Project Report

“Solving various PDEs arising in Physics”

Abstract

This project will be looking at PDEs in physics and how they can be solved using numerical methods. The main methods used are forward and backward difference and matric solving. There will be two scenarios, 1st scenario is the Time independent steady state heat equation (TISSHE). This only used the three main methods and gave a correct answer proving it as a viable way to solve the TISSHE. The 2nd method was the One-Dimensional Heat equation (ODHE). This used both central difference and Crank-Nicolson method to solve as well. This also gave the correct answer and shows that it can be solved by numerical methods. However a short coming for both is that it is not accurate compare to being done by hand. However increasing the data points would increase the accuracy.

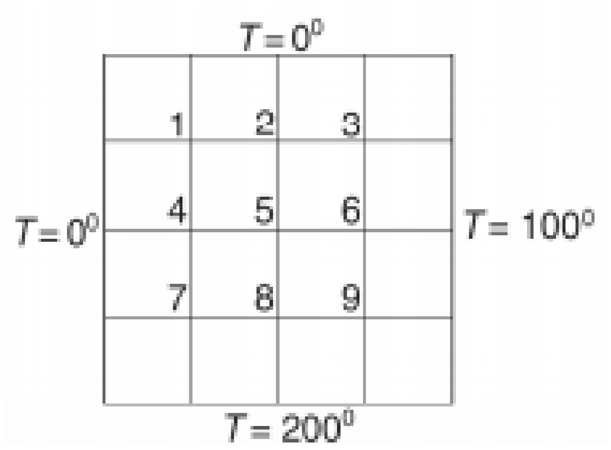
Introduction and Theory

Partial differential equations arise in all sorts of situations in physics. From fluid dynamics to quantum-mechanics. PDEs can be very simple and done by hand but they can also be very complicated. This means for complex PDEs such as the fluid dynamics of water in a pipe or in a dam there can be too many complicated PDEs to calculate taking lots of time and resources to solve, or being simplified and having a large error in it. The use of computers to solve PDEs can help speed up the time it takes to do a calculation and stops the need to simplify causing error.

The problem is computers cannot solve PDEs directly the way people can and as such require numerical methods as a way solve the PDEs. As a result of this, being able to simplify PDEs into simple equations that can be computationally solved is a must when doing physics. One method of doing this is by numerical methods. This project will outline the advantages and disadvantages of solving PDEs by numerical methods. This will be done by going through two different by related situations. First situation is the Time Independent Steady State Heat Equation (TISSHE) the second is the One-Dimensional Heat Equation (ODHE). Both of these require similar and yet different approaches.

In the project the situation for the TDSSHE was on an x-y plane with arbitrary lengths such that it formed a square. Then at a set point try to find the temperature at that point, given that the temperature at the edges is given. Setup like so:

Figure (1): TISSHE situation setup[1]



It is also given that the TISSHE is:

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0 \qquad (1)^2$$

[1]Q4, Exercise 6, Numerical methods, Barry Smalley, School of Chemical and Physical sciences, Keele University
[2]Q4, Exercise 6, Numerical methods, Barry Smalley, School of Chemical and Physical sciences, Keele University

Where T is the temperature. To solve this PDE it be will discussed in more detail in the method as it requires numerous numerical methods to solve.

The situation for the ODHE is that it is time dependent and thus requires the past values to be used in order to find the temperature at the point and time. The scenario is looking at a rod of length L that is cooled to 0^oC at either end. The project will look at the temperature of the rod at set points and time intervals. Due to it being time dependent and in one dimension the following equation is:

$$\frac{\partial u}{\partial t} = K\frac{\partial^2 u}{\partial x^2} \qquad (2)^3$$

Where u is the temperature, t is the time, x is the position and K = k_t/cρ with k_t is the thermal conductivity of the material, c is the specific heat capacity of the material and ρ is the density of the material. This mean K is a constant. If the following changes are made to the transformation variables (x and t) such that x = XL where 0 ≤ X ≤ 1 and T= t/L^2K gives the following equation:

$$\frac{\partial u}{\partial T} = \frac{\partial^2 u}{\partial X^2} \qquad (3)^4$$

The is at a temperature such that Having both of the equations setup like so allows for numerical methods to be used and will be discussed more in the method.

