Using Laplace Approach

Consider the previous example again:

$$\ddot{x} + 3\dot{x} + 2x = 5\sin t$$

$$x(0) = 1$$
$$\dot{x}(0) = 0$$

Apply the Laplace transform to given diff. eqn

$$[s^2 X(s) - sx(0) - x'(0)] + 3[sX(s) - x(0)] + 2X(s) = \frac{5}{s^2+1}$$

Simplify it:

$$X(s) = \underbrace{\frac{s+3}{s^2+3s+2}}_{X_1(s)} + \underbrace{\frac{5}{(s^2+1)(s^2+3s+2)}}_{X_2(s)}$$

Using Laplace Approach (cont.)

Partial fraction expansion:

$$X_1(s) = \frac{A}{s+1} + \frac{B}{s+2} = \frac{s+3}{(s+1)(s+2)}$$

and

$$X_2(s) = \frac{C}{s+1} + \frac{D}{s+2} + \frac{Es+F}{s^2+1} = \frac{5}{(s+1)(s+2)(s^2+1)}$$

Determine the values for **A**,**B**,**C**,**D**,**E** & **F**

Then, $X(s) = X_1(s) + X_2(s)$

Using Laplace Approach (cont.)

Finally, x(t) can be found by applying the inverse Laplace transform of X(s)

$$\boxed{x(t) = L^{-1}\left[X(s)\right]}$$

Laplace Transforms

- Def: $$\boxed{F(s) = L(f) = \int_0^\infty e^{-st} f(t)\,dt \quad \text{for} \quad f(t), t > 0}$$
- Inverse: $$f(t) = L^{-1}(F)$$
- Linearity: $$L\{af(t) + bg(t)\} = aL\{f(t)\} + bL\{g(t)\}$$
- Shifting Theorom:

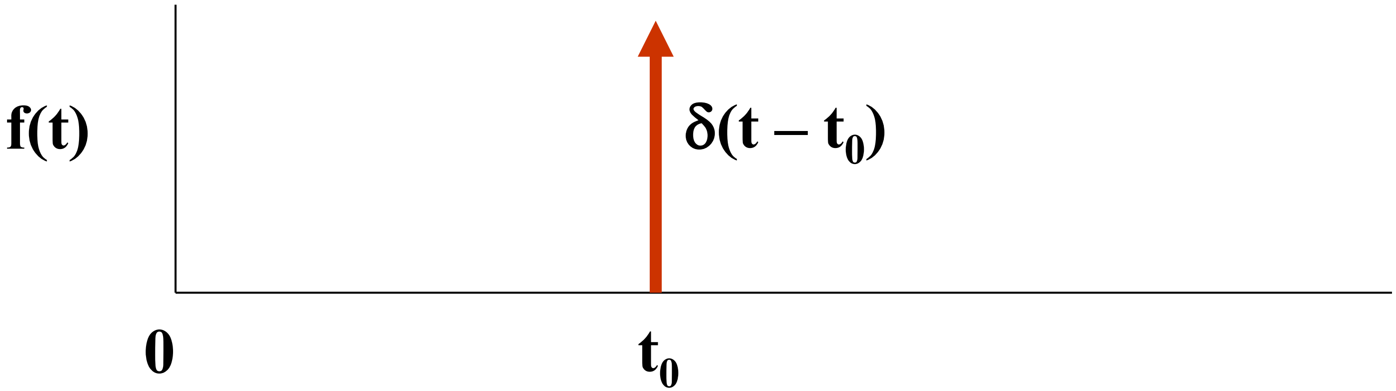
$$L\{e^{at} f(t)\} = F(s-a)$$

$$e^{at} f(t) = L^{-1}\{F(s-a)\}$$

The Laplace Transform

The Laplace transform of a unit impulse:

Pictorially, the unit impulse appears as follows:



Mathematically:

$$\delta(t - t_0) = 0 \quad t \neq 0$$

$$\int_{t_0-\varepsilon}^{t_0+\varepsilon} \delta(t - t_0)\,dt = 1 \qquad \varepsilon > 0$$

*note

The Laplace Transform

Transform Pairs:

f(t)	F(s)
$\delta(t)$	1
$u(t)$	$\frac{1}{s}$
e^{-st}	$\frac{1}{s+a}$
t	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^{n+1}}$

The Laplace Transform

Transform Pairs:

f(t)	F(s)
te^{-at}	$\frac{1}{(s+a)^2}$
$t^n e^{-at}$	$\frac{n!}{(s+a)^{n+1}}$
$\sin(wt)$	$\frac{w}{s^2+w^2}$
$\cos(wt)$	$\frac{s}{s^2+w^2}$

The Laplace Transform

Transform Pairs:

f(t)	F(s)
$e^{-at}\sin(wt)$	$\dfrac{w}{(s+a)^2+w^2}$
$e^{-at}\cos(wt)$	$\dfrac{s+a}{(s+a)^2+w^2}$
$\sin(wt+\theta)$	$\dfrac{s\sin\theta+w\cos\theta}{s^2+w^2}$
$\cos(wt+\theta)$	$\dfrac{s\cos\theta-w\sin\theta}{s^2+w^2}$

The Laplace Transform

Common Transform Properties:

f(t)	F(s)
$f(t-t_0)u(t-t_0), t_0 \geq 0$	$e^{-t_o s}F(s)$
$f(t)u(t-t_0), t \geq 0$	$e^{-t_o s}L[f(t+t_0)]$
$e^{-at}f(t)$	$F(s+a)$
$\dfrac{d^n f(t)}{dt^n}$	$s^n F(s) - s^{n-1}f(0) - s^{n-2}f'(0) - \ldots - s^0 f^{n-1}f(0)$
$tf(t)$	$-\dfrac{dF(s)}{ds}$
$\int_0^t f(\lambda)d\lambda$	$\dfrac{1}{s}F(s)$

The Laplace Transform

Using Matlab with Laplace transform:

Example **Use Matlab to find the transform of** te^{-4t}

The following is written in italic to indicate Matlab code

```
syms t,s
laplace(t*exp(-4*t),t,s)
ans =
      1/(s+4)^2
```

The Laplace Transform

Using Matlab with Laplace transform:

Example **Use Matlab to find the inverse transform of**

$$F(s) = \frac{s(s+6)}{(s+3)(s^2+6s+18)} \qquad prob.12.19$$

syms s t

ilaplace(s*(s+6)/((s+3)*(s^2+6*s+18)))

ans =

-exp(-3*t)+2*exp(-3*t)*cos(3*t)

The Laplace Transform

Theorem: **Initial Value Theorem:**

If the function f(t) and its first derivative are Laplace transformable and f(t) Has the Laplace transform F(s), and the $\lim_{s \to \infty} sF(s)$ exists, then

$$\boxed{\lim_{s \to \infty} sF(s) = \lim_{t \to 0} f(t) = f(0)}$$

Initial Value Theorem

The utility of this theorem lies in not having to take the inverse of F(s) in order to find out the initial condition in the time domain. This is particularly useful in circuits and systems.

The Laplace Transform

Example: Initial Value Theorem:

Given;

$$F(s) = \frac{(s+2)}{(s+1)^2 + 5^2}$$

Find f(0)

$$f(0) = \lim_{s \to \infty} sF(s) = \lim_{s \to \infty} s\frac{(s+2)}{(s+1)^2 + 5^2} = \lim_{s \to \infty}\left[\frac{s^2 + 2s}{s^2 + 2s + 1 + 25}\right]$$

$$= \lim_{s \to \infty} \frac{s^2/s^2 + 2s/s^2}{s^2/s^2 + 2s/s^2 + (26/s^2)} = 1$$

The Laplace Transform

Theorem: **Final Value Theorem:**

If the function f(t) and its first derivative are Laplace transformable and f(t) has the Laplace transform F(s), and the $\lim_{s\to\infty} sF(s)$ exists, then

$$\boxed{\lim_{s\to 0} sF(s) = \lim_{t\to\infty} f(t) = f(\infty)}$$

Final Value Theorem

Again, the utility of this theorem lies in not having to take the inverse of F(s) in order to find out the final value of f(t) in the time domain. This is particularly useful in circuits and systems.

The Laplace Transform

Example: Final Value Theorem:

Given:

$$F(s) = \frac{(s+2)^2 - 3^2}{\left[(s+2)^2 + 3^2\right]} \qquad note\ F^{-1}(s) = te^{-2t}\cos 3t$$

Find $f(\infty)$.

$$f(\infty) = \lim_{s \to 0} sF(s) = \lim_{s \to 0} s\frac{(s+2)^2 - 3^2}{\left[(s+2)^2 + 3^2\right]} = 0$$

Solution of Partial Fraction Expansion

- The solution of each distinct (non-multiple) root, real or complex uses a two step process.
  - The first step in evaluating the constant is to multiply both sides of the equation by the factor in the denominator of the constant you wish to find.
  - The second step is to replace s on both sides of the equation by the root of the factor by which you multiplied in step 1

$$X(s)=\frac{8(s+3)(s+8)}{s(s+2)(s+4)}=\frac{K_1}{s}+\frac{K_2}{s+2}+\frac{K_3}{s+4}$$

$$K_1=\left.\frac{8(s+3)(s+8)}{(s+2)(s+4)}\right|_{s=0}=\frac{8(0+3)(0+8)}{(0+2)(0+4)}=24$$

$$K_2=\left.\frac{8(s+3)(s+8)}{s(s+4)}\right|_{s=-2}=\frac{8(-2+3)(-2+8)}{-2(-2+4)}=-12$$

$$K_3 = \left.\frac{8(s+3)(s+8)}{s(s+2)}\right|_{s=-4} = \frac{8(-4+3)(-4+8)}{-4(-4+4)} = -4$$

The partial fraction expansion is:

$$X(s) = \frac{24}{s} - \frac{12}{s+2} - \frac{4}{s+4}$$

- The inverse Laplace transform is found from the functional table pairs to be:

$$x(t) = 24 - 12e^{-2t} - 4e^{-4t}$$

Repeated Roots

- Any unrepeated roots are found as before.
- The constants of the repeated roots $(s\text{-}a)^m$ are found by first breaking the quotient into a partial fraction expansion with descending powers from *m* to *0:*

$$\frac{B_m}{(s-a)^m}+\cdots+\frac{B_2}{(s-a)^2}+\frac{B_1}{(s-a)}$$

- The constants are found using one of the following:

$$B_i = \frac{1}{(m-i)!} \frac{d^{m-i}}{ds^{m-i}} \left[\frac{P(s)}{Q(s)/(s-a_1)^m} \right]_{s=a_1}$$

$$B_m = \frac{P(a)}{\left[Q(s)/(s-a)^m \right]_{s=a}}$$

$$Y(s) = \frac{8(s+1)}{(s+2)^2} = \frac{K_1}{s+2} + \frac{K_2}{(s+2)^2}$$

$$K_2 = \left. \frac{8(s+1)(s+2)^2}{(s+2)^2} \right|_{s=-2} = 8(s+1)\Big|_{s=-2} = -8$$

$$B_i = \frac{1}{(2-1)!}\frac{d}{ds}\left[\frac{8(s+1)}{(s+2)^2/(s+2)^2}\right]_{s=-2} = 8$$

The partial fraction expansion yields:

$$Y(s) = \frac{8}{s+2} - \frac{8}{(s+2)^2}$$

The inverse Laplace transform derived from the functional table pairs yields:

$$y(t) = 8e^{-2t} - 8te^{-2t}$$

A Second Method for Repeated Roots

$$Y(s) = \frac{8(s+1)}{(s+2)^2} = \frac{K_1}{s+2} + \frac{K_2}{(s+2)^2}$$

$$8(s+1) = K_1(s+2) + K_2$$

$$8s + 8 = K_1 s + 2K_1 + K_2$$

Equating like terms:

$$8 = K_1 \quad and \quad 8 = 2K_1 + K_2$$

$$8 = K_1 \quad and \quad 8 = 2K_1 + K_2$$

$$8 = 2 \times 8 + K_2$$

$$8 - 16 = -8 = K_2$$

Thus

$$Y(s) = \frac{8}{s+2} - \frac{8}{(s+2)^2}$$

$$y(t) = 8e^{-2t} - 8te^{-2t}$$

Another Method for Repeated Roots

$$Y(s) = \frac{8(s+1)}{(s+2)^2} = \frac{K_1}{s+2} + \frac{K_2}{(s+2)^2}$$

As before, we can solve for K_2 in the usual manner.

$$K_2 = \left. \frac{8(s+1)(s+2)^2}{(s+2)^2} \right|_{s=-2} = 8(s+1)\Big|_{s=-2} = -8$$

$$(s+2)^2 \frac{8(s+1)}{(s+2)^2} = (s+2)^2 \frac{K_1}{s+2} - (s+2)^2 \frac{8}{\left(s+2\right)^2}$$

$$\frac{d\left[8(s+1)\right]}{ds} = \frac{d\left[\left(s+2\right)K_1 - 8\right]}{ds}$$

$$8 = K_1$$

$$Y(s) = \frac{8(s+1)}{(s+2)^2} = \frac{8}{s+2} - \frac{8}{\left(s+2\right)^2}$$

$$y(t) = 8e^{-2t} - 8te^{-2t}$$

Unrepeated Complex Roots

- Unrepeated complex roots are solved similar to the process for unrepeated real roots. That is you multiply by one of the denominator terms in the partial fraction and solve for the appropriate constant.
- Once you have found one of the constants, the other constant is simply the complex conjugate.

Complex Unrepeated Roots

$$\frac{5.2}{s^2+2s+5} = \frac{5.2}{s^2+2s+1+4}$$

$$\frac{5.2}{(s+1)^2+2^2} \quad w=2 \;\; a=1$$

$e^{-at}\sin(wt)$	$\dfrac{w}{(s+a)^2+w^2}$
$e^{-at}\cos(wt)$	$\dfrac{s+a}{(s+a)^2+w^2}$
$\sin(wt+\theta)$	$\dfrac{s\sin\theta + w\cos\theta}{s^2+w^2}$
$\cos(wt+\theta)$	$\dfrac{s\cos\theta - w\sin\theta}{s^2+w^2}$

$$\frac{5.2}{2}e^{-t}\sin(2t)$$

$$\frac{s+2}{(s+1)(s^2+2s+5)(s+3)2} = \frac{K_1}{s+1} + \frac{K_2 s + K_3}{s^2+2s+}$$

$$K_1 = \left.\frac{s+2}{(s^2+2s+5)}\right|_{s=-1} = 1/4$$

$$s+2 = K1(s^2+2s+5) + (k2s+k3)(s+1)$$

Chapter 2: Mathematical Models of Systems

Objectives

We use quantitative mathematical models of physical systems to design and analyze control systems. The dynamic behavior is generally described by ordinary differential equations. We will consider a wide range of systems, including mechanical, hydraulic, and electrical. Since most physical systems are nonlinear, we will discuss linearization approximations, which allow us to use Laplace transform methods.

We will then proceed to obtain the input–output relationship for components and subsystems in the form of transfer functions. The transfer function blocks can be organized into block diagrams or signal-flow graphs to graphically depict the interconnections. Block diagrams (and signal-flow graphs) are very convenient and natural tools for designing and analyzing complicated control systems

Basic Elements of Electrical Systems

Symbol ►

- The time domain expression relating voltage and current for the resistor is given by Ohm's law i-e

$$v_R(t) = i_R(t)R$$

- The Laplace transform of the above equation is

$$V_R(s) = I_R(s)R$$

Basic Elements of Electrical Systems

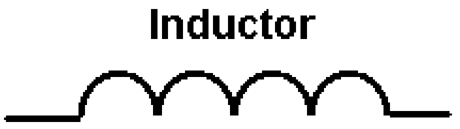
- The time domain expression relating voltage and current for the Capacitor is given as:

$$v_c(t) = \frac{1}{C}\int i_c(t)dt$$

- The Laplace transform of the above equation (assuming there is no charge stored in the capacitor) is

$$V_c(s) = \frac{1}{Cs} I_c(s)$$

Basic Elements of Electrical Systems



- The time domain expression relating voltage and current for the inductor is given as:

$$v_L(t) = L\frac{di_L(t)}{dt}$$

- The Laplace transform of the above equation (assuming there is no energy stored in inductor) is

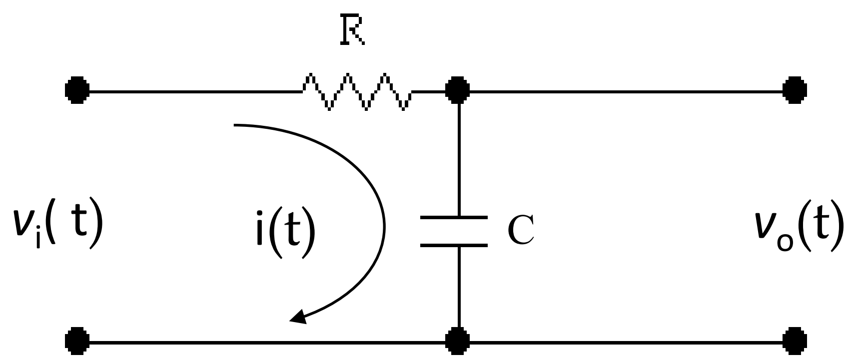
$$V_L(s) = LsI_L(s)$$

V-I and I-V relations

Component	Symbol	V-I Relation	I-V Relation
Resistor		$v_R(t) = i_R(t)R$	$i_R(t) = \dfrac{v_R(t)}{R}$
Capacitor		$v_c(t) = \dfrac{1}{C}\int i_c(t)dt$	$i_c(t) = C\dfrac{dv_c(t)}{dt}$
Inductor		$v_L(t) = L\dfrac{di_L(t)}{dt}$	$i_L(t) = \dfrac{1}{L}\int v_L(t)dt$

Example#1

- The two-port network shown in the following figure has $v_i(t)$ as the input voltage and $v_o(t)$ as the output voltage. Find the transfer function $V_o(s)/V_i(s)$ of the network.



$$v_i(t) = i(t)R + \frac{1}{C}\int i(t)dt$$

$$v_o(t) = \frac{1}{C}\int i(t)dt$$

Example#1

$$v_i(t) = i(t)R + \frac{1}{C}\int i(t)dt \qquad\qquad v_o(t) = \frac{1}{C}\int i(t)dt$$

- Taking Laplace transform of both equations, considering initial conditions to zero.

$$V_i(s) = I(s)R + \frac{1}{Cs}I(s) \qquad\qquad V_o(s) = \frac{1}{Cs}I(s)$$

- Re-arrange both equations as:

$$V_i(s) = I(s)(R + \frac{1}{Cs}) \qquad\qquad CsV_o(s) = I(s)$$

Example#1

$$V_i(s) = I(s)(R + \frac{1}{Cs})$$

$$CsV_o(s) = I(s)$$

- Substitute *I(s)* in equation on left

$$V_i(s) = CsV_o(s)(R + \frac{1}{Cs})$$

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{Cs(R + \frac{1}{Cs})}$$

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{1 + RCs}$$

Example#1

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{1 + RCs}$$

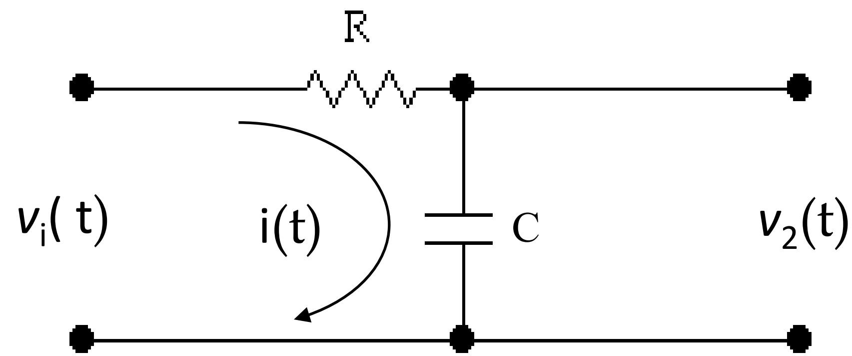
- The system has one pole at

$$1 + RCs = 0 \qquad \Rightarrow s = -\frac{1}{RC}$$

Example#2

- Design an Electrical system that would place a pole at -3 if added to another system.

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{1 + RCs}$$



- System has one pole at

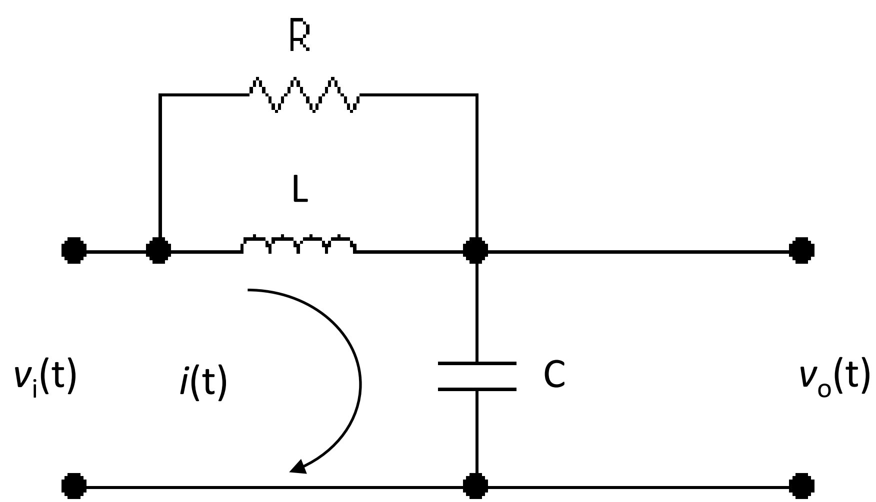
$$s = -\frac{1}{RC}$$

- Therefore,

$$-\frac{1}{RC} = -3 \qquad if \qquad R = 1\ M\Omega \quad and \quad C = 333\ pF$$

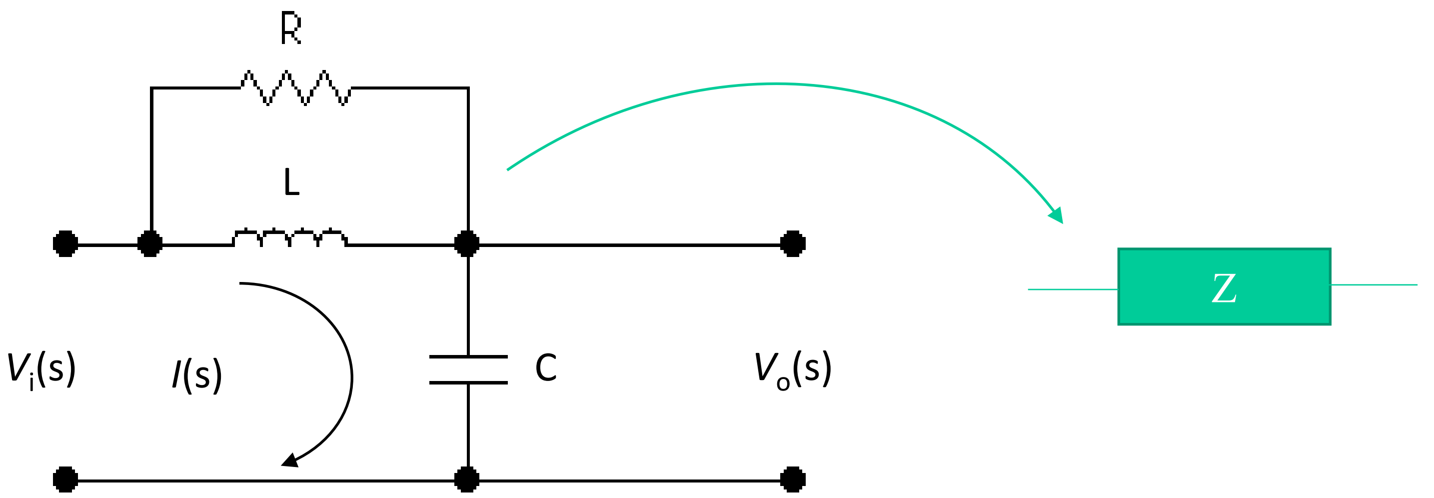
Example#3

- Find the transfer function G(S) of the following two port network.

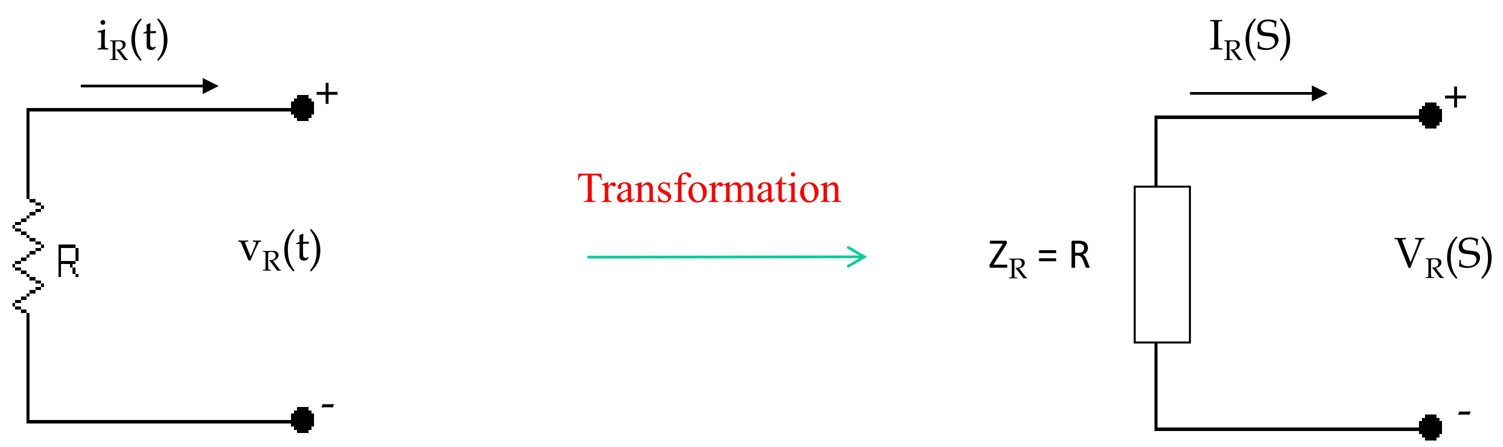


Example#3

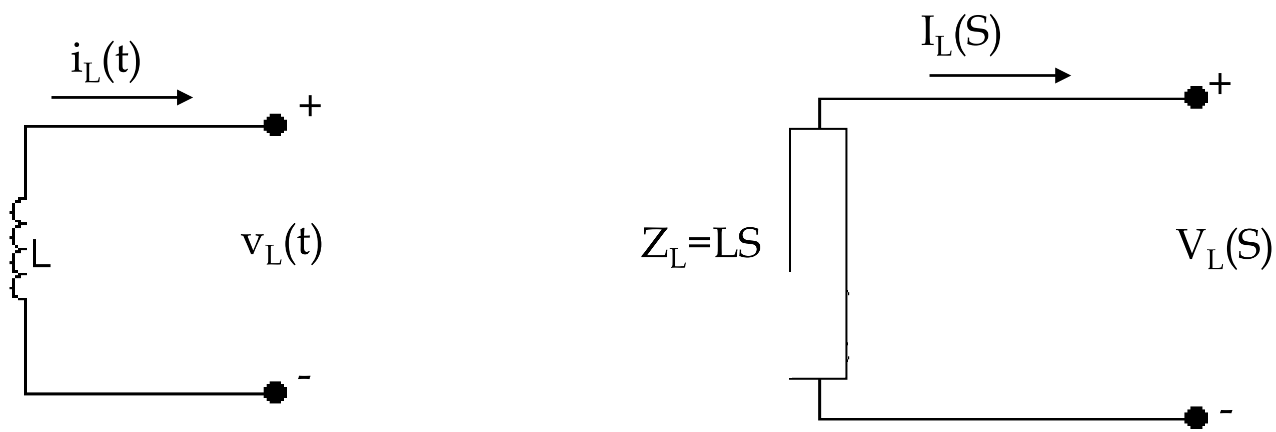
- Simplify network by replacing multiple components with their equivalent transform impedance.



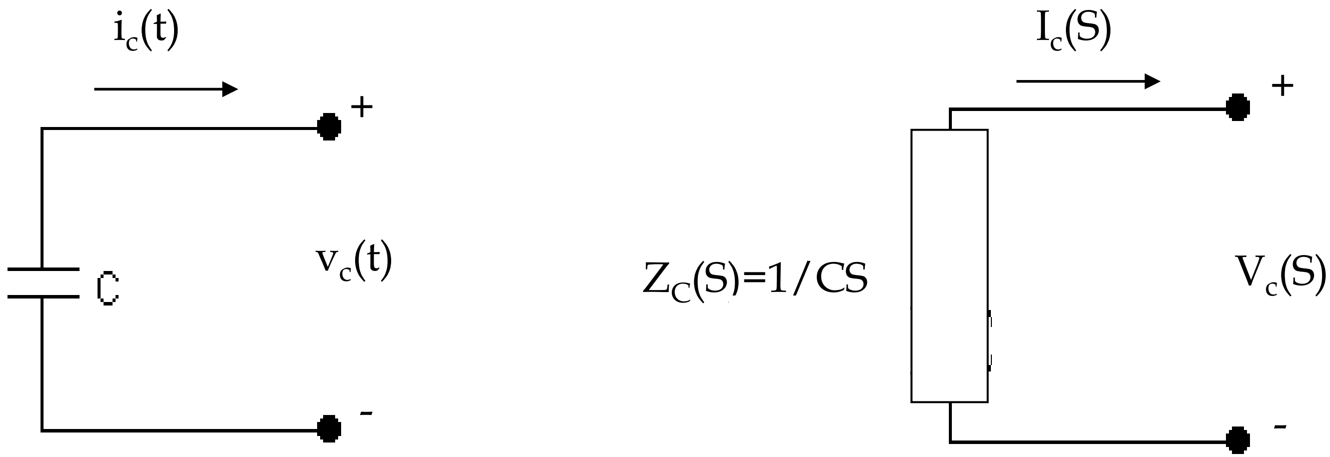
Transform Impedance (Resistor)



Transform Impedance (Inductor)



Transform Impedance (Capacitor)

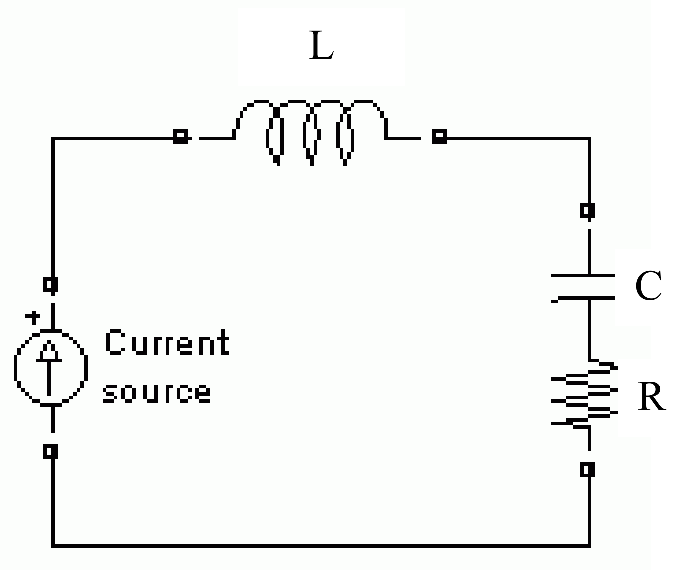


Equivalent Transform Impedance (Series)

- Consider following arrangement, find out equivalent transform impedance.

$$Z_T = Z_R + Z_L + Z_C$$

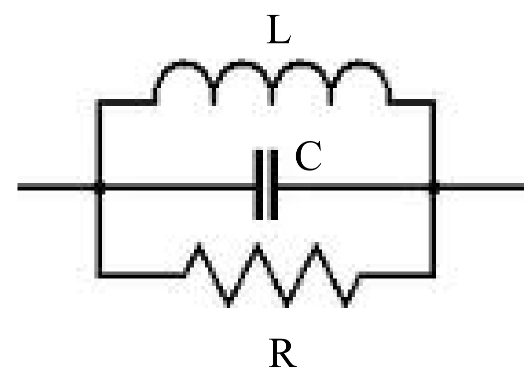
$$Z_T = R + Ls + \frac{1}{Cs}$$



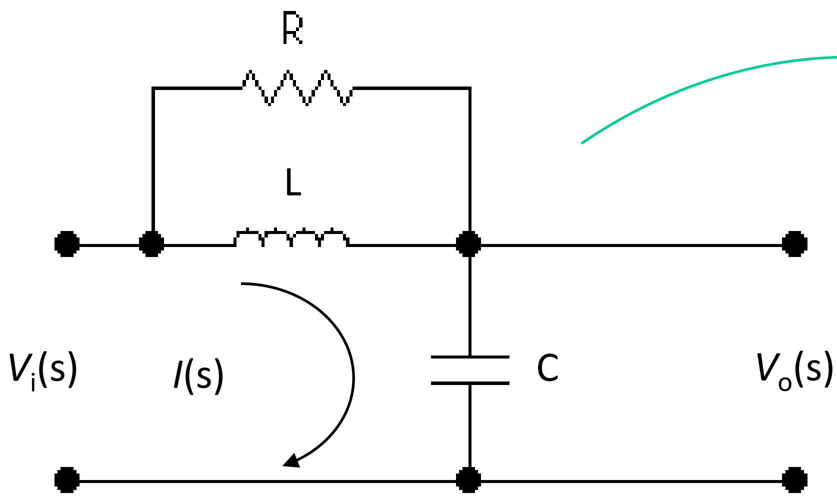
Equivalent Transform Impedance (Parallel)

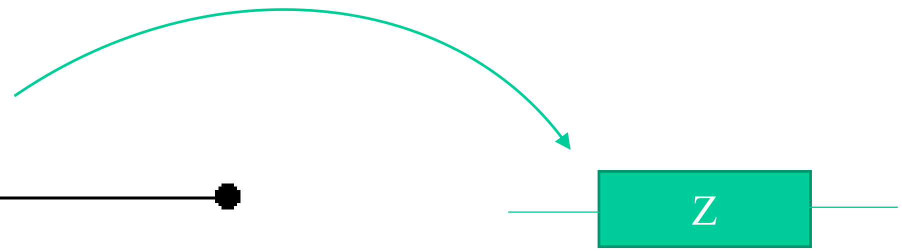
$$\frac{1}{Z_T} = \frac{1}{Z_R} + \frac{1}{Z_L} + \frac{1}{Z_C}$$

$$\frac{1}{Z_T} = \frac{1}{R} + \frac{1}{Ls} + \frac{1}{\frac{1}{Cs}}$$



Back to Example#3





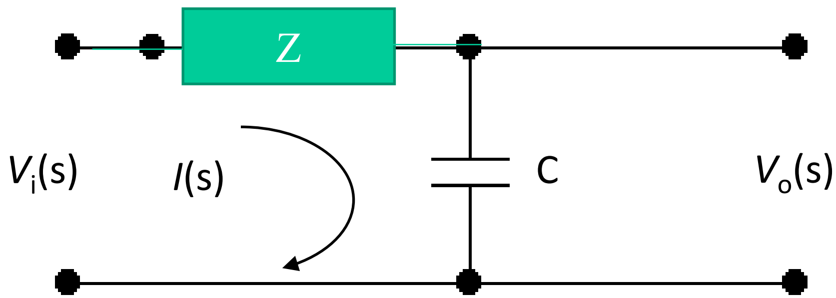
$$\frac{1}{Z} = \frac{1}{Z_R} + \frac{1}{Z_L}$$

$$\frac{1}{Z} = \frac{1}{R} + \frac{1}{Ls}$$

$$Z = \frac{RLs}{1 + RLs}$$

Example#3

$$Z = \frac{RLs}{1 + RLs}$$

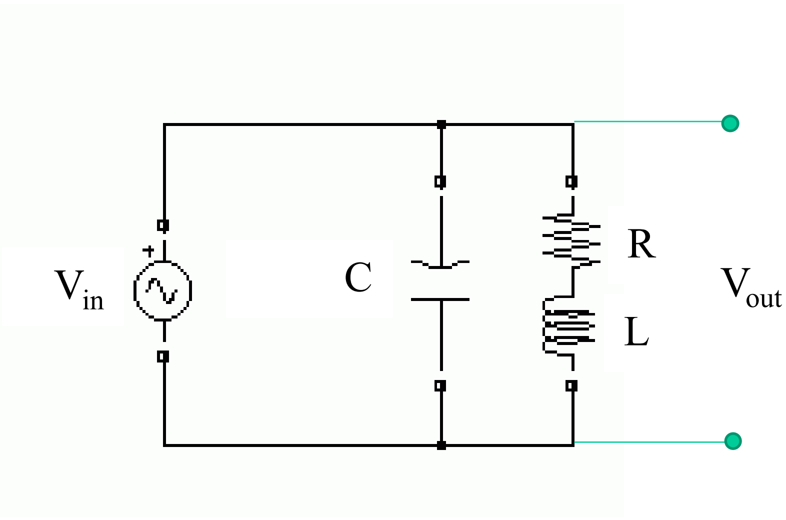


$$V_i(s) = I(s)Z + \frac{1}{Cs} I(s)$$

$$V_o(s) = \frac{1}{Cs} I(s)$$

Example#4

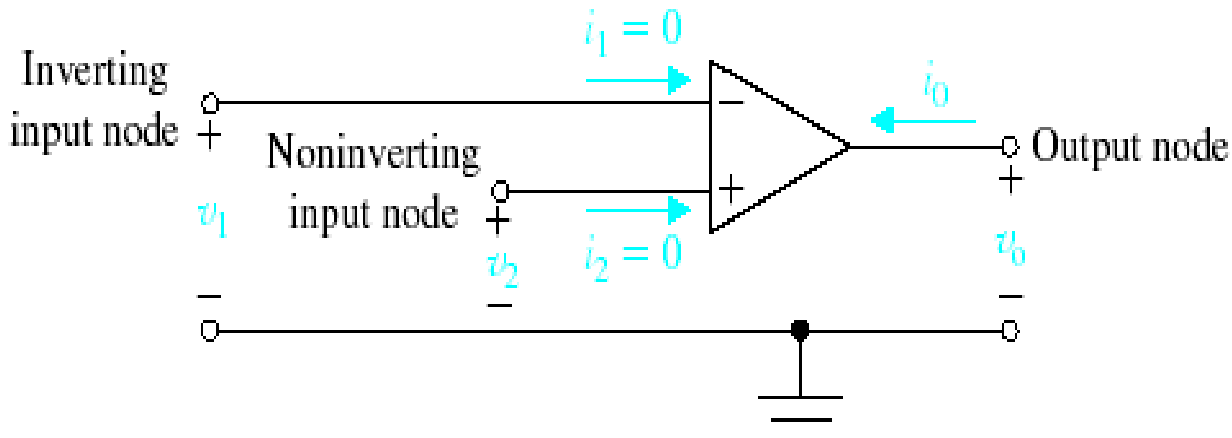
- Find transfer function $V_{out}(s)/V_{in}(s)$ of the following electrical network



