Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

Solution 1

Problem I: $f_1(x) = x + e^{-x^2}\cos x$

Problem II: $f_2(x) = \left(x + e^{-x^2}\cos x\right)^2$

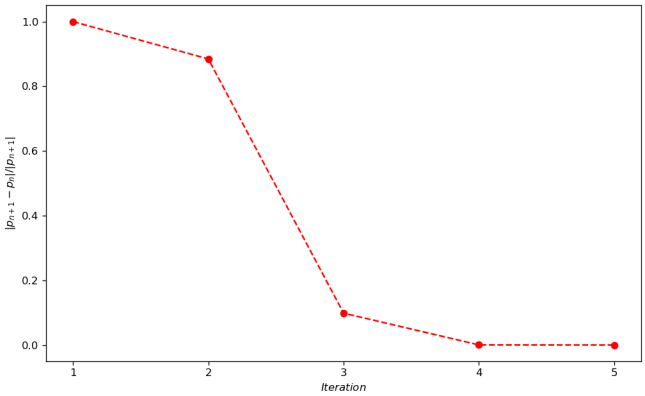
Numerical Order of Convergence: $\alpha = \left(\log \frac{\epsilon_{i+1}}{\epsilon_i}\right)\left(\log \frac{\epsilon_i}{\epsilon_{i-1}}\right)^{-1}$

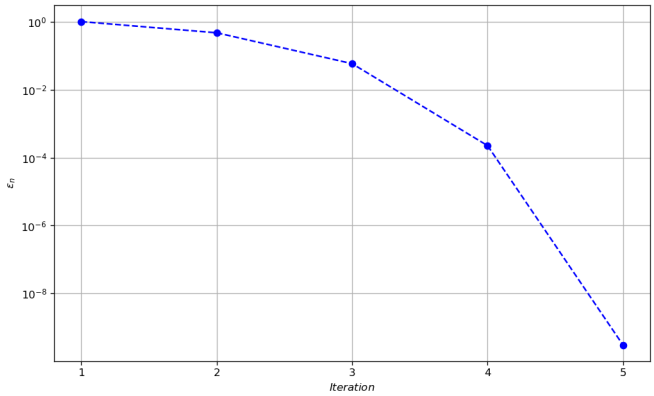
Assumptions: For Newton's Method and Modified Newton's Method, Initial guess is $p_0 = 0$ and for Secant Method, Initial guesses are $p_0 = 0$ and $p_1 = 1$. Relative tolerance is 10^{-6}.

(a). Newton's Method:

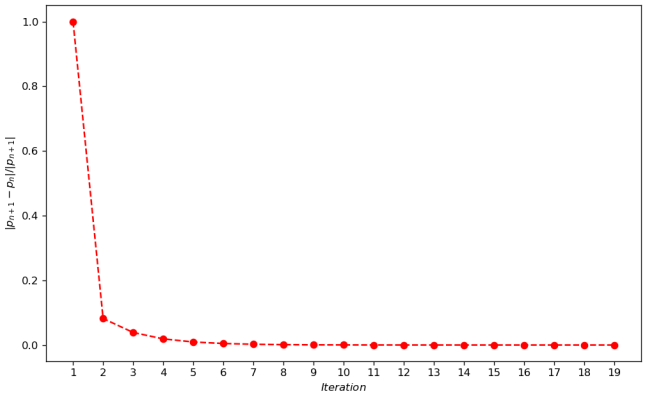
$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}$$

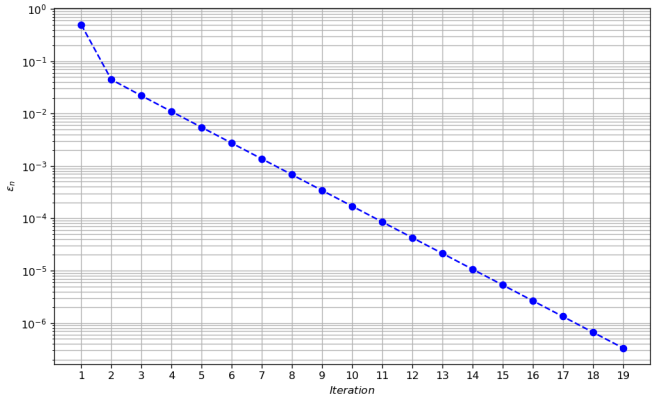
For Problem 1,





For Problem 2,





Analysis:

1. For Problem 1, it is easily observable that only 5 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases quadratically i.e. of order 2. Hence, we can see the same trend as quadratic convergence which is derived analytically. We can't see the exact value of α as derived analytically because of scarcity of iterations.
2. For Problem 2, it is observable that 19 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases linearly after iteration 2 itself i.e. of order 1. Hence, we can see the same trend as linear convergence which is derived analytically.

Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

Results: For Newton's Method,

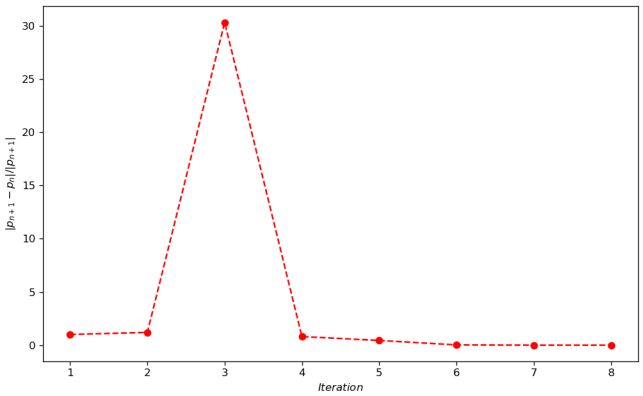
1. ***Problem 1:***
   No. of Iterations: **5**
   Relative Tolerance: **5.21826642749437e-10**
   Final Value of p: **-0.588401776500996**
   Final Value of $f(p)$: **-1.11022302462516e-16**
   Order of Convergence: **2.43180511487112** (Analytically, $\alpha = 2$)

