04.6 emech.dcmtrans Transient DC motor performance

Let's begin by defining the system parameters.

```
Kt_spec = 13.7; % oz-in/A ... torque constant from spec
Kv_spec = 10.2; % V/krpm ... voltage constant from spec
Tmax_spec = 2.82; % N-m ... max (stall) torque from spec
Omax_spec = 628; % rad/s ... max speed (no load) from spec
N_oz = 0.278013851; % N/oz
m_in = 0.0254; % m/in
Kt_si = Kt_spec*N_oz*m_in; % N-m/A
rads_krpm = 1e3*2*pi/60; % (rad/s)/krpm
Kv_si = Kv_spec/rads_krpm; % V/(rad/s)
d = 2.5*m_in; % m ... flywheel diameter
thick = 1*m_in; % m ... flywheel thickness
vol = pi*(d/2)^2*thick; % flywheel volume
rho = 8000; % kg/m^3 ... flywheel density (304 stainless)
m = rho*vol; % kg ... flywheel mass
Jf = 1/2*m*(d/2)^2; % kg-m^2 ... inertia of flywheel
Jr = 56.5e-6; % kg-m^2 ... inertia of rotor
J = Jf+Jr; % kg-m^2 ... total inertia
Bm = 16.9e-6; % N-m/s^2 ... motor damping coef
Bd = 20e-6; % N-m/s^2 ... bearing damping coef
B = Bm + Bd; % N-m/s^2 ... total damping coef
R = 1.6; % Ohm ... armature resistance
L = 4.1e-3; % H ... armature inductance
TF = Kv_si; % N-m/A ... trans ratio/motor constant
```

The state-space model was derived in Lecture 04.4 emech.real. First, we construct the A, B, C, and D matrices (`a`, `b`, `c`, and `d`). Then we define a *MATLAB* LTI system model using the `ss` command.

```
a = [-B/J,TF/J;-TF/L,-R/L];
b = [0;1/L];
c = [1,0;-B,TF;-TF,-R;0,1;1,0;B,0;...
    0,R;0,1;TF,0;0,1;1,0;0,-TF;0,0;0,1];
```

```
d = [0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 0];
sys = ss(a,b,c,d);
```

Simulating the step response

The step input is widely used to characterize the transient response of a system. *MATLAB*'s `step` function conveniently simulates the step response of an LTI system model.

```
[ys_a,t_a] = step(sys);
disp([t_a(1:6),ys_a(1:6,1:4)]) % print a little
```

```
         0         0         0    1.0000         0
    0.0002    0.0018    0.0056    0.9082    0.0573
    0.0005    0.0071    0.0106    0.8245    0.1093
    0.0007    0.0155    0.0152    0.7482    0.1565
    0.0010    0.0267    0.0194    0.6786    0.1993
    0.0012    0.0405    0.0232    0.6151    0.2381
```

The vector `t_a` contains values of time and array `ys_a` contains a vector of time-series values for each output. If one would like the output for a step input $ku_s(t)$ (scaled *unit* step $u_s(t)$), by the principle of superposition for linear systems, one can scale the output by k. The outputs are plotted in Figure dcmtrans.1.

It is also interesting to inspect the power flow and energy associated with each element. Since we have simulated both the across and the through variable for each element, we can compute the instantaneous power by simply taking the product of them at each time step. Moreover, we can cumulatively compute the energy contribution of that power for each element. For energy storage elements, this is the change in energy stored or supplied; for energy dissipative elements, this is the change in energy dissipated; for source elements, this is the energy supplied or absorbed. The results are plotted in Figure dcmtrans.2.

```
P = NaN*ones(size(ys_a,1),size(ys_a,2)/2);
E = NaN*ones(size(P));
```

