Assignment-2: Numerical Solution of System of Non-Linear Algebraic Equation

Name:- Bajarang Mishra

Roll no:- 22CH10016

Group:- 2

Problem-1:

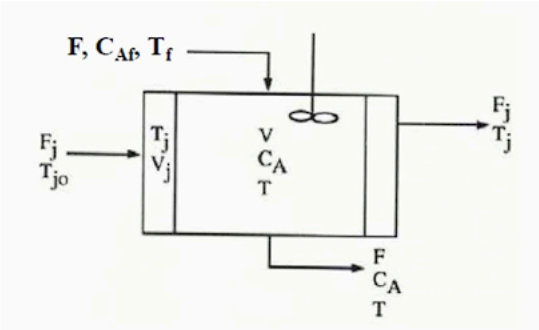
Objective: Numerical solution of a system of nonlinear algebraic equations.

This Assignment contains two problems. Solve both the problems.

Problem - 1:

Consider a perfectly mixed CSTR where a first-order exothermic irreversible reaction takes place (r = rate of reaction).

$$A \rightarrow B, \quad r = k_0 e^{\left(\frac{-E}{RT}\right)} C_A$$



Heat generated by reaction is being removed by the jacket fluid. The reactor volume (V) is constant.

Governing Equations:

(Subscript j indicates parameters related to jacket. Symbols carry their usual significance. Refer to the figure.)

Cont'd

$$V\frac{dC_A}{dt} = FC_{Af} - FC_A - rV$$

$$\rho C_p V\frac{dT}{dt} = \rho C_p F\left(T_f - T\right) + \left(-\Delta H\right)Vr - UA\left(T - T_j\right)$$

$$\rho_j C_j V_j\frac{dT_j}{dt} = \rho_j C_j F_j\left(T_{j0} - T_j\right) + UA\left(T - T_j\right)$$

Model Parameter Values:

Parameter	Value	Parameter	Value
F (m^3/h)	1	C_{Af} (kgmol/m^3)	10
V (m^3)	1	UA (kcal/°C h)	150
k_0 (h^{-1})	36×10^6	T_{j0} (K)	298
$(-\Delta H)$ (kcal/kgmol)	6500	$(\rho_j C_j)$ (kcal/m^3 °C)	600
E (kcal/kgmol)	12000	F_j (m^3/h)	1.25
(ρC_p) (kcal/m^3 °C)	500	V_j (m^3)	0.25
T_f (K)	298		

1. There are three steady states for this system. Identify all the steady states by setting LHS of the above ODE to zero and then solving the resulting algebraic equations simultaneously using Newton-Raphson method. Write your own code.

2. Solve the same problem using MATLAB function `fsolve` and compare your results.

Algorithm:-

So, here we are given with perfectly mixed Jacketed CSTR systems with first order exothermic irreversible reaction , and their corresponding mass balance and energy balance equations.

So to solve for the steady state equations, we take the 3 balance equations and equate the RHS term to 0, as at steady state all the variables will be independent of time thus d(variable)/dt will be 0.

Now we have 3 equations for 3 variables to solve, but as these are non linear solutions, we have to use Newton-Raphson method in matrix form to solve for all these 3 equations.

Xk+1=Xk- (F(Xk))(J(Xk)^-1)

Where X is a 3*1 matrix, which will give us the final solution, and F(Xk) is the Function matrix(containing F1,F2 and F3 3 equations) and J(Xk) is the jacobian matrix of that corresponding Function Matrix.

We will start with some guess value and based on the tolerance value it will converge to a solution.

