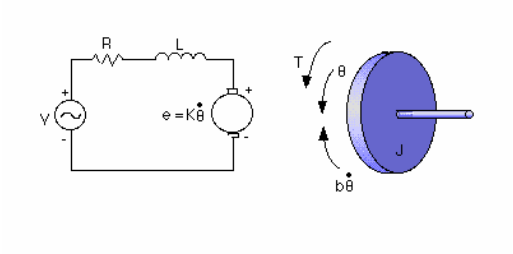
DC Motor Control Design



Introduction

In many applications, it is desirable to operate the motor within some prescribed parameters. For example an electric DC motor should rotate with a prescribed velocity under many different load conditions. In this demonstration, we will illustrate how to model a DC motor in both time domain (as a set of differential equations, as well as in state space form) and frequency domain (as transfer function).

The electric circuit of the armature and the free body diagram of the rotor are shown in the figure above.
The parameters of this model are, with the values in brackets:

J - moment of inertia of the rotor (0.01 kg.m^2/s^2)
b - damping ratio of the mechanical system (0.1 Nms)
K - electromotive force constant (0.01 Nm/Amp)
R - electric resistance (1 ohm)
L - electric inductance (0.5 H)

The input to the system is the applied voltage:
v - input voltage

The output of the system is the angular position of the shaft:
θ - angular position of the shaft

System Definition - Differential Equation Model

```
> d1 := J*diff( theta(t),t,t ) + b*diff( theta(t), t ) =
  K*i(t);
```

Second, we apply Kirchoff's voltage law around the circuit. The back EMF is proportional to the angular velocity of the motor shaft.

```
> d2 := L*diff( i(t), t ) + R*i(t) = v(t) - K * diff(
theta(t), t );
```

The resulting system model is then a set of two differential equations.

$$d1 := J\left(\frac{d^2}{dt^2}\theta(t)\right) + b\left(\frac{d}{dt}\theta(t)\right) = K\,\mathrm{i}(t)$$

$$d2 := L\left(\frac{d}{dt}\mathrm{i}(t)\right) + R\,\mathrm{i}(t) = \mathrm{v}(t) - K\left(\frac{d}{dt}\theta(t)\right)$$

The model parameters are given by:

J = 0.01 kg.m^2/s^2
b = 0.1 Nms
K = 0.01 Nm/Amp
R = 1 ohm
L = 0.5 H

```
> params := {J = 0.01, b = 0.1, K = 0.01, R = 1, L = 0.5};
```

$$params := \{J = 0.0100,\ b = 0.1000,\ K = 0.0100,\ R = 1,\ L = 0.5000\}$$

Open Loop Simulation

Now that we have the set of differential equations describing the system behavior, we can substitute the parameter values into the model and obtain a time response simulation of the system.

Since we have not designed a controller for the system yet, this response is typically referred to as the open-loop characteristic of the system.

Recall the model differential equations.

```
> d1;
  d2;
```

$$J\left(\frac{d^2}{dt^2}\theta(t)\right)+b\left(\frac{d}{dt}\theta(t)\right)=K\,\mathrm{i}(t)$$

$$L\left(\frac{d}{dt}\mathrm{i}(t)\right)+R\,\mathrm{i}(t)=\mathrm{v}(t)-K\left(\frac{d}{dt}\theta(t)\right)$$

Define a step response as the input.

```
> input_eq := v(t) = Heaviside(t): input_eq;
```

$$\mathrm{v}(t)=\mathrm{Heaviside}(t)$$

Set the initial conditions to zero.

```
> initial_conditions := {D(theta)(0)=0, i(0)=0,
  theta(0)=0}: initial_conditions;
```

$$\{\mathrm{D}(\theta)(0)=0,\ \mathrm{i}(0)=0,\ \theta(0)=0\}$$

Plot the response (step response in this case) for the position, i.e. $\theta(t)$, over time.

Substituting in the parameter values

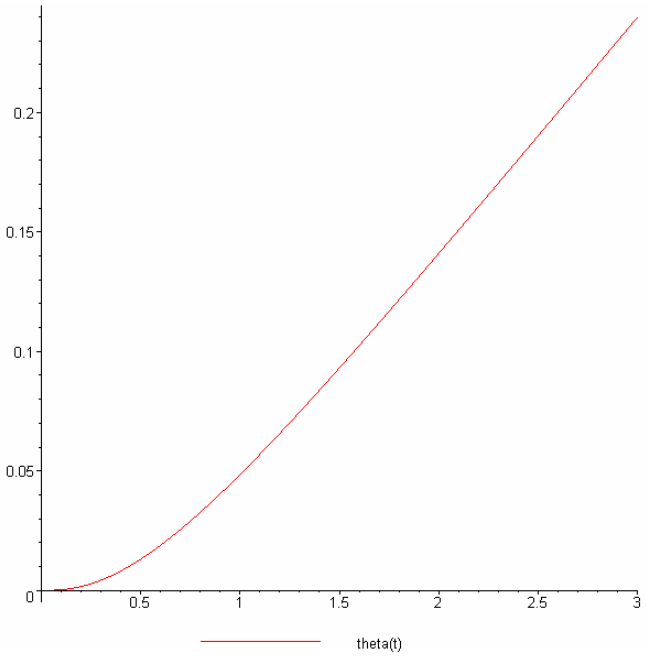
```
> deqs := eval( {d1, d2}, params );
  simplify (deqs);
```

