Q1. (a) Number of iterations and root for $f(x) = x + e^{-x^2}\cos(x)$ are given as follows:

- Newton method:
  no. of iterations = 5
  root = -0.588401776500996
- Secant method:
  no. of iterations = 7
  root = -0.588401776500996
- Modified Newton's method:
  no. of iterations = 5
  root = -0.588401776500996

Asymptotic convergence rate of newton method, secant method and modified newton's method are 2, 1.62 and 2 respectively. As we can see in Table 1. Newton method and modified newton method converges in less no. of iterations than secant method.

Table 1.

n	Newton Method (x_n)	Secant Method(x_n)	Modified Newton Method (x_n)
0	0	0	0
1	-1	1	-0.25
2	-0.530644	-5.031039	-0.506305
3	-0.588627	-0.160519	-0.587167
4	-0.588402	-0.829831	-0.588402
5	-0.588402	-0.575638	-0.588402
6		-0.58871	
7		-0.588402	
8		-0.588402	

(b) Number of iterations and root for $f(x) = \left(x + e^{-x^2}\cos(x)\right)^2$ are given as follows:

- Newton method:
  no. of iterations = 19
  root = -0.588401776500996

- Secant method
  no. of iterations = 34
  root = -0.588401776500996

- Modified Newton's method:
  no. of iterations = 5
  root = -0.588401776500996
  out

problem (b) is related to multiplicity of roots.
no. of iterations in prob 1(a) and prob(b) is same for modified newton's method, because convergence rate for newton modified method is 2 for both simple and multiple roots.

No. of iteration in prob(a) is greater than no. of iteration in prob (b) in case of newton method, it is because of convergence rate of newton method is 2 for simple roots, and 1 for multiple roots.

No. of iteration in prob(a) is greater than no. of iteration in prob(b) in case of secant method, it is because of convergence rate of secant method is 1.62 for simple roots and 1 for multiple roots.

as convergence rate of newton method and secant method is 1, why there are more no. of iterations in secant method than newton method. one of the reasons may be that the asymptotic error constant of secant method is more than asymptotic error constant of newton method.

Table 2

n	Newton Method(x_n)	Secant Method(x_n)	Modified Newton's Method(x_n)
0	0	0	0
1	-0.5	1	-0.25
2	-0.544598	-2.288119	-0.506305
3	-0.56655	2.238743	-0.587167
4	-0.577483	89.857573	-0.588402
5	-0.582943	2.184523	-0.588402
6	-0.585673	2.132907	
7	-0.587037	1.097502	
8	-0.58772	0.572163	
9	-0.588061	-4.825996	
10	-0.588231	0.914284	
11	-0.588316	1.278537	
12	-0.588359	-0.377785	
13	-0.58838	-0.567691	
14	-0.588391	-0.569642	
15	-0.588396	-0.578563	
16	-0.588399	-0.58195	
17	-0.5884	-0.584506	
18	-0.588401	-0.585973	
19	-0.588401	-0.586906	
20		-0.587476	
21		-0.58783	
22		-0.588048	
23		-0.588183	
24		-0.588267	
25		-0.588318	
26		-0.58835	
27		-0.58837	
28		-0.588382	
29		-0.58839	
30		-0.588394	
31		-0.588397	
32		-0.588399	
33		-0.5884	
34		-0.588401	
35		-0.588401	

Question2 :

$$g(x) = x - \phi(x)f(x) - \psi(x)(f(x))^2 \quad -(1)$$

determine $\phi(x)$ and $\psi(x)$ asymptotic order of convergence and asymptotic error constant as $g(x)$ is a cubicly convergent fixed point iteration function $g(p_n)$.