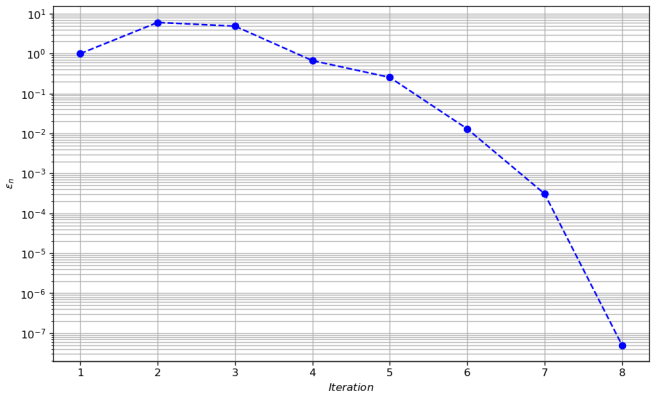
2. ***Problem 2:***
   No. of Iterations: **19**
   Relative Tolerance: **5.66145289942856e-7**
   Final Value of p: **-0.588401443380482**
   Final Value of $f(p)$: **4.82427157114461e-13**
   Order of Convergence: **0.9999999820894209** (Analytically, $\alpha = 1$)

(b). Secant Method:

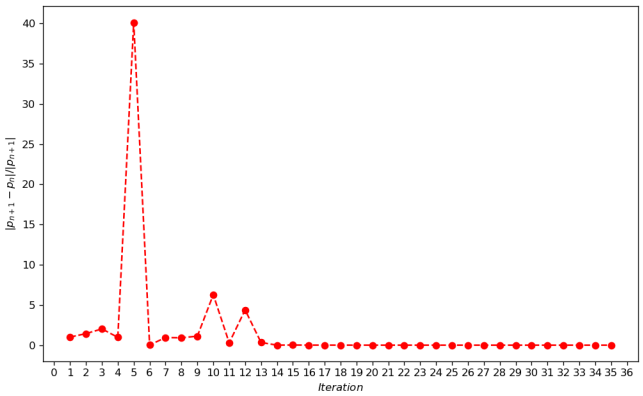
$$p_{n+1} = p_n - \frac{f(p_n)(p_n - p_{n-1})}{f(p_n) - f(p_{n-1})}$$

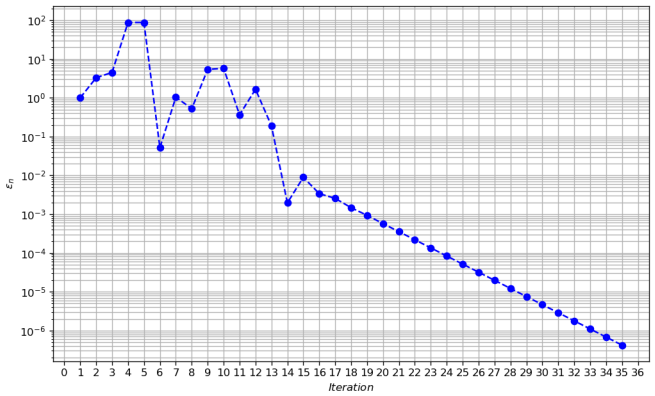
For Problem 1,





For Problem 2,





Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

Analysis:

1. For Problem 1, it is easily observable that only 8 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases almost quadratically i.e. of order between 1 and 2. Hence, we can see the same trend as superlinear convergence which is derived analytically. We can't see the exact value of α as derived analytically because of scarcity of iterations.

2. For Problem 2, it is observable that 35 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases linearly after iteration 17 i.e. of order 1. Hence, we can see the same trend as linear convergence which is derived analytically.

Results: For Secant Method,

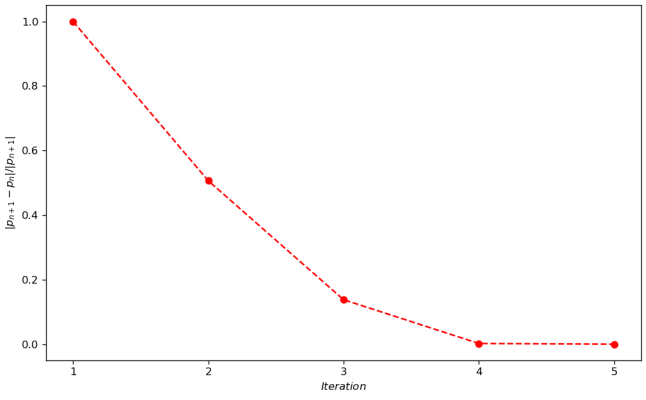
1. ***Problem 1:***
   No. of Iterations: **8**
   Relative Tolerance: **8.50150762073084e-8**
   Final Value of p: **-0.588401776500901**
   Final Value of $f(p)$: **1.98063787593128e-13**
   Order of Convergence: **2.3296554236837026** (Analytically, $\alpha = 1.618$)