$$deqs := \{0.5000\left(\frac{d}{dt}\mathrm{i}(t)\right)+\mathrm{i}(t)=\mathrm{v}(t)-0.0100\left(\frac{d}{dt}\theta(t)\right),$$
$$0.0100\left(\frac{d^2}{dt^2}\theta(t)\right)+0.1000\left(\frac{d}{dt}\theta(t)\right)=0.0100\,\mathrm{i}(t)\}$$

$$\{0.5000\left(\frac{d}{dt}\mathrm{i}(t)\right)+\mathrm{i}(t)=\mathrm{v}(t)-0.0100\left(\frac{d}{dt}\theta(t)\right),$$
$$0.0100\left(\frac{d^2}{dt^2}\theta(t)\right)+0.1000\left(\frac{d}{dt}\theta(t)\right)=0.0100\,\mathrm{i}(t)\}$$

Calling the custom procedure DEResponse( ) to generate the time response

```
> DEResponse (deqs, initial_conditions, {input_eq},
  theta(t), t, 0..3);
```



Notice that given a step input in the voltage, the rotor shaft rotates continuously as expected.

Transfer Function Model

Now, we apply the custom defined procedure DETransferFn( ) to convert the differential equation model into a transfer function model.

```
> TF[pos] := DETransferFn( {d1, d2}, v(t), theta(t), t, s
);
```

$$TF_{pos} := \frac{K}{s\,(K^2 + L\,s^2\,J + L\,s\,b + R\,J\,s + R\,b)}$$

Open Loop Analysis (Frequency Domain)

We can perform the same open loop analysis using the transfer function model.

First, recall the transfer function model:

```
> TF[pos];
```

$$\frac{K}{s\,(K^2 + L\,s^2\,J + L\,s\,b + R\,J\,s + R\,b)}$$

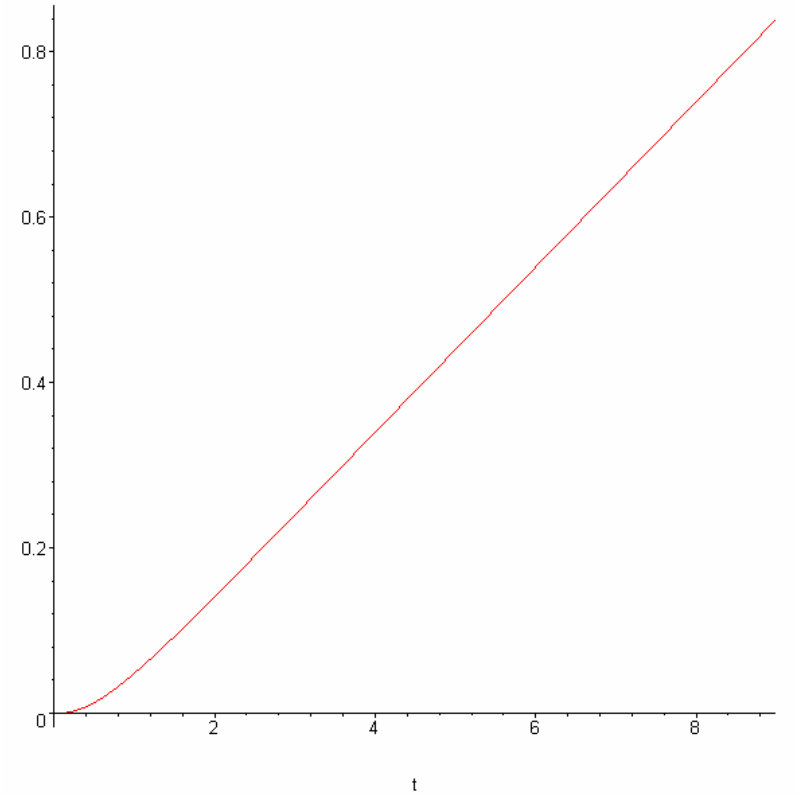
Now, replacing the parameter values in the transfer function.

```
> TF1[pos] := eval( TF[pos], params );
```

$$TF1_{pos} := \frac{0.0100}{s\,(0.1001 + 0.0050\,s^2 + 0.0600\,s)}$$

Calling the custom procedure TFStepResponse( ) to plot the step response of the motor position.