$g(p) = p$ → fixed point
$f(p) = 0$ → finding a root 'p'
$g'(p) = 0$, $g''(p) = 0$ } cubicly convergent

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

$$\boxed{\phi(p) = \frac{1}{f'(p)}} \Rightarrow \phi'(p) = \frac{-f''(p)}{(f'(p))^2}$$

$$g''(x) = 0 - \phi'(x)f'(x) - \phi(x)f''(x) - \phi''(x)f(x) - f(x)\phi'(x) - 2\psi(x)((f'(x))^2 + f(x)f''(x)) - 2\psi'(x)f(x)f'(x) - \psi''(x)f^2(x) - \psi'(x)2f(x)f'(x)$$

$$g''(p) = 0 = 0 - \phi'(p)f'(p) - \phi(p)f''(p) - \phi''(p)f(p) - f(p)\phi'(p) - 2\psi(p)((f'(p))^2 + f(p)f''(p)) - 2\psi'(p)f(p)f'(p) - \psi''(p)f^2(p) - \psi(p)2f(p)f'(p)$$

$$0 = -2\left(\frac{-f''(p)}{(f'(p))^2}\right)f'(p) - \frac{f''(p)}{f'(p)} - 2\psi(p)(f'(p))^2$$

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

$$\psi(p) = \frac{f''(p)}{2f'(p)^3}$$

$$g(x) = x - \frac{f(x)}{f'(x)} - \frac{f''(x)}{2(f'(x))^3}(f(x))^2$$

$p_{n+1} = g(p_n)$ ← computer

$p = g(p)$ ← exact

$|p_{n+1} - p| = |g(p_n) - g(p)|$

$\epsilon_{n+1} = |g(p_n) - g(p)|$ —— ⊕

Taylor series expansion of $g(p_n)$ around p

$$g(p_n) = g(p) + \cancel{g'(p)}^{0}(p_n - p) + \cancel{g''(p)}^{0}\frac{(p_n-p)^2}{2} + \frac{g'''(p)(p_n-p)^3}{3!} + h.o.t \quad ⊛$$

plug ⊛ into ⊕

$$\epsilon_{n+1} = \left| g(p) + \frac{g'''(p)(p_n-p)^3}{3!} + h.o.t - g(p) \right|$$

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

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

$\alpha = 3$ asymptotic convergence rate.

$\lambda = \frac{g'''(p)}{3!}$ asymptotic error constant

Asymptotic error constant in terms of $f(p)$:

$$g'''(x) = -\frac{f^2(x) f''''(x)}{2(f'(x))^3} + \frac{6f^2(x) f''(x) f'''(x)}{(f'(x))^4} + \frac{9f^2(x)(f''(x))^2}{2(f'(x))^4} - \frac{36f^2(x)(f''(x))^2 f''(x)}{(f'(x))^5} + \frac{30f^2(x)(f''(x))^4}{(f'(x))^6}$$

$$\frac{-2f(a)\,f^{IIII}(a)}{(f'(a))^2} + \frac{17\,f(a)\,f''(a)\,f'''(a)}{(f'(a))^3} - \frac{21\,f(a)\,(f''(a))^3}{(f'(a))^4} - \frac{f'''(a)}{f'(a)} +$$

$$\frac{3\,(f''(a))^2}{(f'(a))^2}$$

put $f(a)=0$ we get,

$$g'''(b) = \frac{-f'''(b)}{f'(b)} + \frac{3(f''(b))^2}{(f'(b))^2}$$

$$\lambda = \frac{g'''(b)}{3!} = \frac{-f'''(b)}{6f'(b)} + \frac{(f''(b))^2}{2(f'(b))^2}$$

Question 3:

The relationship among the linkages

$$f_1(\theta_2, \theta_3) = r_2 \cos\theta_2 + r_3 \cos\theta_3 + r_4 \cos\theta_4 - r_1 = 0$$
$$f_2(\theta_2, \theta_3) = r_2 \sin\theta_2 + r_3 \sin\theta_3 + r_4 \sin\theta_4 = 0$$
$$\text{where } r_1 = 10, r_2 = 6, r_3 = 8, r_4 = 7, \theta_4 = 220^\circ$$

Solve for θ_2, θ_3 using Newton's method for systems of nonlinear equations.

