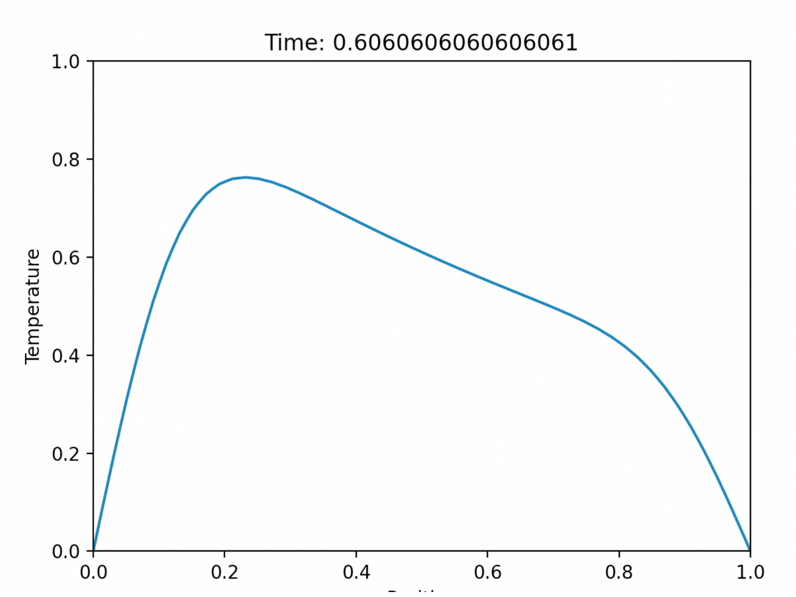
CS5016 : Computational Methods and Applications
Assignment 7 : Partial Differential Equations

112001051
VM Sreeram

Question 1

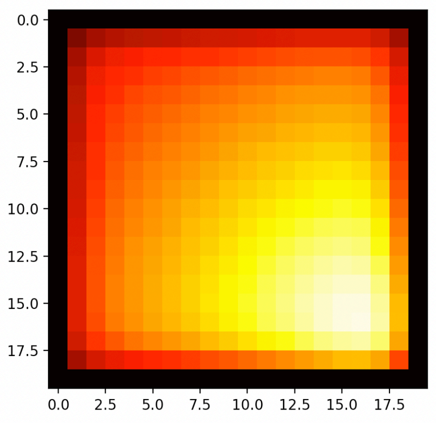
We define a simulation of heat conduction in a rod using a partial differential equation (PDE) and solves it using the `solve_ivp` function from the `scipy.integrate` module. The resulting temperature values are then plotted over time using an animation created with the `matplotlib.animation` module. The animation displays the temperature values at each time step as a function of position along the rod.



A screen grab from the animation is pasted for convenience.

Question 2

We define a two-dimensional heat equation and solves it using a forward Euler method. The `animate` function animates the solution using `matplotlib.animation`. The `visualize_2d` function sets up the parameters for the heat equation and initializes the animation. The resulting animation shows the heat equation's solution over time, represented as a two-dimensional color map.



A screen grab from the animation is pasted for convenience. The input was `visualize_2d(0.9, 0.9)`

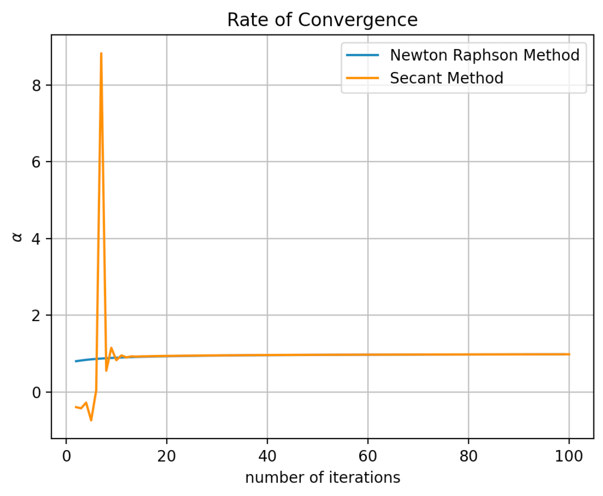
Question 3

We define a function `nthRoot` that uses binary search to find the nth root of a given number with a given tolerance. For testing, the function is called to find the 12th root of 99 raised to the power of 12, and the result is printed to the console.

```
% python Q3.py
98.99999660535454
```

Question 4

We define functions for the Newton-Raphson method and the secant method to find the root of a function, and a function to calculate the rate of convergence for each method. The code applies both methods to find the root of the function $f(x) = x \cdot e^{-x}$ and calculates the convergence rates for each method. Finally, the code plots the rate of convergence for each method as a function of the number of iterations.

