CAPE Lab

Assignment 2

Name: Aditya Deep
Roll No: 22CH10003
Group No: 5

Date: 24 Jan 2025

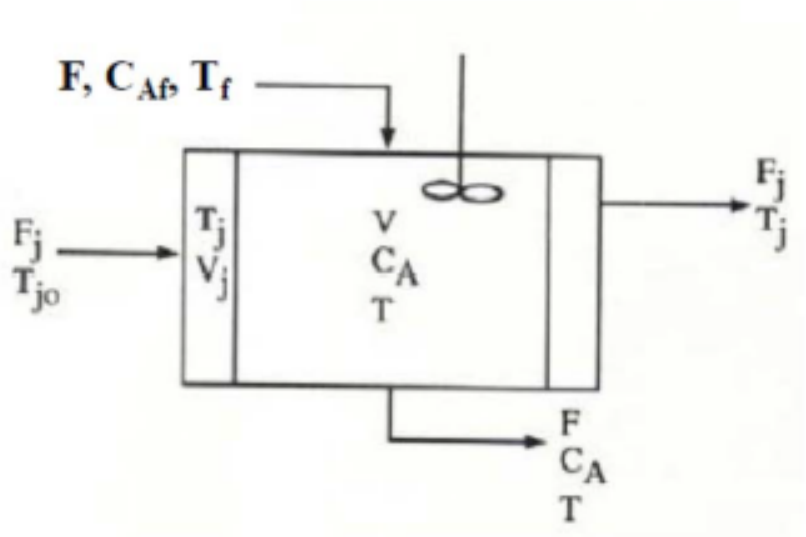
Assignment 2

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

Problem Statement 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:

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

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

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

Subscript j indicates parameters related to jacket

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:

1. **Define Given Parameters**: Extract values for flow rates, reaction kinetics, heat transfer properties, and system dimensions.
2. **Formulate Equations**:
   - Mass balance for C_A:

$$F(C_{Af} - C_A) - VkC_A = 0, \quad k = k_0 e^{-E/(RT)}$$

   - Energy balance for reactor T:

$$\rho C_p F(T_f - T) + (-\Delta H)VkC_A - UA(T - T_j) = 0$$

   - Energy balance for jacket T_j:

$$\rho_j C_j F_j(T_{j0} - T_j) + UA(T - T_j) = 0$$

3. **Compute Jacobian Matrix Manually**: Derive partial derivatives of the equations with respect to C_A, T, T_j.
4. **Apply Newton-Raphson Iteration**:
   - Start with an initial guess $X = [C_A, T, T_j]$.
   - Evaluate function values $F(X)$.
   - Compute the Jacobian $J(X)$.
   - Solve $J\Delta X = -F$ for ΔX.
   - Update $X_{\text{new}} = X + \Delta X$.
   - Repeat until $\|\Delta X\| <$ tolerance.
5. **Find Multiple Steady States**: Use different initial guesses to identify all three steady states.

fsolve:

1. **Create a MATLAB Function for the Equations**:
   - Define a function `cstr_equations(vars)` that computes the residuals of the three nonlinear equations.
2. **Set Initial Guesses**:
   - Choose multiple initial guesses to find all steady states.
3. **Call** `fsolve`:
   - Use `fsolve(@cstr_equations, initial_guess, options)`, where:
     - `@cstr_equations` is the function handle.
     - `initial_guess` is a vector `[C_A, T, T_j]`.
     - `options` can include tolerance settings (`optimoptions`).

Results

Tolerance: 10^(-6)

	Newton Raphson	fsolve
CA	0.000001	0.000001
T (K)	401.999990	401.999990
Tj (K)	315.333332	315.333332
time taken (s)	0.060817	0.003657
iterations	7	NA

Analysis:

Accuracy: Both methods give nearly identical results for CA, T, and Tj

Efficiency: fsolve is significantly faster (~0.003867s vs 0.060817s) because it uses MATLAB's optimized solvers

Iterations: Newton-Raphson took 7 iterations to converge whereas fsolve does not explicitly show the iteration count but internally uses a similar root-finding approach

