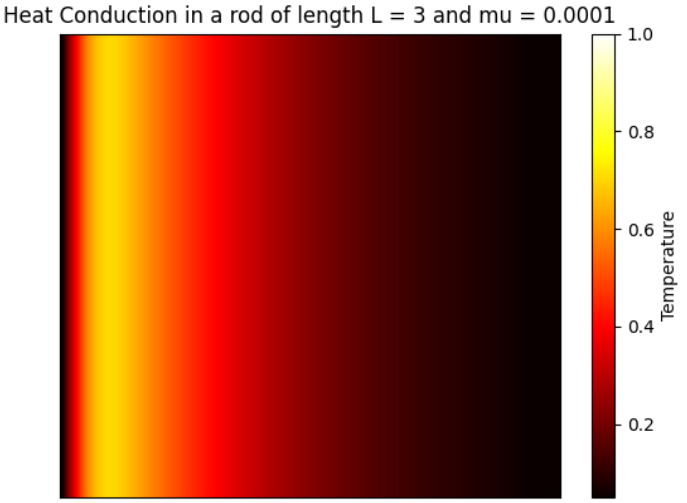
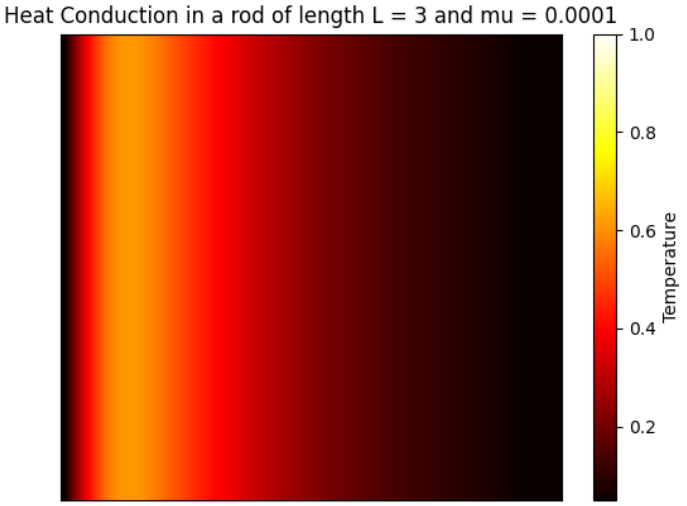
CS5016: Computational Methods and Applications
Assignment 7: Partial Differential Equations and Finding Zeroes