Newton-Raphson for a multi-variable

$$f_i(x) = 0 \quad \Longrightarrow \quad x \to \text{root}$$
$$x \implies x^{(i)}$$

Approximate Taylor series expansion of f_i at $x^{(i)}$

$$f_i(x) = f_i(x^{(i)}) + \left.\frac{\partial f_i}{\partial x_1}\right|_{x^{(i)}} (x_1 - x_1^{(i)}) + \left.\frac{\partial f_i}{\partial x_2}\right|_{x^{(i)}} (x_2 - x_2^{(i)}) + \cdots = 0 \tag{1}$$

$$-f_i(x^{(i)}) = \left.\frac{\partial f_i}{\partial x_1}\right|_{x^{(i)}} (x_1 - x_1^{(i)}) + \left.\frac{\partial f_i}{\partial x_2}\right|_{x^{(i)}} (x_2 - x_2^{(i)})$$
$$-f_i(x^{(i)}) = J_{ij} \Delta x_j$$

For two variables θ_2, θ_3:

$$-f_1(\theta_2^{(i)}, \theta_3^{(i)}) = \left.\frac{\partial f_1}{\partial \theta_2}\right|_{\theta_2^{(i)}, \theta_3^{(i)}} (\theta_2^{(i+1)} - \theta_2^{(i)}) + \left.\frac{\partial f_1}{\partial \theta_3}\right|_{\theta_2^{(i)}, \theta_3^{(i)}} (\theta_3^{(i+1)} - \theta_3^{(i)})$$

$$-f_2(\theta_2^{(i)}, \theta_3^{(i)}) = \left.\frac{\partial f_2}{\partial \theta_2}\right|_{\theta_2^{(i)}, \theta_3^{(i)}} (\theta_2^{(i+1)} - \theta_2^{(i)}) + \left.\frac{\partial f_2}{\partial \theta_3}\right|_{\theta_2^{(i)}, \theta_3^{(i)}} (\theta_3^{(i+1)} - \theta_3^{(i)})$$

In matrix form:

$$\begin{bmatrix} -f_1(\theta_2^{(i)}, \theta_3^{(i)}) \\ -f_2(\theta_2^{(i)}, \theta_3^{(i)}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \theta_3} \\ \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} \end{bmatrix} \begin{bmatrix} \theta_2^{(i+1)} - \theta_2^{(i)} \\ \theta_3^{(i+1)} - \theta_3^{(i)} \end{bmatrix}$$

Let J be the Jacobian:

$$-J^{-1} \begin{bmatrix} f_1(\theta_2^{(i)}, \theta_3^{(i)}) \\ f_2(\theta_2^{(i)}, \theta_3^{(i)}) \end{bmatrix} = \begin{bmatrix} \theta_2^{(i+1)} - \theta_2^{(i)} \\ \theta_3^{(i+1)} - \theta_3^{(i)} \end{bmatrix}$$

Thus, we update:

$$\begin{bmatrix} \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \theta_2^{(i)} \\ \theta_3^{(i)} \end{bmatrix} - J^{-1} \begin{bmatrix} f_1(\theta_2^{(i)}, \theta_3^{(i)}) \\ f_2(\theta_2^{(i)}, \theta_3^{(i)}) \end{bmatrix}$$

Finally:

$$\begin{bmatrix} \theta_2^{(i+1)} \\ \theta_3^{(i+1)} \end{bmatrix} = \begin{bmatrix} \theta_2^{(i)} \\ \theta_3^{(i)} \end{bmatrix} - J^{-1} \begin{bmatrix} f_1(\theta_2^{(i)}, \theta_3^{(i)}) \\ f_2(\theta_2^{(i)}, \theta_3^{(i)}) \end{bmatrix} \tag{2}$$

In equation (1), or approximate Taylor series expansion, the value of θ is taken in radians. Therefore, we will take θ in radians for the solution of systems of nonlinear equations.

The roots of equations f_1 and f_2 were computed using a Python script provided in Appendix 1(b). The script utilizes equation (2) for the calculations. The roots are as follows:

$$\theta_2 = 32.01518^\circ$$

$$\theta_3 = -4.37099^\circ$$

The number of iterations required to compute θ_2 and θ_3 with a relative tolerance of 10^{-4} is 4.

