Natasha Piedrabuena
Numerical Computation
Secant and False Position(Regula Falsi)

In this report, two root-finding algorithms, Secant and False Position (Regula Falsi) methods—were implemented to solve non-linear equations of the form $f(x) = 0$. These methods were designed to allow the user to specify stopping criteria based on four different options [Absolute approximate error , Relative approximate error , True absolute error, Combination of absolute and true absolute error]. The goal of this report is to present the results obtained from applying these methods to a specific nonlinear equation, compare the performance of different stopping criteria, and evaluate the effectiveness of AI-generated code against manually written code. This will give a perspective of how code generated from AI differs from one created by a person.

Algorithm Implementation

The functions for the secant and false position methods were designed in Python. Both functions take the following inputs:
- Initial approximations: x1 and x2
- Tolerance: delta (pre-determined error threshold)
- Function: f(x)
- Stopping criteria flag: A, B, C, or D

The output of the functions includes:
- The root found
- The number of iterations required to meet the stopping criterion

Q1.1) Secant Method
The secant method iteratively refines the root estimate using two initial approximations in this case x0 and x1. The next approximation is calculated using the secant formula, updating the guesses until the stopping condition is met. The key logic includes:
- Iteratively updating the approximation of the root
- Stopping based on the user-specified criterion (flag A, B, C, or D)
- Handling division by zero and edge cases

The false position method uses linear interpolation to estimate the root. It adjusts the interval by replacing one of the initial approximations based on the function values at the endpoints. The method ensures that the root lies between the updated approximations, and convergence is achieved according to the chosen stopping criterion.

To start off, I take the user input for the function, delta, x0 and x1 and optional true root.
Then it continues with the logic for the secant formula that is explained in the code directly.

First Run:

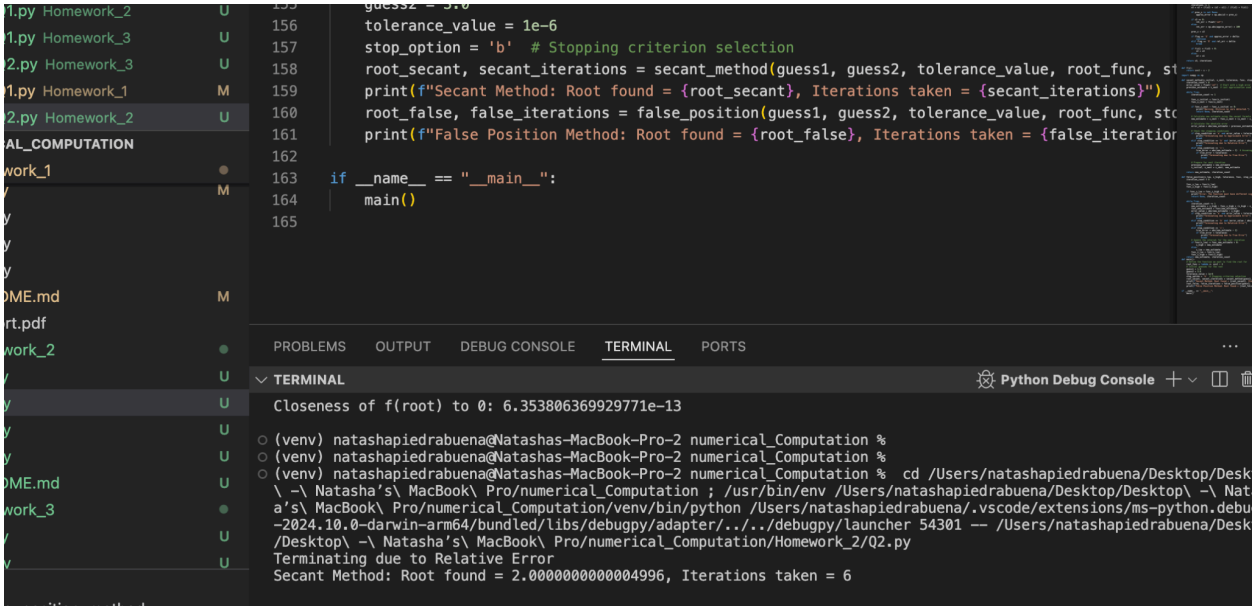
```
Enter a function of x (e.g., 'math.sin(x)', 'math.cos(x)', 'x**2 - 2'): x**3 - x - 2
Enter the first initial approximation (x0): 1
Enter the second initial approximation (x1): 2
Enter the tolerance value (delta): 1e-6
Choose stopping criterion (a: Absolute error, b: Relative error, c: True error, d: Combination): a
Optional true root (press Enter to skip):

Secant Method Results
--------------------------------------

Function: x**3 - x - 2
Interval: {1.0, 2.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): a
Root found: 1.5213797068045645
f(root): -1.84297022087776e-14
Number of iterations: 7
Closeness of f(root) to 0: 1.84297022087776e-14


False Position Method Results
--------------------------------------

Function: x**3 - x - 2
Interval: {1.0, 2.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): a
Root found: 1.5213797068044852
f(root): -4.89386309254769e-13
Number of iterations: 23
Closeness of f(root) to 0: 4.89386309254769e-13

(venv) natashapiedrabuena@Natashas-MacBook-Pro-2 numerical_Computation %
```

