Implement AWE

Name: Mohammed AL Shuaili

Date: 31/1/2025

Objectives:

1. Find a pattern in matrix A to relate it with the number of sections in the RLC ladder
2. Validate AWE using the theory method (as in

$$y(t) = Ce^{at}B$$

3. Find the unit step response by doing one step and then the unit step as in section 5
4. Code the unit step response
5. Look at the Y-parameters and try to implement it.
6. complex frequency hopping will be the novel feature.

AWE is used to find the impulse response as follows:

$$h(t) = k_0\delta(t) + k_1 e^{p_1 t} + \ldots + k_n e^{p_n t}$$

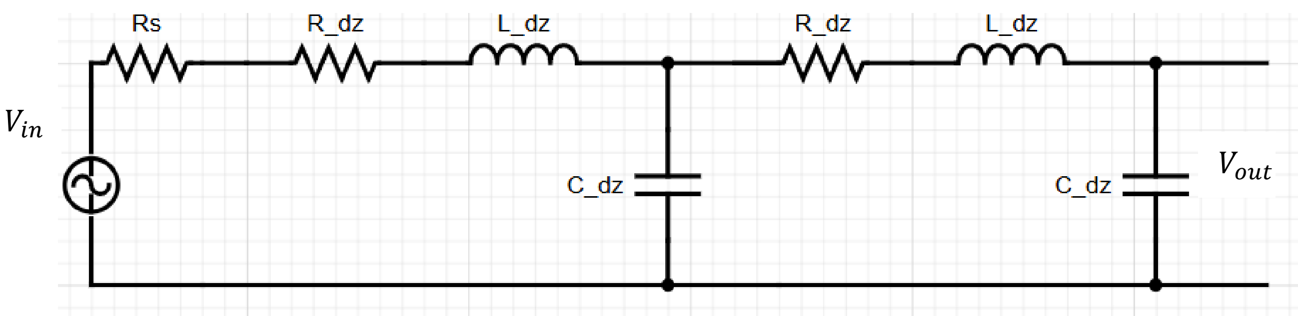
The following code implements AWE with up to q-th approximation where q is the order:

```
clear all
clc
% Input state-space matrices
A = [-2 1 0 0; 1 -2 1 0; 0 1 -2 1; 0 0 1 -1];
B = [1; 0; 0; 0];
C = [1; 0; 0; 0];

% Determine the order of the system
q = length(B);

% Compute moments
num_moments = 2 * q;
moments = zeros(1, num_moments);
for i = 1:num_moments
    moments(i) = -transpose(C) * (A^(-i)) * B;
end

% Generalized approximation for all orders
for approx_order = 1:q
    fprintf('Case %d:\n', approx_order);

    % Construct the moment matrix
    moment_matrix = zeros(approx_order);
    Vector_c = -moments(approx_order+1:2*approx_order)';

    for i = 1:approx_order
        moment_matrix(i, :) = moments(i:i+approx_order-1);
    end
    % find b matrix (deno coef)
    b_matrix = inv(moment_matrix)*Vector_c;

    %find the ploes
    poles = roots([transpose(b_matrix) ,1]);

    % determine residuses
    % form the V matrix
    V = zeros(approx_order);
    for i = 1:approx_order
        for j = 1:approx_order
```

```
                V(i, j) = 1/poles(j)^(i-1);
            end
        end
        % form the A matrix
        A_diag = diag(1 ./ poles);
        r_moments = moments(1:approx_order); % a helper matrix
        % find the residuse
        residues = -1*inv(A_diag)* inv(V)* transpose(r_moments);
        %set a value for t
        %t=0:10;
        t = 0:0.1:5;
        %form the impulse response
        h =0;
        for i = 1:approx_order
            h = h + residues(i) * exp(poles(i) * t);
        end
        % plot the output
        figure(approx_order);
        title(['Apprximation of order',num2str(approx_order)]);
        plot(t,h);
        xlabel('Time (\mus)');
        ylabel('V Load (Volts)');
        grid on
    end

```

- Now, generalise the code to implement AWE with any number of sections for the RLC ladder.

