
86                 Boundary1(i)=0;
87             end
88             for i=j*(delta)+endlength+(epsilon-1)+1+1:j*(delta)+endlength+(epsilon
89                 -1)+1+length
90                 if i == j*(delta)+endlength+(epsilon-1)+1+length
91                     Boundary1(i)=2;
92                 else
93                     Boundary1(i)=1;
94                 end
95             end
96             for i=j*(delta)+endlength+(epsilon-1)+1+length+1:j*(delta)+endlength+(
97                 epsilon-1)+1+length+(epsilon-1)
98                 Boundary1(i) = 0;
99             end
100         end
101     end
102
103 %Defines all the middle insulation lengths and the boundary conditions
104
105 if mod(2*endlengthoriginal,2)==1
106     for i = (M-1)*(delta)+endlength+epsilon+length+epsilon+1:(M-1)*(delta)+
107         endlength+epsilon+length+epsilon+endlength-1
108         Boundary1(i) = 1;
109     end
110     for i = (M-1)*(delta)+endlength+epsilon+length+epsilon+endlength
111         Boundary1(i) = 4;
112     end
113 else

```

```

111     for i=(M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1
112         Boundary1(i)=3;
113     end
114     for i=(M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1+(M-1)*delta
115         +endlength+(epsilon-1)+1+length+(epsilon-1)+1+endlength-1
116         if i == (M-1)*delta+endlength+(epsilon-1)+1+length+(epsilon-1)+1+
117             endlength-1
118             Boundary1(i) = 4;
119         else
120             Boundary1(i)=1;
121         end
122     end
123 %Defines the last endlength boundary conditions
124 Boundary2=[0];
125
126 for i=1:size(Boundary1,2)
127     if Boundary1(i) == 2
128         Boundary2(i) = 3;
129     elseif Boundary1(i) == 3
130         Boundary2(i) = 2;
131     else
132         Boundary2(i) = Boundary1(i);
133     end
134 end
135
136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
137
138 BoundaryInsulator = [[ flip1r (Boundary1) ]; [ flip1r (Boundary2) ]];
139 if epsilon == 0
140     Temp = [4, repmat(1,1,N-1),4];
141     BoundaryInsulator = repmat(Temp,2,1);
142 end
143
144 %Boundary conditions on the insulator
145
146 if mod(N,2) == 1
147     centre_beginning = (N+1)/2; %Perfect symmetry
148     centre_end = centre_beginning+1;
149     BoundaryInsulator = padarray(BoundaryInsulator, centre_beginning-1,0, 'both');
150 else
151     centre_beginning = (N)/2; %NO perfect symmetry so wlog it is slightly to the
152         left of the true centre
153     centre_end = centre_beginning+1;
154     BoundaryInsulator = padarray(BoundaryInsulator, centre_beginning-1,0, 'both');
155     BoundaryInsulator = padarray(BoundaryInsulator, 1,0, 'post');
156 end
157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
158
159 %%%Running the iteration algorithm
160 TOld = ones(size(T));
161 while (sum(sum(TOld-T).^2)/(N+1)^2) > 10^-10 %Root mean square dependent on the
162     previous iteration of the program
163     TOld = T;
164     T(1,:) = 0; %Boundary condition for x=0 side
165     T(N+1,:) = 1; %Boundary condition for x=N side
166     for i = 2:N
167         for j = 2:N

```

