EM215 - Assignment2 - E/20/280

```
import numpy as np
import matplotlib.pyplot as plt
```

a) bisection method

```
# Basic function
def f(x):
    return (4 * (x**3) - 6 * (x**2) + 7 * x - 2.3)
```

```
# Solving using bisection method
def biSectionMethod(xl, xu, eMax):
    # Absolute error
    absE = []
    #check if inital poins are the root
    if f(xl) == 0:
        return xl, absE
    elif f(xu) == 0:
        return xu, absE
    # Check if initail values are wrong
    elif (f(xl) * f(xu) > 0):
        print("Initial guses are invalid!")
        return
    else:
        # List of absolute errors
        while True:
            # Taking the new estimate value
            xr = (xl + xu) / 2

            # Error calculation
            Ea = abs((xl - xu) / 2)
            absE.append(Ea)

            # Percentage error
            precE = abs((Ea / xr) * 100)

            # Check for stopping criteria
            if (precE <= eMax):
                return xr, absE

            # Returning the answer
            if (f(xl) == 0):
                return xl, absE
            if (f(xu) == 0):
                return xu, absE

            # replacing xu, xl
            if (f(xl) * f(xr) < 0):
                xu = xr
            elif (f(xu) * f(xr) < 0):
                xl = xr
```

now calling the function with initial values

```
# biSectionMethod calcuations
xl = 0
xu = 1
ans, biSectionMethodE = biSectionMethod(xl, xu, 0.5)
print(ans)
```

```
0.451171875
```

b) Fixed poin iteration method

```
def g(x):
    return ((6 * (x**2) - 7 * x + 2.3) / 4)**(1/3)
```

```
# Solving by Fixed point iteration method
def fixedPointIterationMethod(x0, eMax):
    # initial value
    xn = x0

    # Absolute error
    absE = []

    while True:
        # Calculate Xn+1
        xn1 = g(xn)

        # store errors
        Ea = abs(xn1 - xn)
        absE.append(Ea)

        # Percentage error
        e = abs(Ea/xn1) * 100

        # Stoping criteria
        if (e <= eMax):
            return xn1, absE

        # For next iteration
        xn = xn1
```

```
# fixed point iteration method calcuations
x0 = 0
ans, fixedPointIterationMethodE = fixedPointIterationMethod(x0, 0.5)
print(ans)
```

```
0.4494624722312006
```

c) Newton raphson method

```
# derivative function
def f1(x):
    return (12 * (x**2) - 12 * x + 7)
```

```
# Newton raphson method
def newtonRephsonMethod(x0, eMax):
    # initial value
    xn = x0

    # Absolute error
    absE = []

    while True:
        # Calculate Xn+1
        xn1 = xn - (f(xn) / f1(xn))

        # store errors
        Ea = abs(xn1 - xn)
        absE.append(Ea)

        # Percentage error
        e = abs(Ea/xn1) * 100

        # Stoping criteria
        if (e <= eMax):
            return xn1, absE

        # For next iteration
        xn = xn1
```

```
# fnewton raphson method calcuations
x0 = -0.5
ans, newtonRephsonMethodE = newtonRephsonMethod(x0, 0.5)
print(ans)
```

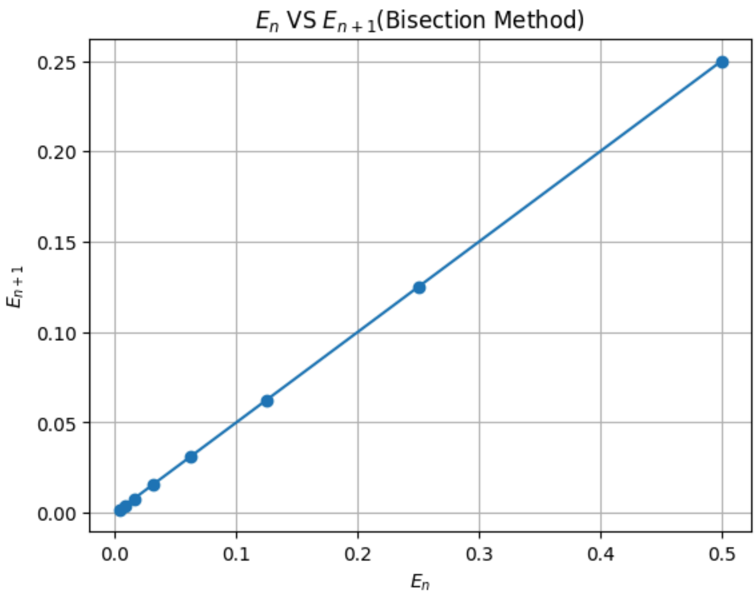
```
0.4501240717634304
```

d) Comparison between methods

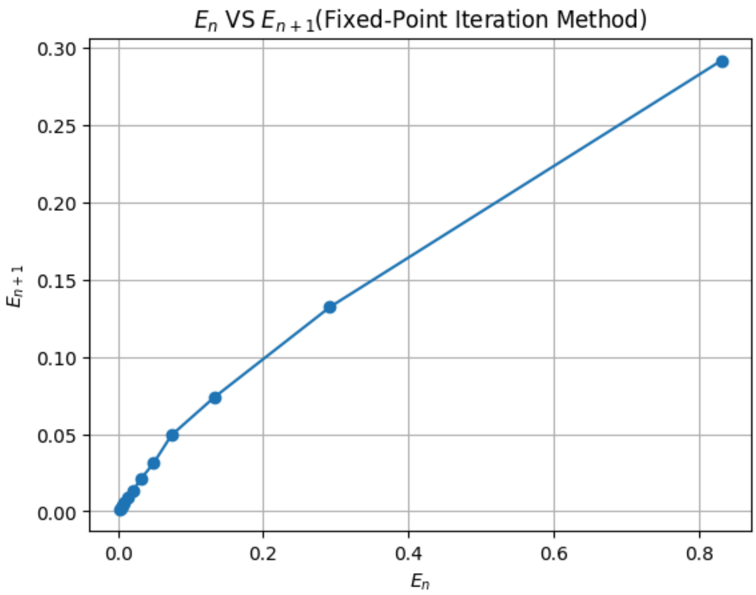
By carefully observing each graph, it can be observed, from Bisection mrthod and Fixed point iteration method, the graphs converged linearly. But in the Newton raphson method, the graph has converge quadratically.

```
# Plotting any method's result
plt.figure()

En = biSectionMethodE[:-1]
En1 = biSectionMethodE[1:]
#En1 vs En
plt.plot(En, En1, marker = 'o')
plt.title('$E_{n}$ VS $E_{n+1}$(Bisection Method)')
plt.xlabel('$E_{n}$')
plt.ylabel('$E_{n+1}$')
plt.grid(True)
plt.show()
```




```
En = fixedPointIterationMethodE[:-1]
En1 = fixedPointIterationMethodE[1:]
#En1 vs En
plt.plot(En, En1, marker = 'o')
plt.title('$E_{n}$ VS $E_{n+1}$(Fixed-Point Iteration Method)')
plt.xlabel('$E_{n}$')
plt.ylabel('$E_{n+1}$')
plt.grid(True)
plt.show()
```



```
En = newtonRephsonMethodE[:-1]
En1 = newtonRephsonMethodE[1:]
#En1 vs En
plt.plot(En, En1, marker = 'o')
plt.title('$E_{n}$ VS $E_{n+1}$(Newton Raphson Method)')
plt.xlabel('$E_{n}$')
plt.ylabel('$E_{n+1}$')
plt.grid(True)
plt.show()
```