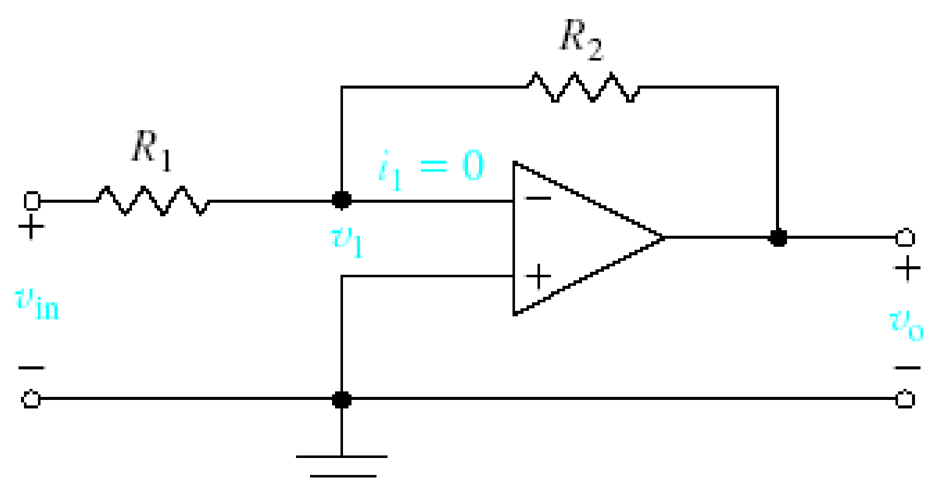
Electronic Systems

Part-II

The Transfer Function of Linear Systems

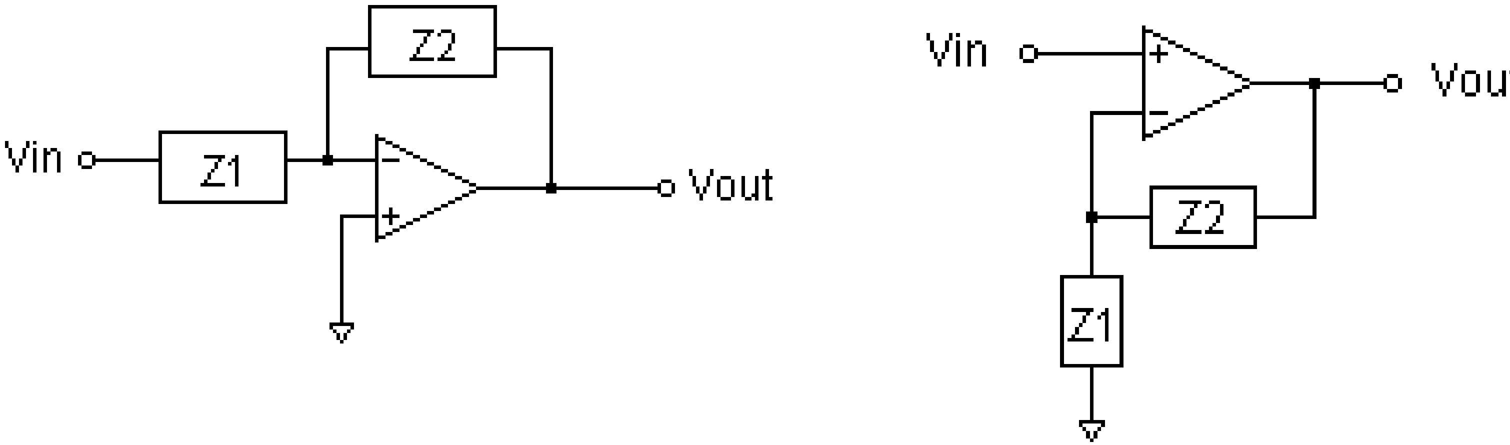


The ideal op-amp



An inverting amplifier operating with ideal conditions.

Operational Amplifiers

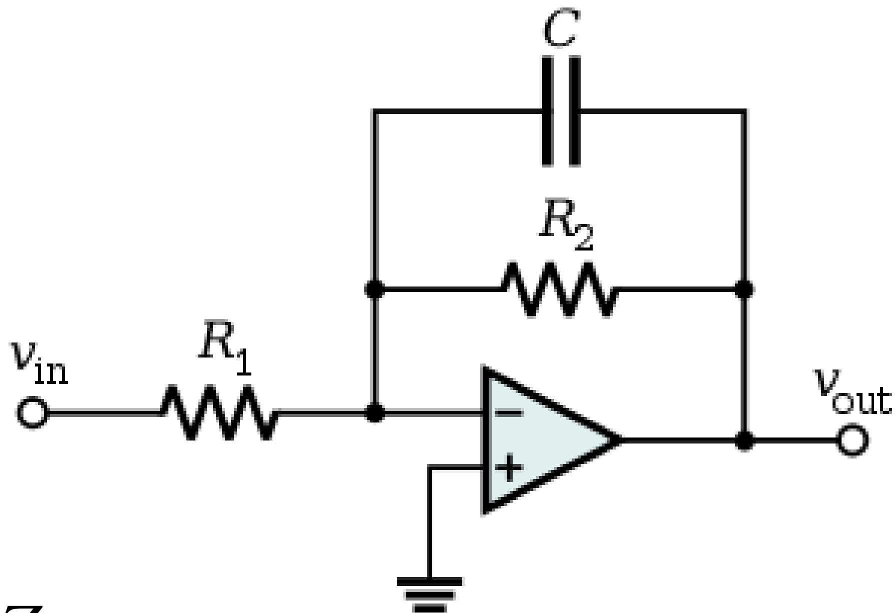


$$\frac{V_{out}}{V_{in}} = -\frac{Z_2}{Z_1}$$

$$\frac{V_{out}}{V_{in}} = 1 + \frac{Z_2}{Z_1}$$

Example#6

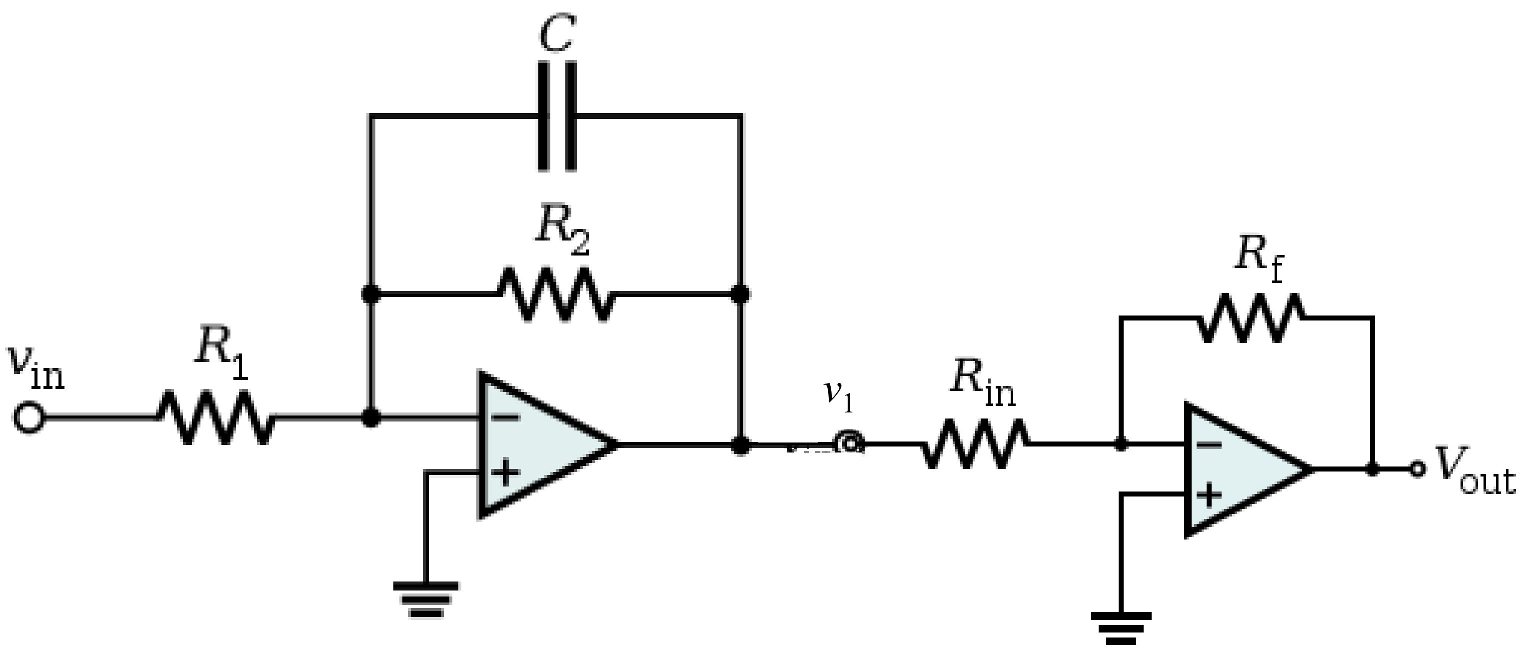
- Find out the transfer function of the following circuit.



$$\frac{V_{out}}{V_{in}} = -\frac{Z_2}{Z_1}$$

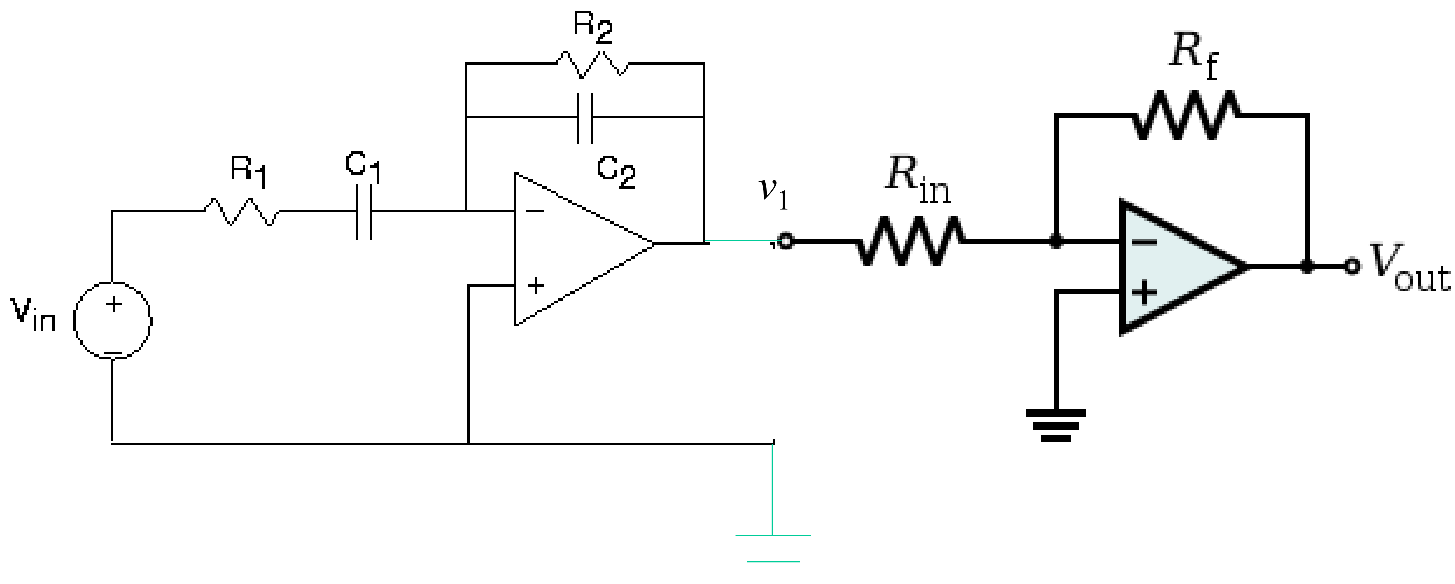
Example#7

- Find out the transfer function of the following circuit.



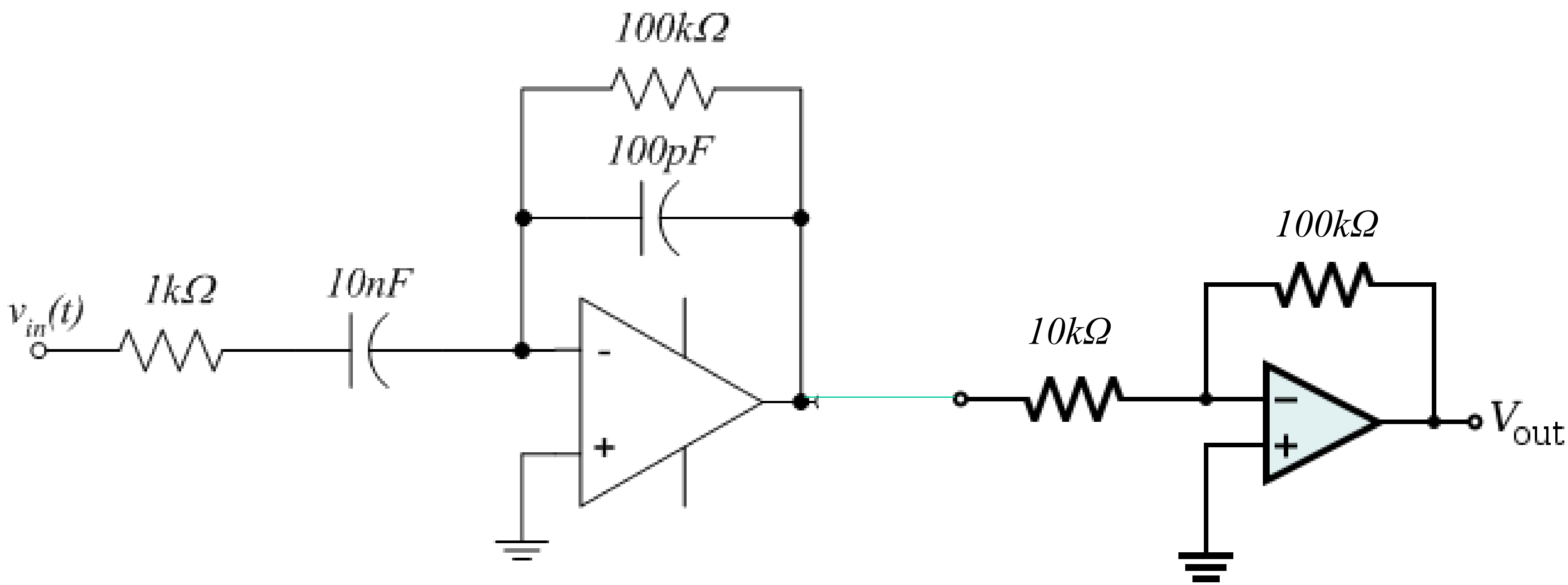
Example#8

- Find out the transfer function of the following circuit.

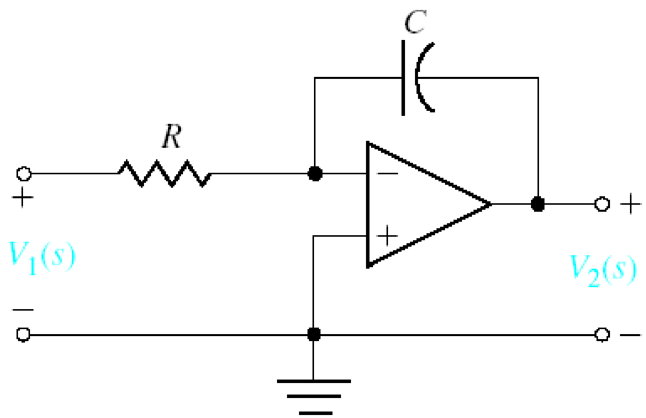


Example#9

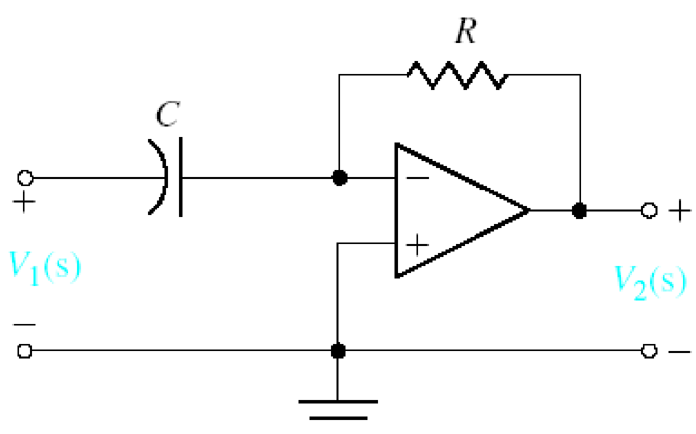
- Find out the transfer function of the following circuit and draw the pole zero map.



Examples write the transfer function for the following systems



$$\frac{V_2(s)}{V_1(s)} = \frac{-1}{RCs}$$



$$\frac{V_2(s)}{V_1(s)} = -RCs$$

Mechanical systems

The modelling of mechanical systems are mainly based on *Newton's second law*

$$F = ma \tag{3.4}$$

F is the *force* acting on the *mass* m and a is the *acceleration* of the mass.

Example 3.3. An undamped pendulum.

Figure 3.4 shows an undamped swinging pendulum. The pendulum can only move in two directions in the plane of the figure. Its point of sus-pension is at a distance u and its center of mass (the round weight at the lower end of the pendulum) is at a distance y from the left-side vertical line.

How does the position y depend on u ?

Notation:

- ℓ = length of pendulum, m = weight of mass
- h = vertical position of the center of mass
- θ = angle of swing away from a vertical position
- F = force acting on the suspension point in the "negative direction" (upwards)

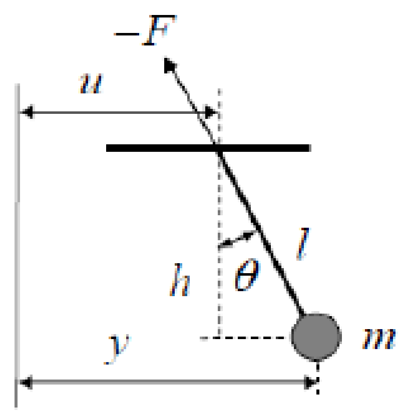


Fig. 3.4. Swinging pendulum.

When the pendulum is affected by the suspension force F and the gravitational force mg, *Newton's second law* yields

- horizontal force components: $m\ddot{y} = -F\sin\theta$ (1)
- vertical force components: $m\ddot{h} = -F\cos\theta + mg$ (2)

Here $\ddot{y}$ and $\ddot{h}$ are second-order time derivatives of y and h, respectively, i.e. the *acceleration* in the respective directions.

Assume that the swing of the pendulum is moderate so that the angle θ is always small. The pendulum then moves very little in the vertical direction and we can assume that $\ddot{h} \approx 0$. Elimination of F then gives

$$\ddot{y} + g\tan\theta = 0 \tag{3}$$

The angle θ is given by the trigonometric identity

$$\tan\theta = \frac{y-u}{h} \approx \frac{y-u}{l} \tag{4}$$

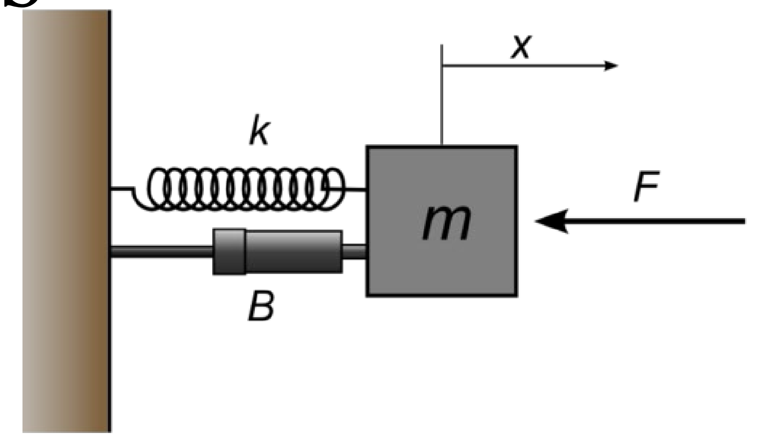
Combination of (3) and (4) yields the model

$$\ddot{y} + \left(\frac{g}{\ell}\right)y = \left(\frac{g}{\ell}\right)u \tag{5}$$

Notice that the approximations $\ddot{h} \approx 0$ and "θ small" ***limit the validity of the model***.

Basic Types of Mechanical Systems

- Translational
  - Linear Motion



- Rotational
  - Rotational Motion

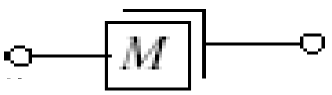
Basic Elements of Translational Mechanical Systems

Translational Spring

i)

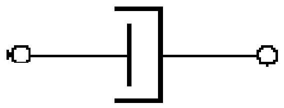
Translational Mass

ii)



Translational Damper

iii)

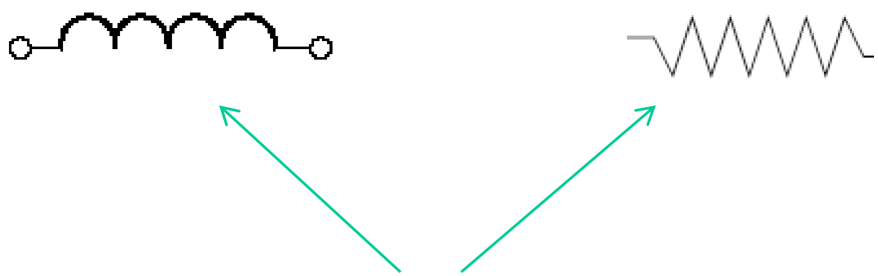


Translational Spring

- A translational spring is a mechanical element that can be deformed by an external force such that the deformation is directly proportional to the force applied to it.

Translational Spring

i)

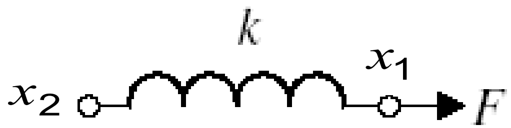


Circuit Symbols

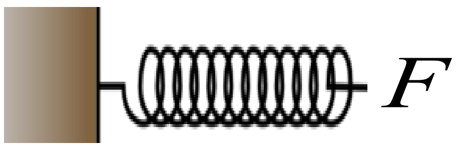
Translational Spring

Translational Spring

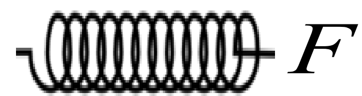
- If F is the applied force



- Then x_1 is the deformation if $x_2 = 0$



- Or $(x_1 - x_2)$ is the deformation.



- The equation of motion is given as

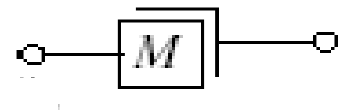
$$F = k(x_1 - x_2)$$

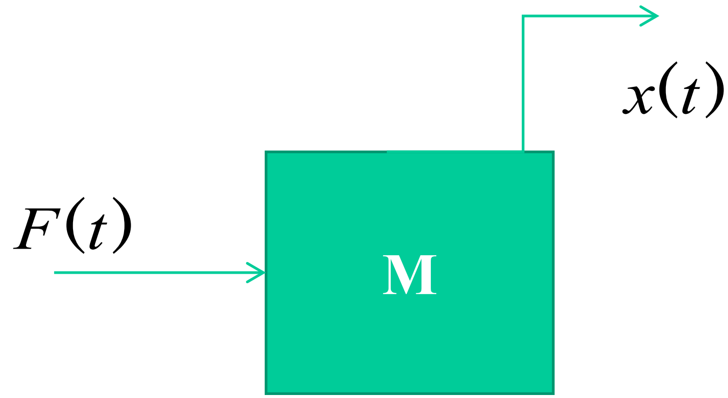
- Where k is stiffness of spring expressed in N/m

Translational Mass

- Translational Mass is an inertia element.
- A mechanical system without mass does not exist.
- If a force *F* is applied to a mass and it is displaced to *x* meters then the relation b/w force and displacements is given by Newton's law.

ii)





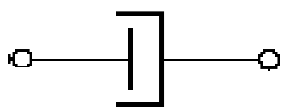
$$F = M\ddot{x}$$

Translational Damper

- Damper opposes the rate of change of motion.
- All the materials exhibit the property of damping to some extent.
- If damping in the system is not enough then extra elements (e.g. Dashpot) are added to increase damping.

Translational Damper

iii)



Common Uses of Dashpots

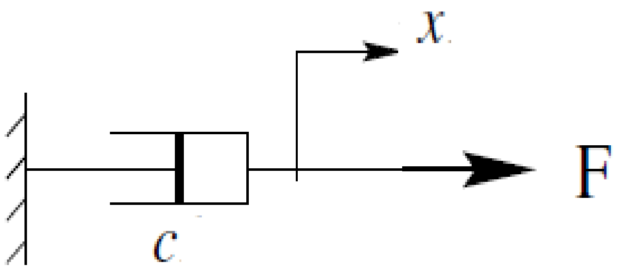
Door Stoppers

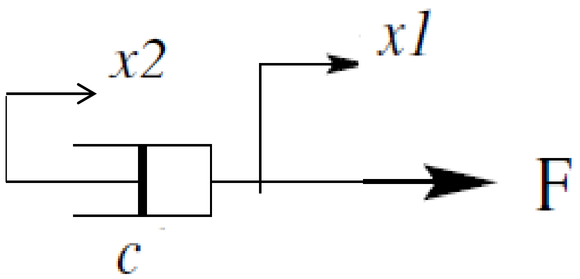
Vehicle Suspension

Bridge Suspension

Flyover Suspension

Translational Damper





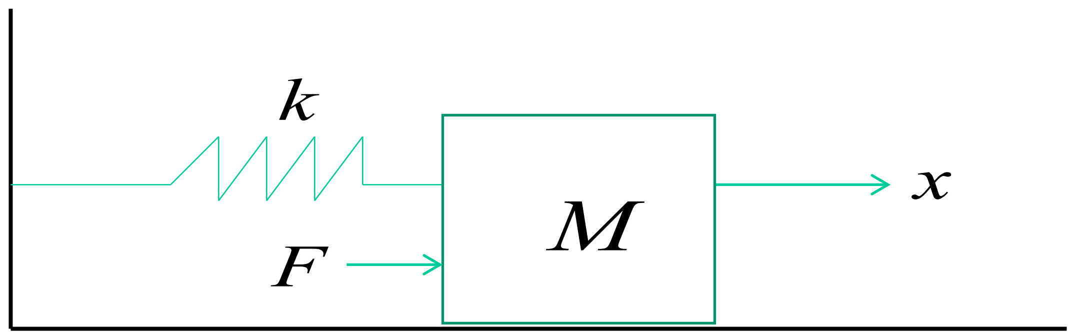
$$F = C\dot{x}$$

$$F = C(\dot{x}_1 - \dot{x}_2)$$

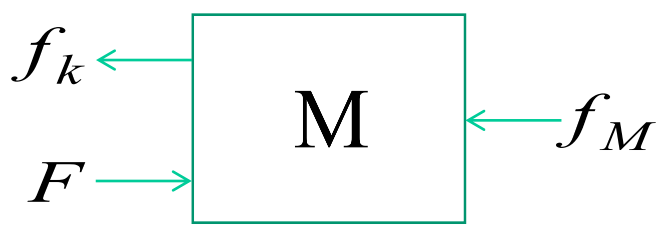
- Where C is damping coefficient (N/ms^{-1}).

Example-1

- Consider the following system (friction is negligible)

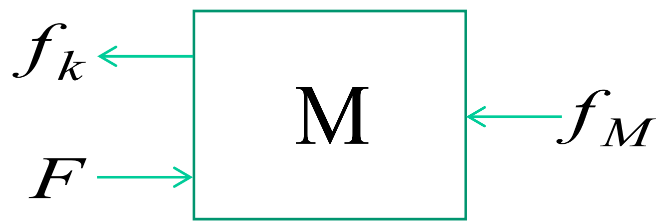


- Free Body Diagram



- Where f_k and f_M are force applied by the spring and inertial force respectively.

Example-1



$$F = f_k + f_M$$

- Then the differential equation of the system is:

$$F = kx + M\ddot{x}$$

- Taking the Laplace Transform of both sides and ignoring initial conditions we get

$$F(s) = Ms^2X(s) + kX(s)$$

Example-1

$$F(s) = Ms^2 X(s) + kX(s)$$

- The transfer function of the system is

$$\boxed{\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + k}}$$

- if

$M = 1000 kg$

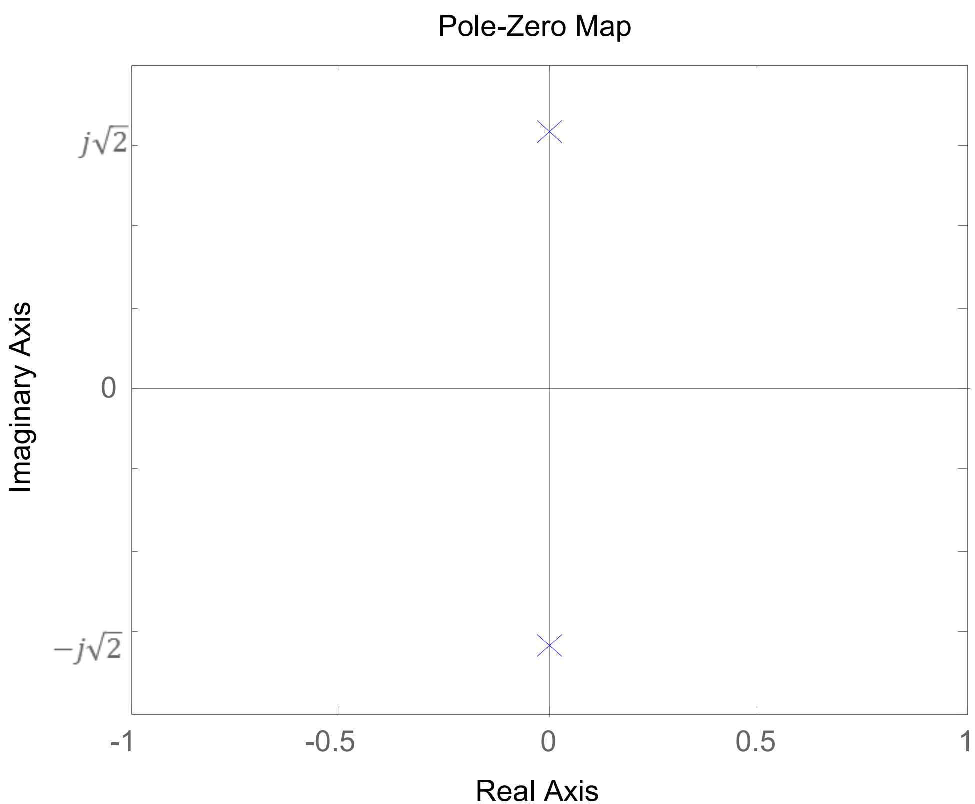
$k = 2000 Nm^{-1}$

$$\frac{X(s)}{F(s)} = \frac{0.001}{s^2 + 2}$$

Example-2

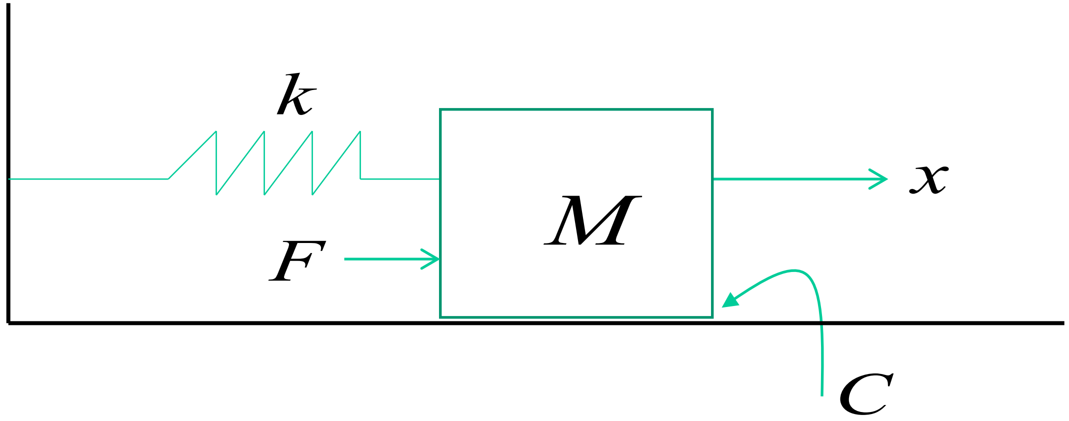
$$\frac{X(s)}{F(s)} = \frac{0.001}{s^2 + 2}$$