First, consider the open voltage RLC ladder for the transmission line with N=2 as follows:



From the circuit, we can say that:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} i_1 \\ v_1 \\ i_2 \\ v_o \end{bmatrix}, \quad u = v_{in}, \quad y = v_{out} \tag{3}$$

Matrix A:

$$A = \begin{bmatrix} \frac{-R_s + R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 \\ \frac{1}{C} & 0 & -\frac{1}{C} & 0 \\ 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} \\ 0 & 0 & \frac{1}{C} & 0 \end{bmatrix}$$

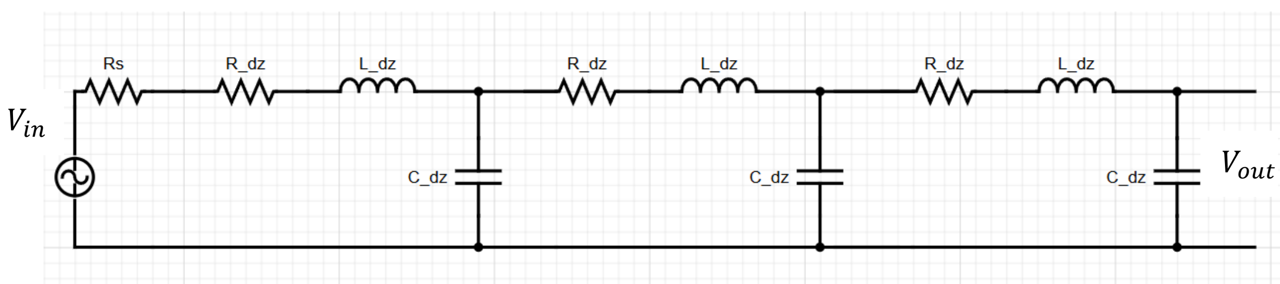
Matix B:

$$B = \begin{bmatrix} \frac{1}{L_{dz}} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Matrix C:

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

Second , Let's consider the following circuit (RLC ladder with N = 3 and open voltage) to find a pattern where we can link the number of sections to the state space model:



$$v_{in} = (R_s + R_{dz})i_1 + L_{dz}\frac{di_1}{dt} + v_1$$

$$v_1 = R_{dz}i_2 + L_{dz}\frac{di_2}{dt} + v_2$$

$$v_2 = R_{dz}i_3 + L_{dz}\frac{di_3}{dt} + v_{out}$$

$$i_1 - i_2 = C\frac{dv_1}{dt}$$

$$i_2 - i_3 = C\frac{dv_2}{dt}$$

$$i_3 = C\frac{dv_o}{dt}$$

$$v_{in} = (R_s + R_{dz})i_1 + L_{dz}\frac{di_1}{dt} + R_{dz}i_2 + L_{dz}\frac{di_2}{dt} + R_{dz}i_3 + L_{dz}\frac{di_3}{dt} + v_{out}$$

Let,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} i_1 \\ v_1 \\ i_2 \\ v_2 \\ i_3 \\ v_o \end{bmatrix}, \quad u = v_{in}, \quad y = v_{out}$$

So, each section of the transmission line contributes two states to the state-space model:

1. For $\frac{dx_1}{dt}$:

$$\frac{dx_1}{dt} = -\frac{(R_s + R_{dz})}{L_{dz}}x_1 - \frac{x_2}{L_{dz}} + \frac{v_{in}}{L_{dz}}$$

2. For $\frac{dx_2}{dt}$:

$$\frac{dx_2}{dt} = \frac{1}{C}(x_1 - x_3)$$

3. For $\frac{dx_3}{dt}$:

$$\frac{dx_3}{dt} = \frac{-R_{dz}x_3}{L_{dz}} - \frac{x_4}{L_{dz}} + x_2$$

4. For $\frac{dx_4}{dt}$:

$$\frac{dx_4}{dt} = \frac{1}{C}\ (x_3 - x_5)$$

5. For $\frac{dx_5}{dt}$:

$$\frac{dx_5}{dt} = \frac{-R_{dz}x_5}{L_{dz}} - \frac{x_6}{L_{dz}} + x_4$$

6. For $\frac{dx_6}{dt}$:

$$\frac{dx_6}{dt} = \frac{1}{C}\ x_5$$

So,

Matrix A:

$$A = \begin{bmatrix} \frac{-R_s + R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 & 0 & 0 \\ \frac{1}{C} & 0 & -\frac{1}{C} & 0 & 0 & 0 \\ 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 \\ 0 & 0 & \frac{1}{C} & 0 & -\frac{1}{C} & 0 \\ 0 & 0 & 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} \\ 0 & 0 & 0 & 0 & \frac{1}{C} & 0 \end{bmatrix}$$

