[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Python Programming for Numerical Analysis		HW number: #5

5.1. 다항식 $x^4 + 2x^3 - 7x^2 + 3 = 0$의 모든 양의 실근을 구하여라. Companion Matrix를 활용한다.

a. 문제

Problem 5.1

Compute all positive real roots of

$$x^4 + 2x^3 - 7x^2 + 3 = 0\,.$$

You might consider a companion matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 0 & 7 & -2 \end{bmatrix}$$

and the eigenvalues of A are indeed the roots of the characteristic polynomial equation

$$x^4 + 2x^3 - 7x^2 + 3 = 0\,.$$

b. 풀이

sol) $A - \lambda I = \begin{bmatrix} -\lambda & 1 & 0 & 0 \\ 0 & -\lambda & 1 & 0 \\ 0 & 0 & -\lambda & 1 \\ -3 & 0 & 7 & -\lambda-2 \end{bmatrix} = \overset{1}{\cancel{(-1)^{1+1}}}(-\lambda)\underbrace{\begin{vmatrix} -\lambda & 1 & 0 \\ 0 & -\lambda & 1 \\ 0 & 7 & -\lambda-2 \end{vmatrix}}_{-\lambda \underbrace{\begin{vmatrix} -\lambda & 1 \\ 7 & -\lambda-2 \end{vmatrix}}_{\lambda(\lambda+2)-7}} + \overset{-1}{\cancel{(-1)^{1+2}}}\underbrace{\begin{vmatrix} 0 & 1 & 0 \\ 0 & -\lambda & 1 \\ -3 & 7 & -(\lambda+2) \end{vmatrix}}_{-\underbrace{\begin{vmatrix} 0 & 1 \\ -3 & -\lambda-2 \end{vmatrix}}_{+3}}$

$= \lambda^2(\lambda^2 + 2\lambda - 7) - 3 = \lambda^4 + 2\lambda^3 - 7\lambda^2 + 3 = 0$

∴ 즉, Companion Matrix 의 characteristic equation은 원래의 다항식과 일치한다.

답 : $x = \dfrac{-3+\sqrt{21}}{2}$ or $x = \dfrac{1+\sqrt{5}}{2}$

(이 값은 계산기를 통해 구한 값으로, Python code 의 Newton Raphson 알고리즘은 통해 구한 값을 포함한다.)

c. Python Code

```python
import numpy as np

# f(x) = x^4 + 2x^3 - 7x^2 + 3
def f(x):
    return x**4 + 2*x**3 - 7*x**2 + 3

# f'(x)
def df(x):
    return 4*x**3 + 6*x**2 - 14*x

# Newton-Raphson
def newton_raphson(f, df, x0, tol=1e-10, max_iter=100):
    x = x0
    for i in range(max_iter):
        fx = f(x)
        dfx = df(x)
        if abs(dfx) < 1e-12:
            print("Derivative is too small.")
            break
        dx = -fx / dfx
        x += dx
        print(f"Iter {i+1}: x = {x:.10f}, f(x) = {f(x):.3e}")
        if abs(fx) < tol:
            break
    return x

# initial guess
root = newton_raphson(f, df, x0=0.8)
print(f"\nfinal root: {root}")
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
Iter 1: x = 0.7912650602, f(x) = 1.217e-04
Iter 2: x = 0.7912878473, f(x) = 7.811e-10
Iter 3: x = 0.7912878475, f(x) = -4.441e-16
Iter 4: x = 0.7912878475, f(x) = 8.882e-16

final root: 0.7912878474779199
```

5.2. (Least Square Method 적용 1) 질량-연비 관계 추정

a. 문제

Problem 5.2

The speed v of a Saturn V rocket in vertical flight near the surface of earth can be approximated by

$$v = u \ln \frac{M_0}{M_0 - \dot{m}t} - gt$$

where

$$\begin{aligned} u &= 2510 \text{ m/s} = \text{ velocity of exhaust relative to the rocket} \\ M_0 &= 2.8 \times 10^6 \text{ kg} = \text{ mass of rocket at liftoff} \\ \dot{m} &= 13.3 \times 10^3 \text{ kg/s} = \text{ rate of fuel consumption} \\ g &= 9.81 \text{ m/s}^2 = \text{ gravitational acceleration} \\ t &= \text{ time measured from liftoff} \end{aligned}$$

(a) Determine the time when the rocket reaches the speed of sound 335 m/s.

b. 풀이

sol) $335 = u \ln \frac{M_0}{M_0 - \dot{m}t} - gt$ $\Rightarrow$ Newton-Raphson 방법으로 근사하자.