Method

For the first part of the method, it will cover TISSHE. Taking equation (1) and the 2nd order PDEs can be found by using Taylor expansion for a small step in h from point x and a small step k from point y. The reason as to why there is a small step of h and k is because to find the gradient at the point x and y there needs to be two points in the change in y/x to be calculated. Therefore having a step of h or k, where h and k tend to 0 will give you the gradient at that point. This is known as the forward or backward difference depending on if the step is positive or negative. To the do forward or backward difference, Taylor expansion is used. Due to it being a second order equation the terms above d^2/dx^2 can be ignored. Due to there being two variables (x+h and y+k) a two variable Taylor expansion is to be used. This gives the following result:

$$T(x+h,y+k) \approx T(x,y) + hT_x + kT_y + \frac{h^2}{2!}T_{xx} + 2hkT_{xy} + \frac{k^2}{2!}T_{yy} \quad (4)$$

Where T_{xx} is the second derivative with respect to x and T_x is the first derivative with respect to x. Once it has been expanded out, first is to find the second partial derivative with respect to one value in this case that will be x+h. Therefore let k =0 giving the following equation:

$$T(x+h,y) \approx T(x,y) + hT_x + \frac{h^2}{2!}T_{xx} \qquad (5)$$

If the Second partial derivative is ignored then the first derivative can be found. For ease later on the backwards difference method will be used, which gives:

$$T_x \approx \frac{T(x,y)-T(x+h,y)}{h} \qquad (6)$$

With this it can be subbed in to and rearranged to find the second partial differential with respect to x giving:

$$T_{xx} \approx \frac{2!}{h^2}[T(x+h,y) - T(x,y) - T(x,y) + T(x-h,y)] \qquad (7)$$

3Chapter 21, Advanced Engineering Mathematic Tenth Edition, Erwin Kreszig
4Chapter 21, Advanced Engineering Mathematic Tenth Edition, Erwin Kreszig

To find the second partial derivative with respect to y the same process can be done except with h=0 giving:

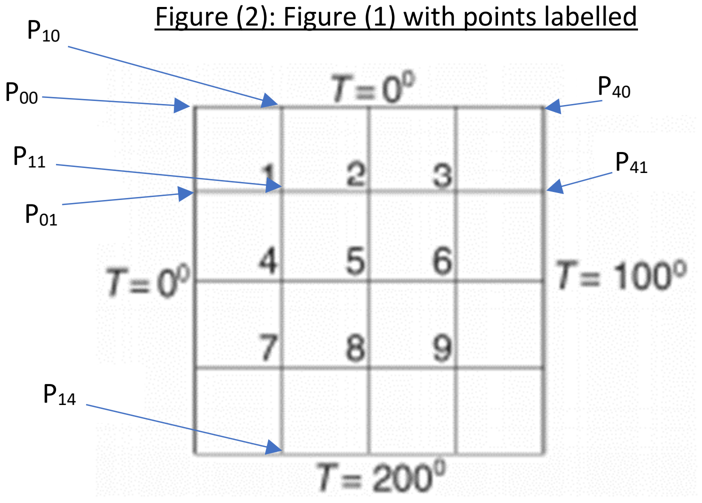
$$T_{yy} \approx \frac{2!}{k^2}[T(x,y-k) - T(x,y) - T(x,y) + T(x,y-k)] \qquad (8)$$

For simplicity when both derivatives where subbed into equation (1) let h = k giving the following equation:

$$\frac{2}{h^2}[T(x+h,y) + T(x-h,y) - 4T(x,y) + T(x,y+h) + T(x,y-h)] = 0 \quad (9)$$

Now using this let h = 1, x=i and y=j. Now referring to figure (1) each point was labelled as seen in this example:

Figure (2): Figure (1) with points labelled



Where P is the point and therefore P$_{ij}$ is the position of the point. This mean that the following equation can be derived from these conditions:

$$T_{i+1,j} + T_{i-1,j} - 4T_{i,j} + T_{i,j+1} + T_{i,j-1} = 0 \qquad (10)$$

This equation can be rearranged to:

$$4T_{i,j} - T_{i+1,j} - T_{i-1,j} - T_{i,j+1} - T_{i,j-1} = 0 \qquad (11)$$

By setting i=1 and j=1 can be derived to give the following:

$$4T_{1,1} - T_{2,1} - T_{0,1} - T_{1,2} - T_{1,0} = 0 \qquad (12)$$

Due to $T_{0,1}$ and $T_{1,0}$ being boundary conditions then it is known what the temperature is means the equation can be rearranged as:

$$4T_{1,1} - T_{2,1} - T_{1,2} = T_{1,0} + T_{0,1} \qquad (13)$$

Knowing this when going through the values of i and j between 1-3 if I ,j = 0,4 then the value of it is known and can have the equation equal to it as I,j=0,4 is a boundary condition. This process is repeated for all variables of I and j where I,j=1,2,3. Once all the equations have been made. All of the equation had 0T$_{i,j}$ variables added. For example for equation (13):

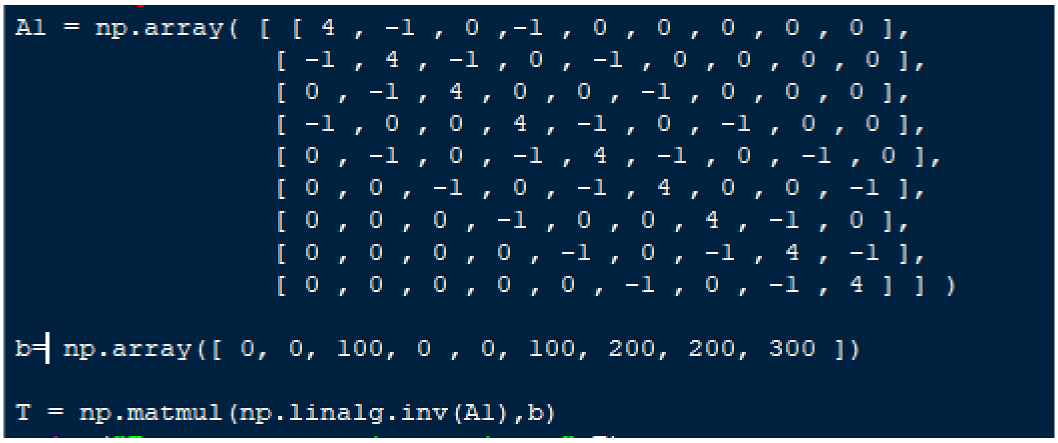
$$4T_{1,1} - T_{2,1} + 0T_{3,1} - T_{1,2} + 0T_{2,2} + 0T_{3,2} + 0T_{1,3} + 0T_{2,3} + 0T_{3,3} = T_{1,0} + T_{0,1} \quad (14)$$

Doing this for all nine variations gives a matrix. Which can be used to find out the values of T at each point. Knowing this the matrix was divided thorough by T$_{i,j}$ giving:

$$\begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} T_{1,1} \\ T_{2,1} \\ T_{3,1} \\ T_{1,2} \\ T_{2,2} \\ T_{2,3} \\ T_{1,3} \\ T_{2,3} \\ T_{3,3} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 100 \\ 0 \\ 0 \\ 100 \\ 200 \\ 200 \\ 300 \end{pmatrix} \qquad (15)$$

With this it can now be coded in to solve and be plotted as a heat map. To do this the following code was used:

Figure (3): code of equation 15 to find the Temperatures at each point



```
A1 = np.array( [ [ 4 , -1 , 0 ,-1 , 0 , 0 , 0 , 0 , 0 ],
                 [ -1 , 4 , -1 , 0 , -1 , 0 , 0 , 0 , 0 ],
                 [ 0 , -1 , 4 , 0 , 0 , -1 , 0 , 0 , 0 ],
                 [ -1 , 0 , 0 , 4 , -1 , 0 , -1 , 0 , 0 ],
                 [ 0 , -1 , 0 , -1 , 4 , -1 , 0 , -1 , 0 ],
                 [ 0 , 0 , -1 , 0 , -1 , 4 , 0 , 0 , -1 ],
                 [ 0 , 0 , 0 , -1 , 0 , 0 , 4 , -1 , 0 ],
                 [ 0 , 0 , 0 , 0 , -1 , 0 , -1 , 4 , -1 ],
                 [ 0 , 0 , 0 , 0 , 0 , -1 , 0 , -1 , 4 ] ] )

b= np.array([ 0, 0, 100, 0 , 0, 100, 200, 200, 300 ])

T = np.matmul(np.linalg.inv(A1),b)
```