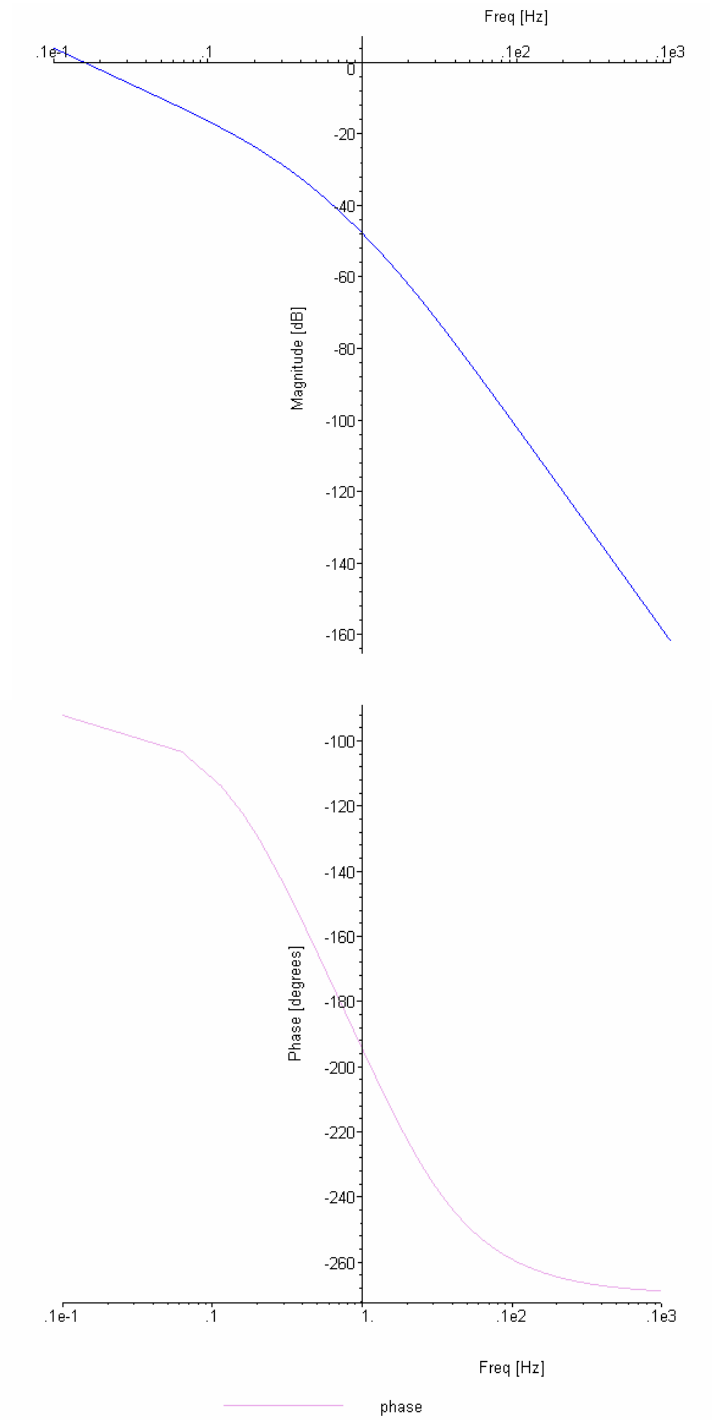
```
> TFStepResponse( TF1[pos], s, 0..9 );
```



In addition to the step response, we can also look at the open loop frequency response of the system in the form of a Bode plot.

Bode plot is a plot showing the magnitude and phase response of a system at different frequencies.

```
> MagnitudePlot( TF1[pos], s, 0.01 .. 100, "Hz" );
  PhasePlot( TF1[pos], s, 0.01 .. 100, "Hz" );
```



Derivation of the State Space Model

Given the Differential Equation model, we can also derive the state space model of the system.