```

167         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));
168     end
169     for j = 1
170         T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j
           +1));
171         %Boundary condition for dT/dy = 0 at y = 0
172     end
173     for j = N+1
174         T(i,j) = (1-sigma)*T(i,j) + (sigma/4)*(T(i+1,j)+T(i-1,j)+2*T(i,j
           -1));
175         %Boundary condition for dT/dy = 0 at y = N
176     end
177 end
178 if epsilon ~= 0
179     for i = centre_beginning:centre_end
180         for j = 1:N+1
181             if BoundaryInsulator(i,j)==1
182                 %%%%%%%%%%%
183                 T(i+1,j)=T(i-1,j);
184             elseif BoundaryInsulator(i,j)==2
185                 T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1));
186             elseif BoundaryInsulator(i,j)==3
187                 T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
188             elseif BoundaryInsulator(i,j)==4
189                 if i == centre_beginning
190                     for j = 1
191                         T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)
                           +2*T(i,j+1));
192                         %Boundary 2
193                     end
194                     for j = N+1
195                         T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i+1,j)
                           +2*T(i,j-1));
196                         %Boundary 3
197                     end
198                 else
199                     for j = 1
200                         T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i-1,j)
                           +2*T(i,j+1));
201                         %Boundary 3
202                     end
203                     for j = N+1
204                         T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i,j-1)
                           +2*T(i-1,j));
205                         %Boundary 2
206                     end
207                 end
208             elseif BoundaryInsulator(i,j)==0
209                 else
210                     disp('Error with insulator boundary')
211                 end
212             end
213         end
214     else
215         %If the insulator is completely along the box with no
216         %holes?
217         %
218         %%%
219         for i = centre_beginning:centre_end

```

```

220     for j = 1:N+1
221         if BoundaryInsulator(i,j)==1
222             T(i+1,j)=T(i-1,j);
223         elseif BoundaryInsulator(i,j)==4
224             if i == centre_beginning
225                 for j = 1
226                     T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i
                        +1,j)+2*T(i,j+1));
227                     %Boundary 2
228                 end
229                 for j = N+1
230                     T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i
                        +1,j)+2*T(i,j-1));
231                     %Boundary 3
232                 end
233             else
234                 for j = 1
235                     T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i
                        -1,j)+2*T(i,j+1));
236                     %Boundary 3
237                 end
238                 for j = N+1
239                     T(i,j) = (1 - sigma)*T(i,j) + (sigma/4)*(2*T(i,j
                        -1)+2*T(i-1,j));
240                     %Boundary 2
241                 end
242             end
243         end
244     end
245 end
246 end
247 Iteration = Iteration + 1;
248 end
249 contour(A,A,T',100);
250 title('Contour Plot of Temperature Distribution with Insulator')
251 xlabel('x')
252 ylabel('y')
253 title(colorbar,'T(x,y)')
254 N_y = N;
255 Xlength=N;
256 end

```

```

1 [T, Iteration] = RelaxationMethodWithInsulator(128,1.9,4,8);
2 x=1:129;
3 figure(2)
4 plot(x,T(x,61)) %Through centre of hole
5 title('Horizontal cross section distribution - centre of hole')
6 xlabel('x')
7 ylabel('T(x,60)')
8 figure(3)
9 plot(x,T(x,65)) %Through centre of insulating section
10 title('Horizontal cross section distribution - centre of insulating section')
11 xlabel('x')
12 ylabel('T(x,64)')
13 figure(4)
14 plot(x,T(x,63)) %edge of insulating section
15 title('Horizontal cross section distribution - edge of insulating section')
16 xlabel('x')
17 ylabel('T(x,62)')
18 figure(5)
19 plot(x,T(x,2)) %one extreme
20 title('Horizontal cross section distribution - near bottom of insulating section
    ')
21 xlabel('x')
22 ylabel('T(x,1)')
23 figure(6)
24 plot(x,T(x,128)) %other extreme
25 title('Horizontal cross section distribution - near top of insulating section')
26 xlabel('x')
27 ylabel('T(x,127)')

```

```

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

```

```

1  clear
2  N = 128;
3  k = 0.25;
4  alpha = 1;
5  delta = (floor(N-sqrt(N)))*(1/N);
6  epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
7  while delta*N+epsilon*N+1>N
8      delta = delta - (1/N);
9      epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
10 end
11 xlimit = delta*N+1;
12 count = 1;
13
14 while delta>=epsilon
15     [T, Iteration, Xlength, Ylength] = RelaxationMethodWithInsulator(N, 1.9,
16         N*epsilon, N*delta);
17     CalculatingQfromT;
18     Qvector(count) = Q;
19     Deltavector(count) = delta*N;
20     Epsilonvector(count) = epsilon*N;
21     count = count + 1;
22     delta = delta - 2*(1/N);
23     epsilon = (round((k*(delta)^(alpha))*N))*(1/N);
24     if delta <= 0
25         break
26     end
27 end
28 figure(2)
29 plot(Deltavector, Qvector)
30 title(sprintf('Q value vs delta, k = %lg, alpha = %lg',k,alpha))
31 xlabel('\delta/\Delta')
32 ylabel('Insulating value Q')
33 ylim([1, 1.75])
34 xlim([0, xlimit])

```

