Scilab for Chemical Engineers

TSEC - Online Certificate Course

SAMPLE PROBLEMS

Aditya Ganesh

Date: 08 - 02 - 2025

1 Functions

1.1 Basic Functions

1.1.1 Temperature Conversion:

Write a function `celsiusToFahrenheit(T_C)` that converts a temperature from Celsius to Fahrenheit. Additionally, write a function `fahrenheitToCelsius(T_F)` that converts from Fahrenheit to Celsius.

```
function T_F = celsiusToFahrenheit(T_C)
    T_F = (T_C * 9/5) + 32;
endfunction

function T_C = fahrenheitToCelsius(T_F)
    T_C = (T_F - 32) * 5/9;
endfunction
```

1.1.2 Pressure Calculation:

Write a function `pressureIdealGas(n, V, T)` that calculates the pressure of an ideal gas using the Ideal Gas Law: $P = \frac{nRT}{V}$. Assume R (the gas constant) is 8.314 J/(mol·K).

```
function P = pressureIdealGas(n, V, T)
    R = 8.314; // J/(mol K)
    P = (n .* R .* T) ./ V;
```

```
endfunction
```

1.2 Intermediate Functions

1.2.1 Reactor Volume Calculation:

Write a function `CSTRVolume(F, k, X)` to calculate the required volume of a Continuous Stirred-Tank Reactor (CSTR) given the feed flow rate F, reaction rate constant k, and conversion X using the formula $V = \frac{F \cdot (X - X_0)}{k \cdot (1 - X)}$.

```
function V = CSTRVolume(F, k, X)
    X0 = 0; // Assuming initial conversion is zero
    V = (F .* (X - X0)) ./ (k .* (1 - X));
endfunction
```

1.2.2 Heat Exchanger Area:

Write a function `heatExchangerArea(Q, U, LMTD)` to calculate the required heat exchanger area using the formula $A = \frac{Q}{U \cdot LMTD}$, where Q is the heat duty, U is the overall heat transfer coefficient, and $LMTD$ is the logarithmic mean temperature difference.

```
function A = heatExchangerArea(Q, U, LMTD)
    A = Q ./ (U .* LMTD);
endfunction
```

1.2.3 Boiling Point Elevation:

Write a function `boilingPointElevation(M, k_b, m)` that calculates the boiling point elevation of a solution given the molality of the solution m, the ebullioscopic constant k_b, and the molar mass M of the solute.

```
function deltaT_b = boilingPointElevation(M, k_b, m)
    deltaT_b = k_b .* m ./ M;
endfunction
```

1.3 Advanced Functions

1.3.1 Reaction Rate Calculation:

Write a function `reactionRate(C, k, order)` that calculates the rate of a chemical reaction given the concentration C, the reaction rate constant k, and the order of the reaction.

```
function rate = reactionRate(C, k, order)
    rate = k .* C.^order;
endfunction
```

1.3.2 Batch Reactor Simulation:

Write a function `batchReactor(C0, k, t)` that simulates the concentration C of a reactant in a batch reactor over time t using first-order kinetics and plots the concentration vs. time.

```
function C = batchReactor(C0, k, t)
    C = C0 .* exp(-k .* t);
endfunction

% // Example usage
% t = 0:0.1:10; // Time array
% C0 = 1; // Initial concentration
% k = 0.1; // Reaction rate constant
% C = batchReactor(C0, k, t);
% plot(t, C);
% xlabel('Time');
% ylabel('Concentration');
% title('Batch Reactor Simulation');
%
```

1.3.3 Distillation Column Calculation:

Write a function `distillationColumn(N, xD, xB, xF)` that calculates the number of theoretical stages N in a distillation column using the Fenske equation, given the distillate mole fraction xD, bottoms mole fraction xB, and feed mole fraction xF.

```
function N = distillationColumn(xD, xB, xF)
    alpha = 2.0; // Assumed relative volatility
    N = log((xD ./ (1 - xD)) * ((1 - xB) ./ xB)) ./ log(alpha
        );
endfunction
```