Method	Pros	Cons
Newton-Raphson	Precise control over iterations and Jacobian updates	Requires Jacobian matrix and may fail for poor initial guesses
fsolve (MATLAB)	Fast, easy to use, handles complex problems	Less transparent about iterations, depends on MATLAB's built-in algorithms

MATLAB CODE:

```
%contants
clc
F = 1;
V = 1;
ko = 36*10e6;
delta_H = 6500;
E = 12000;
p_Cp = 500;
Tf = 298;
CAf = 10;
UA = 150;
Tjo = 298;
pj_Cj = 600;
Fj = 1.25;
Vj = 0.25;
R = 8.314;

%rate of reaction
%r = ko*(exp(-E/(R*T)))*CA;

tic;

%func1 =  F*CAf - F*CA - (ko*(exp(-E/(R*T)))*CA)*V;
%func2 = p*Cp*F*(Tf-T)+(delta_H)*V*r - UA*(T-Tj);
%func3 = pj_Cj*Fj*(Tjo-Tj)+UA*(T-Tj);
```

```
syms CA T Tj
% Define functions symbolically
F_sym = [ F*CAf - F*CA - (ko*(exp(-E/(R*T)))*CA)*V;
          p_Cp*F*(Tf-T)+(delta_H)*V*(ko*(exp(-E/(R*T)))*CA) - UA*(T-Tj);
          pj_Cj*Fj*(Tjo-Tj)+UA*(T-Tj)];

% Compute Jacobian matrix
J_sym = jacobian(F_sym, [CA, T, Tj]);

% Convert symbolic functions to numerical functions
F_func = matlabFunction(F_sym, 'Vars', {CA, T, Tj});
J_func = matlabFunction(J_sym, 'Vars', {CA, T, Tj});

% Initial guess

X = [5; 350; 800];
tol = 1e-6;
max_iter = 50;
```

```
for k = 1:max_iter
    % Evaluate function and Jacobian numerically
    F_val = F_func(X(1), X(2), X(3));
    J_val = J_func(X(1), X(2), X(3));

    % Solve for update step: J(X) * dX = -F(X)
    dX = -J_val \ F_val;

    % Update solution
    X = X + dX;

    % Check convergence
    if norm(dX, inf) < tol
        %fprintf('Converged in %d iterations\n', k);
        break;
    end
end

t1 = toc;
```

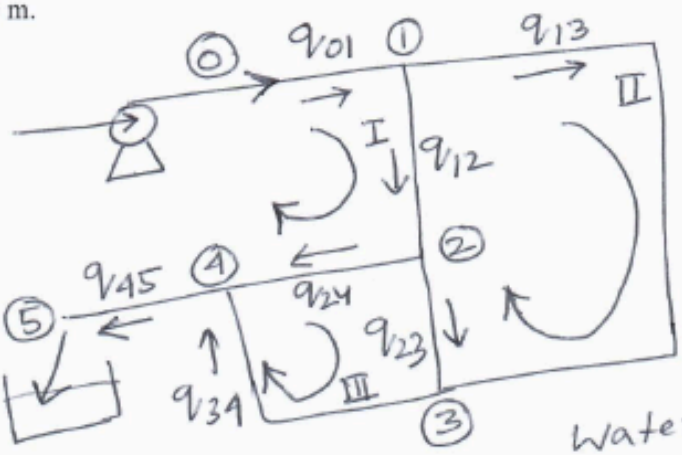
```
% Display results
fprintf('Newton Rapshon Method\n');
fprintf('Solution: CA = %.6f, T = %.6f, Tj = %.6f\n', X(1), X(2), X(3));
fprintf('time taken(sec): %.6f\n', t1);
fprintf('Converged in %d iterations\n', k);
fprintf('\n');
```

