EE414FZ ASSIGNMENT 1

- Student Name: Hanlin Cai
- MU Student ID: 20122161
- FZU Student ID: 832002117

Q1. Find the $IEEE-754$ single precision representation, in hex, for your assigned number. Show its true relative round-off error is less than $\varepsilon_{\text{mach}}/2$. Notation $2e3$ is MATLAB notation for $2{\times}10^3$.

- $assignednumber = N = 1.53710981420157e+27$
- The answer is given at Page 2

Q2. A 3rd degree Taylor polynomial is used to approximate $f(x)$ about about point of expansion $(a, f(a))$. Find the true truncation error and maximum truncation error.

- $f(x) = cos^2(x),\ \ a = \pi/6,\ \ x = -\pi/2$
- The answer is given at Page 3

Q3. MATLAB

- $c = 115.00,\ \ x_1 = -5.00,\ \ x_x = 12.00,\ \ \sqrt[3]{c} = 4.86294413109428$
- The answer is given at Page 4-18

No.

Hanlin CAI 20122161 832002117 Date 2023. 10. 28th

EE414FZ Assignment 1.

Q1. My assigned number is $1.53710981420157e+27$

Firstly, $N = 1.53710981420157 \times 10^{27}$

$= 1.0011\,1101\,1101\,1101\,1110\,1010\,0111\,1111\,1011\,0011\,0001\,10101 \times 2^{90}$

And we need to cut down to 23 digits.

$\therefore N \to 0011\,1101\,1101\,1101\,1110\,101$

Biased exponent is $127+90 = 217 = (11011001)$

So the final number is $(0110\,1100\,1001\,1110\,1110\,1110\,1111\,0101)$

hex = 0x6C9EEEF5.

true round-off error is $\frac{N-A}{N} = \varepsilon_t$.

where $A = 1.53710977739465 96 \times 10^{27}$

$\therefore \frac{N-A}{N} \approx 3.681 \times 10^{-8} < \frac{\varepsilon_{mach}}{2} = 6 \times 10^{-8}$

Overall. $\left(\varepsilon_t < \frac{\varepsilon_{mach}}{2} \right)$.

No.

Date . .

Q2. My assigned numbers : $f(x) = \overset{\cos^2 x}{\cancel{\sin x^2}}$ $a = \frac{\pi}{6}$ $x = -\frac{\pi}{2}$.

Sol

$$\begin{cases} f(x) = \cos^2 x \\ f'(x) = -\sin(2x) \\ f^{(2)}(x) = -2\cos(2x) \\ f^{(3)}(x) = 4\sin(2x) \end{cases}$$

$$f(-\tfrac{\pi}{2}) = \cos^2\tfrac{\pi}{6} - \sin\tfrac{\pi}{3} - 2\cos\tfrac{\pi}{3} + 4\sin\tfrac{\pi}{3} + R_3$$

$$= \frac{6\sqrt{3} - 1}{4} \approx 2.3481$$

$$R_3 = |\cos^2(-\tfrac{\pi}{2}) - 2.3481| = 2.3481$$

$$R_n = \frac{f^{(n+1)}(c)}{(n+1)!}(x-a)^{n+1} \qquad n = 3 \quad x = -\frac{\pi}{2} \quad a = \frac{\pi}{6}$$

$$R_3 = \frac{f^{n+1}(c)}{(n+1)!}(x-a)^{n+1} = \frac{8\cos(2c)}{4!}\left(-\frac{2}{3}\pi\right)^4$$

where $-\frac{\pi}{2} < c < \frac{\pi}{6}$

$\max(\cos^2 c) = 1$ $\max(R_3) = 6.4137$

$R_3 < \max(R_3)$ as expected.