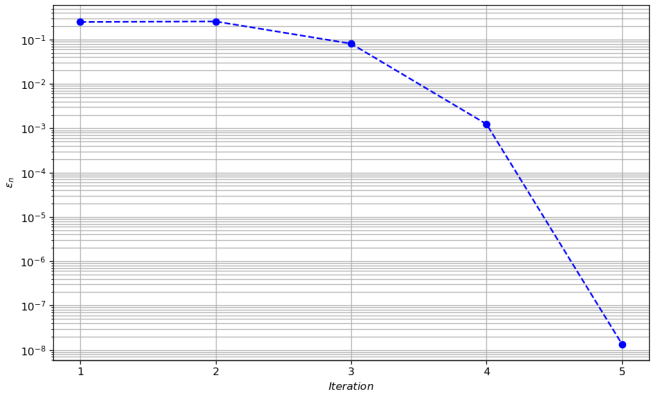
2. ***Problem 2:***
   No. of Iterations: **35**
   Relative Tolerance: **7.12723991679115e-7**
   Final Value of p: **-0.588401097950494**
   Final Value of $f(p)$: **2.00167397132503e-12**
   Order of Convergence: **0.9999999388622839** (Analytically, $\alpha = 1$)

(c). Modified Newton's Method:

$$p_{n+1} = p_n - \frac{f(p_n)f'(p_n)}{(f'(p_n))^2 - f(p_n)f''(p_n)}$$

For Problem 1,





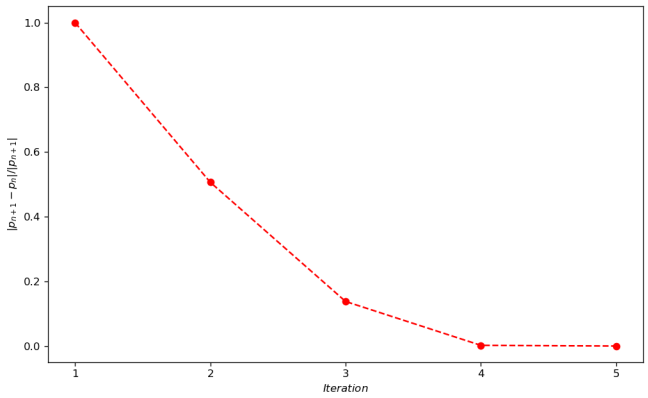
Analysis:

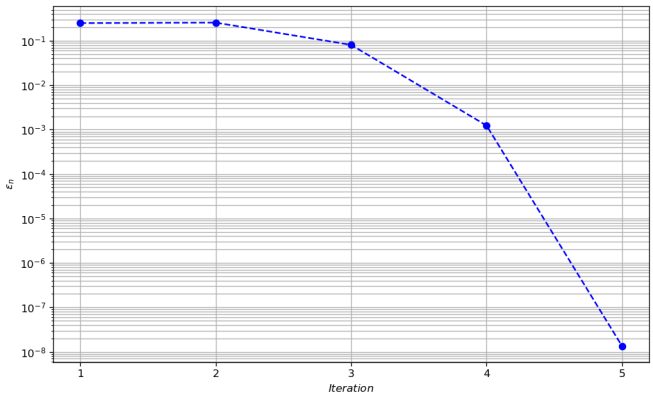
1. For Problem 1, it is easily observable that only 5 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases quadratically i.e. of order 2. Hence, we can see the same trend as quadratic convergence which is derived analytically. We can't see the exact value of α as derived analytically because of scarcity of iterations.

2. For Problem 2, it is easily observable that only 5 iterations are required for reaching the desired tolerance and from log plot we can see that the graph decreases quadratically i.e. of order 2. Hence, we can see the same trend as quadratic convergence which is derived analytically. We can't see the exact value of α as derived analytically because of scarcity of iterations.

Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

For Problem 2,





Results: For Modified Newton's Method,

1. ***Problem 1:***
   No. of Iterations: **5**
   Relative Tolerance: **2.28753315742628e-8**
   Final Value of p: **-0.588401776500996**
   Final Value of f(p): **-1.11022302462516e-16**
   Order of Convergence: **2.7322000706575094** (Analytically, $\alpha = 2$)

2. ***Problem 2:***
   No. of Iterations: **5**
   Relative Tolerance: **2.28753315742628e-8**
   Final Value of p: **-0.588401776500996**
   Final Value of f(p): **1.23259516440783e-32**
   Order of Convergence: **2.7322000706575085** (Analytically, $\alpha = 2$)

Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

Solution 2

Derivation of Scheme

Given fixed point iteration scheme is of type,

$$g(x) = x - \phi(x)f(x) - \psi(x)f^2(x) \tag{1}$$

We have to solve for $f(x) = 0$, i.e. at nth iteration, $p_{n+1} = g(p_n)$ (computational scheme) for fixed point p which analytically corresponds to $p = g(p)$.
Now, expanding $g(p_n)$ in Taylor Series around p we get,

$$g(p_n) = g(p) + g'(p)(p_n - p) + \frac{g''(p)}{2!}(p_n - p)^2 + \frac{g'''(p)}{3!}(p_n - p)^3 + \dots \tag{2}$$

which is equivalent to,

$$\epsilon_{n+1} = g'(p)\epsilon_n + \frac{g''(p)}{2!}\epsilon_n^2 + \frac{g'''(p)}{3!}\epsilon_n^3 + \dots \tag{3}$$

Now, we have to find $\phi(x)$ and $\psi(x)$ such that $f(x) = 0$, i.e., $g(p) = p$.
Now, differentiating equation (1) with respect to x we get,

$$g'(x) = 1 - \phi'(x)f(x) - \phi(x)f'(x) - \psi'(x)f^2(x) - 2\psi(x)f(x)f'(x) \tag{4}$$

[1]**Assume** $g'(p) = 0$. Now, for equation (4), when $x = p$ we get,

$$\cancelto{0}{g'(p)} = 1 - \cancelto{0}{\phi'(p)f(p)} - \phi(p)f'(p) - \cancelto{0}{\psi'(p)f^2(p)} - \cancelto{0}{2\psi(p)f(p)f'(p)}$$

$$\implies \phi(p) = \frac{1}{f'(p)} \quad \equiv \quad \phi(x) = \frac{1}{f'(x)} \tag{5}$$

