

5. Examples of Solving Systems

■ 5.1. Continuous-Time Systems

Introduction

SchematicSolver has many unique features not available in other software: symbolic signal processing brings you

- Computation of transfer functions as closed-form expressions in terms of symbolic system parameters
- Finding the closed-form response from the schematic
- Symbolic optimization of the system response

The derived result is the most general because all system parameters and inputs can be given by symbols.

Other important features include:

- Design of systems for known symbolic transfer function, impulse, or step response; you can generate the schematic of the system and find the system parameters
- Building models from automatically generated schematics; you can change system parameters on the fly and immediately see what happens with the results

SchematicSolver's powerful functions for solving continuous-time (analog) systems are illustrated by the subsequent examples.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[1]:= Needs["SchematicSolver`"];
```

Diving Submarine System

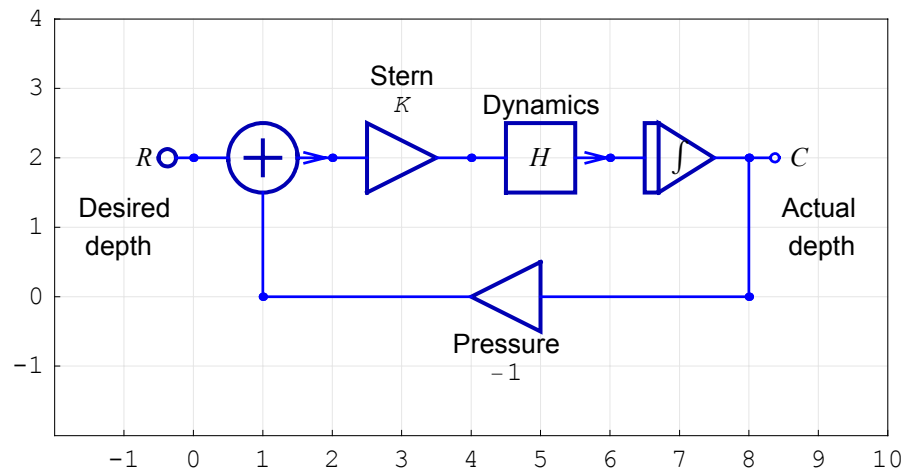
For the simplified block diagram model of a diving submarine find the transfer function C/R . *Stern plane actuator* is an amplifier of gain K , *submarine dynamics* is represented by $H = \frac{(s+a)^2}{s^2+w^2}$, and *pressure transducer* is an amplifier with a gain of negative one.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[2]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[3]:= divingSubmarineSystem = {
  {"Input", {0, 2}, R},
  {"Output", {8, 2}, C},
  {"Integrator", {{6, 2}, {8, 2}}},
  {"Amplifier", {{2, 2}, {4, 2}}, K, "Stern"},
  {"Amplifier", {{8, 0}, {1, 0}}, -1, "Pressure"},
  {"Block", {{4, 2}, {6, 2}}, H, "Dynamics"},
  {"Text", {-1, 1}, "Desired\n depth"},
  {"Text", {9, 1}, "Actual\n depth"},
  {"Adder", {{0, 2}, {1, 0}, {2, 2}, {1, 3}}, {1, 1, 2, 0}},
  {"Line", {{8, 2}, {8, 0}}}
};
ShowSchematic[%, PlotRange -> {{-2, 10}, {-2, 4}}];
```



`ContinuousSystemTransferFunction` computes the transfer function matrix of the system:

```
In[5]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[divingSubmarineSystem]
```

```
Out[5]= {{{{  $\frac{HK}{HK+s}$  }}, {Y[{0, 2}]}, {Y[{8, 2}]}}
```

Here is the given submarine dynamics:

```
In[6]:= submarineDynamics = H → (s + a) ^ 2 / (s ^ 2 + w ^ 2)
```

```
Out[6]= H →  $\frac{(a + s)^2}{s^2 + w^2}$ 
```

The transfer function C/R is the element of the transfer function matrix:

```
In[7]:= actualDepthTF = tfMatrix[[1, 1]] /. submarineDynamics // Together
```

```
Out[7]=  $\frac{K (a + s)^2}{a^2 K + 2 a K s + K s^2 + s^3 + s w^2}$ 
```

This collects together terms involving the same powers of the complex frequency s :

```
In[8]:= Numerator[actualDepthTF] / Collect[Denominator[actualDepthTF], s] //
  TraditionalForm
```

```
Out[8]//TraditionalForm=

$$\frac{K (a + s)^2}{s^3 + K s^2 + (w^2 + 2 a K) s + a^2 K}$$

```

The derived result is the most general because all system parameters are given by symbols. Here is the transfer function with specific parameter values:

```
In[9]:= actualDepthTF1 = actualDepthTF /. {a → 1, w →  $\frac{1}{\sqrt{10}}$ }
```

```
Out[9]=  $\frac{K (1 + s)^2}{K + \frac{s}{10} + 2 K s + K s^2 + s^3}$ 
```

```
In[10]:= Numerator[actualDepthTF1] /
  Collect[Denominator[actualDepthTF1], s] // TraditionalForm
```

```
Out[10]//TraditionalForm=

$$\frac{K (s + 1)^2}{s^3 + K s^2 + (2 K + \frac{1}{10}) s + K}$$

```

Unstable Plant System

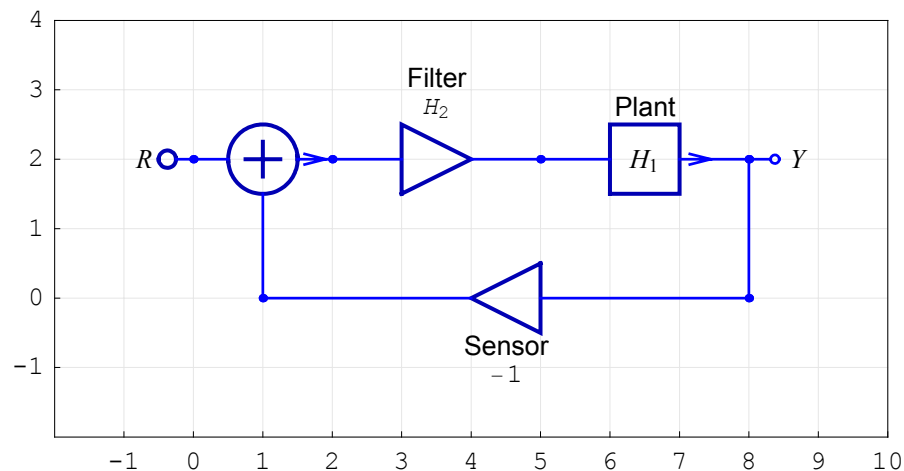
An *unstable plant* described by the transfer function H_1 is made part of a new *feedback system* (shown in Figure) that includes a *filter* H_2 in the forward path and a *sensor* with a gain of negative one in the feedback path. Find the overall transfer function $H = Y/R$.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[11]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[12]:= unstablePlantSystem = {
  {"Input", {0, 2}, R},
  {"Output", {8, 2}, Y},
  {"Amplifier", {{2, 2}, {5, 2}}, H2, "Filter"},
  {"Block", {{5, 2}, {8, 2}}, H1, "Plant"},
  {"Amplifier", {{8, 0}, {1, 0}}, -1, "Sensor"},
  {"Adder", {{0, 2}, {1, 0}, {2, 2}, {1, 3}}, {1, 1, 2, 0}},
  {"Line", {{8, 2}, {8, 0}}}
};
ShowSchematic[% /. {H1 -> H1, H2 -> H2}, PlotRange -> {{-2, 10}, {-2, 4}}];
```



ContinuousSystemTransferFunction computes the transfer function matrix of the system:

```
In[14]:= {tfMatrix, systemInp, systemOut} =
          ContinuousSystemTransferFunction[unstablePlantSystem]
```

```
Out[14]= {{{{ $\frac{H1 H2}{1 + H1 H2}$ }}}, {Y[{0, 2}]}, {Y[{8, 2}]}}
```

Assume that the plant and filter are given by

```
In[15]:= H1H2value = {H1 → 1 / (s * (s - 1)), H2 → (k * s + 8) / (s + 10)};
          TraditionalForm[H1H2value /. {H1 → H1, H2 → H2}]
```

```
Out[15]//TraditionalForm=
          {H1 →  $\frac{1}{(s-1)s}$ , H2 →  $\frac{ks+8}{s+10}$ }
```

The overall transfer function is the element of the transfer function matrix:

```
In[16]:= unstablePlantTF = tfMatrix[[1, 1]] /. H1H2value // Together
```

```
Out[16]=  $\frac{8 + ks}{8 - 10s + ks + 9s^2 + s^3}$ 
```

This collects together terms involving the same powers of the complex frequency s :

```
In[17]:= Numerator[unstablePlantTF] /
          Collect[Denominator[unstablePlantTF], s] // TraditionalForm
```

```
Out[17]//TraditionalForm=
           $\frac{ks+8}{s^3+9s^2+(k-10)s+8}$ 
```

Supply and Demand System

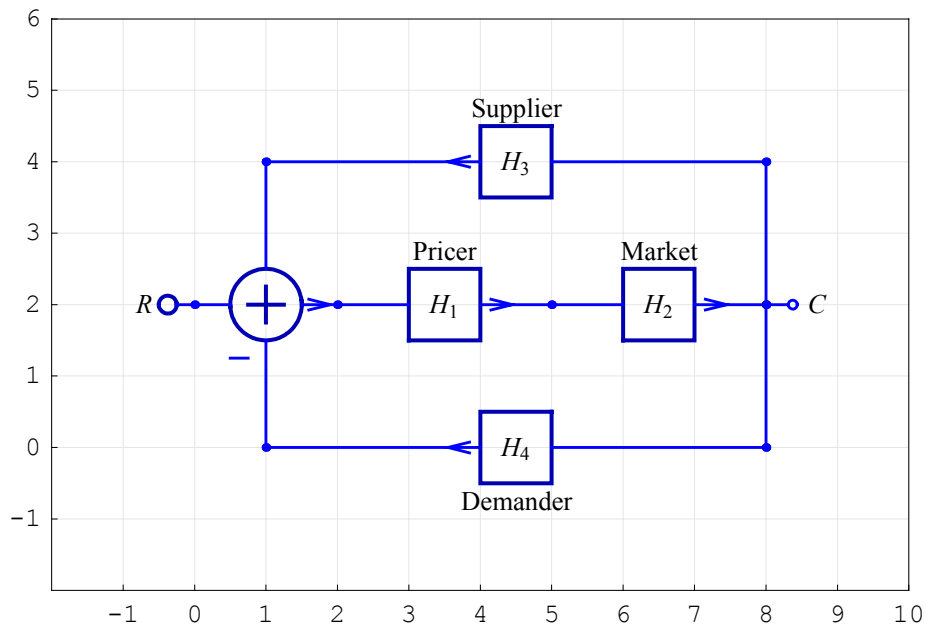
Assume that linear approximations in the form of transfer functions are available for each block of the *supply and demand system*, and that the system can be represented by Figure below. Determine the overall transfer function of the system.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[18]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[19]:= supplyDemandSystem = {
  {"Input", {0, 2}, R},
  {"Output", {8, 2}, C},
  {"Block", {{2, 2}, {5, 2}}, H1, "Pricer"},
  {"Block", {{5, 2}, {8, 2}}, H2, "Market"},
  {"Block", {{8, 0}, {1, 0}}, H4, "Demander"},
  {"Block", {{8, 4}, {1, 4}}, H3, "Supplier", TextOffset -> {0, 1}},
  {"Adder", {{0, 2}, {1, 0}, {2, 2}, {1, 4}}, {1, -1, 2, 1}},
  {"Line", {{8, 2}, {8, 0}}, {"Line", {{8, 2}, {8, 4}}}
};
ShowSchematic[% /. {H1 -> H1, H2 -> H2, H3 -> H3, H4 -> H4},
  PlotRange -> {{-2, 10}, {-2, 6}}];
```



