**AERO 430 – Exam 1**



Antonio Diaz ‘22

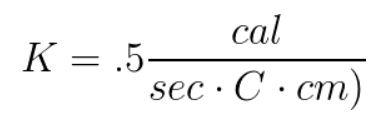
**Due 03/04/2020**

**1)** **Weak formulation of Finite Element Model**

Case 1 Boundary conditions: T(0) = 0, T(L) = 100, T­Ambient­ = 0

Case 2 Boundary conditions: T’(0) = h / k(T­0-TAmbient), T(L) = 100, T­Ambient­ = 0

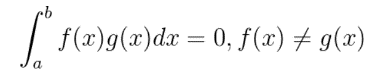
Constants are defined as follows:



Length = 1 cm

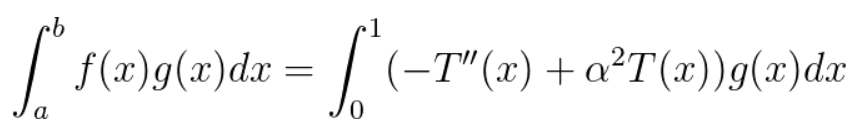
Radius = .1 cm

Cooling fin second order differential equation:

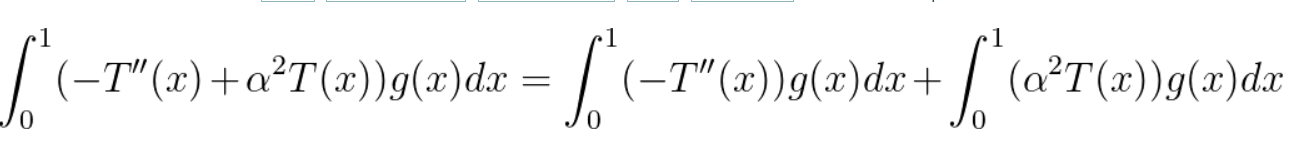


Orthogonality condition:

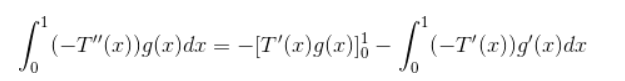
Substituting f(x) and g(x) with our second order differential equation and a weighting function, respectively from our boundaries 0 to 1:



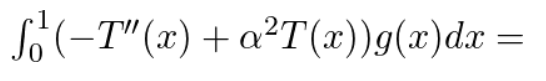
This can be separated into 2 integrals:

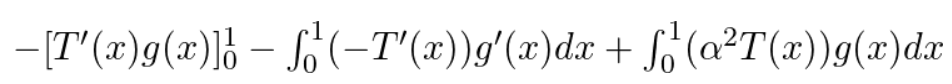


Using integration by parts, this can be reduced:

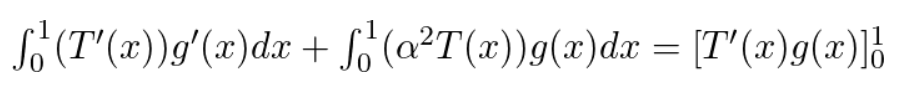


Plugging integration by parts back into equation:





Finally, setting this to 0 from orthogonality, it is reduced to:



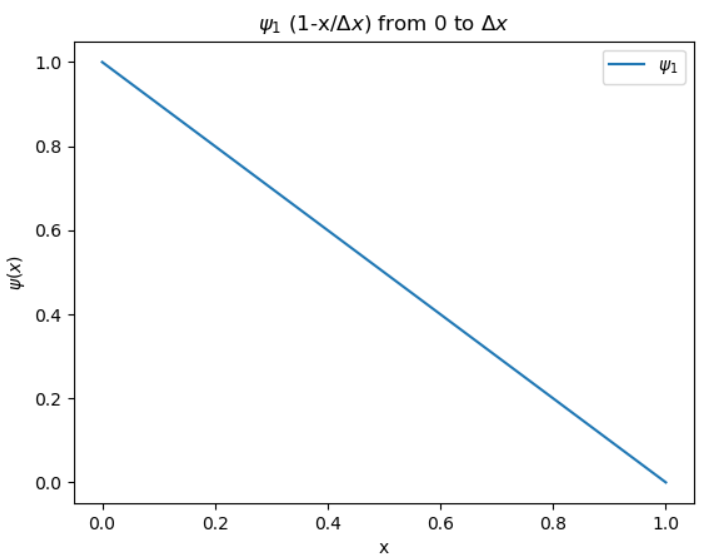
**2) The hierarchical, elementwise system:**

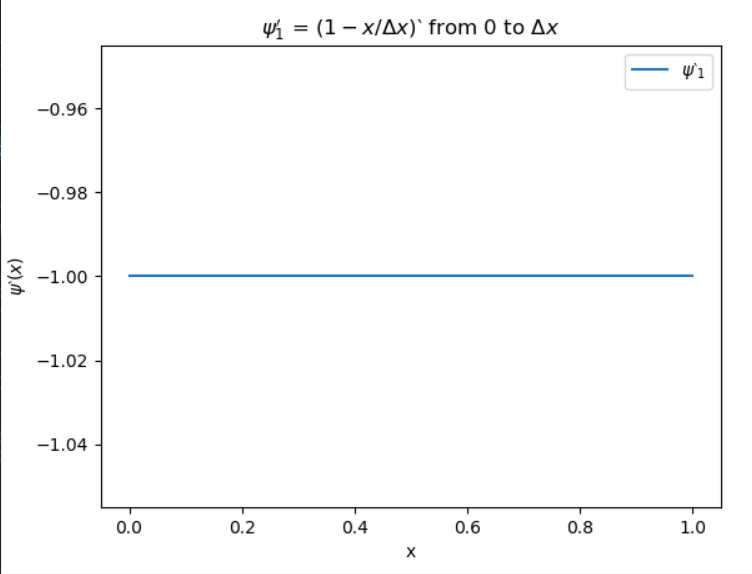
From the weak formulation, a test function for g(x) is needed. The test functions cover a continuous boundary 0 < x < 1, with nodes 0, 1, 2, 3, N. Each hierarchical basis function will be applied between elements.

Graphing each psi function order and it’s respective derivative:

psi[1] = 1 - 1/delta\_x \* x\_i

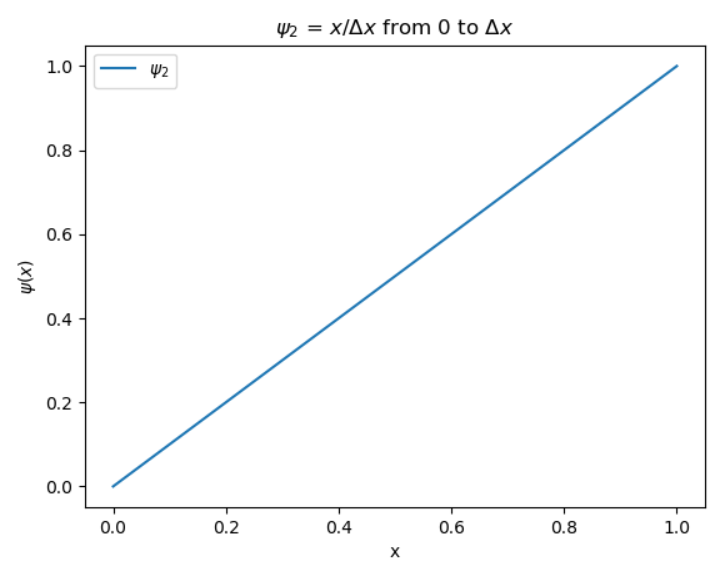
derivative\_psi[1] = -1/delta\_x+(x\_i\*0)

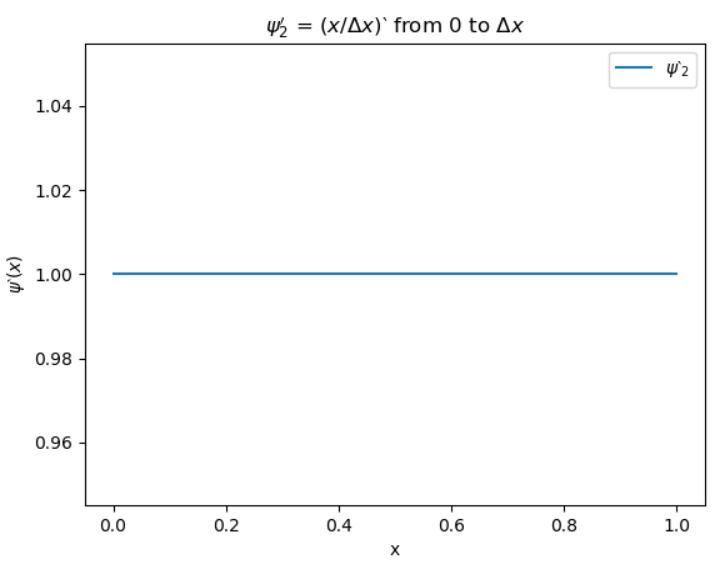




psi[2] = 1/delta\_x \* x\_i

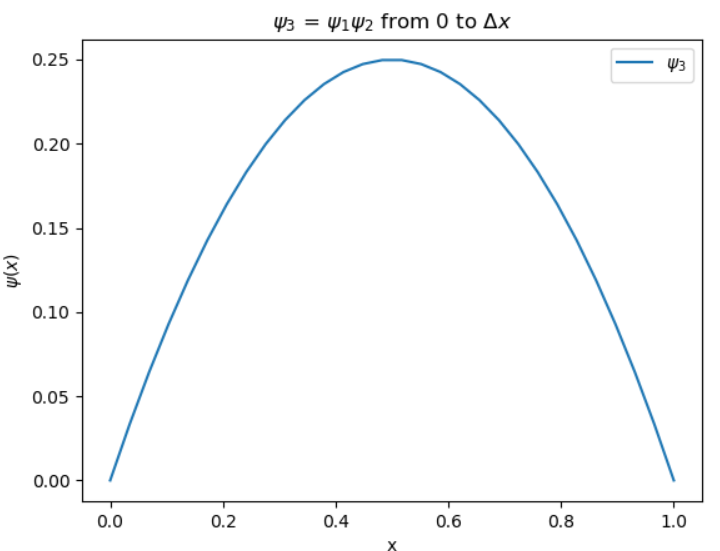
derivative\_psi[2] = 1/delta\_x+(x\_i\*0)

