Week 2 Methods

**Bisection method:**

The bisection method is a root-finding algorithm that has its own set of advantages and disadvantages. Here are some pros and cons of using the bisection method:

**Pros:**

1. **Convergence:** The bisection method is guaranteed to converge to a solution as long as the function is continuous on the interval and changes sign over that interval. This makes it a robust method for finding roots.
2. **Simplicity:** The method is conceptually simple and easy to implement. It involves bisecting the interval and selecting the subinterval where the root must lie.
3. **Robustness**:Bisection is less sensitive to the choice of the initial interval compared to some other root-finding methods. As long as the initial interval brackets a root, the method is likely to converge.
4. **IntervalReduction*:*** The method reduces the interval size by half at each step, ensuring that the interval containing the root becomes smaller with each iteration.

**Cons:**

1. **Slow** **Convergence**: The bisection method converges at a linear rate, meaning that the number of correct decimal places roughly doubles with each iteration. This can make it relatively slow compared to some other methods that converge at a faster rate, especially for functions with rapidly changing behavior.
2. **Requires Bracketing**: The method requires an initial interval where the function changes sign. If the initial interval does not bracket a root, the method may fail to converge or converge to the wrong solution.
3. **Single Root**: The bisection method is designed for finding a single root within a given interval. If there are multiple roots or if the function has complex behavior, the method may not be suitable.
4. **Not Suitable for Non-Continuous Functions**: The bisection method assumes the function is continuous over the interval. If the function has discontinuities or sharp turns within the interval, the method may not work well.
5. **No Information on Multiplicity**: The bisection method does not provide information about the multiplicity of the root (i.e., whether the root is a simple root or a root with multiplicity greater than one).

In summary, the bisection method is a reliable and simple approach for finding roots, especially when the initial interval is well-chosen, and the function is continuous. However, its linear convergence and limitations in handling certain types of functions may make other methods more suitable for specific situations.

**False Position method (Unmodified):**

Explanation:

* The script begins with comments specifying the encoding and providing information about the author and creation date.
* `MAX\_ITER` is defined as the maximum number of iterations to avoid infinite loops.
* The function `func(x)` is defined to represent the example function for which the root will be found.
* The `regulaFalsi` function is implemented to find the root of `func(x)` in the interval `[a, b]` using the False Position method.
* The function checks if the initial values of `a` and `b` are valid for the method to work.
* It iterates using a loop to refine the approximation of the root using the False Position formula.
* The loop breaks if the root is found or the maximum number of iterations is reached.
* The final root value and the number of iterations are printed.
* The driver code initializes the values of `a` and `b` and calls the `regulaFalsi` function to find the root.

**Pros:**

1. **Convergence**: Like the bisection method, the False Position method converges to the root, assuming the function is continuous and the initial interval brackets a root.
2. **Reduction of Interval**: Similar to the bisection method, the False Position method reduces the interval size at each step, improving the accuracy of the approximation.
3. **Linear Convergence**: The method converges faster than the bisection method in some cases because it uses a linear approximation to the function.

**Cons:**

1. **Slow Convergence for Certain Cases**: While the False Position method generally converges faster than the bisection method, it can still converge slowly for functions with certain characteristics.
2. **Dependence on Initial Interval**: The method requires a valid initial interval where the function changes sign. If the initial interval is poorly chosen, the method may converge slowly or fail to find a root.
3. **Not Suitable for Discontinuous Functions**: The False Position method assumes that the function is continuous over the interval. It may not work well for functions with discontinuities or sharp turns.
4. **Potential for Oscillation**: The method may oscillate or diverge in some cases, especially when the initial interval is large or when the function has complex behavior.

In summary, the False Position method shares some advantages with the bisection method, such as convergence and interval reduction, but it may converge faster in certain cases. However, it also has limitations, including sensitivity to the initial interval and potential issues with discontinuous functions. The choice of method depends on the characteristics of the specific problem being solved.

**Inverse Quadratic Interpolation:**

**Explanation:**

* + The script begins with comments specifying the encoding and providing information about the author and creation date.
  + The `inverse\_quadratic\_interpolation` function is defined to find the root of a given function using the Inverse Quadratic Interpolation method.
  + The method takes a function `f` and three initial guesses `x0`, `x1`, and `x2` as parameters, along with optional parameters for maximum iterations (`max\_iter`) and tolerance (`tolerance`).
  + The method iteratively refines the guesses until either the maximum number of iterations is reached or the difference between the last and new guesses is very close (less than the specified tolerance).
  + Interpolation coefficients `L0`, `L1`, and `L2` are calculated based on the current guesses and function evaluations.
  + The new guess is calculated using the interpolation coefficients.
  + The guesses are updated for the next iteration.
  + The function returns the final guess and the number of steps taken.
  + A test function `f` (a cubic function) is defined.
  + The Inverse Quadratic Interpolation method is applied to find the root with initial guesses of `4.3`, `4.4`, and `4.5`.
  + The root and the number of steps taken are printed.