Appendix 1

(a)

```
import numpy as np
import os
import matplotlib.pyplot as plt
import sympy as sp

# Function to calculate Newton's step
def calculate_newton_step(x_old, func, x):
    func_at_xold = func.subs(x, x_old)
    func_at_xold_real = func_at_xold.evalf()
    func_prime = sp.diff(func, x)
    func_prime_at_xold = func_prime.subs(x, x_old)
    func_prime_at_xold_real = func_prime_at_xold.evalf()

    numerator = func_at_xold_real
    denominator = func_prime_at_xold_real
    x_n_plus_1 = x_old - numerator / denominator
    return x_n_plus_1

# Newton's Method
def Newton_method(p_o, func, x):
    x_old = p_o
    x_new = calculate_newton_step(x_old, func, x)
    n = 1
    root = np.array([], dtype=float)
    root = np.append(root, x_old)
    root = np.append(root, x_new)
    while (abs(x_new - x_old)/abs(x_new)) > 0.000001:
        x_old = x_new
        x_new = calculate_newton_step(x_old, func, x)
        root = np.append(root, x_new)
        n = n + 1
    return x_new, root, n

# Function to calculate Secant step
def Calculate_secant_step(x_n, x_n_minus_1, func, x):
    func_at_x_n = func.subs(x, x_n)
    func_at_x_n_real = func_at_x_n.evalf()

    func_at_x_n_minus_1 = func.subs(x, x_n_minus_1)
    func_at_x_n_minus_1_real = func_at_x_n_minus_1.evalf()

    numerator = func_at_x_n_real * x_n_minus_1 - func_at_x_n_minus_1_real * x_n
    denominator = func_at_x_n_real - func_at_x_n_minus_1_real

    x_n_plus_1 = numerator / denominator
    return x_n_plus_1

# Secant Method
def Secant_method(p_o, p_1, func, x):
    x_n_minus_1 = p_o
    x_n = p_1
    x_n_plus_1 = Calculate_secant_step(x_n, x_n_minus_1, func, x)
    n = 1
    root = np.array([], dtype=float)
    root = np.append(root, x_n_minus_1)
    root = np.append(root, x_n)
    root = np.append(root, x_n_plus_1)
    while (abs(x_n_plus_1 - x_n)/abs(x_n_plus_1)) > 0.000001:
        x_temp = x_n
        x_n = x_n_plus_1
```

```
        x_n_plus_1 = Calculate_secant_step(x_n_plus_1, x_temp, func, x)
        root = np.append(root, x_n_plus_1)
        n = n + 1
    return x_n, root, n

# Function to calculate Modified Newton's step
def Calculate_modified_newton_step(x_old, func, x):
    func_prime = sp.diff(func, x)
    func_double_prime = sp.diff(func_prime, x)
    func_at_xold = func.subs(x, x_old)
    func_at_xold_real = func_at_xold.evalf()
    func_prime_at_xold = func_prime.subs(x, x_old)
    func_prime_at_xold_real = func_prime_at_xold.evalf()
    func_double_prime_at_xold = func_double_prime.subs(x, x_old)
    func_double_prime_at_xold_real = func_double_prime_at_xold.evalf()

    numerator = func_at_xold_real * func_prime_at_xold_real
    denominator = func_prime_at_xold_real**2 - func_at_xold_real *
        func_double_prime_at_xold_real
    x_new = x_old - numerator / denominator
    return x_new

# Modified Newton's Method
def Modified_newton_method(p_o, func, x):
    x_old = p_o
    x_new = Calculate_modified_newton_step(x_old, func, x)
    root = np.array([], dtype=float)
    root = np.append(root, x_old)
    root = np.append(root, x_new)
    n = 1
    while (abs(x_new - x_old)/abs(x_new)) > 0.000001:
        x_old = x_new
        x_new = Calculate_modified_newton_step(x_old, func, x)
        root = np.append(root, x_new)
        n += 1
    return x_new, root, n

# Define the symbolic variable and function
x = sp.symbols('x')
f1 = sp.exp(-x**2) * sp.cos(x) + x
f2 = (sp.exp(-x**2) * sp.cos(x) + x)**2

# Get root propagation for Newton, Secant, and Modified Newton methods
root_newton, root_newton_propagation, iterations_newtion = Newton_method(0, f1, x)
root_sec, root_sec_propagation, iterations_sec = Secant_method(0, 1, f1, x)
root_modified_newton, root_newton_modified_propagation, iterations_modified_newton =
    Modified_newton_method(0, f1, x)

print("Q1(a) \n")
print("Newton Method: ")
print("root : {}".format(root_newton))
print("Number of iterations: ", iterations_newtion)
print("\n")
print("Secant Method: ")
print("root : {}".format(root_sec))
print("Number of iterations: ", iterations_sec)
print("\n")
print("Modified Newton Method: ")
print("root : {}".format(root_modified_newton))
print("Number of iterations: ", iterations_modified_newton)
print("\n")

# Get root propagation for Newton, Secant, and Modified Newton methods
```

```
root_newton, root_newton_propagation, iterations_newtion = Newton_method(0, f2, x)
root_sec, root_sec_propagation, iterations_sec = Secant_method(0, 1, f2, x)
root_modified_newton, root_newton_modified_propagation, iterations_modified_newton =
    Modified_newton_method(0, f2, x)

print("Q1(b) \n")
print("Newton Method: ")
print("root : {}".format(root_newton))
print("Number of iterations: ", iterations_newtion)
print("\n")
print("Secant Method: ")
print("root : {}".format(root_sec))
print("Number of iterations: ", iterations_sec)
print("\n")
print("Modified Newton Method: ")
print("root : {}".format(root_modified_newton))
print("Number of iterations: ", iterations_modified_newton)
print("\n")
```

