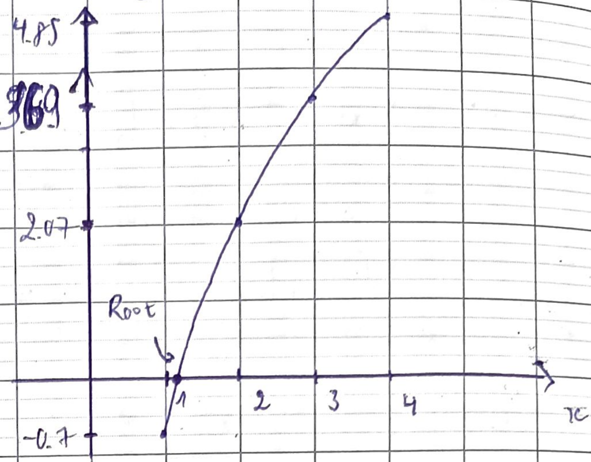
Nguyễn Minh Đức – ITITIU21045 – Lab02 – TMC Lab

Q1: a) Let $f(x) = \ln(x^4) - 0.7$

x	$f(x)$
1	-0.7
2	2.07
3	3.69
4	4.85



b) $x_l = 0.5$, $x_u = 2$

Check $f(x_l)f(x_u) = f(0.5)f(2) < 0$

• First iteration

We have: $x_r = \dfrac{x_l + x_u}{2} = \dfrac{0.5+2}{2} = 1.25$

Check: $f(x_l)f(x_r) < 0$

$\Rightarrow x_u = x_r = 1.25$

• Second iteration

$x_r = \dfrac{x_l + x_u}{2} = 0.875$

Check: $f(x_l)f(x_r) > 0$

$\Rightarrow x_l = x_r = 0.875$

• Third iteration

$x_r = \dfrac{x_l + x_u}{2} = 1.0625$

Check: $f(x_l)f(x_r) > 0$

$\Rightarrow x_l = x_r = 1.0625$

c)

• First iteration

$x_r = x_u - \dfrac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)} = 1.439$

Check: $f(x_l)f(x_r) < 0$

$\Rightarrow x_u = x_r = 1.439$

• Second iteration $x_r = 1.27$

Check $f(x_l)f(x_r) < 0$

$\Rightarrow x_u = x_r = 1.27$

• Third iteration $x_r = 1.217$

Check $f(x_l)f(x_r) < 0$

$\Rightarrow x_u = x_r = 1.217$

Q2) $f(x) = -12x^5 - 6.4x^3 + 12 = g(x)$

• First iteration $x_r = \dfrac{x_l + x_u}{2} = 0.5$

Check: $g(x_l)g(x_r) > 0$

$\Rightarrow x_l = x_r = 0.5$;

• Second iteration $x_r = \dfrac{x_l + x_u}{2} = 0.75$

Check: $g(x_l)g(x_r) > 0$

$\Rightarrow x_l = x_r = 0.75$;

ARE = 33.33%

• Third iteration $x_r = 0.875$

Check $g(x_l)g(x_r) > 0$

$\Rightarrow x_l = x_r = 0.875$;

ARE = 14.29%

• Fourth iteration $x_r = 0.9375$

Check $g(x_l)g(x_r) < 0$

$\Rightarrow x_u = x_r = 0.9375$

ARE = 6.67%

• Fifth iteration $x_r = 0.90625$

Check $g(x_l)g(x_r) < 0$

$\Rightarrow x_u = x_r = 0.90625$

$ARE = 3.45\% < 5\%$

So after 5 iterations, the ARE falls below 5%

Q3:

Exact value: $f(0.5) = 0.0295229$

• We have $f'(x) = 4x^3 e^{-3x^2} - 6x^5 e^{-3x^2}$; $h = 0.5 - 1 = -0.5$

Using Taylor's series first order: $f(x_{i+1}) = f(x_i) + f'(x_i)h$

$\Rightarrow f(0.5) = f(1) + f'(1)(-0.5) = 0.0995741$

Therefore, the exact value is smaller than the first order of Taylor Series

Q4:

a) $x^{3.5} = 80$

$\Rightarrow x = \sqrt[3.5]{80} = 3.4973$

b) Let $f(x) = x^{3.5} - 80$

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$ with $x_l = 2$; $x_u = 5$

$$x_r = 5 - \frac{f(5)(2-5)}{f(2) - f(5)} = 2.7683$$

```
import math


# Function for finding roots
def f(x):
  return math.log(x) - 0.7


# Bisection Method
def bisection(xl,xu,iter):
  for i in range(iter):
    xr = (xl+xu)/2.0
    if (f(xl)*f(xr) < 0):
      xu = xr
    else:
      xl = xr
    print(f"Iteration {i+1}: xr = {xr}")


# False-Position Method
def false_position(xl,xu,iter):
  for i in range(iter):
    xr = xu - (f(xu)*(xl-xu))/(f(xl)-f(xu))
    if (f(xl)*f(xr) < 0):
      xu = xr
    else:
      xl = xr
    print(f"Iteration {i+1}: xr = {xr}")


# Initial guesses and iterations
xl_init = 0.5
xu_init = 2
iteration = 3


# Bisection Method
print("\nBisection Method: ")
bisection(xl_init,xu_init,iteration)
```

```
Bisection Method:
Iteration 1: xr = 1.25
Iteration 2: xr = 1.625
Iteration 3: xr = 1.8125
```

```
# False-Position Method
print("\nFalse-Position Method: ")
false_position(xl_init,xu_init,iteration)
```

```
False-Position Method:
Iteration 1: xr = 2.0074148964667056
Iteration 2: xr = 2.0137310296836053
Iteration 3: xr = 2.0137526333627425
```

```
def f(x):
  return -2*x**6 - 1.6*x**4 + 12*x + 1


def bisection_max(xl,xu,tolerance):
  while True:
    xr = (xl+xu)/2.0

    # Check if xr is close enough to the actual maximum
    if abs((xu-xl)/xr) < tolerance:
      break

    if f(xr) > 0:
      xl = xr
    else:
      xu = xr

  return xr, f(xr)


# Initial guesses and tolerance
xl_init = 0
xu_init = 1
tolerance = 0.05 # 5% relative error


# Find the maximum using the bisection method
max_estimate, max_value = bisection_max(xl_init,xu_init,tolerance)
print(f"Maximum estimate: x = {max_estimate}, f(x) = {max_value}")
```

```
Maximum estimate: x = 0.984375, f(x) = 9.490507160843118
```

```
import sympy as sp


# Define the symbolic variable and the function
x = sp.symbols('x')
fx = x**4 * sp.exp(-3*x**2)


# Calculate the exact value at x = 0.5
exact_value = fx.subs(x, 0.5)


# Calculate the first-order of Taylor series approximation
x_0 = 1
first_order_approx = sp.series(fx, x, x0 = x_0, n=2).removeO().subs(x, 0.5)

Print the results
int(f"Exact value at x = 0.5: {exact_value}")
int(f"First-order Taylor series approximation at x = 0.5: {first_order_approx}")
```

```
Exact value at x = 0.5: 0.0295229095463134
First-order Taylor series approximation at x = 0.5: 2.0*exp(-3)
```

```
# Analytically
analytically = 80 ** (1/3.5)
print(f"Analytically: {analytically}")
```

```
Analytically: 3.4973572431802795
```

```
def f(x):
  return x ** 3.5 - 80


def false_position_method(xl, xu, tolerance):
  while True:
    xr = xu - (f(xu) * (xl - xu)) / (f(xl) - f(xu))

    # Check if xr is close enough to the actual root
    if abs(f(xr)) < tolerance:
        break
    if f(xl) * f(xr) < 0:
        xu = xr
    else:
        xl = xr
  return xr, f(xr)


# Initial guesses and tolerance
xl_init = 2.0
xu_init = 5.0
tolerance = 0.025  # 2.5% relative error


# Find the root using the false-position method
root_estimate, root_value = false_position_method(xl_init, xu_init, tolerance)

print(f"Root estimate: x = {root_estimate}, f(x) = {root_value}")
```

```
Root estimate: x = 3.4971436569869283, f(x) = -0.0170985017447407
```