$\therefore f(t) = u \ln \frac{M_0}{M_0 - \dot{m}t} - gt - 335$ $\Rightarrow$ $f'(t) = \frac{u \cdot \dot{m}}{M_0 - \dot{m}t} - g$

(Newton-Raphson 과정은 Python code 및 결과 참조할 것!)

$\boxed{\text{답: } 70.878 \text{ (초)}}$

c. Python Code

```python
import numpy as np

# constants
u = 2510          # m/s
M0 = 2.8e6        # kg
mdot = 13.3e3     # kg/s
g = 9.81          # m/s^2
v_target = 335    # m/s

# f(t)
def f(t):
    return u * np.log(M0 / (M0 - mdot * t)) - g * t - v_target

# f'(t)
def df(t):
    return (u * mdot) / (M0 - mdot * t) - g

# Newton-Raphson method
def newton(f, df, x0, tol=1e-10, max_iter=100):
    x = x0
    for i in range(max_iter):
        fx = f(x)
        dfx = df(x)
        dx = -fx / dfx
        x += dx
        print(f"Iter {i+1}: t = {x:.10f}, f(t) = {fx:.3e}")
        if abs(fx) < tol:
            break
    return x

# initial guess
t0 = 5
t_solution = newton(f, df, t0)

print(f"\nroot: {t_solution:.6f}(s)")
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
Iter 1: t = 139.7395296913, f(t) = -3.237e+02
Iter 2: t = 99.5852689210, f(t) = 1.030e+03
Iter 3: t = 76.4863630169, f(t) = 2.960e+02
Iter 4: t = 71.1177369255, f(t) = 4.787e+01
Iter 5: t = 70.8784247571, f(t) = 1.961e+00
Iter 6: t = 70.8779722697, f(t) = 3.694e-03
Iter 7: t = 70.8779722681, f(t) = 1.318e-08
Iter 8: t = 70.8779722681, f(t) = 0.000e+00

root: 70.877972(s)
```

5.3. **(Least Square Method 적용 2) Linear vs Quadratic**

a. 문제

Problem 5.3

The equations

$$\sin x + 3\cos x - 2 = 0$$
$$\cos x - \sin y + 0.2 = 0$$

have a solution in the vicinity of the point $(1, 1)$.

(a) Use the Newton-Raphson method to refine the solution.

b. 풀이

sol) Multivariable Newton-Raphson Method를 사용하자

$$\therefore x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)}) \cdot f(x^{(k)})$$

- $x = \begin{bmatrix} x \\ y \end{bmatrix}$
- $f = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} = \begin{bmatrix} \sin x + 3\cos x - 2 \\ \cos x - \sin y + 0.2 \end{bmatrix}$
- $J = \dfrac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \cos x - 3\sin x & 0 \\ -\sin x & -\cos x \end{bmatrix}$

(연산 과정은 Python code 참고할 것!)

$\boxed{\text{답} : x = 1.208,\ y = 0.588}$

c. Python Code

```python
import numpy as np

# f(x)
def f(xy):
    x, y = xy
    return np.array([
        np.sin(x) + 3 * np.cos(x) - 2,
        np.cos(x) - np.sin(y) + 0.2
    ])

# Jacobian
def J(xy):
    x, y = xy
    return np.array([
        [np.cos(x) - 3 * np.sin(x), 0],
        [-np.sin(x), -np.cos(y)]
    ])

# Newton-Raphson for multivariable system
def newton_2d(f, J, x0, tol=1e-10, max_iter=10):
    x = np.array(x0, dtype=float)
    for i in range(max_iter):
        fx = f(x)
        Jx = J(x)
        delta = np.linalg.solve(Jx, -fx)
        x = x + delta
        print(f"Iter {i+1}: x = {x}, |f| = {np.linalg.norm(fx):.3e}")
        if np.linalg.norm(fx) < tol:
            break
    return x

# initial guess (1, 1)
x0 = [1, 1]
sol = newton_2d(f, J, x0)

print(f"\nroot: x = {sol[0]:.6f}, y = {sol[1]:.6f}")
```

[Newton-Raphson Method for Multi-variable]