Results:-

	Newton-Raphson			fsolve()		
Variables	CA (kgmol / m 3)	T (K)	Tj (K)	CA (kgmol / m 3)	T (K)	Tj (K)
Steady State 1	8.9686	308.73	299.79	8.9686	308.73	299.79
Steady State 2	6.1650	337.88	304.65	6.1650	337.88	304.65
Steady State 3	1.4094	387.34	312.89	1.4094	387.34	312.89

Conclusion:-

When comparing the performance of the `fsolve` function and the multivariable Newton-Raphson method for solving the steady-state equations of a Jacketed CSTR system with a first-order exothermic irreversible reaction, both methods yield identical results in terms of accuracy. However, the computational time for the `fsolve` function is generally lower compared to the Newton-Raphson method, as observed through profiling in MATLAB. This makes `fsolve` a more efficient choice for solving such systems when computational speed is a concern.

Nevertheless, care must be taken when selecting initial guesses for both methods, as poor choices can lead to divergence or a very slow convergence rate, which is undesirable. If the initial guesses are too far from the true solution, the Newton-Raphson method, in particular, may experience convergence issues or take a large number of iterations to converge. It is crucial to ensure that the initial guesses are within a reasonable range to avoid such problems.

Additionally, the multivariable Newton-Raphson method is only applicable if the system of equations is continuous and differentiable everywhere. This is important for ensuring the accuracy and reliability of the results. Since the equations in the CSTR system are interdependent, leading to a nonlinear system of equations, it is logical to expect that multiple steady-state solutions could be obtained, which further complicates the solution process.

In summary, while both methods provide the same final results, `fsolve` is more computationally efficient and easier to implement in many cases. However, careful consideration of initial guesses and the system's properties (such as continuity and differentiability) is essential for ensuring reliable and accurate solutions.

Code:-

```
% Problem 1: CSTR Steady States
function solve_cstr_steady_states()
    % Constants
    F = 1; C_Af = 10; V = 1; k0 = 36e6; E = 12000; R = 1.987;
    rhoCp = 500; deltaH = 6500; UA = 150; rhojCj = 600; Fj = 1.25;
    Tj0 = 298; Tf = 298;

    % Newton-Raphson
    func = @(T) 186250 - 625*T + 2.34e11 * exp(-12000/(1.987*T)) * (10 / (1 +
36e6 * exp(-12000/(1.987*T))));
    tolerance = 1e-6; max_iter = 100;
    initial_guesses = [310, 340, 380];
    roots_T = zeros(1,3); roots_Ca = zeros(1,3); roots_Tj = zeros(1,3);

    for i = 1:3
        T_guess = initial_guesses(i);
        for iter = 1:max_iter
            f = func(T_guess);
            df = (func(T_guess + 1e-6) - f) / 1e-6;
            delta = -f / df;
            T_guess = T_guess + delta;
```

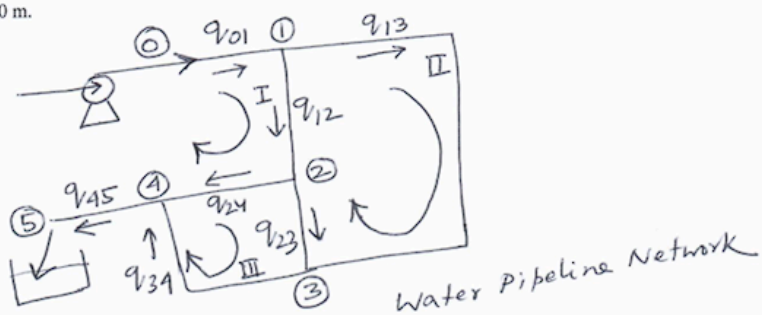
```
        if abs(delta) < tolerance, break; end
    end
    roots_T(i) = T_guess;
    exp_term = exp(-E/(R*T_guess));
    roots_Ca(i) = 10 / (1 + 36e6 * exp_term);
    roots_Tj(i) = (T_guess + 1490)/6;
  end

  % Display NR results
  disp('Newton-Raphson Results:');
  for i = 1:3
    fprintf('State %d: T=%.2f K, C_A=%.4f kmol/m³, Tj=%.2f K\n', ...
        i, roots_T(i), roots_Ca(i), roots_Tj(i));
  end

  % fsolve Validation
  options = optimoptions('fsolve', 'Display', 'off');
  solutions = zeros(3,3);
  for i = 1:3
    [sol, ~, exitflag] = fsolve(@(x) steady_eqs(x, F, C_Af, V, k0, E, R, rhoCp,
deltaH, UA, rhojCj, Fj, Tj0), ...
                    [roots_T(i), roots_Ca(i), roots_Tj(i)], options);
    if exitflag > 0, solutions(i,:) = sol; end
  end

  % Display fsolve results
  disp('fsolve Results:');
  for i = 1:3
    fprintf('State %d: T=%.2f K, C_A=%.4f kmol/m³, Tj=%.2f K\n', ...
        i, solutions(i,1), solutions(i,2), solutions(i,3));
  end
end

function F = steady_eqs(x, F_val, C_Af, V, k0, E, R, rhoCp, deltaH, UA,
rhojCj, Fj, Tj0)
```

```
    T = x(1); C_A = x(2); Tj = x(3);
    r = k0 * exp(-E/(R*T)) * C_A;
    F = [F_val*(C_Af - C_A) - r*V;
        rhoCp*F_val*(298 - T) + deltaH*V*r - UA*(T - Tj);
        rhojCj*Fj*(Tj0 - Tj) + UA*(T - Tj)];
end
```