The state space model is defined as follows:

$$\frac{d}{dt}\,\mathrm{x}(t) = A\,\mathrm{x}(t) + B\,\mathrm{u}(t)$$

$$\mathrm{y}(t) = C\,\mathrm{x}(t) + \mathrm{D}\,\mathrm{u}(t)$$

where

u(t) is a vector (of length p) representing the input signals to the system.
y(t) is a vector (of length m) representing the output signals of the system.
x(t) is a vector (of length n) representing the internal states of the system.
A is an n x n matrix
B is an n x p matrix
C is an m x n matrix
D is an m x p matrix

The state vector x(t) is defined as follows:

$$\mathrm{x1}(t) = \mathrm{i}(t)$$

$$\mathrm{x2}(t) = \theta(t)$$

$$\mathrm{x3}(t) = \frac{d}{dt}\,\theta(t)$$

Consider the two differential equations that define the system dynamics.

```
> d1; d2;
```

$$J\left(\frac{d^2}{dt^2}\,\theta(t)\right) + b\left(\frac{d}{dt}\,\theta(t)\right) = K\,\mathrm{i}(t)$$

$$L\left(\frac{d}{dt}\,\mathrm{i}(t)\right) + R\,\mathrm{i}(t) = \mathrm{v}(t) - K\left(\frac{d}{dt}\,\theta(t)\right)$$

From the two differential equations, rearrange to isolate the differentials:

```
> Xv := Vector ([isolate (d1, diff (theta(t), t, t)),
  isolate (d2, diff (i(t), t))]);
```

$$Xv := \begin{bmatrix} \frac{d^2}{dt^2}\theta(t) = \frac{K\,\mathrm{i}(t) - b\left(\frac{d}{dt}\theta(t)\right)}{J} \\ \frac{d}{dt}\mathrm{i}(t) = \frac{\mathrm{v}(t) - K\left(\frac{d}{dt}\theta(t)\right) - R\,\mathrm{i}(t)}{L} \end{bmatrix}$$

```
> Xv := eval(Xv, i(t) = x1(t)):
  Xv := eval(Xv, theta(t) = x2(t)):
  Xv := eval(Xv, diff (x2(t), t) = x3(t));
```

$$Xv := \begin{bmatrix} \frac{d}{dt}\mathrm{x3}(t) = \frac{K\,\mathrm{x1}(t) - b\,\mathrm{x3}(t)}{J} \\ \frac{d}{dt}\mathrm{x1}(t) = \frac{\mathrm{v}(t) - K\,\mathrm{x3}(t) - R\,\mathrm{x1}(t)}{L} \end{bmatrix}$$

```
> Xv := Vector ([Xv, (diff(x2(t), t) = x3(t))]);
```

$$Xv := \begin{bmatrix} \frac{d}{dt}\mathrm{x3}(t) = \frac{K\,\mathrm{x1}(t) - b\,\mathrm{x3}(t)}{J} \\ \frac{d}{dt}\mathrm{x1}(t) = \frac{\mathrm{v}(t) - K\,\mathrm{x3}(t) - R\,\mathrm{x1}(t)}{L} \\ \frac{d}{dt}\mathrm{x2}(t) = \mathrm{x3}(t) \end{bmatrix}$$

Putting the set of differential equations into vector form.

```
> F := <rhs (Xv[2]), rhs (Xv[3]), rhs (Xv[1])>:
  <lhs (Xv[2]), lhs (Xv[3]), lhs (Xv[1])> = F;
```

$$\begin{bmatrix} \frac{d}{dt}\mathrm{x1}(t) \\ \frac{d}{dt}\mathrm{x2}(t) \\ \frac{d}{dt}\mathrm{x3}(t) \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{v}(t) - K\,\mathrm{x3}(t) - R\,\mathrm{x1}(t)}{L} \\ \mathrm{x3}(t) \\ \frac{K\,\mathrm{x1}(t) - b\,\mathrm{x3}(t)}{J} \end{bmatrix}$$

Extracting the state space model matrices.

```
> Xs := <X1, X2, X3>:
  Fs := eval (F, {x1(t) = X1, x2(t) = X2, x3(t) = X3, v(t)
  = V}):
  A_ss := Matrix( [ seq( [ seq( eval( diff( Fs[i], Xs[j] ),
  {V=0} ), j=1..3 ) ], i=1..3 ) ] );
  B_ss := Vector( map( diff, Fs, V ) );
  C_ss := Vector[row]([0, 1, 0]);
  D_ss := 0;
```

$$A\_ss := \begin{bmatrix} -\frac{R}{L} & 0 & -\frac{K}{L} \\ 0 & 0 & 1 \\ \frac{K}{J} & 0 & -\frac{b}{J} \end{bmatrix}$$

$$B\_ss := \begin{bmatrix} \frac{1}{L} \\ 0 \\ 0 \end{bmatrix}$$

$$C\_ss := [0, 1, 0]$$

$$D\_ss := 0$$

Substituting the numerical values would lead to the following state space model.

```
> Ass := eval(A_ss, params);
  Bss := eval(B_ss, params);
  Css := eval(C_ss, params);
```

$$Ass := \begin{bmatrix} -2.0000 & 0 & -0.0200 \\ 0 & 0 & 1 \\ 1.0000 & 0 & -10.0000 \end{bmatrix}$$

$$Bss := \begin{bmatrix} 2.0000 \\ 0 \\ 0 \end{bmatrix}$$

$$Css := [0, 1, 0]$$

Simulation

Similar to the case for the differential equation model and the transfer function model, we can perform simulation using the state space model.