The reason as to why matrix A1 was inverted and multiplied to matrix b is because T=b/A1. This gave the results of T and then was then plotted on to a heat map. For the project the heat map included the boundary conditions. This was done by making a new matrix and filling in the results manually.

For the second scenario (the ODHE) by doing finite difference like when doing the TISSHE. In this case instead of h and k the use of α and β will be used as the step sizes. So by doing finite difference on equation (3) gives the time dependent side as:

$$u_T \approx \frac{u(X,T+\beta)-u(X,T)}{\beta} \qquad (16)$$

As there is no information of negative T as that would require negative time. For $\frac{\partial^2 u}{\partial X^2}$ the use of central difference approximation was used. Central difference approximation takes the forward difference and subtracts the backward difference. This gave the following second derivative at time T:

$$u_{XX} \approx \frac{u(X+\alpha,T)-2u(X,T)+u(X-\alpha,T)}{\alpha^2} \qquad (17)$$

And at T+β:

$$u_{XX} \approx \frac{u(X+\alpha,T+\beta)-2u(X,T+\beta)+u(X-\alpha,T+\beta)}{\alpha^2} \qquad (18)$$

Using both of the equations this gives the final formula of:

$$\frac{\partial^2 u}{\partial X^2} \approx \frac{1}{2\alpha^2}[u(X+\alpha,T)-2u(X,T)+u(X-\alpha,T)] + \frac{1}{2\alpha^2}[u(X+\alpha,T+\beta)-2u(X,T+\beta)+ u(X-\alpha,T+\beta)] \qquad (19)$$

With that then considering an x-y plot except the y axis is time. It was then set such that X=iα and T=jβ and substituting that in to equation (16) and (19), then like the what was done for TISSHE when going from equation (9) to (1). After that substituting that in to equation (3) gives:

$$2\left(u_{i,j+1} - u_{i,j}\right) = \frac{\beta}{\alpha^2}\left[u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right] \quad (20)$$

By using the Crank-Nicolson method, β and α are constants such that r= β/α^2 and r= 1 therefore α^2=β giving the following equation:

$$4u_{i,j+1} = u_{i+1,j} + u_{i-1,j} + u_{i+1,j+1} + u_{i-1,j+1} \quad (21)$$

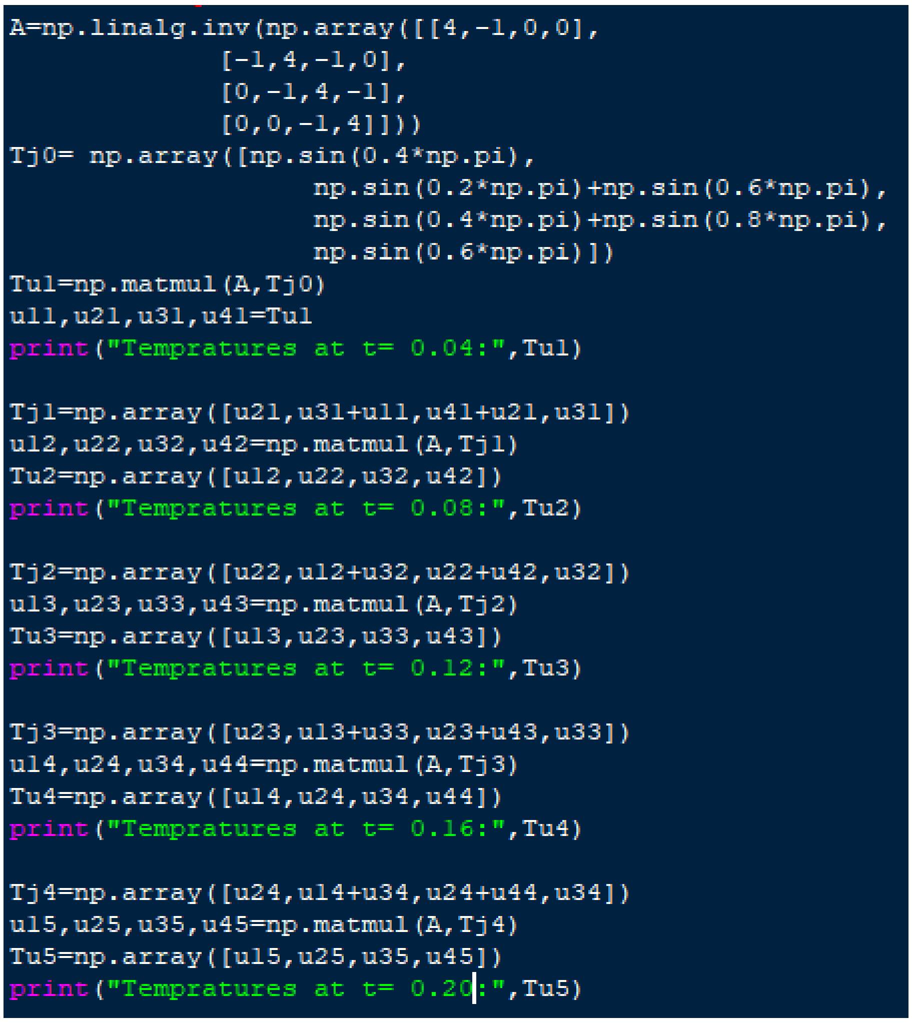
For the scenario the rod is cooled on both ends implies that u(0,T) = 0 and u(1,T)= 0, however the temperature at point X at T= 0 is given by u(X,0)= sin(πX). Due to this for the project the time will look at 0 ≤ T ≤ 0.2, as X include the length L the range will be 0 ≤ X ≤ 1 as any greater is outside the rod. For the project α = 0.2 and that implies β =0.04 due to r=1. Due to these ranges the values for I=1,2,3,4 as X=iα and the values being found for j=1,2,3,4,5 as T=jβ that is for the ranges between the ranges stated

Once this was all done the temperature across the bar at various points was calculated when T= 0 or j=0 which gives: $u_{0,0}$=0,$u_{1,0}$=sin(0.2π),$u_{2,0}$=sin(0.4π),$u_{3,0}$=sin(0.6π),$u_{4,0}$=sin(0.8π),$u_{5,0}$=0. These are the boundary conditions and the initial conditions that will be used in the future of the matrix. As $u_{0,j}$ and $u_{5,j}$ are the boundaries they will always equal 0. With this known the same process can be done as the TISSHE but for the different values of I when j=0. Doing this will give a 4x4 matrix as shown below:

$$\begin{pmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{pmatrix}\begin{pmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{4,1} \end{pmatrix} = \begin{pmatrix} u_{2,0} \\ u_{1,0} + u_{3,0} \\ u_{2,0} + u_{4,0} \\ u_{3,0} \end{pmatrix} \quad (22)$$

As seen by using equation (21) for varying values of I at j=0 the values for j=1 can be calculated. The same can be said for j=2,3,4,5. It should also be noted that the matrix on the left will be identical for all values of j allowing for it to be coded in as such:

Figure (4): Code for the ODHE showing the values being used for the next value of j