Q3 (A) MATLAB

The MATLAB program 1 have been given as follows:

```
format long;

% 初始化参数和变量
maxIterations = 50;
targetRoot = 4.8629;
tolerance = 1e-10;
results = zeros(maxIterations, 4);

% 定义要求根的函数
f = @(x) x^3 - 115;

% 初始区间
lowerBound = -5;
upperBound = 12;
lowerBoundValue = f(lowerBound);
upperBoundValue = f(upperBound);

for iteration = 1:maxIterations
    % 计算当前估计值
    currentRoot = (lowerBound + upperBound) / 2;

    % 计算相对误差
    relativeError = abs(upperBound - lowerBound) / currentRoot;

    % 计算真实误差
    trueError = abs(targetRoot - currentRoot);

    % 存储结果
    results(iteration, :) = [iteration, currentRoot, trueError,
relativeError];

    % 输出当前迭代结果
    fprintf('迭代次数: %d, 当前估计根: %.10f, 真实误差: %.1f, 相对误
差: %.10f\n', iteration, currentRoot, trueError, relativeError);
```

```
    % 检查终止条件
    if relativeError < tolerance
        break;
    end

    % 更新区间
    currentRootValue = f(currentRoot);
    if lowerBoundValue * currentRootValue < 0
        upperBound = currentRoot;
        upperBoundValue = currentRootValue;
    else
        lowerBound = currentRoot;
        lowerBoundValue = currentRootValue;
    end
end

% 保存结果到Excel文件
results = results(1:iteration, :);  % 截取有效的结果
writematrix(results, "data.xls");
```

1	7.750000000000000	2.89	0.55
2	5.625000000000000	0.76	0.38
3	4.562500000000000	0.30	0.23
4	5.093750000000000	0.23	0.10
5	4.828125000000000	0.03	0.06
6	4.960937500000000	0.10	0.03
7	4.894531250000000	0.03	0.01
8	4.861328125000000	0.00	0.01
9	4.877929687500000	0.02	0.00
10	4.869628906250000	0.01	0.00
11	4.865478515625000	0.00	0.00
12	4.863403320312500	0.00	0.00
13	4.862365722656250	0.00	0.00

14	4.862884521484370	0.00	0.00
15	4.863143920898440	0.00	0.00
16	4.863014221191410	0.00	0.00
17	4.862949371337890	0.00	0.00
18	4.862916946411130	0.00	0.00
19	4.862933158874510	0.00	0.00
20	4.862941265106200	0.00	0.00
21	4.862945318222050	0.00	0.00
22	4.862943291664120	0.00	0.00
23	4.862944304943080	0.00	0.00
24	4.862943798303600	0.00	0.00
25	4.862944051623340	0.00	0.00
26	4.862944178283210	0.00	0.00
27	4.862944114953280	0.00	0.00
28	4.862944146618250	0.00	0.00
29	4.862944130785760	0.00	0.00
30	4.862944138702010	0.00	0.00
31	4.862944134743880	0.00	0.00
32	4.862944132764820	0.00	0.00
33	4.862944131775290	0.00	0.00
34	4.862944131280530	0.00	0.00
35	4.862944131033150	0.00	0.00

The MATLAB program 2 have been given as follows:

```
format long;

% 初始化参数和变量
maxIterations = 50;
targetRoot = 4.8629;
tolerance = 1e-10;
results = zeros(maxIterations, 4);

% 定义要求根的函数
f = @(x) x^3 - 115;

% 初始估计值
x0 = -5;
x1 = 12;

k = 0;
ea = 1;  % 初始化相对误差，确保进入循环

while ea > tolerance && k < maxIterations
    k = k + 1;

    % 牛顿迭代公式
    x = x1 - f(x1) / ((f(x1) - f(x0)) / (x1 - x0));

    % 计算相对误差
    ea = abs((x - x1) / x);

    % 计算真实误差
    et = abs(targetRoot - x);

    % 存储结果
    results(k, :) = [k, x, et, ea];

    % 输出当前迭代结果
    fprintf('迭代次数：%d, 当前估计根：%.10f, 真实误差：%.1f, 相对误
差：%.10f\n', k, x, et, ea);

    % 更新估计值
    x0 = x1;
    x1 = x;
```

```
end

% 保存结果到Excel文件
results = results(1:k, :);  % 截取有效的结果
writematrix(results, "data.xls");
```