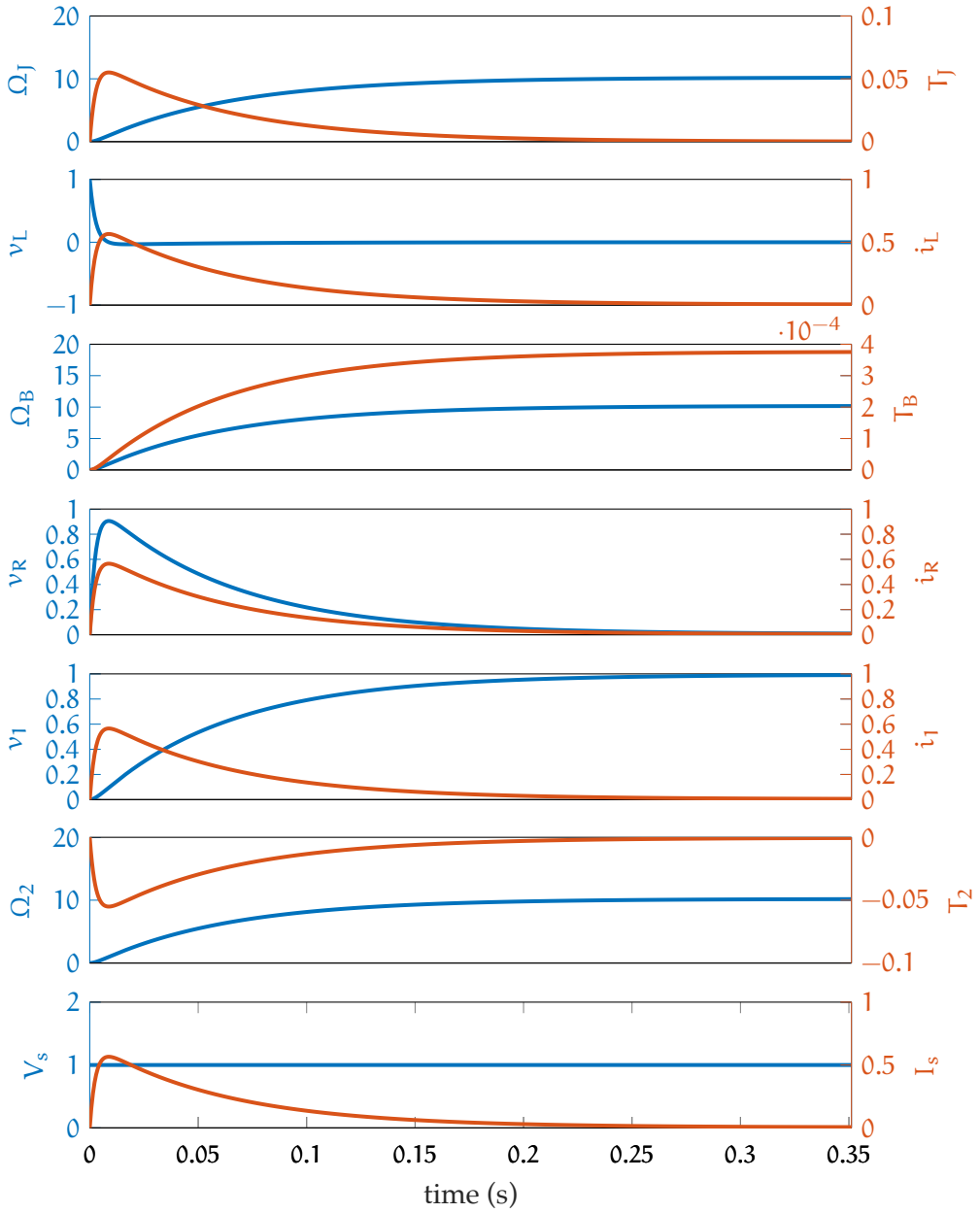

Figure dcmtrans.1: unit step responses for across- (left axes) and through-variables (right axes). Units are as follows: voltage is in V, current is in A, angular velocity is in rad/s, and torque is in N-m. and.