ContinuousSystemTransferFunction computes the transfer function matrix of the system:

```
In[21]:= {tfMatrix, systemInp, systemOut} =
          ContinuousSystemTransferFunction[supplyDemandSystem]
```

```
Out[21]= {{{{- $\frac{H_1 H_2}{-1 + H_1 H_2 H_3 - H_1 H_2 H_4}$ }}}, {Y[{0, 2}]}, {Y[{8, 2}]}}
```

The transfer function C/R is the element of the transfer function matrix:

```
In[22]:= supplyDemandTF = tfMatrix[[1, 1]] // Simplify
```

```
Out[22]=  $\frac{H_1 H_2}{1 + H_1 H_2 (-H_3 + H_4)}$ 
```

that is better typeset with

```
In[23]:= supplyDemandTF /. {H1 -> H1, H2 -> H2, H3 -> H3, H4 -> H4} // TraditionalForm
```

```
Out[23]//TraditionalForm=

$$\frac{H_1 H_2}{H_1 H_2 (H_4 - H_3) + 1}$$

```

Unity Feedback System

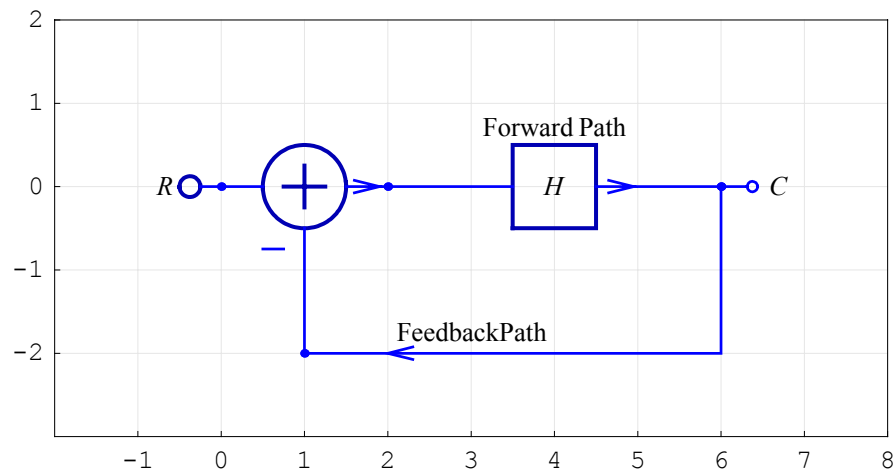
A *unity feedback system* is a feedback system in which the primary feedback is identically equal to the controlled output. Find the response of the system.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[24]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[25]:= unityFeedbackSystem = {
  {"Input", {0, 0}, R},
  {"Output", {6, 0}, C},
  {"Block", {{2, 0}, {6, 0}}, H, "Forward Path"},
  {"Arrow", {{2, -2}, {6, -2}},
  "FeedbackPath", ShowArrowTail → False},
  {"Adder", {{0, 0}, {1, -2}, {2, 0}, {1, 1}}, {1, -1, 2, 0}},
  {"Line", {{6, 0}, {6, -2}, {1, -2}}}
};
ShowSchematic[%, PlotRange → {{-2, 8}, {-3, 2}}];
```



ContinuousSystemEquations sets up the equations of the system:

```
In[27]:= {unityFeedbackEquations, vars} =
  ContinuousSystemEquations[unityFeedbackSystem];
```

It is better typeset with

```
In[28]:= typoSubstYkn = {Y[{k_Integer, n_Integer}] => Yk,n};
```

```
In[29]:= unityFeedbackEquations /. typoSubstYkn // ColumnForm //  
TraditionalForm
```

```
Out[29]//TraditionalForm=
```

$$Y_{0,0} == R$$

$$Y_{6,0} == H Y_{2,0}$$

$$Y_{2,0} == Y_{0,0} - Y_{6,0}$$

ContinuousSystemResponse finds the response of the system:

```
In[30]:= {unityFeedbackResponse, vars} =  
ContinuousSystemResponse[unityFeedbackSystem];
```

```
In[31]:= unityFeedbackResponse /. typoSubstYkn // ColumnForm //  
TraditionalForm
```

```
Out[31]//TraditionalForm=
```

$$Y_{6,0} \rightarrow \frac{HR}{H+1}$$

$$Y_{2,0} \rightarrow \frac{R}{H+1}$$

$$Y_{0,0} \rightarrow R$$

Satellite Elevation Tracking System

A simplified block diagram of a *satellite elevation tracking system* can be represented by Figure below. The multiloop system uses a combination of unity negative feedback and rate feedback to produce a critically damped response. Obtain the closed-loop transfer function of the system.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[32]:= Needs["SchematicSolver`"];
```

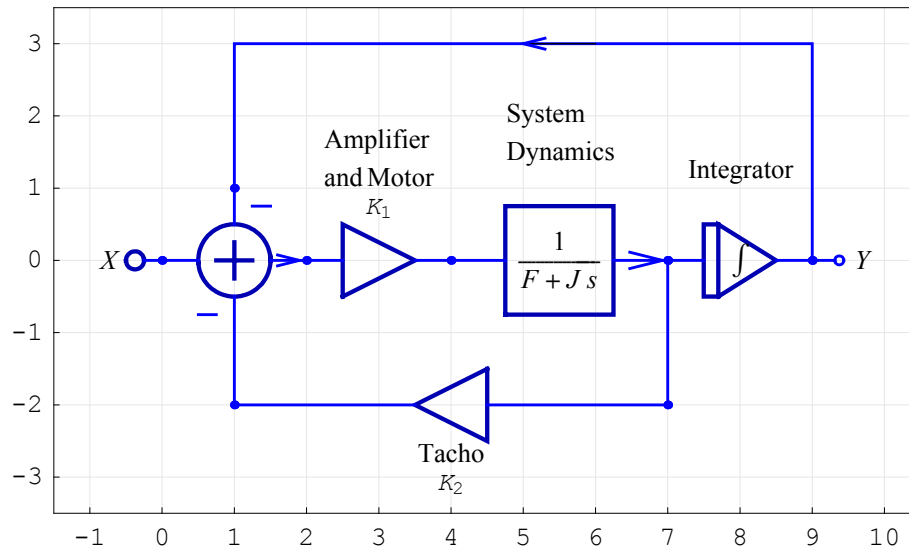
Here is the system schematic:

```
In[33]:= satelliteElevationTrackingSystem = {
  {"Input", {0, 0}, X, "Elevation demand"},
  {"Adder", {{0, 0}, {1, -2}, {2, 0}, {1, 1}}, {1, -1, 2, -1}},
  {"Amplifier", {{2, 0}, {4, 0}}, K1, "Amplifier\nand Motor"},
  {"Block", {{4, 0}, {7, 0}}, 1 / (J*s + F),
   "System\nDynamics\n", ElementSize → {3, 2}},
  {"Integrator", {{7, 0}, {9, 0}}, 1, "Integrator\n"},
  {"Line", {{7, 0}, {7, -2}}},
  {"Amplifier", {{7, -2}, {1, -2}}, K2, "Tacho"},
  {"Line", {{9, 0}, {9, 3}, {1, 3}, {1, 1}}},
  {"Arrow", {{5, 3}, {6, 3}}},
  {"Output", {9, 0}, Y, "Pitch"}
};
```

that is better typeset with

```
In[34]:= typoSubst = {K1 → K1, K2 → K2};
```

```
In[35]:= ShowSchematic[satelliteElevationTrackingSystem/. typoSubst,
  PlotRange -> {{-1.5, 10.5}, {-3.5, 3.5}}];
```



`ContinuousSystemTransferFunction` computes the transfer function matrix of the system:

```
In[36]:= {tfMatrix, systemInp, systemOut} = ContinuousSystemTransferFunction[
  satelliteElevationTrackingSystem];
```

The transfer function of this single-input single-output system is the element of the transfer function matrix:

```
In[37]:= H = tfMatrix[[1, 1]];
  H /. typoSubst // TraditionalForm
```

```
Out[38]//TraditionalForm=

$$\frac{K_1}{Js^2 + Fs + K_1 K_2 s + K_1}$$

```

CD-media Controller

Find the transfer function and step response of the *CD-media controller*.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[39]:= Needs["SchematicSolver`"];
```

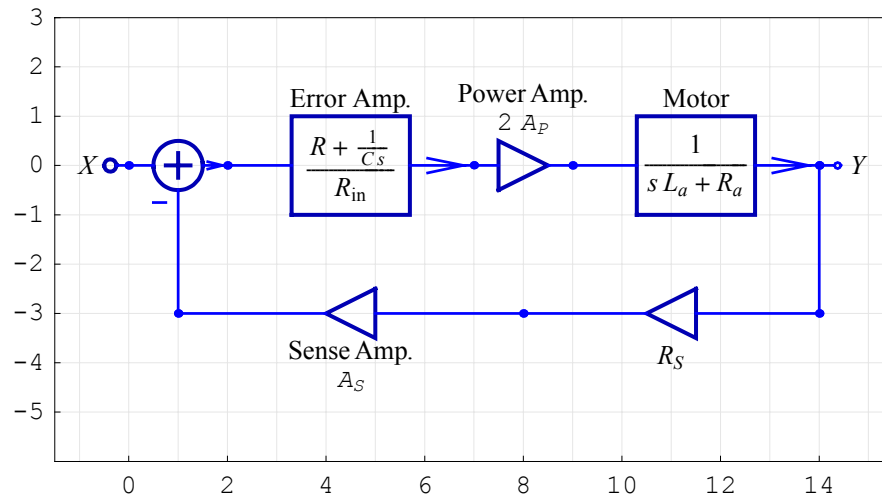
FAN8024D/BD, which is a 4CH motor drive IC suitable for CD-media applications (include CD-ROM, CD-RW, DVDP and DVD-ROM), has 2 current feedback control channels for the Focus and Tracking actuator. The application system, *CD-media controller*, is illustrated by the following block diagram (Application Note 4109, "A guide to the design of current feedback control," Fairchild Semiconductor Corporation, <http://www.fairchildsemi.com>, AN-4109.pdf, ©2001):