1	-1.640390505278130	6.50	0.71
2	6.262231714523180	1.40	1.26
3	2.134484362661380	2.73	1.93
4	3.976951039847440	0.89	0.46
5	5.782160929015960	0.92	0.31
6	4.698109080366260	0.16	0.23
7	4.834823081164500	0.03	0.03
8	4.863923114121210	0.00	0.01
9	4.862938448786820	0.00	0.00
10	4.862944129950500	0.00	0.00
11	4.862944131094280	0.00	0.00
12	4.862944131094280	0.00	0.00

The MATLAB program 3 have been given as follows:

```
format long;

% 初始化参数和变量
maxIterations = 50;
targetRoot = 4.8629;
tolerance = 1e-10;
results = zeros(maxIterations, 4);

% 定义要求根的函数
f = @(x) x^3 - 115;

% 初始估计值
x0 = 2;

k = 0;
ea = 1;  % 初始化相对误差，确保进入循环

while ea > tolerance && k < maxIterations
    k = k + 1;

    % 计算当前估计值处的函数值和导数值
    fx = f(x);
    dfx = 3 * x^2;

    % 牛顿迭代公式
    x_old = x;
    x = x - fx / dfx;

    % 计算相对误差
    ea = abs(x - x_old) / x;

    % 计算真实误差
    et = abs(targetRoot - x);

    % 存储结果
    results(k, :) = [k, x, et, ea];

    % 输出当前迭代结果
    fprintf('迭代次数: %d, 当前估计根: %.15f, 真实误差: %.1f, 相对误
差: %.10f\n', k, x, et, ea);
```

```
end

% 保存结果到Excel文件
results = results(1:k, :);  % 截取有效的结果
writematrix(results, "data.xls");
```

1	10.916666666666700	6.05	0.82
2	7.599437354725510	2.74	0.44
3	5.730055589402850	0.87	0.33
4	4.987542505817770	0.12	0.15
5	4.866030916937960	0.00	0.02
6	4.862946088794970	0.00	0.00
7	4.862944131095070	0.00	0.00
8	4.862944131094280	0.00	0.00

Q3 (B)

Discuss your results. From your results:

- ### (1) does the method converge? how do you know this?

Three methods are convergent. Because the three method's predicted root is closer to the real root after multiple iterations.

1. **Bisection Method:** The first code uses the bisection method. It is a fundamental root-finding method that starts with an interval containing the root and iteratively narrows it down until the root is found. The bisection method halves the length of the interval in each iteration, making it a gradually converging method. It is suitable for continuous and monotonic functions.
2. **Newton-Raphson Method:** The second code utilizes the Newton-Raphson method. This method leverages the derivative of the function and iteratively approaches the root. It often has faster convergence, especially for functions with continuous derivatives. However, it can diverge with poor initial guesses.
3. **Secant Method:** The third code applies the secant method, also known as the chord method. Similar to the Newton-Raphson method, it estimates the derivative value using two neighboring points rather than calculating the derivative. The secant method can serve as an alternative to the Newton-Raphson method in cases where obtaining derivative values is challenging.

- ### (2) what is its convergence order? how do you know this?