First, we setup the state vector.

```
> Xss := Vector([seq (x||i(t), i=1..3)]);
```

$$Xss := \begin{bmatrix} x1(t) \\ x2(t) \\ x3(t) \end{bmatrix}$$

And the set of first order state differential equations.

```
> EQ1ss := map (diff, Xss, t) = Ass.Xss + Bss * u(t);
```

$$EQ1ss := \begin{bmatrix} \frac{d}{dt} x1(t) \\ \frac{d}{dt} x2(t) \\ \frac{d}{dt} x3(t) \end{bmatrix} = \begin{bmatrix} -2.0000\, x1(t) - 0.0200\, x3(t) + 2.0000\, u(t) \\ x3(t) \\ 1.0000\, x1(t) - 10.0000\, x3(t) \end{bmatrix}$$

Here is the output equation.

```
> EQ2ss := y(t) = Css.Xss;
```

$$EQ2ss := y(t) = x2(t)$$

Set all initial conditions for the states to zero.

```
> X_ic := {seq (x||i(0) = 0, i=1..3)};
```

$$X\_ic := \{x1(0) = 0, x2(0) = 0, x3(0) = 0\}$$

Setting the input signal to a step (Heaviside)..

```
> input := {u(t) = Heaviside (t)};
```

$$input := \{u(t) = \text{Heaviside}(t)\}$$

Combine all of the differential equations with the parameter values.

```
> {seq (lhs(EQ1ss)[i] = rhs(EQ1ss)[i], i=1..3), EQ2ss}:
  de_set := eval (%, params);
```

$$de\_set := \{\frac{d}{dt} x1(t) = -2.0000\, x1(t) - 0.0200\, x3(t) + 2.0000\, u(t), y(t) = x2(t),$$

$$\frac{d}{dt} x2(t) = x3(t), \frac{d}{dt} x3(t) = 1.0000\, x1(t) - 10.0000\, x3(t)\}$$

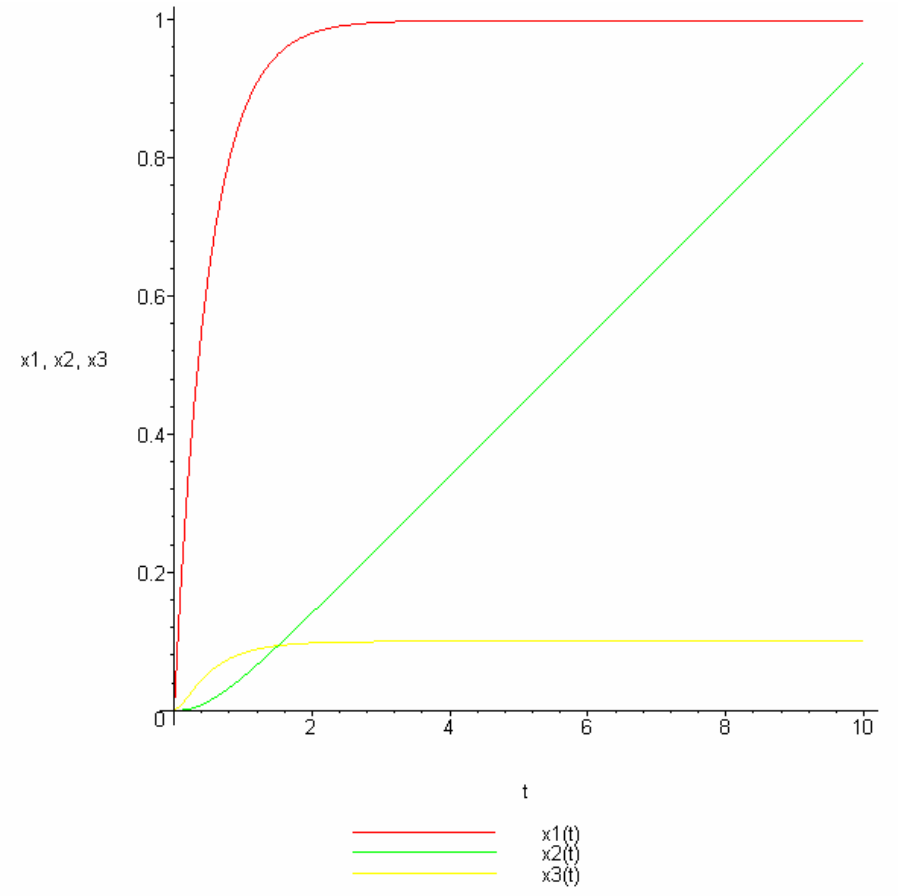
We can then pass the set of first order differential equations to dsolve( ), with the initial conditions and input signal, to generate the numeric solution.

```
> sol := dsolve (de_set union input union X_ic, numeric);
```

$$sol := \textbf{proc}(x\_rkf45\_dae) \ \ldots \ \textbf{end proc}$$

Plotting all the state variables.

```
> odeplot (sol, [[t, x1(t)], [t, x2(t)], [t, x3(t)]],
  0..10,numpoints=100, legend=["x1(t)","x2(t)","x3(t)"]);
```



LQR Design

In this section, we would like to design a Linear Quadratic Regulator (LQR) for our DC motor.