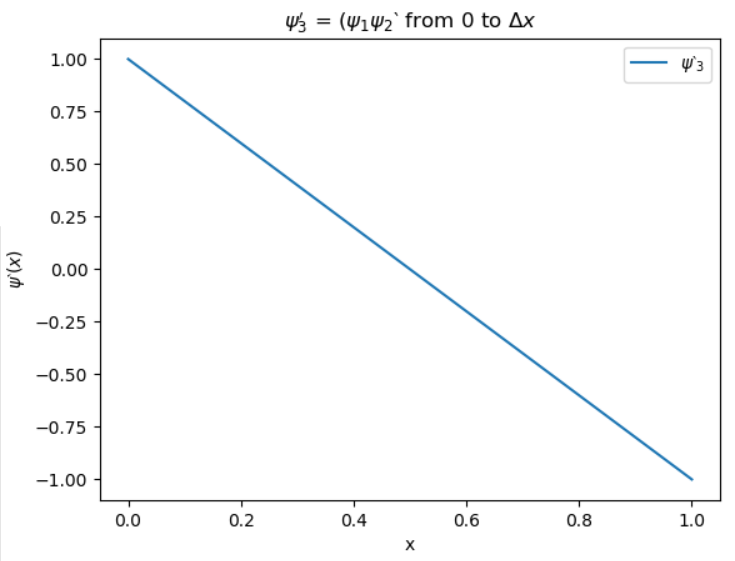




psi[3] = (1/delta\_x) \* x\_i - (1/delta\_x\*\*2) \* x\_i\*\*2

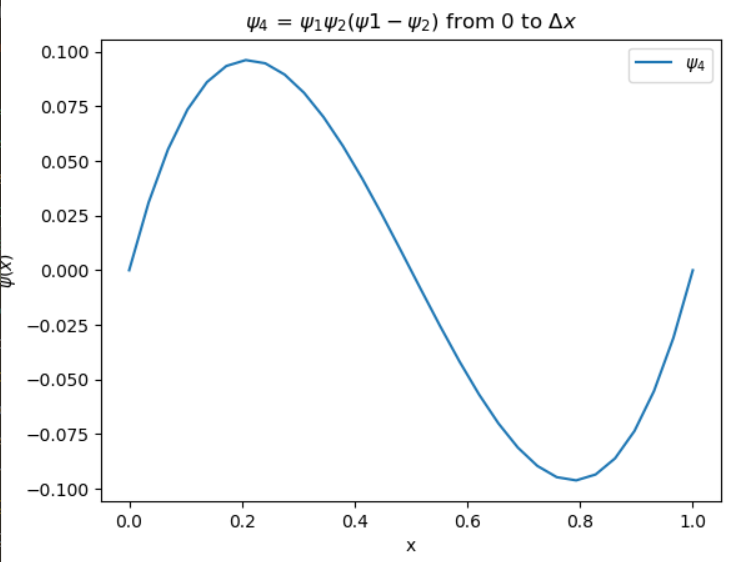
derivative\_psi[3] = 1/delta\_x-2\*x\_i/(delta\_x\*\*2)

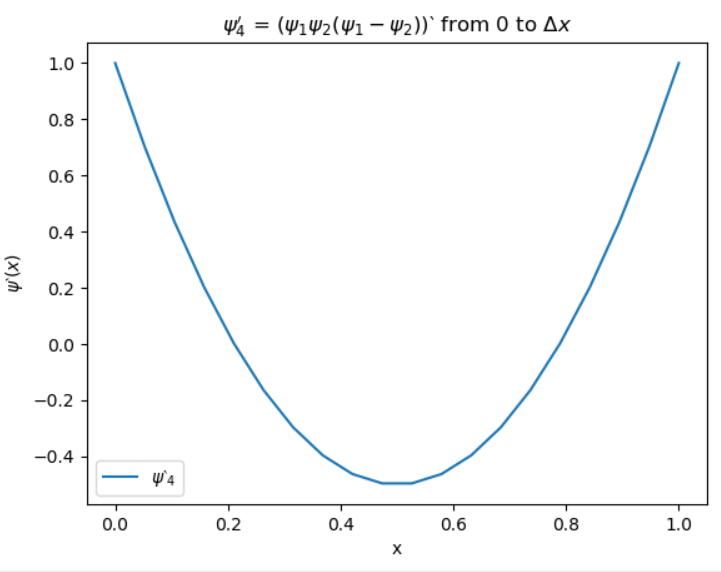




psi[4] = 2\*x\_i\*\*3/(delta\_x\*\*3) - 3\*x\_i\*\*2/(delta\_x\*\*2) + x\_i/delta\_x

derivative\_psi[4] = 6\*x\_i\*\*2/(delta\_x\*\*3) - 6\*x\_i/(delta\_x\*\*2) + 1/delta\_x

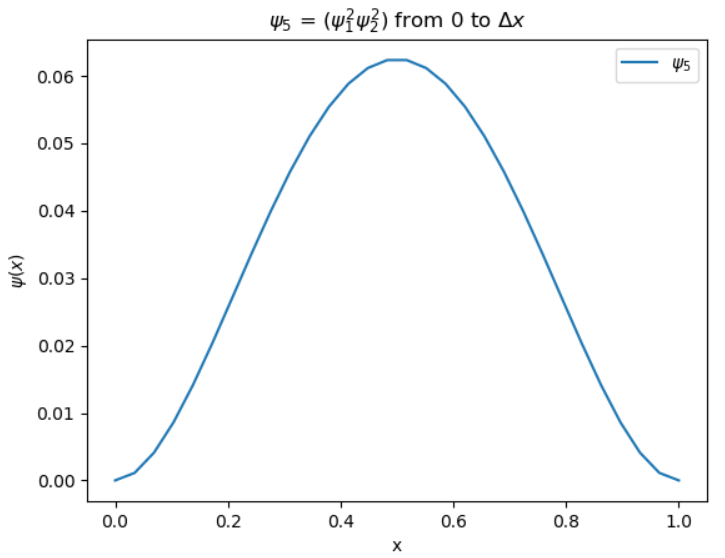


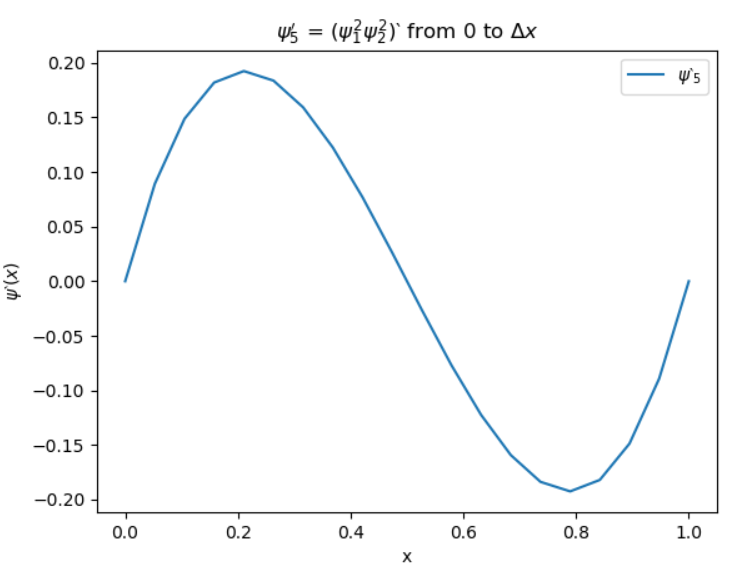


psi[5] = x\_i\*\*4/(delta\_x\*\*4)-2\*x\_i\*\*3/(delta\_x\*\*3) + x\_i\*\*2/(delta\_x\*\*2)

derivative\_psi[5]=

 4\*x\_i\*\*3/(delta\_x\*\*4) - 6\*x\_i\*\*2/(delta\_x\*\*3)+ 2\*x\_i/(delta\_x\*\*2)

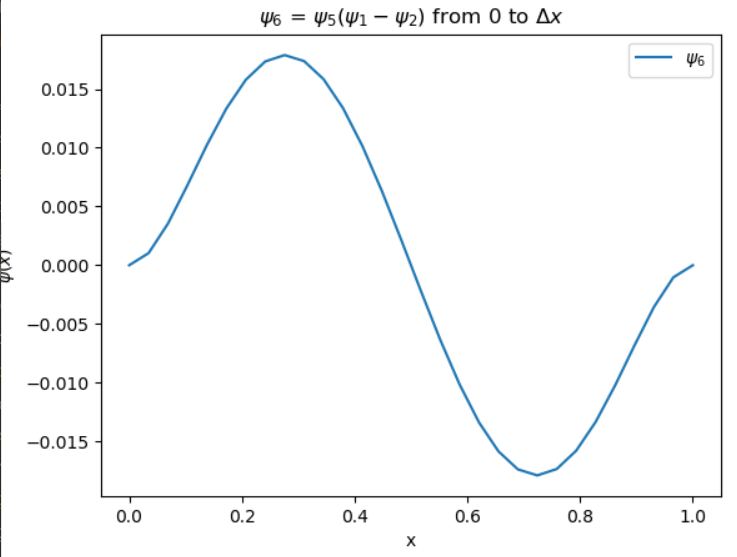


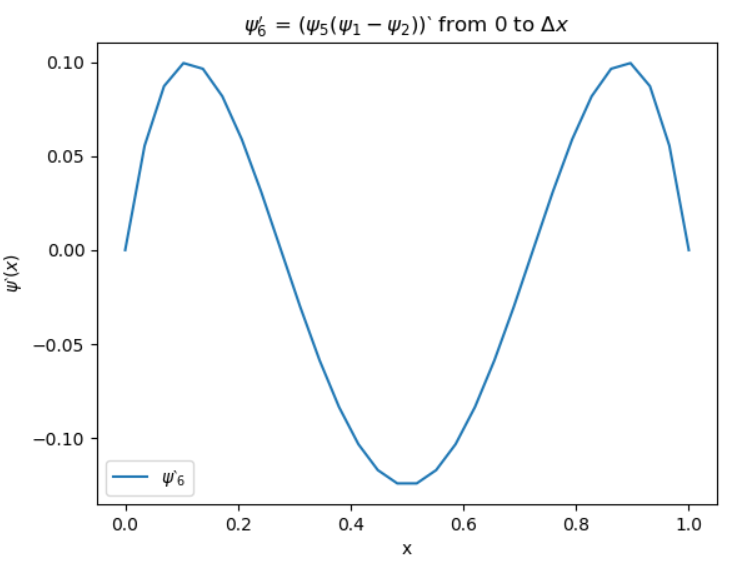


psi[6]=

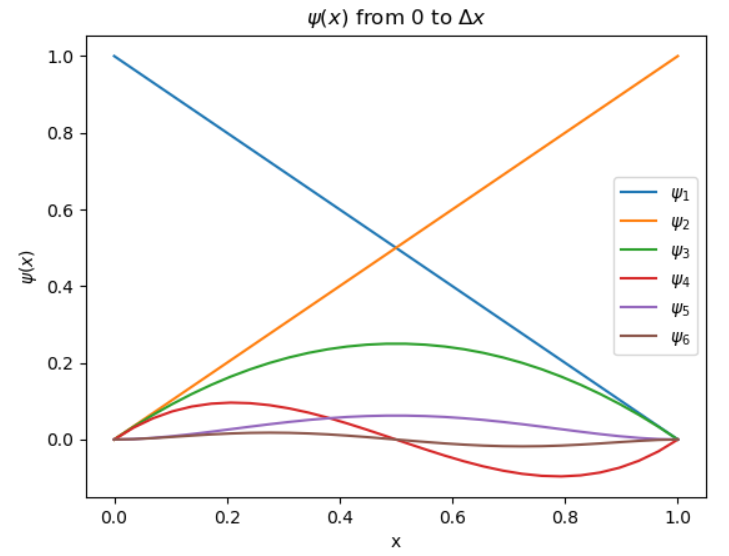
x\_i\*\*2/(delta\_x\*\*2) - 4\*x\_i\*\*3/(delta\_x\*\*3) + 5\*x\_i\*\*4/(delta\_x\*\*4) - 2\*x\_i\*\*5/(delta\_x\*\*5)

derivative\_psi[6] = -10\*x\_i\*\*4/(delta\_x\*\*5) + 20\*x\_i\*\*3/(delta\_x\*\*4)-12\*x\_i\*\*2/(delta\_x\*\*3)+2\*x\_i/(delta\_x\*\*2)