- $x_{k+1} = x_k - J^{-1}(x_k)f(x_k)$

d. Python 출력 결과

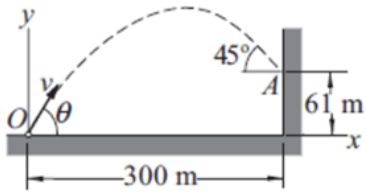
```
Iter 1: x = [1.23304038 0.44981653], |f| = 4.733e-01
Iter 2: x = [1.20807676 0.58320903], |f| = 1.150e-01
Iter 3: x = [1.2078277 0.5884153], |f| = 4.158e-03
Iter 4: x = [1.20782768 0.58842431], |f| = 7.472e-06
Iter 5: x = [1.20782768 0.58842431], |f| = 2.254e-11

root: x = 1.207828, y = 0.588424
```

5.4. (Least Square Method 적용 3) sin, cos 계수 추정

a. 문제

Problem 5.4



A projectile is launched at O with the velocity v at the angle θ to the horizontal. The parametric equations of the trajectory are

$$x = (\nu \cos\theta)t$$
$$y = -\frac{1}{2}gt^2 + (\nu \sin\theta)t$$

where t is the time measured from the instant of launch, and $g = 9.81\ \mathrm{m/s^2}$ represents the gravitational acceleration.

(a) If the projectile is to hit the target A at the $45°$ angle shown in the figure, determine v, θ, and the time of flight.

b. 풀이

sol) $x(t) = (v\cos\theta)t$

$y(t) = -\frac{1}{2}gt^2 + (v\sin\theta)t$

① 타겟 도달 시 위치 = $(x, y) = (300, 61)$

$$\begin{cases} (v\cos\theta)t = 300 & \cdots f_1 \\ -\frac{1}{2}gt^2 + (v\sin\theta)t = 61 & \cdots f_2 \end{cases}$$

② 타겟 도달 시 속도 벡터의 위상 = $-45°$

$\therefore \vec{V} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) = (v\cos\theta, -gt + v\sin\theta)$

$\tan(-45°) = \frac{-gt + v\sin\theta}{v\cos\theta} = -1\ ;\ \underbrace{-gt + v\sin\theta + v\cos\theta}_{f_3} = 0$

$\therefore x = \begin{bmatrix} v \\ \theta \\ t \end{bmatrix}$

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} (v\cos\theta)t - 300 \\ -\frac{1}{2}gt^2 + (v\sin\theta)t - 61 \\ -gt + v\sin\theta + v\cos\theta \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial t} \\ \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial t} \\ \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial \theta} & \frac{\partial f_3}{\partial t} \end{bmatrix} = \begin{bmatrix} (\cos\theta)t & -(v\sin\theta)t & v\cos\theta \\ (\sin\theta)t & (v\cos\theta)t & -gt + v\sin\theta \\ \sin\theta + \cos\theta & v\cos\theta - v\sin\theta & -g \end{bmatrix}$$

(연산 과정은 Python code 참고!)

답: $\begin{bmatrix} v \\ \theta \\ t \end{bmatrix} = \begin{bmatrix} 60.3583\ (m/s) \\ 54.5910\ (°) \\ 8.5789\ (s) \end{bmatrix}$

c. Python Code

```python
import numpy as np

# gravity constant
g = 9.81

# f(x) [v, theta, t] -> R^3
def f(vec):
    v, theta, t = vec
    return np.array([
        v * np.cos(theta) * t - 300,
        v * np.sin(theta) * t - 0.5 * g * t**2 - 61,
        -g * t + v * np.sin(theta) + v * np.cos(theta)
    ])

# Jacobian
def J(vec):
    v, theta, t = vec
    return np.array([
        [np.cos(theta) * t, -v * np.sin(theta) * t, v * np.cos(theta)],
        [np.sin(theta) * t,  v * np.cos(theta) * t, v * np.sin(theta) - g * t],
        [np.sin(theta) + np.cos(theta), v * (np.cos(theta) - np.sin(theta)), -g]
    ])

# Newton-Raphson for nonlinear system
def newton_system(f, J, x0, tol=1e-10, max_iter=20):
    x = np.array(x0, dtype=float)
    for i in range(max_iter):
        fx = f(x)
        Jx = J(x)
        dx = np.linalg.solve(Jx, -fx)
        x += dx
        print(f"Iter {i+1}: x = {x}, |f| = {np.linalg.norm(fx):.3e}")
        if np.linalg.norm(fx) < tol:
            break
    return x

# initial guess[v, theta, t]
x0 = [100, np.radians(45), 3]  # v = 100 m/s, ceta = 45, t = 3(s)
sol = newton_system(f, J, x0)

v, theta_rad, t = sol
theta_deg = np.degrees(theta_rad)

print(f"\nFinal roots:")
print(f"v = {v:.4f} m/s")
print(f"ceta = {theta_deg:.4f} degrees")
print(f"t = {t:.4f} seconds")
```

