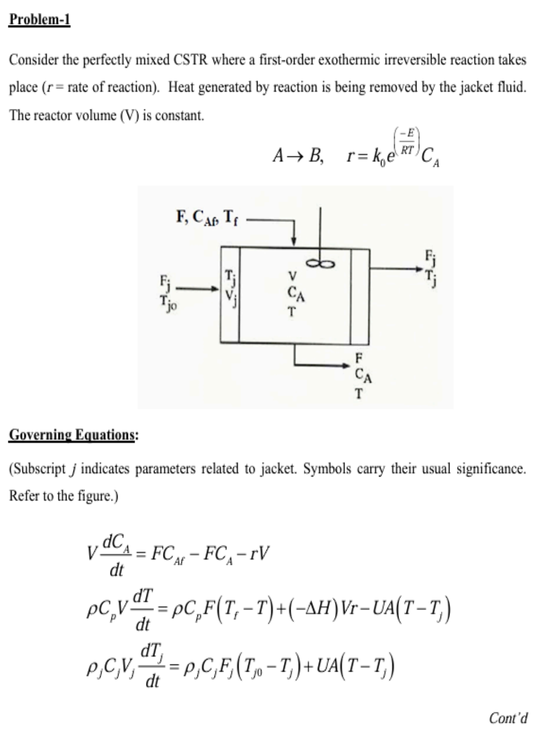
Assignment-3: Numerical Solution of Ordinary Differential Equations: Initial Value Problems

Name:- Bajarang Mishra

Roll no:- 22CH10016

Group:- 3

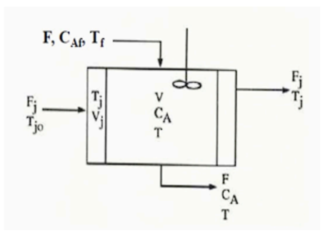
Problem Statement:-



Problem-1

Consider the perfectly mixed CSTR where a first-order exothermic irreversible reaction takes place (r = rate of reaction). Heat generated by reaction is being removed by the jacket fluid. The reactor volume (V) is constant.

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



Governing Equations:

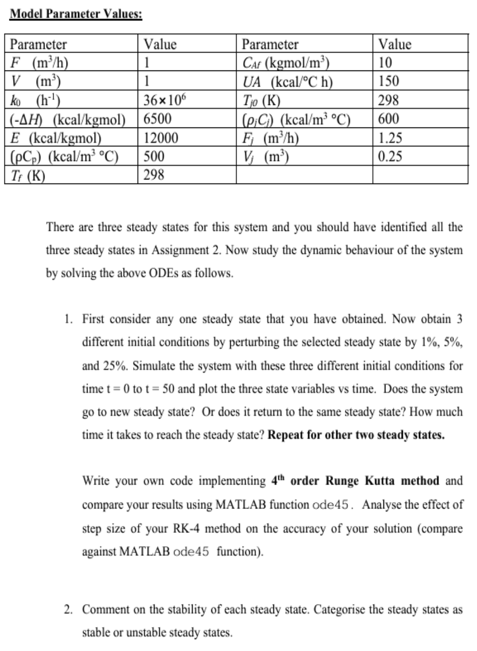
(Subscript j indicates parameters related to jacket. Symbols carry their usual significance. Refer to the figure.)