Our control objective is to minimize the motor shaft position error.
As such, we are weighting high the position state in the Q matrix.

```
> Qss := <<0.1|0|0>,<0|1|0>,<0|0|0.1>>;
Rss := 0.1;
```

$$Qss := \begin{bmatrix} 0.1000 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.1000 \end{bmatrix}$$

$$Rss := 0.1000$$

The controller gain for the full state feedback LQR controller is computed using the custom procedure LQR( ).

```
> Kss := LQR(Ass, convert (Bss, Matrix), Qss, Rss);
```

$$Kss := [0.5239 \quad 3.1623 \quad 0.3222]$$

Now, recall our system equations and initial conditions:

```
> de_set;
X_ic;
```

$$\{\frac{d}{dt}\,\mathrm{x1}(t) = -2.0000\,\mathrm{x1}(t) - 0.0200\,\mathrm{x3}(t) + 2.0000\,\mathrm{u}(t),\ \mathrm{y}(t) = \mathrm{x2}(t),\ \frac{d}{dt}\,\mathrm{x2}(t) = \mathrm{x3}(t),$$

$$\frac{d}{dt}\,\mathrm{x3}(t) = 1.0000\,\mathrm{x1}(t) - 10.0000\,\mathrm{x3}(t)\,\}$$

$$\{\mathrm{x1}(0) = 0,\ \mathrm{x2}(0) = 0,\ \mathrm{x3}(0) = 0\,\}$$

Implementing the feedback control law and setting the input reference to a step signal.

```
> controller := {u(t) = - (Kss.(Xss-Vector([0,
Heaviside(t), 0])))[1]};
```

$$controller :=$$

$$\{\mathrm{u}(t) = -0.5239\,\mathrm{x1}(t) - 3.1623\,\mathrm{x2}(t) + 3.1623\,\mathrm{Heaviside}(t) - 0.3222\,\mathrm{x3}(t)\,\}$$

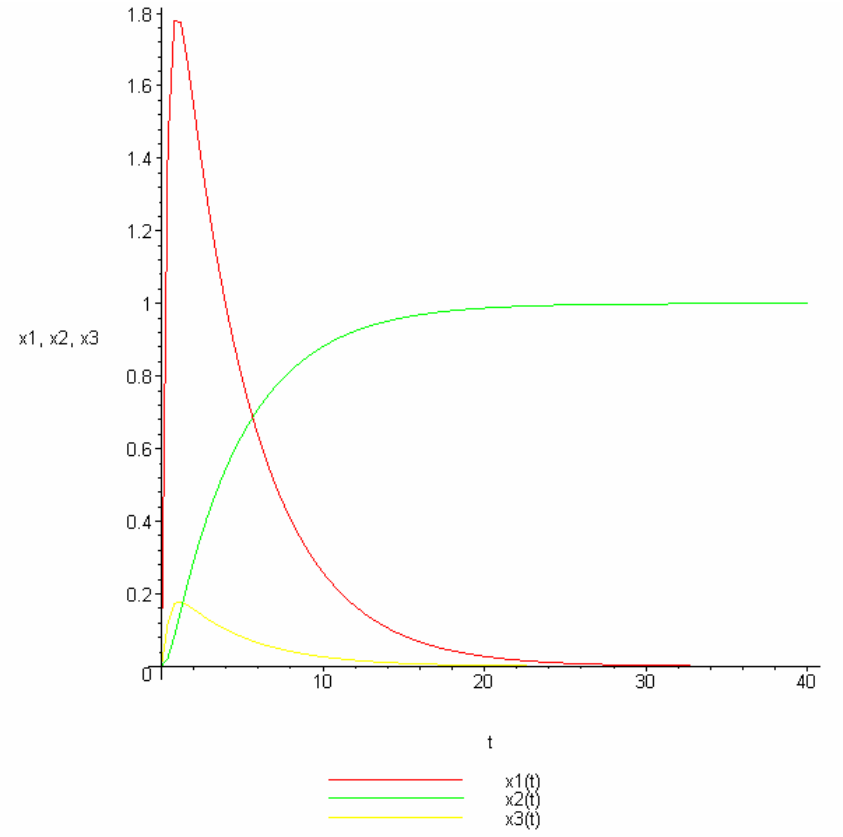
Passing everything to dsolve( ) to get a time response of the system.

```
> sol2 := dsolve (de_set union X_ic union controller,
numeric);
```

