matlab-rico

the docs

Rico A. R. Picone
Department of Mechanical Engineering
Saint Martin's University

Contents

Make sure the `rico` directory is in your Matlab path.

```
addpath('../rico')
```

1 phasor docs

The following is the source code.

```
classdef phasor
    % phasor   Phasor representation for impedance analysis
    % The phasor class enables impedance analysis seamlessly.
    % Phasor instances follow the rules of phasor arithmetic
    % using the usual Matlab operators +, *, /, etc!
    % The phasor can be defined in either polar 'pol' or
    % rectangular/Cartesian 'rec' form.
    % The phasor can be accessed in either form, too!
    % Even symbolic phasor objects can be constructed.
    %
    % Usage:
    %   - Construct:
    %       p1 = phasor(coords,c1,c2)
    %       where:
    %       coords is either 'pol' or 'rec'
    %       c1 is either the radius or real part
    %       c2 is either the angle or imaginary part
    %   - Access:
    %       p1.pol % polar form of p1
    %       p1.rec % rectangular form of p1
    %   - Operate:
    %       All the usual arithmetic operators work, like:
    %       p1+p2 % sum phasors
    %       p1-p2 % subtract phasors
    %       p1*p2 % multiply phasors
    %       p1/p2 % divide phasors
    %       a*p1 % scale a phasor
    %
    % phasor Properties:
    %    pol - polar form representation [radius,angle]
    %    rec - rectangular form as complex number
    properties
        pol % polar form representation [radius,angle]
        rec % rectangular form as complex number
    end
```

```
    methods
        function obj = phasor(coords,c1,c2)
            if nargin > 0
                switch coords
                    case 'pol'
                        obj.pol = [c1,c2]; % r, theta
                        obj.rec = pol2rec(obj);
                    case 'rec'
                        obj.rec = c1+1j*c2; % x + j y
                        obj.pol = rec2pol(obj);
                    otherwise
                        error('Either pol or rec coordinates')
                end

            end
        end
        function c_rec = pol2rec(obj)
            c_rec = obj.pol(1)*cos(obj.pol(2)) + 1j*obj.pol(1)*sin(obj.pol(2));
        end
        function c_pol = rec2pol(obj)
            c_pol =  [sqrt(real(obj.rec)^2 + imag(obj.rec)^2), ...
                atan2(imag(obj.rec),real(obj.rec))];
        end
        function out = plus(obj1,obj2)
            sum = obj1.rec + obj2.rec;
            out = phasor('rec',real(sum),imag(sum));
        end
        function out = minus(obj1,obj2)
            sum = obj1.rec - obj2.rec;
            out = phasor('rec',real(sum),imag(sum));
        end
        function out = uminus(obj1)
            sum = -obj1.rec;
            out = phasor('rec',real(sum),imag(sum));
        end
        function out = mtimes(obj1,obj2)
            if isa(obj1,'phasor')
                if isa(obj2,'phasor')
                    pro = [obj1.pol(1)*obj2.pol(1), ...
                        obj1.pol(2)+obj2.pol(2)];
                else
                    if isreal(obj2)
                        pro = [obj1.pol(1)*obj2, ...
                            obj1.pol(2)];
                    else
```

```
                        error('phasor scalar multiplication supports reals only')
                    end
                end
            elseif isa(obj2,'phasor')
                if isreal(obj1)
                    pro = [obj2.pol(1)*obj1, ...
                        obj2.pol(2)];
                else
                    error('phasor scalar multiplication supports reals only')
                end
            end
            out = phasor('pol',pro(1),pro(2));
        end
        function out = times(obj1,obj2)
            out = mtimes(obj1,obj2);
        end
        function out = mrdivide(obj1,obj2)
            pro = [obj1.pol(1)/obj2.pol(1), ...
                obj1.pol(2)-obj2.pol(2)];
            out = phasor('pol',pro(1),pro(2));
        end
        function out = rdivide(obj1,obj2)
            out = mrdivide(obj1,obj2);
        end
        function out = mpower(obj1,pow)
            pro = [obj1.pol(1)^pow, ...
                obj1.pol(2)*pow];
            out = phasor('pol',pro(1),pro(2));
        end
        function out = power(obj1,pow)
            out = mpower(obj1,pow);
        end
    end
end
```

