trans.sim Simulation

Many control systems are not in the class of those we have described, analytically: first- or second-order and without zeros. In order to evaluate their transient performance, regardless of how well they are approximated by the analytic solutions from before, we will simulate their step responses.

Matlab has several built-in and **Control Systems Toolbox** functions for analyzing the transient response of a system represented by a transfer function system model. We'll explore a few, here.

Consider, for instance, a closed-loop system with transfer function

$$F(s) = \frac{\omega_n^2(-s/z + 1)}{s^2 + 2\zeta\omega_n + \omega_n^2}, \tag{1}$$

where z is some real zero we'll move around, later; $\omega_n = 20\pi$ rad/s is the natural frequency; and $\zeta = 0.3$ is the damping ratio.

Let's explore this system's transient response. Clearly, for large negative values of z, the response should be approximately congruent with the exact analytic solutions of Lecture trans.exact. Specifically, the rise time T_r will be described by Figure exact.2, the peak time $T_p = \pi/\omega_d$, the percent overshoot will be

$$\%OS = 100 \exp\frac{-\zeta\pi}{\sqrt{1 - \zeta^2}}, \tag{2}$$

and the settling time T_s will be

$$T_s = \frac{4}{\zeta\omega_n}. \tag{3}$$

Let's compute these analytic values.

```
z = 0.3;
w_n = 20*pi;
w_d = w_n*sqrt(1-z^2);

T_r_an = 1.39/w_n  % s ... analytic, from Figure 03.3
T_p_an = pi/w_d    % s ... analytic
OS_an = 100*exp(-z*pi/sqrt(1-z^2)) % %, analytic
T_s_an = 4/(z*w_n) % s ... analytic
```

```
T_r_an =
    0.0221
T_p_an =
    0.0524
OS_an =
   37.2326
T_s_an =
    0.2122
```

Now, let's define the transfer function object.

```
z_a = -z*w_n*[1.5,3,5];
n_z = length(z_a);
F = stack(1,tf(1,1));   % init model array
for i = 1:n_z           % for each zero!
  F(:,:,i) = tf(...     % tf def transfer func object
    w_n^2*[-1/z_a(i),1],... % num. polyn. coef's
    [1,2*z*w_n,w_n^2]...    % den. polyn. coef's
  );
end
```

For a step input $u(t) = u_s(t)$ and initial value $y(0) = 0$, let's simulate. The step function would be the easiest way to solve for the step

response. However, we choose the more-general `lsim` for demonstration purposes. We must do so for each zero location z.

```
t_a = linspace(0,.3,100); % time array
u = @(t) ones(size(t)); % input for t>=0
y_0 = 0; % initial condition
y_t = NaN*ones(n_z,length(t_a)); % preallocate
for i = 1:n_z
  y_t(i,:) = lsim(F(:,:,i),u(t_a),t_a,y_0);
end
```

This total solution is shown in Fig. sim.1.

```
figure;
for i = 1:n_z
  p(i) = plot(t_a,y_t(i,:),...
  'displayname', ...
  ['z \approx ', sprintf('%0.2g',z_a(i))], ...
  'linewidth',1.5);hold on
end
hold off
xlabel('time (s)');
ylabel('step response');
grid on
l = legend(p);
```

Now, this indicates that the zero location definitely matters, and that the deviation is worse the closer the zero gets to the poles. Quantifying the response characteristics is tedious, visually, but Matlab has a built-in tool that helps: `stepinfo`.

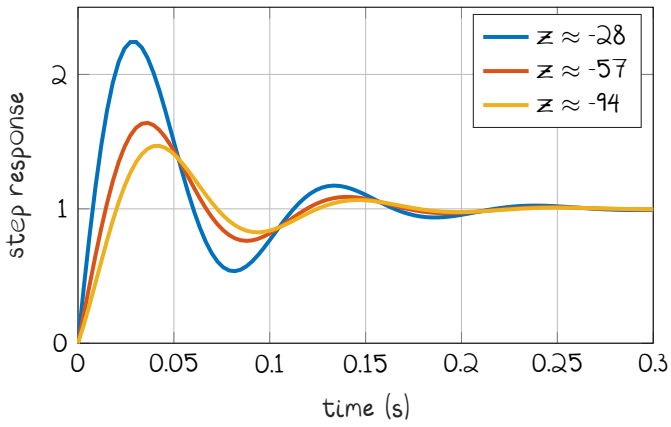

Figure sim.1: step response for different zero locations, from `lsim`.

```
for i = 1:n_z
  si(i) = stepinfo(y_t(i,:),t_a); % struct
  rt(i) = 1e3*si(i).RiseTime;
  pt(i) = 1e3*si(i).PeakTime;
  os(i) = si(i).Overshoot;
  st(i) = 1e3*si(i).SettlingTime;
  row_names{i} = sprintf('z @ %0.2g',z_a(i));
end
labels = {'T_r','T_p','OS','T_s'};
disp('Time in ms:')
disp(table(rt',pt',os',st',...
  'variablenames',labels,...
  'rownames',row_names ...
))
```

```
Time in ms:
                T_r       T_p        OS       T_s
              ______    ______    ______    ______
    z @ -28     6.09    30.303    125.86    251.63
    z @ -57   11.395    36.364    64.707    205.52
    z @ -94   15.635    42.424     47.35    203.21
```

So that zero location drastically affects the overshoot and rise time, but has relatively little effect on settling and peak times. The takeaway here is not so much this specific result, but the tools one can use to find such results and the importance of doing so.