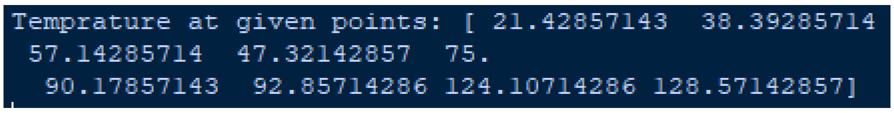


```
A=np.linalg.inv(np.array([[4,-1,0,0],
             [-1,4,-1,0],
             [0,-1,4,-1],
             [0,0,-1,4]]))
Tj0= np.array([np.sin(0.4*np.pi),
                   np.sin(0.2*np.pi)+np.sin(0.6*np.pi),
                   np.sin(0.4*np.pi)+np.sin(0.8*np.pi),
                   np.sin(0.6*np.pi)])
Tu1=np.matmul(A,Tj0)
u11,u21,u31,u41=Tu1
print("Tempratures at t= 0.04:",Tu1)

Tj1=np.array([u21,u31+u11,u41+u21,u31])
u12,u22,u32,u42=np.matmul(A,Tj1)
Tu2=np.array([u12,u22,u32,u42])
print("Tempratures at t= 0.08:",Tu2)

Tj2=np.array([u22,u12+u32,u22+u42,u32])
u13,u23,u33,u43=np.matmul(A,Tj2)
Tu3=np.array([u13,u23,u33,u43])
print("Tempratures at t= 0.12:",Tu3)

Tj3=np.array([u23,u13+u33,u23+u43,u33])
u14,u24,u34,u44=np.matmul(A,Tj3)
Tu4=np.array([u14,u24,u34,u44])
print("Tempratures at t= 0.16:",Tu4)

Tj4=np.array([u24,u14+u34,u24+u44,u34])
u15,u25,u35,u45=np.matmul(A,Tj4)
Tu5=np.array([u15,u25,u35,u45])
print("Tempratures at t= 0.20:",Tu5)
```

Using this data it can be plotted on a distance temperature graph and have multiple plots for the different times showing how it is cooling. This will be discussed in the results and conclusion

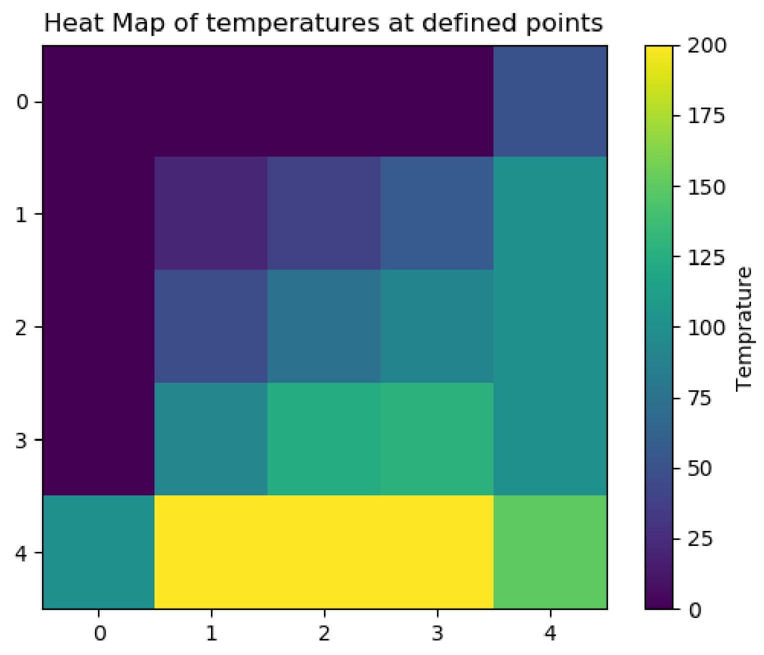
Results and Conclusion

For the TISSHE the following results were found:



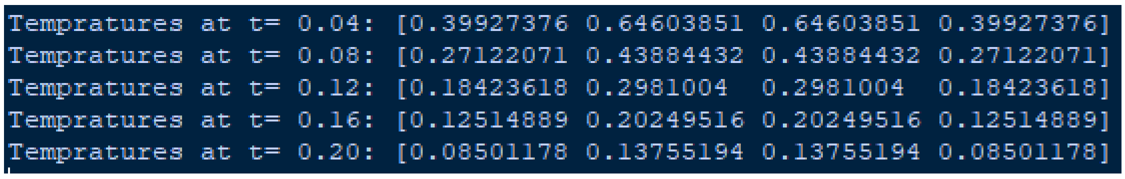
```
Temprature at given points: [ 21.42857143  38.39285714
 57.14285714  47.32142857  75.
  90.17857143  92.85714286 124.10714286 128.57142857]
```

This gave the following Heat map which has the boundary conditions added to it:



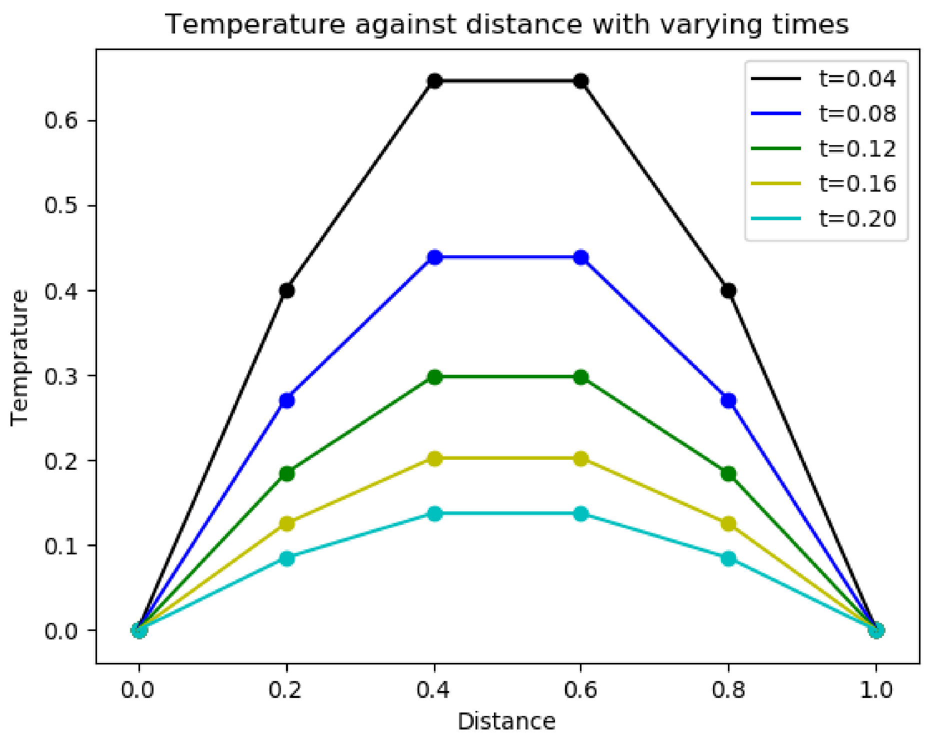
These results are the same as the answer. This means that to derive the formula for the PDE using forward and backward difference method is a correct way of finding the temperature at set points when only given boundary condition.

For the second scenario the ODHE the following results were found:



```
Tempratures at t= 0.04: [0.39927376 0.64603851 0.64603851 0.39927376]
Tempratures at t= 0.08: [0.27122071 0.43884432 0.43884432 0.27122071]
Tempratures at t= 0.12: [0.18423618 0.2981004  0.2981004  0.18423618]
Tempratures at t= 0.16: [0.12514889 0.20249516 0.20249516 0.12514889]
Tempratures at t= 0.20: [0.08501178 0.13755194 0.13755194 0.08501178]
```

Using these results gave the following graph which has the boundary conditions added:



These results are the same as the answer given showing that the central difference method and Crank-Nicolson method both work.

In both cases using forward and backward difference proves to be a relieable way to simplify PDEs in such a way that a computer can be coded in and give a high accuracy result to what the equation would give if done by hand.

<u>Critique</u>

In this project one of the hardest things to get my head around was the forward and backward difference as I had assumed that this needed to be coded in. I also struggled to find many useful resources in explaining what it was that needed to be done and how to do it. However once this obstacle was over come doing the second project was a lot easier and a lot less time consuming as I understood what needed to be done. If I had realised this sooner I would have been able to test more problems out such as the one dimensional wave equation. Or what happens if putting more data points in.

To improve this project if I had more time would have been to try the two scenarios out with more points required. I believe it would take more time process however it would make the data more accurate. I would also of liked to try adding in a system that allowed you to adjust the number of points you want to measure. This however is more of a programming problem than a numerical methods problem but would make finding out how changing the number of points affects the results given.

