HW2. Root finding

2022313047 Boyeon,Kim

In [ ]:
```python
import numpy as np
import matplotlib as plt
```

Consider a non-linear equation system, F(x) = b,

$$F(x) = \begin{bmatrix} x^2 + xyz + y^2z^3 \\ xy^2 - yz^2 - 2x^2 \\ x^2y + y^2z + z^4 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 0 \\ 4 \end{bmatrix}$$

1. Using Newton's method, find a root of the system $F(x) = b$ when an initial guess is $(x0, y0, z0) = (1, 2, 3)$ or $(-1, 1, 1)$ (with line-by-line code and using the NumPy library).

Let $f(x) : F(x) - b = 0$

$$f(x) = \begin{bmatrix} x^2 + xyz + y^2z^3 - 3 \\ xy^2 - yz^2 - 2x^2 \\ x^2y + y^2z + z^4 - 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

In [ ]:
```python
def f(X):
    x = X[0]
    y = X[1]
    z = X[2]
    F1 = (x**2)+ x*y*z + (y**2)*(z**3) -3
    F2 = x*(y**2) - y*(z**2) - 2*(x**2)
    F3 = (x**2)*y + (y**2)*z + z**4 - 4
    return np.array([F1, F2, F3])
```

Let $X = [x, y, z], J : \frac{\partial F}{\partial X}$

$$J(X) = \begin{bmatrix} 2x + yz & xz + 2yz^3 & xy + 3y^2z^2 \\ y^2 - 4x & 2xy - z^2 & -2yz \\ 2xy & x^2 + 2yz & y^2 + 4z^3 \end{bmatrix}$$

In [ ]:
```python
def Jf(X):
    x = X[0]
    y = X[1]
    z = X[2]
    J11 = 2*x + y*z
    J12 = x*z + 2*y*(z**3)
    J13 = x*y + 3*(y**2)*(z**2)
    J21 = (y**2) - 4*x
    J22 = 2*x*y - (z**2)
    J23 = -2*y*z
    J31 = 2*x*y
    J32 = (x**2) + 2*y*z
    J33 = (y**2) + 4*(z**3)
    return np.array([[J11, J12, J13],[J21, J22, J23],[J31, J32, J33]])
```

In [ ]:
```python
def newton(f, Jf, init, tol):
    """ Newton method for sysytems
        f : fucntion
        Jf : Jacobian of F
        init : initial
        tol : stop criterion """
    Xold = init
    # for printing
    err = np.NaN
    for iter in range (0,100):
        print('iter = %02d  X = (%0.6f, %0.6f, %0.6f) error = %0.6f ' %(iter, Xold[0], Xold[1], Xold[2], err))
        J = Jf(Xold)
        F = f(Xold)

        H = np.dot(np.linalg.inv(J),F)

        Xnew = Xold - H
        # error : Xnew - Xold = -H
        err = np.linalg.norm(- H, np.inf)

        if err < tol:
            break

        if f'{Xnew[0]}' == f'{np.NaN}':
            print('======================================')
            print('This initial can not find the solution')
            break
        Xold = Xnew
    print('The solution is X = (%0.6f, %0.6f, %0.6f)' %(Xold[0], Xold[1], Xold[2]))
    return Xold
```

In [ ]:
```python
# Set the parameters
X1 = [1, 2, 3]
X2 = [-1, 1, 1]
tol = 1e-6
```