```

1 function [T, Iteration, Xlength, N_y] = RelaxationMethodPeriodicQ4(N_x, sigma,
    epsilon, delta)
2 %This function solves equation (2) in the project 7pt6 using a relaxation
3 %method
4 Ax = linspace(0,1,N_x+1);
5 T = zeros(N_x+1,delta+1);
6 Iteration = 0; %counts iterations
7
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9
10 % 0 is No BC
11 % 1 is  $dT/dx = 0$  FOR ZERO GRADIENT
12 % 2 is  $dT/dx + dT/dy = 0$  FOR POSITIVE GRADIENT
13 % 3 is  $dT/dx - dT/dy = 0$  FOR NEGATIVE GRADIENT
14
15 Boundary1 = [0];
16 for i=1:delta+1
17     if mod(delta,2)==0 %delta even
18         if mod(epsilon,2)==0 %epsilon even
19             if i < (delta/2)+1-(epsilon/2)
20                 Boundary1(i) = 1;
21             elseif i == (delta/2)+1-(epsilon/2)
22                 Boundary1(i) = 2;
23             elseif i > (delta/2)+1-(epsilon/2) && i < (delta/2)+1+(epsilon/2)
24                 Boundary1(i) = 0;
25             elseif i == (delta/2)+1+(epsilon/2)
26                 Boundary1(i) = 3;
27             else
28                 Boundary1(i) = 1;
29             end
30         else %epsilon odd
31             if i < (delta/2)+1-((epsilon+1)/2)
32                 Boundary1(i) = 1;
33             elseif i > (delta/2)+1-((epsilon+1)/2) && i < (delta/2)+1+((epsilon
34                 +1)/2)
35                 Boundary1(i) = 0;
36             else
37                 Boundary1(i) = 1;
38             end
39         else %delta odd
40             if mod(epsilon,2)==1 %epsilon odd
41                 if i < ((delta+1)/2)-((epsilon-1)/2)
42                     Boundary1(i) = 1;
43                 elseif i == ((delta+1)/2)-((epsilon-1)/2)
44                     Boundary1(i) = 2;
45                 elseif i > ((delta+1)/2)-((epsilon-1)/2) && i < ((delta+1)/2)+1+((
46                     epsilon-1)/2)
47                     Boundary1(i) = 0;
48                 elseif i == ((delta+1)/2)+1+((epsilon-1)/2)
49                     Boundary1(i) = 3;
50                 else
51                     Boundary1(i) = 1;
52                 end
53             else %epsilon even
54                 if i <= ((delta+1)/2)-(epsilon/2)
55                     Boundary1(i) = 1;
56                 elseif i > ((delta+1)/2)-(epsilon/2) && i < ((delta+1)/2)+1+(epsilon
57                     /2)
58                     Boundary1(i) = 0;

```



```

57         else
58             Boundary1(i) = 1;
59         end
60     end
61 end
62
63 end
64
65
66 %Above defines the strip of length delta that we will apply periodic
67 %boundary conditions to.
68
69
70 %Defines all the middle insulation lengths and the boundary condiitons
71
72 Boundary2=[0];
73
74 for i=1:size(Boundary1,2)
75     if Boundary1(i) == 2
76         Boundary2(i) = 3;
77     elseif Boundary1(i) == 3
78         Boundary2(i) = 2;
79     else
80         Boundary2(i) = Boundary1(i);
81     end
82 end
83
84 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
85
86 BoundaryInsulator = [[ flip1r(Boundary1) ]; [ flip1r(Boundary2) ]];
87 %Defines the matrix that represents the boundary conditions
88 if epsilon == 0
89     Temp = [ repmat(1,1,delta+1) ];
90     BoundaryInsulator = repmat(Temp,2,1);
91 end
92
93
94 %Boundary conditions on the insulator are set in the matrix
95 %BoundaryInsulator
96
97 if mod(N_x,2) == 1
98     centre_beginning = (N_x+1)/2; %Perfect symmetry
99     centre_end = centre_beginning+1;
100     BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
101 else
102     centre_beginning = (N_x)/2; %NO perfect symmetry so wlog it is slightly to
        the left of the true centre
103     centre_end = centre_beginning+1;
104     BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
105     BoundaryInsulator = padarray(BoundaryInsulator,1,0,'post');
106 end
107
108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
109
110 %%%Running the iteration algorithm
111 Told = ones(size(T));
112 while (sum(sum(Told-T).^2)/(N_x+1)^2) > 10^-10    %Root mean square dependent on
        the previous iteration of the program
113     Told = T;
114     T(1,:) = 0; %Boundary condition for x=0 side

```