Differentiating equation (5) with respect to x we get,

$$\phi'(x) = -\frac{f''(x)}{(f'(x))^2} \quad \equiv \quad \phi'(p) = -\frac{f''(p)}{(f'(p))^2} \tag{6}$$

Now, differentiating again equation (4) with respect to x we get,

$$g''(x) = -\phi''(x)f(x) - 2\phi'(x)f'(x) - \phi(x)f''(x) - \psi''(x)f^2(x) - 4\psi'(x)f(x)f'(x) - 2\psi(x)(f'(x))^2 - 2\psi(x)f(x)f''(x) \tag{7}$$

[2]**Assume** $g''(p) = 0$. Now, for equation (7), when $x = p$ we get,

$$\cancelto{0}{g''(p)} = -\cancelto{0}{\phi''(p)f(p)} - 2\phi'(p)f'(p) - \phi(p)f''(p) - \cancelto{0}{\psi''(p)f^2(p)} - \cancelto{0}{4\psi'(p)f(p)f'(p)} - 2\psi(p)(f'(p))^2 - \cancelto{0}{2\psi(p)f(p)f''(p)}$$

$$\implies -2\phi'(p)f'(p) - \phi(p)f''(p) = 2\psi(p)(f'(p))^2$$

substituting values of $\phi(p)$ from equation (5) and $\phi'(p)$ from equation (6) we get,

$$2f'(p)\frac{f''(p)}{(f'(p))^2} - \frac{f''(p)}{f'(p)} = 2\psi(p)(f'(p))^2$$

$$\implies \psi(p) = \frac{f''(p)}{2(f'(p))^3} \quad \equiv \quad \psi(x) = \frac{f''(x)}{2(f'(x))^3} \tag{8}$$

Substituting values of $\psi(x)$ from equation (8) and $\phi(x)$ from equation (5) to equation (1) we get the final fixed point iterative scheme as follows,

$$g(x) = x - \frac{f(x)}{f'(x)} - \frac{f^2(x)f''(x)}{2(f'(x))^3} = x - \frac{f(x)}{f'(x)}\left(1 + \frac{f(x)f''(x)}{2(f'(x))^2}\right) \tag{9}$$

[1]Assumption 1
[2]Assumption 2

Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

The final fixed point iterative scheme will be,

$$p_{n+1} = g(p_n) = p_n - \frac{f(p_n)}{f'(p_n)}\left(1 + \frac{f(p_n)f''(p_n)}{2(f'(p_n))^2}\right) \qquad \text{where, } f'(p_n) \neq 0 \tag{10}$$

Convergence Analysis

From Equation (3) along with Assumptions 1 and 2, it is clear that,

$$\epsilon_{n+1} = \cancelto{0}{g'(p)\epsilon_n} + \cancelto{0}{\frac{g''(p)}{2!}}\epsilon_n^2 + \frac{g'''(p)}{3!}\epsilon_n^3 + \cdots = \frac{g'''(p)}{3!}\epsilon_n^3 + \frac{g''''(p)}{4!}\epsilon_n^4 + \cdots = \epsilon_n^3\left(\frac{g'''(p)}{3!} + \frac{g''''(p)}{4!}\epsilon_n + \dots\right) \tag{11}$$

As we know for general scheme with asymptotic convergence rate α,

$$\lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^\alpha} = \lambda \tag{12}$$

where, α is the asymptotic rate of convergence and λ is asymptotic error constant. Also, we know that $\epsilon_n \to 0$ as $n \to \infty$.

Hence, using equation (11) and definition (12), we get,

$$\lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^3} = \frac{g'''(p)}{3!} \tag{13}$$

For $g'''(p)$, we use equations (5), (8) and (7),

$$g'''(p) = -3\phi''(p)f'(p) - 3\phi'(p)f''(p) - 6\psi'(p)\left(f'(p)\right)^2 - 6\psi(p)f'(p)f''(p) - \phi(p)f'''(p) \tag{14}$$

As we know,

$$\phi''(p) = \frac{-f'(p)f'''(p) + 2\left(f''(p)\right)^2}{\left(f'(p)\right)^3} \qquad \text{and, } \psi'(p) = \frac{f'(p)f'''(p) - 3\left(f''(p)\right)^2}{2\left(f'(p)\right)^4} \tag{15}$$

Substituting values of $\phi(p), \phi'(p), \phi''(p), \psi(p), \psi'(p)$ in equation (14) we get,

$$g'''(p) = \frac{3\left(f''(p)\right)^2 - f'(p)f'''(p)}{\left(f'(p)\right)^2} \tag{16}$$

From equation (13) it is clear that for fixed point iterative scheme defined in equation (10), we have,

1. **Asymptotic Rate of Convergence:** $\alpha = 3$ i.e. **Cubic Convergence**.
2. **Asymptotic Error Constant:** $\lambda = \frac{g'''(p)}{3!} = \frac{3\left(f''(p)\right)^2 - f'(p)f'''(p)}{6\left(f'(p)\right)^2}$ i.e. **Costant** for cubic convergence.

Name:	Aneesh Panchal	**Homework No:**	Homework 2
SR No:	06-18-01-10-12-24-1-25223	**Course Code:**	DS288
Email ID:	aneeshp@iisc.ac.in	**Course Name:**	Numerical Methods
Date:	September 22, 2024	**Term:**	AUG 2024

Solution 3

Given system of equations is as follows,

$$f_1(\theta_2, \theta_3) = 6\cos\theta_2 + 8\cos\theta_3 + 4\cos 220^o - 10 = 0$$

$$f_2(\theta_2, \theta_3) = 6\sin\theta_2 + 8\sin\theta_3 + 4\sin 220^o = 0$$

Initial conditions: $\theta_2 = 30^o$ and $\theta_3 = 0^o$
Relative Tolerance: 10^{-4} (here, we assumed infinite norm of relative tolerance vector.)