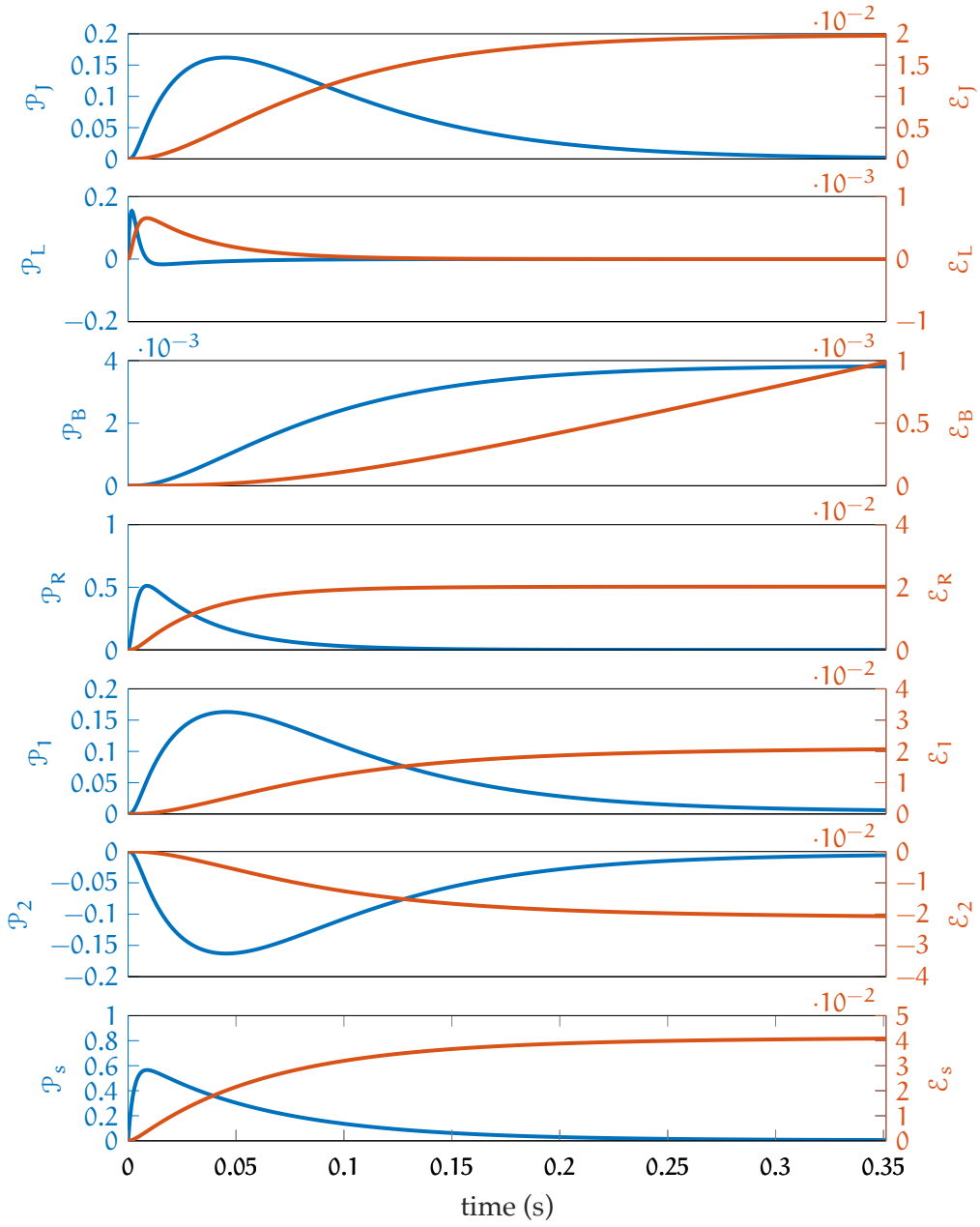

Figure dcmtrans.2: power flow (left axes) and energy storage/dissipation/transformation (right axes) for a unit step response. The unit of power is W and the unit of energy is J.

```
j = 0;
for i = 1:2:size(ys_a,2)
    j = j+1;
    P(:,j) = ys_a(:,i).*ys_a(:,i+1);
    E(:,j) = cumtrapz(t_a,P(:,j));
end
disp('power:');
disp(P(1:6,1:4)) % print a little
disp('energy change:')
disp(E(1:6,1:4)) % print a little
```

```
power:
         0         0         0         0
    0.0000    0.0520    0.0000    0.0052
    0.0001    0.0901    0.0000    0.0191
    0.0002    0.1171    0.0000    0.0392
    0.0005    0.1352    0.0000    0.0635
    0.0009    0.1465    0.0000    0.0907
energy change:
   1.0e-03 *
         0         0         0         0
    0.0000    0.0064    0.0000    0.0006
    0.0000    0.0239    0.0000    0.0036
    0.0001    0.0494    0.0000    0.0108
    0.0001    0.0805    0.0000    0.0235
    0.0003    0.1152    0.0000    0.0425
```

Estimating parameters from the step response

Often, our model has a couple parameters we don't know well from the specifications, but must attempt to measure. For the system under consideration, perhaps the two parameters most interesting to measure are the dominant time constant and the transformer ratio TF (most important). In this section, we explore how one might estimate them from a measured step response. Other parameters in the system could be similarly estimated. By way of the transfer function, the state-space model can be transformed into input-output differential equations.

```