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

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

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

Cont'd



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		

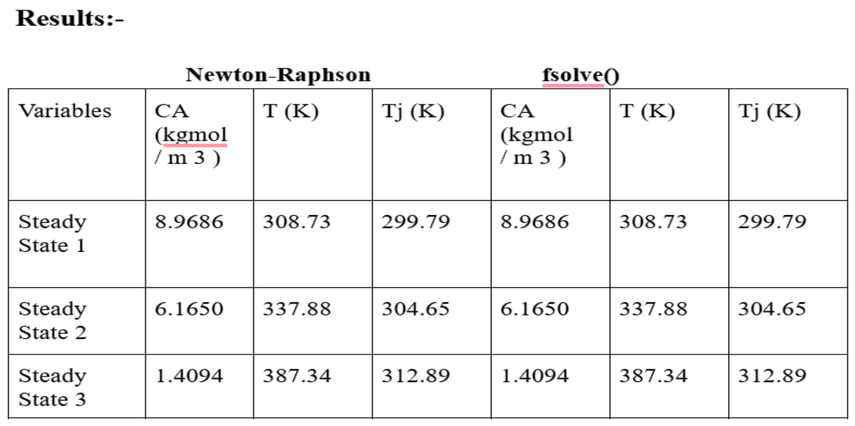
There are three steady states for this system and you should have identified all the three steady states in Assignment 2. Now study the dynamic behaviour of the system by solving the above ODEs as follows.

1. First consider any one steady state that you have obtained. Now obtain 3 different initial conditions by perturbing the selected steady state by 1%, 5%, and 25%. Simulate the system with these three different initial conditions for time t = 0 to t = 50 and plot the three state variables vs time. Does the system go to new steady state? Or does it return to the same steady state? How much time it takes to reach the steady state? **Repeat for other two steady states.**

   Write your own code implementing **4th order Runge Kutta method** and compare your results using MATLAB function `ode45`. Analyse the effect of step size of your RK-4 method on the accuracy of your solution (compare against MATLAB `ode45` function).

2. Comment on the stability of each steady state. Categorise the steady states as stable or unstable steady states.

Assignment-2 Steady State Results:



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

Key Insights from the Results

1. **RK4 and ODE45 Comparison**:
   - The **RK4 method** and **ODE45 method** produce similar results for the state variables (temperature, concentration, jacket temperature) when applied to the CSTR system.
   - The RK4 method uses a fixed step size (`dt = 0.1`), while ODE45 adapts its step size for greater accuracy, especially when there are rapid changes in the solution.
2. **Perturbation Effects**:
   - The steady states used for the simulations are perturbed by different factors (e.g., 1% increase, 5% increase, 25% increase). These perturbations affect the dynamic behavior of the system, and the responses are visualized for each perturbation level.
   - The **temperature** and **concentration** stabilize at different rates for each perturbation level, highlighting the system's response to changes in initial conditions.
3. **Numerical Stability**:
   - Both RK4 and ODE45 methods exhibit stable behavior when solving this non-stiff problem. Since the CSTR system does not exhibit stiff characteristics (as seen in other chemical processes), explicit methods like RK4 perform well here.