```

115     T(N_x+1,:) = 1; %Boundary condition for x=N side
116     for i = 2:N_x
117         for j = 2:delta-1
118             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));
119         end
120         for j = 1
121             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));
122             %Boundary condition for dT/dy = 0 at y = 0
123         end
124         for j = delta
125             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));
126             %Periodic Boundary COndition at j=delta
127         end
128         for j = delta+1
129             T(i,j) = T(i,1);
130             %Periodic Boundary Condition for j=delta+1 and j=1
131         end
132     end
133     if epsilon ~= 0 && epsilon ~= delta
134         for i = centre_beginning:centre_end
135             for j = 1:delta+1
136                 if BoundaryInsulator(i,j)==1
137                     %%%%%%%%%%%%%%
138                     T(i+1,j)=T(i-1,j); %No flux in the x direction (dT/dx = 0)
139                 elseif BoundaryInsulator(i,j)==2
140                     T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1)); %dT/dx + dT/dy = 0
141                 elseif BoundaryInsulator(i,j)==3
142                     T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
143                 elseif BoundaryInsulator(i,j)==0
144                     else
145                         disp('Error with insulator boundary')
146                     end
147                 end
148             end
149             elseif epsilon == delta
150
151             elseif epsilon == 0
152                 % If the insulator is completely along the box with no
153                 % holes?
154                 %
155                 %
156                 for i = centre_beginning:centre_end
157                     for j = 1:delta+1
158                         if BoundaryInsulator(i,j)==1
159                             T(i+1,j)=T(i-1,j);
160                         end
161                     end
162                 end
163             else
164                 disp('Error with epsilon and delta')
165                 return
166             end
167             Iteration = Iteration + 1;
168         end
169         if mod(delta,2)==0
170             if mod(N_x,2)==0
171                 Ay = linspace((((N_x/2)-(delta/2)))/N_x,((N_x/2)+(delta/2))/N_x,delta+1)

```

```

172         ;
173     else
174         Ay = linspace((((N_x-1)/2)-(delta/2))/N_x,(((N_x-1)/2)+(delta/2))/N_x,
175             delta+1);
176     end
177 else
178     if mod(N_x,2)==0
179         Ay = linspace(((N_x/2)-(delta/2))/N_x,((N_x/2)+(delta/2))/N_x,delta+1);
180     else
181         Ay = linspace(((N_x/2)-(delta/2))/N_x,((N_x/2)+(delta/2))/N_x,delta+1);
182     end
183 end
184 contour(Ax,Ay,T',100);
185 title('Contour Plot of Temperature Distribution with Periodic Boundary
186     Conditions')
187 xlabel('x')
188 ylabel('y')
189 title(colorbar,'T(x,y)')
190 N_y=delta;
191 Xlength = N_x;
192 end

```