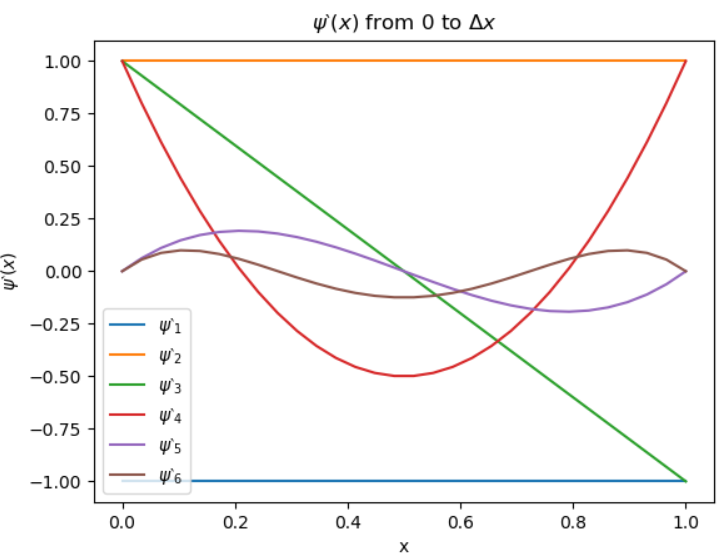




All overlapping Psi functions:

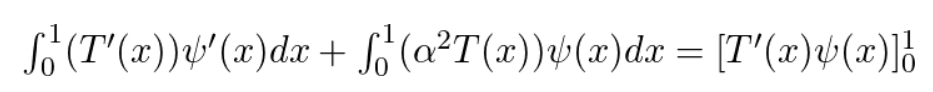


All overlapping derivatives of Psi functions:

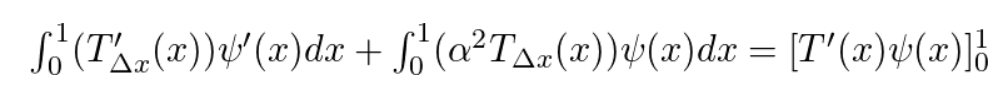


**3) Element equations using weak formulation with hierarchical basis function:**

Replacing the previously derived g(x) function with psi functions from the hierarchical basis function leads to:

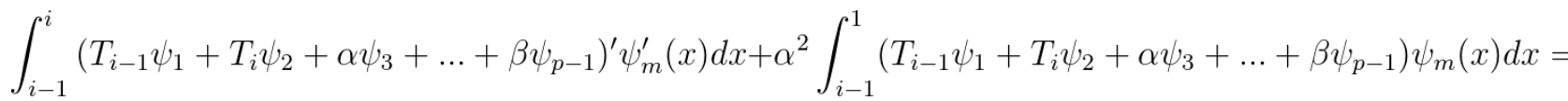


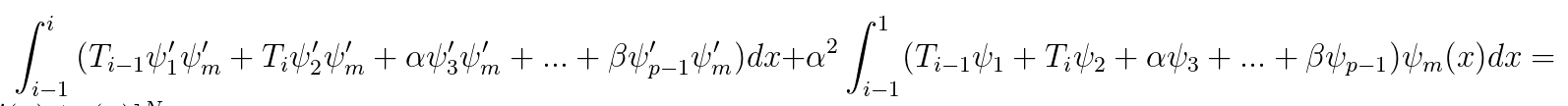
Applying to a discretized T:

,

where

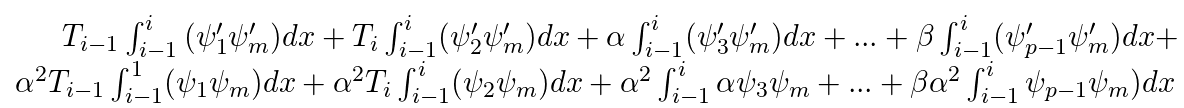
Plugging in discretized T:







Expansion to the alpha term can be simplified by factoring out the psi terms:

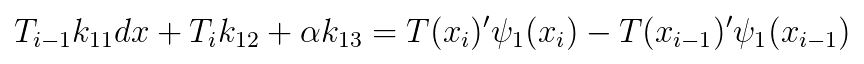


A system of equations can now be setup for the first element by setting

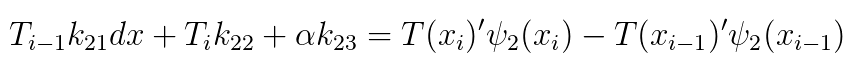
dx:

For p = 2, the following system of equations is formed:

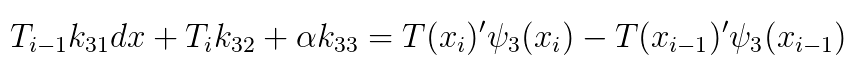
m = 1:



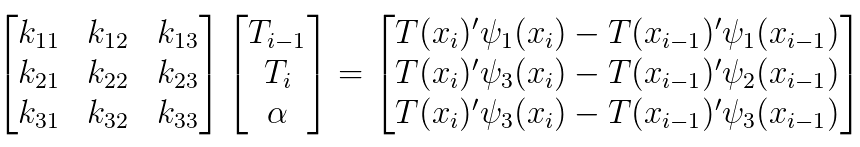
m = 2:



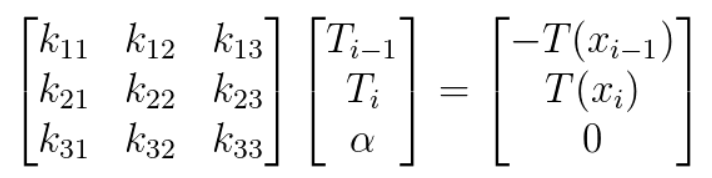
m = 3:



This can be reorganized into matrix form for the element (“local stiffness”) matrix:

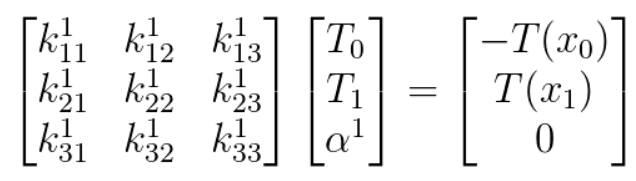


Which can be further simplified using our basis function:

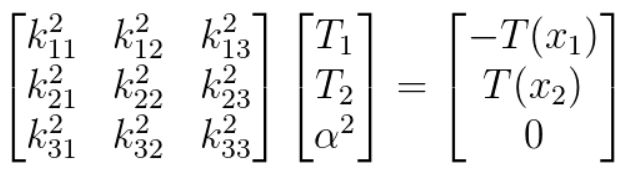


**4. Assembly and condensation of element matrix into the global stiffness matrix:**

Two consecutive element matrices result in the following:

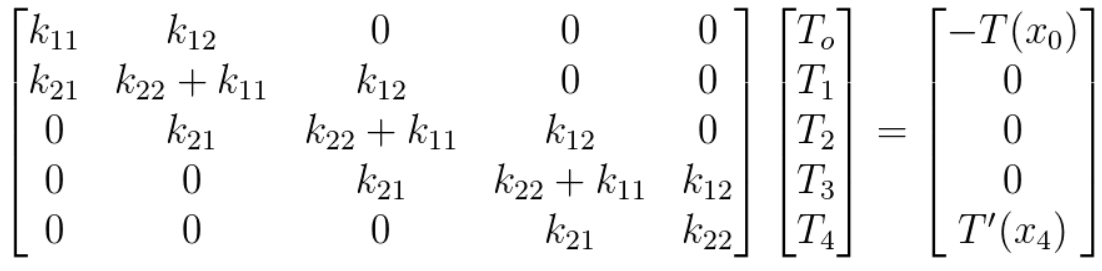


:



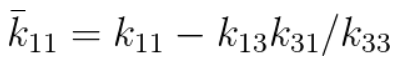
To make use of the element matrices, the elements are combined in a global matrix dependent on the number of nodes that are formed from 2x2 matrices that are formed with condensation.

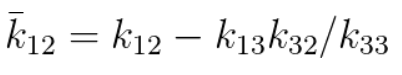
p = 1, 4 elements:

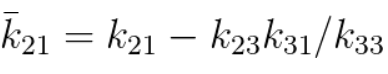


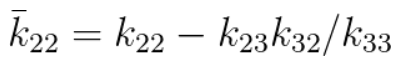
p = 2, 4 elements:

Condensed matrix: Using coefficients for each inner node, each 3x3 matrix can be condensed into a 2x2 matrix, then assembled in a similar way to p = 1, 4 elements:





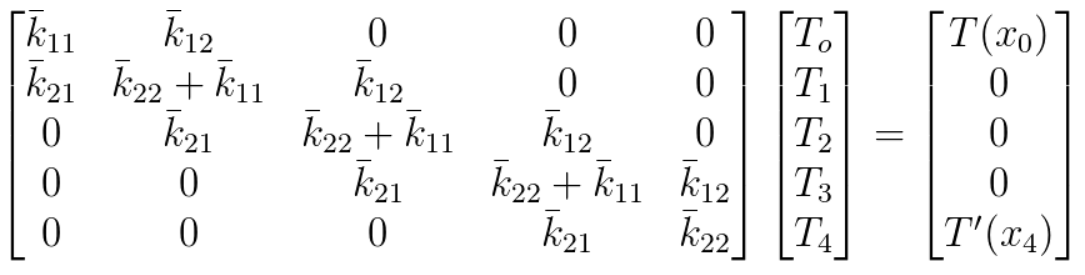




\*these values are found with Kramer’s rule and can be replicated using Schur’s complement derived for p = 5

These values are used to reduce each node 3x3 matrix to 2x2 matrices, and assembled into the following:

for p =2, 4 elements



p = 3 condensation. Out of convenience, the next coefficients will be shown as variables in Python. The method of condensation for each new coefficient starts to become complex very fast:

k\_11\_bar = k\_11 + k\_13\*(-k\_31\*k\_44 + k\_41\*k\_34)/(k\_33\*k\_44 - k\_34\*k\_43) +

k\_14\*(-k\_41\*k\_33+k\_31\*k\_43)/(k\_33\*k\_44-k\_34\*k\_43)

k\_12\_bar = k\_12 + k\_13\*(-k\_32\*k\_44 + k\_42\*k\_34)/(k\_33\*k\_44 - k\_34\*k\_43) +

k\_14\*(-k\_42\*k\_33+k\_32\*k\_43)/(k\_33\*k\_44-k\_34\*k\_43)

k\_21\_bar = k\_21 + k\_23\*(-k\_31\*k\_44 + k\_41\*k\_34)/(k\_33\*k\_44 - k\_34\*k\_43)+

k\_24\*(-k\_41\*k\_33+k\_31\*k\_43)/(k\_33\*k\_44-k\_34\*k\_43)

k\_22\_bar = k\_22 + k\_23\*(-k\_32\*k\_44 + k\_42\*k\_34)/(k\_33\*k\_44 - k\_34\*k\_43) +

 k\_24\*(-k\_42\*k\_33+k\_32\*k\_43)/(k\_33\*k\_44-k\_34\*k\_43)

p = 4 condensation coefficients

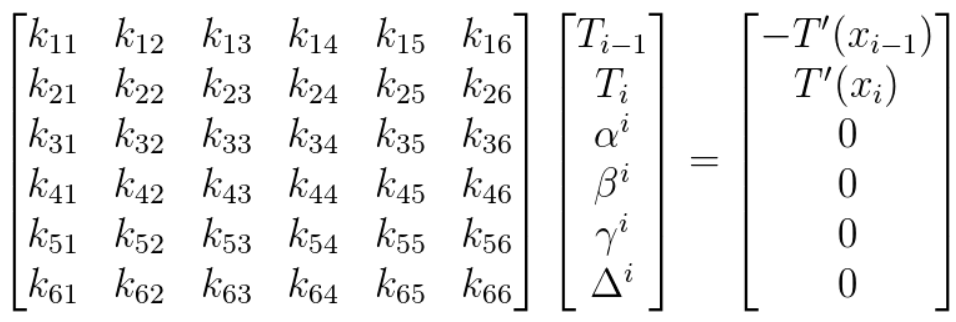
k\_11\_bar = k\_11 + k\_13 \* (-k\_31\*k\_44\*k\_55 - k\_51\*k\_34\*k\_45 - k\_41\*k\_35\*k\_54 + k\_51\*k\_44\*k\_35 + k\_31\*k\_54\*k\_45 + k\_41\*k\_34\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_14\*(-k\_41\*k\_33\*k\_55 - k\_31\*k\_53\*k\_45 - k\_51\*k\_35\*k\_43 + k\_41\*k\_53\*k\_35 + k\_51\*k\_45\*k\_33 + k\_31\*k\_43\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_15\*(-k\_51\*k\_33\*k\_44 - k\_41\*k\_53\*k\_34 - k\_31\*k\_43\*k\_54 + k\_31\*k\_53\*k\_44 + k\_41\*k\_54\*k\_33 + k\_51\*k\_43\*k\_34)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34)