Problem-2:

Problem – 2:

Consider the following water pipeline network. All the pipes have insider diameter (D) of 0.154 m. The pressure at the exit of the pump is 15 bar above atmospheric. The water is discharged at atmospheric pressure at the end of the pipeline. The equivalent lengths of the pipes connecting different nodes are: L_{01} = 100 m, $L_{12} = L_{23} = L_{45}$ = 300 m, and $L_{13} = L_{24} = L_{34}$ = 1200 m.



1. Calculate all the flow rates and pressures at nodes 1, 2, 3, and 4.
2. Suppose the pipeline between node 2 to node 4 becomes blocked. Calculate how the flow rates and pressure drops will change.

Given:

The pressure drop from node i to node j can be expressed as $\Delta P_{ij} = k_{ij}(q_{ij})^2$ where q_{ij} is the volumetric flow rate between nodes i and j. The terms k_{ij} are given as

$$k_{ij} = \frac{2 f_F \rho \Delta L_{ij}}{\pi^2 D^5}$$

Assume Fanning friction factor, $f_F = 0.005$ for all pipelines.

Take the density of water as, $\rho = 1000\ \text{kg/m}^3$.

Algorithm:-

The problem involves solving the flow and pressure distribution in a water pipeline network. The system includes multiple pipes connecting different nodes, with mass and energy balance equations governing the flow. The goal is to compute the flow rates and pressures at each node of the network.

Pipe diameter, length, Fanning friction factor, water density, pump pressure, and atmospheric pressure are provided.

The function `k_ij` (coefficients related to pressure drop across each pipe) is computed based on the pipe's length, diameter, friction factor, and the density.

Mass conservation equations: These represent the conservation of flow at each node. The flow entering each node must equal the flow exiting the node.

Energy balance equations: These represent the pressure drop across each pipe based on the flow rate using the Darcy-Weisbach equation.

The function `pipeline_equations` defines the system of mass conservation and energy balance equations based on the flow rates and pressures at different nodes.is defined to return the system of equations, which `fsolve` will solve.

The system includes 12 equations, which include both mass and energy balances.