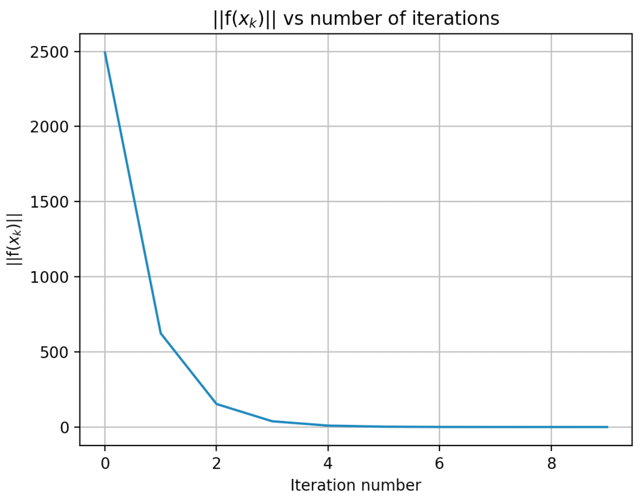


The output is pasted above for convenience.

Question 5

We define a system of equations as a function of x and the Jacobian matrix of the system of equations as another function of x. We then implement the Newton-Raphson method to solve the system of equations. The code sets an initial guess for x and solves the system of equations using the Newton-Raphson method. Finally, the code prints the solution and plots the sequence of f norms over the iterations. This allows us to see how well the Newton-Raphson method is converging to the solution.

It may be noted that we stop computation when f_norm_val is below a certain tolerance (can be passed as arg to the fn).



```
x = [ 0.83328161  0.03533462 -0.49854928]
```
The output is pasted above for convenience.

Question 6

We define a function `FindRoots_Aberth` that finds the roots of a polynomial using the Aberth method. The function takes a list of actual roots and an optional maximum number of iterations as input. It initializes a polynomial with a single root at 1 and multiplies it by factors (x - root) for each actual root. It then generates a list of random initial guesses for the roots and starts the Aberth method to compute the roots of the polynomial.

The Aberth method is an iterative process that involves computing the derivative and denominator sum for each root and using them to update the root value. The function checks if the computed roots are within a small number `EPS` of the actual roots and prints and returns the roots if they are. If the maximum number of iterations is reached, the function prints the polynomial and the computed roots. Finally, the function is called with some example inputs to demonstrate its usage.

For this call:

```
FindRoots_Aberth([8,9,22,44],maxIter=20)
```

The output was:

```
Coefficients of the polynomial are:
69696 -21208 2162 -83 1
Roots:  [7.999999999999957,  9.000000000000023,  21.999999999997804,
43.999999999722036]
```

Note that the output won't be unique each run.

Question 7

This Python code defines a modified version of the Aberth method for finding the roots of a function. The function `FindRoots_Aberth_modified` takes as input the function to find the roots of, the interval `[a,b]` over which to find the roots, the degree of the polynomial to be fitted to the function, and an optional maximum number of iterations.
The function first creates a list of `(x,y)` points of the function over the interval `[a,b]`. It then fits a polynomial of degree n to the points using the function `bestFitPoly` and computes its derivative. The function initializes a list of equally spaced points between `a` and `b` as initial guesses for the roots and starts the Aberth method to compute the roots of the polynomial. The function checks if the absolute value of the polynomial at the computed roots is less than `EPS` and if the computed roots are within `[a-EPS,b+EPS]`.
If the computed roots are within `EPS` of the actual roots, the function returns the roots. If the maximum number of iterations is reached, the function prints a message and returns the computed roots. Finally, the function is called with a sample function `fun` and prints the roots and their corresponding values of the function.

For fun np.exp(x)-4 and range -1,4,

Output:

```
fun( 1.386294382762556 ) = 8.657066263850766e-08
```