```

1 function [T, Iteration, Xlength, N_y] = RelaxationMethodPeriodicQ5(N_x, sigma,
    epsilon, delta)
2 %This function solves equation (2) in the project 7pt6 using a relaxation
3 %method
4 N=128;
5 T = zeros((N_x)+1,delta+1);
6 Iteration = 0; %counts iterations
7
8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9
10 % 0 is No BC
11 % 1 is  $dT/dx = 0$  FOR ZERO GRADIENT
12 % 2 is  $dT/dx + dT/dy = 0$  FOR POSITIVE GRADIENT
13 % 3 is  $dT/dx - dT/dy = 0$  FOR NEGATIVE GRADIENT
14
15
16 length = delta - epsilon; %length of one piece of insulator in wall between
    adjacent holes
17
18 Boundary1 = [0];
19 for i=1:delta+1
20     if mod(delta,2)==0 %delta even
21         if mod(epsilon,2)==0 %epsilon even
22             if i < (delta/2)+1-(epsilon/2)
23                 Boundary1(i) = 1;
24             elseif i == (delta/2)+1-(epsilon/2)
25                 Boundary1(i) = 2;
26             elseif i > (delta/2)+1-(epsilon/2) && i < (delta/2)+1+(epsilon/2)
27                 Boundary1(i) = 0;
28             elseif i == (delta/2)+1+(epsilon/2)
29                 Boundary1(i) = 3;
30             else
31                 Boundary1(i) = 1;
32             end
33         else %epsilon odd
34             if i < (delta/2)+1-((epsilon+1)/2)
35                 Boundary1(i) = 1;
36             elseif i > (delta/2)+1-((epsilon+1)/2) && i < (delta/2)+1+((epsilon
                +1)/2)
37                 Boundary1(i) = 0;
38             else
39                 Boundary1(i) = 1;
40             end
41         end
42     else %delta odd
43         if mod(epsilon,2)==1 %epsilon odd
44             if i < ((delta+1)/2)-((epsilon-1)/2)
45                 Boundary1(i) = 1;
46             elseif i == ((delta+1)/2)-((epsilon-1)/2)
47                 Boundary1(i) = 2;
48             elseif i > ((delta+1)/2)-((epsilon-1)/2) && i < ((delta+1)/2)+1+((
                epsilon-1)/2)
49                 Boundary1(i) = 0;
50             elseif i == ((delta+1)/2)+1+((epsilon-1)/2)
51                 Boundary1(i) = 3;
52             else
53                 Boundary1(i) = 1;
54             end
55         else %epsilon even
56             if i <= ((delta+1)/2)-(epsilon/2)

```

```

57         Boundary1(i) = 1;
58     elseif i > ((delta+1)/2)-(epsilon/2) && i < ((delta+1)/2)+1+(epsilon
        /2)
59         Boundary1(i) = 0;
60     else
61         Boundary1(i) = 1;
62     end
63 end
64 end
65
66 end
67
68
69 %Above defines the strip of length delta that we will apply periodic
70 %boundary conditions to.
71
72
73 %Defines all the middle insulation lengths and the boundary condiitons
74
75 Boundary2=[0];
76
77 for i=1:size(Boundary1,2)
78     if Boundary1(i) == 2
79         Boundary2(i) = 3;
80     elseif Boundary1(i) == 3
81         Boundary2(i) = 2;
82     else
83         Boundary2(i) = Boundary1(i);
84     end
85 end
86
87 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
88
89 BoundaryInsulator = [[fliplr(Boundary1)];[fliplr(Boundary2)]];
90 %Defines the matrix that represents the boundary conditions
91 if epsilon == 0
92     Temp = [repmat(1,1,delta+1)];
93     BoundaryInsulator = repmat(Temp,2,1);
94 end
95
96
97 %Boundary conditions on the insulator are set in the matrix
98 %BoundaryInsulator
99
100 if mod(N,2) == 1
101     centre_beginning = (N+1)/2; %Perfect symmetry
102     centre_end = centre_beginning+1;
103 else
104     centre_beginning = (N)/2; %NO perfect symmetry so wlog it is slightly to the
        left of the true centre
105     centre_end = centre_beginning+1;
106 end
107
108 if mod(N_x,2)==0
109     x_1 = (centre_beginning*(1/N))-(((N_x)/2)*(1/N));
110     x_2 = (centre_beginning*(1/N))+((N_x/2)*(1/N));
111 else
112     x_1 = (centre_beginning*(1/(N)))-(((N_x+1)/2)*(1/(N)));
113     x_2 = (centre_beginning*(1/(N)))+(((N_x+1)/2)*(1/(N)));
114 end

```