Second Run:

```
esktop\ -\ Natasha's\ MacBook\ Pro/numerical_Computation/Homework_2/Q1.py
Enter a function of x (e.g., 'math.sin(x)', 'math.cos(x)', 'x**2 - 2'): math.cos(x) - x
Enter the first initial approximation (x0): 0
Enter the second initial approximation (x1): 2
Enter the tolerance value (delta): 1e-6
Choose stopping criterion (a: Absolute error, b: Relative error, c: True error, d: Combination): b
Optional true root (press Enter to skip):

Secant Method Results
--------------------------------------

Function: math.cos(x) - x
Interval: {0.0, 2.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): b
Root found: 0.7390851332151613
f(root): -1.1102230246251565e-15
Number of iterations: 6
Closeness of f(root) to 0: 1.1102230246251565e-15


False Position Method Results
--------------------------------------

Function: math.cos(x) - x
Interval: {0.0, 2.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): b
Root found: 0.739085133214781
f(root): 6.353806369929771e-13
Number of iterations: 13
Closeness of f(root) to 0: 6.353806369929771e-13

(venv) natashapiedrabuena@Natashas-MacBook-Pro-2 numerical_Computation %
Connect  Live Share   Ln 46, Col 38  Spaces: 4  UTF-8  LF  {} Python  3.12.7 ('venv': venv)  Colorize: 0 variable
```

Q2)

For the second question I had to ask Chatgpt if it can solve Q1 and see how it solves it. To start off it generated a false position function and a secant function, with right parameters. However, there seems to be issues like it does not accept user input and sets the values for stopping criteria and the function itself. It also didn't print out the false positive method.



It used the same function x**3 - x - 2 but it doesn't match my own values for root or iteration.