- The pole-zero map of the system is

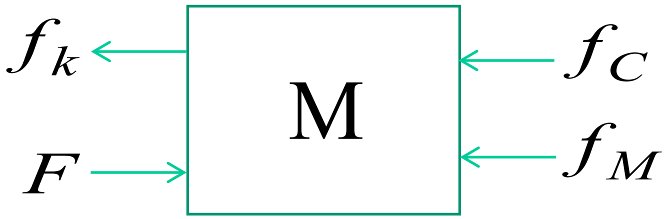


Example-2

- Consider the following system



- Free Body Diagram



$$F = f_k + f_M + f_C$$

Example-3

Differential equation of the system is:

$$F = M\ddot{x} + C\dot{x} + kx$$

Taking the Laplace Transform of both sides and ignoring Initial conditions we get

$$F(s) = Ms^2 X(s) + CsX(s) + kX(s)$$

$$\boxed{\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + Cs + k}}$$

Example-3

$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + Cs + k}$$

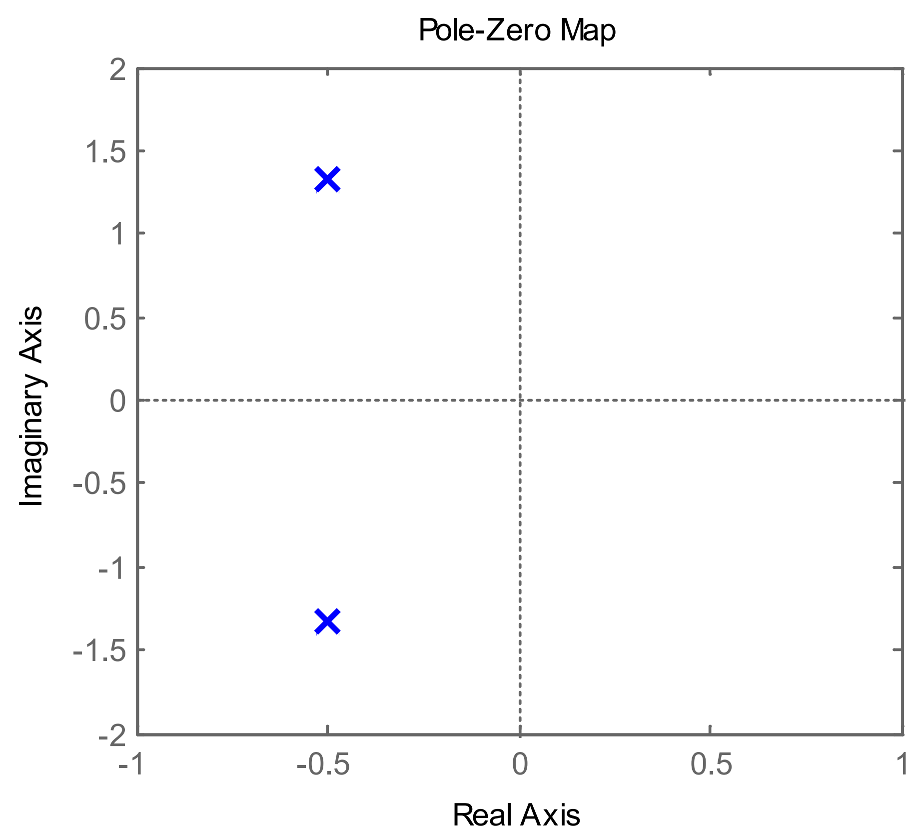
- if

$$M = 1000kg$$

$$k = 2000Nm^{-1}$$

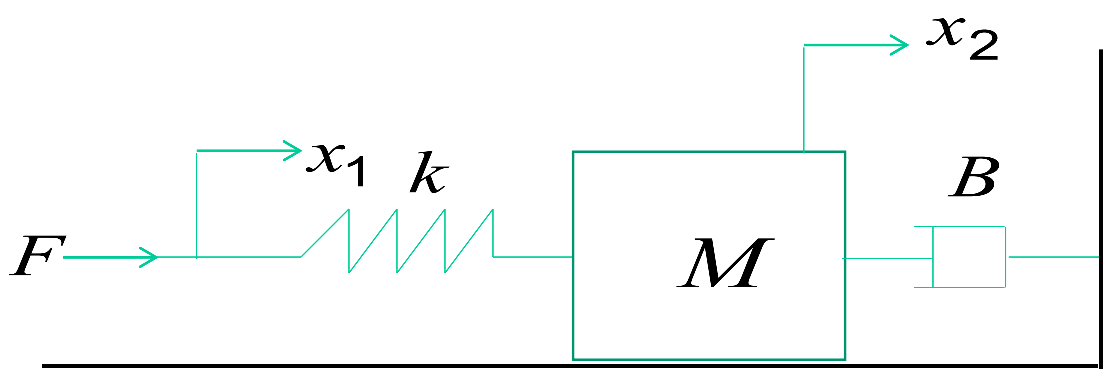
$$C = 1000N/ms^{-1}$$

$$\frac{X(s)}{F(s)} = \frac{0.001}{s^2 + s + 1000}$$

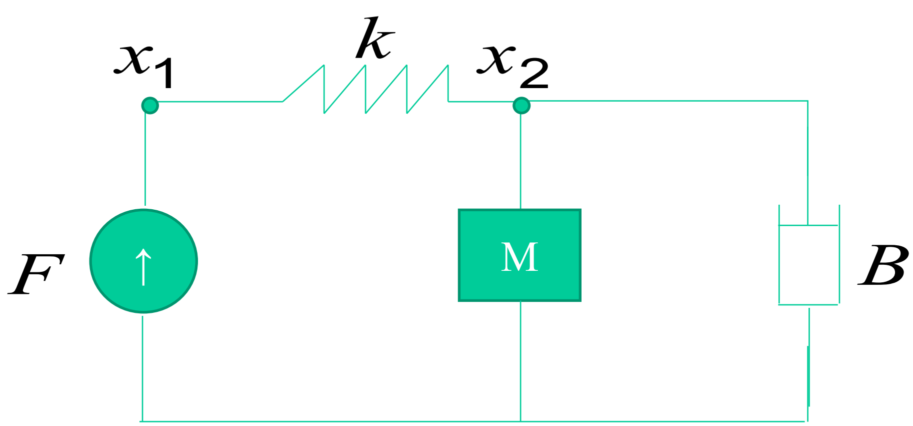


Example-4

- Consider the following system

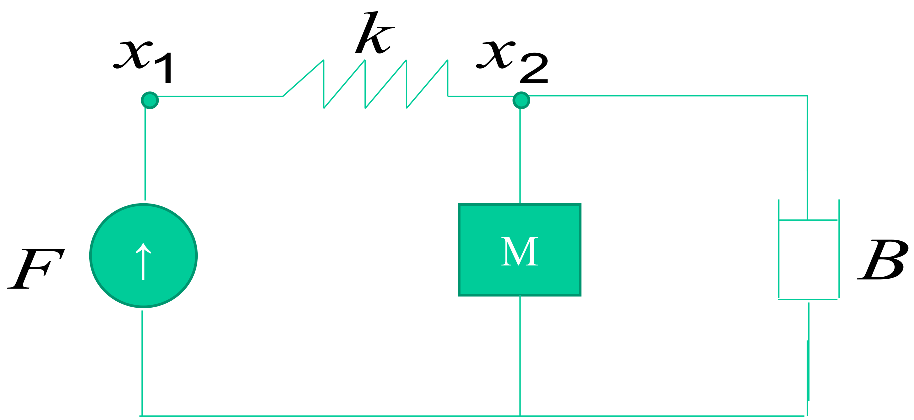


- Mechanical Network



Example-4

- Mechanical Network



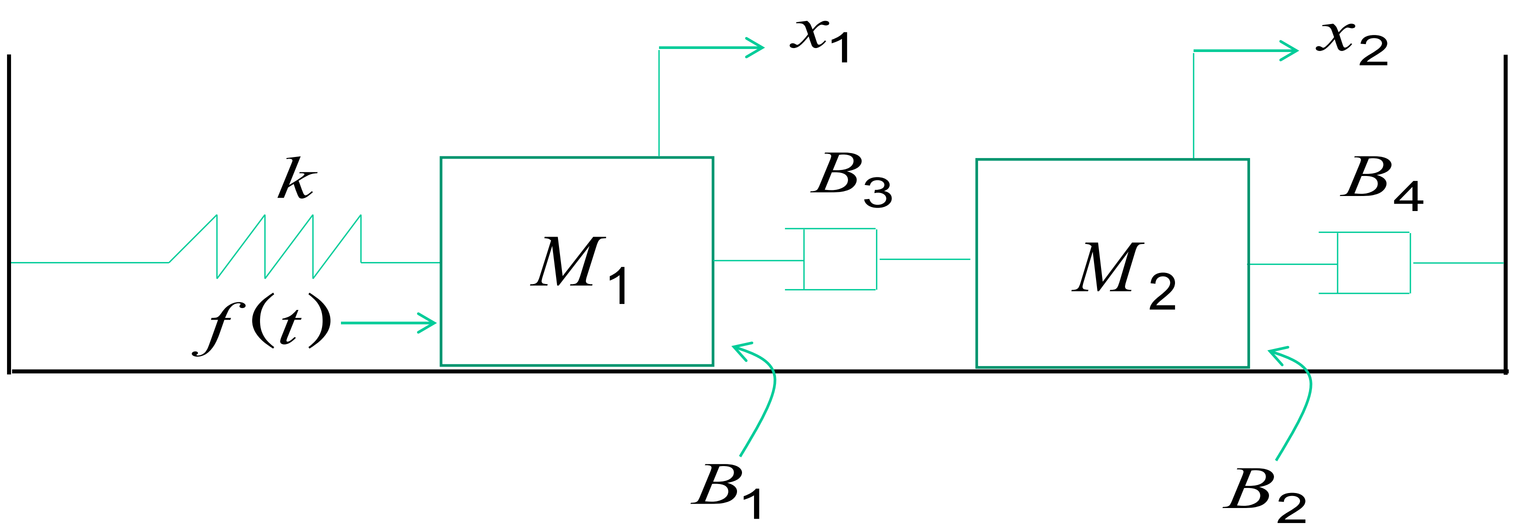
At node x_1

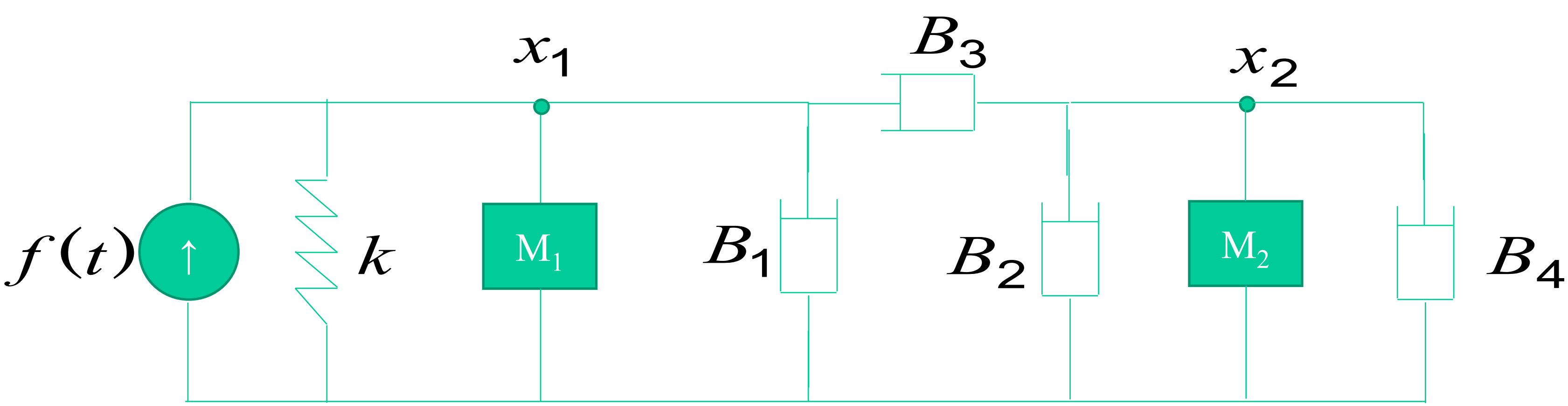
$$F = k(x_1 - x_2)$$

At node x_2

$$0 = k(x_2 - x_1) + M\ddot{x}_2 + B\dot{x}_2$$

Example-6





Example-7

- Find the transfer function of the mechanical translational system given in Figure-1.

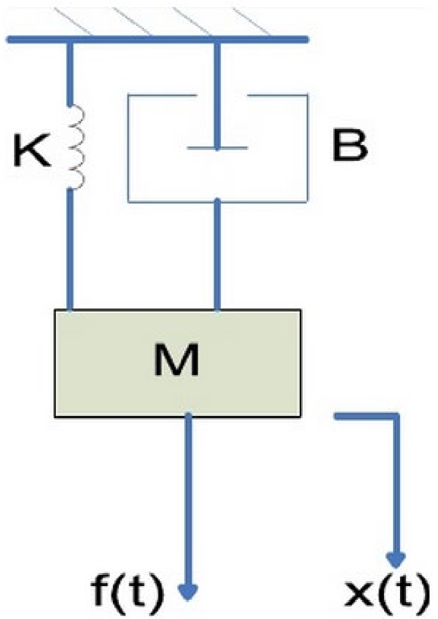
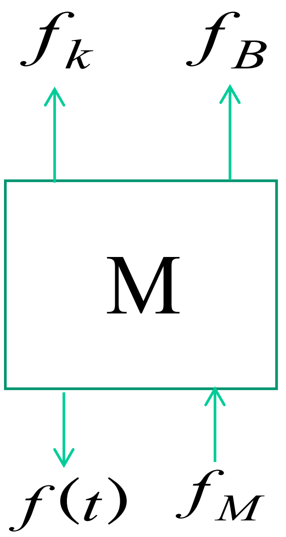


Figure-1

Free Body Diagram

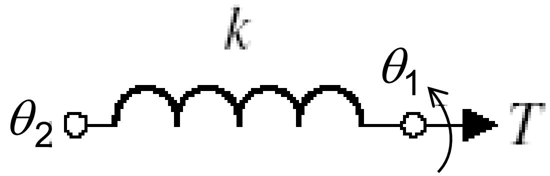


$$f(t) = f_k + f_M + f_B$$

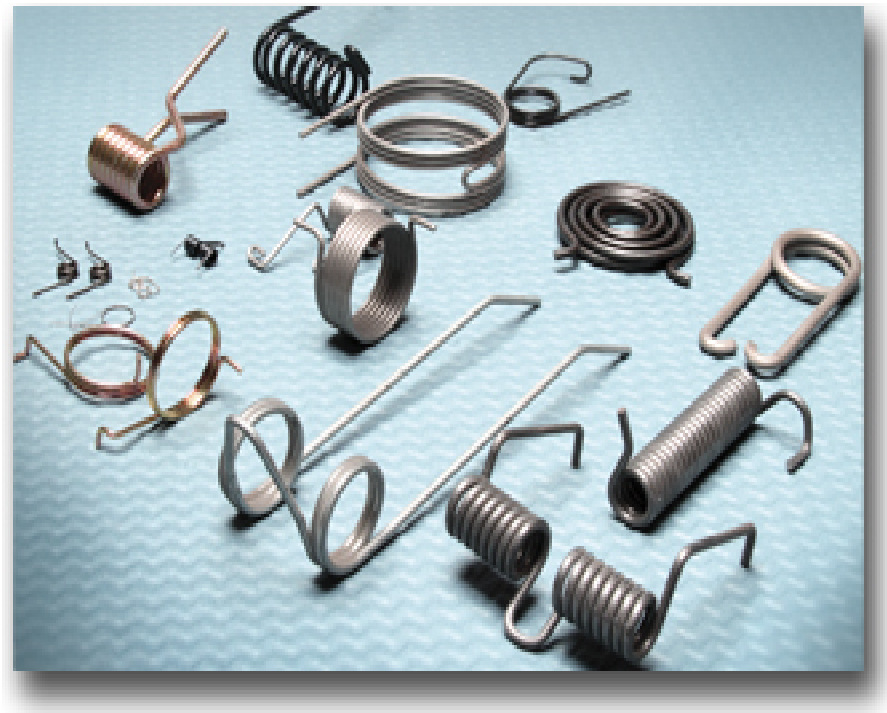
$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + Bs + k}$$

Basic Elements of Rotational Mechanical Systems

Rotational Spring

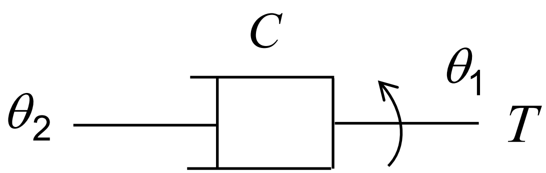


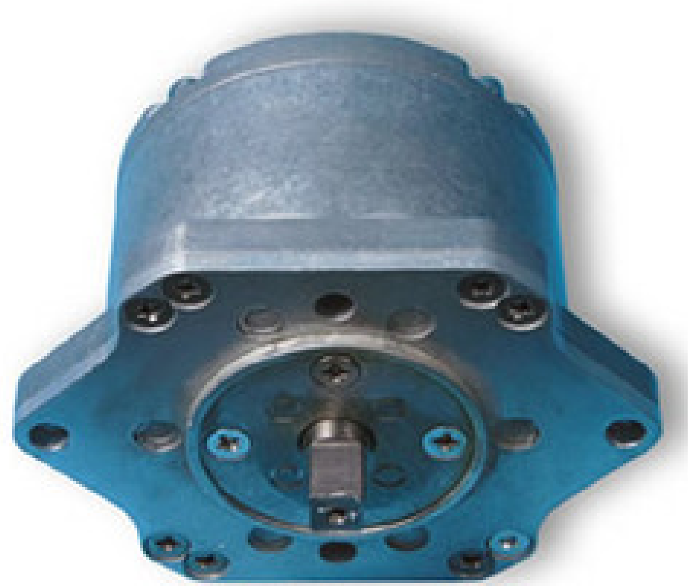
$$T = k(\theta_1 - \theta_2)$$



Basic Elements of Rotational Mechanical Systems

Rotational Damper

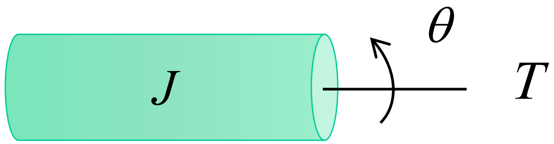




$$T = C(\dot{\theta}_1 - \dot{\theta}_2)$$

Basic Elements of Rotational Mechanical Systems

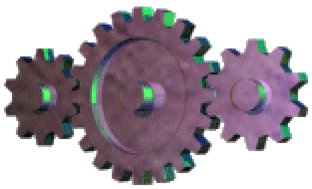
Moment of Inertia

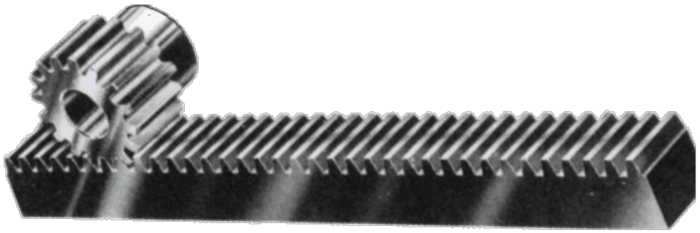


$$T = J\ddot{\theta}$$

Gear

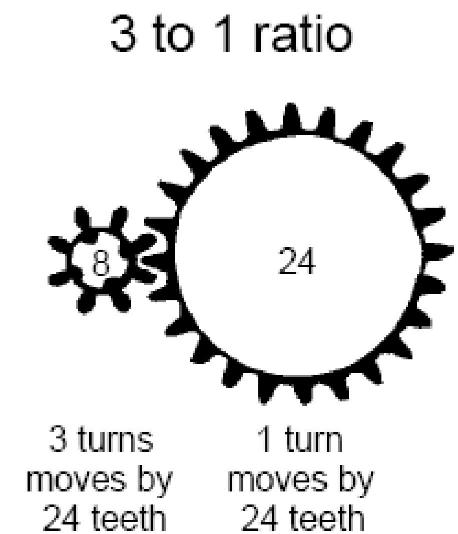
- **Gear** is a toothed machine part, such as a wheel or cylinder, that meshes with another toothed part to transmit motion or to change speed or direction.





Gearing Up and Down

- Gearing up is able to convert torque to velocity.
- The more velocity gained, the more torque sacrifice.
- The ratio is exactly the same: if you get three times your original angular velocity, you reduce the resulting torque to one third.
- This conversion is symmetric: we can also convert velocity to torque at the same ratio.
- The price of the conversion is power loss due to friction.

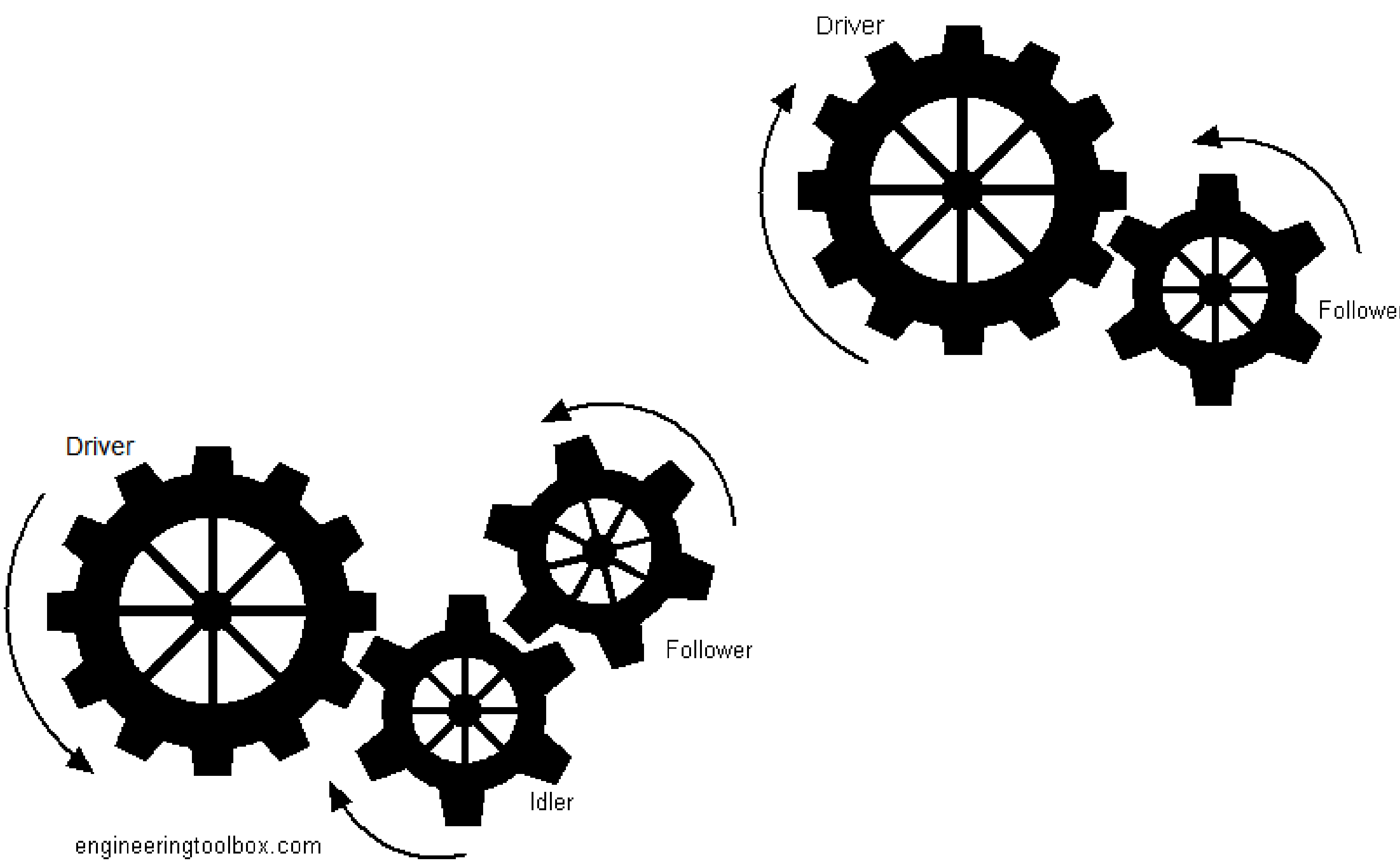


Why Gearing is necessary?

- A typical DC motor operates at speeds that are far too high to be useful, and at torques that are far too low.

- *Gear reduction* is the standard method by which a motor is made useful.

Gear Trains


Driver
Follower
Driver
Follower
Idler
engineeringtoolbox.com

Gear Ratio

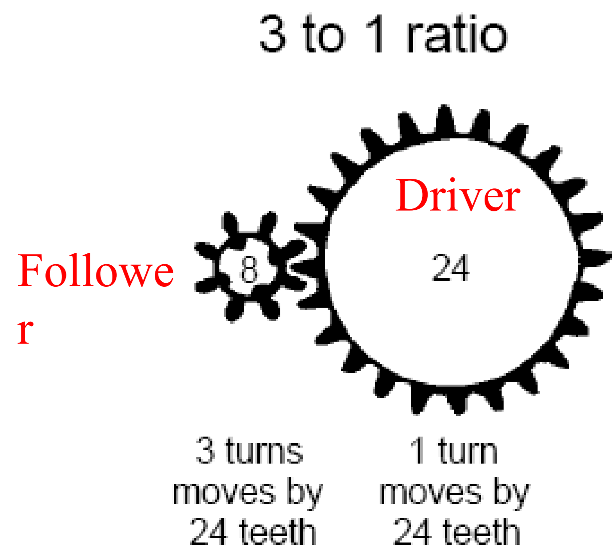
- You can calculate the **gear ratio** by using the number of teeth of the *driver* divided by the number of teeth of the *follower*.

- We *gear up* when we increase velocity and decrease torque.

  Ratio: 3:1

- We *gear down* when we increase torque and reduce velocity.

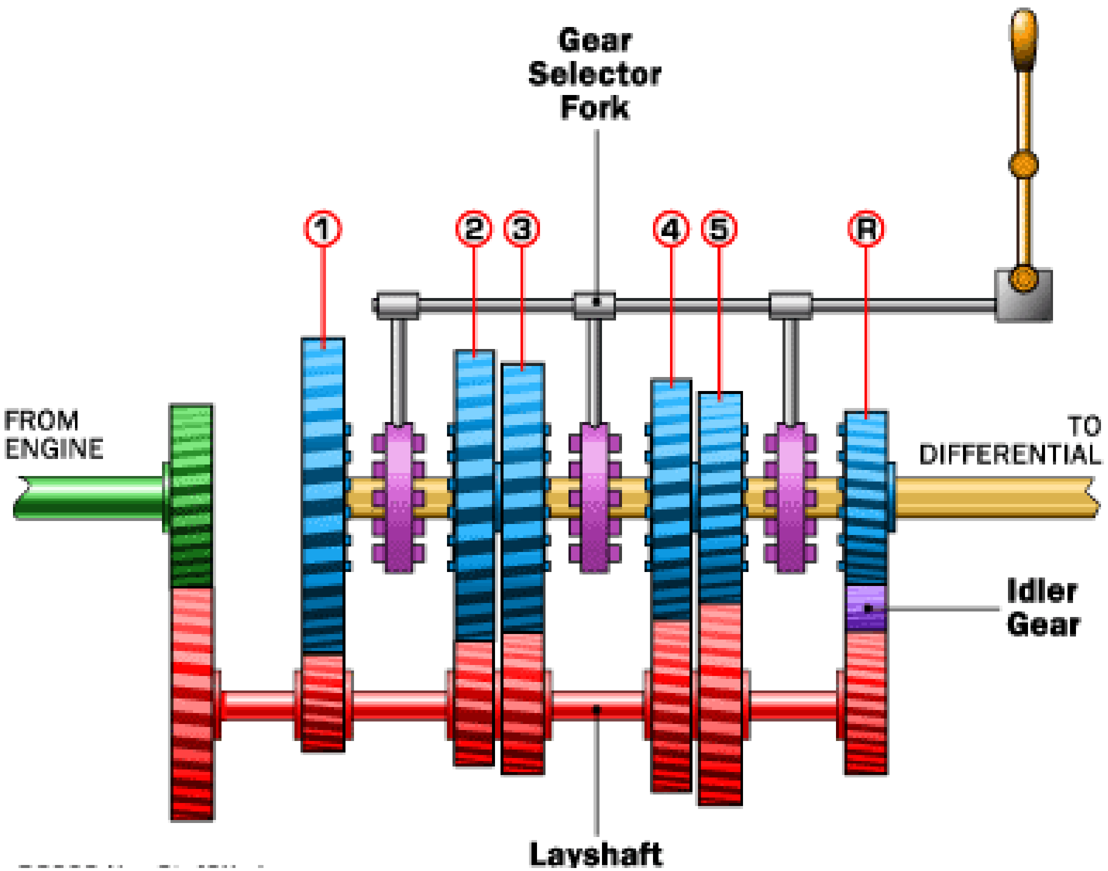
  Ratio: 1:3



$$Gear\ ratio = \frac{number\ of\ teeth\ of\ input\ gear}{number\ of\ teeth\ of\ ouput\ gear} = \frac{Input\ Torque}{Ouput\ Torque} = \frac{Output\ Speed}{Input\ Speed}$$

Example of Gear Trains

- A most commonly used example of gear trains is the gears of an automobile.

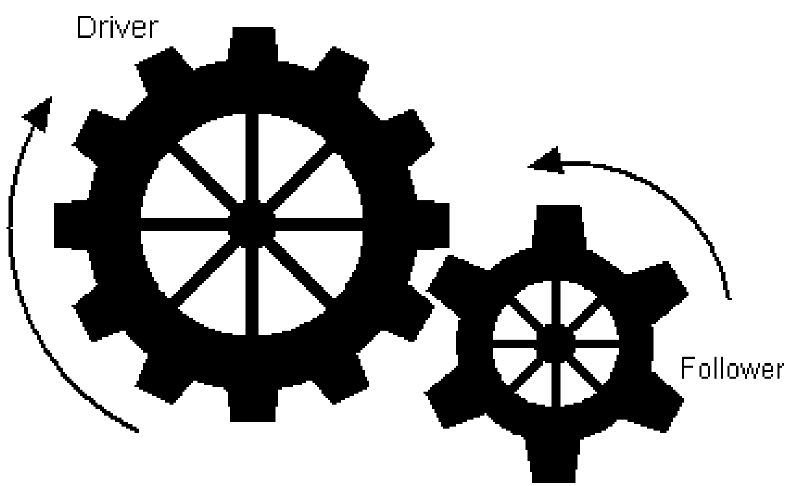


Mathematical Modeling of Gear Trains

- Gears increase or descrease angular velocity (while simultaneously decreasing or increasing torque, such that energy is conserved).

Energy of Driving Gear = Energy of Following Gear

$$N_1\theta_1 = N_2\theta_2$$



$N_1 \longrightarrow$ Number of Teeth of Driving Gear

$\theta_1 \longrightarrow$ Angular Movement of Driving Gear

$N_2 \longrightarrow$ Number of Teeth of Following Gear

$\theta_2 \longrightarrow$ Angular Movement of Following Gear

$$\boxed{J_{eq} = J_1 + \left(\frac{N_1}{N_2}\right)^2 J_2}$$

$$\boxed{B_{eq} = B_1 + \left(\frac{N_1}{N_2}\right)^2 B_2}$$

Mathematical Modelling of Gear Trains

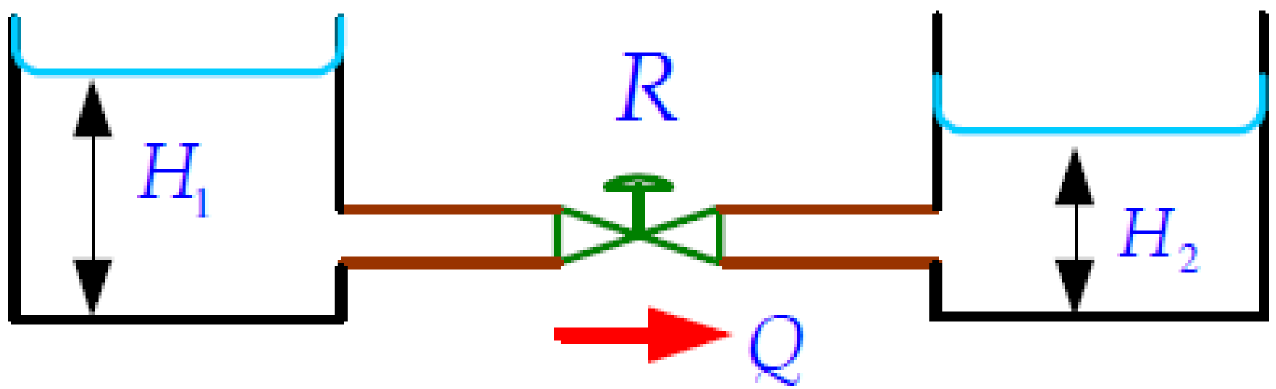
- For three gears connected together

$$J_{eq} = J_1 + \left(\frac{N_1}{N_2}\right)^2 J_2 + \left(\frac{N_1}{N_2}\right)^2 \left(\frac{N_3}{N_4}\right)^2 J_3$$

$$B_{eq} = B_1 + \left(\frac{N_1}{N_2}\right)^2 B_2 + \left(\frac{N_1}{N_2}\right)^2 \left(\frac{N_3}{N_4}\right)^2 B_3$$

Resistance of Liquid-Level Systems

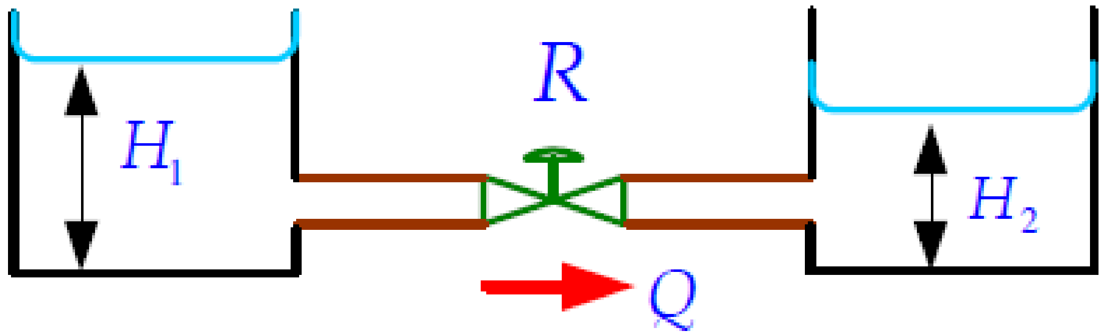
- Consider the flow through a short pipe connecting two tanks as shown in Figure.



- Where H_1 is the height (or level) of first tank, H_2 is the height of second tank, R is the resistance in flow of liquid and Q is the flow rate.

Resistance of Liquid-Level Systems

- The resistance for liquid flow in such a pipe is defined as the change in the level difference necessary to cause a unit change inflow rate.



$$Resistance = \frac{change\ in\ level\ difference}{change\ in\ flow\ rate} = \frac{m}{m^3/s}$$

$$R = \frac{\Delta(H_1 - H_2)}{\Delta Q} = \frac{m}{m^3/s}$$

Resistance in Laminar Flow

- For laminar flow, the relationship between the steady-state flow rate and steady state height at the restriction is given by:

$$Q = k_l H$$

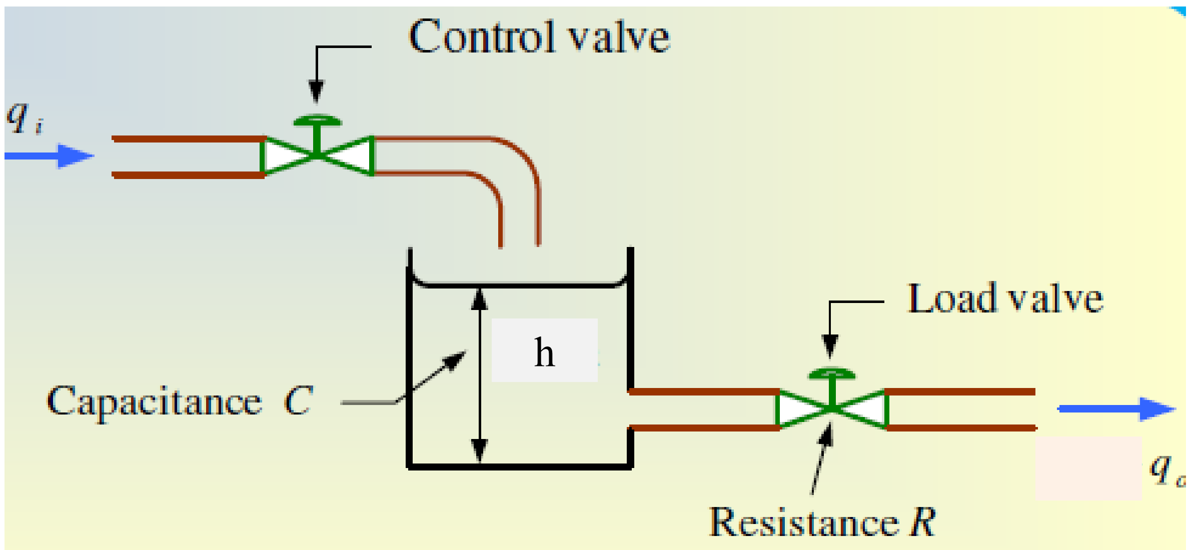
- Where Q = steady-state liquid flow rate in m/s^3
- *K_l = constant in m/s^2*
- *and H = steady-state* height in *m*.

- The resistance R_ℓ is

$$R_l = \frac{dH}{dQ}$$

Capacitance of Liquid-Level Systems

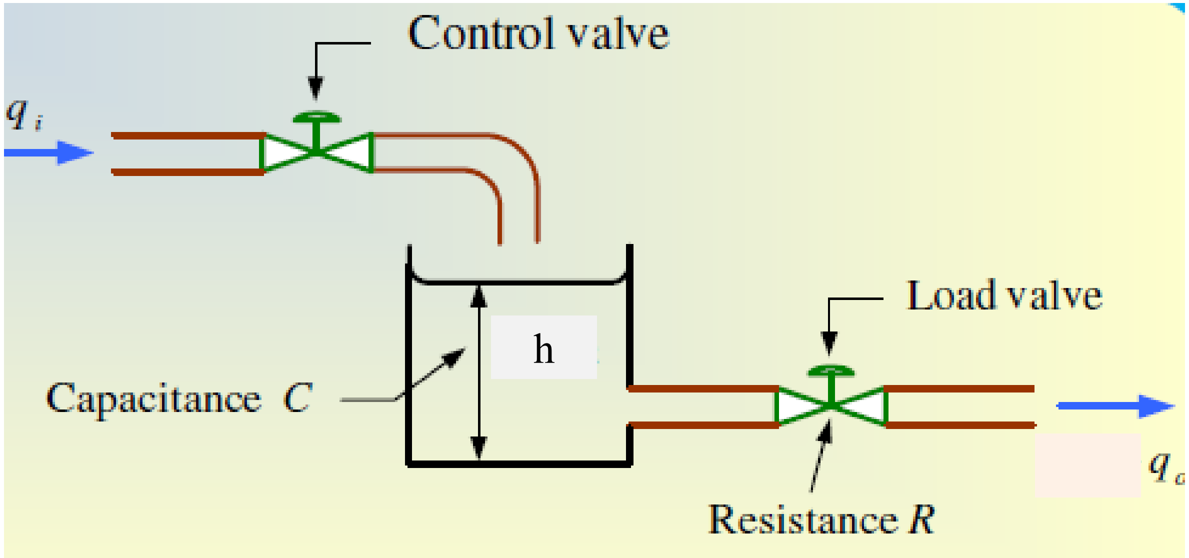
- The capacitance of a tank is defined to be the change in quantity of stored liquid necessary to cause a unity change in the height.



$$Capacitance = \frac{change\ in\ liquid\ \ stored}{change\ in\ height} = \frac{m^3}{m}\ or\ m^2$$

- Capacitance (C) is cross sectional area (A) of the tank.