```

115
116 Ax = linspace(x_1,x_2,(N_x)+1);
117
118 if mod(N_x,2) == 1
119     centre_beginning = (N_x+1)/2; %Perfect symmetry
120     centre_end = centre_beginning+1;
121     BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
122 else
123     centre_beginning = (N_x)/2; %NO perfect symmetry so wlog it is slightly to
        the left of the true centre
124     centre_end = centre_beginning+1;
125     BoundaryInsulator = padarray(BoundaryInsulator,centre_beginning-1,0,'both');
126     BoundaryInsulator = padarray(BoundaryInsulator,1,0,'post');
127 end
128
129 %%%%%%%%%%%%%%
130
131 %%%Running the iteration algorithm
132
133 TOld = ones(size(T));
134 while (sum(sum(TOld-T).^2)/(N_x+1)^2) > 10^-10 %Root mean square dependent on
        the previous iteration of the program
135     TOld = T;
136     T(1,:) = 0; %Boundary condition for x=0 side
137     T(N_x+1,:) = 1; %Boundary condition for x=N side
138     for i = 2:N_x
139         for j = 2:delta-1
140             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));
141         end
142         for j = 1
143             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));
144             %Boundary condition for dT/dy = 0 at y = 0
145         end
146         for j = delta
147             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));
148             %Periodic Boundary COndition at j=delta
149         end
150         for j = delta+1
151             T(i,j) = T(i,1);
152             %Periodic Boundary Condition for j=delta+1 and j=1
153         end
154     end
155     if epsilon ~ = 0 && epsilon ~ = delta
156         for i = centre_beginning:centre_end
157             for j = 1:delta+1
158                 if BoundaryInsulator(i,j)==1
159                     %%%%%%%%%%%%%%
160                     T(i+1,j)=T(i-1,j); %No flux in the x direction (dT/dx = 0)
161                 elseif BoundaryInsulator(i,j)==2
162                     T(i+1,j)=T(i-1,j)-(T(i,j+1)-T(i,j+1)); %dT/dx + dT/dy = 0
163                 elseif BoundaryInsulator(i,j)==3
164                     T(i+1,j)=T(i-1,j)+(T(i,j+1)-T(i,j+1));
165                 elseif BoundaryInsulator(i,j)==0
166                     else
167                         disp('Error with insulator boundary')
168                     end
169             end
170         end
171     end

```

```

170         end
171         elseif epsilon == delta
172
173         elseif epsilon == 0
174             % If the insulator is completely along the box with no
175             % holes?
176             %
177             %
178             for i = centre_beginning:centre_end
179                 for j = 1:delta+1
180                     if BoundaryInsulator(i,j)==1
181                         T(i+1,j)=T(i-1,j);
182                     end
183                 end
184             end
185         else
186             disp('Error with epsilon and delta')
187             return
188         end
189         Iteration = Iteration + 1;
190     end
191     if mod(delta,2)==0
192         if mod(N,2)==0
193             Ay = linspace(((N/2)-(delta/2))/N,((N/2)+(delta/2))/N,delta+1);
194         else
195             Ay = linspace(((N-1)/2)-(delta/2))/N,(((N-1)/2)+(delta/2))/N,delta+1);
196         end
197     else
198         if mod(N,2)==0
199             Ay = linspace((N/2)-(delta/2))/N,((N/2)+(delta/2))/N,delta+1);
200         else
201             Ay = linspace((N/2)-(delta/2))/N,((N/2)+(delta/2))/N,delta+1);
202         end
203     end
204     contour(Ax,Ay,T',abs(x_1-x_2)*100);
205     %normalise the number of contour lines so that between 0 and 1 there is alwyas
206     100
207     title('Contour Plot of Temperature Distribution with Periodic Boundary
208     Conditions')
209     xlabel('x')
210     ylabel('y')
211     title(colorbar,'T(x,y)')
212     N_y=delta;
213     Xlength = N_x;
214
215     clear QMatrix Q
216     for j=2:N_y
217         QMatrix(j-1) = (T(N_x+1,j)-T(N_x-1,j))/(2*(1/N));
218     end
219     Q = sum(QMatrix)/(N_y-1)
220 end

```