$$sol2 := \mathbf{proc}(x\_rkf45\_dae)\ \ldots\ \mathbf{end\ proc}$$

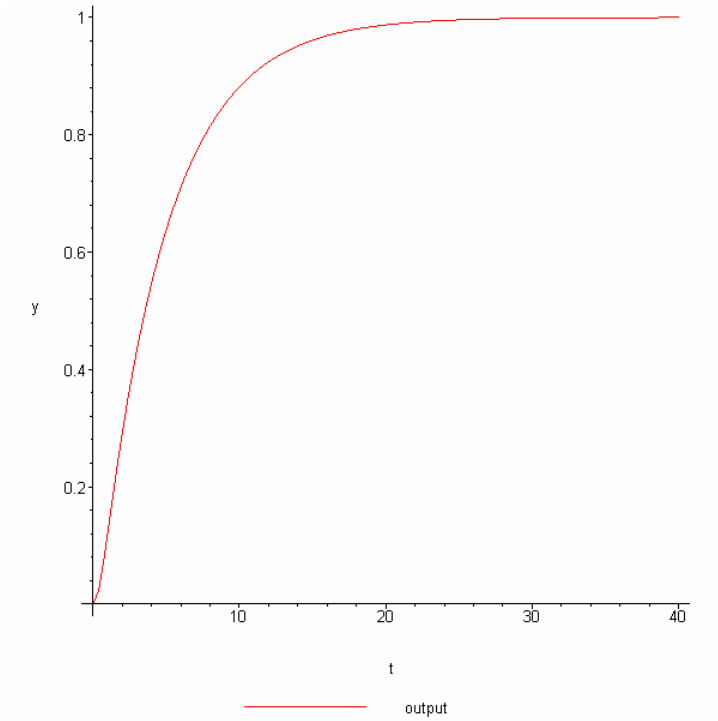
Showing all of the system states.

```
> odeplot(sol2, [[t, x1(t)], [t, x2(t)], [t, x3(t)]],
0..40,numpoints=100, legend=["x1(t)","x2(t)","x3(t)"]);
```



Focusing on the output only. Notice there is no steady state error.

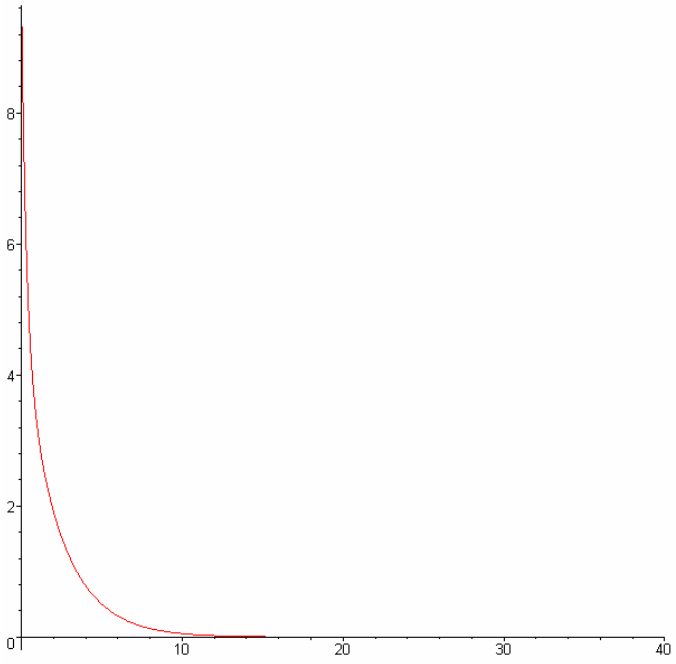
```
> odeplot (sol2, [t, y(t)], 0..40,numpoints=100,
legend=["output"]);
```



Now, we define an energy-like quantity (square of the input voltage) to give us an idea of the "cost" or the "control effort" required in operating the DC motor.

```
> f_u := z-> eval (u(t), sol2(z))^2;
plot (f_u, 0..40);
```

$$f\_u := z \rightarrow \mathrm{eval}(\mathrm{u}(t), \mathrm{sol2}(z))^2$$



The above plot shows that a large control effort is required initially to drive the motor. This is in response to the step change in the reference signal. Once the motor shaft is at the desired location, no more control is necessary and therefore, the control effort goes to zero (after around 10 seconds).

Animation

As a final step in our current example, we would like to vary the weighting factor R for the LQR controller to see how it affects the system output response.