syms B_ J_ TF_ L_ R_ Vs_ s % using underscore for syms
```

```
a_ = [-B_/J_,TF_/J_;-TF_/L_,-R_/L_];
b_ = [0;1/L_];

(s*eye(2)-a_)^-1*b_
```

```
ans =
         TF_/(TF_^2 + B_*R_ + B_*L_*s + J_*R_*s + J_*L_*s^2)
 (B_ + J_*s)/(TF_^2 + B_*R_ + B_*L_*s + J_*R_*s + J_*L_*s^2)
```

The differential equation for Ω_J is

$$\frac{d^2\Omega_J}{dt^2} + \left(\frac{R}{L} + \frac{B}{J}\right)\frac{d\Omega_J}{dt} + \frac{TF^2 + BR}{JL}\Omega_J = \frac{TF}{JL}V_s. \tag{1}$$

The corresponding characteristic equation is

$$\lambda^2 + \left(\frac{R}{L} + \frac{B}{J}\right)\lambda + \frac{TF^2 + BR}{JL} = 0 \tag{2}$$

which has solution

$$\lambda_{1,2} = -\frac{1}{2}\left(\frac{R}{L} + \frac{B}{J}\right) \pm \frac{1}{2}\sqrt{\left(\frac{R}{L} + \frac{B}{J}\right)^2 - 4\frac{TF^2 + BR}{JL}}. \tag{3}$$

For a step input $V_s(t) = \overline{V}_s$, $\Omega_J(0) = d\Omega_J(0)/dt = 0$, and distinct roots λ_1 and λ_2, the solution is

$$\Omega_J(t) = \overline{V}_s\frac{TF}{TF^2 + BR}\left(1 - \frac{1}{\lambda_2 - \lambda_1}\left(\lambda_2 e^{\lambda_1 t} - \lambda_1 e^{\lambda_2 t}\right)\right) \tag{4}$$

Let's compute λ_1 and λ_2.

```
lambda12 = -1/2*(R/L+B/J) + ...
    [1,-1]*1/2*sqrt((R/L+B/J)^2 - 4*(TF^2+B*R)/(J*L))
```

```
lambda12 =
  -16.3467 -373.9941
```

Both values are *real*, so we expect not an oscillation, but a decay to a final value. However, that decay occurs with two different time constants: $\tau_1 = -1/\lambda_1$ and $\tau_2 = -1/\lambda_2$.

```
tau12 = -1./lambda12
disp(['ratio: ',num2str(tau12(1)/tau12(2))])
```

```
tau12 =
    0.0612    0.0027
ratio: 22.8788
```

