W6

Name: Mohammed AL Shuaili

Date: 17/2/2025

Consider the exact solution that is:

Consider the following values for the impedance and an input of 30 volts:

The unit step response looks as follows:

A graph on a white background

AI-generated content may be incorrect.

we can generate Y parameter models out of this, consider 100 points for until 9e5. And 100 points for the output TL.

To simplify this, consider 2 models of Y parameters and follow the same steps for as many Y models.

The following code will implement this.

clear

clc

f = 9e5;

f = linspace(0,f,100+1);

w = 2\*pi\*f(2:end);

s = i \*w;

vo =1./(cosh(400.\*(0 + 1e-10.\*s).^(1/2).\*(0.1 + 2.5e-7.\*s).^(1/2)));

N=1; % number of section basicly 100/N

H\_total = @(s)0;

section\_size = ceil(length(vo)/N); % Points per section (except last)

for i = 1:N

start\_idx = (i-1)\*section\_size + 1;

end\_idx = min(i\*section\_size, length(w));

seg\_idx = start\_idx:end\_idx;

w\_i = w(seg\_idx);

s\_i = 1i \* w\_i;

vo\_i = vo(seg\_idx);

H\_prev\_eval = H\_total(s\_i);

% Calculate residual response

vo\_residual = vo\_i - H\_prev\_eval;

% Fit new model to residual

Hi = generate\_yp2(real(vo\_residual), imag(vo\_residual), w\_i);

% Add to total transfer function

H\_total = @(s) H\_total(s) + Hi(s);

end

H = @(s) H\_total(s) \* 30 ./ s;

v =@(s)30./(s.\*cosh(400.\*(0 + 1e-10.\*s).^(1/2).\*(0.1 + 2.5e-7.\*s).^(1/2)));

[y,t]=niltcv(H,50e-6,'pt1');

[y1,t1]=niltcv(v,50e-6,'pt1');

RMSE = sqrt(sum((y-y1).^2)/length(y1));

plot(t,y,t1,y1)

grid on

xlabel('time s')

ylabel('Vo')

legend('approximated','exact');

|  |  |
| --- | --- |
| N (number of Y models) | RMSE |
| 1 | 25.5234 |
| 2 | 27.8445 |
| 3 | 4.9876 |
| 4 | 1.8179 |
| 5 | 8.3383 |
| 6 | 3.6791 |
| 7 | 3.1194 |
| 8 | 3.5229 |
| 9 | 1.6736 |
| 10 | 4.9612 |
| 11 | Code error |

A graph of a graph

AI-generated content may be incorrect.

A graph on a white surface

AI-generated content may be incorrect.

We can see that the accuracy depends on the number of models used but not at all times as there are other things that could affect this. This was done using NILTcv,

* Implementation using AWE.

So in this section AWE is used to get the step response and H(s) where :

The frequency passed to AWE is the beginning of each model. The following code is an example on only using one model.

clear

clc

% generate 100 points.

f = 9e5;

f = linspace(0,f,100);

w = 2\*pi\*f;

s = i \*w;

vo =1./(cosh(400.\*(0 + 1e-10.\*s).^(1/2).\*(0.1 + 2.5e-7.\*s).^(1/2)));

v =@(s)30./(s.\*cosh(400.\*(0 + 1e-10.\*s).^(1/2).\*(0.1 + 2.5e-7.\*s).^(1/2)));

%H is as H = @(s) a1s+a0/s^2+b1s+b0;

[H,num,deno] = generate\_yp2(real(vo(1:13)),imag(vo(1:24)),w(1:13));

[A,B,C,D] = create\_state\_space(num,deno);

%HAWE is Hs= @(s) resdue/s-pole + ...;30 is the inout, 50e-6 is t for plot

[h\_impulse,HAWE, y, t] = AWE(A,B,C,D,w(1),30,50e-6);

[y1,t1]=niltcv(v,50e-6,'pt1');

RMSE = sqrt(sum((y-y1).^2)/length(y1));

plot(t,y,t1,y1);

grid on

xlabel('time s')

ylabel('Vo')

legend('approximated','exact');

abs(RMSE)

in this code, 100 points are generated for the exact solution and the first 13 (i.e, ) points were used to generate Y rational approximation using code 1. Then it’s converted to state space mode using code 2 in the appendix.