Capacitance of Liquid-Level Systems

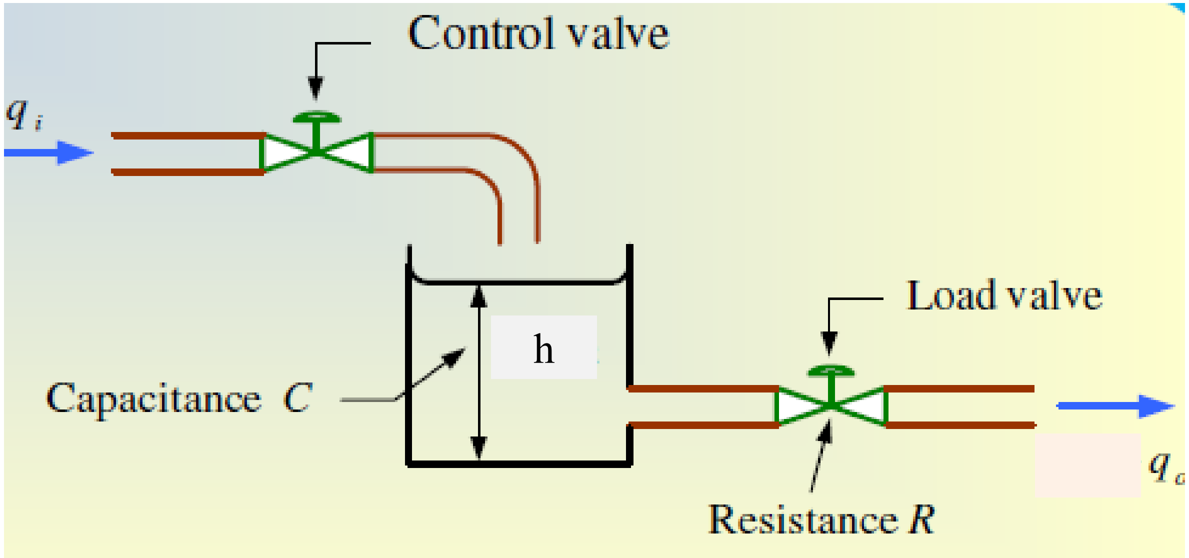


$$Rate\ of\ change\ of\ fluid\ volume\ in\ the\ tank = flow\ in - flow\ out$$

$$\frac{dV}{dt} = q_i - q_o$$

$$\frac{d(A \times h)}{dt} = q_i - q_o$$

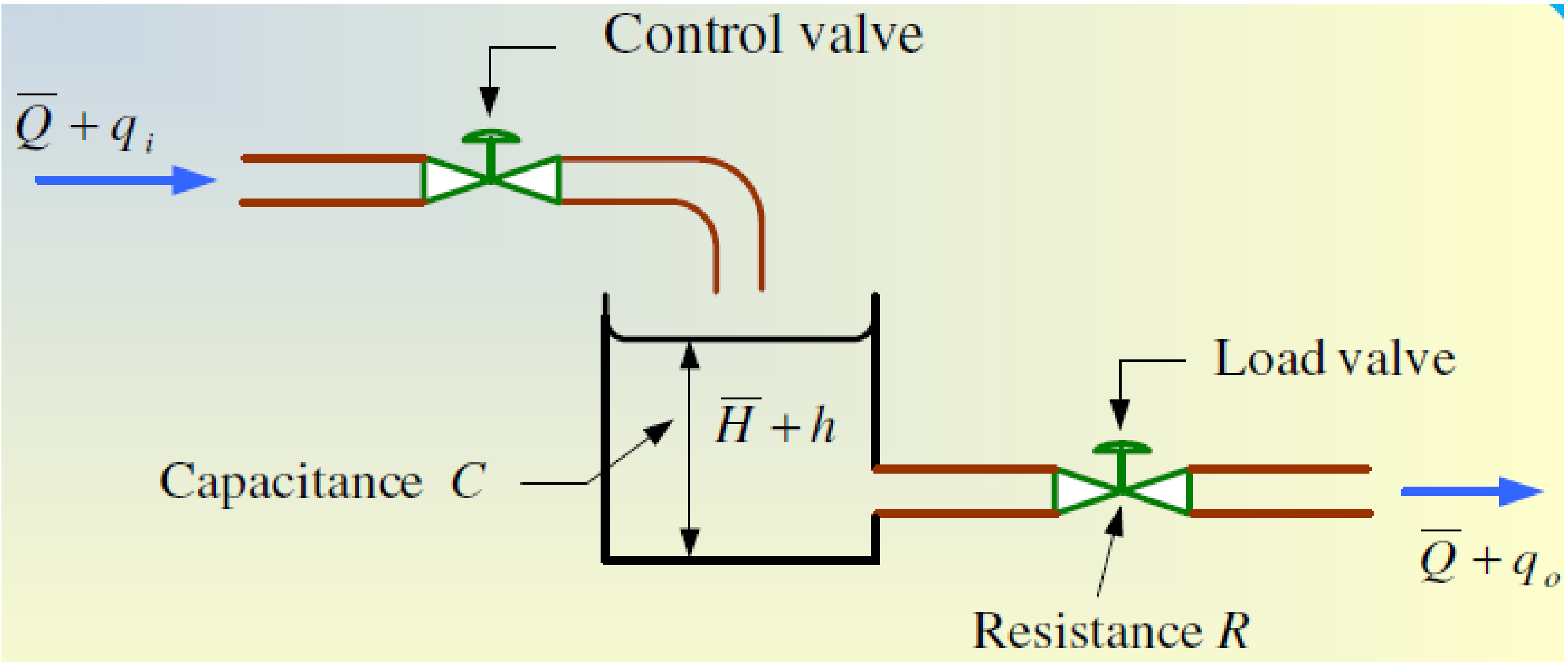
Capacitance of Liquid-Level Systems



$$A\frac{dh}{dt} = q_i - q_o$$

$$\boxed{C\frac{dh}{dt} = q_i - q_o}$$

Modelling Example#1



$\overline{H} =$ steady-state head (before any change has occurred), m.

$h =$ small deviation of head from its steady-state value, m.

$\overline{Q} =$ steady-state flow rate (before any change has occurred), m^3/s.

$q_i =$ small deviation of inflow rate from its steady-state value, m^3/s.

$q_o =$ small deviation of outflow rate from its steady-state value, m^3/s.

Modelling Example#1

- The rate of change in liquid stored in the tank is equal to the flow in minus flow out.

$$C\frac{dh}{dt} = q_i - q_o \longrightarrow \quad (1)$$

- The resistance *R* may be written as

$$R = \frac{dH}{dQ} = \frac{h}{q_0} \longrightarrow \quad (2)$$

- Rearranging equation (2)

$$q_0 = \frac{h}{R} \longrightarrow \quad (3)$$

Modelling Example#1

$$C\frac{dh}{dt} = q_i - q_o \longrightarrow \quad (1) \qquad q_0 = \frac{h}{R} \longrightarrow \quad (4)$$

- Substitute q_o in equation (3)

$$C\frac{dh}{dt} = q_i - \frac{h}{R}$$

- After simplifying above equation

$$RC\frac{dh}{dt} + h = Rq_i$$

- Taking Laplace transform considering initial conditions to zero

$$RCsH(s) + H(s) = RQ_i(s)$$

Modelling Example#1

$$RCsH(s) + H(s) = RQ_i(s)$$

- The transfer function can be obtained as

$$\frac{H(s)}{Q_i(s)} = \frac{R}{(RCs + 1)}$$

Example 3.7. A water heater.

The inflow of water to the water heater has the mass flow rate $\dot{m}_1$ and temperature T_1 whereas the outflow has the mass flow rate $\dot{m}_2$ and temperature T_2. The mass of water in the heater is M and it is heated to a temperature T with a heating power $\dot{Q}$. The mixing of water in the heater is assumed to be perfect.

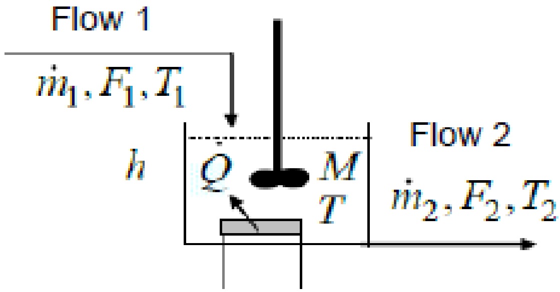


Fig. 3.8. A water heater.

How do the amount of water and the temperature in the heater depend on other variables?

Mass balance:

$$\frac{dM}{dt} = \dot{m}_1 - \dot{m}_2 \tag{1}$$

Energy balance:

$$\frac{dE}{dt} = \dot{E}_1 - \dot{E}_2 + \dot{Q} \tag{2}$$

Here, $\dot{E}_1$ and $\dot{E}_2$ are energy flows associated with the inflow and the outflow, respectively.

The energy in a substance is proportional to its mass or mass flow rate. For liquids it applies with good accuracy that the energy is also proportional to its temperature. This results in the

constitutive relationships: $E = c_p T M,\ \dot{E}_1 = c_p T_1 \dot{m}_1,\ \dot{E}_2 = c_p T_2 \dot{m}_2$ (3)

Here c_p is the *specific heat capacity* for water, which in this case is assumed to be constant independently of the water temperature. Combination of (2) and (3) and development of the derivative according to the product rule give

$$T \frac{\mathrm{d}M}{\mathrm{d}t} + M \frac{\mathrm{d}T}{\mathrm{d}t} = T_1 \dot{m}_1 - T_2 \dot{m}_2 + \frac{\dot{Q}}{c_p} \tag{4}$$

Because of the assumption of perfect mixing, there is also a

constitutive relationship: $T_2 = T$ (5)

Elimination of $\mathrm{d}M/\mathrm{d}t$ from (4) by (1) and substitution of (5) give

$$M \frac{\mathrm{d}T}{\mathrm{d}t} = \dot{m}_1 (T_1 - T) + \frac{\dot{Q}}{c_p} \tag{6}$$

Equation (1) and (6) show how the mass and the temperature in the heater depend on the inflow and the heating power $\dot{Q}$.

The energy in a substance is proportional to its mass or mass flow rate. For liquids it applies with good accuracy that the energy is also proportional to its temperature. This results in the

constitutive relationships: $E = c_\mathrm{p} T M,\ \dot{E}_1 = c_\mathrm{p} T_1 \dot{m}_1,\ \dot{E}_2 = c_\mathrm{p} T_2 \dot{m}_2$ (3)

Here c_p is the *specific heat capacity* for water, which in this case is assumed to be constant independently of the water temperature. Combination of (2) and (3) and development of the derivative according to the product rule give

$$T \frac{\mathrm{d}M}{\mathrm{d}t} + M \frac{\mathrm{d}T}{\mathrm{d}t} = T_1 \dot{m}_1 - T_2 \dot{m}_2 + \frac{\dot{Q}}{c_\mathrm{p}} \tag{4}$$

Because of the assumption of perfect mixing, there is also a

constitutive relationship: $T_2 = T$ (5)

Elimination of $\mathrm{d}M/\mathrm{d}t$ from (4) by (1) and substitution of (5) give

$$M \frac{\mathrm{d}T}{\mathrm{d}t} = \dot{m}_1 (T_1 - T) + \frac{\dot{Q}}{c_\mathrm{p}} \tag{6}$$

Equation (1) and (6) show how the mass and the temperature in the heater depend on the inflow and the heating power $\dot{Q}$.

If we want to use volumetric units instead of mass units in the model, this can easily be accomplished by the substitutions

$$M = \rho A h, \quad \dot{m}_1 = \rho_1 F_1 \tag{7}$$

which applied to (6) yield

$$\rho A h \frac{\mathrm{d}T}{\mathrm{d}t} = \rho_1 F_1 (T_1 - T) + \frac{\dot{Q}}{c_\mathrm{p}} \tag{8}$$

Note that the *water density is not assumed to be constant* in equation (8).

Equation (1) expressed in volumetric units becomes more complicated when the water density is non-constant., i.e.,

$$A \frac{\mathrm{d}\rho h}{\mathrm{d}t} = \rho_1 F_1 - \rho_2 F_2 = \rho_1 F_1 - \rho F_2 \tag{9}$$

It is possible to show that even if $\rho \neq \rho_1$ due to the fact that $T \neq T_1$, the effects tend to cancel out in such a way that

$$A \frac{\mathrm{d}h}{\mathrm{d}t} \approx F_1 - F_2 \tag{10}$$

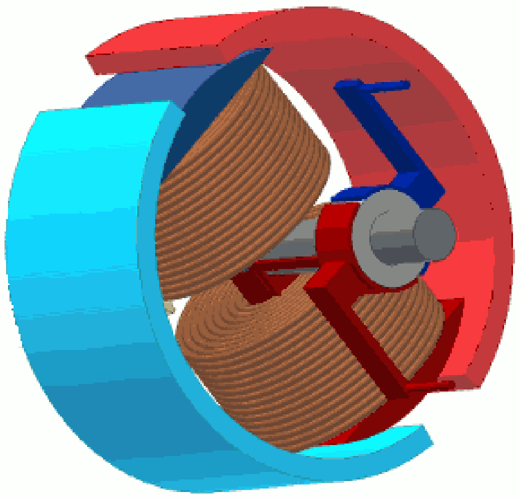
becomes a good approximation of (1) and (9).

Electromechanical Systems

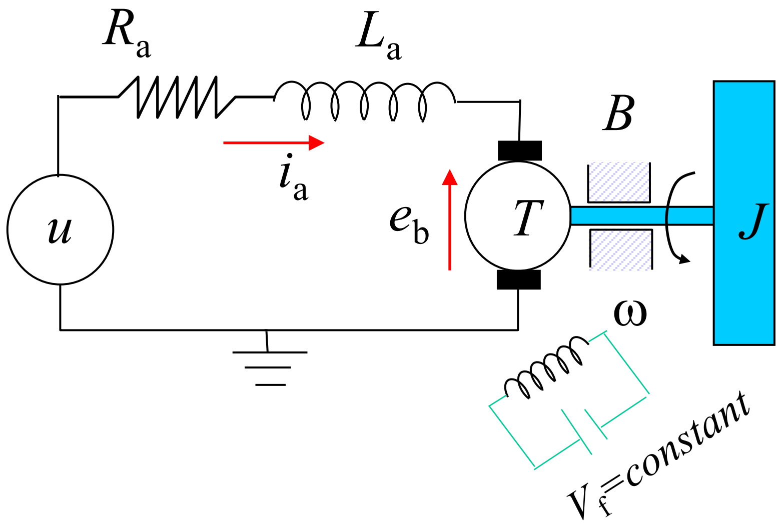
- **Electromechanics** combines electrical and mechanical processes.
- Devices which carry out electrical operations by using moving parts are known as electromechanical.
  - Relays
  - Solenoids
  - Electric Motors
  - Switches and e.t.c

D.C Drives

- Speed control can be achieved using DC drives in a number of ways.
- Variable Voltage can be applied to the armature terminals of the DC motor .
- Another method is to vary the flux per pole of the motor.
- The first method involve adjusting the motor’s armature while the latter method involves adjusting the motor field. These methods are referred to as “armature control” and “field control.”



Example-2: Armature Controlled D.C Motor



<u>Input</u>: voltage u
<u>Output</u>: Angular velocity ω

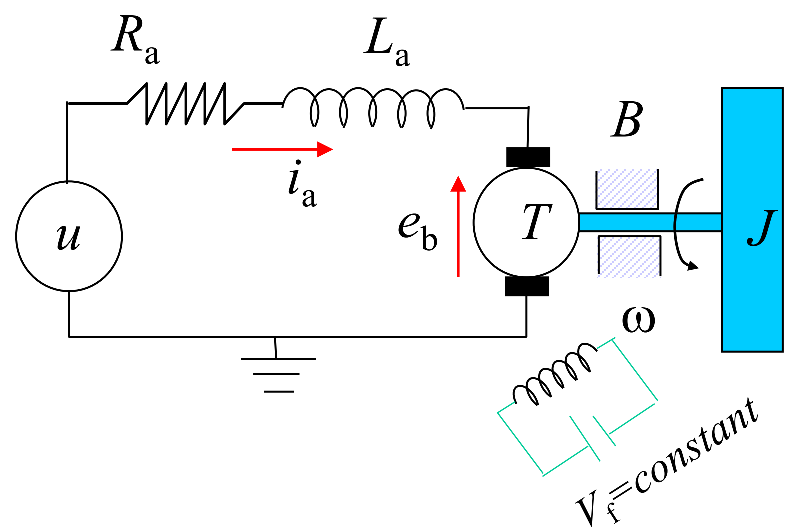
<u>Electrical Subsystem</u> (loop method):

$$u = R_a i_a + L_a \frac{di_a}{dt} + e_b, \qquad where \quad e_b = back\text{-}emf\ voltage$$

<u>Mechanical Subsystem</u>

$$T_{motor} = J\dot{\omega} + B\omega$$

Example-2: Armature Controlled D.C Motor



Power Transformation:

Torque-Current: $T_{motor} = K_t i_a$

Voltage-Speed: $e_b = K_b \omega$

where K_t: torque constant, K_b: velocity constant For an ideal motor

$$K_t = K_b$$

Combing previous equations results in the following mathematical model:

$$\begin{cases} L_a \dfrac{di_a}{dt} + R_a i_a + K_b \omega = u \\ J\dot{\omega} + B\omega - K_t i_a = 0 \end{cases}$$

Example-2: Armature Controlled D.C Motor

Taking Laplace transform of the system's differential equations with zero initial conditions gives:

$$\begin{cases} \left(L_a s + R_a\right) I_a(s) + K_b \Omega(s) = U(s) \\ \left(Js + B\right)\Omega(s) - K_t I_a(s) = 0 \end{cases}$$

Eliminating I_a yields the input-output transfer function

$$\frac{\Omega(s)}{U(s)} = \frac{K_t}{L_a J s^2 + \left(JR_a + BL_a\right)s + BR_a + K_t K_b}$$

Example-2: Armature Controlled D.C Motor Reduced Order Model

Assuming small inductance, $L_a \approx 0$

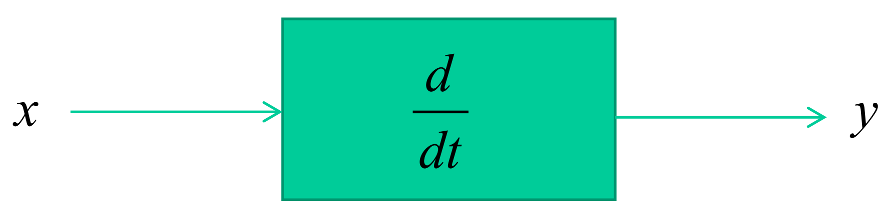
$$\frac{\Omega(s)}{U(s)} = \frac{\left(K_t / R_a\right)}{Js + \left(B + K_t K_b / R_a\right)}$$

Introduction

- A Block Diagram is a shorthand pictorial representation of the cause-and-effect relationship of a system.

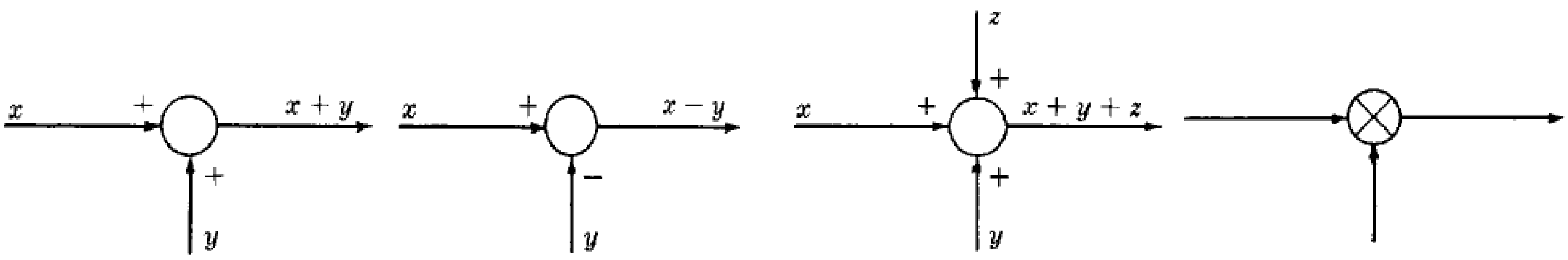
- The interior of the rectangle representing the block usually contains a description of or the name of the element, gain, or the symbol for the mathematical operation to be performed on the input to yield the output.

- The arrows represent the direction of information or signal flow.



Introduction

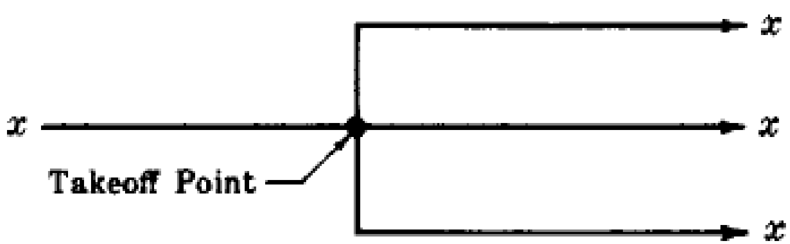
- The operations of addition and subtraction have a special representation.
- The block becomes a small circle, called a summing point, with the appropriate plus or minus sign associated with the arrows entering the circle.
- The output is the algebraic sum of the inputs.
- Any number of inputs may enter a summing point.
- Some books put a cross in the circle.

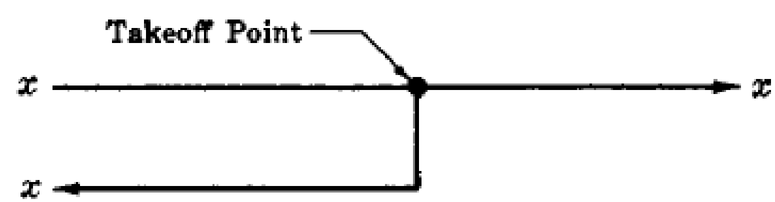


Introduction

- In order to have the same signal or variable be an input to more than one block or summing point, a takeoff (or pickoff) point is used.

- This permits the signal to proceed unaltered along several different paths to several destinations.

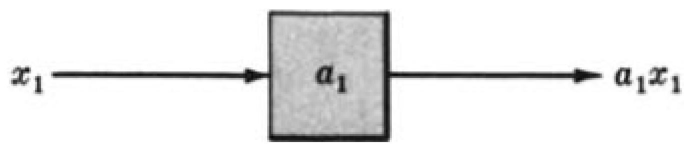


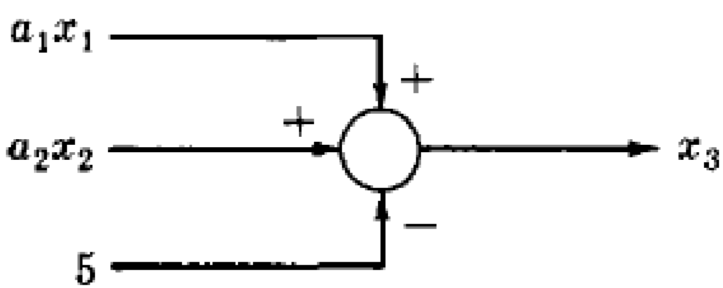


Example-1

- Consider the following equations in which x_1, x_2, x_3, are variables, and a_1, a_2 *are* general coefficients or mathematical operators.

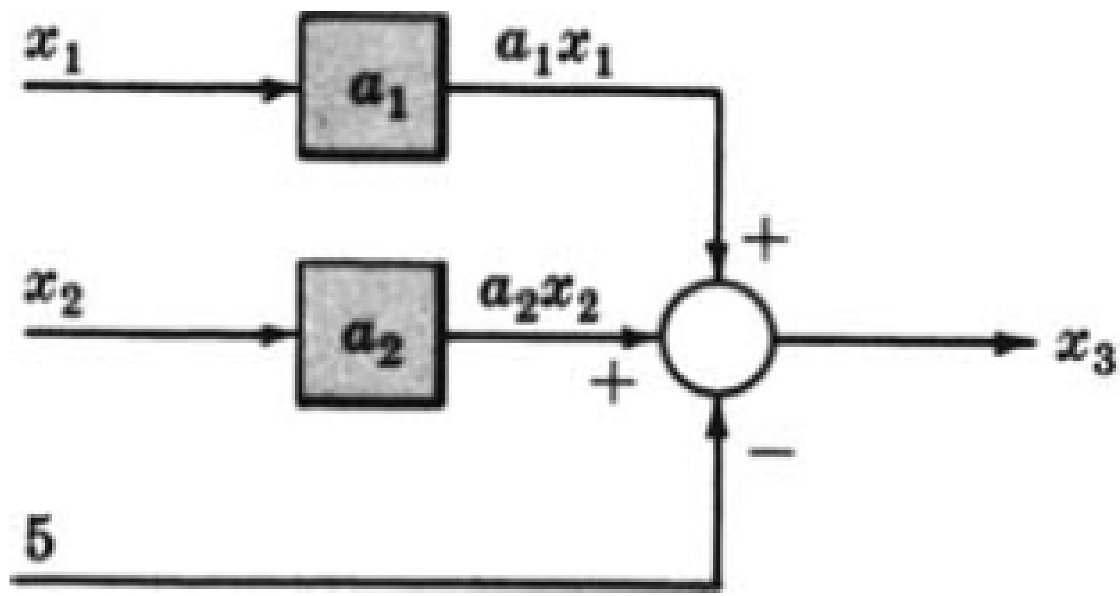
$$x_3 = a_1x_1 + a_2x_2 - 5$$





Example-1

$$x_3 = a_1 x_1 + a_2 x_2 - 5$$



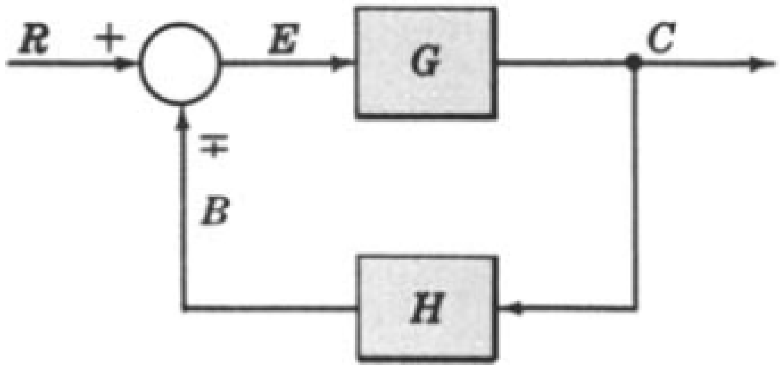
Example-2

- Draw the Block Diagrams of the following equations.

(1) $$x_2 = a_1 \frac{dx_1}{dt} + \frac{1}{b}\int x_1 dt$$

(2) $$x_3 = a_1 \frac{d^2 x_2}{dt^2} + 3\frac{dx_1}{dt} - bx_1$$

Canonical Form of A Feedback Control System



$G \equiv$ direct transfer function $\equiv$ forward transfer function

$H \equiv$ feedback transfer function

$GH \equiv$ loop transfer function $\equiv$ open-loop transfer function

$C/R \equiv$ closed-loop transfer function $\equiv$ control ratio $\quad \dfrac{C}{R} = \dfrac{G}{1 \pm GH}$

$E/R \equiv$ actuating signal ratio $\equiv$ error ratio $\quad \dfrac{E}{R} = \dfrac{1}{1 \pm GH}$

$B/R \equiv$ primary feedback ratio $\quad \dfrac{B}{R} = \dfrac{GH}{1 \pm GH}$

Characteristic Equation

- The control ratio is the closed loop transfer function of the system.

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 \pm G(s)H(s)}$$

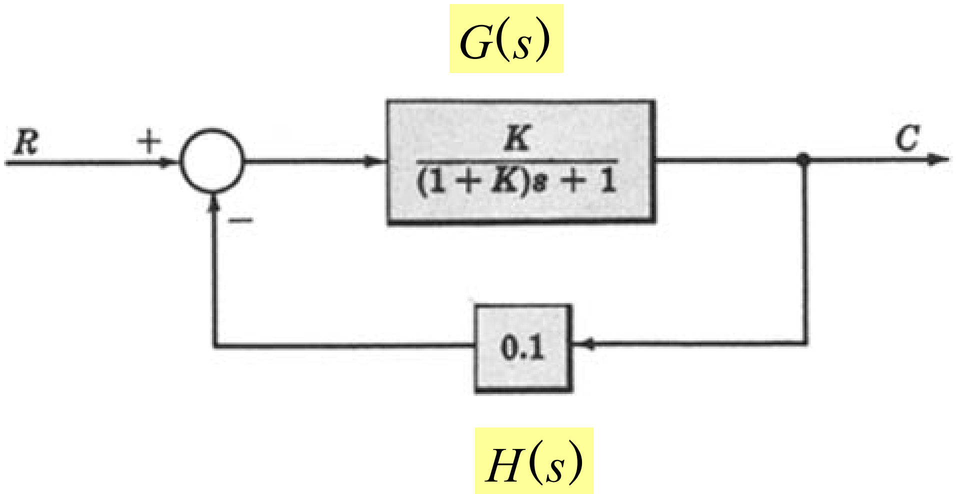
- The denominator of closed loop transfer function determines the characteristic equation of the system.

- Which is usually determined as:

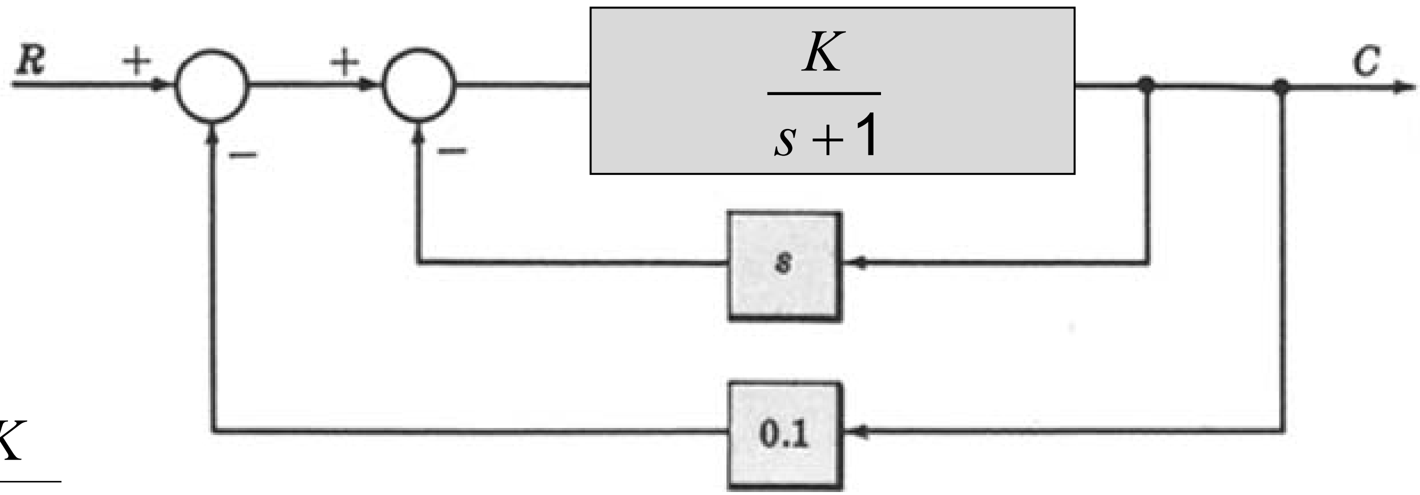
$$1 \pm G(s)H(s) = 0$$

Example-3

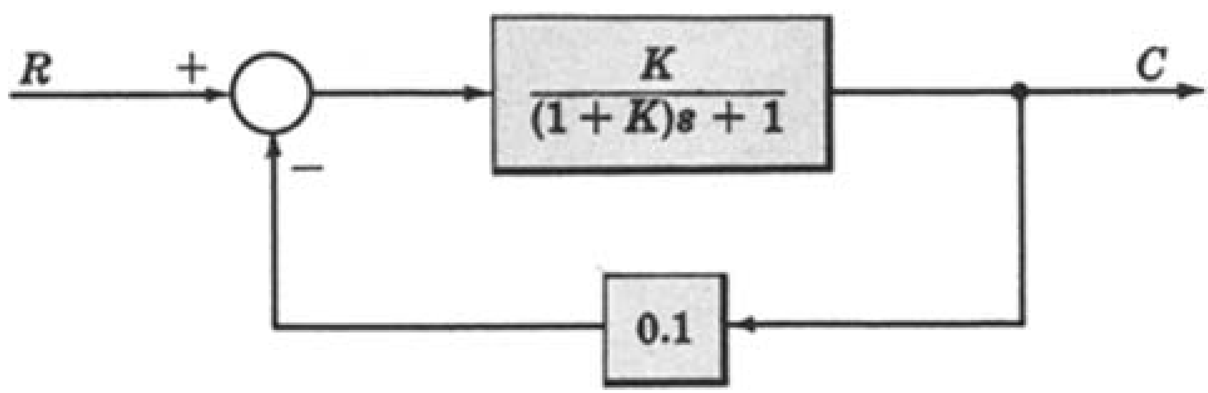
1. Open loop transfer function $$\frac{B(s)}{E(s)} = G(s)H(s)$$
2. Feed Forward Transfer function $$\frac{C(s)}{E(s)} = G(s)$$
3. control ratio $$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}$$
4. feedback ratio $$\frac{B(s)}{R(s)} = \frac{G(s)H(s)}{1+G(s)H(s)}$$
5. error ratio $$\frac{E(s)}{R(s)} = \frac{1}{1+G(s)H(s)}$$
6. closed loop transfer function $$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}$$
7. characteristic equation $$1+G(s)H(s) = 0$$
8. Open loop poles and zeros if 9. closed loop poles and zeros if K=10.



Example-5



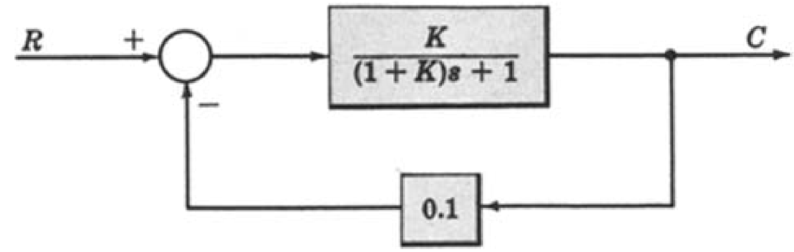
$$\frac{G}{1+GH} = \frac{\frac{K}{s+1}}{1+\frac{K}{s+1}s}$$



Example-5 (see example-3)

1. Open loop transfer function $$\frac{B(s)}{E(s)} = G(s)H(s)$$
2. Feed Forward Transfer function $$\frac{C(s)}{E(s)} = G(s)$$
3. control ratio $$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}$$
4. feedback ratio $$\frac{B(s)}{R(s)} = \frac{G(s)H(s)}{1+G(s)H(s)}$$
5. error ratio $$\frac{E(s)}{R(s)} = \frac{1}{1+G(s)H(s)}$$
6. closed loop transfer function $$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}$$
7. characteristic equation $$1+G(s)H(s) = 0$$
8. closed loop poles and zeros if K=10.