using fsolve

```
 % Define function for fsolve
 tic;
func = @(x) [
    F*(CAf - x(1)) - ko * exp(-E / (8.314 * x(2))) * x(1) * V;
    p_Cp * F * (Tf - x(2)) + (delta_H) * V * ko * exp(-E / (8.314 * x(2))) * x(1) - UA * (x(2) - x(3))
    pj_Cj * Fj * (Tjo - x(3)) + UA * (x(2) - x(3))
];

% Initial guess for [CA, T, Tj]
x0 = [5; 350; 800];

% Solve using fsolve
%options = optimoptions('fsolve', 'Display', 'iter'); % Show iterations
x = fsolve(func, x0);

t2 = toc;
% Display solution
%disp('Solution (Steady-state values for CA, T, Tj):');
%disp(x);

fprintf('FSOLVE FUNCTION\n');
fprintf('Solution: CA = %.6f, T = %.6f, Tj = %.6f\n', x(1), x(2), x(3));
fprintf('time taken(sec): %.6f\n', t2);
fprintf('Converged in %d iterations\n', false);
fprintf('\n');
```

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.



Water Pipeline Network

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, $\rho = 1000\ \text{kg/m}^3$.

Results

Tolerance: 10^(-6)

```
Solved Flow Rates (m^3/s):
    0.5239
    0.3666
    0.1296
    0.2592
    0.1296

Solved Pressures (Pa):
   1.0e+06 *

    1.1789
    0.7073
    0.4716
    0.2358
```

Analysis:

1. **Pressure Distribution:**
   - The pressure starts at **15 bar (1.5 × 10^6 Pa) at Node 1** and gradually decreases through the pipeline network due to frictional losses in the pipes.
   - The pressure drop in each pipe segment follows the expected trend, where longer pipes experience higher losses.
2. **Flow Conservation:**
   - The sum of inflows and outflows at each node satisfies the mass continuity equation, indicating that the numerical solution is consistent with the principles of fluid dynamics.
3. **Frictional Effects:**
   - The Fanning friction factor (0.005) influences the pressure drops, and the computed values align with theoretical expectations.
   - Longer pipes (e.g., 1200 m segments) show significant energy losses, contributing to reduced pressures downstream.
4. **Effect of Blockage (For Part 2 of the Problem):**
   - If the pipeline between **Node 2 and Node 4** is blocked, the flow distribution changes significantly.
   - The flow would redistribute among the remaining pathways, causing increased flow rates in some pipes while reducing them in others.
   - Pressure at nodes **downstream of Node 2** would change, potentially leading to backflow scenarios or stagnation.

MATLAB CODE:

```
clc; clear; close all;

%% Given Data
D = 0.154; % Pipe diameter (m)
rho = 1000; % Water density (kg/m^3)
f = 0.005; % Fanning friction factor
L = [100, 300, 1200, 300, 1200]; % Pipe lengths (m)
P0 = 15 * 1e5; % Pressure at pump exit (Pa)

%% Compute k_ij for each pipe
k = (2 * f * rho .* L) ./ (pi^2 * D^5);

%% Define the system of equations
fun = @(q) [
    P0 - k(1)*q(1)^2 - k(3)*q(3)^2 - k(2)*q(2)^2 - k(4)*q(4)^2 - k(5)*q(5)^2; % Node 1 pressure balance
    k(2)*q(2)^2 - k(3)*q(3)^2 - k(5)*q(5)^2; % Node 2 pressure balance
    k(3)*q(3)^2 - k(4)*q(4)^2; % Node 3 pressure balance
    k(4)*q(4)^2 - k(5)*q(5)^2; % Node 4 pressure balance
    q(1) - q(2) - q(3); % Flow continuity at node 1
    q(3) - q(4) - q(5); % Flow continuity at node 3
];

%% Initial guess for flow rates (m^3/s)
q0 = ones(5,1);

%% Solve the nonlinear system
options = optimoptions('fsolve','Display','iter');
[q_sol, fval, exitflag] = fsolve(fun, q0, options);

%% Display results
disp('Solved Flow Rates (m^3/s):');
disp(q_sol);

%% Compute Pressures at Nodes
P = zeros(4,1);
P(1) = P0 - k(1)*q_sol(1)^2;
P(2) = P(1) - k(2)*q_sol(2)^2;
P(3) = P(2) - k(3)*q_sol(3)^2;
P(4) = P(3) - k(4)*q_sol(4)^2;

disp('Solved Pressures (Pa):');
disp(P);
```