Results:

part_1)

Flow rates (m^3/s):

q01: 0.3920 m^3/s
q12: 0.2589 m^3/s
q23: 0.0617 m^3/s
q13: 0.1331 m^3/s
q24: 0.1972 m^3/s
q34: 0.1948 m^3/s
q45: 0.3920 m^3/s

Pressures at nodes (Pa):

P1: 1.4203e+06 Pa
P2: 1.1851e+06 Pa
P3: 1.1717e+06 Pa
P4: 6.3918e+05 Pa
P5: 1.0000e+05 Pa

part_2)
Flow rates (m^3/s):
q01: 0.2665 m^3/s
q12: 0.1561 m^3/s
q23: 0.1104 m^3/s
q13: 0.1561 m^3/s
q24: 0 m^3/s
q34: 0.2665 m^3/s
q45: 0.2665 m^3/s
Pressures at nodes (Pa):
P1: 1.4169 e+06 Pa
P2: 1.3314 e+06 Pa
P3: 1.2459 e+06 Pa
P4: 0.2492 e+05 Pa
P5: 1.0000e+05 Pa

Conclusion:-
This algorithm models a water pipeline network, computes flow rates and pressures using a set of mass and energy balance equations, and solves the system using `fsolve`. By providing initial guesses, it iteratively solves the nonlinear equations for steady-state conditions. The results give the flow rates and pressures at various nodes, which are crucial for understanding the behavior of the pipeline system

Code:-
Q2_part_1

```
% Water Pipeline Network Analysis
tic

% Given Data
D = 0.154;  % Pipe diameter (m)
```

```
L = [100, 300, 300, 1200, 1200, 300, 1200]; % Pipe lengths (m) for L01, L12, L23, L13, L34, L45, L24
jf = 0.005; % Fanning friction factor
density = 1000; % Density of water (kg/m^3)
pressure_pump = 15e5; % 15 bar above atmospheric in Pascals
pressure_atm = 1e5; % Atmospheric pressure at outlet (1 bar in Pascals)

% Compute k_ij values
k = zeros(1, length(L));
for i = 1:length(L)
  k(i) = (2 * jf * density * L(i)) / (pi^2 * D^5);
end

% Define initial guesses for flow rates and pressures
x_initial = [10;10;10;10;10;10;10]; % Initial guess for fsolve
x_initial(8) = pressure_pump + pressure_atm; % P1 will be pump pressure + atmospheric pressure
x_initial(9) = pressure_pump + pressure_atm; % P2 is initialized similarly
x_initial(10) = pressure_atm; % P3 starts at atmospheric pressure
x_initial(11) = pressure_atm; % P4 starts at atmospheric pressure
x_initial(12) = pressure_atm; % P5 will be atmospheric pressure

disp('Initial guess values:');
disp(x_initial);

% Solve the system using fsolve
x_solution = fsolve(@(x) pipeline_equations(x, k, pressure_pump, pressure_atm), x_initial);

% Extract results
q01 = x_solution(1);  q12 = x_solution(2);  q23 = x_solution(3);  q13 = x_solution(4);
q24 = x_solution(5);  q34 = x_solution(6);  q45 = x_solution(7);
P1 = x_solution(8);   P2 = x_solution(9);   P3 = x_solution(10);  P4 = x_solution(11);
```

```
P5 = x_solution(12);

% Display results with proper formatting
disp('Flow rates (m^3/s):');
disp(['q01: ', num2str(q01, '%.4f'), ' m^3/s']);
disp(['q12: ', num2str(q12, '%.4f'), ' m^3/s']);
disp(['q23: ', num2str(q23, '%.4f'), ' m^3/s']);
disp(['q13: ', num2str(q13, '%.4f'), ' m^3/s']);
disp(['q24: ', num2str(q24, '%.4f'), ' m^3/s']);
disp(['q34: ', num2str(q34, '%.4f'), ' m^3/s']);
disp(['q45: ', num2str(q45, '%.4f'), ' m^3/s']);

disp('Pressures at nodes (Pa):');
disp(['P1: ', num2str(P1, '%.4e'), ' Pa']);
disp(['P2: ', num2str(P2, '%.4e'), ' Pa']);
disp(['P3: ', num2str(P3, '%.4e'), ' Pa']);
disp(['P4: ', num2str(P4, '%.4e'), ' Pa']);
disp(['P5: ', num2str(P5, '%.4e'), ' Pa']);

toc

% FUNCTION DEFINITIONS MUST BE AT THE END OF THE SCRIPT
function F = pipeline_equations(x, k, pressure_pump, pressure_atm)
  q01 = x(1); q12 = x(2); q23 = x(3); q13 = x(4);
  q24 = x(5); q34 = x(6); q45 = x(7);
  P1 = x(8); P2 = x(9); P3 = x(10); P4 = x(11); P5 = x(12);

  % Mass conservation equations
  F(1) = q01 - q12 - q13; % Node 1
  F(2) = q12 - q24 - q23; % Node 2
  F(3) = q13 + q23 - q34; % Node 3
  F(4) = q24 + q34 - q45; % Node 4

  % Energy equations (pressure drop)
  F(5) = pressure_pump + pressure_atm - k(1) * q01^2 - P1;
```

```
  F(6) = P1 - k(2) * q12^2 - P2;
  F(7) = P2 - k(3) * q23^2 - P3;
  F(8) = P1 - k(4) * q13^2 - P3;
  F(9) = P3 - k(5) * q34^2 - P4;
  F(10) = P4 - k(6) * q45^2 - P5;
  F(11) = P2 - k(7) * q24^2 - P4;
  F(12) = P5 - pressure_atm; % Ensure P5 reaches atmospheric pressure
end
```