Thirdly, if N =4, then

$$A = \begin{bmatrix} \frac{-R_s + R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{C} & 0 & -\frac{1}{C} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{C} & 0 & -\frac{1}{C} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{C} & 0 & -\frac{1}{C} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L_{dz}} & -\frac{R_{dz}}{L_{dz}} & -\frac{1}{L_{dz}} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{C} & 0 \end{bmatrix}$$

So, the dimensions of matrix A is N*2 and there is a pattern.

The following code generate matrix A, B and C based on N

```
clear all
clc
Rs =1;
Rdz = 5;
Ldz = 2;
Cdz =10;
N = 4;
numStates = 2 * N; % Each section has 2 states (current and voltage)
A = zeros(numStates, numStates);% Initialize A matrix
for i = 1:N
    if i == 1
        A(1, 1) = -(Rs + Rdz) / Ldz; % first term (Rs + Rdz)
    else
        A(2*i-1, 2*i-1) = -Rdz / Ldz;
    end
    if i > 1
        A(2*i-1, 2*(i-1)) = 1 / Ldz;
        A(2*i-1, 2*i) = -1 / Ldz;
    end
    if i < N
        A(2*i-1, 2*i) = -1 / Ldz;
        A(2*i, 2*i-1) = 1 / Cdz;
        A(2*i, 2*i+1) = -1 / Cdz;
    else
        A(2*i, 2*i-1) = 1 / Cdz;
    end
end
% Initialize B matrix
B = zeros(numStates, 1);
B(1) = 1 / Ldz;
%C matrix
C = zeros(1, numStates);
C(end) = 1;
A

```

This code can be put in a function and just by passing R,Rs,L, length of the line and N will generate the matrices.

- To validate AWE, we can use the following method.

The impulse response of a state space model is given by:

$$y(t) = Ce^{at}B$$

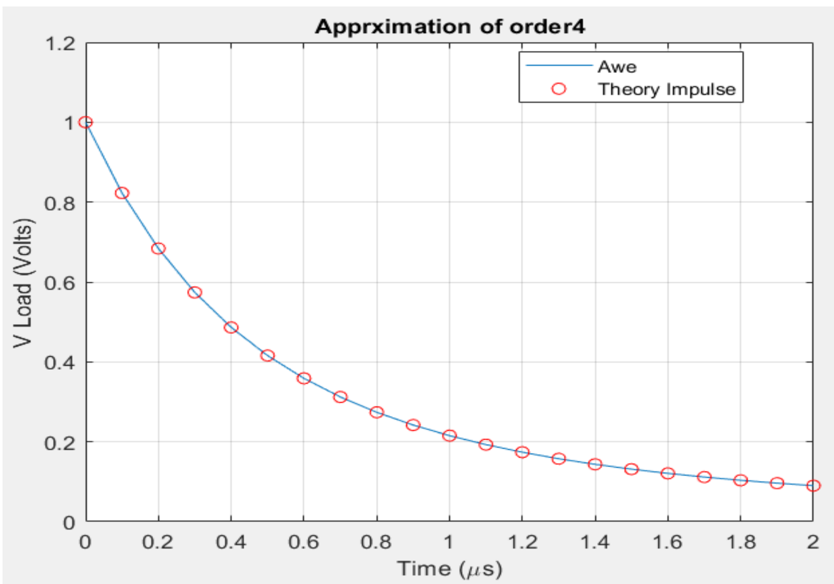
$$where, \quad e^{at} = Me^{\Omega t}M^{-1}$$

