In [1]:

```
#Numerical Exercise 4
#Md Shariar Imroze Khan
#Matriculation Number: 220202354

import numpy as np
import math
from sympy import *
import random
```

In [2]:

```
########################### (a): Criteria 1 ###########################
def newton(f, df, x0, ea, er, N=50):

    def g(x):
        return x - f(x)/df(x)

    def g0(x_0):
        return x0-f(x0)/df(x0)

    approx = [x0]

    n = 0
    ea = 10**-3
    er = 10**-10

    while n < 100:
        # Calculate p[n+1]
        approx.append(g(approx[n]))

        # Check for convergence

        if abs(approx[n]) < ea:                              #Criteria 1
        #if abs(approx[n]) < er*abs(approx[0])+ea:            #Criteria 2
        #if abs(approx[n+1] - approx[n]) < ea :               #Criteria 3
        #if abs(approx[n+1]-approx[n]) < er*abs(approx[0])+ea: #Criteria 4
            return approx
        n = n + 1


def f(x):
    return exp(-x)-10**(-9)


def df(x):
    return -exp(-x)

# Start Newton's method
approx = newton(f,df,0,10**-3,10**-10)

for i in range(0,len(approx)):
    if approx[i] > 0:
        print(f"{i}  {approx[i]}")
    else:
        print(f"{i} {approx[i]}")
```

```
0 0
1  0.999999999000000
```

In [3]:

```python
######################## (a): Criteria 2 ################################
def newton(f, df, x0, ea, er, N=50):

    def g(x):
        return x - f(x)/df(x)

    def g0(x_0):
        return x0-f(x0)/df(x0)

    approx = [x0]

    n = 0
    ea = 10**-3
    er = 10**-10

    while n < 100:
        # Calculate p[n+1]
        approx.append(g(approx[n]))

        # Check for convergence

        #if abs(approx[n]) < ea:                              #Criteria 1
        if abs(approx[n]) < er*abs(approx[0])+ea:            #Criteria 2
        #if abs(approx[n+1] - approx[n]) < ea :               #Criteria 3
        #if abs(approx[n+1]-approx[n]) < er*abs(approx[0])+ea: #Criteria 4
            return approx
        n = n + 1


def f(x):
    return exp(-x)-10**(-9)


def df(x):
    return -exp(-x)

# Start Newton's method
approx = newton(f,df,0,10**-3,10**-10)

for i in range(0,len(approx)):
    if approx[i] > 0:
        print(f"{i}  {approx[i]}")
    else:
        print(f"{i} {approx[i]}")
```

```
0 0
1  0.999999999000000
```

In [4]:

```python
############################ (a): Criteria 3 #####################
def newton(f, df, x0, ea, er, N=50):

    def g(x):
        return x - f(x)/df(x)

    def g0(x_0):
        return x0-f(x0)/df(x0)

    approx = [x0]

    n = 0
    ea = 10**-3
    er = 10**-10

    while n < 100:
        # Calculate p[n+1]
        approx.append(g(approx[n]))

        # Check for convergence

        #if abs(approx[n]) < ea:                             #Criteria 1
        #if abs(approx[n]) < er*abs(approx[0])+ea:           #Criteria 2
        if abs(approx[n+1] - approx[n]) < ea :              #Criteria 3
        #if abs(approx[n+1]-approx[n]) < er*abs(approx[0])+ea: #Criteria 4
            return approx
        n = n + 1


def f(x):
    return exp(-x)-10**(-9)


def df(x):
    return -exp(-x)

# Start Newton's method
approx = newton(f,df,0,10**-3,10**-10)

for i in range(0,len(approx)):
    if approx[i] > 0:
        print(f"{i}  {approx[i]}")
    else:
        print(f"{i} {approx[i]}")
```

```
0 0
1  0.999999999000000
2  1.99999999628172
3  2.99999998889266
4  3.99999996880713
5  4.99999991420898
6  5.99999976579583
7  6.99999936236713
8  7.99999826573467
9  8.99999528478185
10  9.99998718173613
11  10.9999651555527
12  11.9999052834972
```

```
13  12.9997425441206
14  13.9993002446159
15  14.9980984815662
16  15.9948356743842
17  16.9859953363382
18  17.9621763078303
19  18.8989534601427
20  19.7376249208985
21  20.3644249604722
22  20.6659394724895
23  20.7216536347528
24  20.7232645380466
25  20.7232658369456
```