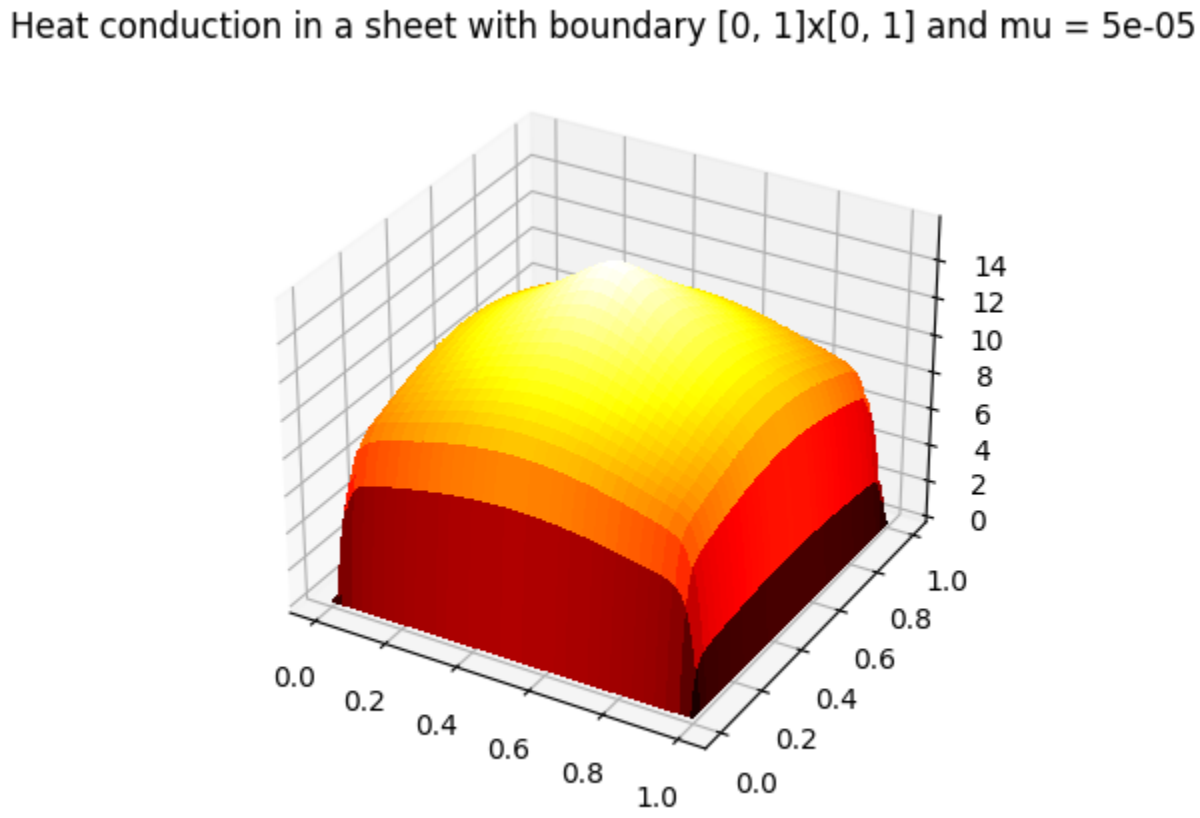
Sujit Mandava
112001043

Q1. The goal is to create a function that solves the given heat equation and visualizes the conduction of heat in a rod under the given conditions. The method plotHeatEquation() solves the heat equation numerically using finite difference method, and creates the heat transfer animation. The equations are solved by the solvePDE() method using finite difference method. It takes several parameters including the length of the rod (L), total time for simulation (T), spatial step size (h), time step size (ht), initial condition at t=0 (f0), boundary conditions at x=0 (fT) and x=L (fL), source term or heat generation term (f), and thermal diffusivity (mu). It returns the resulting temperature distribution over a period of time. This result is then used to create the heat transfer animation, using FuncAnimation.