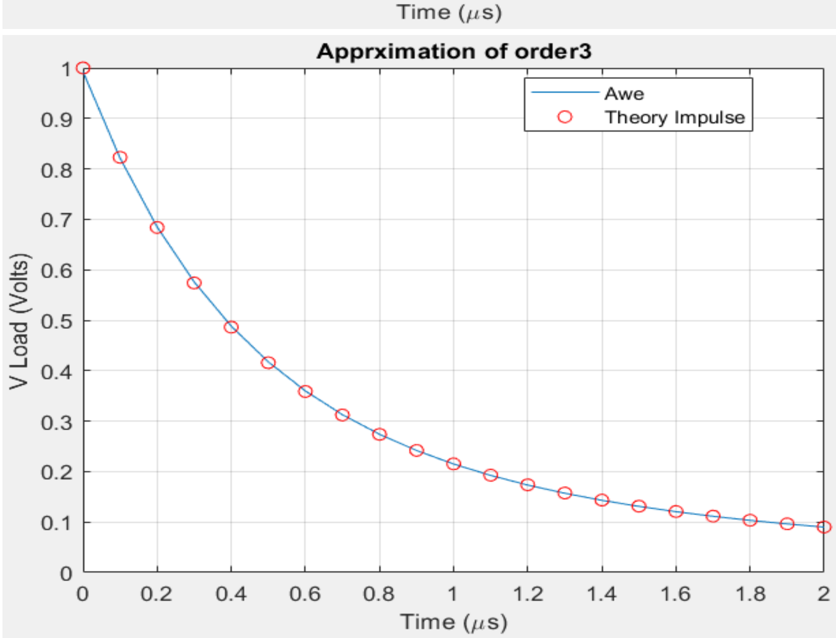
$$and\ \Omega = M^{-1}AM = \begin{bmatrix} \lambda_1 & 0 & 0 \\ \dots & \lambda_2 & 0 \\ 0 & \dots. & \lambda_n \end{bmatrix}$$

On MATLAB we can simply use the following code to obtain the impulse response.

```
clear all
clc
%find the theorytical impulse response
A = [-2 1 0 0; 1 -2 1 0; 0 1 -2 1; 0 0 1 -1];
B = [1; 0; 0; 0];
C = [1; 0; 0; 0];
C = transpose(C);
eat = @(t) expm(A.*t);
y = @(t)mtimes(mtimes(C,eat(t)),B);
t = 0:0.01:2;
y_values = arrayfun(@(t) y(t), t);
% Plot y(t)
plot(t, y_values);

```

Comparing AWE with this method, case 3 and 4 of 4th order matrix gives the best results.





- Find the unit step response by doing one step and then the unit step as in section 5

Integration Method (Section V):

To avoid computationally expensive explicit convolution, the paper employs recursive convolution based on the pole-residue representation of the transfer function. For a system described in the Laplace domain as:

$$Y(s) = \frac{k_i}{s + p_i} X(s) \ldots \ldots . \frac{d}{dt} y(t) + p_i y(t) = k_i x(t)$$

Assume *x*(*t*) is piecewise constant over each time interval $[t_n - t_{n-1}]$. Solving the above over the time interval using the recursive solution is:

$$y(t_n) = k_{\text{in}} x(t_n) + \sum_{i=1}^{q} y_i'(t_n)$$

This equation updates the output at t_n using only the previous state $y_i'(t_{n-1})$ and the input $x(t_{n-1})$ eliminating the need to store the entire history of x(t) [1].

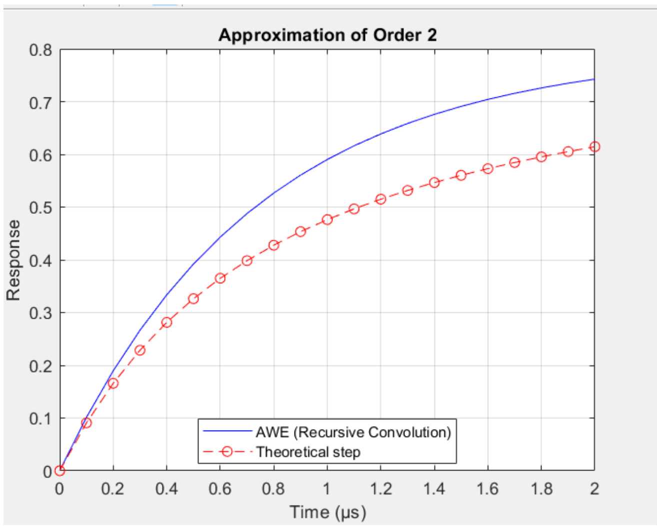
Considering the previous example, let's say the input is a unit step as x(t) = 1, and with zero initial condition. Then, the step response can be found by forming a state space and using step. Additionally, x(t) = $x(t_{n-1})$, below is an attempt to integrate the recursive method with awe to obtain the unit step.