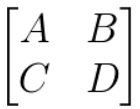
k\_12\_bar = k\_12 + k\_13 \* (-k\_32\*k\_44\*k\_55 - k\_52\*k\_34\*k\_45 - k\_42\*k\_35\*k\_54 + k\_52\*k\_44\*k\_35 + k\_32\*k\_54\*k\_45 + k\_42\*k\_34\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_14\*(-k\_42\*k\_33\*k\_55 - k\_32\*k\_53\*k\_45 - k\_52\*k\_35\*k\_43 + k\_42\*k\_53\*k\_35 + k\_52\*k\_45\*k\_33 + k\_32\*k\_43\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_15\*(-k\_52\*k\_33\*k\_44 - k\_42\*k\_53\*k\_34 - k\_32\*k\_43\*k\_54 + k\_32\*k\_53\*k\_44 + k\_42\*k\_54\*k\_33 + k\_52\*k\_43\*k\_34)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34)

k\_21\_bar = k\_21 + k\_23 \* (-k\_31\*k\_44\*k\_55 - k\_51\*k\_34\*k\_45 - k\_41\*k\_35\*k\_54 + k\_51\*k\_44\*k\_35 + k\_31\*k\_54\*k\_45 + k\_41\*k\_34\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_24\*(-k\_41\*k\_33\*k\_55 - k\_31\*k\_53\*k\_45 - k\_51\*k\_35\*k\_43 + k\_41\*k\_53\*k\_35 + k\_51\*k\_45\*k\_33 + k\_31\*k\_43\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_25\*(-k\_51\*k\_33\*k\_44 - k\_41\*k\_53\*k\_34 - k\_31\*k\_43\*k\_54 + k\_31\*k\_53\*k\_44 + k\_41\*k\_54\*k\_33 + k\_51\*k\_43\*k\_34)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34)

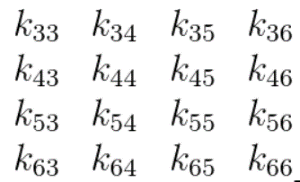
k\_22\_bar = k\_22 + k\_23 \* (-k\_32\*k\_44\*k\_55 - k\_52\*k\_34\*k\_45 - k\_42\*k\_35\*k\_54 + k\_52\*k\_44\*k\_35 + k\_32\*k\_54\*k\_45 + k\_42\*k\_34\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_24\*(-k\_42\*k\_33\*k\_55 - k\_32\*k\_53\*k\_45 - k\_52\*k\_35\*k\_43 + k\_42\*k\_53\*k\_35 + k\_52\*k\_45\*k\_33 + k\_32\*k\_43\*k\_55)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34) + k\_25\*(-k\_52\*k\_33\*k\_44 - k\_42\*k\_53\*k\_34 - k\_32\*k\_43\*k\_54 + k\_32\*k\_53\*k\_44 + k\_42\*k\_54\*k\_33 + k\_52\*k\_43\*k\_34)/(k\_33\*k\_44\*k\_55 + k\_34\*k\_45\*k\_53 + k\_35\*k\_43\*k\_54 - k\_53\*k\_44\*k\_35 - k\_54\*k\_45\*k\_33 - k\_55\*k\_43\*k\_34)

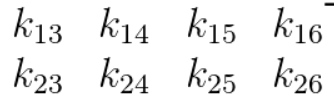
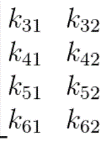
p = 5 condensation and assembly using Schur’s complement:





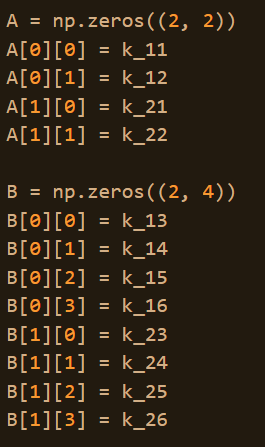
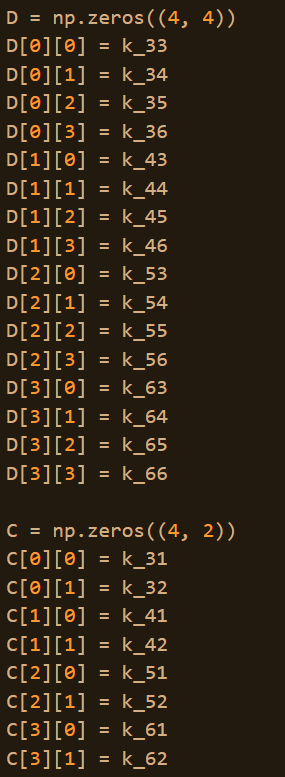
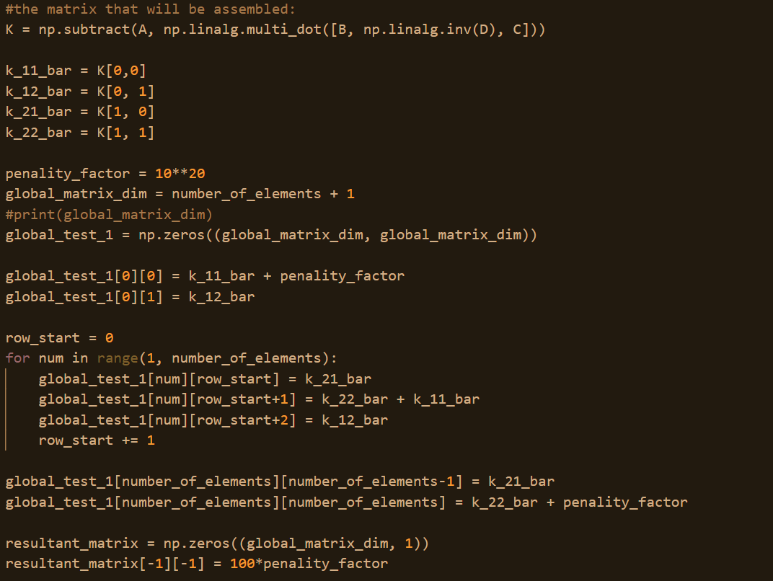
For a matrix M, Schur’s complement is equal to

This can then be extended to the inner nodal matrix by setting the following:



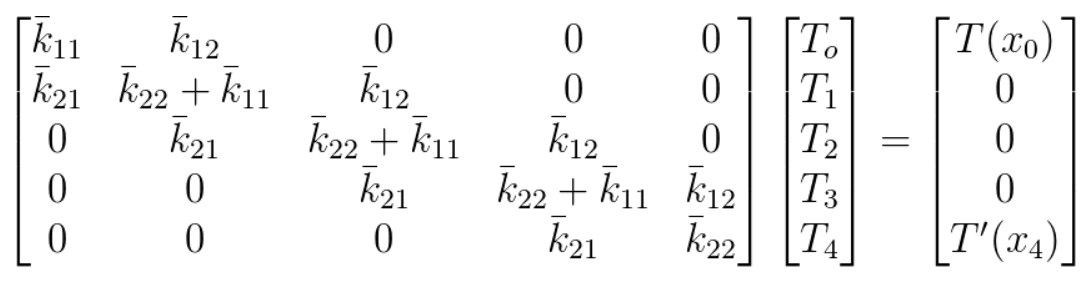
A = , B = C = D =

The complement is equal to a 2x2 matrix that is then assembled in the global matrix, with some example code below:

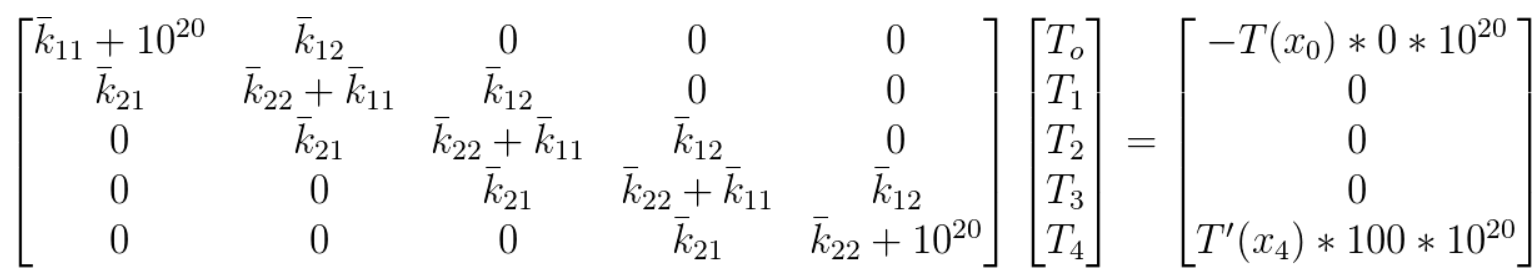


**5. Finding temperature distribution from global matrix with a penalty method**

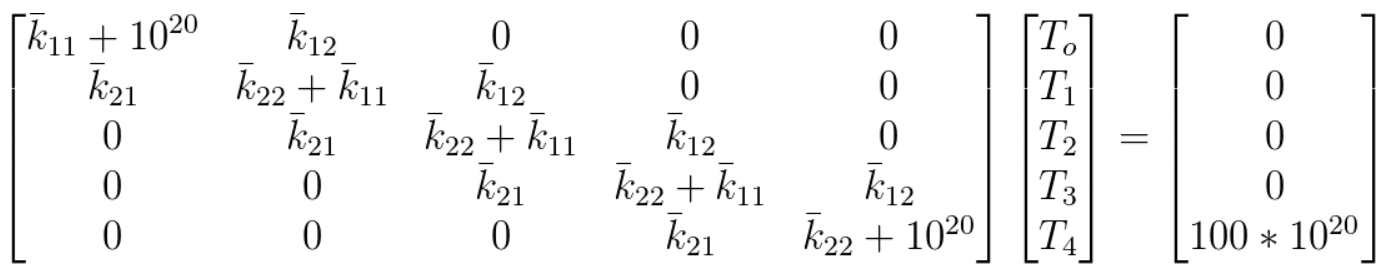
The global matrix operation leaves us with the following equation for p =2, 4 elements:



To diminish the effect of the rates of temperature at the end for case 1, a constant is applied to the boundary conditions, forcing the matrix to reduce the effect of unnecessary values in the boundary condition:

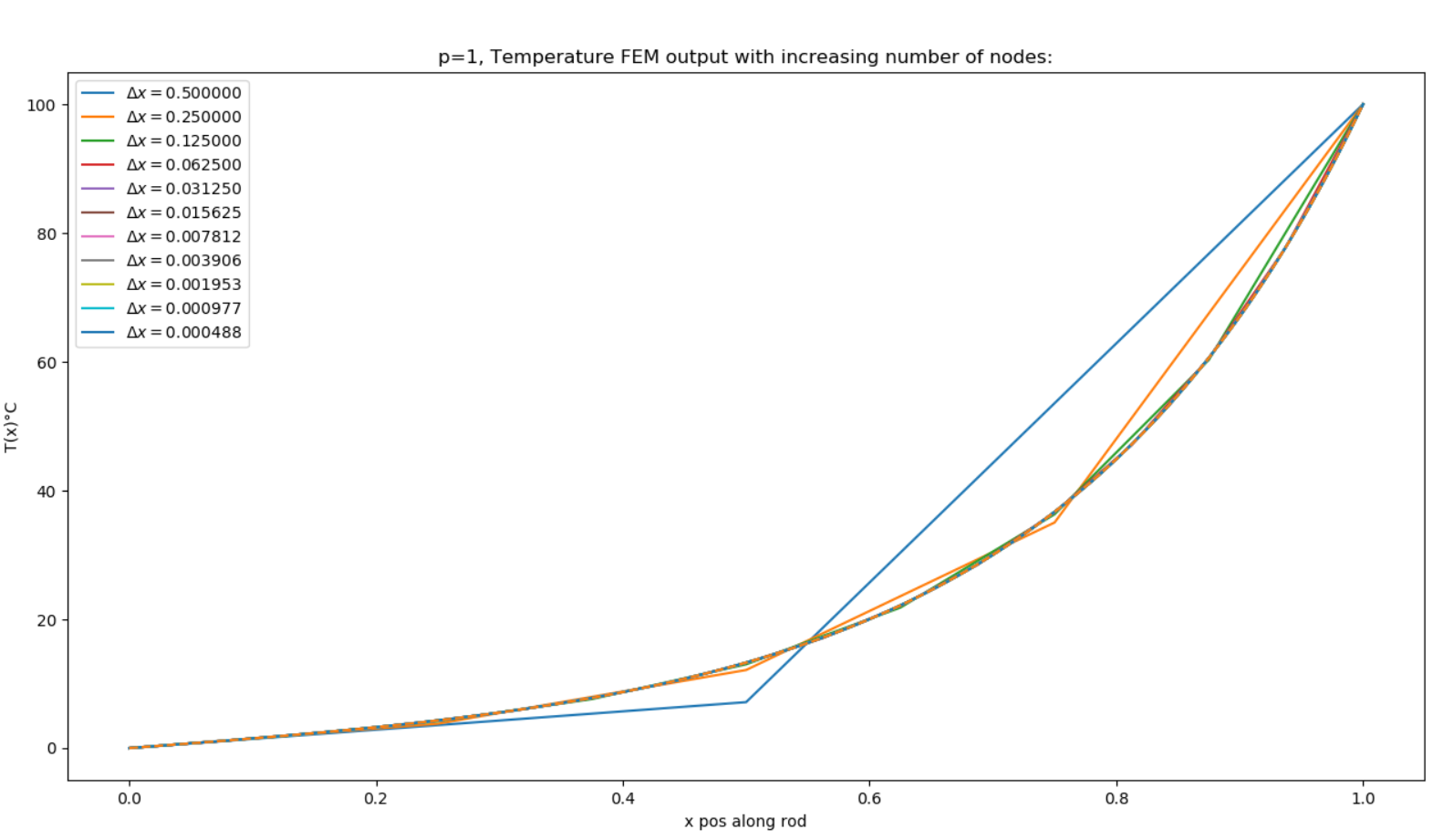


Relative to the large multiplication factors, the T’(x) values diminish, becoming:

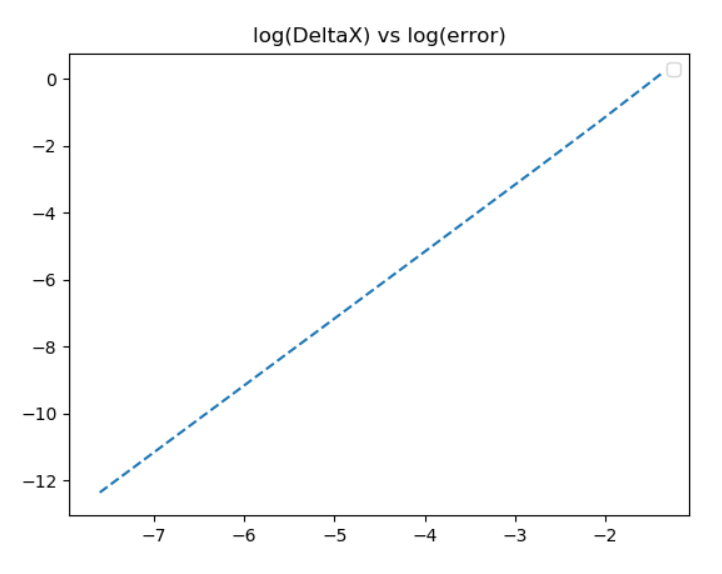


**6. Results and convergence:**

p = 1, case 1:

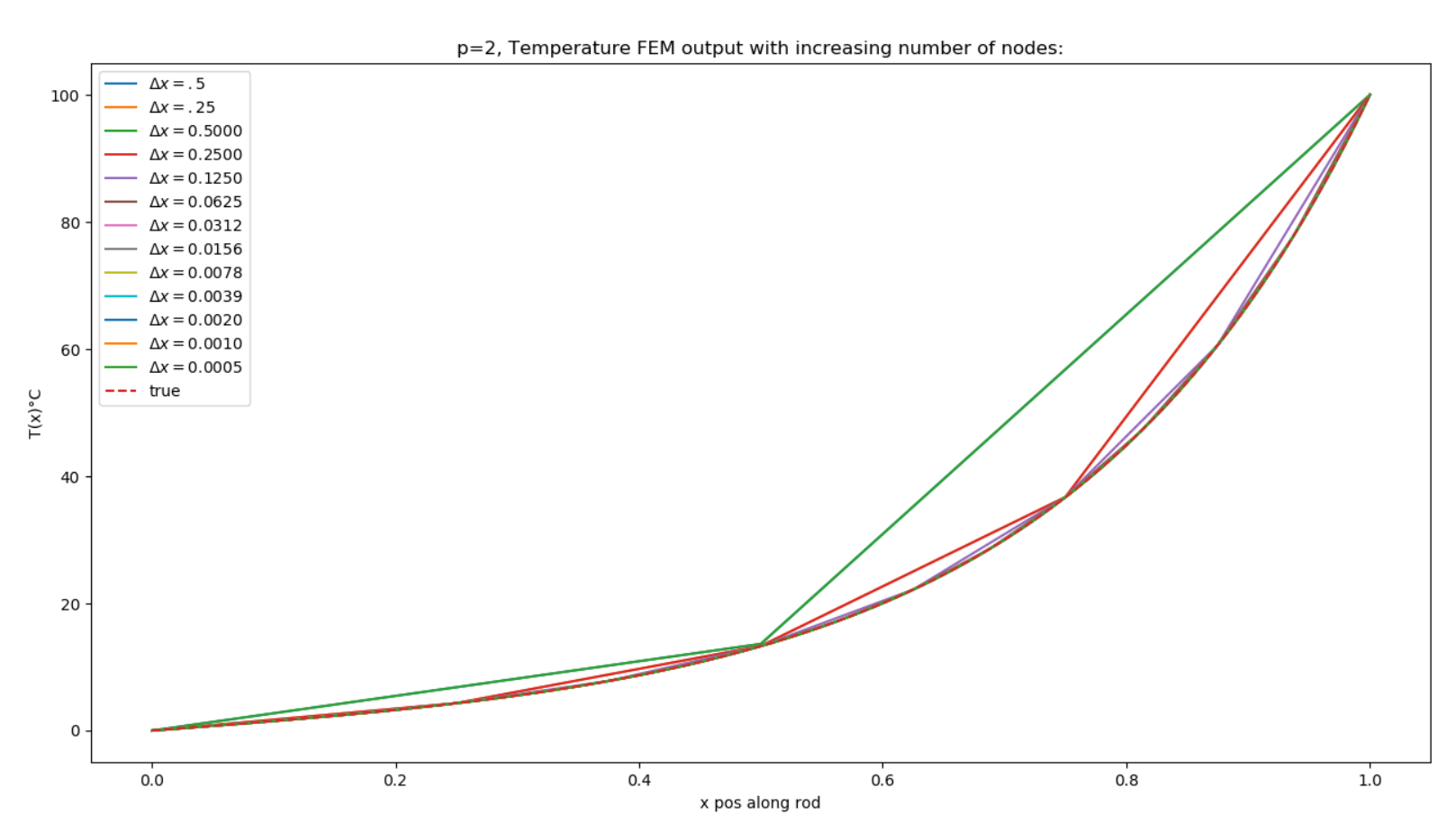


p = 1, case 1 convergence plot:

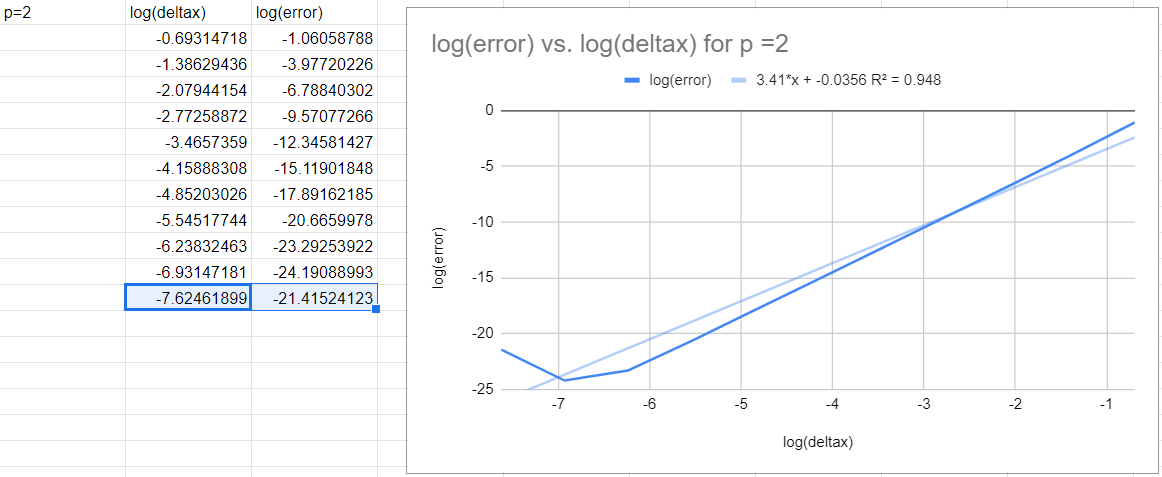


\*after plotting values on google sheets, convergence found to be .5 with an value of 1.0