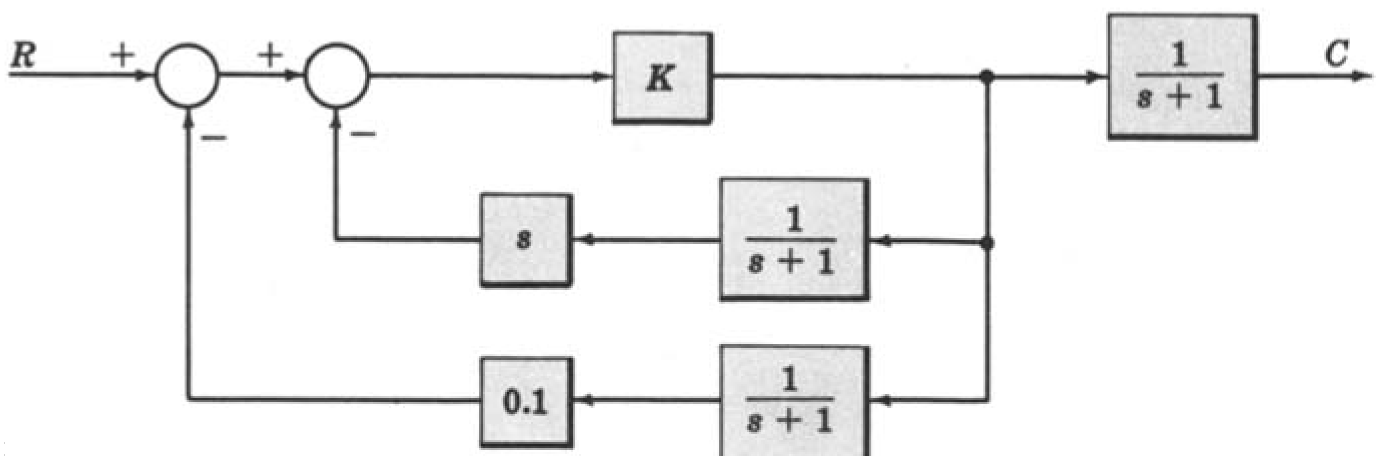
$G(s)$



$H(s)$

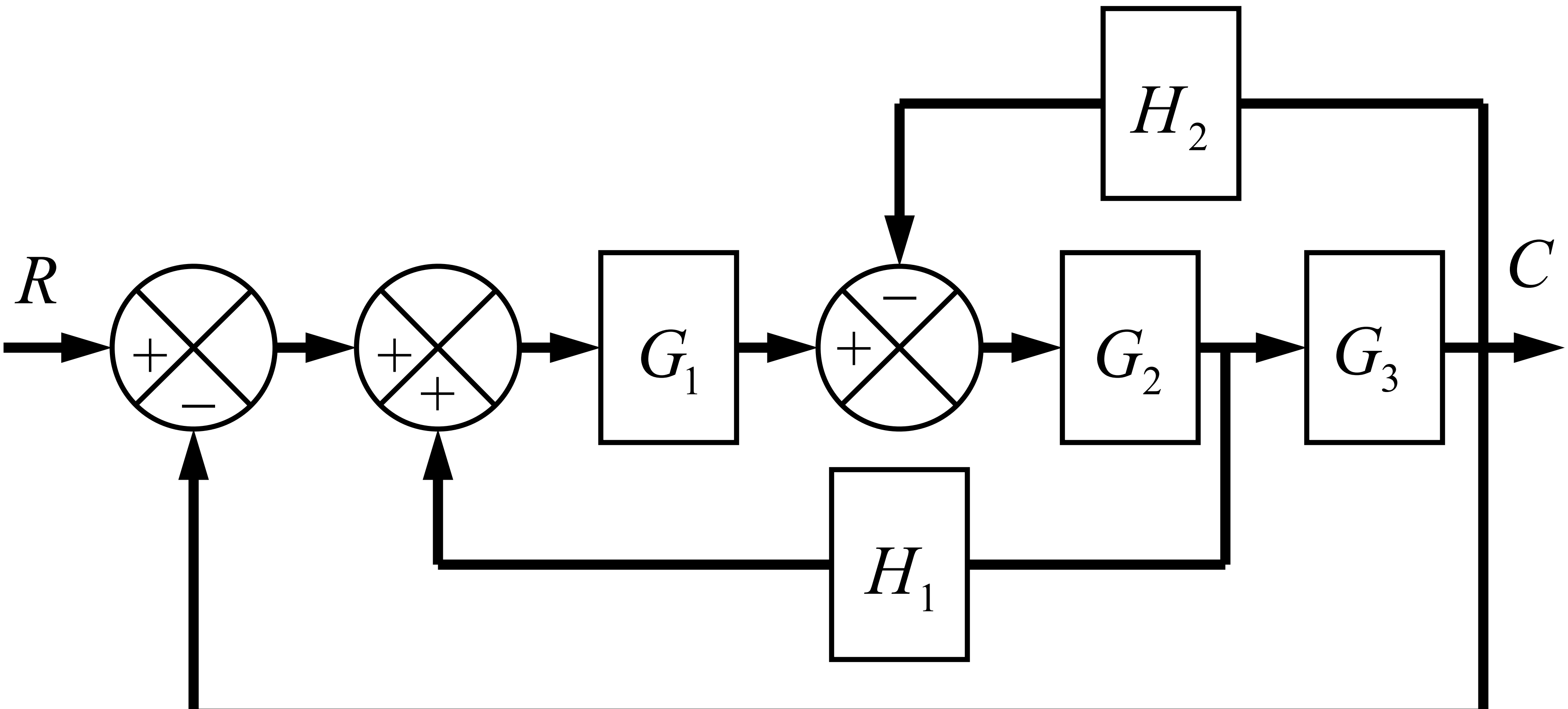
Example-6

- For the system represented by the following block diagram determine:
  1. Open loop transfer function
  2. Feed Forward Transfer function
  3. control ratio
  4. feedback ratio
  5. error ratio
  6. closed loop transfer function
  7. characteristic equation
  8. closed loop poles and zeros if K=100.

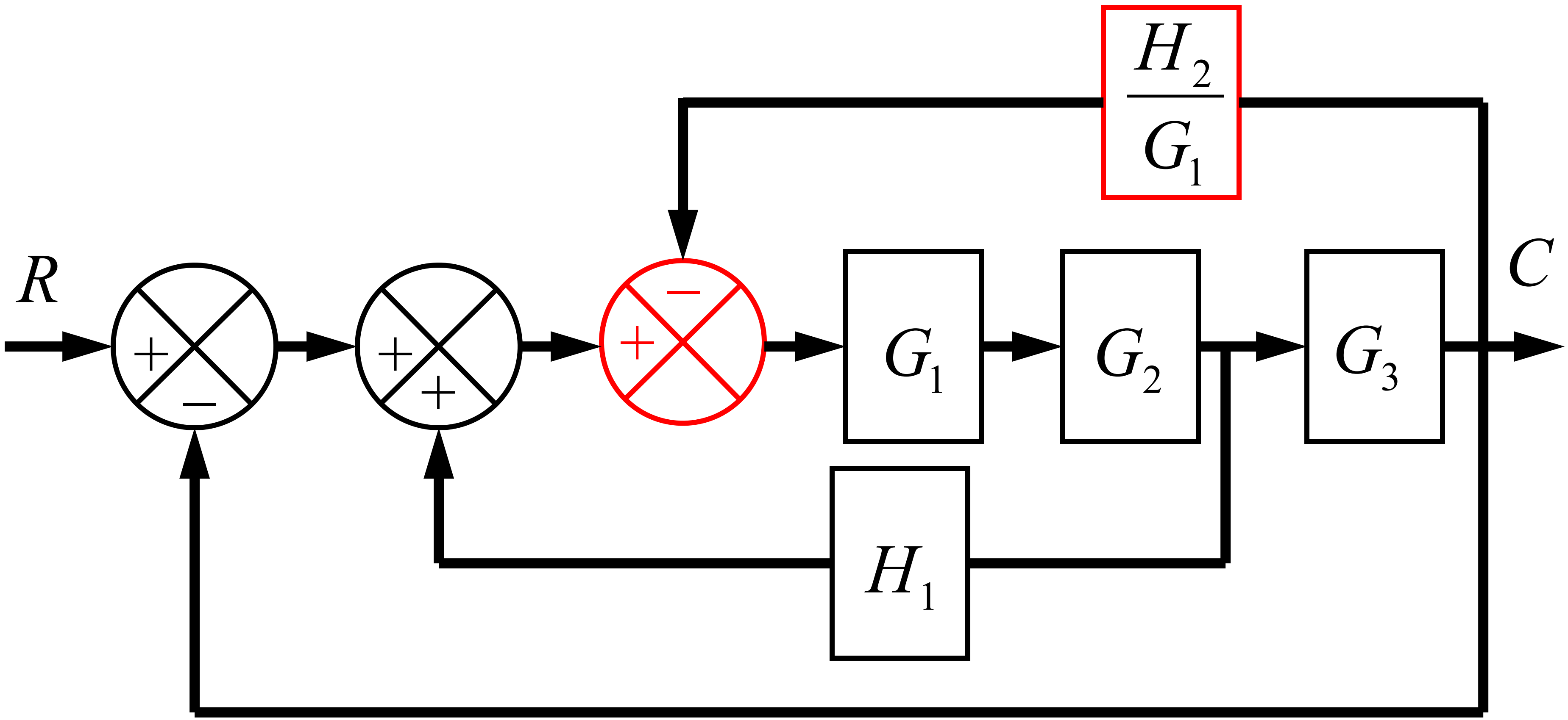


Example-7

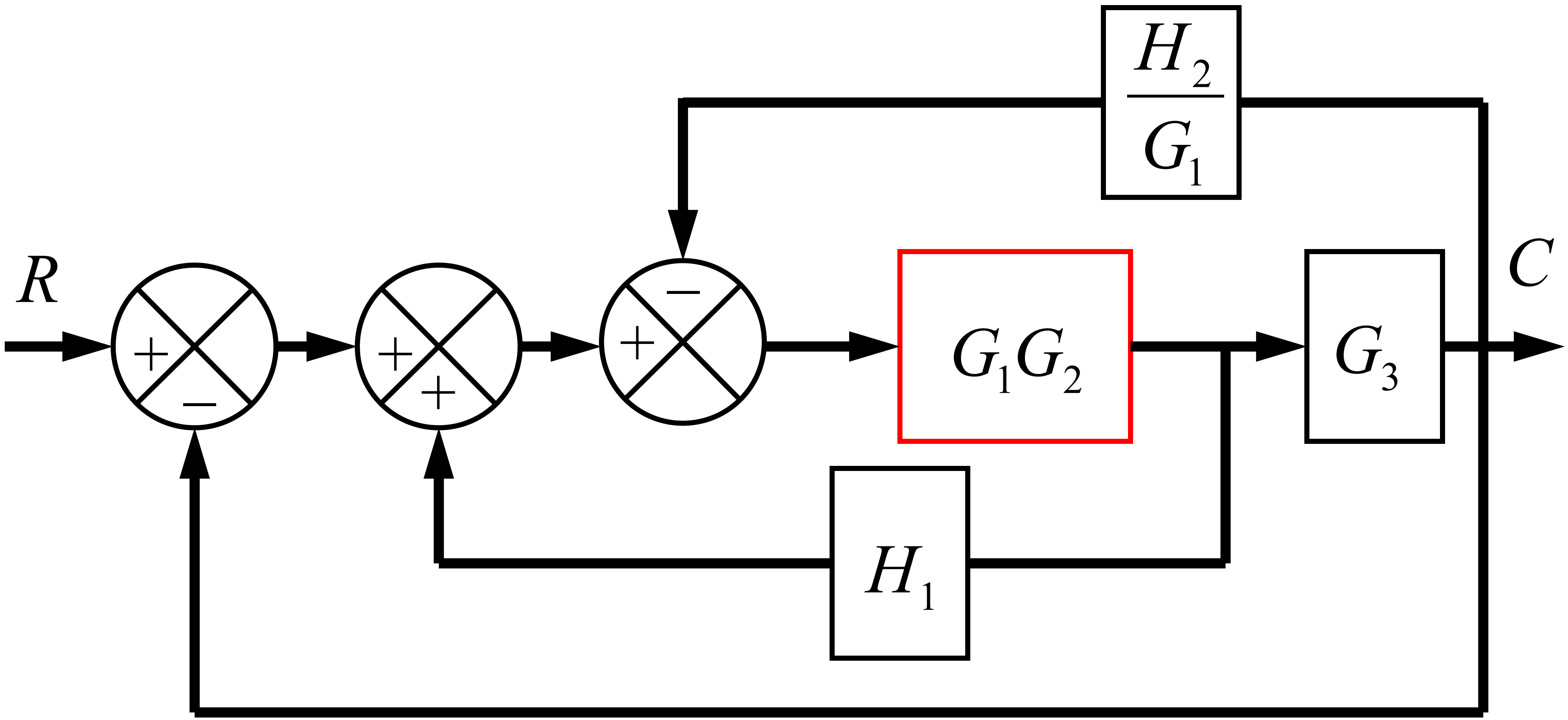
- Reduce the following block diagram to canonical form.



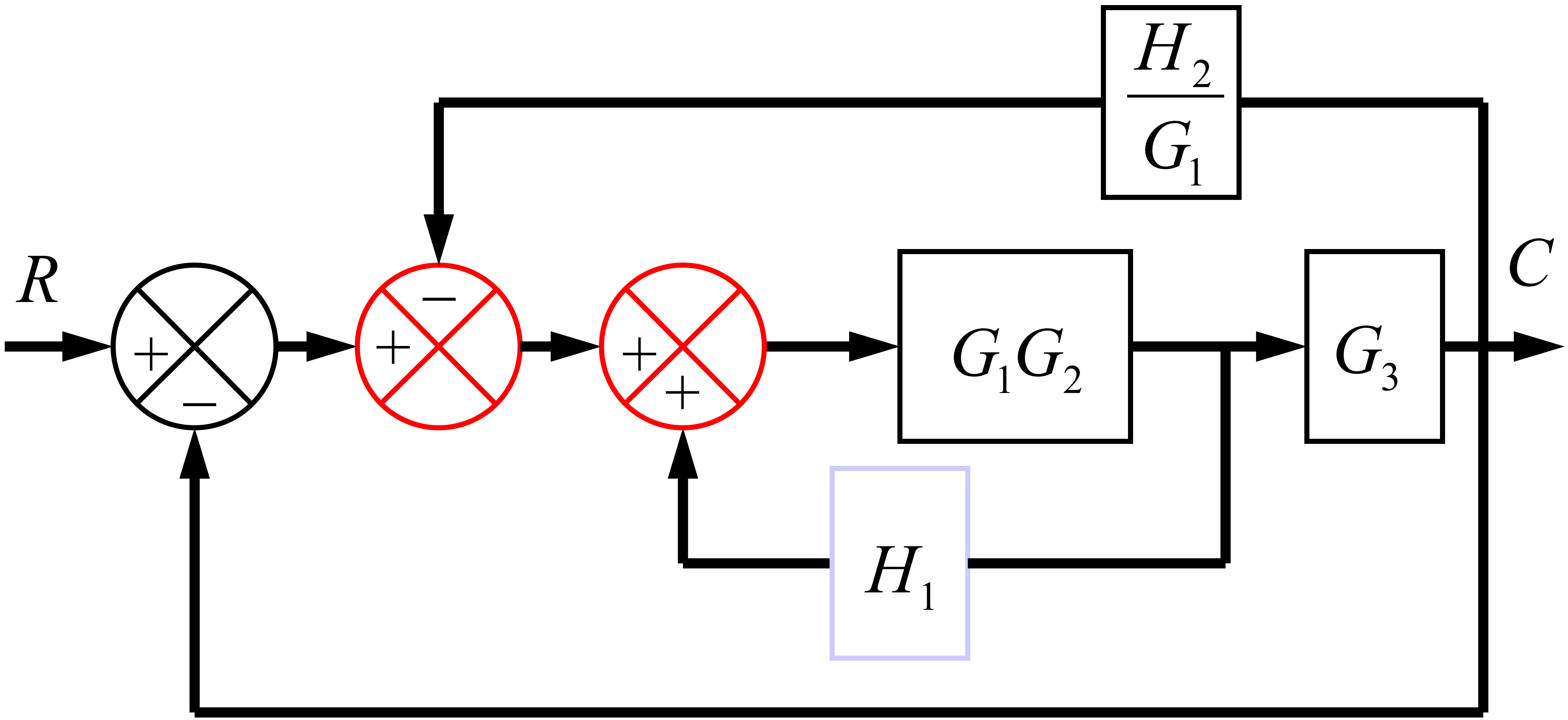
Example-7



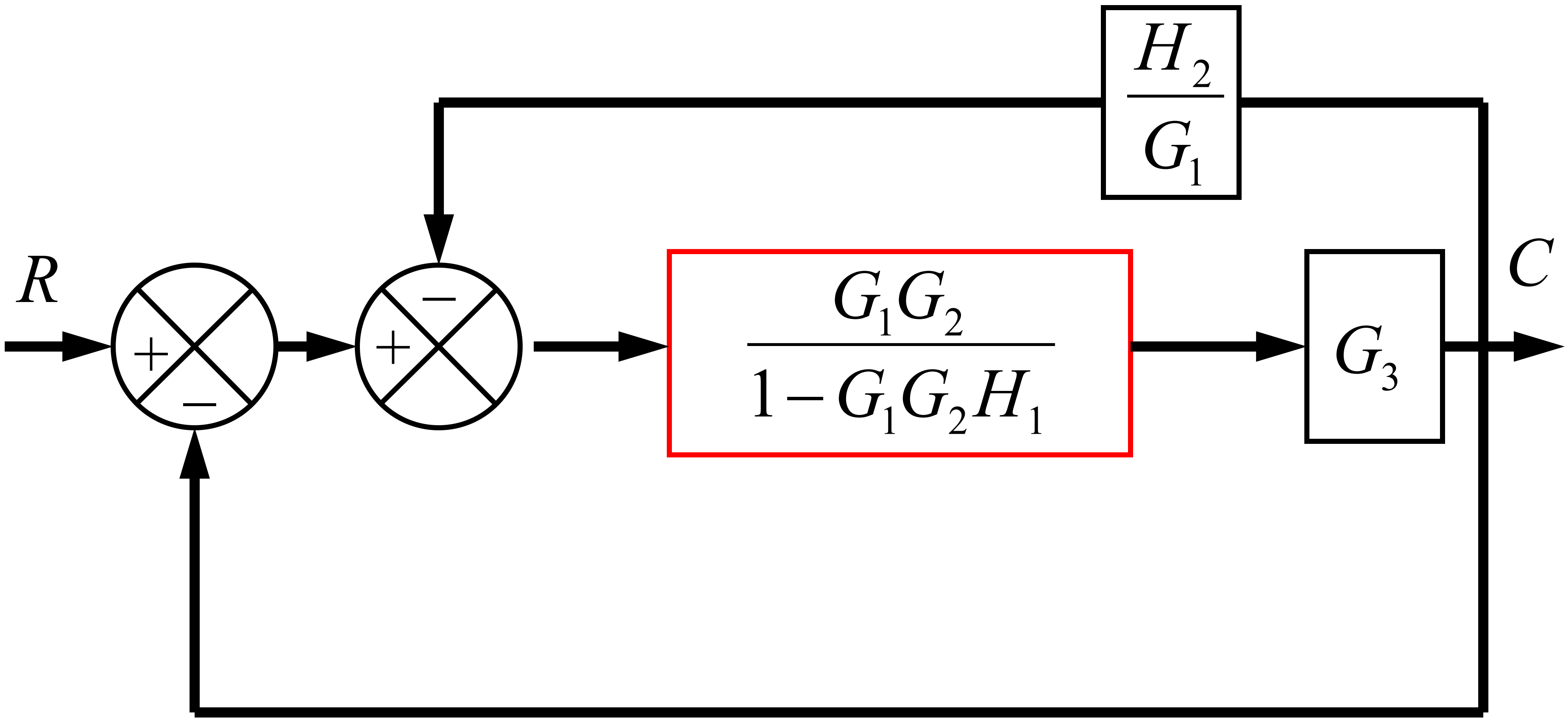
Example-7



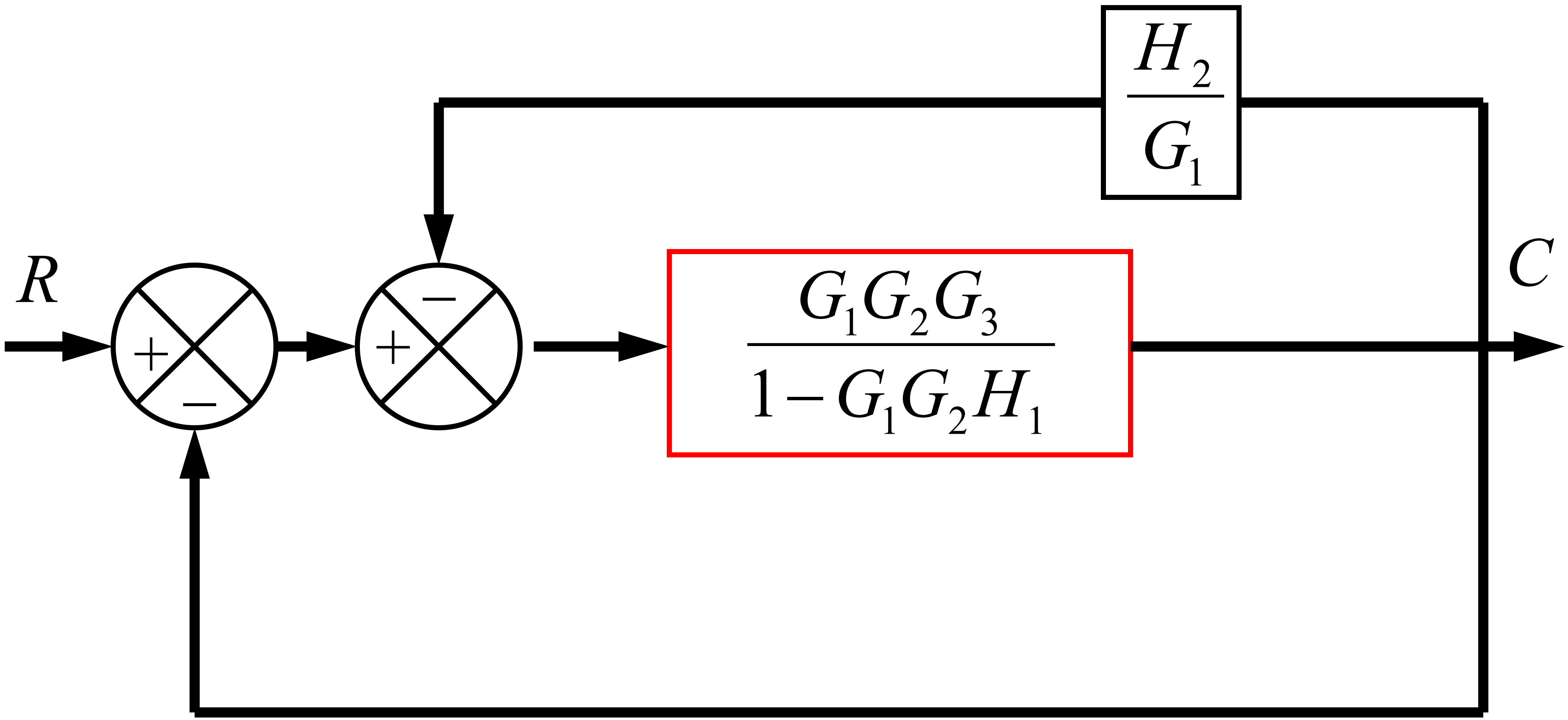
Example-7



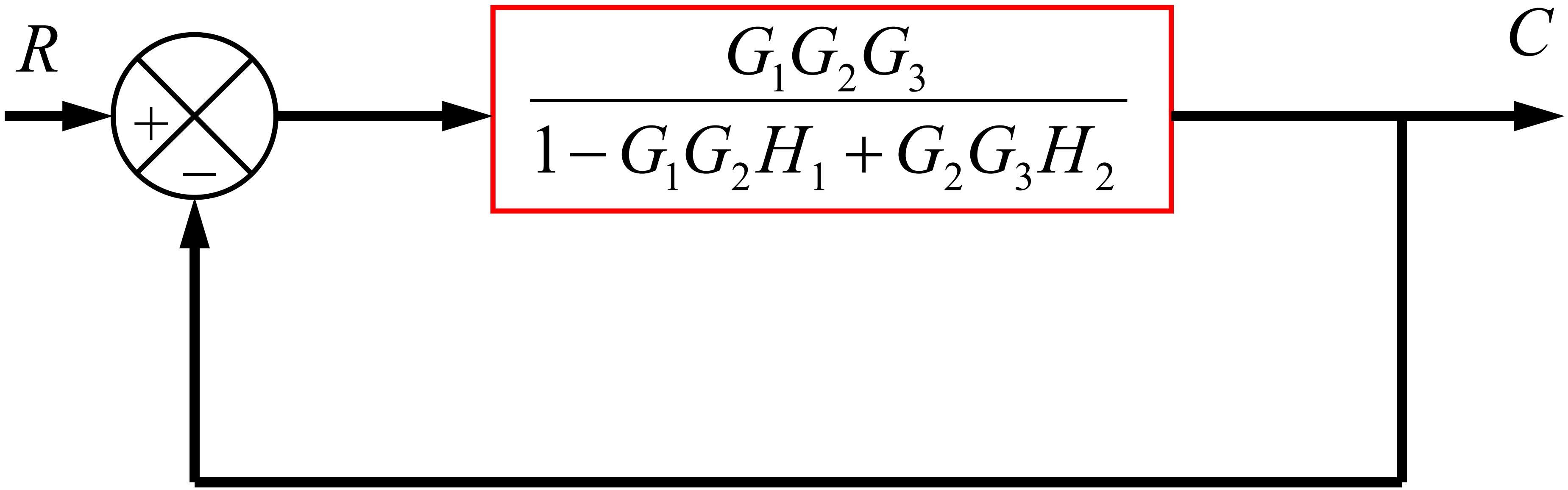
Example-7



Example-7

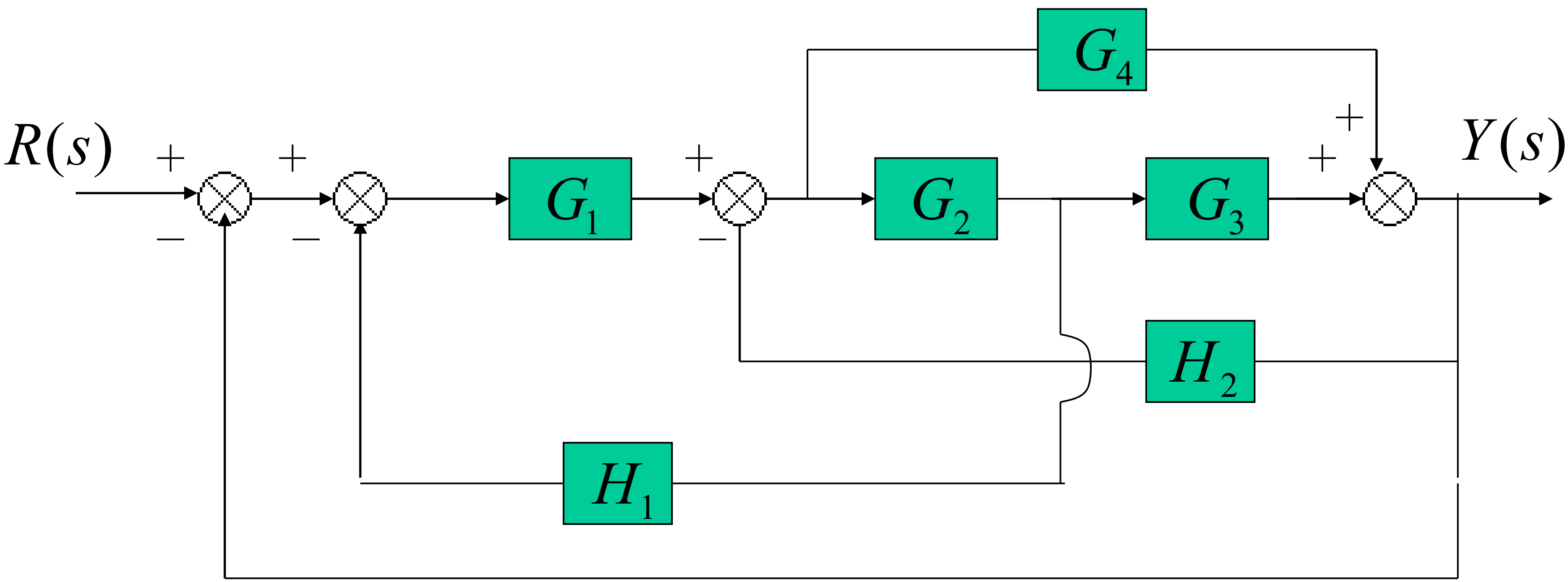


Example-7

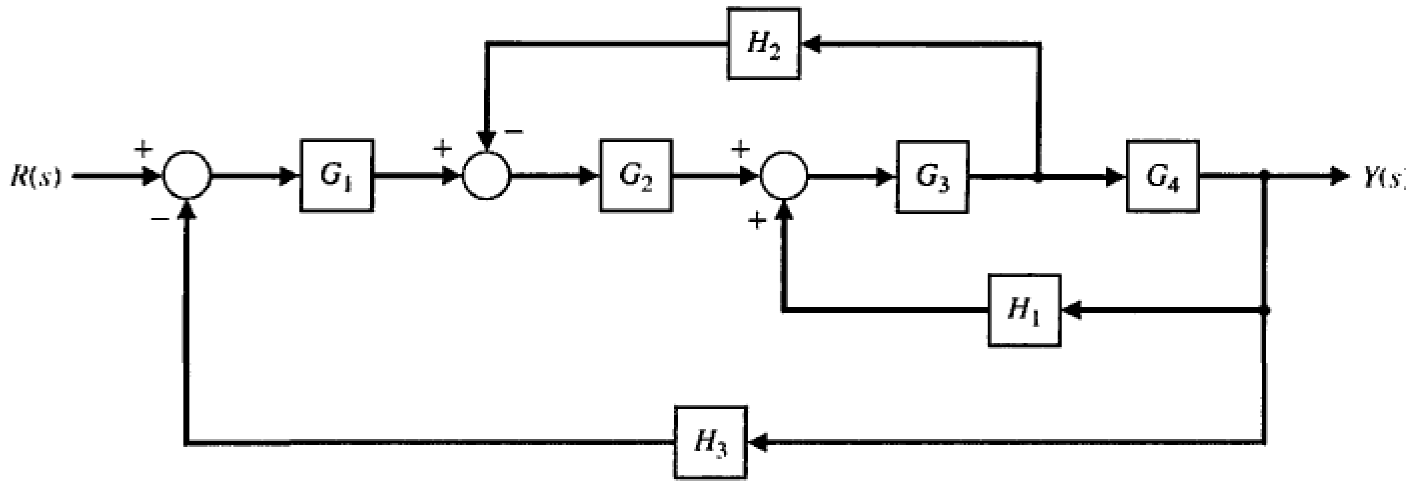


Example 8

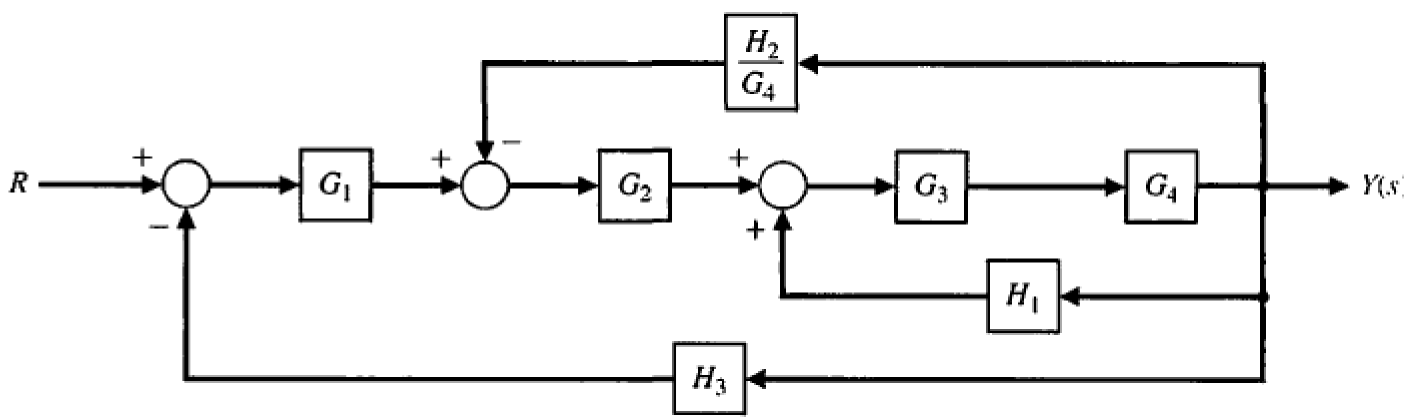
Find the transfer function of the following block diagram



Example-10: Reduce the Block Diagram.

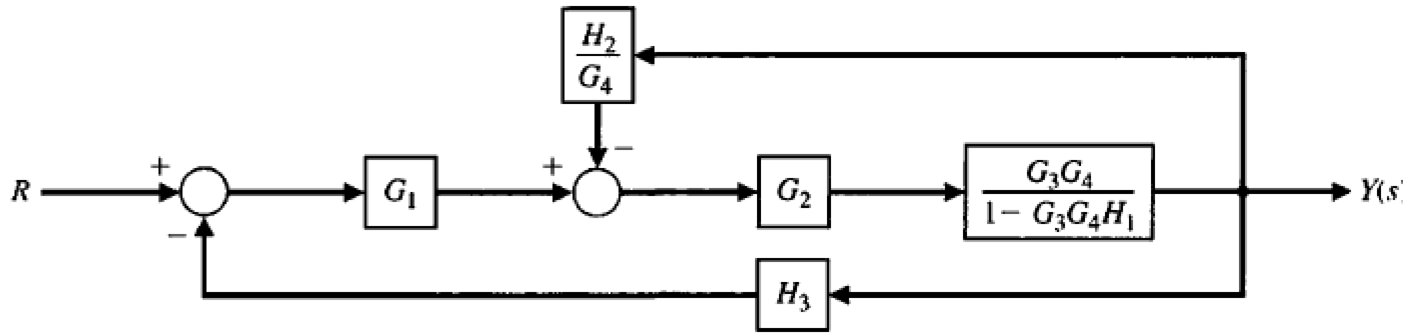


First, to eliminate the loop $G_3G_4H_1$, we move H_2 behind block G_4

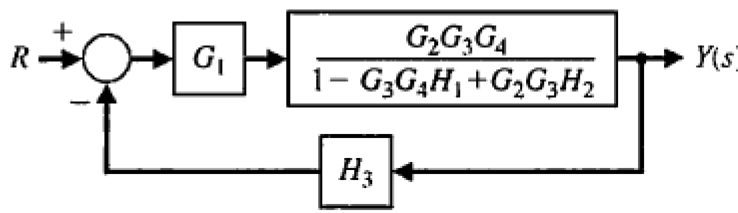


Example-10: Continue.

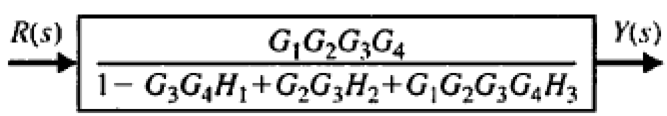
Eliminating the loop $G_3G_4H_1$ we obtain



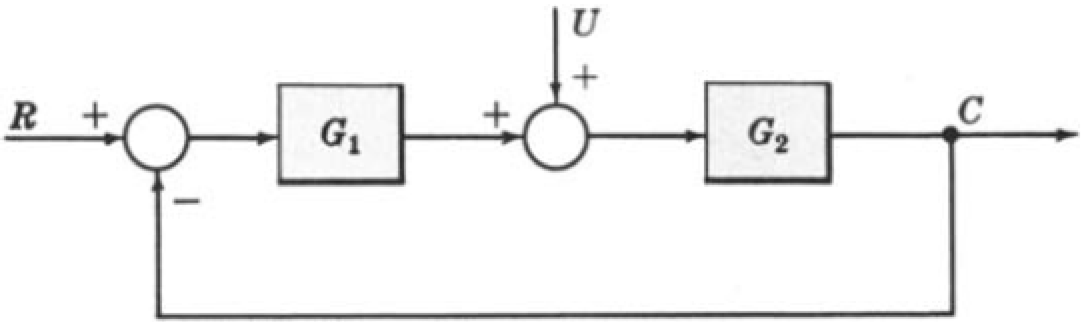
Then, eliminating the inner loop containing H_2/G_4, we obtain



Finally, by reducing the loop containing H_3, we obtain

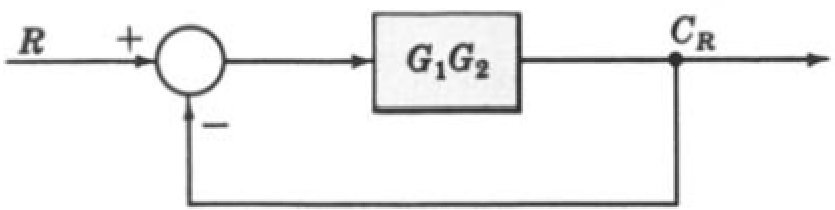


Example-12: Multiple Input System. Determine the output C due to inputs R and U using the Superposition Method.