AWE is used with w(1) (i.e, ). Code 3. The following unit step response was obtained.

A graph of a graph

AI-generated content may be incorrect.

With RMSE = 2.6373.

Now consider 2 Y rational approximations.

Code 1:

function [H\_impulse,num,deno]=generate\_yp2(realV,imagV,wo)

Yr = realV; % Real part of Y11

Yi = imagV; % Imaginary part of Y11

w = wo;

A = [];

C = [];

% Loop through each frequency point to construct A and C

A = [];

C = [];

% Loop through each frequency point to construct A and C

for k = 1:length(w)

wk = w(k);

Yr\_k = Yr(k);

Yi\_k = Yi(k);

% Construct rows for A and C

A\_row1 = [-1, Yr\_k, 0, -wk\*Yi\_k]; % Real part

A\_row2 = [0, Yi\_k, -wk, wk\*Yr\_k]; % Imaginary part

% Append to A

A = [A; A\_row1; A\_row2];

% C

C\_row1 = wk^2 \* Yr\_k; % Real part

C\_row2 = wk^2 \* Yi\_k; % Imaginary part

% Append to C

C = [C; C\_row1; C\_row2];

end

% Solve for B = [a0; b0; a1; b1]

B = A \ C;

% get cof

a0 = B(1);

b0 = B(2);

a1 = B(3);

b1 = B(4);

num = [a1,a0];

deno = [1,b1,b0];

% generated H

H\_impulse =@(s) (a1\*s+a0)./(s.^2+b1\*s+b0);

end

Code 2:

function [A,B,C,D]= create\_state\_space(nem,deno)

N = nem;

D = deno;

% Perform polynomial division to make it strictly proper

[Q, R] = deconv(N, D);

% check leading coefficient (assumed to be 1)

if D(1) ~= 1

D=D/D(1);

N = N/D(1);

[Q, R] = deconv(N, D);

end

% Extract coefficients for state-space representation

g = D(2:end); % Exclude leading coefficient ( g terms )

% f terms

if R(1) ==0

f=R(2:end);

else

f = R;

end

% Construct state-space matrices

n = length(g); % Order of system

A = [zeros(n-1,1), eye(n-1); -flip(g)];

B = [zeros(n-1,1); 1];

C = flip(f);

D = Q;

end

Code 3:

function [h\_impulse,h\_s, y\_step, t] = AWE(A,B,C,D,w,input,time)

t = linspace(0,time,250);

q = length(B);

num\_moments = 2 \* q;

s0=1i\*w;

moments = zeros(1, num\_moments);

[r,c]=size(C);

if r~=1

C= C';

end

for k = 1:num\_moments

moments(k) = (-1)^(k-1) \* C \* (s0 \* eye(size(A)) - A)^-(k) \* B;

end

moments(1)=moments(1)+D;

approx\_order = length(B);

% 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

% Solve for denominator coefficients

b\_matrix = moment\_matrix^-1 \* Vector\_c;

% Compute poles

poles = roots([b\_matrix', 1]);

% Compute residues

V = zeros(approx\_order);

for i = 1:approx\_order

for j = 1:approx\_order

V(i, j) = 1 / poles(j)^(i-1);

end

end

A\_diag = diag(1 ./ poles);

r\_moments = moments(1:approx\_order);

residues = -1 \* (A\_diag \ (V \ r\_moments'));

% Impulse response

h\_impulse = zeros(size(t));

for i = 1:approx\_order

h\_impulse = h\_impulse + residues(i) \* exp(poles(i) \* t);

end

h\_s =@(s) 0;

for i = 1:length(poles)

h\_s =@(s) h\_s(s) + residues(i)./(s-poles(i));

end

% Step response using recursive convolution

y\_step = zeros(size(t));

y = zeros(length(poles), 1);

for n = 2:length(t)

dt = t(n) - t(n-1);

exp\_term = exp(poles \* dt);

for i = 1:length(poles)

y(i) = residues(i) \* (1 - exp\_term(i))/(-poles(i)) \* input + exp\_term(i) \* y(i);

end

y\_step(n) = sum(y);

end

end