```
In[40]:= CDmediaController = {
  {"Input", {0, 0}, X, "Reference Voltage"},
  {"Adder", {{0, 0}, {1, -3}, {2, 0}, {1, 1}}, {1, -1, 2, 0}},
  {"Block", {{2, 0}, {7, 0}},
   (R + 1 / (C * s)) / Rin, "Error Amp.", ElementSize -> {2.4, 2}},
  {"Amplifier", {{7, 0}, {9, 0}}, 2 * AP, "Power Amp."},
  {"Block", {{9, 0}, {14, 0}},
   1 / (Ra + s * La), "Motor", ElementSize -> {2.4, 2}},
  {"Line", {{14, 0}, {14, -3}}},
  {"Amplifier", {{8, -3}, {1, -3}}, AS, "Sense Amp."},
  {"Amplifier", {{14, -3}, {8, -3}}, RS},
  {"Output", {14, 0}, Y, "Armature Current"}
};
```

It is better typeset with

```
In[41]:= typoSubst = {AP -> Ap, AS -> As, La -> La, Ra -> Ra, Rin -> Rin, RS -> Rs};
```

```
In[42]:= ShowSchematic[CDmediaController /. typoSubst,
  PlotRange -> {{-1.5, 15.5}, {-6, 3}}];
```



ContinuousSystemTransferFunction computes the transfer function matrix of the system:

```
In[43]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[CDmediaController];
```

The transfer function of this single-input single-output system is the element of the transfer function matrix:

```
In[44]:= H = tfMatrix[[1, 1]];
  H /. typoSubst // TraditionalForm
```

```
Out[45]//TraditionalForm=
  2(CR s + 1) Ap
  -----
  C La Rin s^2 + C Ra Rin s + 2 C R Ap As Rs s + 2 Ap As Rs
```

This collects together terms involving the same powers of the complex frequency s :

```
In[46]:= Numerator[H] / Collect[Denominator[H], s] /. typoSubst //
  TraditionalForm
```

```
Out[46]//TraditionalForm=
  2(CR s + 1) Ap
  -----
  C La Rin s^2 + (C Ra Rin + 2 C R Ap As Rs) s + 2 Ap As Rs
```

The load impedance of Maker1 40X focus actuator is 18.5Ω and $228.5 \mu\text{H}$ at 1 kHz, 0.1 V. Consider the required system bandwidth of 60 kHz

```
In[47]:= wBandwidth = 2 * π * 60 * 103;
```

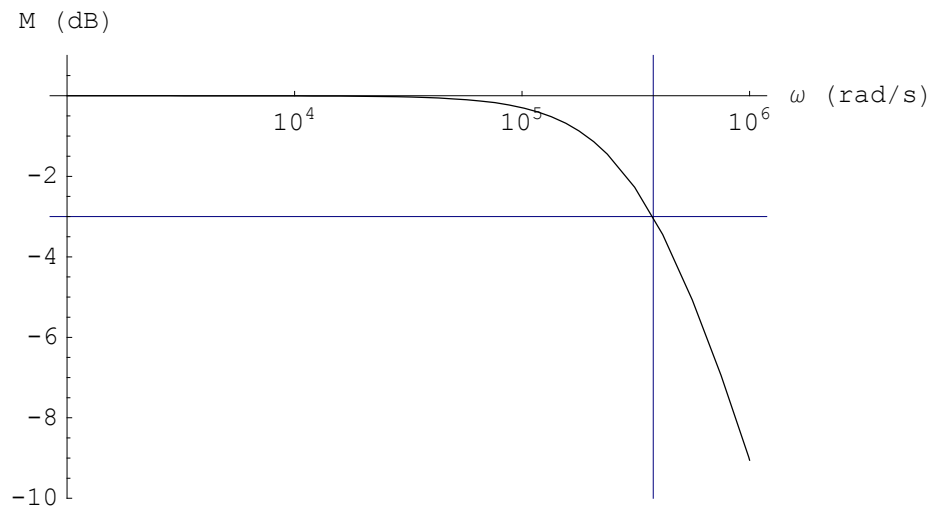
Assume the sensing resistor of 0.5Ω and other parameters as follows:

```
In[48]:= values = {RS → 0.5, AP → 2, AS → 2, C → 76.5 * 10(-12),  
La → 228.5 * 10(-6), R → 161.5 * 103, Ra → 18.5, Rin → 7.5 * 103};
```

Here is the magnitude response in terms of angular frequency:

```
In[49]:= M = 20 * Log[10, Abs[H /. s → I * w /. values]];
```

```
In[50]:= Plot[M /. w → 10n, {n, 3, 6},  
AxesLabel → {"ω (rad/s)", "M (dB)"},  
PlotRange → {Automatic, {-10, 1}},  
GridLines → {{Log[10, wBandwidth]}, {-3}},  
Ticks →  
{{3, "103"}, {4, "104"}, {5, "105"}, {6, "106"}, Automatic];
```

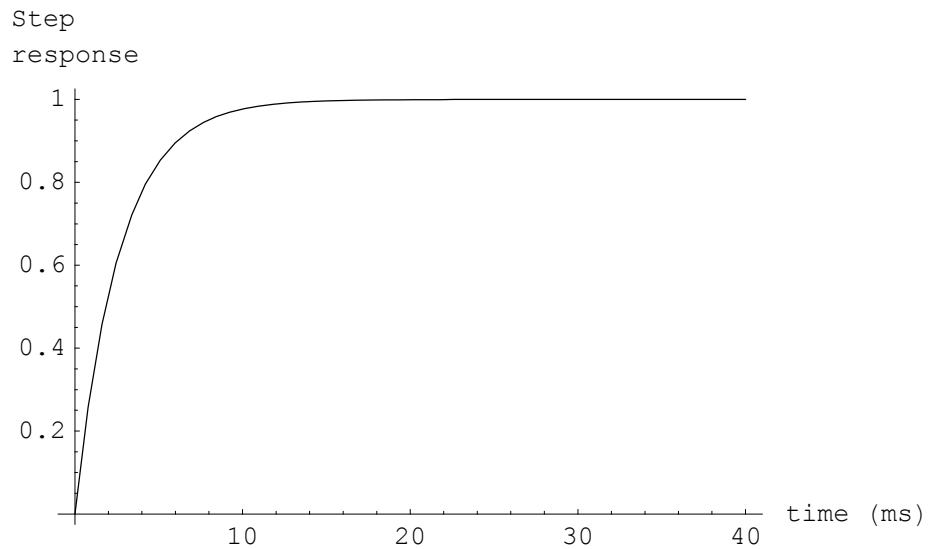


The step response follows:

```
In[51]:= stepResp = InverseLaplaceTransform[H / s /. values, s, t]
```

```
Out[51]= 4 (0.25 - 0.249976 e-376979. t - 0.0000239502 e-80934.4 t)
```

```
In[52]:= Plot[stepResp /. t -> x * 10^-6, {x, 0, 40},  
PlotRange -> All, AxesLabel -> {"time (ms)", "Step\nresponse"}];
```



Shuttle Pitch Control

Find the transfer function matrix of a pitch control MIMO system.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[53]:= Needs["SchematicSolver`"];
```

Here is an example of a *shuttle pitch control* system that incorporates feedback to control the pitch of a vehicle. Measurements are made by the vehicle's inertial unit, gyros and accelerometers. A simplified model of a pitch controller is shown for the space shuttle (Nise, N.S., *Control Systems Engineering*, 3/e, John Wiley and Sons, New York, NY, 2000):

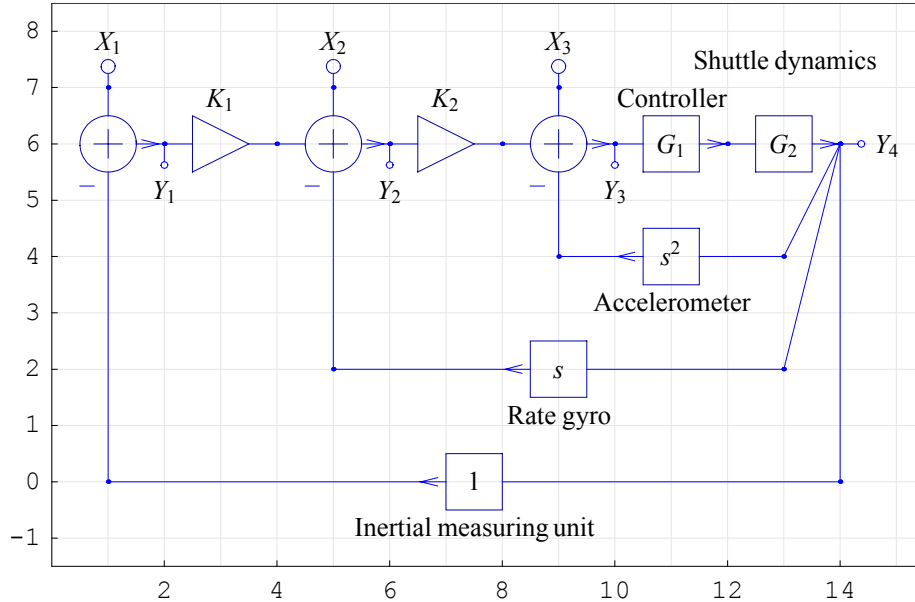
```
In[54]:= shuttlePitchController = {
  {"Input", {1, 7}, X1, "Commanded pitch", TextOffset → {0, -1}},
  {"Input", {5, 7}, X2,
   "Commanded pitch rate", TextOffset → {0, -1}},
  {"Input", {9, 7}, X3, "Commanded pitch acceleration",
   TextOffset → {0, -1}},
  {"Adder", {{0, 6}, {1, 0}, {2, 6}, {1, 7}}, {0, -1, 2, 1}},
  {"Adder", {{4, 6}, {5, 2}, {6, 6}, {5, 7}}, {1, -1, 2, 1}},
  {"Adder", {{8, 6}, {9, 4}, {10, 6}, {9, 7}}, {1, -1, 2, 1}},
  {"Amplifier", {{2, 6}, {4, 6}}, K1},
  {"Amplifier", {{6, 6}, {8, 6}}, K2},
  {"Block", {{10, 6}, {12, 6}}, G1, "Controller"},
  {"Block", {{12, 6}, {14, 6}}, G2, "Shuttle dynamics\n"},
  {"Block", {{14, 0}, {1, 0}}, 1, "Inertial measuring unit"},
  {"Block", {{13, 2}, {5, 2}}, s, "Rate gyro"},
  {"Block", {{13, 4}, {9, 4}}, s2, "Accelerometer"},
  {"Output", {2, 6}, Y1, "Pitch error", TextOffset → {0, 1}},
  {"Output", {6, 6}, Y2, "Pitch rate error", TextOffset → {0, 1}},
  {"Output", {10, 6}, Y3,
   "Pitch acceleration error", TextOffset → {0, 1}},
  {"Output", {14, 6}, Y4, "Pitch"},
  {"Line", {{14, 6}, {14, 0}}},
  {"Line", {{14, 6}, {13, 2}}},
  {"Line", {{14, 6}, {13, 4}}}
};
```