2 tf_factor docs

The following is the source code.

```
function out = tf_factor(sys)
% TF_FACTOR  factors a transfer function TF object
%   SYS_ARRAY = TF_FACTOR(SYS) factors SYS into
%   constant, real pole/zero, and
```

```
%   conjugate pole/zero pair sub-TF models.
%   It returns a TF model array.
%   The last entry is the appropriate gain.
%   The product of entries of the model array
%   should equal sys.
%
%   Dependencies:
%     - matlab-rico functions
%       - POLE
%       - ZERO
%     - toolboxes
%       - Control Systems
%
%   Example:
%
%   sys=tf(...
%     [-0.64 -0.4101 0.00783],...
%     [1 1.489 0.7681 0.09455 0.0424 .7]...
%   );
%   tf_factor(sys)
%
%   source: https://github.com/ricopicone/matlab-rico
%
%   See also TF, STACK.

if ~isa(sys,'tf')
  sys = tf(sys);
end

% extract poles and zeros
poles=pole(sys);
zeros=zero(sys);

% make sure they're in coupled pairs
poles_cplx = cplxpair(poles);
zeros_cplx = cplxpair(zeros);

% loop through and extract sub-tfs into model array, each in standard form, keepi
F = stack(1,tf(1,1)); % init model array
num_gain = sys.Num{:}(...
  find(cell2mat(sys.Num),1,'first')...
); % gain of numerator
den_gain = sys.Den{:}(...
  find(cell2mat(sys.Den),1,'first')...
); % gain of denominator
```

```
F_gain = num_gain/den_gain;
F_gain_o = F_gain ; % original gain
k=1;
jskip=0;
% poles first
for j=1:length(poles_cplx)
  if ~jskip
    if ~isreal(poles_cplx(j))
      F(:,:,k) = zpk([],[poles_cplx(j),poles_cplx(j+1)],poles_cplx(j)*poles_cplx(
      F_gain = F_gain/abs(poles_cplx(j)*poles_cplx(j+1));
      jskip=1;% skip next index
    else
      F(:,:,k) = zpk([],poles_cplx(j),abs(poles_cplx(j)));
      F_gain = F_gain/abs(poles_cplx(j));
      jskip=0;
    end
    k=k+1;
  else
    jskip=0;
  end
end
% now zeros
for j=1:length(zeros_cplx)
  if ~jskip
    if ~isreal(zeros_cplx(j))
      F(:,:,k) = zpk([zeros_cplx(j),zeros_cplx(j+1)],[],1/(zeros_cplx(j)*zeros_cp
      F_gain = F_gain*abs(zeros_cplx(j)*zeros_cplx(j+1));
      jskip=1;% skip next index
    else
      F(:,:,k) = zpk(zeros_cplx(j),[],1/abs(zeros_cplx(j)));
      F_gain = F_gain*abs(zeros_cplx(j));
      jskip=0;
    end
    k=k+1;
  else
    jskip=0;
  end
end
F(:,:,k) = F_gain; % drop the overall gain into the model array

% check by concatenation
tf_composite = 1;
for j=1:k
  tf_composite = tf_composite*F(:,:,j);
end
```

27 September 2020, 17:08:01

```
num_gain = sys.Num{:}(find(cell2mat(tf_composite.Num),1,'first'));
den_gain = sys.Den{:}(find(cell2mat(tf_composite.Den),1,'first'));
if (... % check that the factorization is correct
  isequal(num_gain/den_gain,F_gain_o) && ... % gain
  round(sum(poles),5) == round(sum(pole(tf_composite)),5) && ... % poles ... not
  round(sum(zeros),5) == round(sum(zero(tf_composite)),5) ... % zeros ... not per
)
  out = F;
else
  error('composite check failed!')
  out = 1;
end
```