Formula used are as follows,

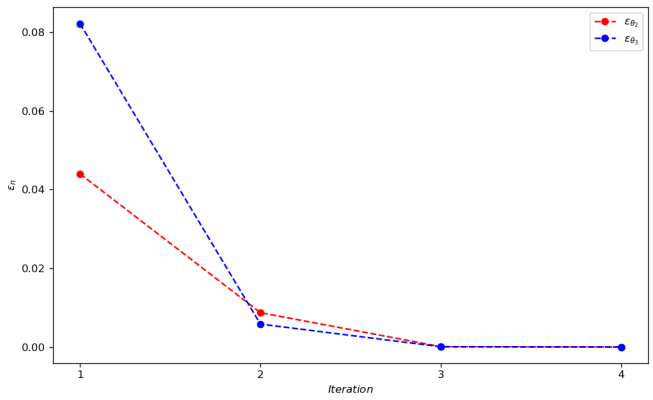
$$f(\theta_2, \theta_3) = \begin{bmatrix} 6\cos\theta_2 + 8\cos\theta_3 - 13.0641777724759 \\ 6\sin\theta_2 + 8\sin\theta_3 - 2.57115043874616 \end{bmatrix}$$

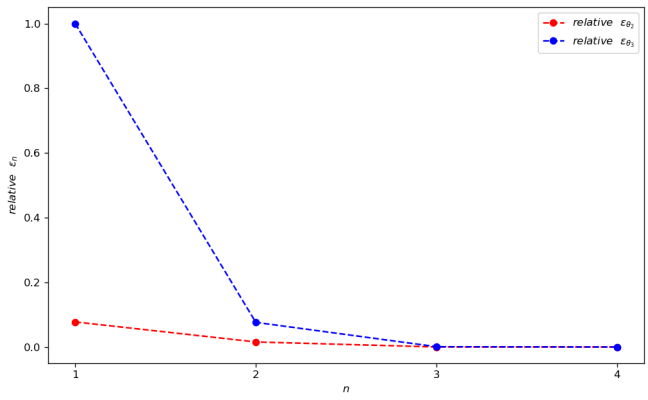
$$J_f^{-1}(\theta_2, \theta_3) = \begin{bmatrix} \frac{-\cos\theta_3}{6\sin(\theta_2-\theta_3)} & \frac{-\sin\theta_3}{6\sin(\theta_2-\theta_3)} \\ \frac{\cos\theta_2}{8\sin(\theta_2-\theta_3)} & \frac{\sin\theta_2}{8\sin(\theta_2-\theta_3)} \end{bmatrix}$$

Newton's Method for solving the system of equation is as follows,

$$X_{n+1} = X_n - J_f^{-1}(X_n) f(X_n)$$

$$\begin{bmatrix}\theta_2 \\ \theta_3\end{bmatrix}_{n+1} = \begin{bmatrix}\theta_2 \\ \theta_3\end{bmatrix}_n - \begin{bmatrix} \frac{-\cos\theta_3}{6\sin(\theta_2-\theta_3)} & \frac{-\sin\theta_3}{6\sin(\theta_2-\theta_3)} \\ \frac{\cos\theta_2}{8\sin(\theta_2-\theta_3)} & \frac{\sin\theta_2}{8\sin(\theta_2-\theta_3)} \end{bmatrix}_n \begin{bmatrix} 6\cos\theta_2 + 8\cos\theta_3 - 13.0641777724759 \\ 6\sin\theta_2 + 8\sin\theta_3 - 2.57115043874616 \end{bmatrix}_n$$





Analysis:

We have used radians here for the computations as the inbuilt functions of python only take radians as input and radians are used to maps the circle physically and hence, are more usable and carry more physical significance than the degrees.
Here, Newton's method is used to solve the system of equations and hence, we can see the convergence in 4 steps with relative error infinity norm with thresold relative error tolerance as 10^{-4}. Newton method shows quadratic convergence rate i.e. order $\alpha = 2$.

Results: Numerically Solving System of Equations,

No. of Iterations: **4**
Max. Relative Tolerance: **1.11372373822926e-7**
Final Value of p (in radian): [0.558770307890069, -0.0762881217804664]
Final Value of p (in degrees): [32.0151803593265, 355.629012594999]
Final Value of $f(p)$: [0, -4.44089209850063e-16]

Hence, final value are $\theta_2 = \mathbf{32.0151803593265^o}$ and $\theta_3 = \mathbf{355.629012594999^o}$

1 Appendix: Assignment 2 Programming

1.1 Ques 1

[1]:
```
##########################################################################
import sympy as sym
import math
from IPython.display import display, Math, Markdown
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

x = sym.symbols('x')

##########################################################################
def f1(x):
    return x + sym.cos(x)*sym.exp(-x**2)
def f1p(x):
    return sym.diff(f1(x),x)
def f1pp(x):
    return sym.diff(f1p(x),x)

##########################################################################
def f2(x):
    return (x + sym.cos(x)*sym.exp(-x**2))**2
def f2p(x):
    return sym.diff(f2(x),x)
def f2pp(x):
    return sym.diff(f2p(x),x)
```