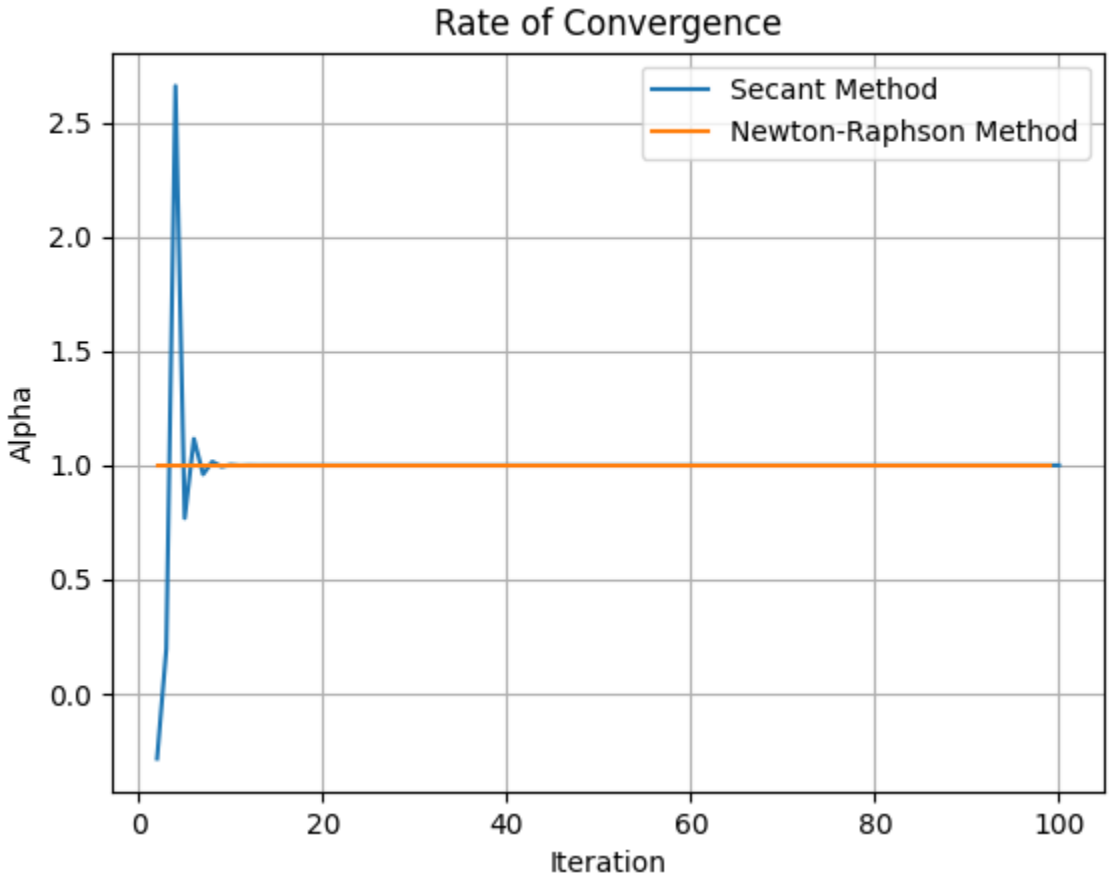
Q2. The goal of the code is to solve the given heat equation and visualize the conduction of heat in a sheet under the given conditions. The method plot2DHeatEquation() accomplishes this by solving the heat equation numerically using the finite difference method, and creating a heat transfer animation. The equations are solved by the solvePDE() method, which uses the finite difference method to discretize the spatial and temporal derivatives. It takes parameters such as the length of the sheet (xmin, xmax, ymin, ymax), total time for simulation (T), spatial step size (h), time step size (ht), initial condition at t=0 (fT), boundary conditions at the top (fT) and bottom (fB) edges of the sheet, source term or heat generation term (f), and thermal diffusivity (mu). It returns the resulting temperature distribution over a period of time. The resulting temperature distribution is then used to create the heat transfer animation using FuncAnimation, which animates the temperature changes over time on a 3D plot.



Q3. The computeNthRoot() function aims to compute the nth root of a given number with a specified precision using the bisection method. The function takes three input parameters: n for the desired root, x for the number to compute the root of, and e for the desired precision. Inside the function, a while loop is used to iteratively update the lower and upper bounds of the possible root until the desired precision is achieved. The function evaluates the function f(c, n, x), representing the difference between c raised to the power of n and x, and updates the bounds based on its sign. Once the loop terminates, the function returns the estimated nth root of x with the specified precision. The provided code calls the function with specific values of n, x, and e,