So second decays much faster than the first. That's good news for our estimation project because we can easily ignore the step response's first $5\tau_2 \approx 0.0134$ s and assume the rest is decaying at τ_1, which we call the *dominant time constant* and which we would like to estimate.
Let's generate some fake response data to get the idea. We'll layer on some Gaussian noise with `randn` to be more realistic. The data is plotted in Figure dcmtrans.3.

```
t_data = linspace(0,-6/lambda12(1),200);

O_fun = @(t) TF/(TF^2+B*R)*...
    (1-1/(lambda12(2)-lambda12(1))*...
    (lambda12(2)*exp(lambda12(1)*t)-...
    lambda12(1)*exp(lambda12(2)*t)));
rng(2);
O_data = O_fun(t_data) + .5*randn(size(t_data));
```

Let's trim the data to eliminate the time interval corresponding to the first five of the "fast" time constant τ_2.

```
[t_5,i_5] = min(abs(t_data-(-5/lambda12(2)))); % delete
t_data_trunc = t_data((i_5+1):end);
O_data_trunc = O_data((i_5+1):end);
```

We need want to take the natural logarithm of the data so we can perform a linear regression to estimate the "experimental" slow time constant $\tilde{\tau}_1$. We must first estimate the steady-state value $\Omega_{J\infty}$ (which we'll also need). We don't want to just take the last value in the array due to its noisiness. The data goes for six slow time constants, so averaging the data for the last time constant is a good estimate.

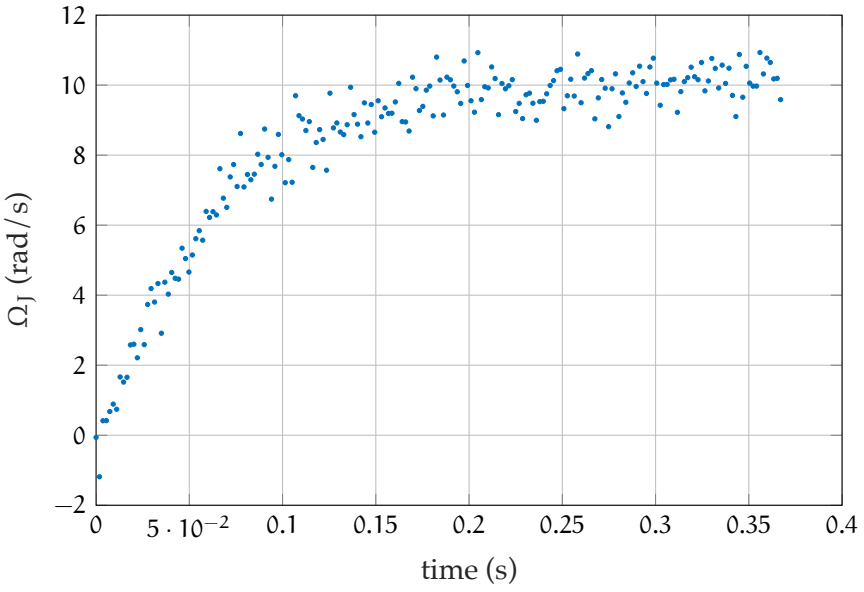

Figure dcmtrans.3: unit step response "data."

```
[t_ss,i_ss] = ...
    min(abs(t_data_trunc-(-5/lambda12(1)))); % start here
O_data_ss = O_data_trunc((i_ss+1):end);
mu_O_ss = mean(O_data_ss)
S_mu_O_ss = std(O_data_ss)/sqrt(length(O_data_ss))
```

```
mu_O_ss =
   10.1801
S_mu_O_ss =
    0.0763
```

Let's use this result to transform the data into its linear form.

```
O_lin = log(-(O_data_trunc-mu_O_ss));
O_lin_complex = find(imag(O_lin)>0);
disp(['number of complex values: ',...
    num2str(length(O_lin_complex))])
```

```
number of complex values: 33
```