1.2 Newton Method

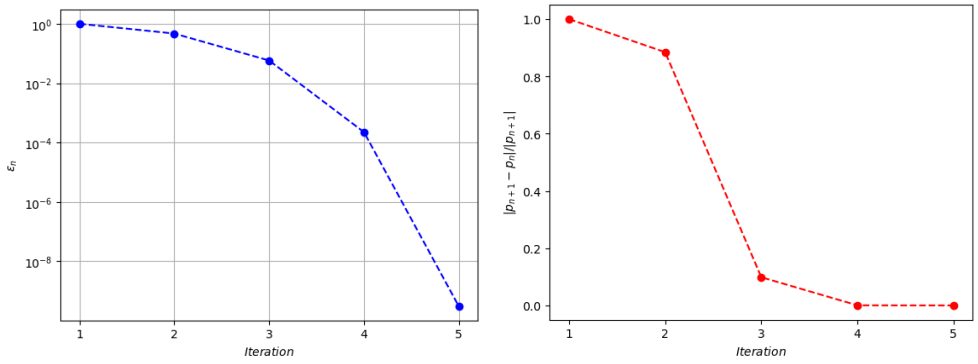
[2]:
```
##########################################################################
p = [0]
tol = 1e-6
iter = 0
epsilon = []
while True:
    p_new = p[iter] - (f1(x).subs(x,p[iter])/f1p(x).subs(x,p[iter]))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter+1])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)
```

```
print("\nPart (a)\n")
print("No. of Iterations: ", iter)
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
print("Final Value of f(p): ", f1(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.yscale("log")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()

##########################################################################
p = [0]
tol = 1e-6
iter = 0
epsilon = []
while True:
    p_new = p[iter] - (f2(x).subs(x,p[iter])/f2p(x).subs(x,p[iter]))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter+1])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)

print("\n\nPart (b)\n")
print("No. of Iterations: ", iter)
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
```

```
print("Final Value of f(p): ", f2(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.yscale("log")
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()
```

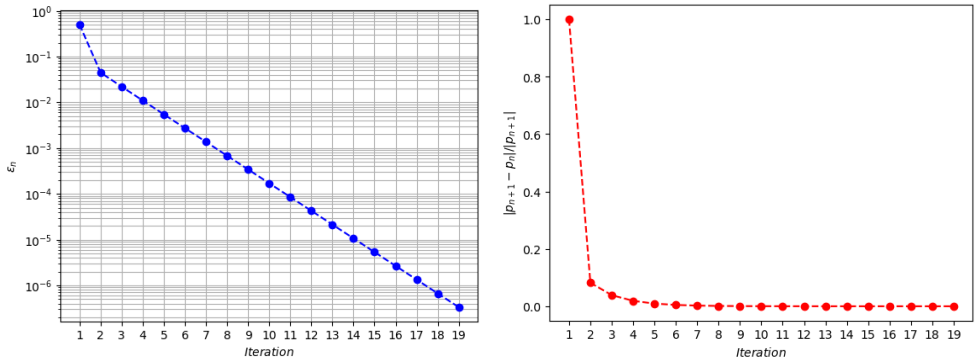
Part (a)

```
No. of Iterations:  5
Relative Tolerance:  5.21826642749437e-10
Final Value of p:  -0.588401776500996
Final Value of f(p):  -1.11022302462516e-16
Order of Convergence:  2.43180511487112
```



Part (b)

```
No. of Iterations:  19
Relative Tolerance:  5.66145289942856e-7
Final Value of p:  -0.588401443380482
Final Value of f(p):  4.82427157114461e-13
Order of Convergence:  0.9999999820894209
```



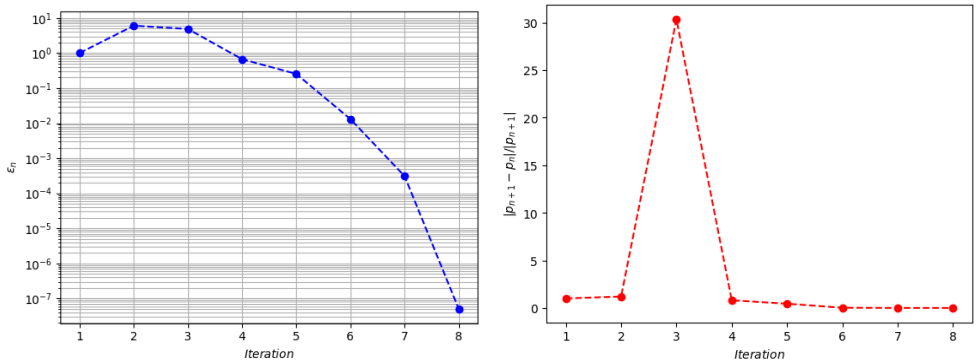
1.3 Secant Method

[3]:
```
##########################################################################
p = [0,1]
tol = 1e-6
iter = 1
epsilon = [1]

def f1val(num):
    return f1(x).subs(x,num)

while True:
    p_new = p[iter] - ((f1val(p[iter])*(p[iter] - p[iter-1]))/(f1val(p[iter]) -␣
 ↪f1val(p[iter-1])))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter+1])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)

print("\nPart (a)\n")
print("No. of Iterations: ", iter)
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
print("Final Value of f(p): ", f1(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
```