In [ ]:
```python
print('\n Case1) initial = (1, 2, 3)')
sol1_new = newton(f, Jf, X1, tol)

print('\n Case2) initial = (-1, 1, 1)')
sol2_new = newton(f, Jf, X2, tol)
```

```
 Case1) initial = (1, 2, 3)
iter = 00  X = (1.000000, 2.000000, 3.000000) error = nan
iter = 01  X = (10.374528, 1.381132, 1.924528) error = 9.374528
iter = 02  X = (5.475135, 1.444590, 0.977572) error = 4.899394
iter = 03  X = (3.002071, 1.424665, 0.551072) error = 2.473064
iter = 04  X = (1.815242, 1.464958, 0.430732) error = 1.186829
iter = 05  X = (1.339556, 1.587389, 0.506166) error = 0.475686
iter = 06  X = (1.196295, 1.662526, 0.560200) error = 0.143261
iter = 07  X = (1.175165, 1.674642, 0.565389) error = 0.021130
iter = 08  X = (1.174659, 1.674928, 0.565552) error = 0.000506
The solution is X = (1.174659, 1.674928, 0.565552)

 Case2) initial = (-1, 1, 1)
iter = 00  X = (-1.000000, 1.000000, 1.000000) error = nan
iter = 01  X = (-2.666667, -5.333333, 4.333333) error = 6.333333
iter = 02  X = (-1.474977, -5.034424, 3.044520) error = 1.288814
iter = 03  X = (0.030803, -6.225164, 1.565757) error = 1.505780
iter = 04  X = (-0.171280, -2.890846, 1.607110) error = 3.334317
iter = 05  X = (-1.454389, 3.099239, 3.239177) error = 5.990085
iter = 06  X = (0.281805, 2.717992, 2.417169) error = 1.736194
iter = 07  X = (0.861428, 2.573774, 1.678405) error = 0.738765
iter = 08  X = (-0.139344, 2.985221, 1.036732) error = 1.000772
iter = 09  X = (0.121502, 2.089010, 0.986590) error = 0.896211
iter = 10  X = (0.369438, 2.004077, 0.857619) error = 0.247935
iter = 11  X = (0.427181, 1.979512, 0.818367) error = 0.057743
iter = 12  X = (0.430244, 1.978996, 0.815108) error = 0.003259
iter = 13  X = (0.430244, 1.979000, 0.815094) error = 0.000013
The solution is X = (0.430244, 1.979000, 0.815094)
```

2. Explain the behavior of Newton's method when an initial guess is $(x0, y0, z0) = (0, 0, 1)$.

In [ ]:
```python
X3 = [0, 0, 1]

print('\n initial = (0, 0, 1)')
newton(f, Jf, X3, tol)
```

```
 initial = (0, 0, 1)
iter = 00  X = (0.000000, 0.000000, 1.000000) error = nan
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In [7], line 4
      1 X3 = [0, 0, 1]
      3 print('\n initial = (0, 0, 1)')
----> 4 newton(f, Jf, X3, tol)

Cell In [4], line 15, in newton(f, Jf, init, tol)
     12 J = Jf(Xold)
     13 F = f(Xold)
---> 15 H = np.dot(np.linalg.inv(J),F)
     17 Xnew = Xold - H
     18 # error : Xnew - Xold = -H

File <__array_function__ internals>:180, in inv(*args, **kwargs)

File /opt/anaconda3/envs/epi/lib/python3.10/site-packages/numpy/linalg/linalg.py:552, in inv(a)
    550 signature = 'D->D' if isComplexType(t) else 'd->d'
    551 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 552 ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    553 return wrap(ainv.astype(result_t, copy=False))

File /opt/anaconda3/envs/epi/lib/python3.10/site-packages/numpy/linalg/linalg.py:89, in _raise_linalgerror_singular(err, flag)
     88 def _raise_linalgerror_singular(err, flag):
---> 89     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

Discussion

When the initial guess is (0, 0, 1), the Jacobian matrix of f is singular.
Hence we don't have inverse matrix, so that we don't have solution by using Newton's method.

3. Using broyden2 in the SciPy library, find a root of the system $F(x) = b$ when an initial guess is $(x0, y0, z0) = (1, 2, 3)$ or $(-1, 1, 1)$, and compare the result obtained in Question 1.

In [ ]:
```python
import scipy as sp
```

In [ ]:
```python
sol1_brd = sp.optimize.broyden2(f,[1, 2, 3])
sol2_brd = sp.optimize.broyden2(f,[-1, 1, 1], f_tol=1e-6, x_tol=1e-6)

print('\n Case1) initial = (1, 2, 3)')
print(' The solution of X:')
print('Newton : X = (%0.6f, %0.6f, %0.6f)' %(sol1_new[0], sol1_new[1], sol1_new[2]))
print('Broyden : X = (%0.6f, %0.6f, %0.6f)' %(sol1_brd[0], sol1_brd[1], sol1_brd[2]))

print('\n Case2) initial = (-1, 1, 1)')
print(' The solution of X:')
print('Newton : X = (%0.6f, %0.6f, %0.6f)' %(sol2_new[0], sol2_new[1], sol2_new[2]))
print('Broyden : X = (%0.6f, %0.6f, %0.6f)' %(sol2_brd[0], sol2_brd[1], sol2_brd[2]))
```

```
 Case1) initial = (1, 2, 3)
 The solution of X:
Newton : X = (1.174659, 1.674928, 0.565552)
Broyden : X = (1.174658, 1.674929, 0.565552)

 Case2) initial = (-1, 1, 1)
 The solution of X:
Newton : X = (0.430244, 1.979000, 0.815094)
Broyden : X = (-0.722942, -0.807187, 1.370593)
```