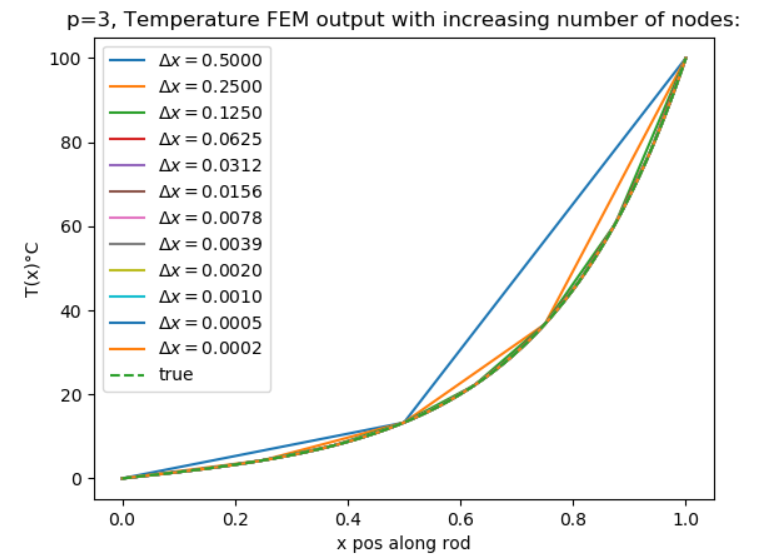
p=2, case 1:



p = 2, case 1, convergence plot, converges with beta of approximately 3.41:

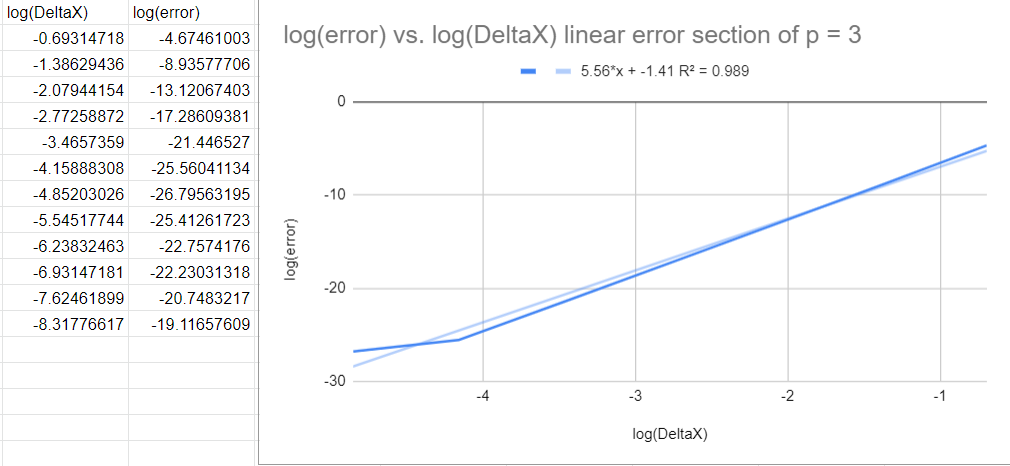


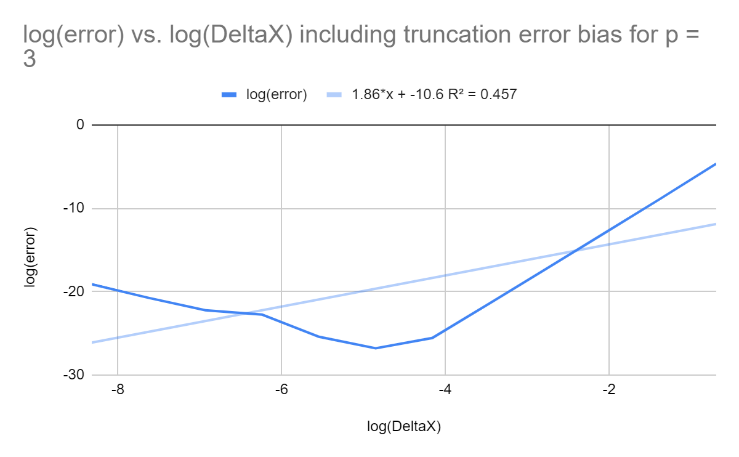
p = 3, case 1:



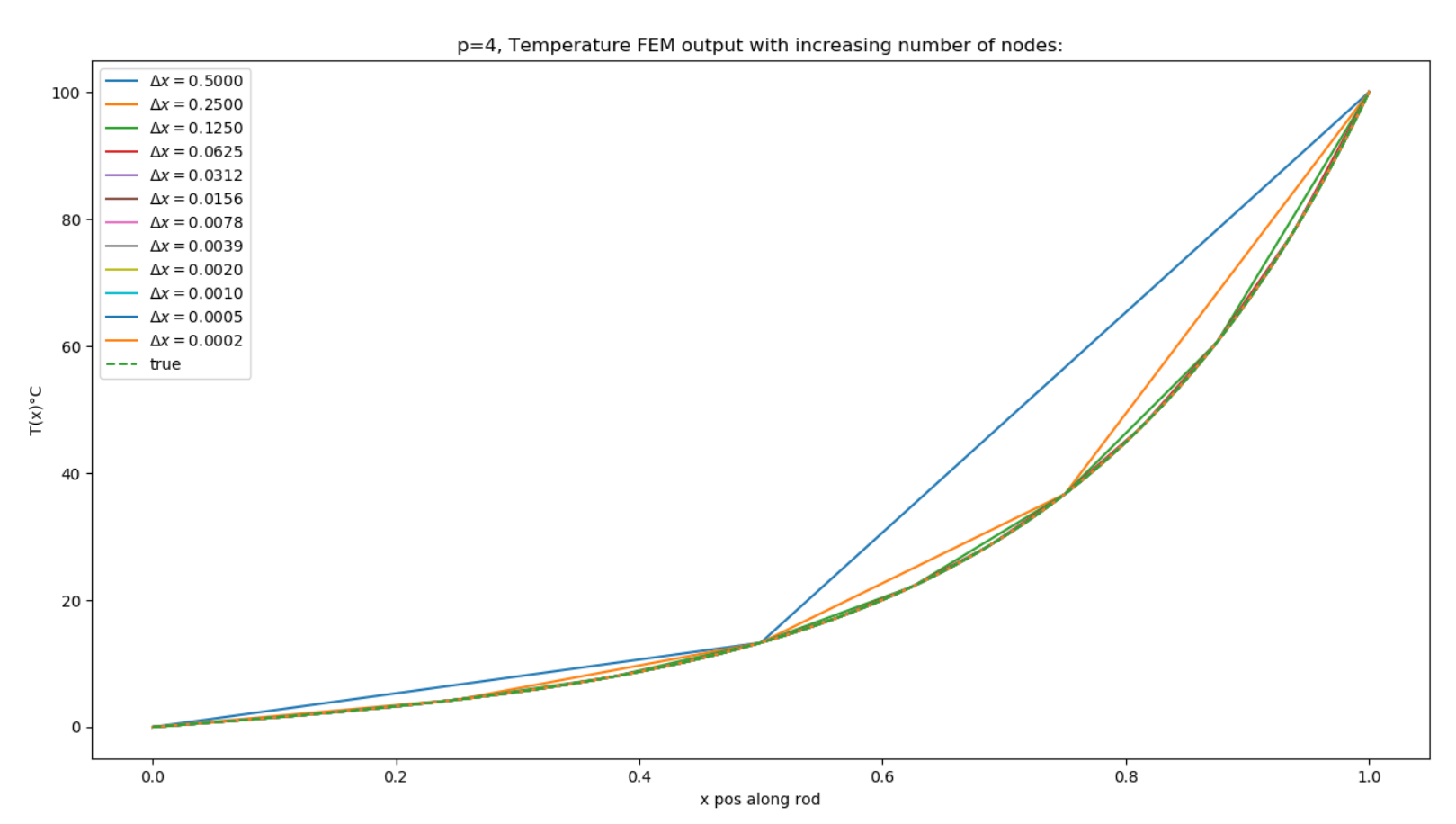
p = 3, case 1, convergence plot and data:

Beta approximately equal to 5.56

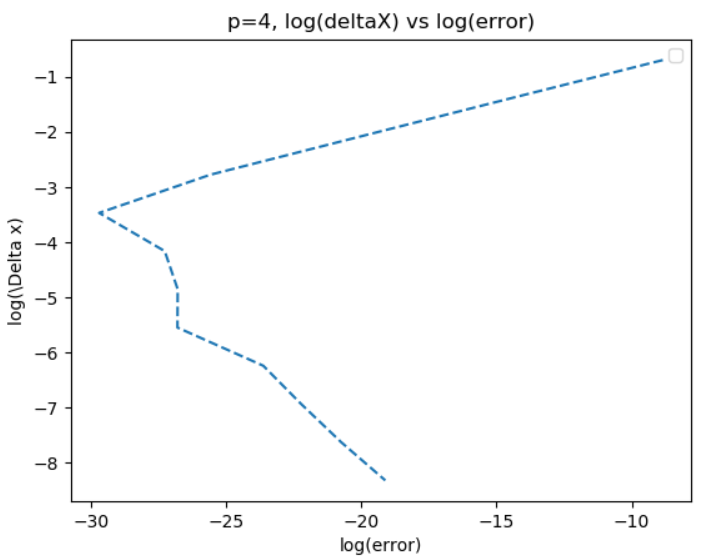




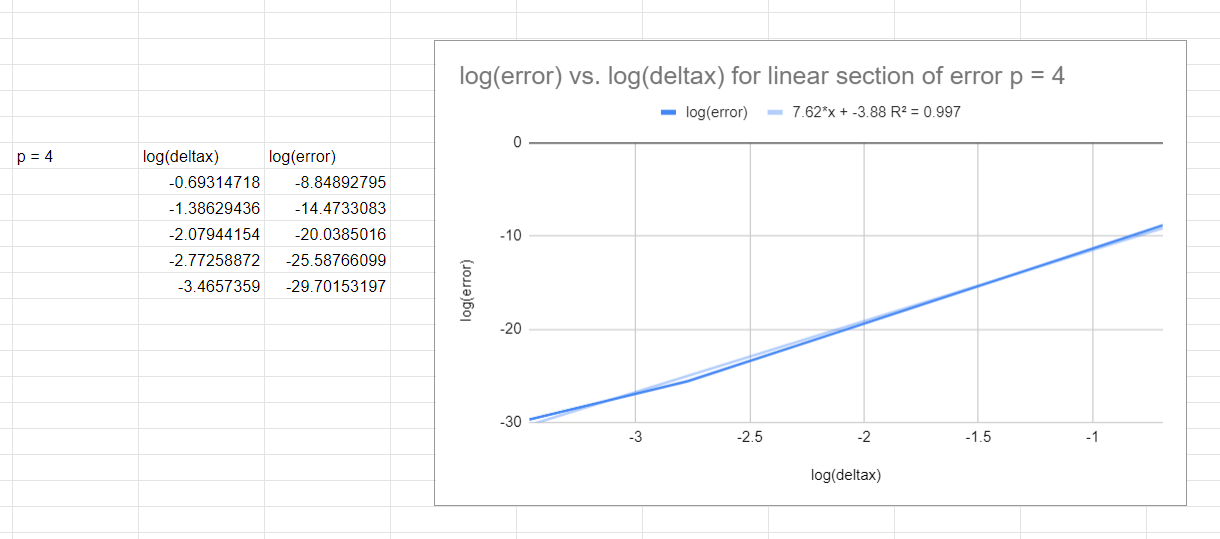
p = 4, case 1:



p = 4, case 1, convergence plot and data:

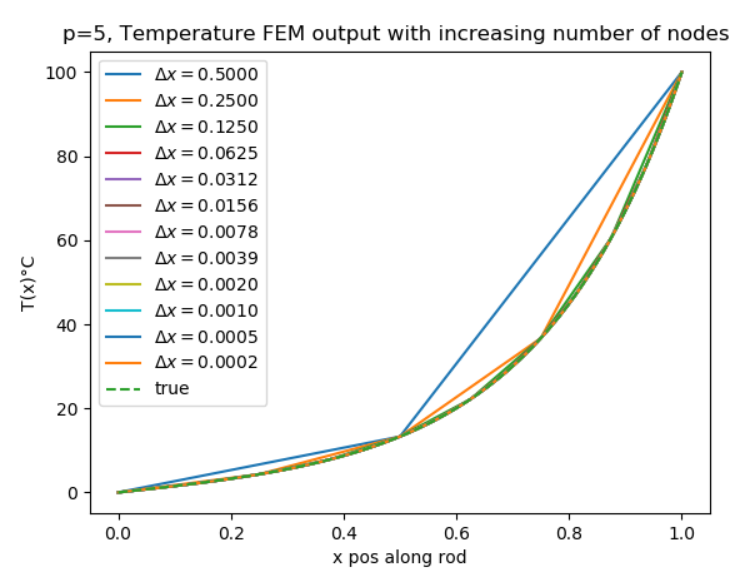


Beta approximately equal to 7.62

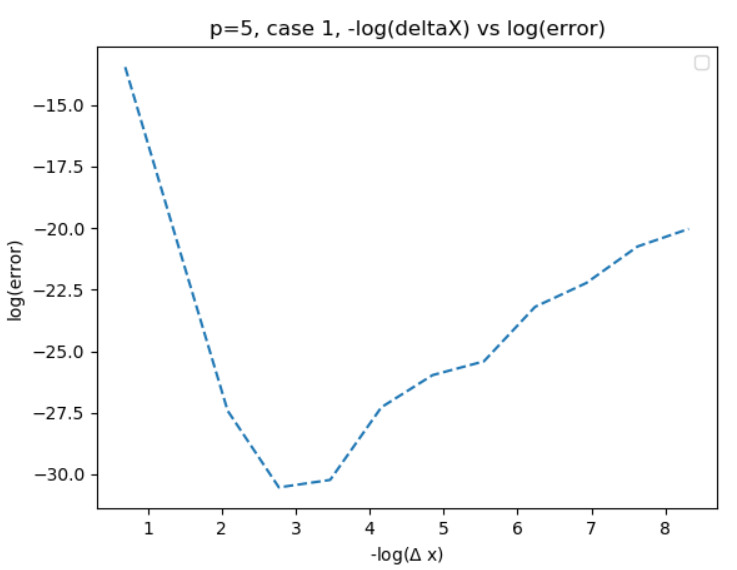


\*truncation error starts to have large effects at earlier values of

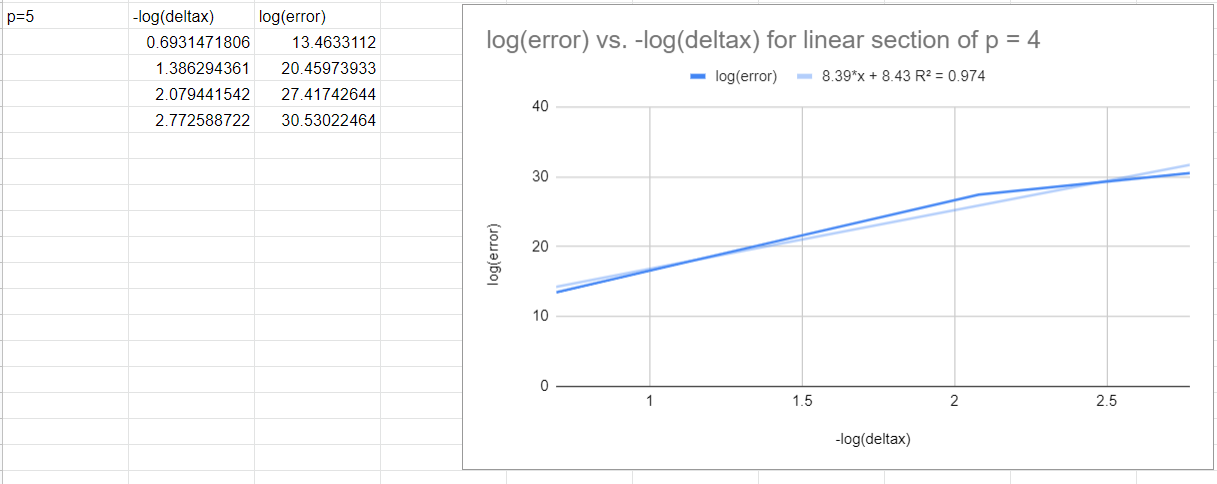
p = 5, case 1:



p = 5, case 1, convergence plot and data:



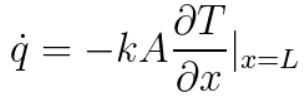
Linear section convergence rate approximately equal to 8.39:



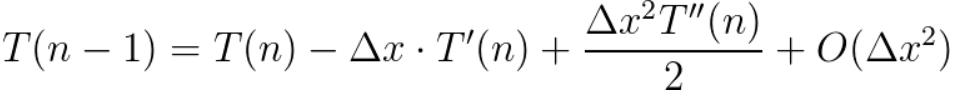
4. Heat flux stuff

1. By computing the heat entering the domain at x=L;

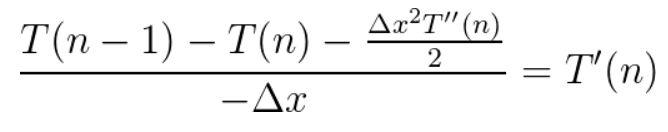
Heat loss entering the domain at x= L is found by using the definition of heat transfer across the bar:



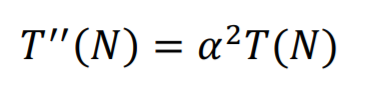
This can be approximated using Taylor series as follows:



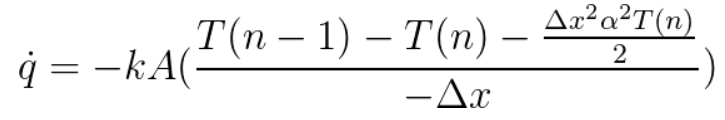
Rearranging for T’(n) leads to



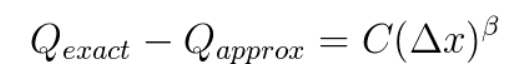
From our original second order differential equation, the following becomes true:



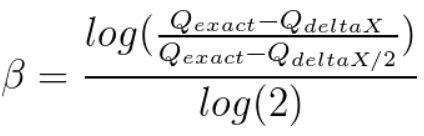
Solving for T’(n) and replacing this value in our heat loss equation leads to:



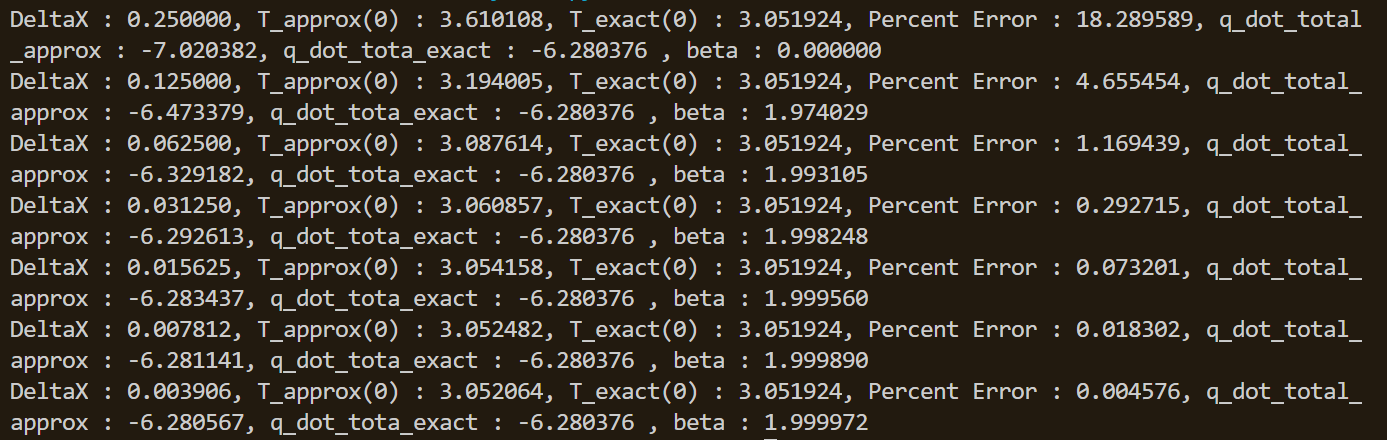
The rate of convergence between the exact solution can be defined by



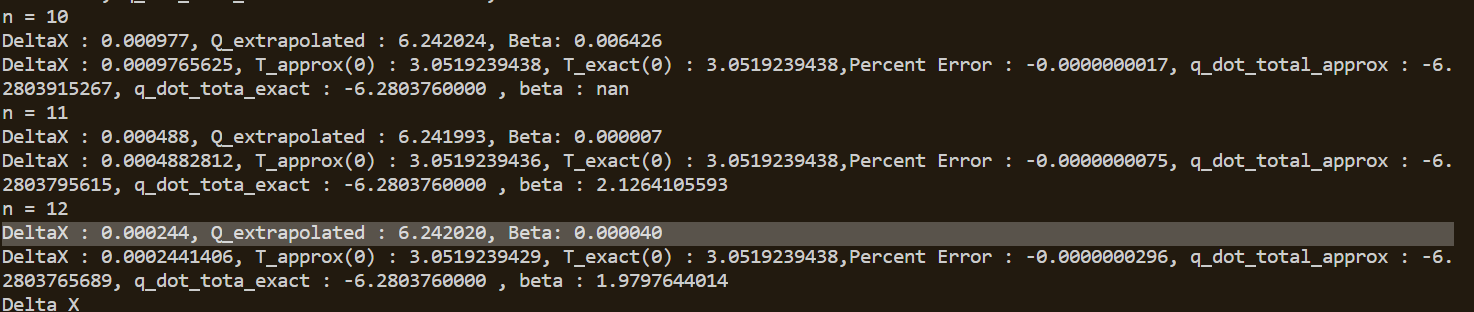
This can be rearranged with a successive approximation with:



The result of using these equations in python leads to the following results the agree with Austin Bradshaw’s results:

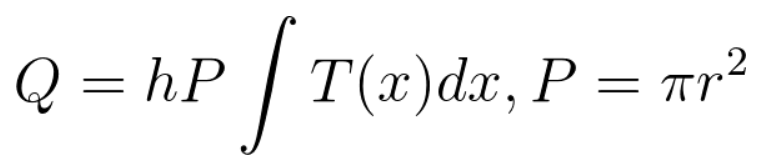


Heat entering using 10th order convergence using the same method of finding dq/dt: This data also indicates a convergence of 2 with this method.