It is better typeset with

```
In[55]:= SetOptions[DrawElement, PlotStyle → DrawElementPlotStyleLight];
```

```
In[56]:= typoSubst = {G1 → G1, G2 → G2, K1 → K1, K2 → K2,
  X1 → X1, X2 → X2, X3 → X3,
  Y1 → Y1, Y2 → Y2, Y3 → Y3, Y4 → Y4};
```

```
In[57]:= ShowSchematic[shuttlePitchController /. typoSubst,
  PlotRange → {{0, 15.5}, {-1.5, 8.5}}];
```



ContinuousSystemTransferFunction computes the transfer function matrix of this three-input four-output system:

```
In[58]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[shuttlePitchController];
```

```
In[59]:= tfMatrix /. typoSubst // Together // TraditionalForm
```

Out[59]//TraditionalForm=

$$\begin{pmatrix} \frac{G_1 G_2 s^2 + G_1 G_2 K_2 s + 1}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & -\frac{G_1 G_2 K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & -\frac{G_1 G_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} \\ \frac{G_1 G_2 K_1 s^2 + K_1}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{G_1 G_2 s^2 + 1}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{-s G_1 G_2 - G_1 K_1 G_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} \\ \frac{K_1 K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{1}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} \\ \frac{G_1 G_2 K_1 K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{G_1 G_2 K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} & \frac{G_1 G_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1} \end{pmatrix}$$

System inputs are the commanded pitch X_1 , commanded pitch rate X_2 , and commanded pitch acceleration X_3 . They correspond to the following nodes, respectively:

```
In[60]:= systemInp
```

```
Out[60]= {Y[{1, 7}], Y[{5, 7}], Y[{9, 7}]}
```

System outputs are the pitch error Y_1 , pitch rate error Y_2 , pitch acceleration error Y_3 , and pitch Y_4 . They correspond to the following nodes, respectively:

```
In[61]:= systemOut
```

```
Out[61]= {Y[{2, 6}], Y[{6, 6}], Y[{10, 6}], Y[{14, 6}]}
```

Here is the transfer function from the commanded pitch input X_1 to the actual pitch output Y_4 :

```
In[62]:= H = tfMatrix[[4, 1]];
```

```
H /. typoSubst // TraditionalForm
```

```
Out[63]//TraditionalForm=
```

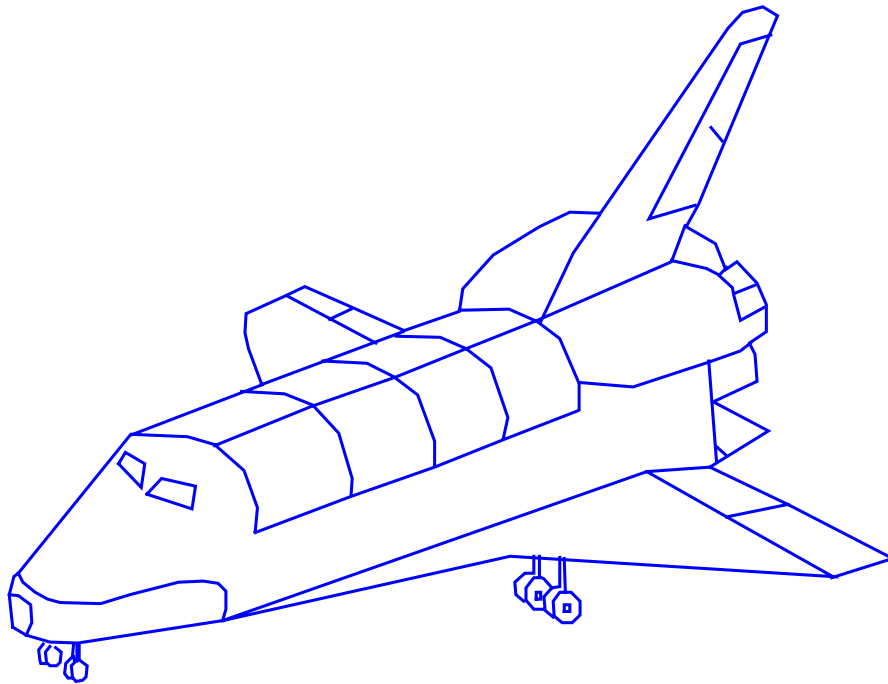
$$\frac{G_1 G_2 K_1 K_2}{G_1 G_2 s^2 + G_1 G_2 K_2 s + G_1 G_2 K_1 K_2 + 1}$$

SchematicSolver can be used to draw lineart figures. Let us restore the default plot style options:

```
In[64]:= SetOptions[DrawElement, PlotStyle → DrawElementPlotStyleDefault];
```

Here is a predefined specification to sketch the shuttle:

```
In[65]:= ShowSchematic[SchematicSolverFigureShuttle,  
                        Frame → False, GridLines → None];
```



You can add various annotations to the above lineart figure. For example, you can use the *SchematicSolver*'s Arrow element and Text element to add labels. The corresponding coordinates, you can obtain with the *SchematicSolver*'s palettes.

```

In[66]:= shuttleAnnotations = {
  {"Arrow", {{13, 61}, {-9, 42}}},
  {"Arrow", {{674, 228}, {734, 237}}},
  {"Arrow", {{243, 318}, {179, 380}}},
  {"Arrow", {{715, 138}, {768, 165}}},
  {"Arrow", {{648, 171}, {768, 165}}},
  {"Arrow", {{661, 281}, {734, 335}}},
  {"Arrow", {{622, 388}, {731, 353}}},
  {"Arrow", {{115, 195}, {29, 272}}},
  {"Arrow", {{505, 68}, {604, 54}}},
  {"Arrow", {{79, 6}, {116, 2}}},
  {"Arrow", {{482, 270}, {341, 406}}},
  {"Arrow", {{203, 241}, {328, 387}}},
  {"Arrow", {{637, 507}, {691, 513}}},
  {"Arrow", {{624, 541}, {538, 549}}},
  {"Text", {531, 550}, "Vertical tail", TextOffset → {1, 0}},
  {"Text", {698, 515},
   "Split rudder\n speed brake", TextOffset → {-1, 0}},
  {"Text", {771, 166}, "Elevons", TextOffset → {-1, 0}},
  {"Text", {747, 350}, "Engines", TextOffset → {-1, 0}},
  {"Text", {333, 430}, "Payload doors"},
  {"Text", {167, 400}, "Delta wing"},
  {"Text", {27, 283}, "Flight deck"},
  {"Text", {612, 48}, "Main landing gear", TextOffset → {-1, 0}},
  {"Text", {122, 5}, "Nose landing gear", TextOffset → {-1, 0}},
  {"Text", {746, 243}, "Body flap", TextOffset → {-1, 0}},
  {"Text", {-11, 34}, "Nose\n cone", TextOffset → {1, 0}}
};

```

It is better typeset with

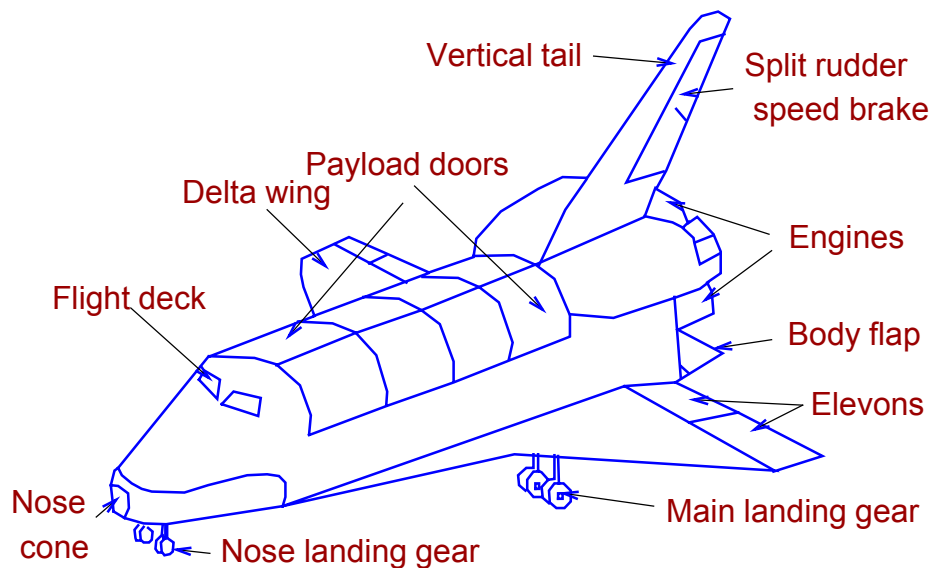
```

In[67]:= SetOptions[DrawElement, TextStyle →
  {FontFamily → "Helvetica", FontColor → RGBColor[0.6, 0, 0]}
];

```

Use `Join` to combine the predefined lineart figure and your annotations:

```
In[68]:= ShowSchematic[
  SchematicSolverFigureShuttle~Join~shuttleAnnotations,
  Frame → False, GridLines → None, ElementScale → 50]
```



Body flap – a flap located at the bottom rear of the shuttle; it provides a thermal shield for the engines during re-entry and also provides pitch control (the movement of the nose up and down) during atmospheric flight (landings).

Delta wing – the triangular-shaped wings on either side of the Space Shuttle.

Elevons – flaps located on the trailing edge of each wing, used during atmospheric flight; elevons are used to control pitch (the movement of the nose up and down) and roll. The elevons work only in the presence of air (they do not work in space).

Engines – engines are located at the rear of the shuttle and are used to maneuver the Space Shuttle into orbit, to make adjustments while in orbit, and to control the Space Shuttle during re-entry into the Earth's atmosphere: controlling the roll, pitch (the movement of the nose up and down), and yaw (the movement of the nose to the left and right) in the absence of air.

Flight deck – the part of the Space Shuttle in which the astronauts travel.

Main landing gear – the landing gear (wheels used for landing) located at the rear of the Space Shuttle.

Nose cone – the front of the Space Shuttle.

Nose landing gear – the landing gear (wheels used for landing) located at the front of the Space Shuttle.

Payload doors – the doors of the large storage compartment of the Space Shuttle; items like satellites can be carried into space in the cargo (payload) bay.

Split rudder/speed brake – a divided flap located on the trailing edge of the tail fin, used during atmospheric flight. This two-part rudder steers the shuttle, controls yaw (the movement of the nose to the left and right), and acts as a brake (when the split rudder is opened like book). The rudder works only in the presence of air.

Vertical tail – the fin at the top rear of the shuttle. It provides stability while flying.