Q2_Part_2

```
% Given parameters
ff = 0.005; % Fanning friction factor
rho = 1000; % Density of water (kg/m^3)
D = 0.154; % Diameter of pipe (m)
pi_sq = pi^2; % Square of pi for reuse

% Pipe lengths (m)
L01 = 100;   % Length of pipe 01
L12 = 300;   % Length of pipe 12
L13 = 1200;  % Length of pipe 13
L23 = 300;   % Length of pipe 23
L34 = 1200;  % Length of pipe 34
L45 = 300;   % Length of pipe 45

% Calculate k_ij for all pipes
k01 = (2 * ff * rho * L01) / (pi_sq * D^5);
k12 = (2 * ff * rho * L12) / (pi_sq * D^5);
k13 = (2 * ff * rho * L13) / (pi_sq * D^5);
k23 = (2 * ff * rho * L23) / (pi_sq * D^5);
k34 = (2 * ff * rho * L34) / (pi_sq * D^5);
k45 = (2 * ff * rho * L45) / (pi_sq * D^5);
```

```
% Known pressure at the pump exit
P0 = 15e5; % 15 bar in Pascals
P5 = 0;    % Atmospheric pressure at the outlet

% Solve for flow rates and pressures
syms q01 q12 q13 q23 q34 q45 P1 P2 P3 P4

% Flow conservation at each node
eq1 = q01 == q12 + q13;      % Node 1
eq2 = q23 == q12;      % Node 2
eq3 = q13 + q23 == q34;      % Node 3
eq4 = q34 == q45;           % Node 4

% Pressure drop equations
eq5 = P0 - P1 == k01 * q01^2; % Pipe 01
eq6 = P1 - P2 == k12 * q12^2; % Pipe 12
eq7 = P2 - P3 == k23 * q23^2; % Pipe 23
eq8 = P3 - P4 == k34 * q34^2; % Pipe 34
eq10 = P4 - P5 == k45 * q45^2; % Pipe 45
eq11 = P1 - P3 == k13 * q13^2; % Pipe 13

% Solve the system of equations
sol = solve([eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq10, eq11], ...
    [q01, q12, q13, q23, q34, q45, P1, P2, P3, P4]);

% Display results
disp('Flow rates (m^3/s):');
disp(double([sol.q01, sol.q12, sol.q13, sol.q23, sol.q34, sol.q45]));

disp('Pressures (Pa):');
disp(double([sol.P1, sol.P2, sol.P3, sol.P4]));
```