2.1 Usage and examples

3 bode_multi docs

The following is the source code.

```
function [out,ax1,ax2] = bode_multi(sys_a)

syms s

if ~isa(sys_a,'tf')
  sys_a = tf(sys_a);
end
n = length(sys_a); % > 1 if system model array

out = figure;
ax1 = subplot(2,1,1);
ax2 = subplot(2,1,2);
omega_a = {.1,1};
mag_lims = [0,1];
phase_lims = [-90,0];
for i = 1:n
  sys = sys_a(1,1,i);
  [mag,phase,omega] = bode(sys);
  if omega(1) < omega_a{1}
    omega_a{1} = omega(1);
  end
  if omega(end) > omega_a{end}
    omega_a{end} = omega(end);
  end
```

```
  if mag_lims(1) < mag_lims(1)
    mag_lims(1) = mag_lims(1);
  end
  if mag_lims(2) > mag_lims(2)
    mag_lims(2) = mag_lims(2);
  end
  if phase_lims(1) < phase_lims(1)
    phase_lims(1) = phase_lims(1);
  end
  if phase_lims(2) > phase_lims(2)
    phase_lims(2) = phase_lims(2);
  end
end

olog = num2cell(cellfun(@(x) log10(x),omega_a));
omega = logspace(olog{:},100);

for i = 1:n
  sys = sys_a(1,1,i);
  [mag,phase] = bode(sys,omega);
  mag = squeeze(mag);
  phase = squeeze(phase);
  % size(omega)
  % omega = squeeze(omega);
  axes(ax1);
  hold on;
  [num,den] = tfdata(sys);
  sys_sym = poly2sym(cell2mat(num),s)/poly2sym(cell2mat(den),s);
  semilogx(...
    omega,db(mag),...
    'linewidth',1,...
    'displayname',['$',latex(sys_sym),'$']...
  );
  ylabel('|H(j\omega)|, dB')
  axes(ax2);
  hold on;
  semilogx(omega,phase,'linewidth',1);
  xlabel('frequency \omega, rad/s')
  ylabel('\angle{H(j\omega)}, deg')
  h = findobj(gcf,'type','line');
  set(h,'linewidth',1);
end
% log scale
ax1.XScale = 'log';
ax2.XScale = 'log';
```

```
% adjust limits and ticks
mag_tick_array = ax1.YLim(1):20:ax1.YLim(2);
[m0db,i0db_a] = min(abs(mag_tick_array));
i0db = i0db_a(1); % first index closest to zero
mag_tick_array = mag_tick_array-mag_tick_array(i0db);
ax1.YTick = mag_tick_array;
phase_tick_array = ax2.YLim(1):45:ax2.YLim(2);
[p0db,i0_a] = min(abs(phase_tick_array));
i0 = i0_a(1); % first index closest to zero
phase_tick_array = phase_tick_array-phase_tick_array(i0);
ax2.YTick = phase_tick_array;
% grid lines
ax1.XGrid = 'on';
ax1.YGrid = 'on';
ax2.XGrid = 'on';
ax2.YGrid = 'on';
% legend
axP = get(ax1,'Position'); % so we can keep size
l = legend(ax1,'show');
l.Interpreter = 'latex';
l.Location = 'northeastoutside';
ax1.Position = axP; % reset size
```