1. **Bisection Method:**
   - The bisection method is a linearly convergent method.
   - Its error reduces by a factor of 0.5 with each iteration.
   - This means that after each iteration, the number of correct digits approximately doubles.
   - The convergence order is 1.
2. **Newton-Raphson Method:**
   - The Newton-Raphson method is typically quadratically convergent under certain conditions.
   - Its error decreases by a factor of approximately the square of the previous

    error.
    - This means that the number of correct digits roughly quadruples with each iteration.
    - The convergence order is 2.
    - However, the method's convergence order can decrease to 1 under certain conditions if the derivative becomes zero or if the initial guess is far from the root.
3. **Secant Method:**
    - The secant method is also typically linearly convergent.
    - Its error decreases by a constant factor with each iteration.
    - Like the bisection method, it has a convergence order of 1.
    - However, the secant method can be more efficient than the bisection method if the initial guess is close to the root.
    - **In this case, the secant method has superlinear convergence with convergence order of about 1.6**

• (3) what do you observe about the errors (E_a and E_t)?

1. **Bisection Method:**
    - The code computes the relative error (E_a) and the true error (E_t) at each iteration.
    - The relative error decreases with each iteration, as the interval containing the root is halved in each step.
    - The true error also decreases, but the rate of decrease may not be uniform, as it depends on the position of the midpoint within the interval.
    - The errors generally tend to decrease steadily until they satisfy the stopping criterion $(1e-10)$.
2. **Newton-Raphson Method:**
    - The code computes the relative error (E_a) and the true error (E_t) at each iteration.
    - The relative error generally decreases rapidly with each iteration, indicating quadratic convergence, assuming the method is converging.
    - The true error also decreases, often at a faster rate than the relative error.
    - The method should drive the errors towards zero as it approaches the root.
3. **Secant Method:**
    - Similar to the Newton-Raphson method, the code computes the relative error (E_a) and the true error (E_t) at each iteration.
    - The relative error decreases with each iteration, but this method typically exhibits linear convergence.

    - The true error also decreases, but at a somewhat slower rate compared to the relative error.
    - The method aims to reduce the errors and approach the root, but the convergence rate is linear rather than quadratic.

- ## (4) how much computation (per step and total)?

1. Bisection method need 35 steps, ecah step does one function evaluation
2. Newton's method need 8 steps, ecah step does one function evaluation
3. Secant method need 12 steps, ecah step does one function evaluation

---

- ## (5) how precise are your results? how do you know this?

In a nutshell, the answer is accurate to 6 digits after the decimal point, because the 7th digit after the decimal point and the value after are rounded off.

1. **Bisection Method:**
    - The precision of the bisection method's results is determined by the chosen stopping criterion, in this case, $1e-10$.
    - The method aims to ensure that the relative error (E_a) is smaller than the specified tolerance (1e-10).
    - The results are considered precise up to the specified tolerance. In this case, the results are accurate up to 10 decimal places.
    - We can assess precision by verifying that the relative error is smaller than the specified tolerance for the final result.
2. **Newton-Raphson Method:**
    - The precision of the Newton-Raphson method's results is also determined by the chosen stopping criterion, $1e-10$ in this case.
    - The method aims to ensure that the relative error (E_a) is smaller than the specified tolerance $(1e-10)$.
    - The results are considered precise up to the specified tolerance, accurate to 10 decimal places.
    - Precision is assessed by checking that the relative error is below the specified tolerance for the final result.
3. **Secant Method:**
    - Similar to the other methods, the precision of the secant method's results is governed by the chosen stopping criterion, $1e-10$.
    - The method seeks to ensure that the relative error (E_a) is smaller than the specified tolerance $(1e-10)$.

    - The results are regarded as precise up to the specified tolerance, accurate to 10 decimal places.
    - Precision is confirmed by verifying that the relative error is less than the specified tolerance for the final result.

- ## (6) how do the methods compare?