In [5]:

```python
################################### (a): Criteria 4 ##################
def newton(f, df, x0, ea, er, N=50):

    def g(x):
        return x - f(x)/df(x)

    def g0(x_0):
        return x0-f(x0)/df(x0)

    approx = [x0]

    n = 0
    ea = 10**-3
    er = 10**-10

    while n < 100:
        # Calculate p[n+1]
        approx.append(g(approx[n]))

        # Check for convergence

        #if abs(approx[n]) < ea:                          #Criteria 1
        #if abs(approx[n]) < er*abs(approx[0])+ea:        #Criteria 2
        #if abs(approx[n+1] - approx[n]) < ea :           #Criteria 3
        if abs(approx[n+1]-approx[n]) < er*abs(approx[0])+ea: #Criteria 4
            return approx
        n = n + 1


def f(x):
    return exp(-x)-10**(-9)


def df(x):
    return -exp(-x)

# Start Newton's method
approx = newton(f,df,0,10**-3,10**-10)

for i in range(0,len(approx)):
    if approx[i] > 0:
        print(f"{i}  {approx[i]}")
    else:
        print(f"{i} {approx[i]}")
```

```
0 0
1  0.999999999000000
2  1.99999999628172
3  2.99999998889266
4  3.99999996880713
5  4.99999991420898
6  5.99999976579583
7  6.99999936236713
8  7.99999826573467
9  8.99999528478185
10  9.99998718173613
11  10.9999651555527
12  11.9999052834972
```

```
13  12.9997425441206
14  13.9993002446159
15  14.9980984815662
16  15.9948356743842
17  16.9859953363382
18  17.9621763078303
19  18.8989534601427
20  19.7376249208985
21  20.3644249604722
22  20.6659394724895
23  20.7216536347528
24  20.7232645380466
25  20.7232658369456
```

In [6]:

```python
############################# (b) #################################
def f(X):
    x, y = X
    return (x-1)**4 + 2*(x-1)**2*(y-1)**2+(y+1)**4-2*(y-1)**2-(2*y+1)**2+1

def f_grad(X):
    x, y = X
    return [4*(x-1)*((x-1)**2+(y-1)**2),
            4*((x**2)*(y-1)-2*x*(y-1)+y*(y**2+3*y+1))]

def f_hess(X):
    x, y = X
    return [[12*(x-1)**2+4*(y-1)**2, 8*(x-1)*(y-1)],
        [8*(x-1)*(y-1), 4*(x-1)**2+12*(y+1)**2-12]]
```

In [7]:

```python
def newton_ST1(x_init, epsilon=1e-4, er=10**-10, max_iterations=100): #with stopping cr
    x = x_init
    prev_x = np.zeros(2)
    for i in range(max_iterations):
        x = x - np.linalg.solve(np.array(f_hess(x)), np.array(f_grad(x)))

        #Criteria 1
        if np.linalg.norm((f_grad(prev_x),f_grad(x)),ord=np.inf) < epsilon:
            return x, i + 1
        prev_x=x
        print(x,'-iterations: ', i+1)
    return x, max_iterations
```

In [8]:

```
def newton_ST2(x_init, epsilon=1e-4, er=10**-10, max_iterations=100): # with stopping c
    x = x_init
    prev_x = np.zeros(2)
    for i in range(max_iterations):
        x = x - np.linalg.solve(np.array(f_hess(x)), np.array(f_grad(x)))

        #Criteria 2
        if (np.linalg.norm((f_grad(prev_x),f_grad(x)),ord=np.inf)) < er*np.linalg.norm(
            return x, i + 1
        prev_x=x
        print(x,'-iterations: ', i+1)
    return x, max_iterations
```