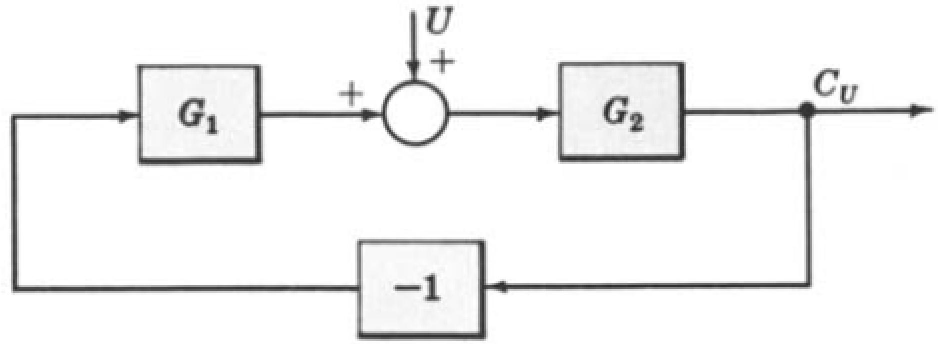
Step 1: Put $U \equiv 0$.

Step 2: The system reduces to



Step 3: the output C_R due to input R is $C_R = [G_1G_2/(1 + G_1G_2)]R$.

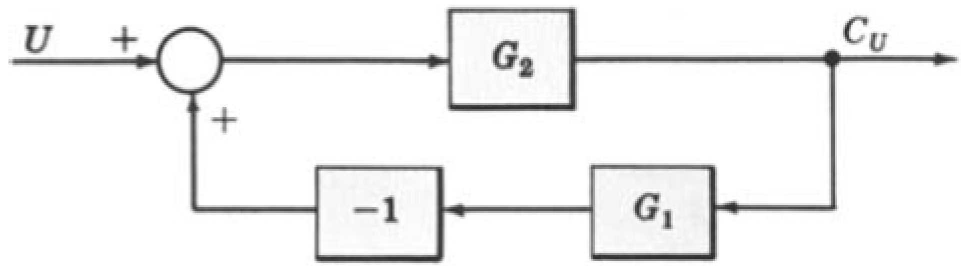
Example-12: Continue.



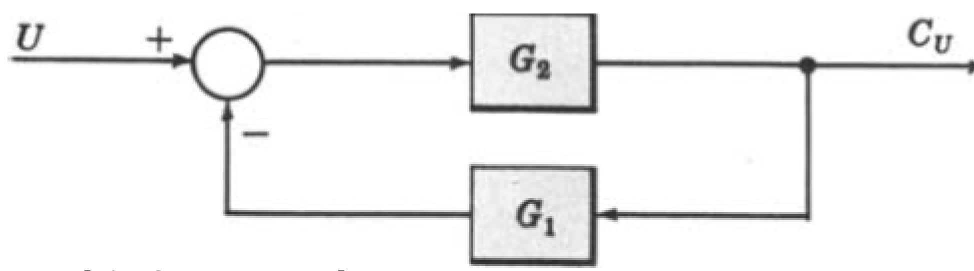
Step 4a: Put $R = 0$.

Step 4b: Put -1 into a block, representing the negative feedback effect:

Rearrange the block diagram:



Let the -1 block be absorbed into the summing point:



Step 4c: the output C_U due to input U is $C_U = [G_2/(1 + G_1G_2)]U$.

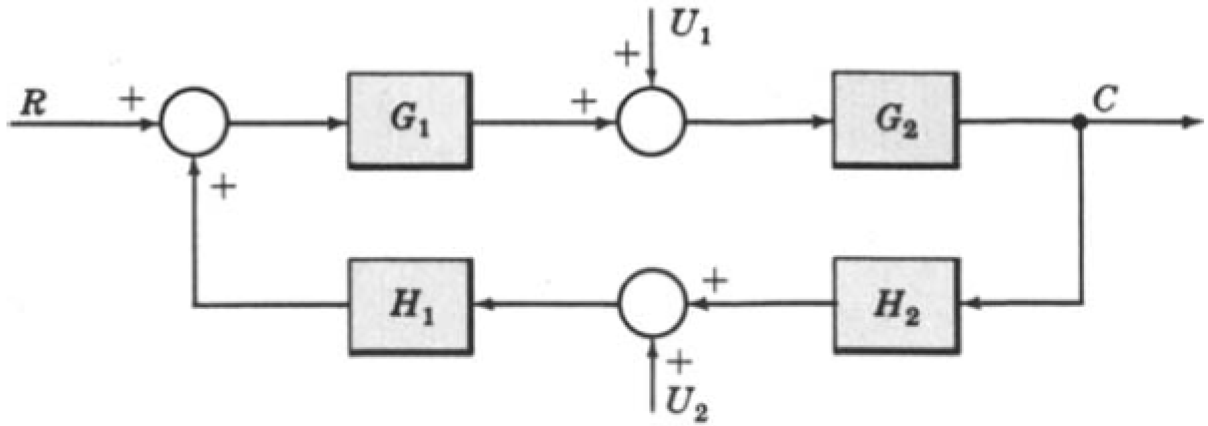
Example-12: Continue.

Step 5: The total output is $C = C_R + C_U$

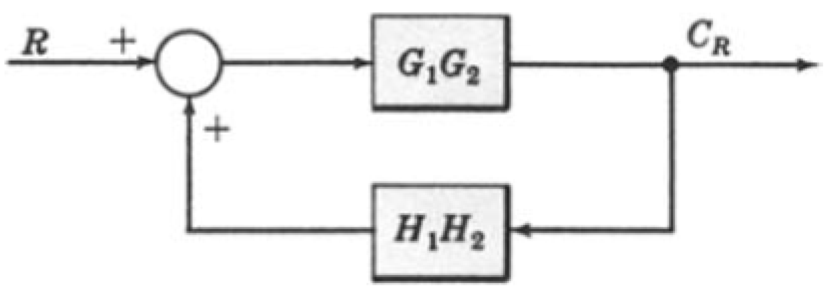
$$= \left[\frac{G_1G_2}{1 + G_1G_2} \right] R + \left[\frac{G_2}{1 + G_1G_2} \right] U$$

$$= \left[\frac{G_2}{1 + G_1G_2} \right] [G_1R + U]$$

Example-13: Multiple-Input System. Determine the output C due to inputs R, U1 and U2 using the Superposition Method.



Let $U_1 = U_2 = 0$.

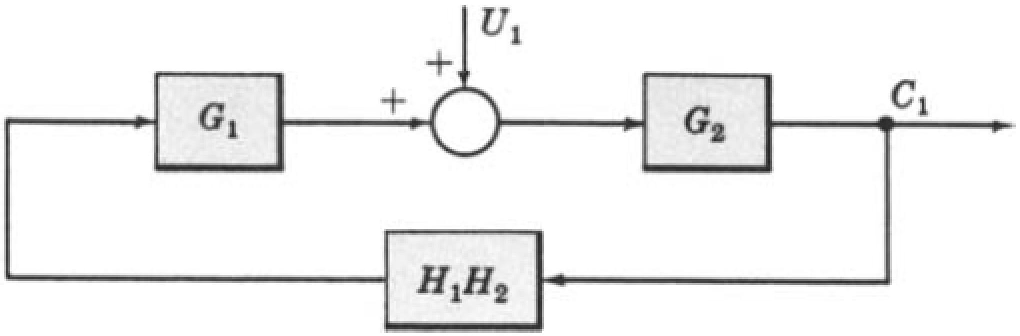


$$C_R = [G_1G_2/(1 - G_1G_2H_1H_2)]R$$

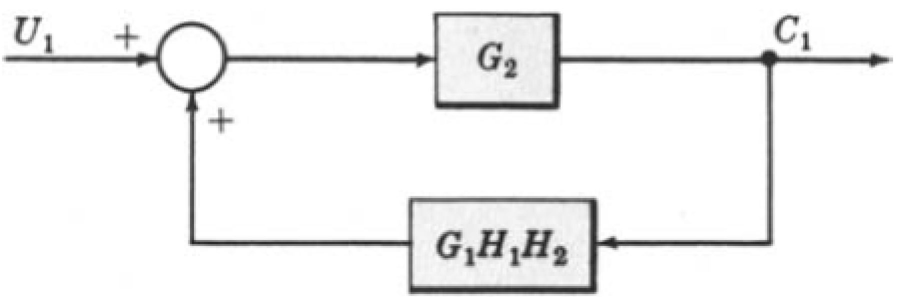
where C_R is the output due to R acting alone.

Example-13: Continue.

Now let $R = U_2 = 0$.



Rearranging the blocks, we get

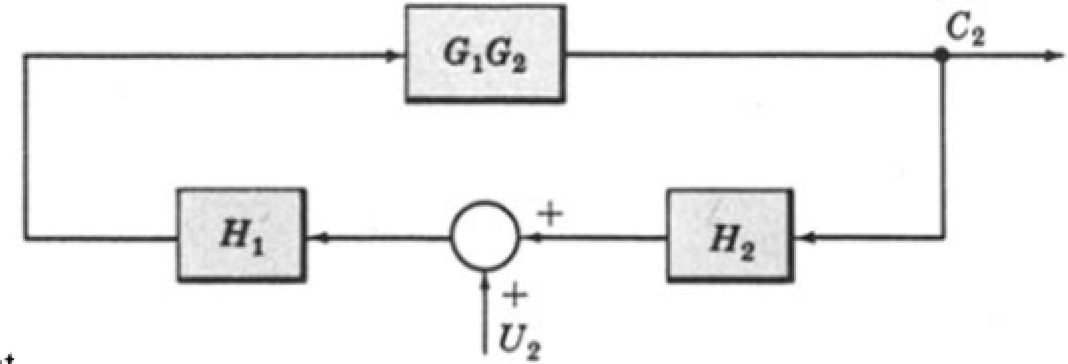


$$C_1 = [G_2/(1 - G_1G_2H_1H_2)]U_1$$

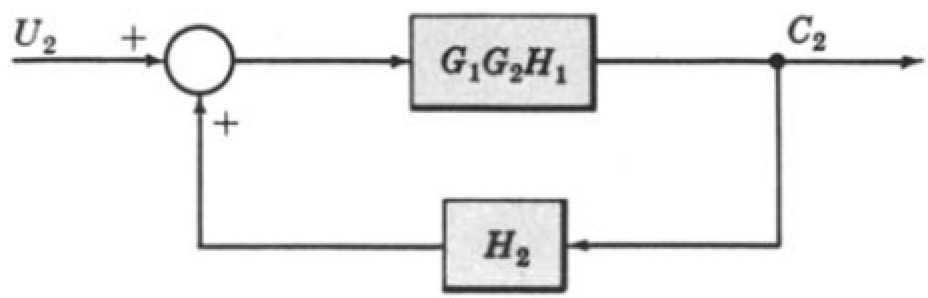
where C_1 is the response due to U_1 acting alone.

Example-13: Continue.

Finally, let $R = U_1 = 0$.



Rearranging the blocks, we get



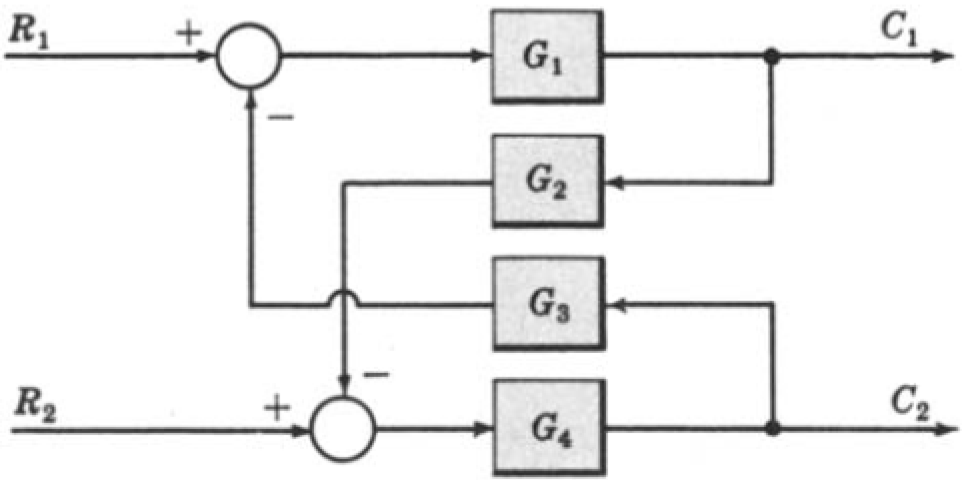
$$C_2 = [G_1G_2H_1/(1 - G_1G_2H_1H_2)]U_2$$

where C_2 is the response due to U_2 acting alone.

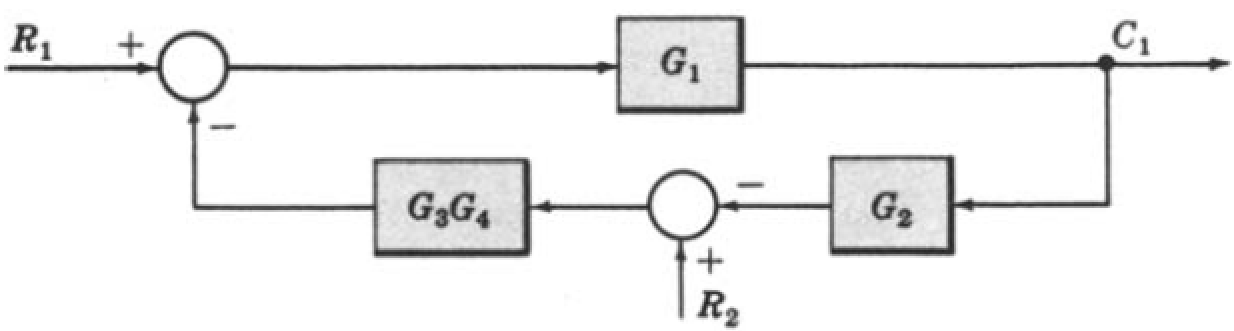
By superposition, the total output is

$$C = C_R + C_1 + C_2 = \frac{G_1G_2R + G_2U_1 + G_1G_2H_1U_2}{1 - G_1G_2H_1H_2}$$

Example-14: Multi-Input Multi-Output System. Determine C1 and C2 due to R1 and R2.

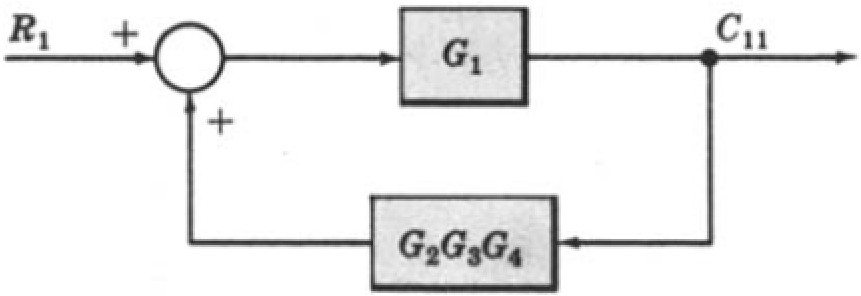


First ignoring the output C_2.



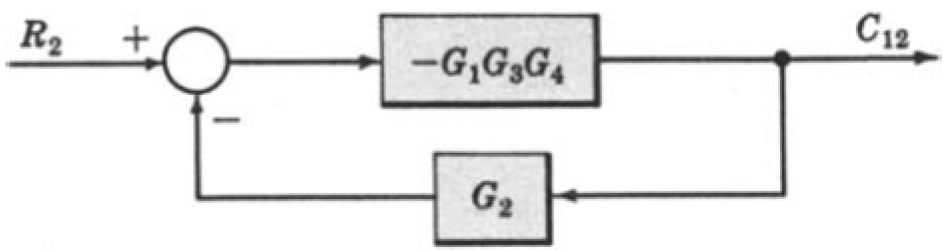
Example-14: Continue.

Letting $R_2 = 0$ and combining the summing points,



Hence C_{11}, the output at C_1 due to R_1 alone, is $C_{11} = G_1R_1/(1 - G_1G_2G_3G_4)$.

For $R_1 = 0$,

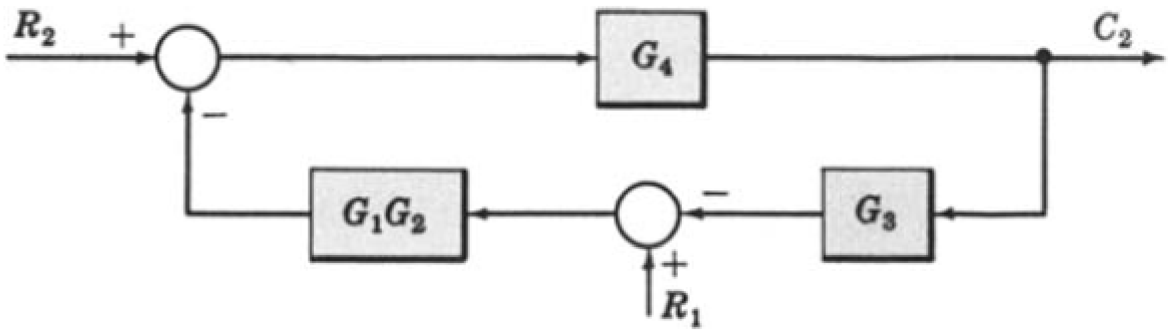


Hence $C_{12} = -G_1G_3G_4R_2/(1 - G_1G_2G_3G_4)$ is the output at C_1 due to R_2 alone.

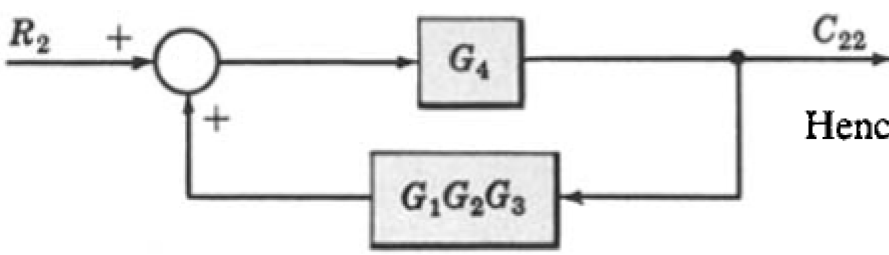
Thus $C_1 = C_{11} + C_{12} = (G_1R_1 - G_1G_3G_4R_2)/(1 - G_1G_2G_3G_4)$

Example-14: Continue.

Now we reduce the original block diagram, ignoring output C_1.

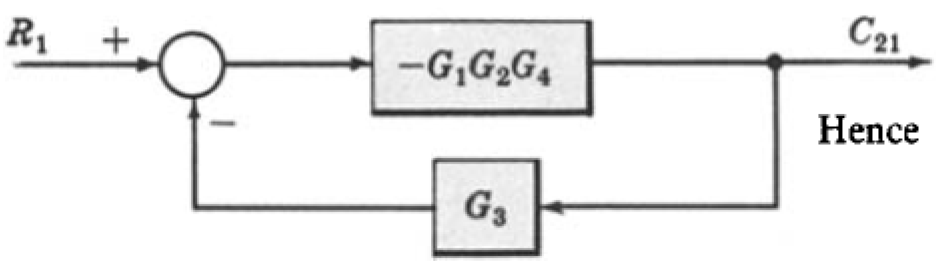


When R1 = 0,



Hence $C_{22} = G_4 R_2/(1 - G_1G_2G_3G_4)$

When R2 = 0,



Hence $C_{21} = -G_1G_2G_4 R_1/(1 - G_1G_2G_3G_4)$

Finally, $C_2 = C_{22} + C_{21} = (G_4 R_2 - G_1G_2G_4 R_1)/(1 - G_1G_2G_3G_4)$

Introduction

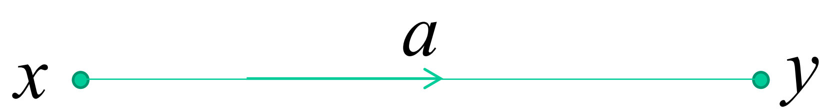
- Alternative method to block diagram representation, developed by Samuel Jefferson Mason.
- Advantage: the availability of a flow graph gain formula, also called Mason's gain formula.
- A signal-flow graph consists of a network in which nodes are connected by directed branches.
- It depicts the flow of signals from one point of a system to another and gives the relationships among the signals.

Fundamentals of Signal Flow Graphs

- Consider a simple equation below and draw its signal flow graph:

$$y = ax$$

- The signal flow graph of the equation is shown below;

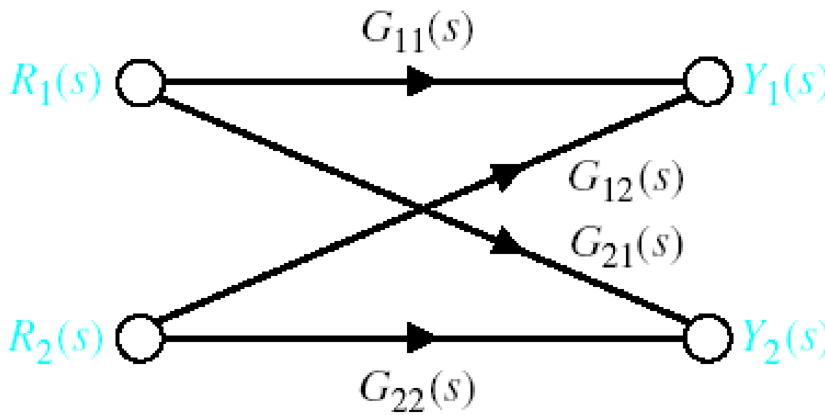


- Every variable in a signal flow graph is designed by a **Node**.
- Every transmission function in a signal flow graph is designed by a **Branch**.
- Branches are always **unidirectional**.
- The arrow in the branch denotes the **direction** of the signal flow.

Signal-Flow Graph Models

$$Y_1(s) = G_{11}(s)\cdot R_1(s) + G_{12}(s)\cdot R_2(s)$$

$$Y_2(s) = G_{21}(s)\cdot R_1(s) + G_{22}(s)\cdot R_2(s)$$

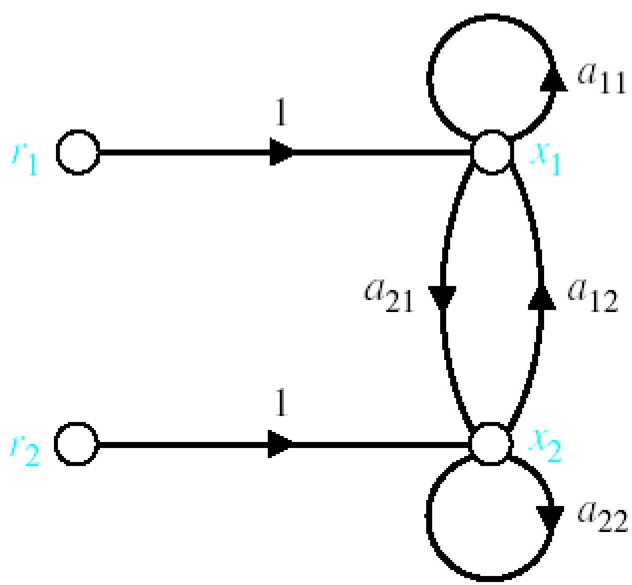


Signal-Flow Graph Models

r_1 and r_2 are inputs and x_1 and x_2 are outputs

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + r_1 = x_1$$

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + r_2 = x_2$$



Signal-Flow Graph Models

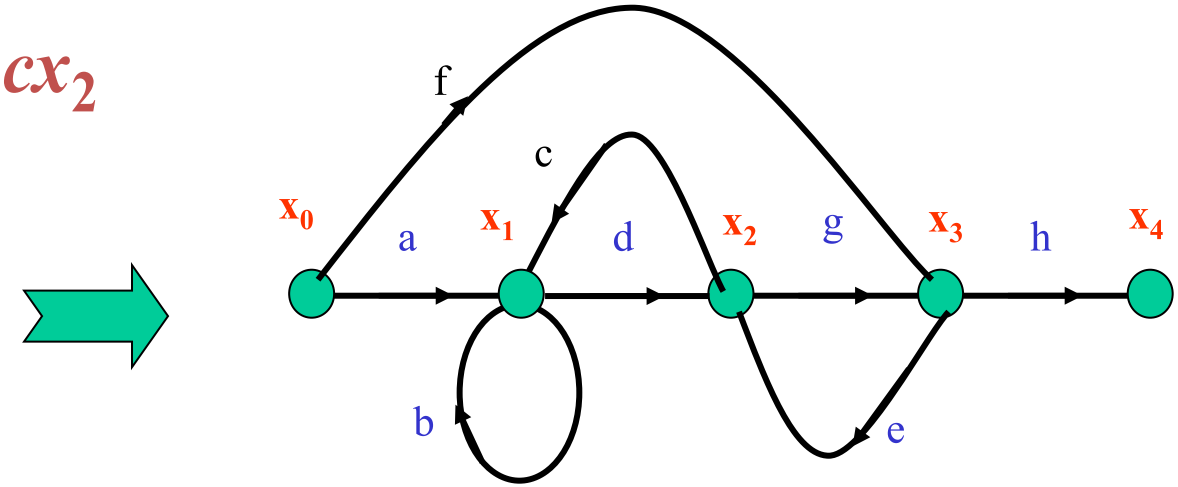
x_o is input and x_4 is output

$$x_1 = ax_0 + bx_1 + cx_2$$

$$x_2 = dx_1 + ex_3$$

$$x_3 = fx_0 + gx_2$$

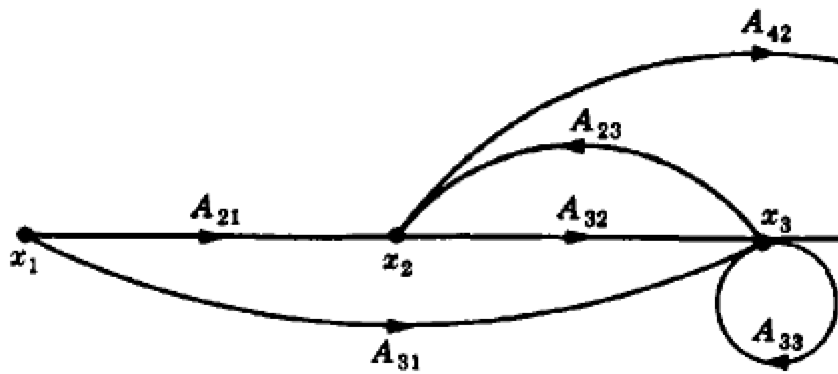
$$x_4 = hx_3$$



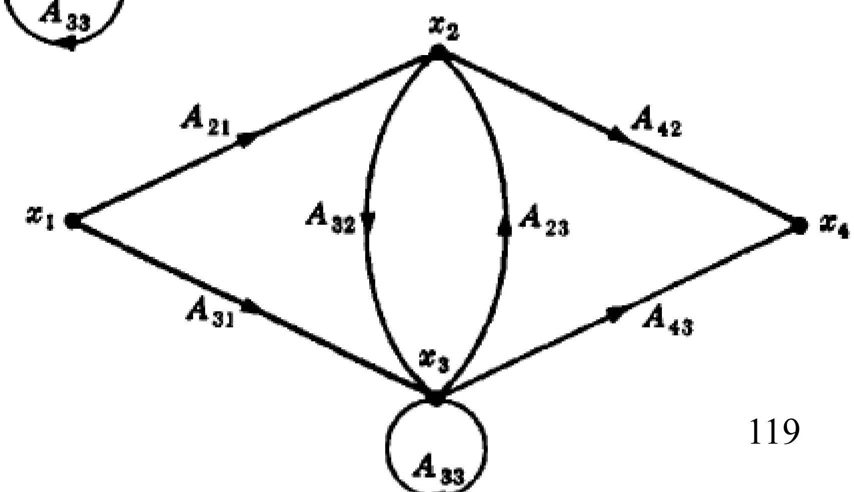
Construct the signal flow graph for the following set of simultaneous equations.

$$x_2 = A_{21}x_1 + A_{23}x_3 \qquad x_3 = A_{31}x_1 + A_{32}x_2 + A_{33}x_3 \qquad x_4 = A_{42}x_2 + A_{43}x_3$$

- There are four variables in the equations (i.e., x_1,x_2,x_3,and x_4) therefore four nodes are required to construct the signal flow graph.
- Arrange these four nodes from left to right and connect them with the associated branches.



- Another way to arrange this graph is shown in the figure.



Terminologies

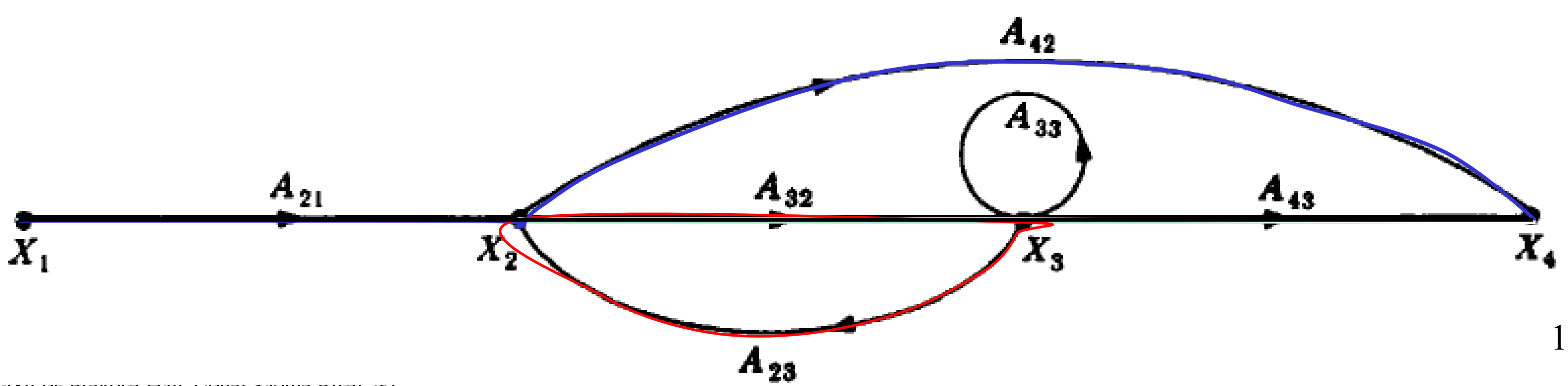
- An **input node** or source contain only the outgoing branches. i.e., X_1
- An **output node** or sink contain only the incoming branches. i.e., X_4
- A **path** is a continuous, unidirectional succession of branches along which no node is passed more than ones. i.e.,

  X_1 *to* X_2 *to* X_3 *to* X_4 X_1 *to* X_2 *to* X_4 X_2 *to* X_3 *to* X_4

- A **forward path** is a path from the input node to the output node. i.e.,

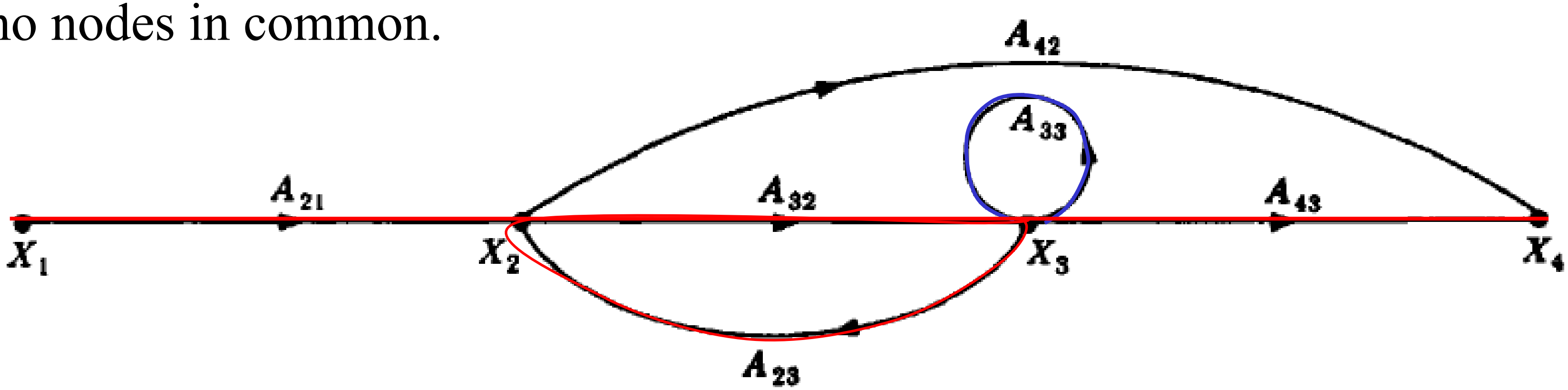
  X_1 *to* X_2 *to* X_3 *to* X_4, and X_1 *to* X_2 *to* X_4, are forward paths.

- A **feedback path** or feedback loop is a path which originates and terminates on the same node. i.e.; X_2 *to* X_3 **and back to** X_2 is a feedback path.



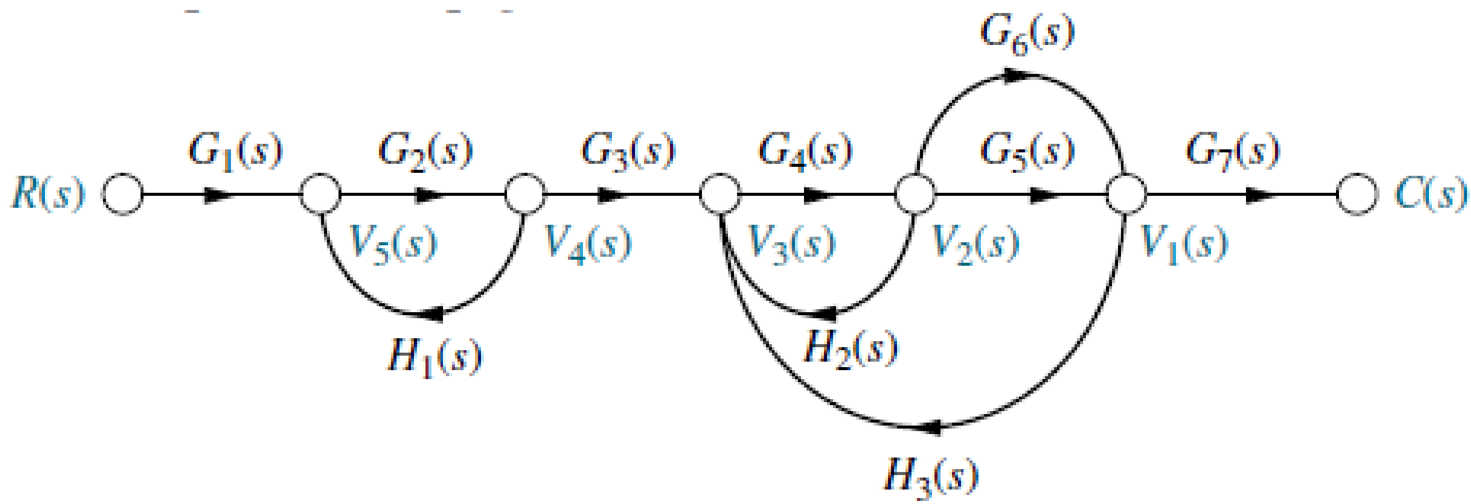
Terminologies

- A **self-loop** is a feedback loop consisting of a single branch. i.e.; A_{33} is a self loop.
- The **gain** of a branch is the transmission function of that branch.
- The **path gain** is the product of branch gains encountered in traversing a path. i.e. the gain of forwards path X_1 *to* X_2 *to* X_3 *to* X_4 is $A_{21}A_{32}A_{43}$
- The **loop gain** is the product of the branch gains of the loop. i.e., the loop gain of the feedback loop from X_2 to X_3 and back to X_2 is $A_{32}A_{23}$.
- Two loops, paths, or loop and a path are said to be **non-touching** if they have no nodes in common.