(b)

```
import numpy as np

# Constants
r1 = 10
r2 = 6
r3 = 8
r4 = 4
theta4 = np.deg2rad(220)

# Initial guesses (in radians)
theta2_old = np.deg2rad(30)
theta3_old = np.deg2rad(0)

# Function definitions
def f1(theta2, theta3):
    return r2 * np.cos(theta2) + r3 * np.cos(theta3) + r4 * np.cos(theta4) - r1

def f2(theta2, theta3):
    return r2 * np.sin(theta2) + r3 * np.sin(theta3) + r4 * np.sin(theta4)

# Jacobian matrix
def jacobian(theta2, theta3):
    return np.array([
        [-r2 * np.sin(theta2), -r3 * np.sin(theta3)],
        [r2 * np.cos(theta2), r3 * np.cos(theta3)]
    ])

F = np.array([f1(theta2_old, theta3_old), f2(theta2_old, theta3_old)])

# Compute Jacobian and its inverse
J = jacobian(theta2_old, theta3_old)
J_inv = np.linalg.inv(J)

```

```
# Update guess using Newton's method
delta = np.dot(J_inv, F)
theta2 = theta2_old - delta[0]
theta3 = theta3_old - delta[1]

# Newton's Method parameters
tolerance = 1e-4
iterations = 1

# Initialize error array

err = np.array([[abs(theta2 - theta2_old)/(abs(theta2)), abs(theta3 - theta3_old)/(abs(theta3))]])


while np.sqrt((abs(theta3 - theta3_old)/abs(theta3))**2 + (abs(theta2 - theta2_old)/abs(theta2))**2) > tolerance:
    theta2_old = theta2
    theta3_old = theta3
    # Compute function values
    F = np.array([f1(theta2, theta3), f2(theta2, theta3)])

    # Compute Jacobian and its inverse
    J = jacobian(theta2, theta3)
    J_inv = np.linalg.inv(J)

    # Update guess using Newton's method
    delta = np.dot(J_inv, F)
    theta2 = theta2 - delta[0]
    theta3 = theta3 - delta[1]

    # Update error array
    err = np.vstack([err, [abs(theta2 - theta2_old)/(abs(theta2)), abs(theta3 - theta3_old)/(abs(theta3))]])


    iterations += 1

# Output final results
theta2_deg = np.rad2deg(theta2)
theta3_deg = np.rad2deg(theta3)
print("Number of iterations: ", iterations)
print("theta2: ", theta2_deg)
print("theta3: ", theta3_deg)
```