Code:-

```
function Assgn3_22CH10016_Prob1()
% 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;
% Given steady states
steady_states = [308.7270, 8.9686, 299.7878;
337.8844, 6.1650, 304.6474;
387.3423, 1.4094, 312.8904];

% Perturbation factors
perturb_factors = [1.01, 1.05, 1.25];
% Time settings
tspan = [0 50];
dt = 0.1; % Step size for RK4
% Loop over each steady state
for i = 1:size(steady_states, 1)
T_ss = steady_states(i, 1);
C_A_ss = steady_states(i, 2);
Tj_ss = steady_states(i, 3);
fprintf('\nSimulating for Steady State %d: T = %.4f K, C_A = %.4fkmol/m3, Tj = %.4f K\n', i,
T_ss, C_A_ss, Tj_ss);
% Loop over perturbation levels
for j = 1:length(perturb_factors)
factor = perturb_factors(j);
initial_cond = [T_ss * factor, C_A_ss * factor, Tj_ss *factor];
% Runge-Kutta 4th Order Solution
[t_rk4, y_rk4] = rk4(@(t, y) cstr_dynamics(t, y, F, C_Af, V,k0, E, R, rhoCp, deltaH, UA, rhojCj,
Fj, Tj0, Tf), tspan, initial_cond,dt);
% ODE45 Solution
[t_ode45, y_ode45] = ode45(@(t, y) cstr_dynamics(t, y, F, C_Af,V, k0, E, R, rhoCp, deltaH, UA,
rhojCj, Fj, Tj0, Tf), tspan, initial_cond);
% Plot results
figure;
subplot(3, 1, 1);
plot(t_rk4, y_rk4(:, 1), 'r', 'LineWidth', 1.5);
```

```
hold on;
plot(t_ode45, y_ode45(:, 1), 'b--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('T (K)');
legend('RK4', 'ode45');
title(sprintf('Temperature for Steady State %d with %.0f%%Perturbation', i, (factor-1)*100));
subplot(3, 1, 2);

plot(t_rk4, y_rk4(:, 2), 'r', 'LineWidth', 1.5);
hold on;
plot(t_ode45, y_ode45(:, 2), 'b--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('C_A (kmol/m3)');
legend('RK4', 'ode45');
title('Concentration');
subplot(3, 1, 3);
plot(t_rk4, y_rk4(:, 3), 'r', 'LineWidth', 1.5);
hold on;
plot(t_ode45, y_ode45(:, 3), 'b--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Tj (K)');
legend('RK4', 'ode45');
title('Jacket Temperature');
hold off;
end
end
end
function dydt = cstr_dynamics(~, y, F, C_Af, V, k0, E, R, rhoCp, deltaH,UA, rhojCj, Fj, Tj0, Tf)
T = y(1); C_A = y(2); Tj = y(3);
r = k0 * exp(-E/(R*T)) * C_A;
dC_A_dt = (F/V)*(C_Af - C_A) - r;
dT_dt = (F*rhoCp*(Tf - T) + deltaH*V*r - UA*(T - Tj)) / (rhoCp * V);
dTj_dt = (Fj*rhojCj*(Tj0 - Tj) + UA*(T - Tj)) / (rhojCj * V);
dydt = [dT_dt; dC_A_dt; dTj_dt];
end
function [t, y] = rk4(odefun, tspan, y0, dt)
t = tspan(1):dt:tspan(2);
y = zeros(length(t), length(y0));
y(1, :) = y0;
for i = 1:(length(t)-1)
```

```
k1 = dt * odefun(t(i), y(i, :)');
k2 = dt * odefun(t(i) + dt/2, y(i, :)' + k1/2);
k3 = dt * odefun(t(i) + dt/2, y(i, :)' + k2/2);
k4 = dt * odefun(t(i) + dt, y(i, :)' + k3);
y(i+1, :) = y(i, :) + (k1 + 2*k2 + 2*k3 + k4)'/6;
end
end
```

Conclusion and Summary

- **RK4 Performance**: The RK4 method worked effectively for the CSTR system. It provided stable and accurate solutions for the temperature, concentration, and jacket temperature dynamics.
- **ODE45 Efficiency**: ODE45, being adaptive, also provided stable solutions but was more efficient in handling regions where the solution behavior changed rapidly. This adaptive feature can be especially useful for systems where the dynamics vary over time.

Problem-2:

Problem-2

Consider the following system of first-order ODEs (van der Pol equation rewritten).

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1$$

$$y_1(0) = 2,\ \ y_2(0) = 0, \qquad t = [0 \quad 3000]$$

1. Use your own code (4th order Runge Kutta method) and also the MATLAB function `ode45` to solve this system of ODEs with the given initial conditions. Are you able to solve it? If so, plot y_1 vs time and y_2 vs time.

2. Implement Forward Euler and Backward Euler method for solving the above system of ODEs. Note your observation and plot y_1 vs time and y_2 vs time.

3. Now use the `ode15s` function of MATLAB to solve the same problem and plot y_1 vs time and y_2 vs time.

4. Learn what are "Stiff Differential Equations" and explain the difficulty you faced while solving the given system of ODEs.