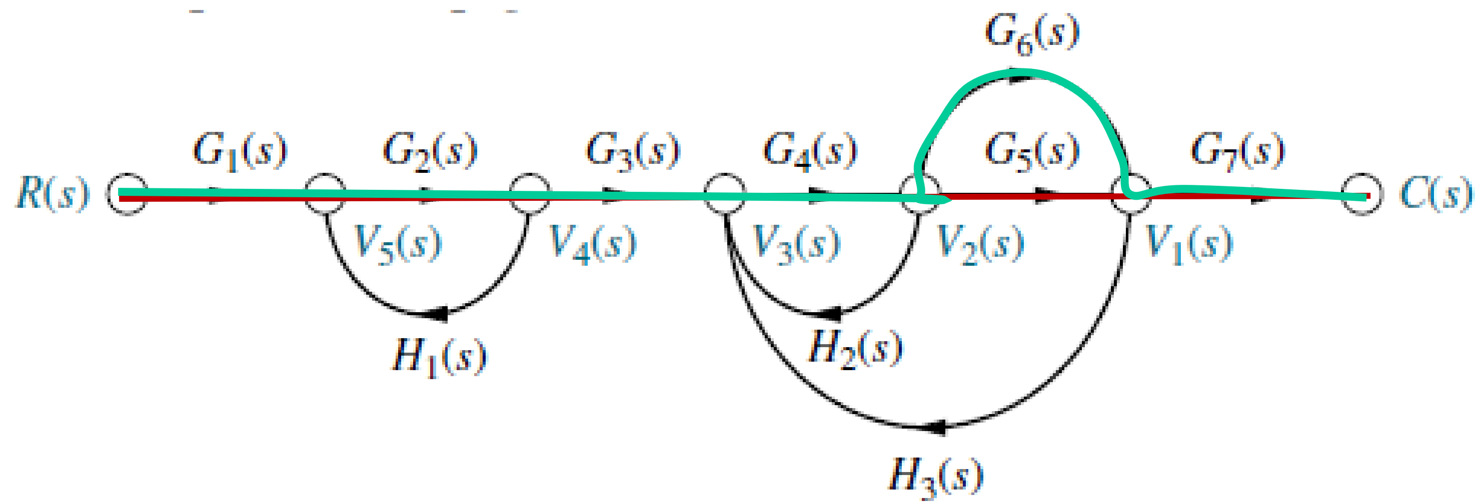
Illustrat

Consider the signal flow graph below and identify the following



a) Input node.
b) Output node.
c) Forward paths.
d) Feedback paths (loops).
e) Determine the loop gains of the feedback loops.
f) Determine the path gains of the forward paths.
g) Non-touching loops

Consider the signal flow graph below and identify the following

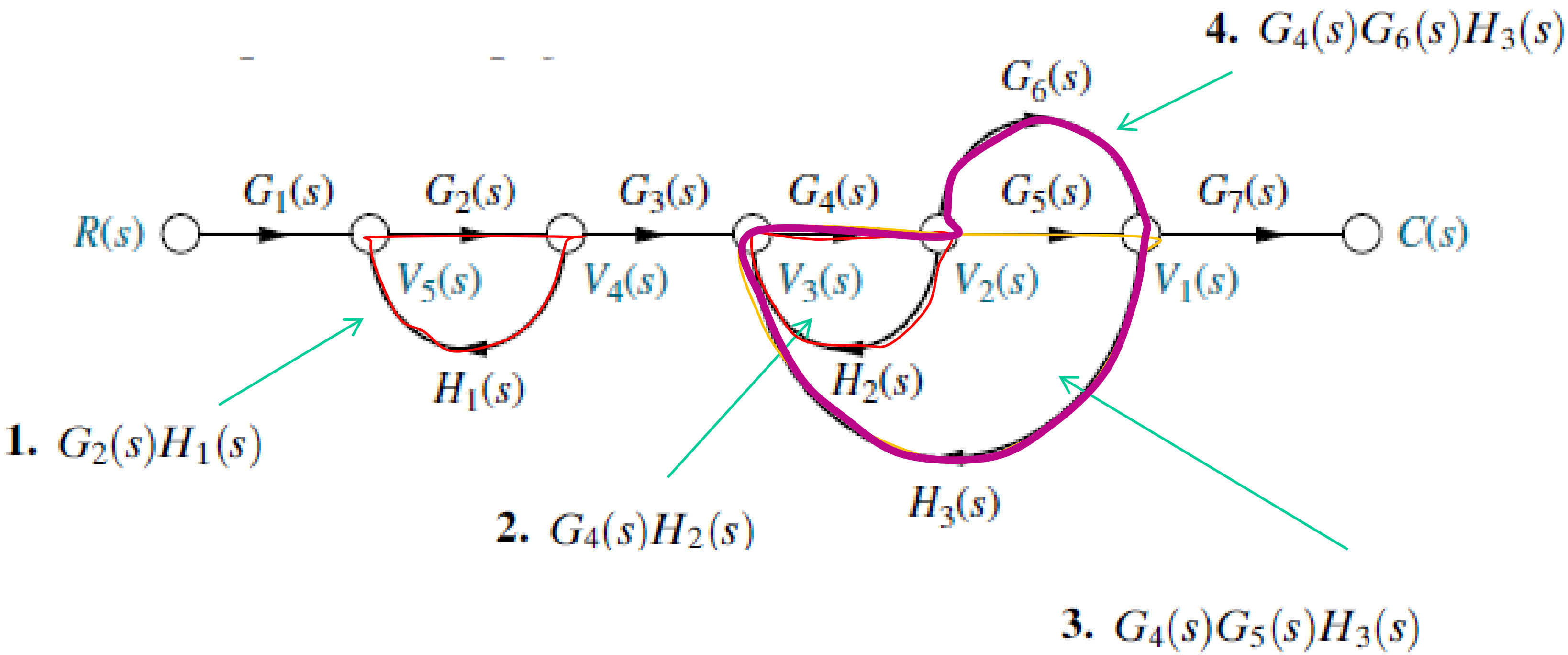


- There are two forward path gains;

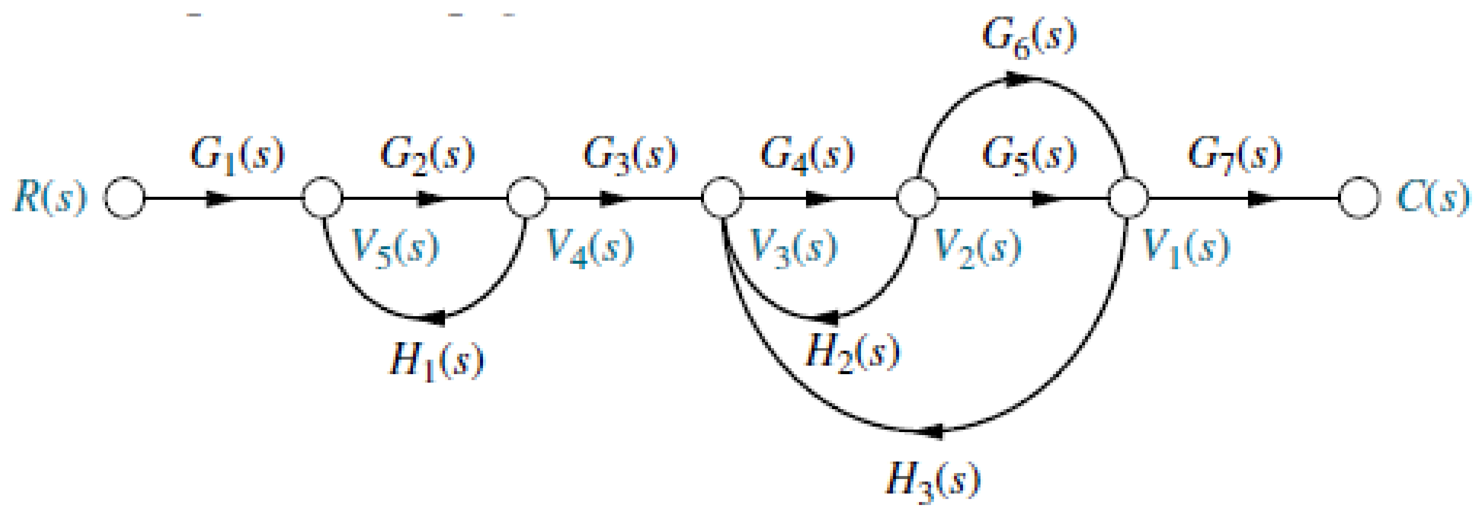
1. $G_1(s)G_2(s)G_3(s)G_4(s)G_5(s)G_7(s)$ **2.** $G_1(s)G_2(s)G_3(s)G_4(s)G_6(s)G_7(s)$

Consider the signal flow graph below and identify the following

- There are four loops



Consider the signal flow graph below and identify the following



- Nontouching loop gains;

1. $[G_2(s)H_1(s)][G_4(s)H_2(s)]$
2. $[G_2(s)H_1(s)][G_4(s)G_5(s)H_3(s)]$
3. $[G_2(s)H_1(s)][G_4(s)G_6(s)H_3(s)]$

Mason’s Rule (Mason, 1953)

- The block diagram reduction technique requires successive application of fundamental relationships in order to arrive at the system transfer function.
- On the other hand, Mason’s rule for reducing a signal-flow graph to a single transfer function requires the application of one formula.
- The formula was derived by S. J. Mason when he related the signal-flow graph to the simultaneous equations that can be written from the graph.

Mason's Rule:

- The transfer function, ***C(s)/R(s),*** of a system represented by a signal-flow graph is;

$$\frac{C(s)}{R(s)} = \frac{\sum_{i=1}^{n} P_i \Delta_i}{\Delta}$$

Where

n = number of forward paths.
P_i = the i th forward-path gain.
Δ = Determinant of the system
Δ_i = Determinant of the i^{th} forward path

- Δ is called the signal flow graph determinant or characteristic function. Since $\Delta=0$ is the system characteristic equation.

Mason's Rule:

$$\frac{C(s)}{R(s)} = \frac{\sum_{i=1}^{n} P_i \Delta_i}{\Delta}$$

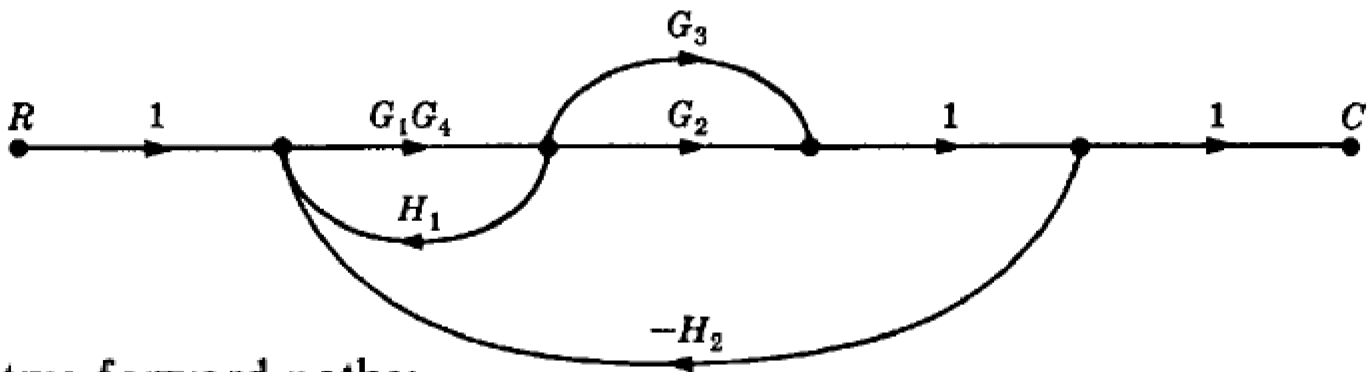
Δ = 1- (sum of all individual loop gains) + (sum of the products of the gains of all possible two loops that do not touch each other) – (sum of the products of the gains of all possible three loops that do not touch each other) + … and so forth with sums of higher number of non-touching loop gains

Δ_i = value of Δ for the part of the block diagram that does not touch the i-th forward path ($\Delta_i = 1$ if there are no non-touching loops to the i-th path.)

Systematic approach

1. Calculate forward path gain P_i for each forward path i.
2. Calculate all loop transfer functions
3. Consider non-touching loops 2 at a time
4. Consider non-touching loops 3 at a time
5. etc
6. Calculate Δ from steps 2,3,4 and 5
7. Calculate Δ_i as portion of Δ not touching forward path i

Example#1: Apply Mason's Rule to calculate the transfer function of the system represented by following Signal Flow Graph



There are two forward paths:

$P_1 = G_1G_2G_4 \qquad P_2 = G_1G_3G_4$

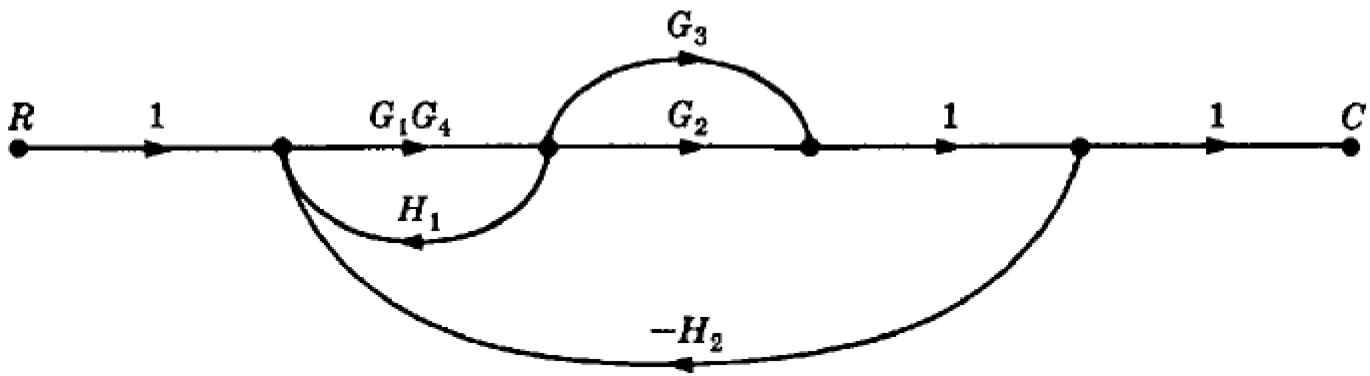
Therefore,

$$\frac{C}{R} = \frac{P_1\Delta_1 + P_2\Delta_2}{\Delta}$$

There are three feedback loops

$$L_1 = G_1G_4H_1, \quad L_2 = -G_1G_2G_4H_2, \quad L_3 = -G_1G_3G_4H_2$$

Example#1: Apply Mason's Rule to calculate the transfer function of the system represented by following Signal Flow Graph



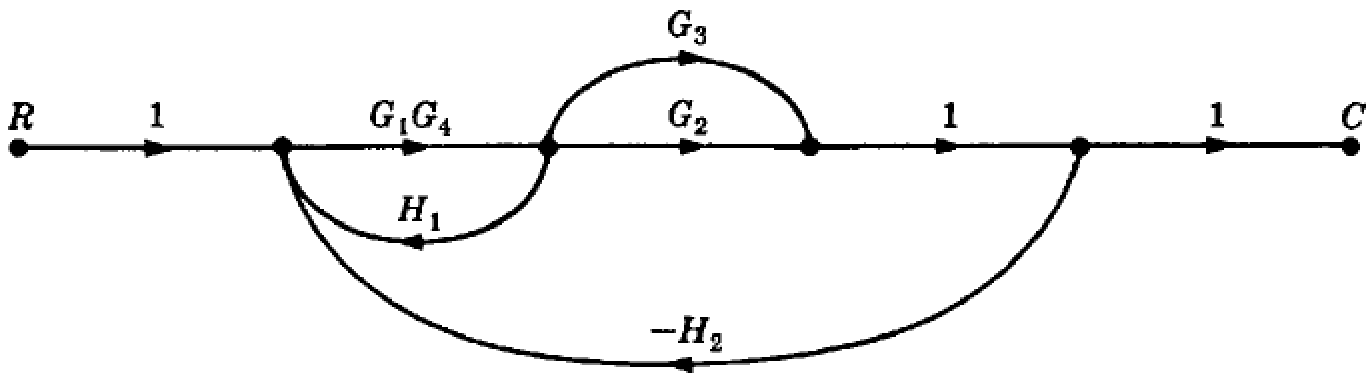
There are no non-touching loops, therefore

Δ = 1- (sum of all individual loop gains)

$$\Delta = 1 - \left(L_1 + L_2 + L_3\right)$$

$$\Delta = 1 - \left(G_1G_4H_1 - G_1G_2G_4H_2 - G_1G_3G_4H_2\right)$$

Example#1: Apply Mason's Rule to calculate the transfer function of the system represented by following Signal Flow Graph



Eliminate forward path-1

$$\Delta_1 = 1\text{- (sum of all individual loop gains)+...}$$
$$\Delta_1 = 1$$

Eliminate forward path-2

$$\Delta_2 = 1\text{- (sum of all individual loop gains)+...}$$
$$\Delta_2 = 1$$

Example#1: Continue

$$\frac{C}{R} = \frac{P_1\Delta_1 + P_2\Delta_2}{\Delta} = \frac{G_1G_2G_4 + G_1G_3G_4}{1 - G_1G_4H_1 + G_1G_2G_4H_2 + G_1G_3G_4H_2}$$

$$= \frac{G_1G_4(G_2 + G_3)}{1 - G_1G_4H_1 + G_1G_2G_4H_2 + G_1G_3G_4H_2}$$

CHAPTER 4

Transient & Steady State Response Analysis

Introduction

The time response of a control system consists of two parts:

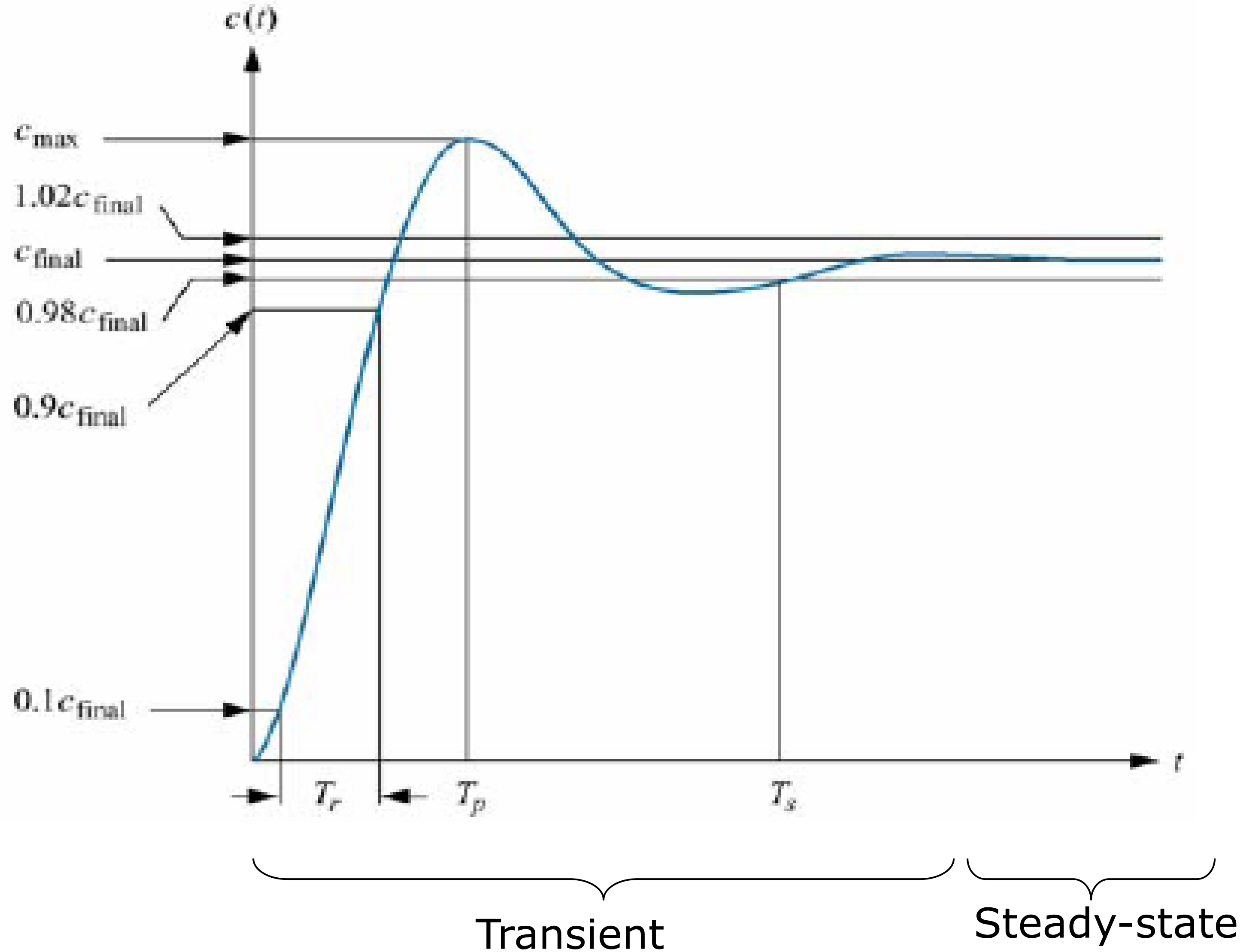
1. Transient response

   - from initial state to the final state – purpose of control systems is to provide a desired response.

2. Steady-state response

   - the manner in which the system output behaves as *t* approaches infinity – the error after the transient response has decayed, leaving only the continuous response.

Introduction



First – order system

A first-order system without zeros can be represented by the following transfer function

$$\frac{C(s)}{R(s)} = \frac{1}{\tau s + 1}$$

- Given a step input, i.e., R(s) = 1/s , then the system output (called **step response** in this case) is

$$C(s) = \frac{1}{\tau s + 1} R(s) = \frac{1}{s(\tau s + 1)} = \frac{1}{s} - \frac{1}{s + \frac{1}{\tau}}$$

First – order system

Taking inverse Laplace transform, we have the step response

$$\boxed{c(t) = 1 - e^{-\frac{t}{\tau}}}$$

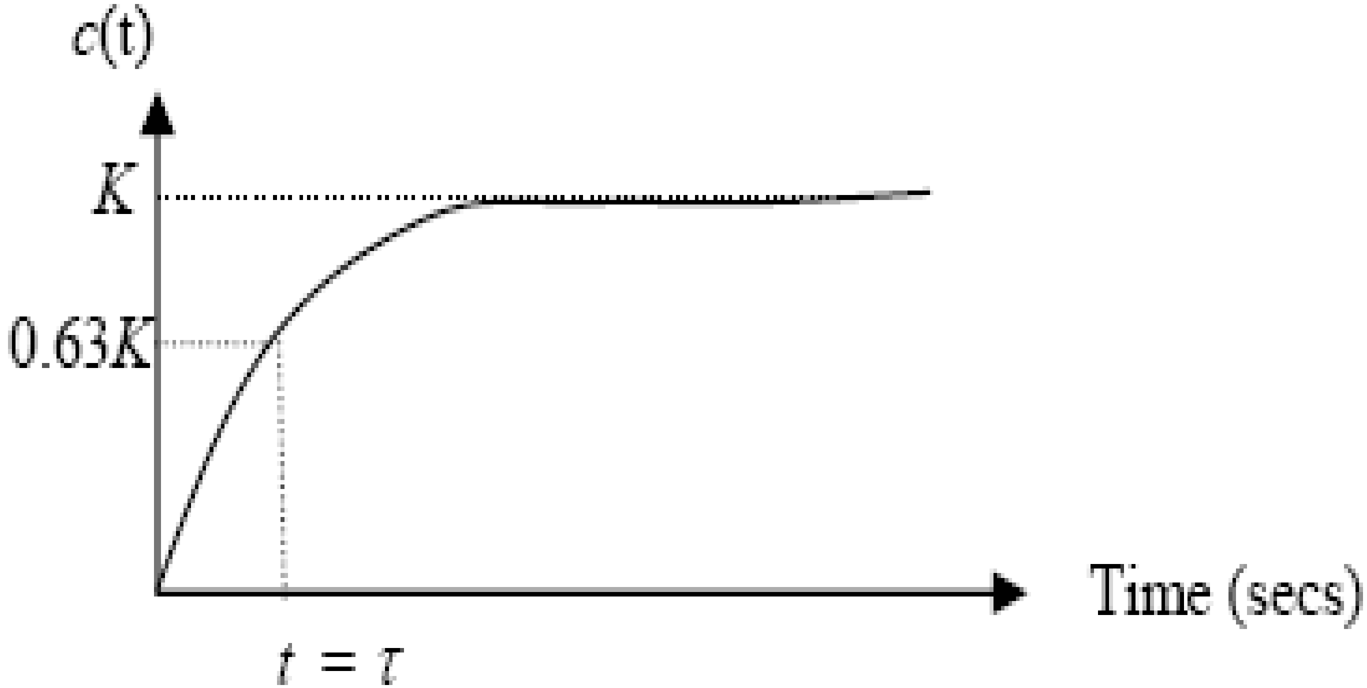
Time Constant: If t= τ , So the step response is

$C(\tau)$ = (1− 0.37) = 0.63

τ is referred to as the **time constant** of the response. In other words, the time constant is the time it takes for the step response to rise to 63% of its final value. Because of this, the time constant is used to measure how fast a system can respond. The time constant has a unit of seconds.

First – order system

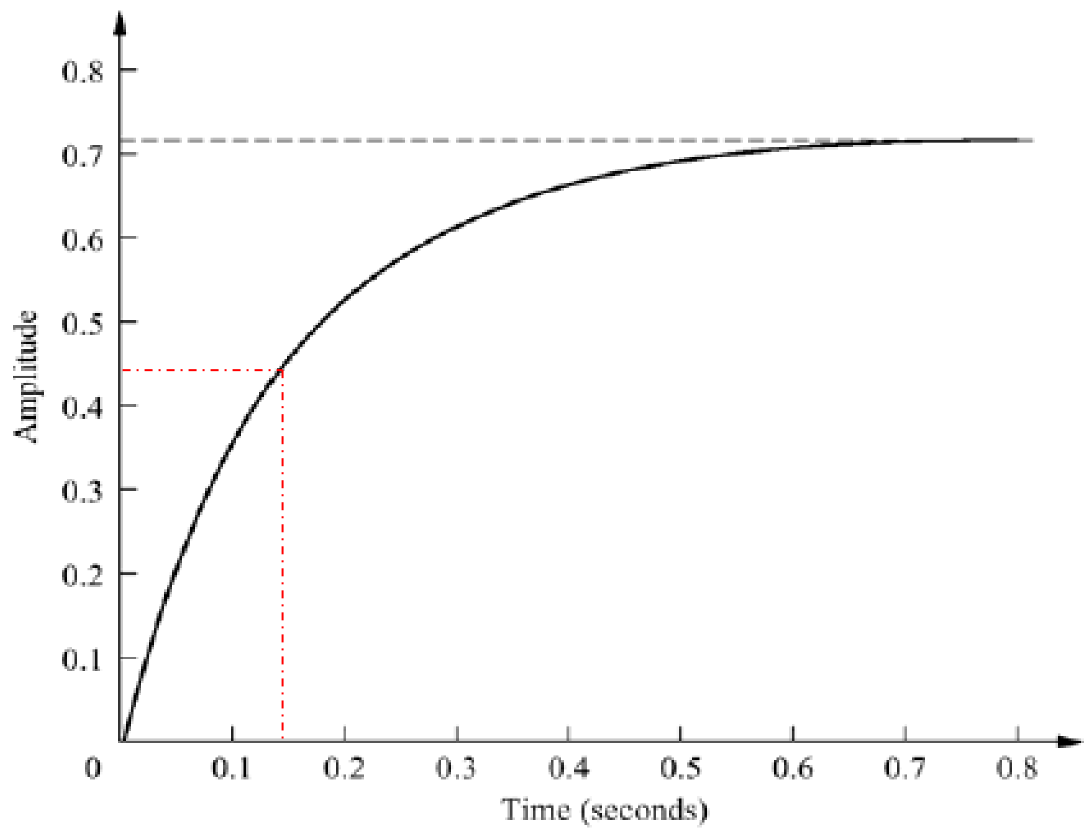
Plot $c(t)$ versus time:



First – order system

Example 1

The following figure gives the measurements of the step response of a first-order system, find the transfer function of the system.



First – order system
Transient Response Analysis

Rise Time *T_r*:

The rise-time (symbol T_r units s) is defined as the time taken for the step response to go from **10% to 90%** of the final value.

$$\boxed{T_r = 2.31\tau - 0.11\tau = 2.2\tau}$$

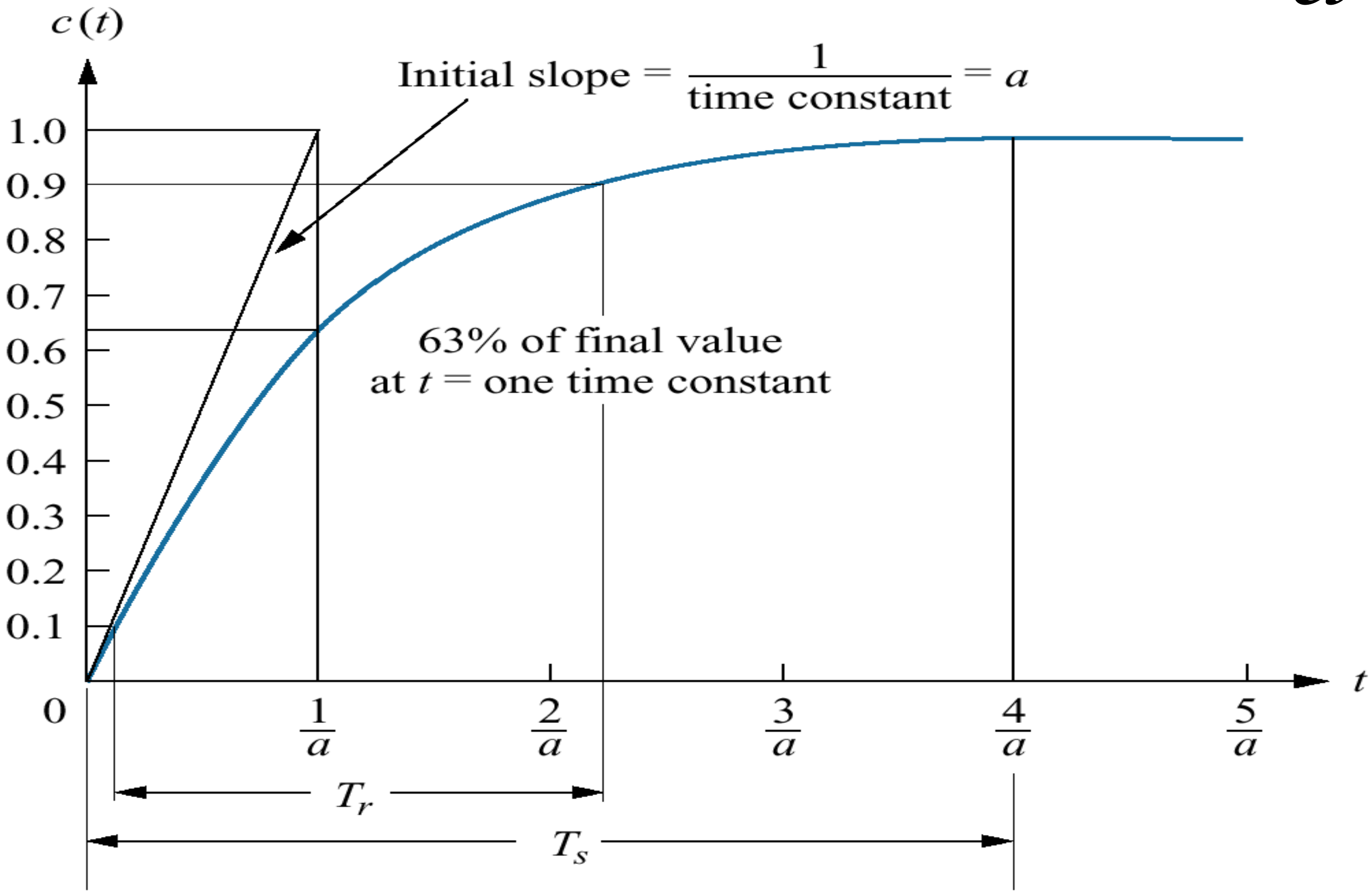
Settling Time *T_s*:

Defined the settling-time (symbol T_s units s) to be the time taken for the step response to come to within **2% of the final value** of the step response.

$$\boxed{T_s = 4\tau}$$

First – order system

$$\tau = \frac{1}{a}$$



Second – Order System

- *Second-order systems* exhibit a wide range of responses which must be analyzed and described.
  - Whereas for a *first-order system*, varying a single parameter changes the speed of response, changes in the parameters of a *second order system* can change the form of the response.

- *For example*: a second-order system can display characteristics much like a first-order system or, depending on component values, display damped or *pure oscillations* for its *transient response*.

Second – Order System

- A general second-order system is characterized by the following transfer function:

$$G(s) = \frac{b}{s^2 + as + b}$$

- We can re-write the above transfer function in the following form (closed loop transfer function):

$$\boxed{G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}}$$

Second – Order System

$\omega_n \quad (\omega_n = \sqrt{b})$ - referred to as *the* *un-damped natural frequency* of the second order system, which is the frequency of oscillation of the system without damping.

$\zeta \; (\zeta = \dfrac{a}{2\sqrt{b}})$ - referred to as *the* *damping ratio* of the second order system, which is a measure of the degree of resistance to change in the system output.

Poles;

$$-\omega_n\zeta + \omega_n\sqrt{\zeta^2 - 1}$$

$$-\omega_n\zeta - \omega_n\sqrt{\zeta^2 - 1}$$

Poles are complex if ζ< 1!

Second – Order System

- According the value of ζ, a second-order system can be set into one of the four categories:

1. *Overdamped* - when the system has two real distinct poles ($\zeta > 1$).
2. *Underdamped* - when the system has two complex conjugate poles ($0 < \zeta < 1$)
3. *Undamped* - when the system has two imaginary poles ($\zeta = 0$).
4. *Critically damped* - when the system has two real but equal poles ($\zeta = 1$).