**Pros:**

1. Convergence: Inverse Quadratic Interpolation exhibits quadratic convergence, which means it can converge faster than linear methods like the bisection method under certain conditions.
2. Fewer Function Evaluations: The method often requires fewer function evaluations compared to linear methods, contributing to computational efficiency.
3. No Derivative Required: Unlike Newton's method, Inverse Quadratic Interpolation does not require the calculation of derivatives.

**Cons:**

1. Sensitivity to Initial Guesses: The method can be sensitive to the choice of initial guesses. In some cases, it may converge to a solution quickly, while in others, it may diverge.
2. Potential for Divergence: In certain scenarios, especially if the initial guesses are not well-chosen, the method may diverge instead of converging to a solution.
3. Limited Applicability: The method may not work well for functions with complex behavior or for finding multiple roots.

In summary, Inverse Quadratic Interpolation offers quadratic convergence and efficiency in terms of function evaluations. However, its sensitivity to initial guesses and potential for divergence in certain cases should be considered. The choice of method depends on the specific characteristics of the problem being solved.

**Modified False Position:**

Explanation:

* The script begins with comments specifying the implementation of the Bisection Method for solving equations in Python 3.
* `MAX\_ITER` is defined as the maximum number of iterations to avoid infinite loops.
* The function `func(x)` is defined to represent the example function for which the root will be found.
* The `regulaFalsi` function is implemented to find the root of `func(x)` in the interval `[a, b]` using the False Position method.
* The function checks if the initial values of `a` and `b` are valid for the method to work.
* It iterates using a loop for a maximum number of iterations, refining the approximation of the root using the False Position formula.
* The loop breaks if the root is found or the maximum number of iterations is reached.
* The final root value and the number of iterations are printed.
* The driver code initializes the values of `a` and `b` and calls the `regulaFalsi` function to find the root.

**Pros:**

1. **Convergence**: The False Position method converges to the root, assuming the function is continuous and the initial interval brackets a root.
2. **Reduction of Interval**: The method reduces the interval size at each step, improving the accuracy of the approximation.
3. Linear Convergence: The method converges faster than the bisection method in some cases because it uses a linear approximation to the function.

**Cons:**

1. **Slow Convergence for Certain Cases**: While the False Position method generally converges faster than the bisection method, it can still converge slowly for functions with certain characteristics.
2. **Dependence on Initial Interval**: The method requires a valid initial interval where the function changes sign. If the initial interval is poorly chosen, the method may converge slowly or fail to find a root.
3. **Not Suitable for Discontinuous Functions**: The False Position method assumes that the function is continuous over the interval. It may not work well for functions with discontinuities or sharp turns.

In summary, the False Position method shares some advantages with the bisection method, such as convergence and interval reduction, but it may converge faster in certain cases. However, it also has limitations, including sensitivity to the initial interval and potential issues with discontinuous functions. The choice of method depends on the specific characteristics of the problem being solved.

**Naïve Line Search:**

**Explanation:**

* The naive\_root function is defined to find a root of a given function using a naive approach.
* The function takes parameters f (the function), x\_guess (initial guess for the root), tolerance (acceptable error in the root), and step\_size (the size of each iteration step).
* The function iterates until the absolute value of f(x\_guess) is less than the specified tolerance.
* Inside the loop, the guess is adjusted based on the sign of f(x\_guess) using a naive method: decrementing the guess if f(x\_guess) is positive, incrementing if negative.
* If f(x\_guess) becomes exactly 0, the function returns the current guess.
* The number of steps taken is incremented in each iteration.
* The function returns the final guess and the number of steps taken.
* A test function f (a quadratic function with a root at sqrt(20)) is defined.
* The naive\_root function is used to find the root with an initial guess of 4.5, tolerance of 0.01, and step size of 0.001.
* The root and the number of steps taken are printed.

**Pros:**

1. **Simplicity**: The method is straightforward and easy to implement.
2. **No Derivative Required**: The method does not require the calculation of derivatives.

**Cons:**

1. **Slow Convergence**: The method may converge slowly, especially for functions with complex behavior.
2. **Sensitivity to Step Size**: The choice of step size can affect the convergence speed and accuracy.
3. **Limited Applicability**: The method may not work well for functions with multiple roots or sharp turns.

In summary, the naive approach is simple but may have slow convergence and sensitivity to step size. It is suitable for simple cases but may not be efficient for more complex functions or when higher accuracy is required.