```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in ./env/lib/python3.8/site-packages (3.6.3)
Requirement already satisfied: numpy>=1.19 in ./env/lib/python3.8/site-packages (from matplotlib) (1.24.0)
Requirement already satisfied: contourpy>=1.0.1 in ./env/lib/python3.8/site-packages (from matplotlib) (1.0.7)
Requirement already satisfied: python-dateutil>=2.7 in ./env/lib/python3.8/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in ./env/lib/python3.8/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in ./env/lib/python3.8/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: pillow>=6.2.0 in ./env/lib/python3.8/site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: fonttools>=4.22.0 in ./env/lib/python3.8/site-packages (from matplotlib) (4.38.0)
Requirement already satisfied: packaging>=20.0 in ./env/lib/python3.8/site-packages (from matplotlib) (22.0)
Requirement already satisfied: kiwisolver>=1.0.1 in ./env/lib/python3.8/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: six>=1.5 in ./env/lib/python3.8/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

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

情報計算科学の基礎

レポート4

(1) $ E(x) = 3(x-2)^4 + (x-2)^2 + 1 $を最小化するxをニュートン法で求める。

$ E'(x)/2 = 6(x-2)^3 + (x-2) $ なので、 $ E'(x)/2 = 0 $　となるx を求める。

```
EPSILON = 1e-8 # 0.0000001

def E_OBJ_FUNC(x) -> float:
    return 6*(x-2)**3 + (x-2)

def E_D_FUNC(x) -> float :
    return 18*(x - 2)**2 + 1

def calc_slice(liner_func):
    pass

def solve_eq_1(objfunc,dfunc,dimension:int = 1) -> float :
    x = 20000*( random.random() - 0.5 )
    print(f'x:init:{x}')
    err = objfunc(x)
    while abs(err) > EPSILON:
        x_formar = x
        x = x_formar - (objfunc(x_formar) / dfunc(x_formar))
        err = objfunc(x)

    print(f'err:{err}')
    print(f'x:{x}')

    return x

print(solve_eq_1(E_OBJ_FUNC,E_D_FUNC))
```

```
x:init:92.88988434797676
err:0.0
x:2.0
2.0
```

```
print(solve_eq_1(E_OBJ_FUNC,E_D_FUNC))
```

```
x:init:-8002.999486444318
err:0.0
x:2.0
2.0
```

かなり早いスピードで $ x=2 $ に収束した。また、初期値がかなり大きくても早いスピードで収束した。

(2)

```
def F_OBJ_FUNC(x) -> list:
    ans = [0,0]
    ans[0] = (6*(x[0]-3) + 6*(x[0]-3)*(x[1]-2)**2)
    ans[1] = (6*(x[1]-2)*(x[0]-3)**2 + 8*(x[1]-2))
    return ans

def F_D_FUNC(x) -> list :
    ans =[[0,0],[0,0]]
    ans[0][0] = 6*((x[1] - 2)**2 + 1)
    ans[0][1] = 12*(x[0] - 3)*(x[1] - 2)
    ans[1][0] = ans[0][1]
    ans[1][1] = 6*(x[0] - 3)**2 + 8
    return ans


EPSILON = 1e-8 # 0.0000001
# EPSILON = 0.1 # 0.1

def inv_mat(A):
    n_size_of_A = len(A)
```

```
    X = np.eye(n_size_of_A)
    #まずAを上三角行列にする。
    A = A.astype('float64')
    for p in range(n_size_of_A):
        pivot = A[p][p]
        for j in range(p+1, n_size_of_A):
            coef = A[j][p] / pivot
            A[j] -= A[p] * coef
            X[j] -= X[p] * coef
    #途中経過確認。
    #print(A)
    #print(X)
    #対角成分を1にする。
    for i in range(n_size_of_A):
        X[i] /= A[i][i]
        A[i] /= A[i][i]
    #途中経過確認。
    #print(A)
    #print(X)
    #答えを出す。
    for i in range(n_size_of_A-1,0,-1):
        for j in range(i):
            X[j] -= X[i] * A[j][i]
            A[j][i] = 0
    return X

def solve_eq_2(objfunc,dfunc,dimension:int = 2) -> float :
    x = 20000*(np.random.rand(dimension) - np.array(([0 for i in range(dimension)])) )
    print(f'x:init:{x}')
    err = objfunc(x)
    while np.linalg.norm(np.array(err)) > EPSILON:
        x_formar = x
        inv = inv_mat(np.array( (dfunc(x_formar)) ))
        x = x_formar - (inv @   np.array( objfunc(x_formar) ) )
        err = objfunc(x)
        # print(f'err:{err}')
        # print(f'x:{x}')

    print(f'err:{err}')
    print(f'x:{x}')

    return x

print(solve_eq_2(F_OBJ_FUNC,F_D_FUNC))
```

```
x:init:[19212.23383629 17477.89416071]
err:[5.329070518200751e-15, 2.842170943040401e-14]
x:[3. 2.]
[3. 2.]
```

なので、解は $ (x_1,x_2) = (3,2) $ これもかなり早いスピードで収束した。