This restores default drawing options:

```
In[69]:= SetOptions[DrawElement,  
             TextStyle → {FontFamily → "Times", FontColor → RGBColor[0, 0, 0]}  
           ];
```

■ 5.2. Symbolic Optimization of a Continuous-Time System

Find the optimal value of a selected system parameter for a given value of the steady-state response.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[70]:= Needs["SchematicSolver`"];
```

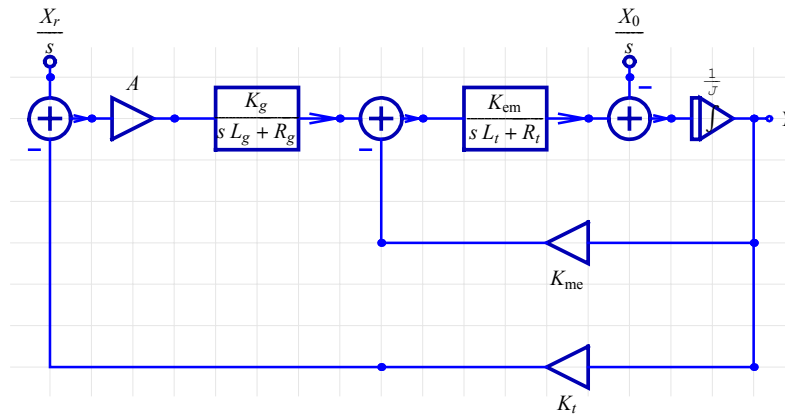
Consider a continuous-time *velocity servo system* represented by the following schematic:

```
In[71]:= velocityServoSchematic = {
  {"Line", {{18, 3}, {18, 0}}},
  {"Line", {{18, 6}, {18, 3}}},
  {"Adder", {{0, 6}, {9, 0}, {2, 6}, {1, 7}}, {0, -1, 2, 1}},
  {"Adder", {{8, 6}, {9, 3}, {10, 6}, {9, 7}}, {1, -1, 2, 0}},
  {"Adder", {{14, 6}, {15, 5}, {16, 6}, {15, 7}}, {1, 0, 2, -1}},
  {"Amplifier", {{2, 6}, {4, 6}}, A},
  {"Amplifier", {{18, 0}, {9, 0}}, Kt},
  {"Amplifier", {{18, 3}, {9, 3}}, Kme},
  {"Input", {1, 7}, Xr/s, "", TextOffset → {0, -1}},
  {"Integrator", {{16, 6}, {18, 6}}, 1/J},
  {"Output", {18, 6}, Y},
  {"Block", {{4, 6}, {8, 6}},
  Kg/(Rg + s*Lg), "", ElementSize → {2, 1.5}},
  {"Block", {{10, 6}, {14, 6}}, Kem/(Rt + s*Lt),
  "", ElementSize → {2, 1.5}},
  {"Input", {15, 7}, X0/s, "", TextOffset → {0, -1}}
};
```

It is better typeset with

```
In[72]:= typoSubst = {Kem → Kem, Kg → Kg, Kme → Kme, Kt → Kt,
  Lg → Lg, Lt → Lt, Rg → Rg, Rt → Rt,
  X0 → X0, Xr → Xr, Yref → Yref};
```

```
In[73]:= ShowSchematic[velocityServoSchematic /. typoSubst,
  FontSize -> 8, Frame -> False];
```



The system has two inputs and one output. Both inputs are step stimuli whose transforms are $\frac{X_r}{s}$ and $\frac{X_0}{s}$.

ContinuousSystemTransferFunction computes the transfer function matrix of the system:

```
In[74]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[velocityServoSchematic];
```

The transfer functions of this two-input single-output system are the elements of the transfer function matrix:

```
In[75]:= H1 = tfMatrix[[1, 1]] // Together;
  % /. typoSubst // TraditionalForm
```

```
Out[76]//TraditionalForm=
```

$$\frac{(A K_{em} K_g)}{(J L_g L_t s^3 + J L_t R_g s^2 + J L_g R_t s^2 + K_{em} K_{me} L_g s + J R_g R_t s + A K_{em} K_g K_t + K_{em} K_{me} R_g)}$$

```
In[77]:= H2 = tfMatrix[[1, 2]] // Together;
  % /. typoSubst // TraditionalForm
```

```
Out[78]//TraditionalForm=
```

$$\frac{-((s L_g + R_g)(s L_t + R_t))}{(J L_g L_t s^3 + J L_t R_g s^2 + J L_g R_t s^2 + K_{em} K_{me} L_g s + J R_g R_t s + A K_{em} K_g K_t + K_{em} K_{me} R_g)}$$

ContinuousSystemResponse finds the system response at all nodes:

```
In[79]:= {systemResponse, systemVars} =
  ContinuousSystemResponse[velocityServoSchematic];
```

Here is the transform of the output signal:

```
In[80]:= Yout = systemOut[[1]] /. systemResponse // Together;
         % /. typoSubst // TraditionalForm
```

```
Out[81]//TraditionalForm=
```

$$\frac{(-L_g L_t X_0 s^2 - L_t R_g X_0 s - L_g R_t X_0 s - R_g R_t X_0 + A K_{em} K_g X_r)}{(s (J L_g L_t s^3 + J L_t R_g s^2 + J L_g R_t s^2 + K_{em} K_{me} L_g s + J R_g R_t s + A K_{em} K_g K_t + K_{em} K_{me} R_g))}$$

SchematicSolver has a unique feature: it finds the symbolic response keeping all system parameters as symbols. The response Y_{out} is closed-form and purely symbolic. All system parameters are given by symbols, so the obtained result is the most general.

You can find the optimal value of a selected parameter for the given steady-state value. For instance, the gain of the amplifier A can be optimized to provide the output steady state of some value Y_{ref} . Notice that no numerical value appears in the calculation.

```
In[82]:= Aopt = A /. First[Solve[Limit[s * Yout, s → 0] == Yref, A]];
         % /. typoSubst // TraditionalForm
```

```
Out[83]//TraditionalForm=
```

$$\frac{R_g R_t X_0 + K_{em} K_{me} R_g Y_{ref}}{K_{em} K_g (X_r - K_t Y_{ref})}$$

For a particular set of numeric values

```
In[84]:= parameterValues =
         {J → 0.005, Kem → 0.0005, Kg → 20, Kme → 0.0005, Kt → 0.1, Lg → 10,
         Lt → 0.020, Rg → 25, Rt → 1.6, X0 → 2.5, Xr → 150, Yref → 150};
```

the optimum gain becomes

```
In[85]:= Aopt /. parameterValues
```

```
Out[85]= 74.0748
```

Substituting the numeric values into the symbolic expression Y_{out} yields

```
In[86]:= numericYout = Yout /. A → Aopt /. parameterValues
```

$$Out[86]= \frac{11.1122 - 41.25 s - 0.5 s^2}{s (0.074081 + 0.200003 s + 0.0825 s^2 + 0.001 s^3)}$$

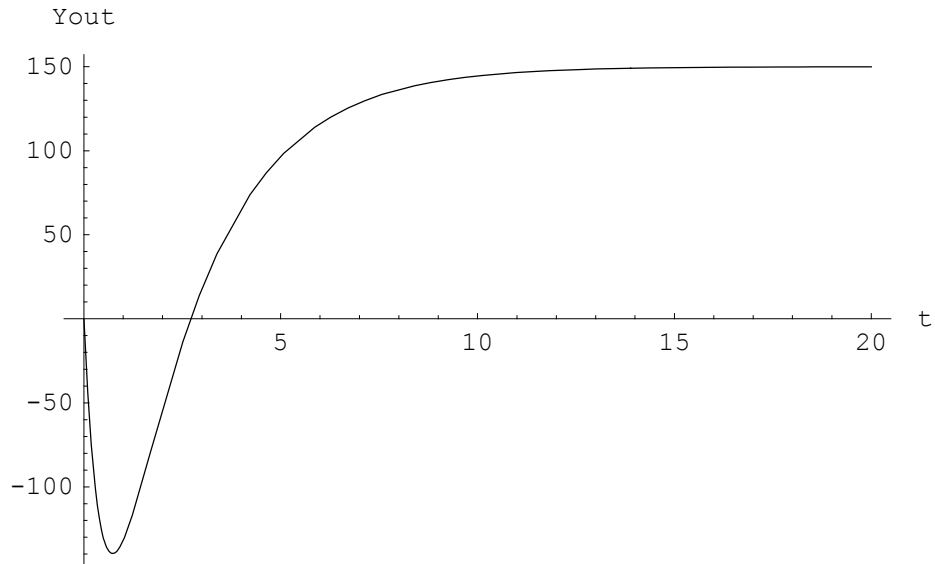
Using the inverse Laplace transform provided by *Mathematica* the time response can be found:

```
In[87]:= timeYout = InverseLaplaceTransform[numericYout, s, t]
```

```
Out[87]= 150. - 0.222917 e-80.0119 t + 371.62 e-2.03257 t - 521.397 e-0.45552 t
```

The corresponding graph of this waveform proves the expected steady state value:

```
In[88]:= Plot[timeYout, {t, 0, 20}, AxesLabel -> {"t", "Yout"}];
```



■ 5.3. Design of a Continuous-Time System from the Step Response

Linear system can be designed in a straightforward manner if its step response is known as a closed-form expression.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[89]:= Needs["SchematicSolver`"];
```

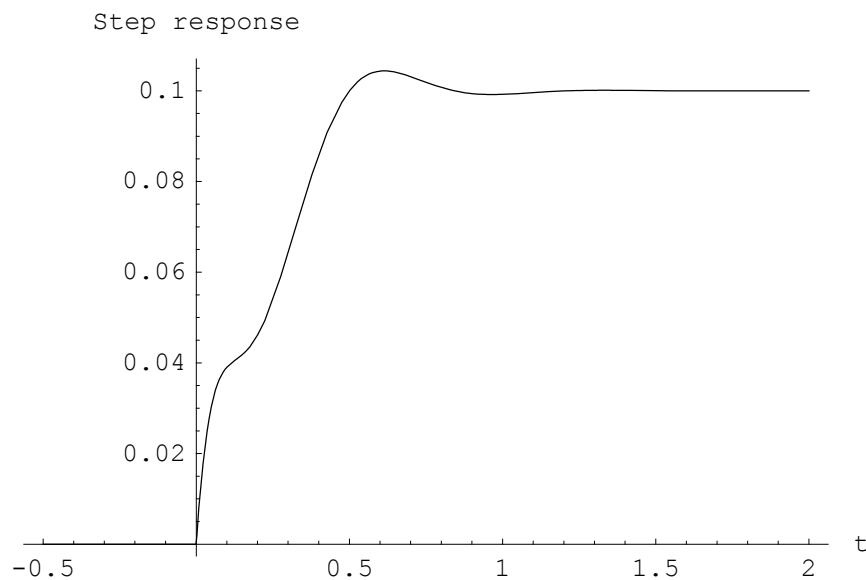
Here is the step response of a linear continuous-time system:

```
In[90]:= stepResponse =
  (  $\frac{1}{10} - \frac{1}{5} e^{-10t} + \frac{1}{30} e^{-5t} (3 \cos[5\sqrt{3}t] - \sqrt{3} \sin[5\sqrt{3}t])$  )
  UnitStep[t];
  % // TraditionalForm
```

```
Out[91]//TraditionalForm=
```

$$\left(\frac{1}{30} e^{-5t} (3 \cos(5\sqrt{3}t) - \sqrt{3} \sin(5\sqrt{3}t)) - \frac{e^{-10t}}{5} + \frac{1}{10} \right) \theta(t)$$

```
In[92]:= Plot[stepResponse, {t, -0.5, 2},
  PlotRange → All, AxesLabel → {"t", "Step response"}];
```