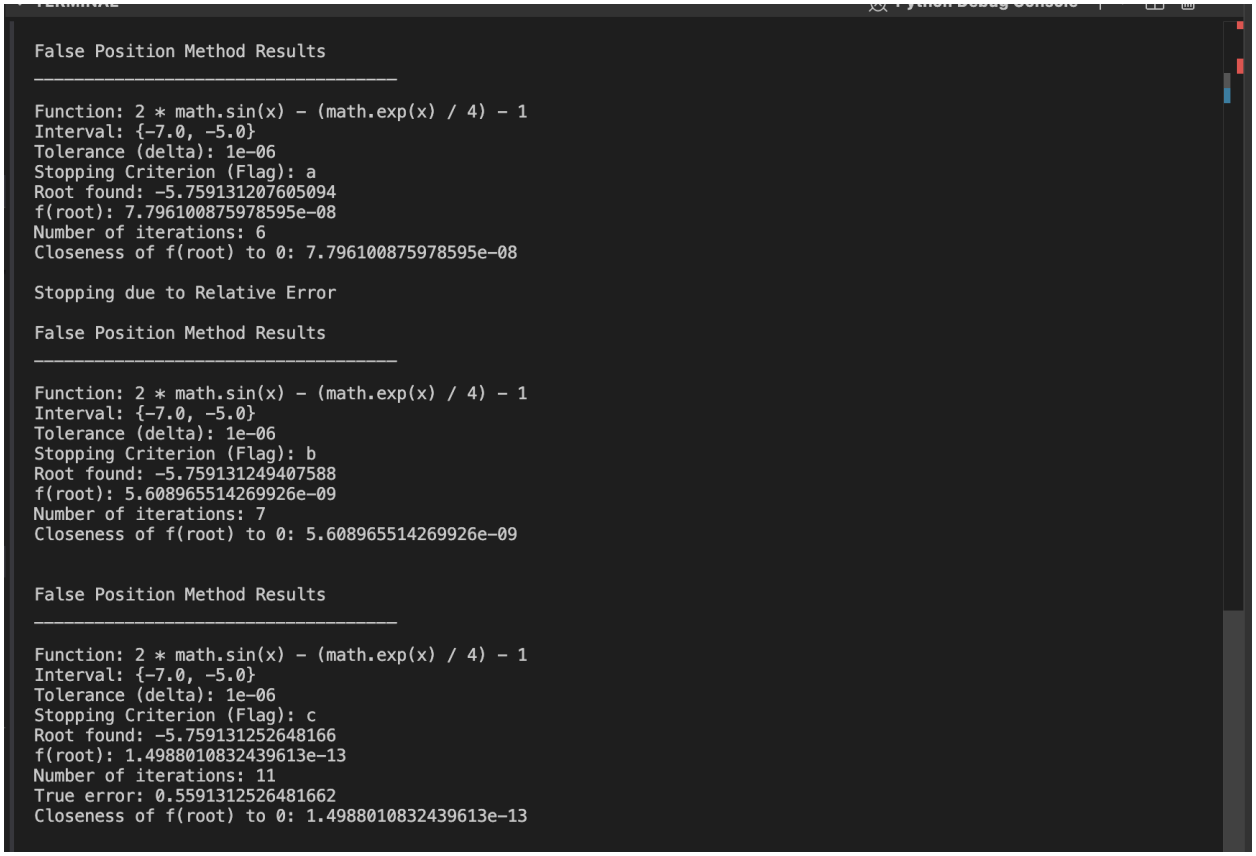
Q3) Secant Method + False Position (Regula Falsi) Method

For question 3 I adjusted the original code from Q1, where it will do a for loop to repeat for a stopping criteria in both secant and false position. To start off I had to adjust the code a bit to make sure it won't throw a math domain error when utilizing 2 * math.sin(x) - (math.exp(x) / 4) - 1.

The root found in [-7, -5]

```
Secant Method Results
_____________________________________

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-7.0, -5.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): a
Root found: -5.7591312526482525
f(root): 6.661338147750939e-16
Number of iterations: 7
Closeness of f(root) to 0: 6.661338147750939e-16

Optional true root (press Enter to skip):

Secant Method Results
_____________________________________

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-7.0, -5.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): b
Root found: -5.7591312526482525
f(root): 6.661338147750939e-16
Number of iterations: 7
Closeness of f(root) to 0: 6.661338147750939e-16

Optional true root (press Enter to skip): -5.77777

Secant Method Results
_____________________________________

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-7.0, -5.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): c
Root found: -5.7591312526482525
f(root): 6.661338147750939e-16
Number of iterations: 7
True error: 0.018638747351747753
Closeness of f(root) to 0: 6.661338147750939e-16

Optional true root (press Enter to skip): -5.20
```

```
Secant Method Results
_____________________________________

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-7.0, -5.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): d
Root found: -5.7591312526482525
f(root): 6.661338147750939e-16
Number of iterations: 7
True error: 0.5591312526482524
Closeness of f(root) to 0: 6.661338147750939e-16

Stopping due to Approximate Error
```