[Newton-Raphson Method for Multi-variable]

- $x_{k+1} = x_k - J^{-1}(x_k)f(x_k)$

d. Python 출력 결과

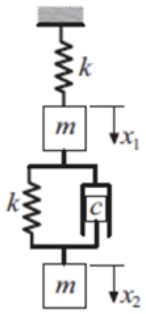
```
Iter 1: x = [36.35189032  0.4815379   5.24050319], |f| = 1.781e+02
Iter 2: x = [56.99191769  1.76602711  9.85365752], |f| = 1.696e+02
Iter 3: x = [37.7145373   1.1421785   7.28419029], |f| = 4.124e+02
Iter 4: x = [64.77990959  0.85287524  9.29996295], |f| = 2.002e+02
Iter 5: x = [60.22471872  0.93873411  8.60819923], |f| = 1.013e+02
Iter 6: x = [60.34653028  0.95276452  8.57890343], |f| = 8.833e+00
Iter 7: x = [60.353346    0.95279201  8.57894918], |f| = 5.995e-02
Iter 8: x = [60.35334598  0.95279201  8.57894918], |f| = 1.683e-06
Iter 9: x = [60.35334598  0.95279201  8.57894918], |f| = 7.105e-15

Final roots:
v = 60.3533 m/s
ceta = 54.5910 degrees
t = 8.5789 seconds
```

5.5. (Least Square Method 적용 4) 방사능 물질의 반감기 추정

a. 문제

Problem 5.5



The two blocks of mass m each are connected by springs and a dashpot. The stiffness of each spring is k, and c is the coefficient of damping of the dashpot. When the system is displaced and released, the displacement of each block during the ensuing motion has the form

$$x_k(t) = A_k e^{\omega_r t} \cos(\omega_i t + \phi_k), k = 1, 2$$

where A_k and ϕ_k are constants, and $\omega = \omega_r \pm i\omega_i$ are the roots of

$$\omega^4 + 2\frac{c}{m}\omega^3 + 3\frac{k}{m}\omega^2 + \frac{c}{m}\frac{k}{m}\omega + \left(\frac{k}{m}\right)^2 = 0$$

(a) Determine the two possible combinations of ω_r and ω_i if $c/m = 12\ \mathrm{s}^{-1}$ and $k/m = 1500\ \mathrm{s}^{-2}$.

b. 풀이

sol) $f(\omega) = \omega^4 + 24\omega^3 + 4500\omega^2 + 18000\omega + (1500)^2$ 이라 하자.

① Newton - Raphson Method

■ $f'(\omega) = \frac{df}{d\omega} = 4\omega^3 + 72\omega^2 + 9000\omega + 18000$

■ $\omega_{k+1} = \omega_k - \frac{f(\omega_k)}{f'(\omega_k)}$

(연산 과정은 Python code 참고!)

답: $\omega_r = -11.3769$

$\omega_i = 61.3545$

c. Python Code

```python
# f(w)
def f(w):
    return w**4 + 24*w**3 + 4500*w**2 + 18000*w + 2.25e6

def df(w):
    return 4*w**3 + 72*w**2 + 9000*w + 18000

# Newton-Raphson iteration
def newton_raphson(f, df, w0, tol=1e-10, max_iter=100):
    w = w0
    for i in range(max_iter):
        fw = f(w)
        dfw = df(w)
        if abs(dfw) < 1e-12:
            print(f"Derivative near to zero , Iter {i}")
            break
        w_next = w - fw / dfw
        print(f"Iter {i+1}: w = {w_next:.10f}, f(w) = {f(w_next):.3e}")
        if abs(f(w_next)) < tol:
            return w_next
        w = w_next
    return w

# initial guess
w0 = -10 + 60j
root = newton_raphson(f, df, w0)

print(f"\nroot: w = {root:.6f}")
```