First, find the corresponding transfer function from the formula $r(t) = \mathcal{L}^{-1} \frac{1}{s} H(s)$, or equivalently, $H(s) = s \mathcal{L} r(t)$, where \mathcal{L} represents the Laplace transform, $r(t)$ denotes the step response, $H(s)$ is the transfer function, and s stands for the complex frequency:

```
In[93]:= stepResponseLT = LaplaceTransform[stepResponse, t, s]
Out[93]=  $\frac{1}{10 s} - \frac{1}{5 (10 + s)} - \frac{1}{2 (100 + 10 s + s^2)} + \frac{5 + s}{10 (100 + 10 s + s^2)}$ 
In[94]:= transferFunction = s * stepResponseLT // Together;
% // TraditionalForm
Out[95]//TraditionalForm=

$$\frac{s^2 + 100}{(s + 10)(s^2 + 10 s + 100)}$$

```

Second, design the system from the transfer function. Minimal number of integrators equals the order of the transfer function

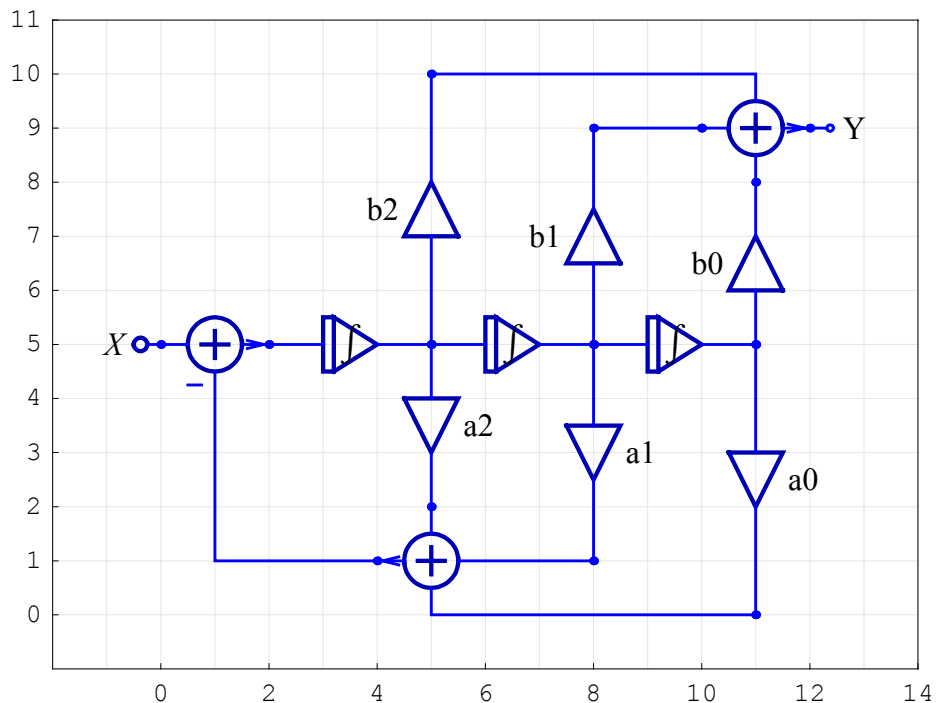
```
In[96]:= numTF = Numerator[transferFunction]
Out[96]=  $100 + s^2$ 
In[97]:= denTF = Denominator[transferFunction] // Expand
Out[97]=  $1000 + 200 s + 20 s^2 + s^3$ 
In[98]:= numberOfIntegrators = Max[Exponent[numTF, s], Exponent[denTF, s]]
Out[98]= 3
```

The general block-diagram of this system, with symbolic parameters and 3 integrators, can be represented by the following schematic:

```

In[99]:= generalSchematic3 = {
  {"Input", {0, 5}, X},
  {"Output", {12, 9}, "Y"},
  {"Integrator", {{2, 5}, {5, 5}}, 1},
  {"Integrator", {{5, 5}, {8, 5}}, 1},
  {"Integrator", {{8, 5}, {11, 5}}, 1},
  {"Amplifier", {{5, 5}, {5, 2}}, a2},
  {"Amplifier", {{8, 5}, {8, 1}}, a1},
  {"Amplifier", {{11, 5}, {11, 0}}, a0},
  {"Amplifier", {{5, 5}, {5, 10}}, b2},
  {"Amplifier", {{8, 5}, {8, 9}}, b1},
  {"Amplifier", {{11, 5}, {11, 8}}, b0},
  {"Adder", {{0, 5}, {4, 1}, {2, 5}, {1, 6}}, {1, -1, 2, 0}},
  {"Adder", {{4, 1}, {11, 0}, {8, 1}, {5, 2}}, {2, 1, 1, 1}},
  {"Adder", {{10, 9}, {11, 8}, {12, 9}, {5, 10}}, {1, 1, 2, 1}},
  {"Line", {{8, 9}, {10, 9}}};
ShowSchematic[%, PlotRange -> {{-2, 14}, {-1, 11}}];

```



`ContinuousSystemTransferFunction` computes the transfer function matrix of the system:

```

In[101]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[generalSchematic3];

```

The transfer function of this single-input single-output system is the element of the transfer function matrix:

```
In[102]:= H = tfMatrix[[1, 1]]
```

$$\text{Out[102]} = \frac{b_0 + b_1 s + b_2 s^2}{a_0 + a_1 s + a_2 s^2 + s^3}$$

Numeric coefficient values are found as follows.

Numerator picks out the numerator of the transfer function:

```
In[103]:= numH = Numerator[H]
```

$$\text{Out[103]} = b_0 + b_1 s + b_2 s^2$$

Denominator picks out the denominator of the transfer function:

```
In[104]:= denH = Denominator[H]
```

$$\text{Out[104]} = a_0 + a_1 s + a_2 s^2 + s^3$$

CoefficientList finds a list of the coefficients of polynomials:

```
In[105]:= CoefficientList[denH, s]
```

$$\text{Out[105]} = \{a_0, a_1, a_2, 1\}$$

Note that the leading coefficient equals 1.

For the known transfer function coefficients, that are computed from the step response, and for the symbolic coefficients, that are computed from the general schematic of the system, we compute the system parameters as follows:

```
In[106]:= numParameters =
  Solve[CoefficientList[numTF, s] == CoefficientList[numH, s],
    {b0, b1, b2}] // Flatten
```

$$\text{Out[106]} = \{b_0 \rightarrow 100, b_1 \rightarrow 0, b_2 \rightarrow 1\}$$

```
In[107]:= denParameters =
  Solve[CoefficientList[denTF, s] == CoefficientList[denH, s],
    {a0, a1, a2}] // Flatten
```

$$\text{Out[107]} = \{a_0 \rightarrow 1000, a_1 \rightarrow 200, a_2 \rightarrow 20\}$$

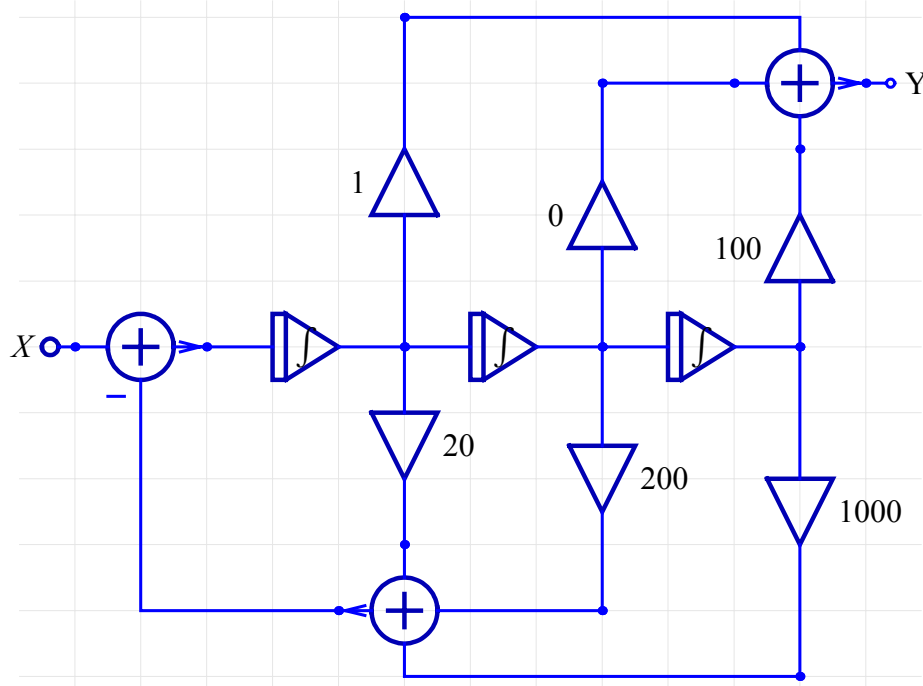
```
In[108]:= systemParameters = Join[numParameters, denParameters]
```

```
Out[108]= {b0 → 100, b1 → 0, b2 → 1, a0 → 1000, a1 → 200, a2 → 20}
```

The schematic specification of the system with numeric parameters is

```
In[109]:= numericSchematic3 = generalSchematic3 /. systemParameters;
```

```
In[110]:= ShowSchematic[numericSchematic3, Frame → False];
```



`ContinuousSystemTransferFunction` finds the transfer function of the above system:

```
In[111]:= {tfMatrix, systemInp, systemOut} =
           ContinuousSystemTransferFunction[numericSchematic3];
           numericH = tfMatrix[[1, 1]]
```

```
Out[112]=  $\frac{100 + s^2}{1000 + 200s + 20s^2 + s^3}$ 
```

The corresponding step response can be computed as the inverse Laplace transform of `numericH/s`:

```
In[113]:= numericStepResponse =
           InverseLaplaceTransform[numericH/s, s, t] // Simplify
Out[113]=  $\frac{1}{30} (3 - 6 e^{-10t} + e^{-5t} (3 \cos[5\sqrt{3} t] - \sqrt{3} \sin[5\sqrt{3} t]))$ 
```

FullSimplify proves that, for positive time, the step response computed from the schematic is the same as the given step response:

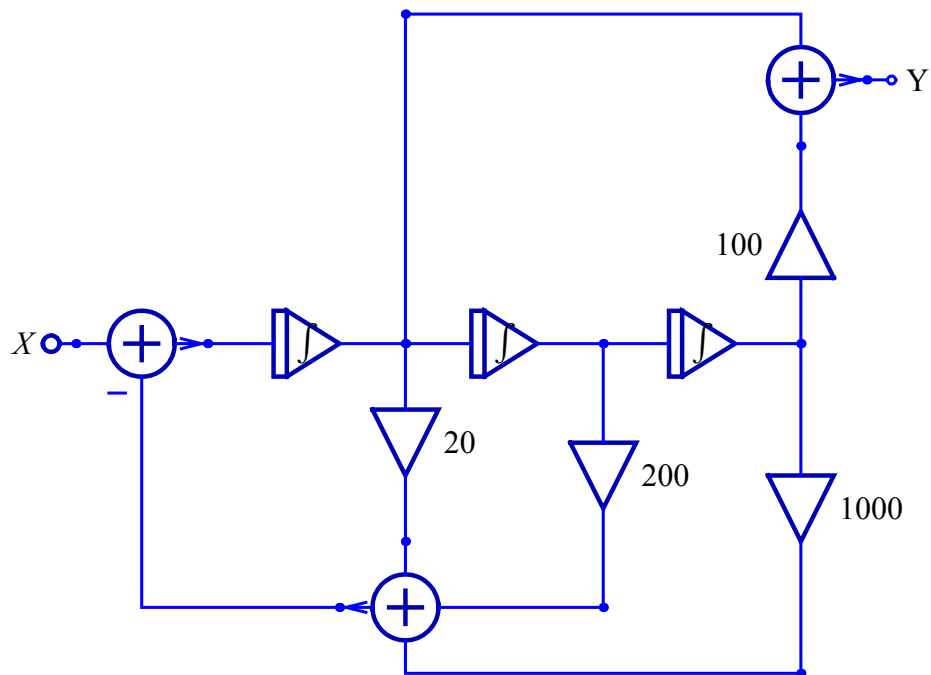
```
In[114]:= FullSimplify[stepResponse - numericStepResponse, t > 0]
Out[114]= 0
```

The system representation can be simplified by

- removing the Amplifier elements with zero gain
- replacing the unity-gain Amplifier elements with the Line elements.

```
In[115]:= simpleSchematic3 = {
           {"Input", {0, 5}, X},
           {"Output", {12, 9}, "Y"},
           {"Integrator", {{2, 5}, {5, 5}}, 1},
           {"Integrator", {{5, 5}, {8, 5}}, 1},
           {"Integrator", {{8, 5}, {11, 5}}, 1},
           {"Amplifier", {{5, 5}, {5, 2}}, a2},
           {"Amplifier", {{8, 5}, {8, 1}}, a1},
           {"Amplifier", {{11, 5}, {11, 0}}, a0},
           {"Line", {{5, 5}, {5, 10}}},
           {"Amplifier", {{11, 5}, {11, 8}}, b0},
           {"Adder", {{0, 5}, {4, 1}, {2, 5}, {1, 6}}, {1, -1, 2, 0}},
           {"Adder", {{4, 1}, {11, 0}, {8, 1}, {5, 2}}, {2, 1, 1, 1}},
           {"Adder", {{10, 9}, {11, 8}, {12, 9}, {5, 10}}, {0, 1, 2, 1}};
```

```
In[116]:= ShowSchematic[simpleSchematic3 /. systemParameters,
  Frame -> False, GridLines -> None];
```



Obviously, the transfer function remains the same:

```
In[117]:= {tfMatrix, systemInp, systemOut} =
  ContinuousSystemTransferFunction[simpleSchematic3];
simpleH = tfMatrix[[1, 1]] /. systemParameters
```

```
Out[118]= 
$$\frac{100 + s^2}{1000 + 200s + 20s^2 + s^3}$$

```

```
In[119]:= SameQ[numericH, simpleH]
```

```
Out[119]= True
```

■ 5.4. Automated Drawing and Solving of General Systems

SchematicSolver comes with functions that create schematics important for practice. You can easily build new models from these automatically generated schematics.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[120]:= Needs["SchematicSolver`"];
```

We shall adjust some options to obtain better appearance of the example schematics:

```
In[121]:= SetOptions[ShowSchematic, Frame → False, GridLines → None];
```

Typically, for the specified system order

```
In[122]:= systemOrder = 3;
```

the system parameters can be generated automatically with

```
In[123]:= paramsNum =
          UnitSymbolicSequence[systemOrder, a, 0] // Flatten // Reverse
```

```
Out[123]= {a2, a1, a0}
```

```
In[124]:= paramsDen =
          UnitSymbolicSequence[systemOrder + 1, b, 0] // Flatten // Reverse
```

```
Out[124]= {b3, b2, b1, b0}
```

Here is an example schematic specification of a discrete system that is generated automatically for the specified system parameters:

```
In[125]:= {schematicSpec, inputCoordinates, outputCoordinates} =
          TransposedDirectForm2IIRFilterSchematic[{paramsDen, paramsNum}];
```

You should add input and output to form the system:

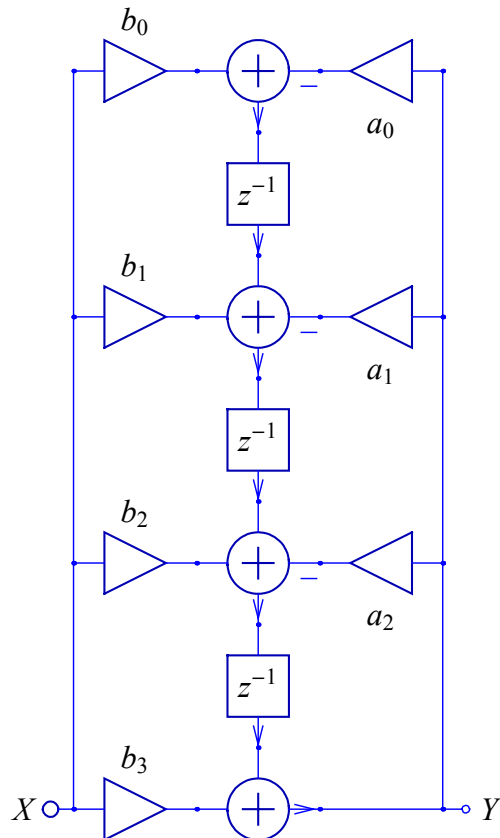
```
In[126]:= discreteSchematic = Join[schematicSpec, {
          {"Input", inputCoordinates[[1]], X},
          {"Output", outputCoordinates[[1]], Y}
        }];
```

Note that the coordinates of input and output have been returned by `DirectFormFIRFilterSchematic`.

For better typesetting, you may use

```
In[127]:= typoSubst =
      {a0 → a0, a1 → a1, a2 → a2, b0 → b0, b1 → b1, b2 → b2, b3 → b3};
```

```
In[128]:= ShowSchematic[discreteSchematic /. typoSubst]
```



`DiscreteSystemTransferFunction` finds the transfer function directly from the schematic:

```
In[129]:= {tfMatrix, myInputs, myOutputs} =
      DiscreteSystemTransferFunction[discreteSchematic];
      % /. typoSubst // DiscreteSystemDisplayForm
```

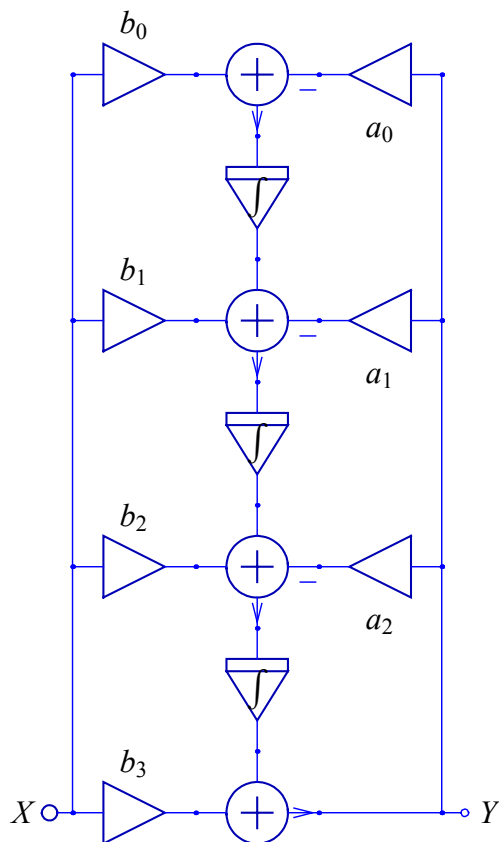
```
Out[130]//DisplayForm=
      
$$\frac{b_3 + b_2 z^{-1} + b_1 z^{-2} + b_0 z^{-3}}{1 + a_2 z^{-1} + a_1 z^{-2} + a_0 z^{-3}}$$

```

The graphical representation of a system is not a frozen picture. Once when you have a schematic of the discrete-time system, and when you find out that the same structure can be used to build a schematic of a continuous-time system, you can do that by the following simple replacements:

- the Delay element is replaced by the Integrator element
- the Multiplier element is replaced by the Amplifier element

```
In[131]:= continuousSchematic = discreteSchematic /.
          {"Delay" → "Integrator", "Multiplier" → "Amplifier"};
          % /. typoSubst // ShowSchematic
```



ContinuousSystemTransferFunction finds the transfer function directly from the schematic:

```
In[133]:= {tfMatrix, myInputs, myOutputs} =  
          ContinuousSystemTransferFunction[continuousSchematic];  
          tfMatrix[[1, 1]] /. typoSubst // Together // TraditionalForm
```

```
Out[134]//TraditionalForm=
```

$$\frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0}$$

Automated drawing and solving of general continuous-time and discrete-time systems is a unique feature of *SchematicSolver* not available in other software for system modeling and analysis.

This restores default drawing options:

```
In[135]:= SetOptions[ShowSchematic, Frame → True, GridLines → Automatic];
```

■ 5.5. Discrete-Time Systems

Introduction

SchematicSolver has many unique features not available in other software: symbolic signal processing brings you

- Computation of transfer functions as closed-form expressions in terms of symbolic system parameters
- Finding the closed-form response from the schematic

The derived result is the most general because all system parameters and inputs can be given by symbols.

Other important features include building models from automatically generated schematics; you can change system parameters on the fly and immediately see what happens with the results.

See other chapters for illustrations of unique features not available in other software:

Chapter 6 Solving Large Systems

Chapter 9 Examples of Discrete System Implementation

Chapter 10 Hilbert Transformer

Chapter 11 Multirate Systems

Chapter 12 Hierarchical Systems

Chapter 15 Processing with *SchematicSolver*

SchematicSolver's powerful functions for solving discrete-time (digital) systems are illustrated by the subsequent examples.

Direct Form 2 Transposed IIR Filter

Find the transfer function of a digital filter realization known as *direct form 2 transposed IIR*.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[136]:= Needs["SchematicSolver`"];
```