In [9]:

```
x_init = np.array([1.21,-1.15])
```

In [10]:

```
g_x0 = f_grad(x_init)
h_x0 = f_hess(x_init)
```

In [11]:

```python
x_min, it = newton_ST1(x_init)
print('Extremum at, x* =', x_min)
print('Value of Extremum, f(x*) =', f(x_min))
print('Iterations =', it-1)
```

```
[1.22347794 0.0062247 ] -iterations:  1
[ 5.74413364 12.10449565] -iterations:  2
[4.53149904 7.73513692] -iterations:  3
[3.72503909 4.84502883] -iterations:  4
[3.15949736 2.95707653] -iterations:  5
[2.69901629 1.75793293] -iterations:  6
[2.26836071 1.01879421] -iterations:  7
[1.85023558 0.54058313] -iterations:  8
[1.37603724 0.11254893] -iterations:  9
[-0.16325096 -2.15878114] -iterations:  10
[-1.91947523  2.98941097] -iterations:  11
[-1.14706992  1.81596466] -iterations:  12
[-0.54191668  1.086685  ] -iterations:  13
[-0.04461626  0.61249457] -iterations:  14
[0.42701068 0.21970153] -iterations:  15
[ 1.2862043  -0.71118192] -iterations:  16
[1.25202334 0.05758623] -iterations:  17
[-1.18666097 -4.97534666] -iterations:  18
[ 0.13794453 -4.12450733] -iterations:  19
[ 0.76711646 -3.51856926] -iterations:  20
[ 0.96540618 -3.19229074] -iterations:  21
[ 0.99861546 -3.10866569] -iterations:  22
[ 0.99999673 -3.10381862] -iterations:  23
[ 1.         -3.1038034] -iterations:  24
Extremum at, x* = [ 1.         -3.1038034]
Value of Extremum, f(x*) = -40.21219689670468
Iterations = 24
```

In [12]:

```python
x_min, it = newton_ST2(x_init)
print('Extremum at, x* =', x_min)
print('Value of Extremum, f(x*) =', f(x_min))
print('Iterations =', it-1)
```

```
[1.22347794 0.0062247 ] -iterations:  1
[ 5.74413364 12.10449565] -iterations:  2
[4.53149904 7.73513692] -iterations:  3
[3.72503909 4.84502883] -iterations:  4
[3.15949736 2.95707653] -iterations:  5
[2.69901629 1.75793293] -iterations:  6
[2.26836071 1.01879421] -iterations:  7
[1.85023558 0.54058313] -iterations:  8
[1.37603724 0.11254893] -iterations:  9
[-0.16325096 -2.15878114] -iterations:  10
[-1.91947523  2.98941097] -iterations:  11
[-1.14706992  1.81596466] -iterations:  12
[-0.54191668  1.086685  ] -iterations:  13
[-0.04461626  0.61249457] -iterations:  14
[0.42701068 0.21970153] -iterations:  15
[ 1.2862043  -0.71118192] -iterations:  16
[1.25202334 0.05758623] -iterations:  17
[-1.18666097 -4.97534666] -iterations:  18
[ 0.13794453 -4.12450733] -iterations:  19
[ 0.76711646 -3.51856926] -iterations:  20
[ 0.96540618 -3.19229074] -iterations:  21
[ 0.99861546 -3.10866569] -iterations:  22
[ 0.99999673 -3.10381862] -iterations:  23
[ 1.         -3.1038034] -iterations:  24
Extremum at, x* = [ 1.         -3.1038034]
Value of Extremum, f(x*) = -40.21219689670468
Iterations = 24
```

In [ ]:

```python

```