Now we have encountered a problem. The noisiness of the data makes some of our points wander into negative-land. Logarithms of negative numbers are complex. Naive approaches like just taking real parts, excluding complex values, or coercing complex values to $-\infty$ all have the issue of biasing the data.
There are a lot of approaches we could take. The best approaches include nonlinear regression and discrete filtering to smooth the data (e.g. `filtfilt`).
We opt for an easier approach: we find the index at which the time series first transgresses the boundary and exclude the data beyond the previous index.

```
i_bad = O_lin_complex(1);
t_lin_trunc = t_data_trunc(1:i_bad-1);
O_lin_trunc = O_lin(1:i_bad-1);
```

This is plotted in Figure dcmtrans.4 along with the linear regression least-squares fit, computed below.

```
pf = polyfit(t_lin_trunc,O_lin_trunc,1);
O_lin_fit = polyval(pf,t_lin_trunc);
tau_1_est = -1/pf(1)
```

```
tau_1_est =
    0.0603
```

So our estimate for τ_1 is $\tilde{\tau}_1 = 60.3$ ms. Recall that our analytic expression for τ_1 is known in terms of other parameters. Similarly, the steady-state value of Ω_J, which has already been estimated to be $\Omega_{J\infty} = 10.18$ (i.e. `mu_O_ss`). This occurs when the time-derivatives of Ω_J are zero. From the solution for Ω_J (or its differential equation), for constant $V_s(t) = \overline{V_s}$, this occurs when

$$\Omega_{J\infty} = \frac{TF}{TF^2 + BR}\overline{V_s}. \tag{5}$$

An analytic expression for TF can be found by solving Equation 5, which yields

$$TF = \overline{V_s} \pm \frac{1}{2\tilde{\Omega}_{J\infty}}\sqrt{V_s^2 - 4BR\tilde{\Omega}_{J\infty}^2} \tag{6}$$

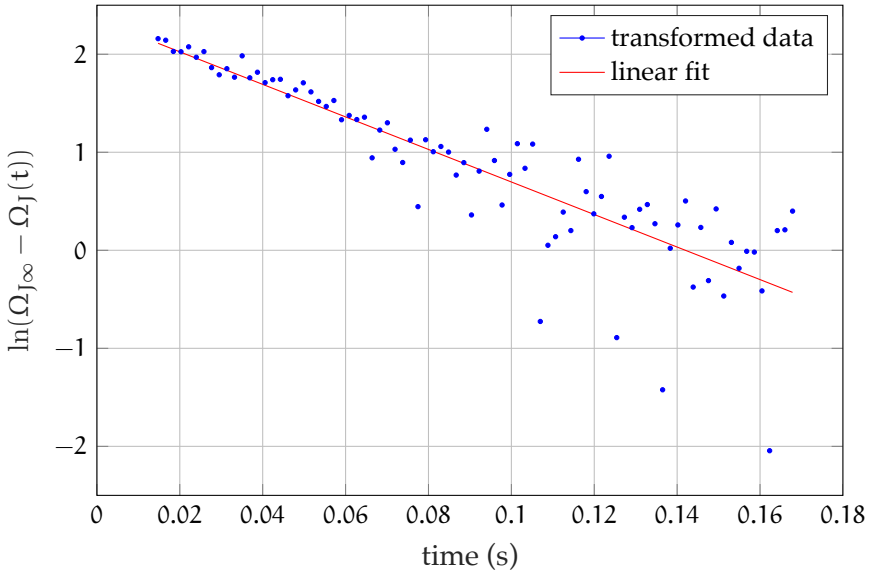

Figure dcmtrans.4: transformed angular velocity "data" with a linear fit.

We choose the solution closer to the *a priori* (spec) value of 0.0974.

```
TF_est = (1 + (- 4*B*R*mu_0_ss^2 + 1^2)^(1/2))/(2*mu_0_ss)
```

```
TF_est =
    0.0976
```

This estimate $\tilde{\mathrm{TF}} = 0.0976$ is very close to the value given in the specification sheet *because we constructed it to be so*. Real measurements would probably yield an estimate further from the specification, which is why we would estimate it.