Here is the schematic:

```
In[137]:= DirectForm2TransposedIIRSchematic = {
  {"Input", {0, 15}, X},
  {"Output", {8, 15}, Y},
  {"Multiplier", {{8, 10}, {5, 10}}, a1},
  {"Multiplier", {{8, 5}, {5, 5}}, a2},
  {"Multiplier", {{8, 0}, {5, 0}}, a3},
  {"Multiplier", {{0, 15}, {3, 15}}, b0},
  {"Multiplier", {{0, 10}, {3, 10}}, b1},
  {"Multiplier", {{0, 5}, {3, 5}}, b2},
  {"Multiplier", {{0, 0}, {3, 0}}, b3},
  {"Delay", {{4, 1}, {4, 4}}, 1},
  {"Delay", {{4, 6}, {4, 9}}, 1},
  {"Delay", {{4, 11}, {4, 14}}, 1},
  {"Adder", {{3, 0}, {4, -1}, {5, 0}, {4, 1}}, {1, 0, -1, 2}},
  {"Adder", {{3, 5}, {4, 4}, {5, 5}, {4, 6}}, {1, 1, -1, 2}},
  {"Adder", {{3, 10}, {4, 9}, {5, 10}, {4, 11}}, {1, 1, -1, 2}},
  {"Adder", {{3, 15}, {4, 14}, {5, 15}, {4, 16}}, {1, 1, 2, 0}},
  {"Line", {{0, 5}, {0, 0}}, {"Line", {{0, 10}, {0, 5}}},
  {"Line", {{0, 15}, {0, 10}}, {"Line", {{8, 15}, {5, 15}}},
  {"Line", {{8, 5}, {8, 0}}, {"Line", {{8, 10}, {8, 5}}},
  {"Line", {{8, 15}, {8, 10}}}
};
```

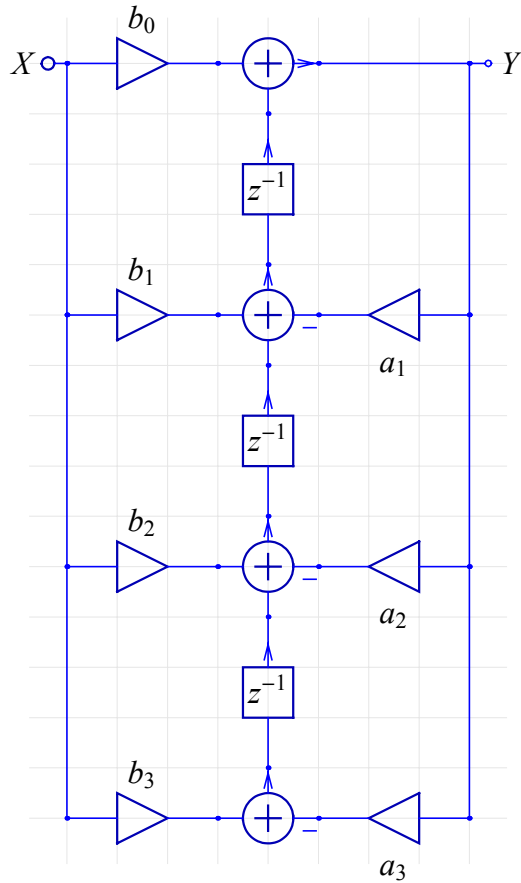
It is better typeset with

```
In[138]:= typoSubst =
```

```
{a1 → a1, a2 → a2, a3 → a3, b0 → b0, b1 → b1, b2 → b2, b3 → b3};
```

```
In[139]:= ShowSchematic[
```

```
DirectForm2TransposedIIRschematic /. typoSubst, Frame → False];
```



`DiscreteSystemTransferFunction` computes the transfer function matrix of the system:

```
In[140]:= {tfMatrix, systemInp, systemOut} = DiscreteSystemTransferFunction[
    DirectForm2TransposedIIRSchematic];
```

The transfer function of this single-input single-output system is the element of the transfer function matrix:

```
In[141]:= df2TF = tfMatrix[[1, 1]];
    df2TF /. typoSubst // Together // TraditionalForm
```

```
Out[142]//TraditionalForm=
```

$$\frac{b_0 z^3 + b_1 z^2 + b_2 z + b_3}{z^3 + a_1 z^2 + a_2 z + a_3}$$

SchematicSolver can express transfer functions of discrete systems in terms of z^{-1} with its function `DiscreteSystemDisplayForm`:

```
In[143]:= DiscreteSystemDisplayForm[df2TF /. typoSubst]
```

```
Out[143]//DisplayForm=
```

$$\frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

State-Space Model of Discrete System

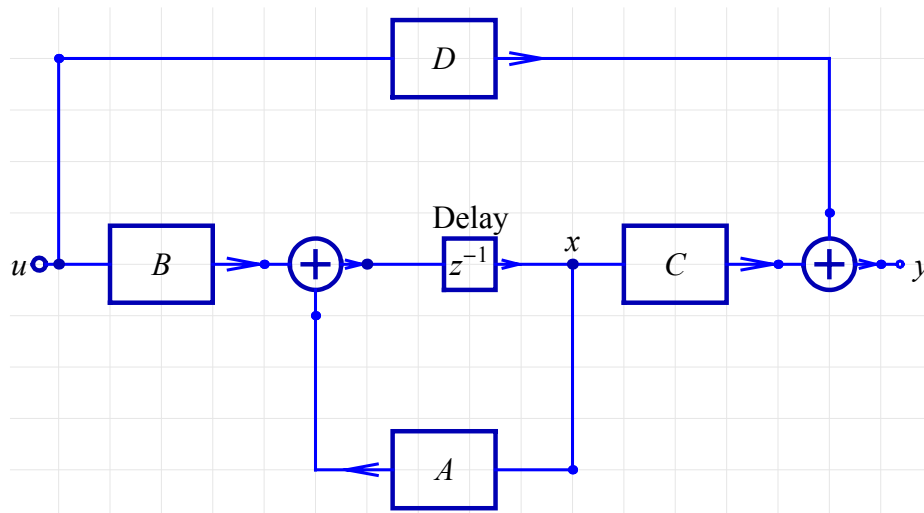
Compute the transfer function for the *state-space model* of the discrete-time system shown below.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[144]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[145]:= CSPFigure3p2 = {
  {"Input", {0, 4}, u},
  {"Output", {16, 4}, y},
  {"Block", {{10, 0}, {5, 3}}, A, "", ElementSize -> {2, 1.5}},
  {"Block", {{0, 4}, {4, 4}}, B, "", ElementSize -> {2, 1.5}},
  {"Block", {{10, 4}, {14, 4}}, C, "", ElementSize -> {2, 1.5}},
  {"Block", {{0, 8}, {15, 5}}, D, "", ElementSize -> {2, 1.5}},
  {"Delay", {{6, 4}, {10, 4}}, 1, "Delay"},
  {"Adder", {{4, 4}, {5, 3}, {6, 4}, {5, 5}}, {1, 1, 2, 0}},
  {"Adder", {{14, 4}, {15, 3}, {16, 4}, {15, 5}}, {1, 0, 2, 1}},
  {"Line", {{0, 4}, {0, 8}}}, {"Line", {{10, 4}, {10, 0}}},
  {"Node", {0, 4}, ""}, {"Node", {6, 4}, ""},
  {"Node", {10, 4}, x, "", TextOffset -> {0, -1}}
};
ShowSchematic[%, Frame -> False];
```



DiscreteSystemTransferFunction computes the transfer function matrix of the system:

```
In[147]:= DiscreteSystemTransferFunction[CSPFigure3p2] //  
          DiscreteSystemDisplayForm
```

```
Out[147]//DisplayForm=
```

$$\frac{-D + (-B C + A D) z^{-1}}{-1 + A z^{-1}}$$

Unity Feedback System

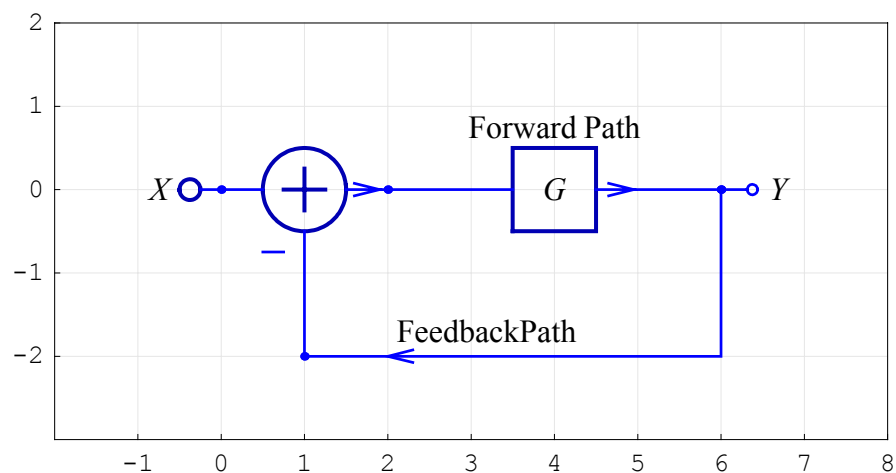
A *unity feedback system* is a feedback system in which the primary feedback is identically equal to the controlled output. Find the response of the system.

This section assumes that you have already loaded *SchematicSolver*. Otherwise, you can load the package with

```
In[148]:= Needs["SchematicSolver`"];
```

Here is the system schematic:

```
In[149]:= unityFeedbackSystem = {
  {"Input", {0, 0}, X},
  {"Output", {6, 0}, Y},
  {"Block", {{2, 0}, {6, 0}}, G, "Forward Path"},
  {"Arrow", {{2, -2}, {6, -2}},
  "FeedbackPath", ShowArrowTail → False},
  {"Adder", {{0, 0}, {1, -2}, {2, 0}, {1, 1}}, {1, -1, 2, 0}},
  {"Line", {{6, 0}, {6, -2}, {1, -2}}}
};
ShowSchematic[%, PlotRange → {{-2, 8}, {-3, 2}}];
```



DiscreteSystemEquations sets up the equations of the system:

```
In[151]:= {unityFeedbackEquations, vars} =
  DiscreteSystemEquations[unityFeedbackSystem];
```

It is better typeset with

```
In[152]:= typoSubstYkn = {Y[{k_Integer, n_Integer}] => Y_{k,n}};
```

```
In[153]:= unityFeedbackEquations /. typoSubstYkn // ColumnForm //
TraditionalForm
```

```
Out[153]//TraditionalForm=
```

$$Y_{0,0} == X$$

$$Y_{6,0} == G Y_{2,0}$$

$$Y_{2,0} == Y_{0,0} - Y_{6,0}$$

DiscreteSystemResponse finds the response of the system:

```
In[154]:= {unityFeedbackResponse, vars} =
DiscreteSystemResponse[unityFeedbackSystem];
```

```
In[155]:= unityFeedbackResponse /. typoSubstYkn // ColumnForm //
TraditionalForm
```

```
Out[155]//TraditionalForm=
```

$$Y_{6,0} \rightarrow \frac{GX}{G+1}$$

$$Y_{2,0} \rightarrow \frac{X}{G+1}$$

$$Y_{0,0} \rightarrow X$$

DiscreteSystemTransferFunction computes the transfer function matrix of the system:

```
In[156]:= {tfMatrix, systemInp, systemOut} =
DiscreteSystemTransferFunction[unityFeedbackSystem];
```

The transfer function of this single-input single-output system is the element of the transfer function matrix:

```
In[157]:= unityFeedbackTF = tfMatrix[[1, 1]];
% // TraditionalForm
```

```
Out[158]//TraditionalForm=
```

$$\frac{G}{G+1}$$