[Newton-Raphson Method]

- $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

d. Python 출력 결과

```
Iter 1: w = -11.5638388594+61.3730474036j, f(w) = -2.391e+04+7.342e+04j
Iter 2: w = -11.3775997069+61.3530093709j, f(w) = -6.518e+02-3.586e+00j
Iter 3: w = -11.3769802761+61.3544728691j, f(w) = 3.905e-02-2.591e-02j
Iter 4: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 5: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 6: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 7: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 8: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 9: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 10: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 11: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 12: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 13: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 14: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 15: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 16: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 17: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 18: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 19: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 20: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 21: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 22: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 23: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 24: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 25: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 26: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 27: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 28: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 29: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 30: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 31: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 32: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 33: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 34: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 35: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 36: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 37: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 38: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 39: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 40: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 41: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 42: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 43: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 44: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 45: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 46: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 47: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 48: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 49: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 50: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 51: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 52: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
```

```
Iter 53: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 54: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 55: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 56: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 57: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 58: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 59: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 60: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 61: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 62: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 63: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 64: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 65: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 66: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 67: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 68: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 69: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 70: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 71: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 72: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 73: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 74: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 75: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 76: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 77: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 78: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 79: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 80: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 81: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 82: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 83: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 84: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 85: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 86: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 87: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 88: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 89: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 90: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 91: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 92: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 93: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 94: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 95: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 96: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 97: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 98: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j
Iter 99: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10-1.164e-09j
Iter 100: w = -11.3769803717+61.3544728066j, f(w) = -4.657e-10+6.985e-10j

root: w = -11.376980+61.354473j
```

5.6. (Least Square Method 적용 5) 다변수

a. 문제

Problem 5.6

Write a function $my\_newton(f, df, x0, tol)$, that returns $[R, E]$, where f is a function object, df is a function object to the derivative of f, $x0$ is an initial estimation of the root, and tol is a strictly positive scalar. The function should return an array, R, where $R[i]$ is the Newton-Raphson estimation of the root of f for the i-th iteration. Remember to include the initial estimate. The function should also return an array, E, where $E[i]$ is the value of $|f(R[i])|$ for the i-th iteration of the Newton-Raphson method. The function should terminate when $E(i) < tol$. You may assume that the derivative of f will not hit 0 during any iteration for any of the test cases given.

Test Cases:

```
In: f = lambda x: x**2 - 2
   df = lambda x: 2*x
    [R, E] = my_newton(f, df, 1, 1e-5)
Out: R = [1, 1.5, 1.4166666666666667, 1.4142156862745099]
     E = [1, 0.25, 0.006944444444444642, 6.007304882871267e-06]

In: f = lambda x: np.sin(x) - np.cos(x)
   df = lambda x: np.cos(x) + np.sin(x)
    [R, E] = my_newton(f, df, 1, 1e-5)
Out: R = [1, 0.782041901539138, 0.7853981759997019]
     E = [0.30116867893975674, 0.004746462127804163, 1.7822277875723103e-08]
```

b. Python Code

```python
import numpy as np

def my_newton(f, df, x0, tol):
    R = [x0]                  # 추정값 저장 리스트
    E = [abs(f(x0))]          # 오차 저장 리스트

    while E[-1] > tol:
        x_new = R[-1] - f(R[-1]) / df(R[-1])  # Newton-Raphson 식
        R.append(x_new)
        E.append(abs(f(x_new)))

    return R, E

# Test 1
f = lambda x: x**2 - 2
df = lambda x: 2*x
R, E = my_newton(f, df, 1, 1e-5)
print("R =", R)
print("E =", E)

# Test 2
f = lambda x: np.sin(x) - np.cos(x)
df = lambda x: np.cos(x) + np.sin(x)
R, E = my_newton(f, df, 1, 1e-5)
print("R =", R)
print("E =", E)
```

c. Test Case 검증결과

```
R = [1, 1.5, 1.4166666666666667, 1.4142156862745099]
E = [1, 0.25, 0.006944444444444642, 6.007304882871267e-06]
```

Test Case 1

```
R = [1, 0.782041901539138, 0.7853981759997019]
E = [0.30116867893975674, 0.004746462127804163, 1.7822277875723103e-08]
```

Test Case 2