3.1 Usage and examples

```
sys = tf([5,3,5],[1,6,1,20])
```

```
sys =

    5 s^2 + 3 s + 5
  --------------------
  s^3 + 6 s^2 + s + 20

Continuous-time transfer function.
```

```
sys_a = tf_factor(sys)
```

```
sys_a(:,:,1,1) =

          3.155
  ---------------------
  s^2 - 0.3399 s + 3.155
```

```
sys_a(:,:,2,1) =

    6.34
  --------
  s + 6.34


sys_a(:,:,3,1) =

  s^2 + 0.6 s + 1


sys_a(:,:,4,1) =

  0.25

4x1 array of continuous-time transfer functions.
```

```
% [f,ax_mag,ax_phase] = bode_multi(G); % get axis handles
f = bode_multi(sys_a);

hgsave(f,'figures/temp');
```

4 pole docs

The following is the source code.

```
function out = pole(sys)

out = roots(cell2mat(sys.Den));
```

4.1 Usage and examples

5 zero docs

The following is the source code.

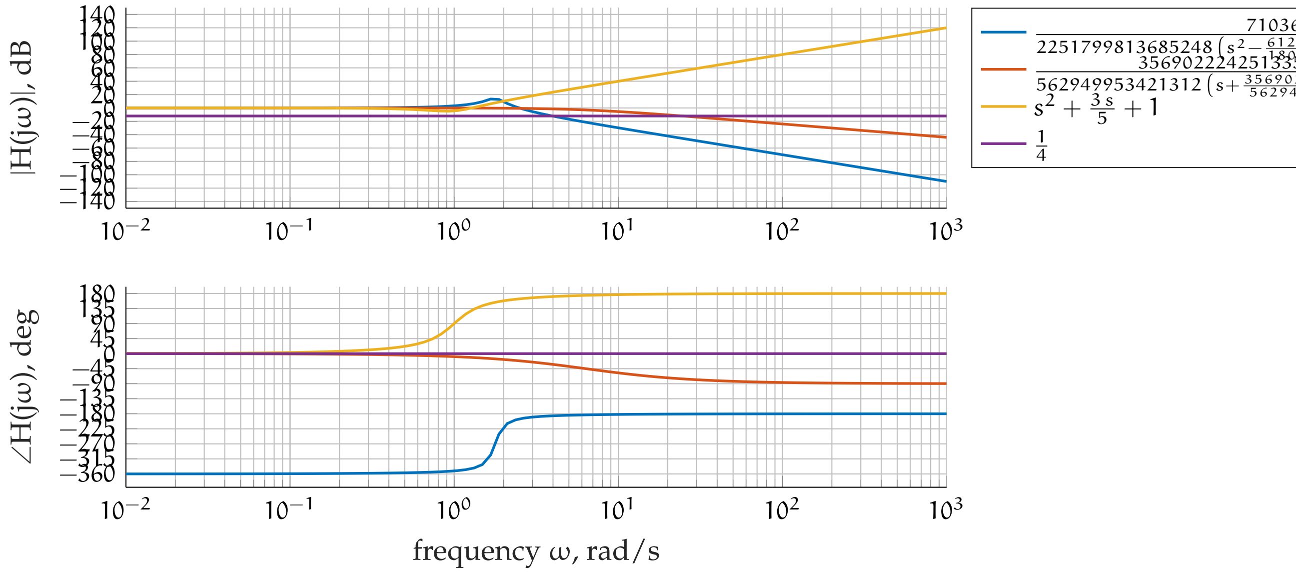

Figure 1: a bode multi example output.

```
function out = zero(sys)

out = roots(cell2mat(sys.Num));
```

5.1 Usage and examples

6 `tf2latex` docs

The following is the source code.

```
function out = tf2latex(sys)
% TF2LATEX  converts tf model to LaTeX code
%   TEXT = TF2LATEX(SYS) converts the
%   tf model SYS to LaTeX text.
%   Dependencies:
%     - toolboxes
%       - Control Systems
%       - Symbolic
```

```
syms s

num = sys.Numerator;
den = sys.Denominator;

out = latex(...
  poly2sym(...
    cell2mat(num),...
    s...
  )/...
  poly2sym(...
    cell2mat(den),...
    s...
  )...
);
```

6.1 Usage and examples

27 September 2020, 17:08:01