1. **Bisection Method:**
    - **Convergence:** The bisection method converges linearly, meaning it reduces the error by a fixed factor in each iteration. It is generally slower but robust.
    - **Initial Guess Requirement:** It does not require a very accurate initial guess; it only requires the interval to bracket the root.
    - **Function Characteristics:** Suitable for continuous and monotonic functions with known bracketing intervals.
    - **Computational Complexity:** Simple arithmetic operations per iteration, making it computationally efficient.
    - **Stability:** It is stable and less sensitive to the choice of initial interval.
2. **Newton-Raphson Method:**
    - **Convergence:** The Newton-Raphson method converges rapidly, often quadratically, when close to the root, but can diverge with poor initial guesses.
    - **Initial Guess Requirement:** It requires a reasonably accurate initial guess and continuous differentiability of the function.
    - **Function Characteristics:** Well-suited for functions with continuous derivatives, but may require special handling for certain functions.
    - **Computational Complexity:** Involves more complex computations per iteration, including the evaluation of the derivative.
    - **Stability:** It can be sensitive to the choice of initial guess and may fail to converge in some cases.
3. **Secant Method:**
    - **Convergence:** The secant method generally converges linearly, similar to the bisection method, but can be more efficient if the initial guesses are close to the root.
    - **Initial Guess Requirement:** It does not require a highly accurate initial guess, but a good initial guess can improve efficiency.
    - **Function Characteristics:** Suitable for functions where obtaining the derivative is challenging.
    - **Computational Complexity:** Similar to the Newton-Raphson method, it involves more complex computations per iteration compared to the

bisection method.
- **Stability:** It can be less sensitive to the choice of initial guess compared to the Newton-Raphson method.

In summary, the choice of method depends on the specific problem and the available information. The bisection method is robust but slower, while the Newton-Raphson method and the secant method can converge rapidly with suitable initial guesses but are more sensitive to the choice of initial conditions. The relative computational complexity of the methods can also be a factor in choosing the most appropriate method for a particular problem.

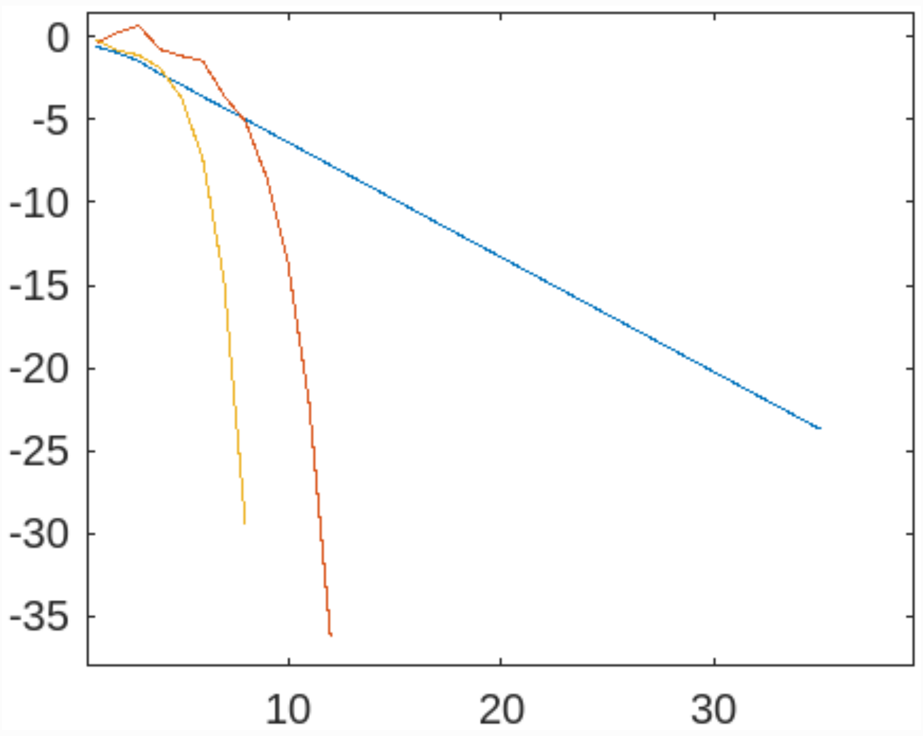
Q3 (C)