<u>Appendix</u>

Word count – 2170

<u>All of the Code Used</u>

```
import numpy as np
import matplotlib.pyplot as plt

#Heat equation
A1 = np.array([[4,-1,0,-1,0,0,0,0,0],
                [-1,4,-1,0,-1,0,0,0,0],
                [0,-1,4,0,0,-1,0,0,0],
                [-1,0,0,4,-1,0,-1,0,0],
                [0,-1,0,-1,4,-1,0,-1,0],
                [0,0,-1,0,-1,4,0,0,-1],
                [0,0,0,-1,0,0,4,-1,0],
                [0,0,0,0,-1,0,-1,4,-1],
                [0,0,0,0,0,-1,0,-1,4]])

b= np.array([0,0,100,0,0,100,200,200,300])

T = np.matmul(np.linalg.inv(A1),b)
print("Temprature at given points:",T)
Z = np.array([[0,0,0,0,50],
                [0,T[0],T[1],T[2],100],
                [0,T[3],T[4],T[5],100],
                [0,T[6],T[7],T[8],100],
                [100,200,200,200,150]])

plt.imshow(Z)
plt.colorbar(label="Temprature")
plt.title('Heat Map of temperatures at defined points')
plt.show()


#1-D wave equation
A=np.linalg.inv(np.array([[4,-1,0,0],
             [-1,4,-1,0],
             [0,-1,4,-1],
             [0,0,-1,4]]))
Tj0= np.array([np.sin(0.4*np.pi),
                   np.sin(0.2*np.pi)+np.sin(0.6*np.pi),
                   np.sin(0.4*np.pi)+np.sin(0.8*np.pi),
                   np.sin(0.6*np.pi)])
Tu1=np.matmul(A,Tj0)
u11,u21,u31,u41=Tu1
print("\n"+"Tempratures at t= 0.04:",Tu1)

Tj1=np.array([u21,u31+u11,u41+u21,u31])
u12,u22,u32,u42=np.matmul(A,Tj1)
Tu2=np.array([u12,u22,u32,u42])
print("Tempratures at t= 0.08:",Tu2)

Tj2=np.array([u22,u12+u32,u22+u42,u32])
u13,u23,u33,u43=np.matmul(A,Tj2)
Tu3=np.array([u13,u23,u33,u43])
print("Tempratures at t= 0.12:",Tu3)

Tj3=np.array([u23,u13+u33,u23+u43,u33])
u14,u24,u34,u44=np.matmul(A,Tj3)
Tu4=np.array([u14,u24,u34,u44])
```

```
print("Tempratures at t= 0.16:",Tu4)

Tj4=np.array([u24,u14+u34,u24+u44,u34])
u15,u25,u35,u45=np.matmul(A,Tj4)
Tu5=np.array([u15,u25,u35,u45])
print("Tempratures at t= 0.20:",Tu5)

#The border points are =0 meaning that
#appending them on will give the correct
#result when plotted
u1=np.append([0],np.append(Tu1,[0]))
u2=np.append([0],np.append(Tu2,[0]))
u3=np.append([0],np.append(Tu3,[0]))
u4=np.append([0],np.append(Tu4,[0]))
u5=np.append([0],np.append(Tu5,[0]))

x=np.arange(0,1.2,0.2)
plt.plot(x,u1,label='t=0.04',c='k')
plt.scatter(x,u1,c='k')

plt.plot(x,u2,label='t=0.08',c='b')
plt.scatter(x,u2,c='b')

plt.plot(x,u3,label='t=0.12',c='g')
plt.scatter(x,u3,c='g')

plt.plot(x,u4,label='t=0.16',c='y')
plt.scatter(x,u4,c='y')

plt.plot(x,u5,label='t=0.20',c='c')
plt.scatter(x,u5,c='c')

plt.title("Temperature against distance with varying times")
plt.ylabel("Temprature")
plt.xlabel("Distance")
plt.legend(loc=0)
plt.show()
```

<u>Software Resources</u>

1) Develpoment Enviroment: Pyhton 3.6 (32-bit) IDLE (Integrated Development Eviroment)
2) NimPy and MatPlotLib packages for Python

<u>Bibliography</u>

- Q4, Exercise 6, Numerical methods, Barry Smalley, School of Chemical and Physical sciences, Keele University
- Chapter 21, Advanced Engineering Mathematic Tenth Edition, Erwin Kreszig
- blogs.ubc.ca/mrahmani/file/2019/01/Numerical_methods.pdf