Errors:

True Error – True error refers to the difference between the exact or true value and the approximate or computed value obtained through a numerical method.
Equation:

$E_t = |x_{true} - x_{approx}|$

Estimated Error – Estimated error is an approximation of the true error, often derived from information available during the computation. It provides an estimate of how accurate the numerical method's result is likely to be.

Determined by analysing the convergence behaviour of an iterative method:
$E_t = |X_{final} - x(final-1)|$

Maximum true error in a range – is the maximum absolute value of the difference between the computed and the exact solution

Euler method:

Dec 2022 Question 2)b)
What did you choose the value of h and explain how it was selected?
Step size h – changes the accuracy and the efficiency of the numerical solution.
Small h:

- If the function has rapid changes or has regions of high curvature
- Stability – satisfy stability conditions to prevent numerical instability
- Large enough for the computational power

ODE stability:

- Stability is a consideration in ensuring the reliability and accuracy of the computed solution.
- Stability is linked to the behaviour of the numerical solution over time
- A stable numerical method – produces solutions that do not exhibit unbounded growth or oscillation as the computation progresses

-Explicit Euler method – size of h maintains stability
-implicit Euler method – advantage is the method is unconditionally stable for linear and time-invariant problems, Can handle large h without stability concerns

Difference between explicit and implicit Euler methods:

Explicit Euler:

- Value of the next time step is computed based on the information available at the current time step
- Yn+1 = yn + h*derivative

Implicit Euler:

- Solving through iterative process.
- Yn+1 = yn + h*(derivative of the next time step)

Dec 2021 – Question 1

Explain the operating principle of the code in detail: **numpy.roots**

- Input: Accepts an array representing polynomial coefficients.
- Construction: Constructs a square companion matrix from coefficients.
- Eigenvalue Computation: Uses linear algebra (e.g., `numpy.linalg.eig`) to compute eigenvalues of the companion matrix.
- Roots output : Returns an array of roots, potentially complex, resulting from eigenvalue computation
- Numerical Stability: Accuracy of roots relies on the numerical stability of eigenvalue computations, especially for polynomials with multiple or closely spaced roots.
- Use Case: Widely applied in scientific and engineering contexts for solving polynomial equations.
- complex roots: Can handle and return complex roots for polynomials with complex coefficients or complex roots.
- order of roots - The roots are returned without a specific order; their arrangement is dependent on the underlying algorithm.
- Efficiency: Efficiency is influenced by the size and structure of the polynomial, suitable for many cases but may have limitations for extremely large polynomials.

Numpy.roots:

The Numpy.roots function is from the NumPy library in python. It computes the polynomial as a matrix. The algorithm then uses linear algebra and the eigenvalues of the matrix to find the roots.

The function takes a 1-D array of the polynomial coefficients as inputs, where the coefficients are in decreasing order of power. It then constructs a companion matrix from the polynomial coefficients which is a square matrix. The roots of the polynomial are the eigenvalues of the companion matrix.

Ranking For Root Finding Methods:

1. Newton's Method:
   - Precision: High
   - Explanation: Newton's method is a fast converging method when the initial guess is close to the actual root. It exhibits quadratic convergence under favorable conditions, making it one of the most precise methods.

2. Secant Method:
   - Precision: High
   - Explanation: Similar to Newton's method, the secant method converges quickly, but it doesn't require the calculation of derivatives. It has good convergence properties, though not as fast as Newton's method under ideal conditions.

3. Root-script (Optimize.Newton):
   - Precision: High
   - xplanation: This method uses Newton's method but may include additional enhancements for stability and robustness, depending on the specific implementation. It's generally quite precise.

4. Modified False Position:
   - Precision: Medium to High
   - Explanation: Modified False Position is an improvement over the regular False Position method, providing better convergence in some cases. It generally converges faster than bisection and false position.

5. False Position: yes
   - Precision:Medium
   - Explanation: False Position converges at a linear rate, which is slower than quadratic convergence but still faster than bisection. It is more robust than some methods but may be slower in certain cases.

6. Inverse Quadratic Interpolation:
   - Precision: Medium
   - Explanation: This method is an extension of the secant method and may converge faster in some cases. However, its precision can vary depending on the specific problem.

7. Muller's Method:
   - Precision: Medium
   - Explanation: Muller's method is a modification of the secant method designed to handle complex roots. It can be precise but might not always outperform simpler methods.

8. Bisection: yes
   - Precision: Low to Medium
   - Explanation: Bisection converges at a linear rate and is less sensitive to the choice of initial guesses. It's a reliable method but is generally slower compared to faster converging methods like Newton's.

9. Single Fixed Point Iteration:
   - Precision: Low to Medium
   - Explanation: Single fixed point iteration may converge slowly and is highly dependent on the choice of the fixed point function. It might not be as precise as more sophisticated methods.

10. Synthetic Division Polynomial:
   - Precision: Low
   - Explanation: Synthetic division is a method primarily used for polynomial root finding. It's not as precise or robust as other numerical methods, especially for non-polynomial functions.

11. Naive Line Search:
   - Precision: Low
   - Explanation: Naive line search is the least precise among the methods listed. It involves searching along a line and might not converge well, especially for complex functions.