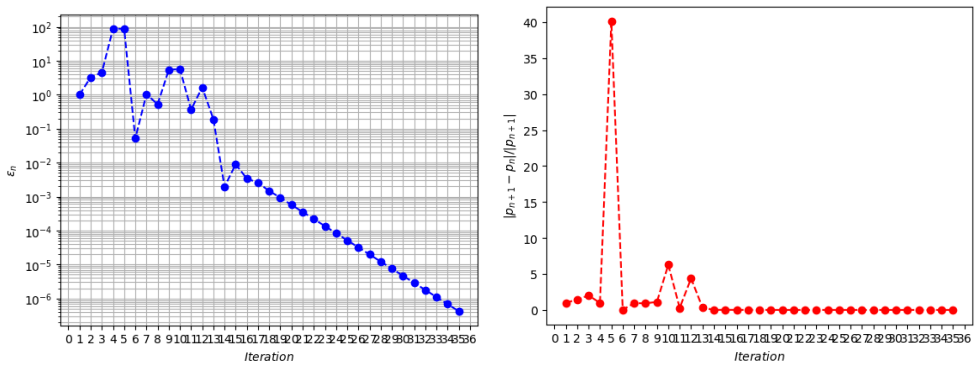
```
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.yscale("log")
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()

##########################################################################
p = [0,1]
tol = 1e-6
iter = 1
epsilon = [1]

def f2val(num):
    return f2(x).subs(x,num)

while True:
    p_new = p[iter] - ((f2val(p[iter])*(p[iter] - p[iter-1]))/(f2val(p[iter]) -␣
 ↪f2val(p[iter-1])))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter+1])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)

print("\n\nPart (b)\n")
print("No. of Iterations: ", iter)
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
print("Final Value of f(p): ", f2(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
```

```
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.yscale("log")
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()
```

```
Part (a)

No. of Iterations:  8
Relative Tolerance:  8.50150762073084e-8
Final Value of p:  -0.588401776500901
Final Value of f(p):  1.98063787593128e-13
Order of Convergence:  2.3296554236837026
```




```
Part (b)

No. of Iterations:  35
Relative Tolerance:  7.12723991679115e-7
Final Value of p:  -0.588401097950494
Final Value of f(p):  2.00167397132503e-12
Order of Convergence:  0.9999999388622839
```



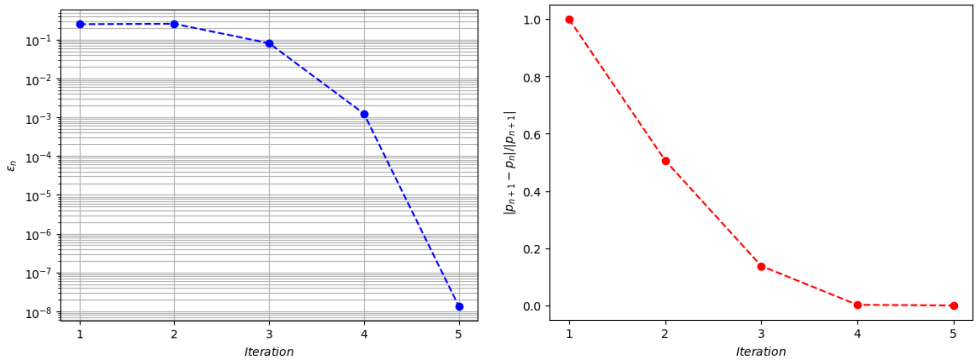
1.4 Modified Newton Method

[4]:
```
##########################################################################
p = [0]
tol = 1e-6
iter = 0
epsilon = []

def f1val(num):
    return f1(x).subs(x,num)
def f1pval(num):
    return f1p(x).subs(x,num)
def f1ppval(num):
    return f1pp(x).subs(x,num)

while True:
    p_new = p[iter] - ((f1val(p[iter])*f1pval(p[iter]))/
 ↪(f1pval(p[iter])*f1pval(p[iter]) - f1val(p[iter])*f1ppval(p[iter])))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)

print("\nPart (a)\n")
print("No. of Iterations: ", iter)
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
print("Final Value of f(p): ", f1(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
```

```
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.yscale("log")
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()

##########################################################################
p = [0]
tol = 1e-6
iter = 0
epsilon = []

def f2val(num):
    return f2(x).subs(x,num)
def f2pval(num):
    return f2p(x).subs(x,num)
def f2ppval(num):
    return f2pp(x).subs(x,num)

while True:
    p_new = p[iter] - ((f2val(p[iter])*f2pval(p[iter]))/
 ↪(f2pval(p[iter])*f2pval(p[iter]) - f2val(p[iter])*f2ppval(p[iter])))
    p_new = sym.N(p_new)
    p.append(p_new)
    epsilon.append(abs(p[iter+1]-p[iter]))
    if abs((p[iter+1]-p[iter])/p[iter])<=tol:
        iter = iter+1
        break
    iter = iter+1

def find_order(epsilon):
    order = []
    for i in range(1,len(epsilon)-1):
        order.append(math.log(epsilon[i+1]/epsilon[i])/math.log(epsilon[i]/
 ↪epsilon[i-1]))
    return order
order = find_order(epsilon)

print("\n\nPart (b)\n")
print("No. of Iterations: ", iter)
```

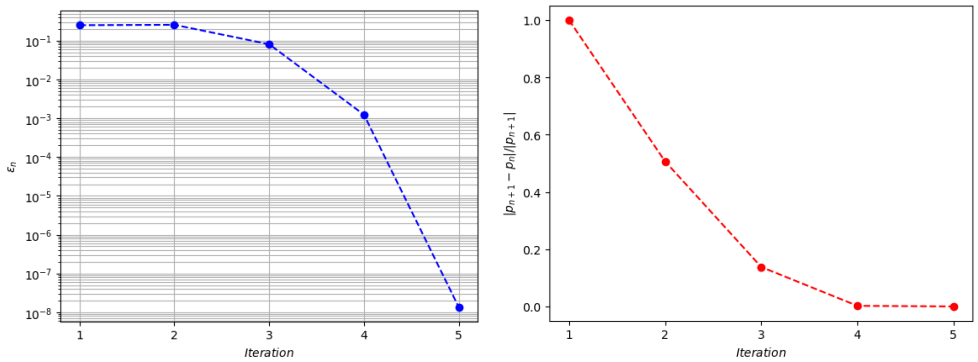
```
print("Relative Tolerance: ", abs((p[iter]-p[iter-1])/p[iter-1]))
print("Final Value of p: ", p[iter])
print("Final Value of f(p): ", f2(x).subs(x,p[iter]))
print("Order of Convergence: ", order[len(order)-1])
print("\n")

plt.plot(range(1, len(epsilon) + 1), epsilon, 'bo--')
plt.grid(True, which="both", ls="-")
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.yscale("log")
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_n$")
plt.show()

relerr = []
for i in range(len(epsilon)):
    relerr.append(epsilon[i]/abs(p[i+1]))
plt.plot(range(1, len(relerr) + 1), relerr,'ro--')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$|p_{n+1}-p_n|/|p_{n+1}|$")
plt.show()
```

```
Part (a)

No. of Iterations:  5
Relative Tolerance:  2.28753315742628e-8
Final Value of p:  -0.588401776500996
Final Value of f(p):  -1.11022302462516e-16
Order of Convergence:  2.7322000706575094
```




```
Part (b)

No. of Iterations:  5
Relative Tolerance:  2.28753315742628e-8
Final Value of p:  -0.588401776500996
Final Value of f(p):  1.23259516440783e-32
```

```
Order of Convergence:  2.7322000706575085
```