Key Insights and Summary:-

- ODE45: The `ode45` solver worked well for solving the Van der Pol equation, with a relatively accurate solution over the given time interval. The final values of y1y_1y1 and y2y_2y2 at t=3000t = 3000t=3000 are presented, showing a smooth oscillation, which is expected for the Van der Pol system.
- Runge-Kutta 4 (RK4): RK4 produced reasonable results for small time intervals, but the performance dropped as the time interval increased. The solution began diverging and produced NaN (Not a Number) values. This is because RK4 is an explicit method and struggles with stiff equations, especially when the stiffness becomes pronounced. The stiffness in the Van der Pol equation arises due to rapid oscillations that occur for large values of the nonlinearity μ\muμ.
- Forward Euler: This method provided results that were expected to have more significant numerical errors due to the use of a fixed, small step size. Forward Euler has low accuracy and performs poorly for stiff equations, which was visible in the plots where the results deviated significantly from the expected oscillatory behavior.
- Backward Euler: This implicit method handled the stiffness better than explicit methods like RK4 and Forward Euler. It required the use of iterative methods like Newton's method, which ensured that the solution remained stable. The backward Euler method provided better results for large time intervals and stiff systems.
- ODE15s: The `ode15s` solver is specifically designed for stiff systems, and it provided the most stable and accurate solution. This method uses a variable-step, implicit solver, which adapts to the stiffness of the problem, making it more reliable for problems like the Van der Pol equation with high nonlinearity.