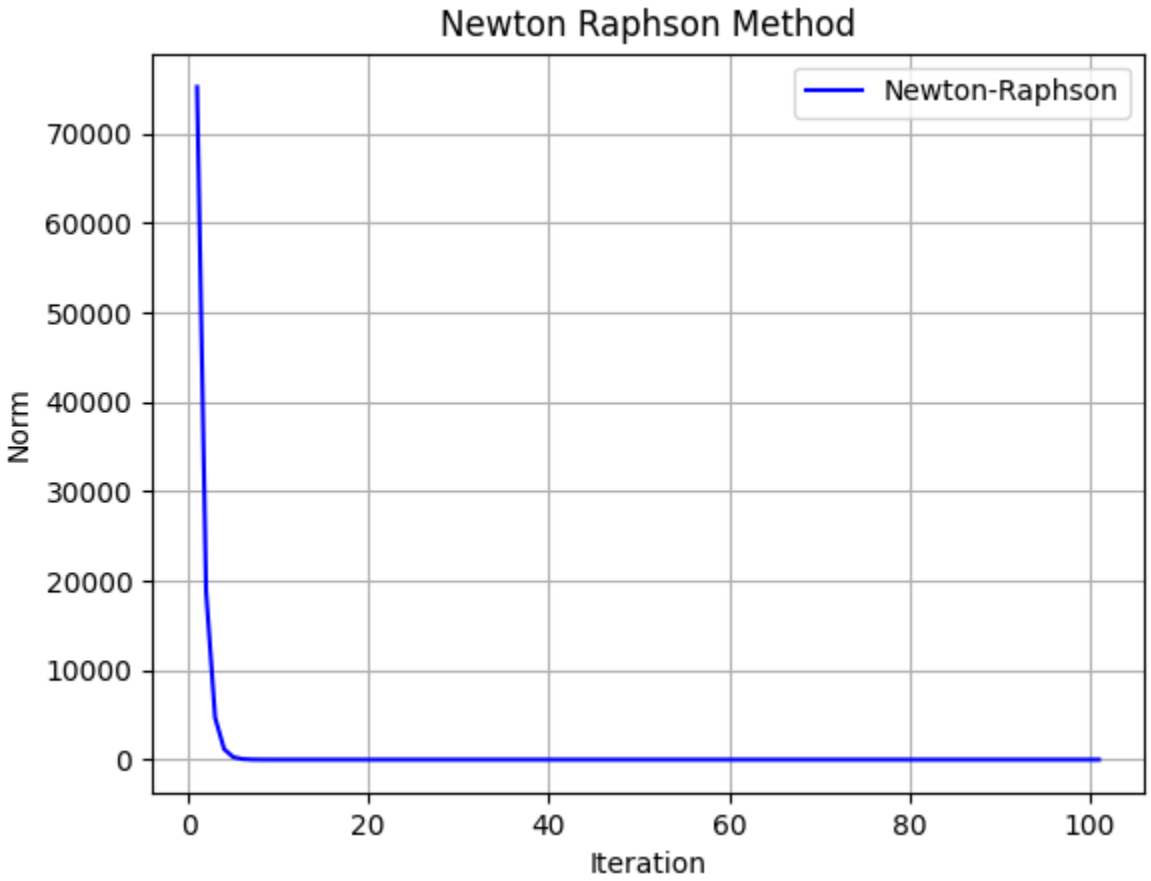
and prints the computed result. The computeNthRoot() function provides an efficient and accurate way to compute the nth root of a given number using the bisection method with a specified precision.

Q4. The code provided implements the Newton-Raphson method and the secant method for numerical equation solving, along with a function for comparing their rates of convergence. The newtonRaphsonMethod() function updates the approximation using the Newton-Raphson formula until the maximum number of iterations is reached or the desired accuracy is achieved. The secantMethod() function updates the approximation using the secant formula. The compareMethods() function calculates the rate of convergence for each method using the getROC() function and plots the ROC values against the number of iterations using matplotlib. The resulting plot allows for visual comparison of the rates of convergence of the Newton-Raphson method and the secant method.



Q5. The code provided implements the Newton-Raphson method for solving a system of three nonlinear equations. The functions f(x) and fj(x) define the equations and their corresponding Jacobian matrix, respectively. The newtonRaphsonMethod() function takes these functions as inputs, along with an initial guess x0 and a maximum number of iterations N. It updates the

guess using the Newton-Raphson method and stores the results in a list. This list is then used to plot the norm of the residual (f(x)) at each iteration against the number of iterations, to check the convergence of the method.



Q6. The code utilizes the enhanced Polynomial class to approximate the roots of a polynomial using the Aberth method. The aberthMethod method first creates a polynomial with the given roots. We add a method, printRoots() to the Polynomial class to approximate the roots. This method first generates n random numbers between the lower and upper bounds of the roots. Then, it checks whether the error is less than the provided limit or not. While this is not true, it iteratively updates the roots as defined by the Aberth Method, and finally returns the roots when the error is satisfactory.

Q7. The code first generates multiple points between the two limits a,b, where y = f(x). The bestFitFunc method is defined to find the polynomial that best fits the generated points. In the findRootsInInterval method, we then call the printRoots method to approximate all the possible roots for this generated polynomial. Finally, we check whether the roots lie in the provided range or not.