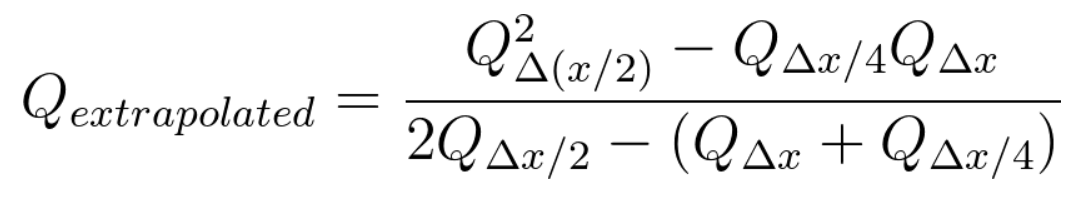


1. By computing the total heat flux exiting the domain through Newton Cooling from the lateral surface and the cross-section at x=0. Present your results in the same way as Bradshaw using Tables & Graphs and report the Convergence Rates

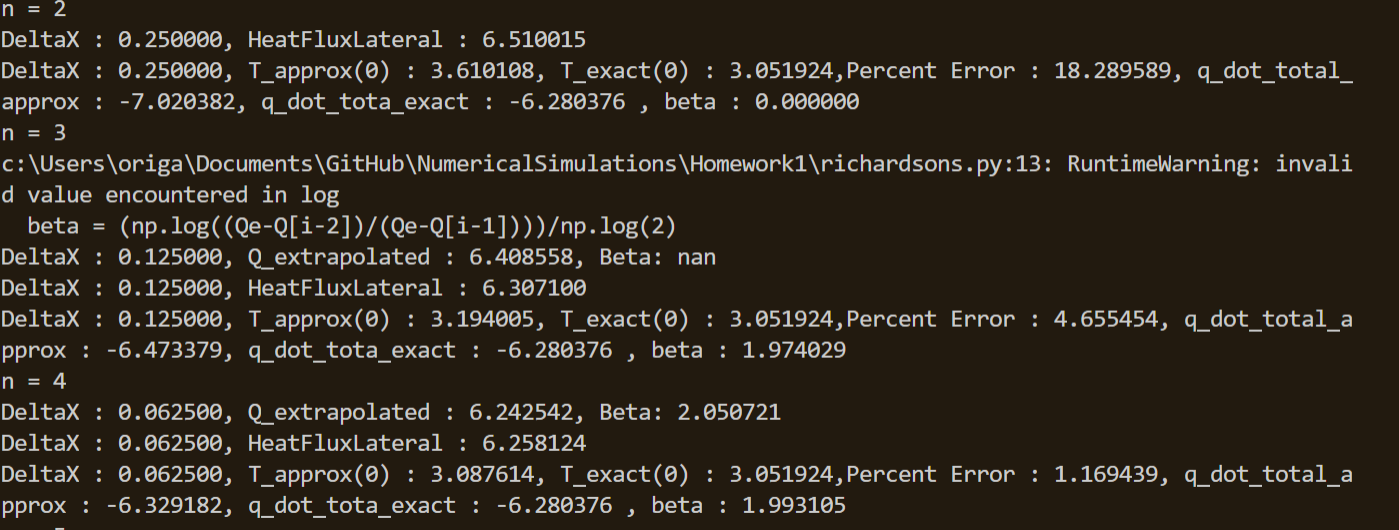
Total heat flux can be calculated by

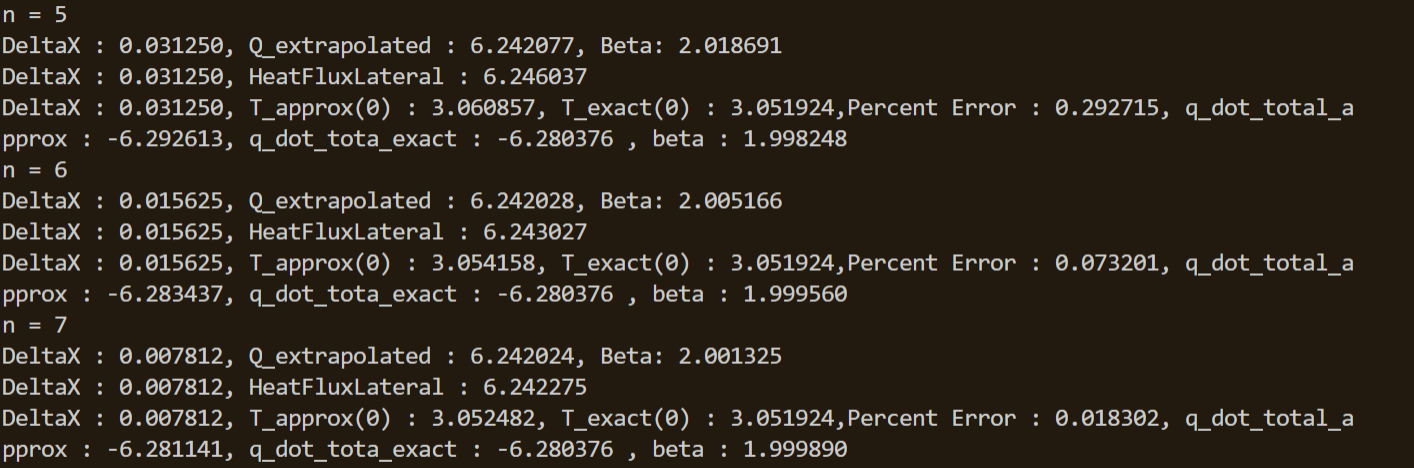


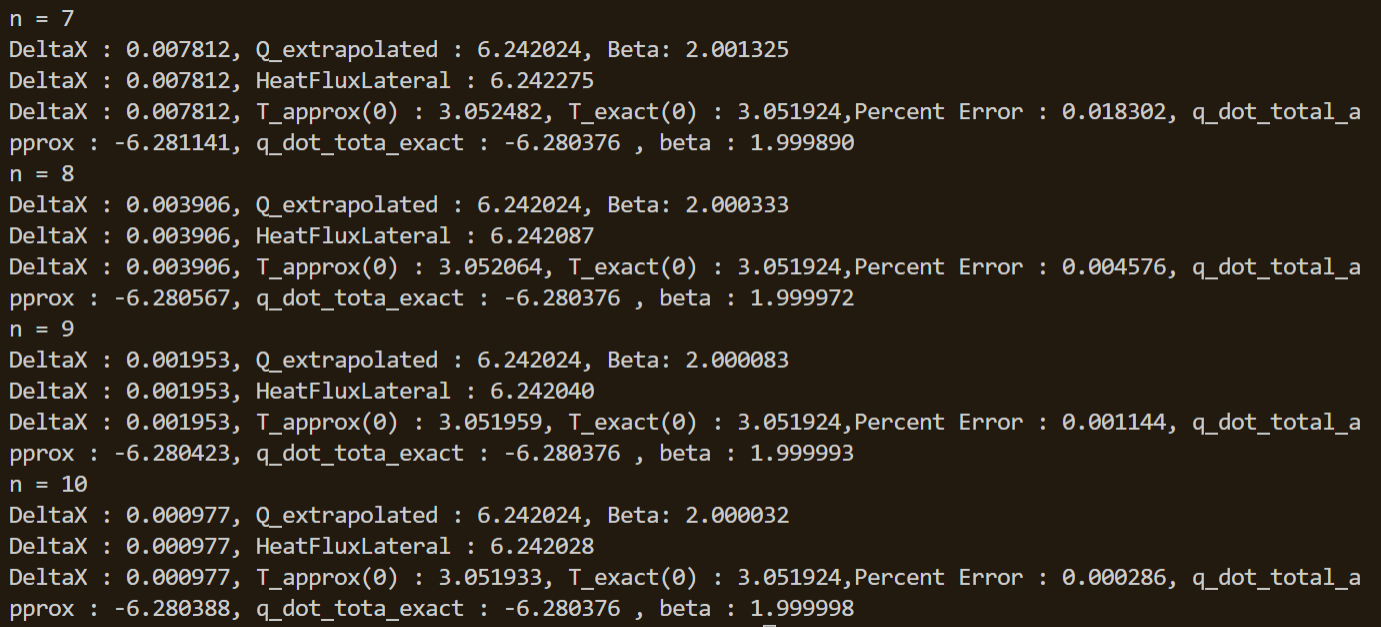
Using Simpsons rule, the total lateral heat flux can found and programmed in python. The lateral heat can also be extrapolated using Richardson extrapolation with the following relationship:



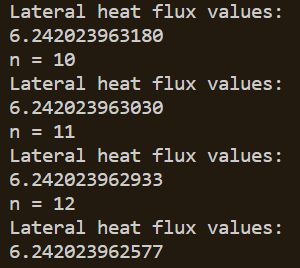
Convergence for case 2, with heat flux, extrapolated values, and Beta values are given from the code below:



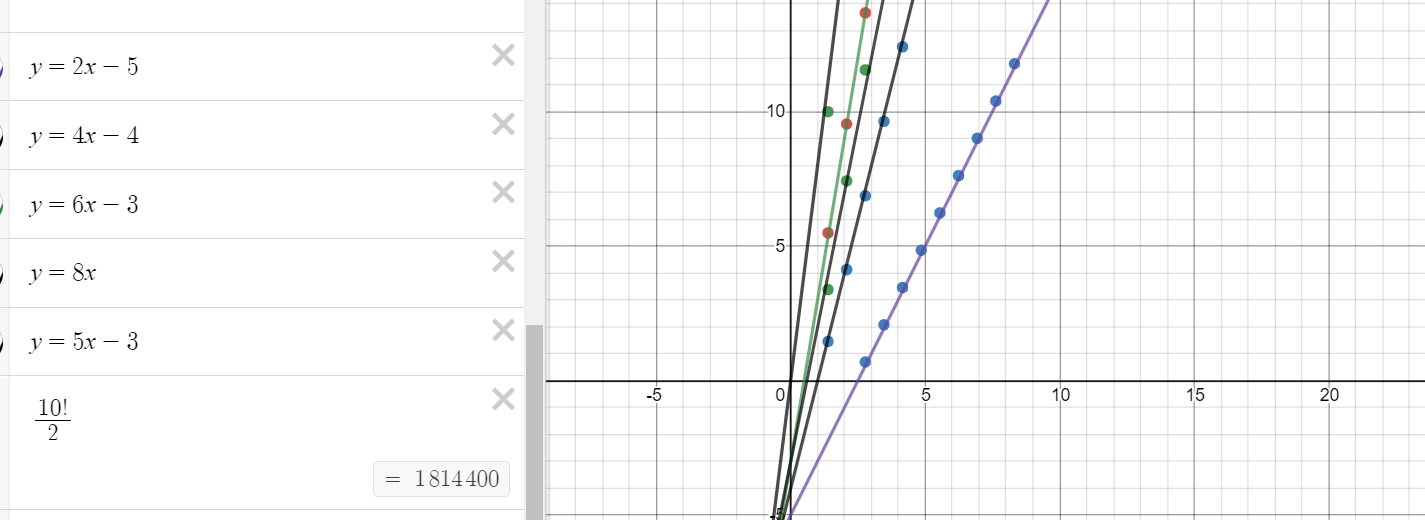




Results of calculating total heat flux using Simpson’s method on a 10th order convergence:



Overall convergence values of graphs, graphed on Desmos:



The last approximate fit on the left demonstrates the issue of convergence from 10th order.

Conclusion:

Using finite difference models, the Temperature as a function of position on the heat rod can be estimated. The current models used in this homework assignment make use of second order finite difference models, 3rd order FDM, 4th order FDM, 6th order FDM, 8th order FDM, and 10th order FDM. The values of convergence reach their expected values for all orders except 10th order FDM. The code I created in Python for the class uses the same models and ideas, however it is inefficient in solving the temperatures. The code does not use the Thomas algorithm, instead just using the numpy packages to find the inverse.

Homework 2 was useful in seeing how quickly the accuracy can improve with higher orders of FDM convergence. ­­