The R factor is the weighting coefficient for the input signal. By increasing R, we are putting more penalty on the control signal.
As such, the resulting controller will try to minimize the control efforts used to achieve the desired output.

First, recall the Q matrix setting:

```
> Qss := <<0.1|0|0>,<0|1|0>,<0|0|0.1>>;
```

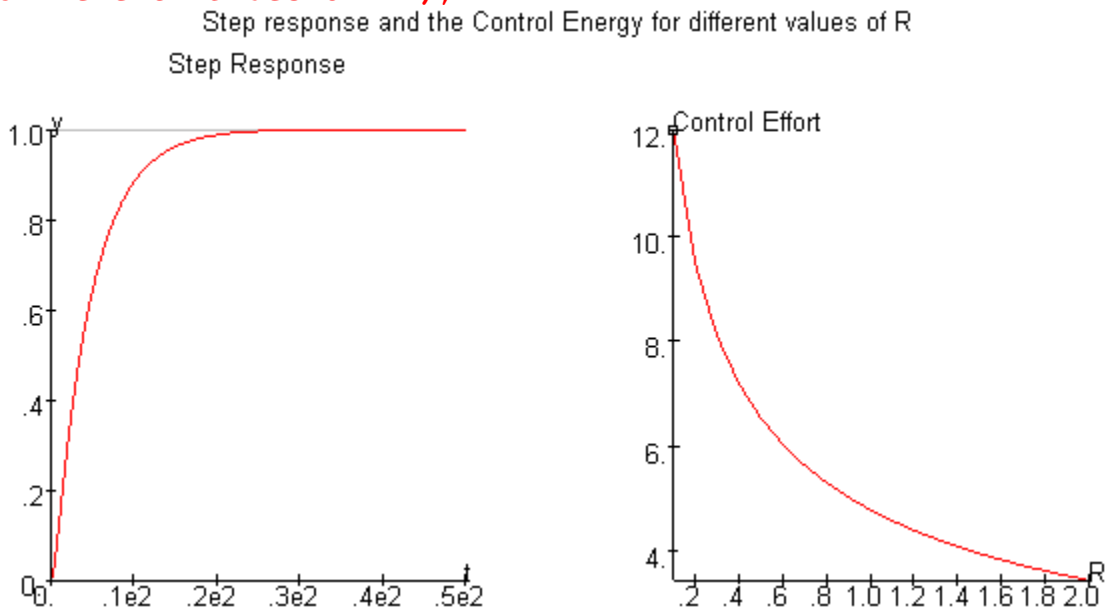
$$Qss := \begin{bmatrix} 0.1000 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.1000 \end{bmatrix}$$

Next, we will generate the controller and the corresponding step time response for each value of R from 0.1 to 2.
To compare the "performance" of each of the controllers, we define the "control effort" of the controller as the square of the input signal, summed over time.
This associated "control effort" (or "energy-like") value for each controller is saved for each controller.

```
> Lp := NULL:
  Le := NULL:
  Lr := NULL:
  # Here, we vary R from 0.1 to 2
  for rx from 0.1 to 2 by 0.1 do
        energy, plt := frame (Ass, Bss, Qss, rx, de_set);
        Lp := Lp,plt;
        Le := Le,[rx, energy];
        Lr := Lr, rx;
  end do:
  Lp := [Lp]:
  Le := [Le]:
  Lr := [Lr]:
```

Finally, we are compiling the output responses and the corresponding control efforts into an animation.
On the left plot, the closed-loop time response of each controller is shown.
On the right plot, the "control efforts" as a function of the value of R is shown.
For each step response on the left plot, a box is marked in the "Control Effort" plot on the right.
It shows the control effort required to generate the corresponding step response on the left.

```
> e_plt := plot (Le, labels=["R","Control Effort"]):
  L_anim := NULL:
  num := nops (Lp):
  for rx from 1 to num do
        p1 := plots[display] ([e_plt, pointplot ({Le[rx]},
          symbol=BOX, symbolsize=10)]);
        _P := array (1..1, 1..2);
        _P[1,1] := Lp[rx];
        _P[1,2] := p1;
        L_anim := L_anim, plots[display] (_P);
  end do:
  plots[display] ([L_anim], insequence=true,
  title = "Step response and the Control Energy for
  different values of R");
```



From the above animation, we can see that, as expected, the control effort decreases as R increase. The decrease in the control effort (or "operating cost") results in a slower step response of the system. In other words, it is a trade-off between energy spent and response speed: The more energy we are willing to spend, the faster our response will be.

Conclusion

We have demonstrated how we can use Maple to seamlessly model and analyze both the open loop and closed loop characteristic of a physical system in both the time and the frequency domain.