Code:-

```
clc; clear; % Clear the command window and workspace

%% Define the Van der Pol Equation
mu = 1000; % Set the value of the nonlinearity parameter
func = @(t, y) [y(2); mu*(1 - y(1)^2)*y(2) - y(1)]; % Van der Pol ODE
y0 = [2; 0]; % Initial conditions
tspan = [0 3000]; % Time span for the solution

%% ode45 Implementation
[t_ode45, y_ode45] = ode45(func, tspan, y0);
% Display results
y1 = y_ode45(end,1);
y2 = y_ode45(end,2);
fprintf('Final ode45 y1: %.6f\n', y1);
fprintf('Final ode45 y2: %.6f\n', y2);
% Plot results for ode45
figure;
subplot(2, 1, 1);
plot(t_ode45, y_ode45(:, 1), 'b');
legend('ode45');
title('y1 vs Time');
xlabel('Time');
ylabel('y1');
subplot(2, 1, 2);
plot(t_ode45, y_ode45(:, 2), 'b');
legend('ode45');
title('y2 vs Time');
xlabel('Time');
ylabel('y2');

%% RK4 Implementation
h_rk4 = 0.1; % Step size for RK4
t_rk4 = tspan(1):h_rk4:tspan(2);
y_rk4 = zeros(length(t_rk4), 2);
y_rk4(1, :) = y0;
for i = 1:length(t_rk4)-1
   k1 = func(t_rk4(i), y_rk4(i, :)');
   k2 = func(t_rk4(i) + h_rk4/2, y_rk4(i, :) + h_rk4/2 * k1);
   k3 = func(t_rk4(i) + h_rk4/2, y_rk4(i, :) + h_rk4/2 * k2);
   k4 = func(t_rk4(i) + h_rk4, y_rk4(i, :) + h_rk4 * k3);
   y_rk4(i+1, :) = y_rk4(i, :) + (h_rk4/6) * (k1 + 2*k2 + 2*k3 + k4)';
end
% Display RK4 results
y1 = y_rk4(end,1);
y2 = y_rk4(end,2);
fprintf('Final RK4 y1: %.6f\n', y1);
fprintf('Final RK4 y2: %.6f\n', y2);
% Plot RK4 results
figure;
subplot(2, 1, 1);
plot(t_rk4, y_rk4(:, 1), 'r--');
legend('RK4');
title('y1 vs Time');
xlabel('Time');
```

```
ylabel('y1');
subplot(2, 1, 2);
plot(t_rk4, y_rk4(:, 2), 'r--');
legend('RK4');
title('y2 vs Time');
xlabel('Time');
ylabel('y2');

%% Forward Euler Implementation
h_fe = 0.01; % Step size for Forward Euler
[t_fe, y_fe] = forward_euler(func, tspan, y0, h_fe);
% Plot Forward Euler results
figure;
subplot(2, 1, 1);
plot(t_fe, y_fe(:,1), 'g-');
title('Forward Euler: y1 vs time');
xlabel('Time');
ylabel('y1');
grid on;
subplot(2, 1, 2);
plot(t_fe, y_fe(:,2), 'g-');
title('Forward Euler: y2 vs time');
xlabel('Time');
ylabel('y2');
grid on;
%% Backward Euler Implementation (Implicit method using Newton's method)
h_be = 0.01; % Step size for Backward Euler
[t_be, y_be] = backward_euler(func, tspan, y0, h_be, mu); % Pass mu as argument
% Plot Backward Euler results
figure;
subplot(2, 1, 1);
plot(t_be, y_be(:,1), 'm-');
title('Backward Euler: y1 vs time');
xlabel('Time');
ylabel('y1');
grid on;
subplot(2, 1, 2);
plot(t_be, y_be(:,2), 'm-');
title('Backward Euler: y2 vs time');
xlabel('Time');
```

```
ylabel('y2');
grid on;
%% ODE15s Implementation (stiff solver)
options = odeset('RelTol', 1e-6, 'AbsTol', 1e-8); % Set solver options for ODE15s
[t_ode15s, y_ode15s] = ode15s(func, tspan, y0, options);
% Display ode15s results
y1 = y_ode15s(end, 1);
y2 = y_ode15s(end, 2);
fprintf('Final ode15s y1: %.6f\n', y1);
fprintf('Final ode15s y2: %.6f\n', y2);
% Plot ODE15s results
figure;
subplot(2, 1, 1);
plot(t_ode15s, y_ode15s(:, 1));
title('ODE15s');
xlabel('Time');
ylabel('y1');
grid on;
subplot(2, 1, 2);
plot(t_ode15s, y_ode15s(:, 2));
title('ODE15s');
xlabel('Time');
ylabel('y2');
grid on;


%% Function Definitions (Move to the end)
function [t, y] = forward_euler(f, tspan, y0, h)
   t = tspan(1):h:tspan(2);
   n = length(t);
   y = zeros(n, length(y0));
   y(1,:) = y0';
   for i = 1:(n-1)
      y(i+1,:) = y(i,:) + h*f(t(i), y(i,:)')';
   end
end

function [t, y] = backward_euler(f, tspan, y0, h, mu)
   t = tspan(1):h:tspan(2);
   n = length(t);
```

```
    y = zeros(n, length(y0));
    y(1,:) = y0';
    for i = 1:(n-1)
        % Initial guess for Newton iteration
        y_guess = y(i,:)';
        % Newton iteration
        for iter = 1:10
            % System equations
            F = y_guess - y(i,:)' - h*[y_guess(2); mu*(1-y_guess(1)^2)*y_guess(2) - y_guess(1)];
            % Jacobian matrix
            J = eye(2) - h*[0, 1; -1 - 2000*y_guess(1)*y_guess(2), mu*(1-y_guess(1)^2)];
            % Newton update
            delta = -J\F;
            y_guess = y_guess + delta;
            % Check convergence
            if norm(delta) < 1e-6
                break;
            end
        end
        y(i+1,:) = y_guess';
    end
end
```

Conclusion:-

- **RK4's Failure for Stiff Problems**: As demonstrated, RK4 fails to handle stiff systems, such as the Van der Pol equation with large μ\muμ, because it is an explicit method. For such systems, implicit solvers like backward Euler or specialized solvers like `ode15s` should be used.
- **Importance of Choosing the Right Solver**: The performance of a numerical solver depends on the nature of the differential equation. For stiff systems, implicit methods provide better stability, while for non-stiff systems, explicit methods like RK4 and `ode45` perform well.
- **Recommendation for Solving Stiff Equations**: For stiff systems, it is crucial to use implicit methods or specialized solvers designed to handle stiffness, such as `ode15s` in MATLAB. These solvers adapt the step size to the problem's requirements, ensuring stable solutions even for long time intervals.