1.5 Ques 3

[5]:
```
##########################################################################
import sympy as sym
import math
from IPython.display import display, Math, Markdown
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

theta2, theta3 = sym.symbols('theta2 theta3')

##########################################################################
def f1(theta2, theta3):
    return 6*sym.cos(theta2) + 8*sym.cos(theta3) + 4*math.cos(220*(math.pi/180)) - 10
def f1theta2(theta2, theta3):
    return sym.diff(f1(theta2,theta3),theta2)
def f1theta3(theta2, theta3):
    return sym.diff(f1(theta2,theta3),theta3)

##########################################################################
def f2(theta2, theta3):
    return 6*sym.sin(theta2) + 8*sym.sin(theta3) + 4*math.sin(220*(math.pi/180))
def f2theta2(theta2, theta3):
    return sym.diff(f2(theta2,theta3),theta2)
def f2theta3(theta2, theta3):
    return sym.diff(f2(theta2,theta3),theta3)

##########################################################################
def j(theta2, theta3):
    return sym.Matrix([[f1theta2(theta2, theta3), f1theta3(theta2,␣
 ↪theta3)],[f2theta2(theta2, theta3), f2theta3(theta2, theta3)]])

##########################################################################
def f(theta2, theta3):
    return sym.Matrix([[f1(theta2, theta3)],[f2(theta2, theta3)]])

def jinv(theta2, theta3):
```

```
    return sym.simplify(j(theta2, theta3).inv())
```

1.6 Newton Method for System of Equations

[6]:
```
def rad(theta):
    return theta*(math.pi/180)

def deg(theta):
    val = theta*(180/math.pi)
    if val>=0:
        return val
    else:
        return 360+val

def maxval(theta, i):
    val1 = abs(theta[i+1][0] - theta[i][0])
    val2 = abs(theta[i+1][1] - theta[i][1])
    return [val1, val2]

def maxrel(theta, i):
    val1 = abs((theta[i+1][0] - theta[i][0])/theta[i+1][0])
    val2 = abs((theta[i+1][1] - theta[i][1])/theta[i+1][1])
    return [val1, val2]

theta = [[rad(30), rad(0)]]
tol = 1e-4
iter = 0
epsilon = []
relative = []
while True:
    p_new = ((jinv(theta2,theta3).
 ↪subs([(theta2,theta[iter][0]),(theta3,theta[iter][1])]))*(f(theta2,theta3).
 ↪subs([(theta2,theta[iter][0]),(theta3,theta[iter][1])]))).T
    p_new_flat = [item for sublist in p_new.tolist() for item in sublist]
    theta_iter_flat = theta[iter]
    theta_new = [theta_iter_flat[i] - p_new_flat[i] for i in range(len(p_new_flat))]
    theta.append(theta_new)
    abserrval = maxval(theta,iter)
    relerrval = maxrel(theta,iter)
    epsilon.append(abserrval)
    relative.append(relerrval)
    if max(relerrval)<=tol:
        iter = iter+1
        break
    iter = iter+1

print("\n")
print("No. of Iterations: ", iter)
print("Max. Relative Tolerance: ", max(maxrel(theta,iter-1)))
print("Final Value of p (in radian): ", theta[iter])
print("Final Value of p (in degrees): ", [deg(theta[iter][i]) for i in range(2)])
print("Final Value of f(p): ", f(theta2,theta3).
 ↪subs([(theta2,theta[iter][0]),(theta3,theta[iter][1])]).T.tolist()[0])
```

```
print("\n")

theta2err = [epsilon[i][0] for i in range(iter)]
theta3err = [epsilon[i][1] for i in range(iter)]

plt.plot(range(1, len(theta2err) + 1), theta2err,'ro--', label =␣
 ↪r"$\epsilon_{\theta_2}$")
plt.plot(range(1, len(theta3err) + 1), theta3err,'bo--', label =␣
 ↪r"$\epsilon_{\theta_3}$")
plt.legend()
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$Iteration$")
plt.ylabel(r"$\epsilon_{n}$")
plt.show()

theta2rel = [relative[i][0] for i in range(iter)]
theta3rel = [relative[i][1] for i in range(iter)]

plt.plot(range(1, len(theta2rel) + 1), theta2rel,'ro--', label =␣
 ↪r"$relative~~\epsilon_{\theta_2}$")
plt.plot(range(1, len(theta3rel) + 1), theta3rel,'bo--', label =␣
 ↪r"$relative~~\epsilon_{\theta_3}$")
plt.legend()
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.xlabel(r"$n$")
plt.ylabel(r"$relative~~\epsilon_{n}$")
plt.show()
```

```
No. of Iterations:  4
Max. Relative Tolerance:  1.11372373822926e-7
Final Value of p (in radian):  [0.558770307890069, -0.0762881217804664]
Final Value of p (in degrees):  [32.0151803593265, 355.629012594999]
Final Value of f(p):  [0, -4.44089209850063e-16]
```