2 Numerical Methods

2.1 Root finding algorithms

2.1.1 Bisection Method

Find the root of the function $f(x) = x^3 - 6x^2 + 11x - 6.1$ within the interval [2, 4] using the Bisection Method. (Use tolerance value of 10^{-6})

```
function result = bisection_method(f,a,b,tol)
    if b < a then
        a = b;
        b = a;
    end
    if f(a)*f(b) >= 0 then
        result = "Error";
    else
        while (b - a)  > tol
            midpoint = (a+b) / 2.0;
            if f(midpoint) == 0 then
                break;
            end
            if f(a)*f(midpoint) < 0.0 then
                b = midpoint;
            else
                a = midpoint;
            end
            // disp(midpoint);
        end
        result = midpoint;
```

```
        end
end

// Define the function
deff('y=f(x)','y = x.^3 -6.*x.^2 + 11.*x - 6.1');

// Intervals and tolerance
a = 2;
b = 4;
tol = 1e-06;

// Calculating the root
root = bisection_method(f,a,b,tol);
disp(root);
```

2.1.2 Newton-Raphson Method

Find the root of the function $f(x) = \cos(x) - x$ with an initial guess of $x_0 = 0.5$ using the Newton-Raphson Method.

```
function root = newtonRaphson(f, f_prime, x0, tol)
    x = x0;
    while abs(f(x)) > tol
        x = x - f(x) / f_prime(x);
    end
    root = x;
end

// Define the function and its derivative
deff('y = f(x)', 'y = cos(x) - x');
deff('y_prime = f_prime(x)', 'y_prime = -sin(x) - 1');

// Find the root
tol = 1e-6;
x0 = 0.5;
root = newtonRaphson(f, f_prime, x0, tol);
disp(root);
```

2.1.3 Secant Method

Find the root of the function $f(x) = e^x - 3x^2$ with initial guesses $x_0 = 0$ and $x_1 = 1$ using the Secant Method.

```
function root = secant(f, x0, x1, tol)
    x_prev = x0;
    x_curr = x1;

    while abs(f(x_curr)) > tol
        x_temp = x_curr - f(x_curr) * (x_curr - x_prev) / (f(
           x_curr) - f(x_prev));
        x_prev = x_curr;
        x_curr = x_temp;
    end

    root = x_curr;
end

// Define the function
deff('y = f(x)', 'y = exp(x) - 3*x.^2');

// Find the root
tol = 1e-6;
x0 = 0;
x1 = 1;
root = secant(f, x0, x1, tol);
disp(root);
```

2.1.4 Fixed-Point Iteration

Find the root of the function $f(x) = x^3 - 2x - 5$ by rewriting it as $g(x) = \sqrt[3]{2x + 5}$ and using Fixed-Point Iteration with an initial guess of $x_0 = 2.0$.

```
function root = fixedPoint(g, x0, tol)
    x = x0;
    while abs(x - g(x)) > tol
        x = g(x);
    end
    root = x;
end

// Define the function g(x)
deff('y = g(x)', 'y = nthroot(2*x + 5, 3)');

// Find the root
tol = 1e-6;
x0 = 2.0;
root = fixedPoint(g, x0, tol);
disp(root);
```

2.2 System of linear equations

2.2.1 Gauss Elimination

Solve the following system of linear equations using Gauss elimination

$$\begin{bmatrix} 2 & -1 & 1 \\ 3 & 3 & 9 \\ 3 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 5 \end{bmatrix} \tag{1}$$

```
function [A, b, x] = GaussElimination(A, b)
    n = size(A, 1);

    // Forward elimination
    for k = 1:n-1
        for i = k+1:n
            if A(i,k) ~= 0
                factor = A(i,k) / A(k,k);
                A(i,k:n) = A(i,k:n) - factor * A(k,k:n);
```

```
                b(i) = b(i) - factor * b(k);
            end
        end
    end

    // Back substitution
    x = zeros(n, 1);
    x(n) = b(n) / A(n,n);
    for i = n-1:-1:1
        x(i) = (b(i) - A(i,i+1:n) * x(i+1:n)) / A(i,i);
    end
end
```

2.2.2 Gauss-Seidel

Solve the following system of linear equations with Gauss-Seidel method

$$\begin{bmatrix} 4 & 1 & 2 \\ 3 & 5 & 1 \\ 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 3 \end{bmatrix} \tag{2}$$

```
function x = gaussSeidel(A, b, tol, max_iter)
    n = size(A, 1);
    x = zeros(n, 1);
    x_new = x;
    iter = 0;

    while iter < max_iter
        iter = iter + 1;
        for i = 1:n
            sum = 0;
            for j = 1:n
                if i ~= j
                    sum = sum + A(i, j) * x_new(j);
                end
            end
            x_new(i) = (b(i) - sum) / A(i, i);
```

```
        end

        if norm(x_new - x) < tol
            break;
        end

        x = x_new;
    end

    if iter == max_iter
        disp("Maximum iterations reached without convergence.");
    else
        disp("Converged in " + string(iter) + " iterations.");
    end
end
```