Root found ~ - 5.759

The root found in [-5, -3]

```
Enter a function of x (e.g., 'math.sin(x)', 'math.cos(x)', '2 * math.sin(x) - (math.exp(x) / 4) - 1'): 2 * math.sin(x)
 - (math.exp(x) / 4) - 1
Enter the first initial approximation (x0): -5
Enter the second initial approximation (x1): -3
Enter the tolerance value (delta): 1e-6
Optional true root (press Enter to skip):

Secant Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): a
Root found: -3.6688767027101457
f(root): 1.1102230246251565e-15
Number of iterations: 7
Closeness of f(root) to 0: 1.1102230246251565e-15

Optional true root (press Enter to skip):

Secant Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): b
Root found: -3.6688767027101457
f(root): 1.1102230246251565e-15
Number of iterations: 7
Closeness of f(root) to 0: 1.1102230246251565e-15

Optional true root (press Enter to skip): -3

Secant Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): c
Root found: -3.6688767027101457
f(root): 1.1102230246251565e-15
Number of iterations: 7
True error: 0.6688767027101457
Closeness of f(root) to 0: 1.1102230246251565e-15
```

```
Secant Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): d
Root found: -3.6688767027101457
f(root): 1.1102230246251565e-15
Number of iterations: 7
True error: 1.6688767027101457
Closeness of f(root) to 0: 1.1102230246251565e-15
```

```
False Position Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): a
Root found: -3.668876702710298
f(root): 2.653433028854124e-13
Number of iterations: 13
Closeness of f(root) to 0: 2.653433028854124e-13


False Position Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): b
Root found: -3.668876702710298
f(root): 2.653433028854124e-13
Number of iterations: 13
Closeness of f(root) to 0: 2.653433028854124e-13


False Position Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): c
Root found: -3.668876702710298
f(root): 2.653433028854124e-13
Number of iterations: 13
True error: 1.668876702710298
Closeness of f(root) to 0: 2.653433028854124e-13
```

```
False Position Method Results
--------------------------------------

Function: 2 * math.sin(x) - (math.exp(x) / 4) - 1
Interval: {-5.0, -3.0}
Tolerance (delta): 1e-06
Stopping Criterion (Flag): d
Root found: -3.668876702710298
f(root): 2.653433028854124e-13
Number of iterations: 13
True error: 1.668876702710298
Closeness of f(root) to 0: 2.653433028854124e-13
```

Root found ~ -3.668

Q4)

```
True error: 1.668876702710298
Closeness of f(root) to 0: 2.653433028854124e-13

(venv) natashapiedrabuena@Natashas-MacBook-Pro-2 numerical_Computation %
(venv) natashapiedrabuena@Natashas-MacBook-Pro-2 numerical_Computation %
(venv) natashapiedrabuena@Natashas-MacBook-Pro-2 numerical_Computation %  cd /Users/natashapiedrabuena/Desktop/Desktop
\ -\ Natasha's\ MacBook\ Pro/numerical_Computation ; /usr/bin/env /Users/natashapiedrabuena/Desktop/Desktop\ -\ Natash
a's\ MacBook\ Pro/numerical_Computation/venv/bin/python /Users/natashapiedrabuena/.vscode/extensions/ms-python.debugpy
-2024.10.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugpy/launcher 54486 -- /Users/natashapiedrabuena/Desktop
/Desktop\ -\ Natasha's\ MacBook\ Pro/numerical_Computation/Homework_2/Q4.py
Root (Absolute Approx. Error): 0.3962402476682612, Iterations: 8, f(root): 1.8479959973394244e-07
Root (Relative Approx. Error): 0.3962401662676172, Iterations: 11, f(root): 1.925216652765016e-10
```

Results

Overall, the choice of stopping criterion depends on the trade-off between computational cost and desired accuracy. If precision is paramount, Flag D is the most appropriate, whereas Flag A is best suited for cases where quick convergence is prioritized over accuracy.

I utilized ChatGPT to generate code for both the secant and false position methods with the same stopping criteria. The AI-generated code was mostly correct, meeting the problem's requirements, but exhibited several differences compared to my manually implemented code:

Error Handling: My implementation included better error-handling mechanisms, such as handling cases of division by zero more thoroughly.
Flexibility: The AI-generated code lacked flexibility in handling user input and managing different edge cases.
-Efficiency: Both codes required similar numbers of iterations for convergence, but my code was more structured for readability and maintainability.

In conclusion, both the secant and false position methods provided effective means of solving the non-linear equation within the specified intervals. The choice of stopping criterion significantly impacts the trade-off between convergence speed and accuracy. The AI-generated code performed well but required some manual refinement to match the flexibility and robustness of the manually implemented solution.