$$k_\infty \; is \; set \; to \; zero, x(t_n) = 1$$

```
clear all
clc
% Input state-space matrices
A = [-2 1 0 0; 1 -2 1 0; 0 1 -2 1; 0 0 1 -1];
B = [1; 0; 0; 0];
C = [1; 0; 0; 0];
t = 0:0.1:2;
% impulse response
%{
eat = @(t) expm(A.*t);
y = @(t)mtimes(mtimes(transpose(C),eat(t)),B);
y_values = arrayfun(@(t) y(t), t);
%}

% step response
sys = ss(A,B,transpose(C),0);
[y_theory, t] = step(sys, t); % get the step input response using step
% Determine the order of the system
q = length(B);
% Compute moments
num_moments = 2 * q;
moments = zeros(1, num_moments);
```

```
for i = 1:num_moments
    moments(i) = -transpose(C) * (A^(-i)) * B;
end

% Generalized approximation for all orders
for approx_order = 1:q
    fprintf('Case %d:\n', approx_order);

    % Construct the moment matrix
    moment_matrix = zeros(approx_order);
    Vector_c = -moments(approx_order+1:2*approx_order)';

    for i = 1:approx_order
        moment_matrix(i, :) = moments(i:i+approx_order-1);
    end
    % find b matrix (deno coef)
    b_matrix = inv(moment_matrix)*Vector_c;

    %find the ploes
    poles = roots([transpose(b_matrix) ,1]);

    % determine residuses
    % form the V matrix
    V = zeros(approx_order);
    for i = 1:approx_order
        for j = 1:approx_order
            V(i, j) = 1/poles(j)^(i-1);
        end
    end
    % form the A matrix
    A_diag = diag(1 ./ poles);
    r_moments = moments(1:approx_order); % a helper matrix
    % find the residuse
    residues = -1*inv(A_diag)* inv(V)* transpose(r_moments);
    %form the impulse response
    h =0;
    for i = 1:approx_order
        h = h + residues(i) * exp(poles(i) * t);
    end
    % Recursive Convolution (Section V)

    y_awe = zeros(size(t)); % AWE response

    % y variables for each pole
    y = zeros(length(poles), 1);

    % Recursive convolution loop
    for n = 2:length(t)
    dt = t(n) - t(n-1); % Time step
    exp_term = exp(poles * dt); % Precompute exponentials
        for i = 1:length(poles)
            % Updat state using Eq. (15)
            y(i) = residues(i) * (1 - exp_term(i)) * 1 + exp_term(i)*y(i);
        end
        y_awe(n) = sum(y); % Total response
    end

    % Plot Results
    figure(approx_order);
    plot(t, y_awe, 'b-'); hold on;
    plot(t, y_theory, 'ro--');
    xlabel('Time (μs)');
    ylabel('Response');
    title(['Approximation of Order ', num2str(approx_order)]);
    legend('AWE (Recursive Convolution)', 'Theoretical Impulse', 'Location', 'Best');
```

```
    grid on;
    % plot the output
    %{
    figure(approx_order);
    plot(t,h);
    hold on
    plot(t, y_values,'ro');
    xlabel('Time (\mus)');
    ylabel('V Load (Volts)');
    title(['Apprximation of order',num2str(approx_order)]);
    legend('Awe', 'Theory Impulse', 'Location', 'Best');
    grid on
    %}
end
```



- Y-parameters

Will look at this after making sure that the previous code works as expected.

- Complex frequency hoping
- Final FYP report