The MATLAB program 4 have been given as follows:

```
format long;

% Define the function and the target root
f = @(x) x^3 - 115;
ans = 4.8629;

% Initialize variables
maxIterations = 50;
bisection = zeros(maxIterations, 1);
secant = zeros(maxIterations, 1);
Newton = zeros(maxIterations, 1);

% Bisection Method
a = -5;
b = 12;
k = 0;
x = (a + b) / 2;
ea = abs(b - a) / x;  % Calculate the initial relative error
x_old = x;

while ea > 1e-10 && k < maxIterations
    fx = f(x);
    if f(a) * fx < 0
        b = x;  % Update the right interval boundary
    else
        a = x;  % Update the left interval boundary
    end
    k = k + 1;
    x = (a + b) / 2;  % Calculate the new midpoint
    ea = abs(x - x_old) / x;  % Calculate the relative error
    et = abs(ans - x);  % Calculate the true error
    bisection(k) = ea;  % Store relative error for plotting
end

% Secant Method
x0 = -5;
x1 = 12;
k = 0;
x = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0));  % Calculate the
initial estimate
ea = abs(x - x1) / abs(x);  % Calculate the initial relative
error

while ea > 1e-10 && k < maxIterations
    k = k + 1;
    x0 = x1;
    x1 = x;
    x = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0));  % Calculate
the new estimate
    ea = abs((x - x1) / x);  % Calculate the relative error
    et = abs(ans - x);  % Calculate the true error
    secant(k) = ea;  % Store relative error for plotting
end

% Newton-Raphson Method
x0 = 2;
k = 0;
x = x0;
ea = abs(f(x)) / x;  % Calculate the initial relative error
x_old = x;

while ea > 1e-10 && k < maxIterations
    fx = f(x);
    dfx = 3 * x^2;
    k = k + 1;
    x = x - fx / dfx;  % Calculate the new estimate
    ea = abs(x - x_old) / x;  % Calculate the relative error
    et = abs(ans - x);  % Calculate the true error
    Newton(k) = ea;  % Store relative error for plotting
end

% Plot the results
semilogy(bisection, 'r');
hold on;
semilogy(secant, 'g');
semilogy(Newton, 'b');
legend('Bisection', 'Secant', 'Newton');
xlabel('Iteration');
ylabel('Logarithm of Relative Error');
```

Here's a summary of what the code does:

1. It initializes variables and functions for the three methods (Bisection, Secant, and Newton).
2. The code calculates the relative error for each method during the iterations.
3. It logs the relative errors for each method in the arrays `bisection`, `secant`, and `Newton`.
4. The code then plots the logarithm of these relative errors for visual comparison.



SUMMARY OF THIS ASSIGNMENT

Bisection Method:

- The bisection method is a simple and robust method for finding the root of an equation.
- It operates by narrowing down the root within a given interval by iteratively bisecting the interval.
- The method converges linearly, reducing the error by a fixed factor in each iteration.
- It does not require a very accurate initial guess, only an interval that brackets the root.

Secant Method:

- The secant method is an iterative technique for root finding.
- It approximates the root by fitting a secant line through two points.
- The method generally converges linearly, similar to the bisection method.
- It can be more efficient than the bisection method if the initial guesses are close to the root.
- Like the bisection method, it does not require a highly accurate initial guess.

Newton-Raphson Method:

- The Newton-Raphson method is a highly efficient and rapid convergence method.
- It approximates the root using the tangent line at the current estimate.
- The method can converge quadratically if the initial guess is close to the root.
- It requires a reasonably accurate initial guess and the function's continuous differentiability.
- It can be sensitive to the choice of the initial estimate and may fail to converge in some cases.

These methods differ in terms of their convergence characteristics, initial guess requirements, and sensitivity to the choice of initial conditions. The bisection method is robust but slower, the secant method can be efficient with near-accurate initial guesses, and the Newton-Raphson method is the fastest but demands a relatively accurate initial guess and a continuous differentiable function.