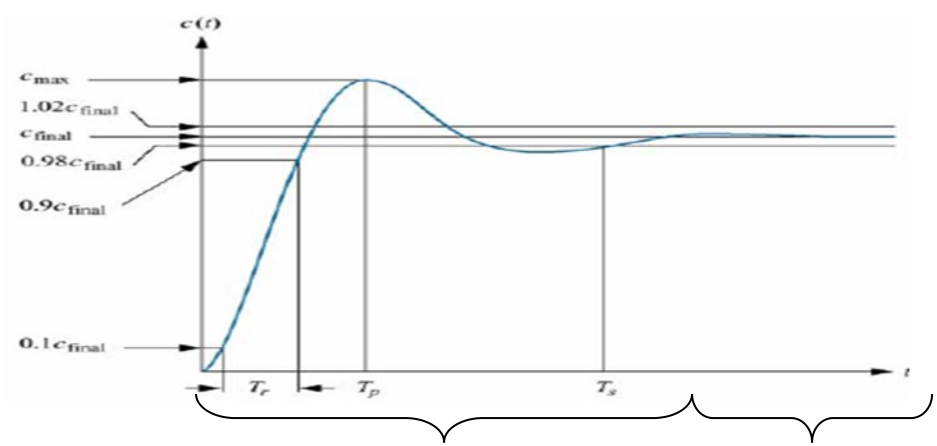
Time-Domain Specification

Given that the closed loop TF

$$T(s)=\frac{C(s)}{R(s)}=\frac{\omega_n^{\;2}}{s^2+2\varsigma\omega_n s+\omega_n^{\;2}}$$

The system (2nd order system) is parameterized by ς and ω_n

For $0<\varsigma<1$ and $\omega_n>0$, we like to investigate its response due to a unit step input



Transient   Steady State

Two types of responses that are of interest:
(A)Transient response
(B)Steady state response

(A) For transient response, we have 4 specifications:

(a) T_r – rise time = $\dfrac{\pi - \theta}{\omega_n \sqrt{1 - \varsigma^2}}$

(b) T_p – peak time = $\dfrac{\pi}{\omega_n \sqrt{1 - \varsigma^2}}$

(c) %MP – percentage maximum overshoot = $e^{-\frac{\pi \varsigma}{\sqrt{1-\varsigma^2}}} x100\%$

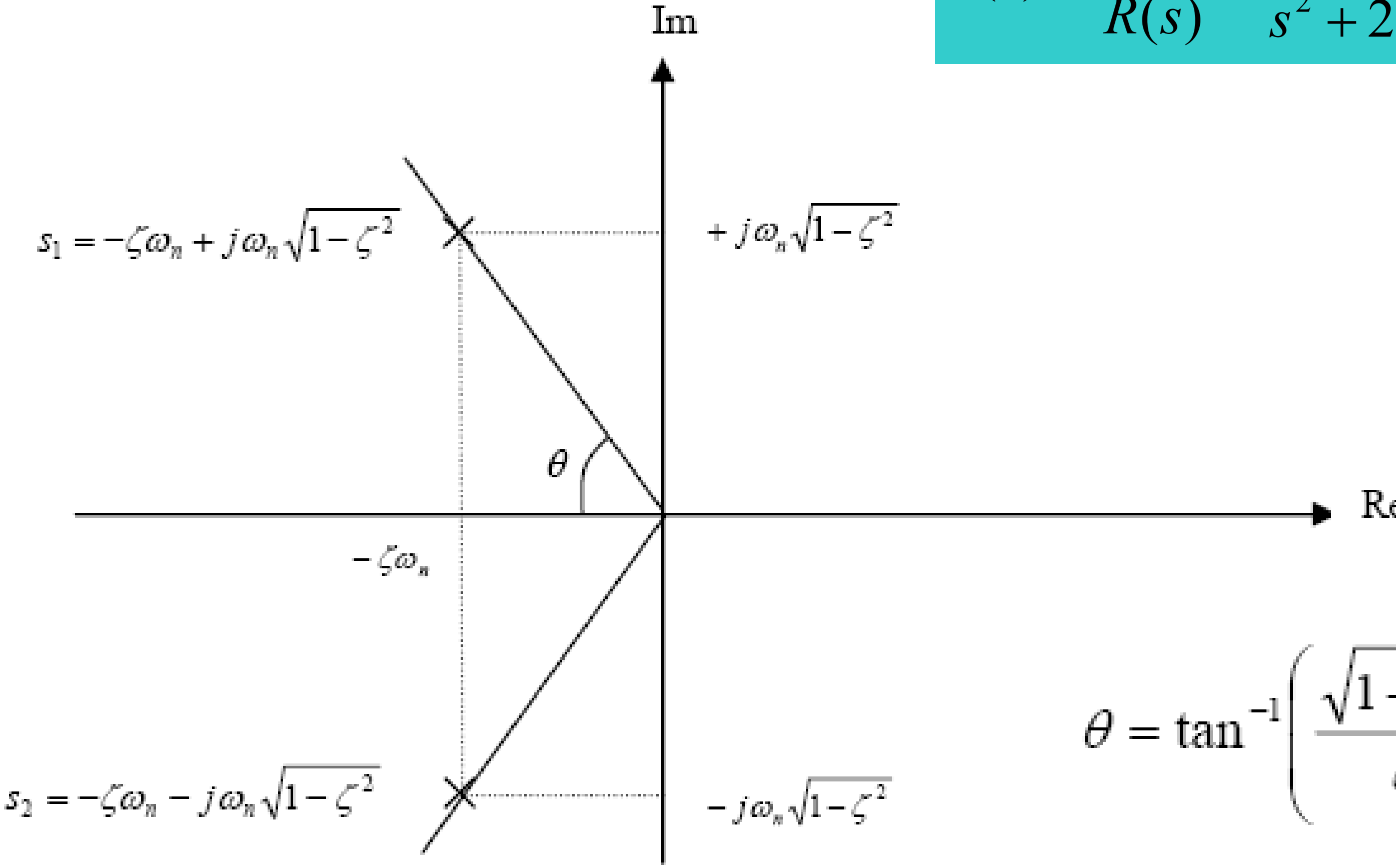
(d) T_s – settling time (2% error) = $\dfrac{4}{\varsigma \omega_n}$

(B) Steady State Response

(a) Steady State error

Second – Order System

$$T(s) = \frac{C(s)}{R(s)} = \frac{\omega_n^{\ 2}}{s^2 + 2\varsigma\omega_n s + \omega_n^{\ 2}}$$



$$\theta = \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right).$$

Mapping the poles into s-plane

Therefore, $$\boxed{\%MP = e^{-\frac{\pi\varsigma}{\sqrt{1-\varsigma^2}}} x100\%}$$

- For given $\%OS$, the damping ratio can be solved from the above equation;

$$\varsigma = \frac{-\ln(\%MP/100)}{\sqrt{\pi^2 + \ln^2(\%MP/100)}}$$

UNDERDAMPED

Example 2: *Find the natural frequency and damping ratio for the system with transfer function*

$$G(s)=\frac{36}{s^2+4.2s+36}$$

Solution:

Compare with general TF

$$G(s)=\frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$$

- $\omega_n = 6$
- $\xi = 0.35$

UNDERDAMPED

Example 3: Given the transfer function

$$G(s) = \frac{100}{s^2 + 15s + 100}$$

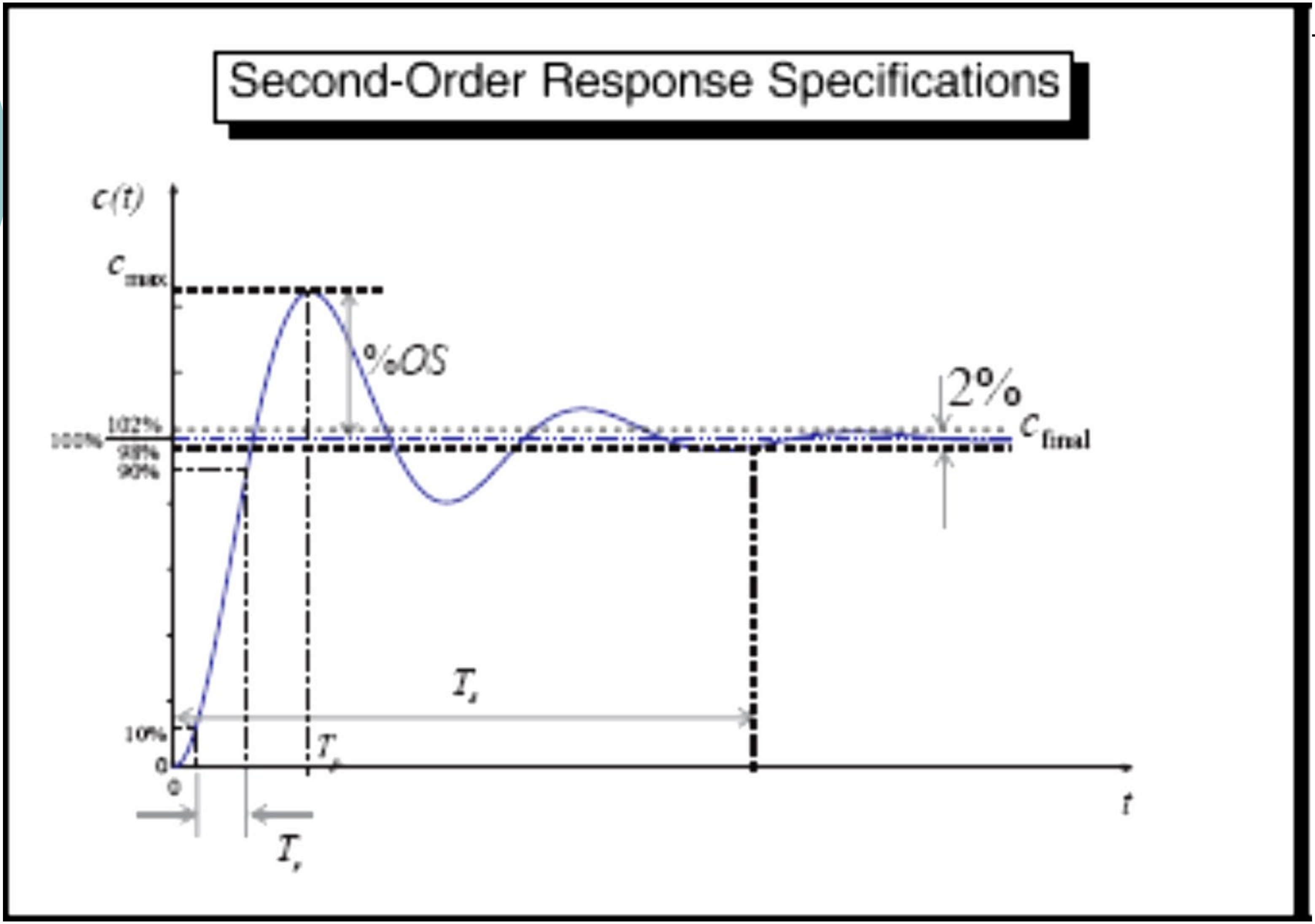
$$find \quad T_s, \%OS, T_p$$

Solution:

$$\omega_n = 10 \qquad \xi = 0.75$$

$$T_s = 0.533s, \ \%OS = 2.838\%, \ T_p = 0.475s$$

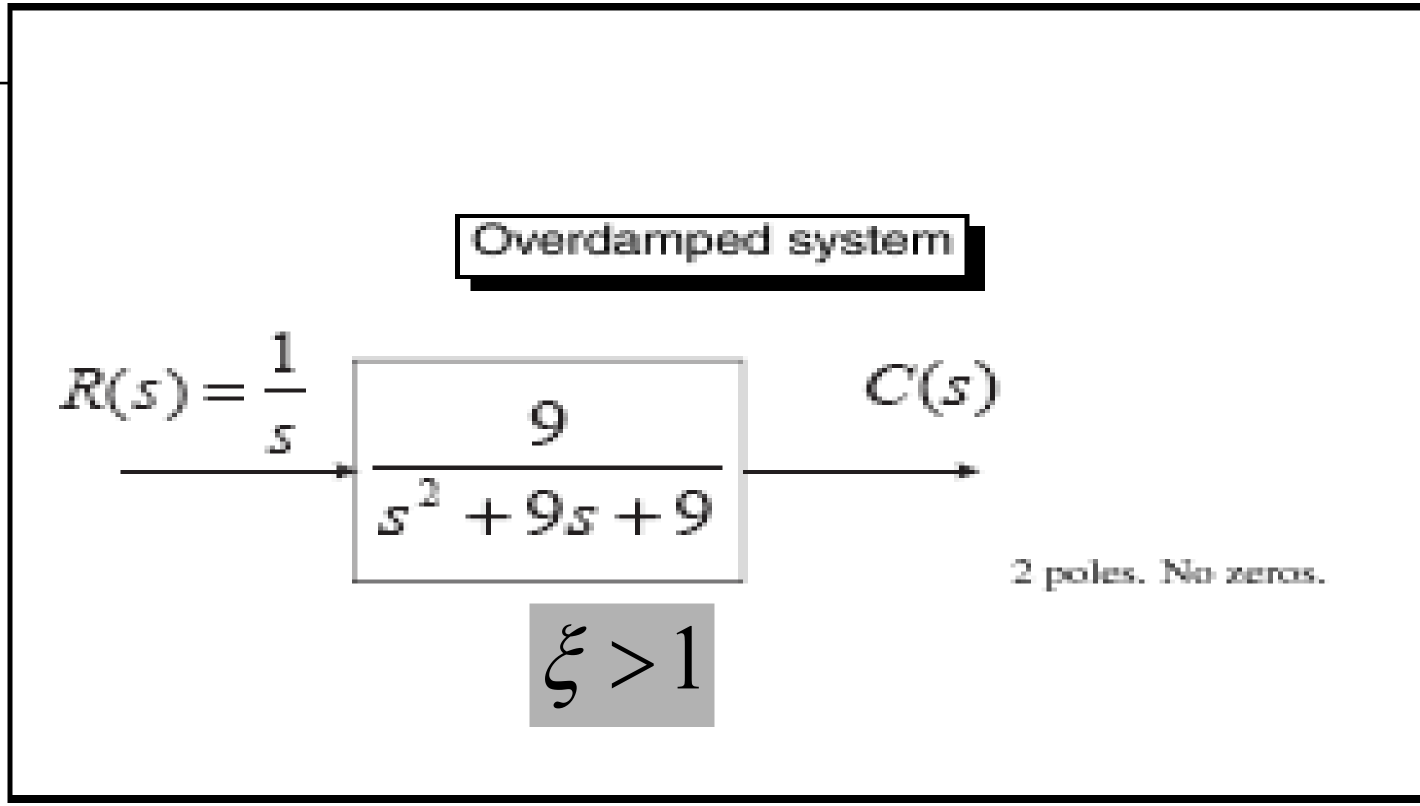
UNDERDAMPED



$$G(s) = \frac{b}{s^2 + as + b}$$

Overdamped Response

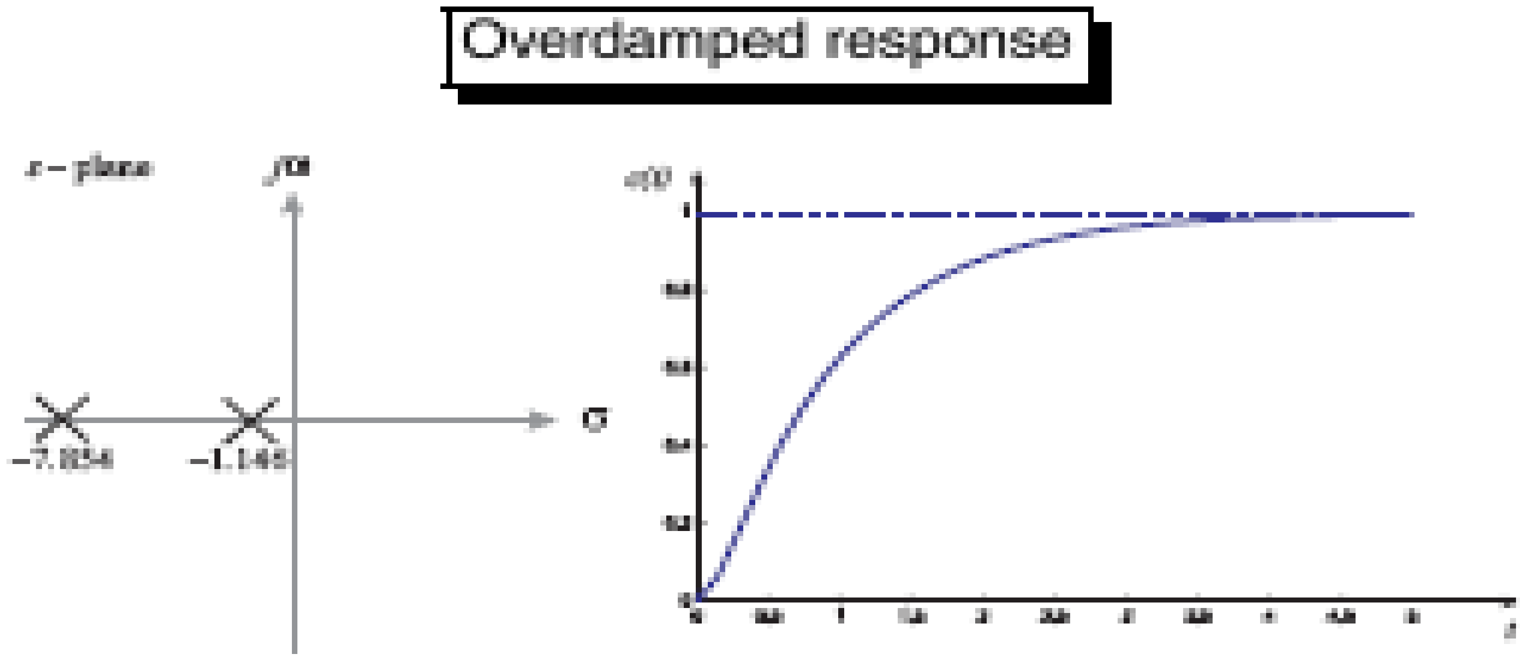
a = 9



$$C(s) = \frac{9}{s(s^2 + 9s + 9)} = \frac{9}{s(s + 7.854)(s + 1.146)}$$

s= 0; s = -7.854; s = -1.146 *(two real poles)*

$$c(t) = K_1 + K_2 e^{-7.854t} + K_3 e^{-1.146t}$$

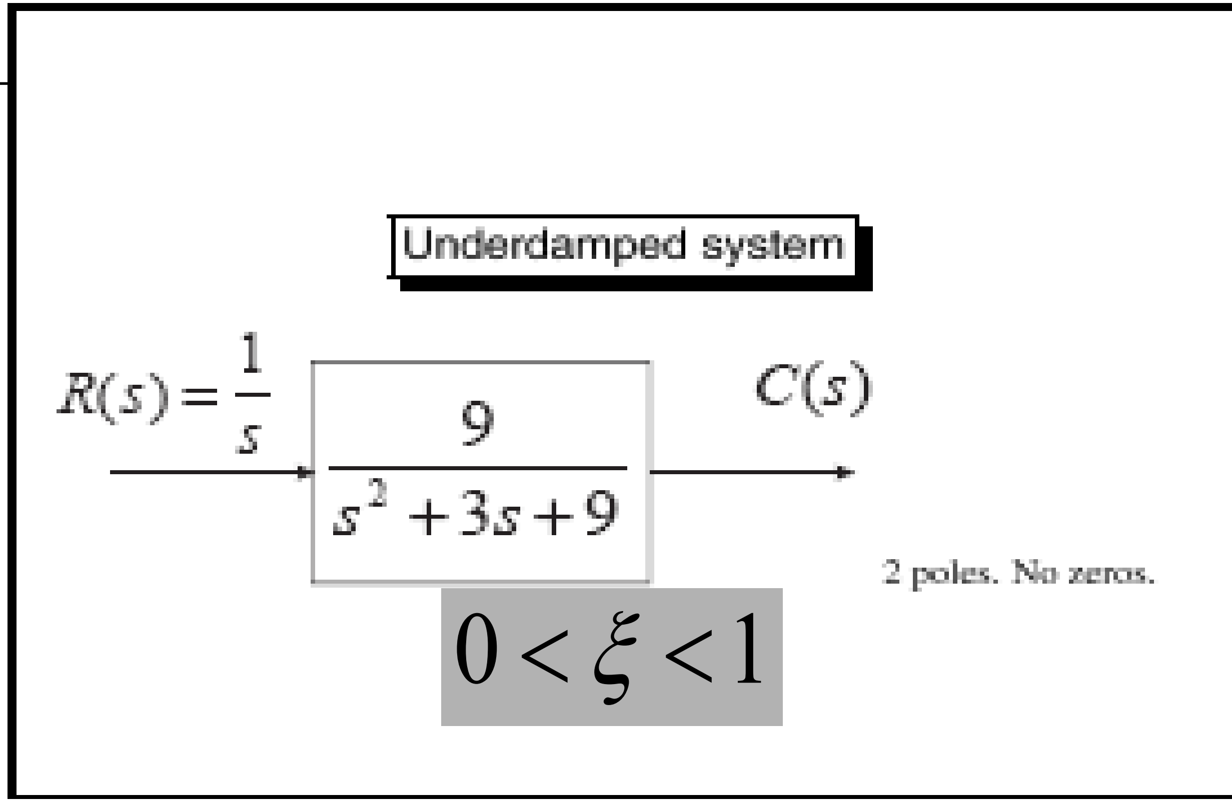


OVERDAMPED RESPONSE !!!

$$G(s) = \frac{b}{s^2 + as + b}$$

Underdamped Response

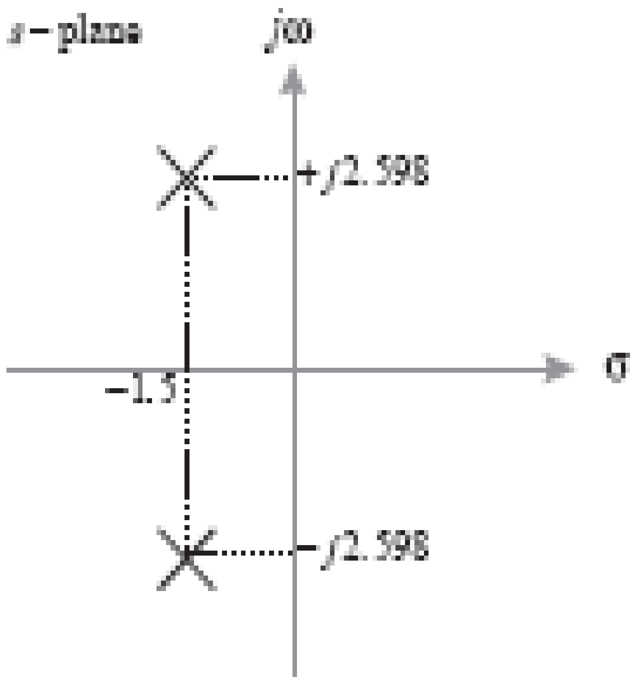
a = 3

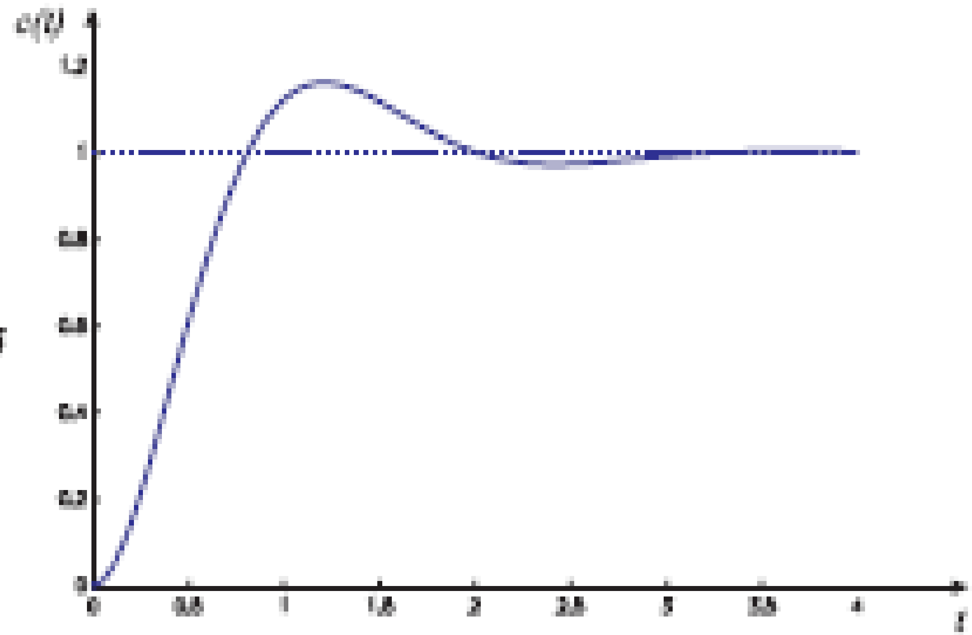


$$c(t) = K_1 + e^{-1.5t}(K_2 \cos 2.598t + K_3 \sin 2.598t)$$

s = 0; s = -1.5 ± j2.598 *(two complex poles)*

Underdamped response



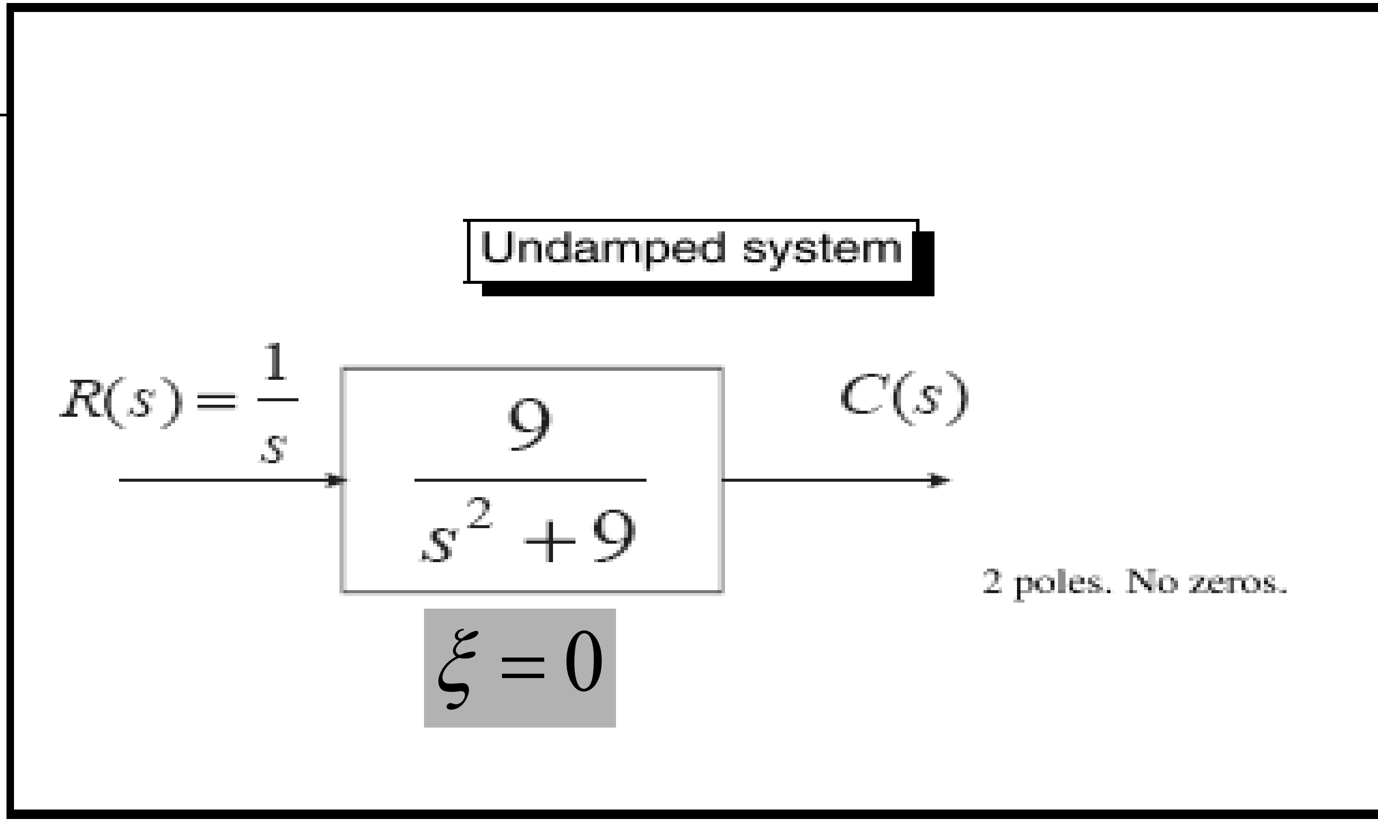


UNDERDAMPED RESPONSE !!!

Undamped Response

$$G(s) = \frac{b}{s^2 + as + b}$$

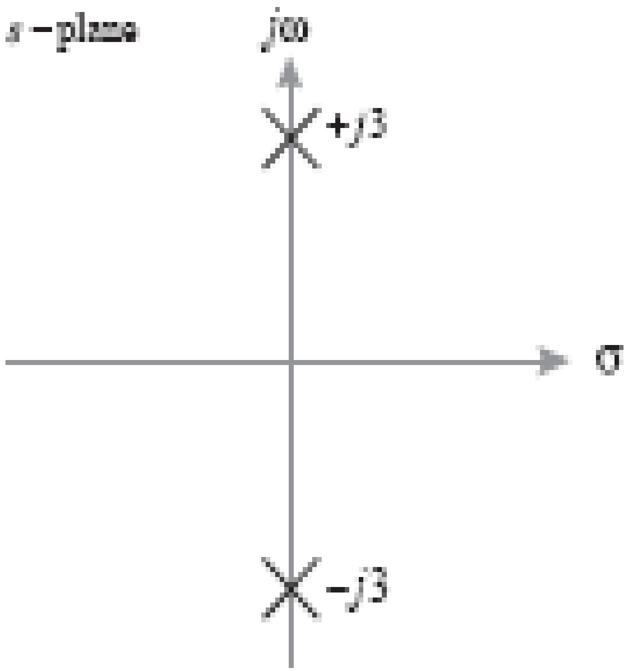
a = 0

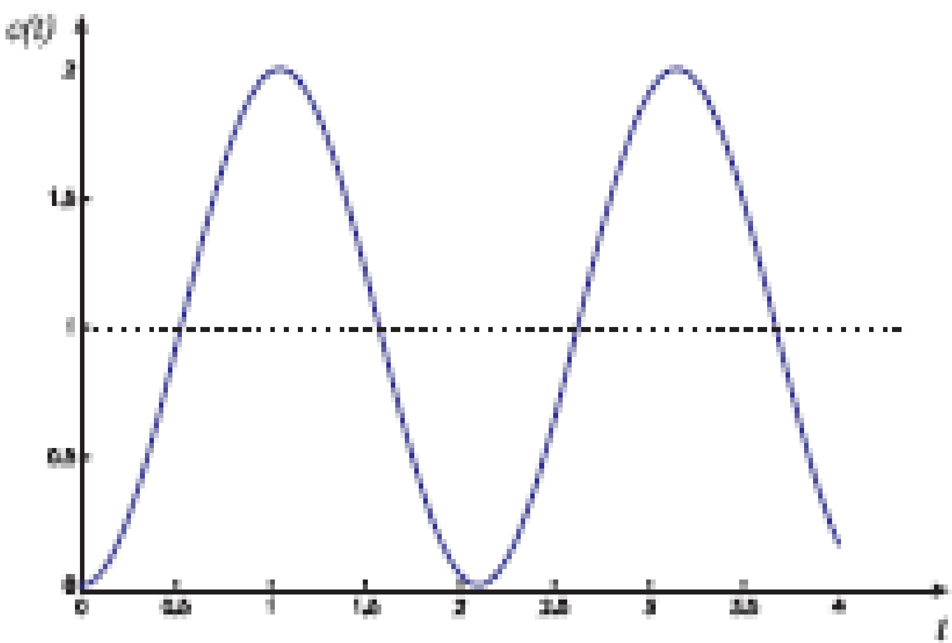


$$c(t) = K_1 + K_2 \cos 3t$$

s = 0; s = ± j3 (*two imaginary poles)*

Undamped response



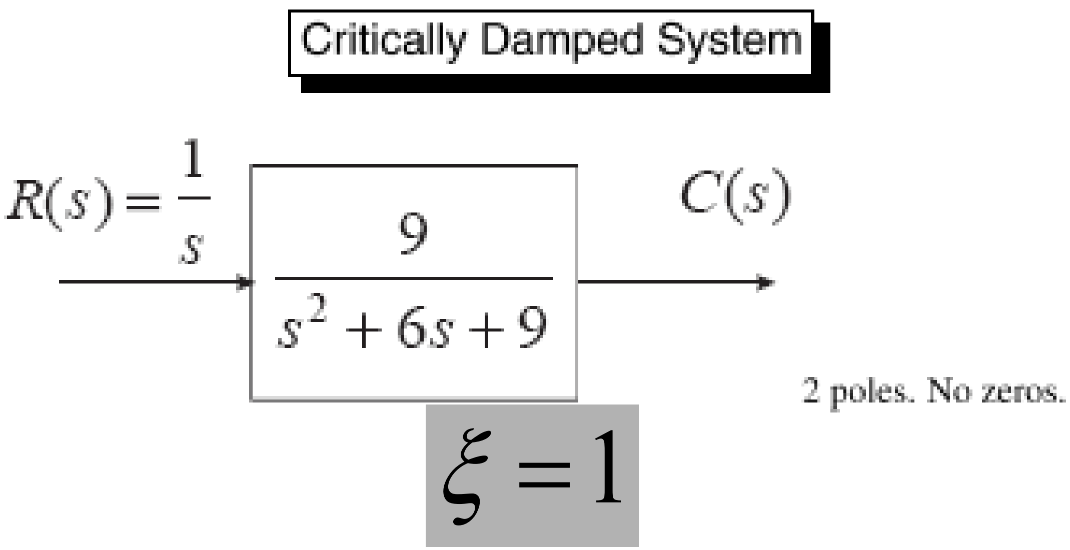


UNDAMPED RESPONSE !!!

Critically Damped System

$$G(s) = \frac{b}{s^2 + as + b}$$

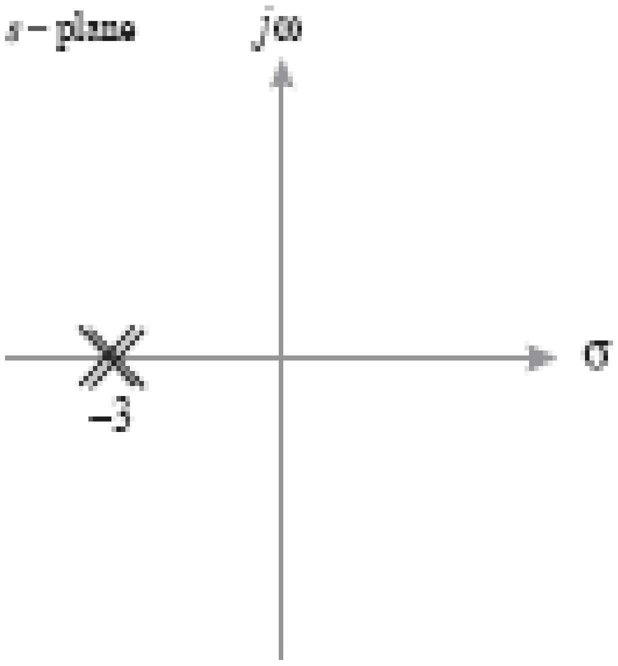
a = 6

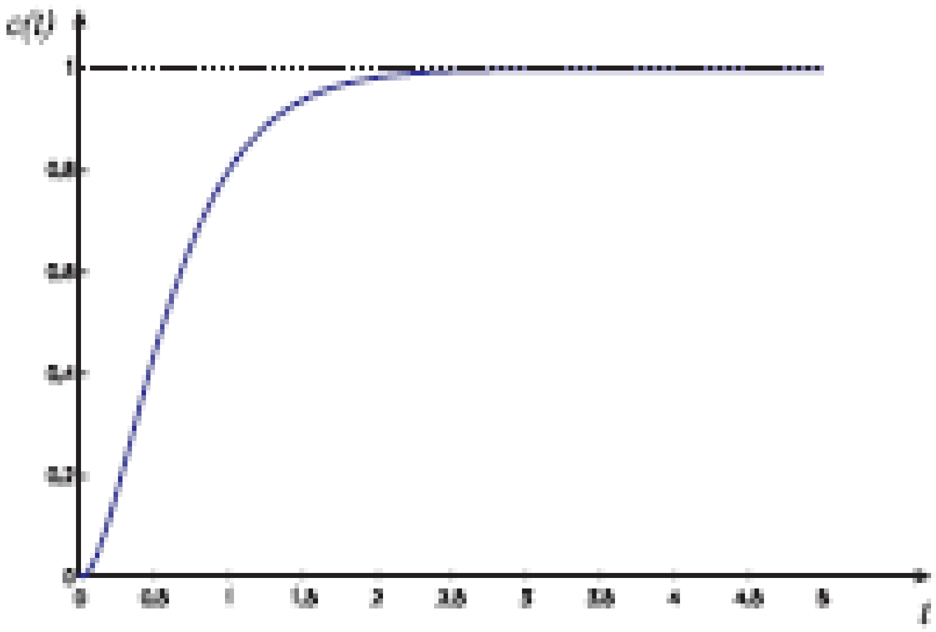


$$c(t) = K_1 + K_2 e^{-3t} + K_3 t e^{-3t}$$

S = 0; s = -3,-3 (*two real and equal poles)*

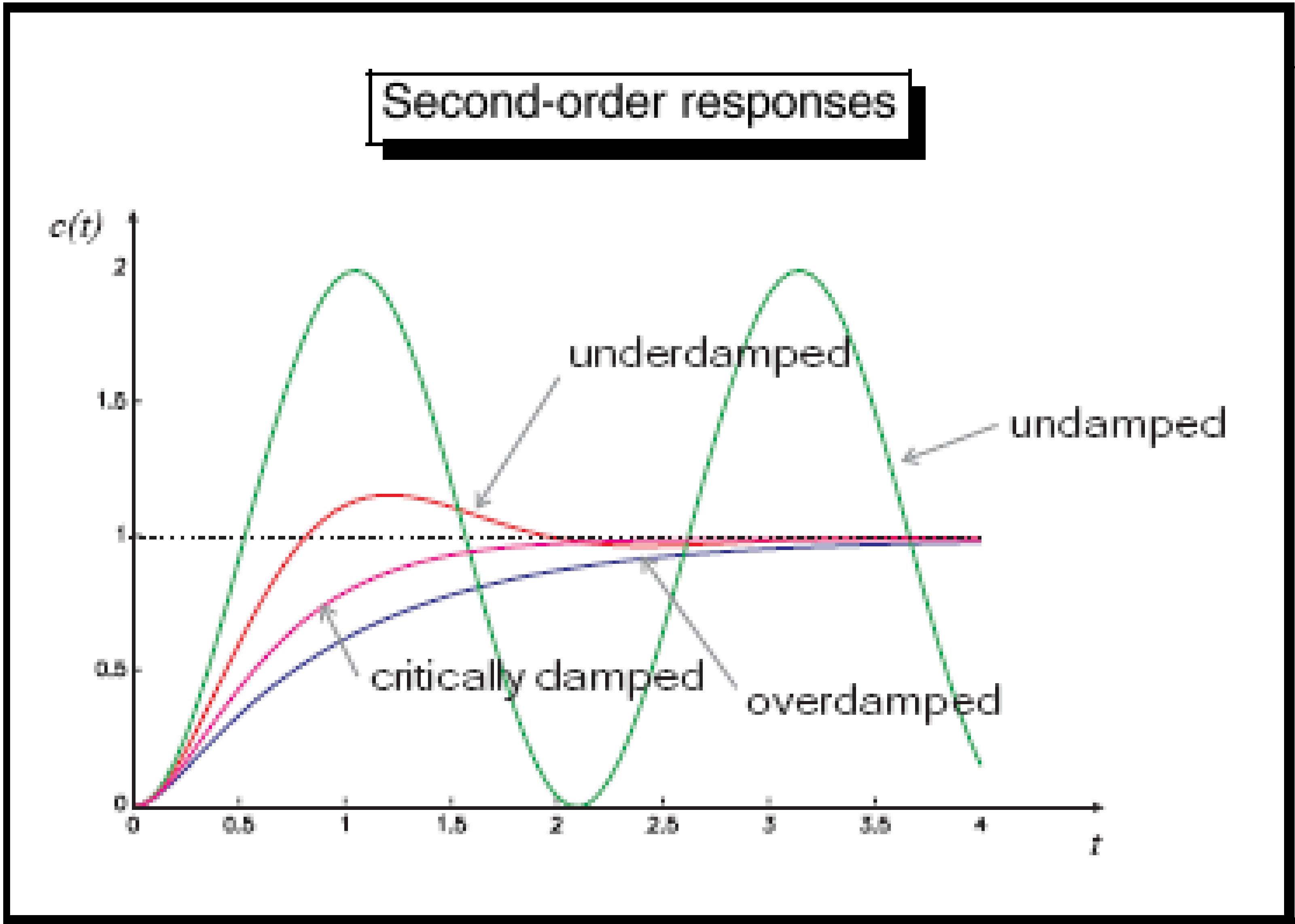
Critically Damped Response

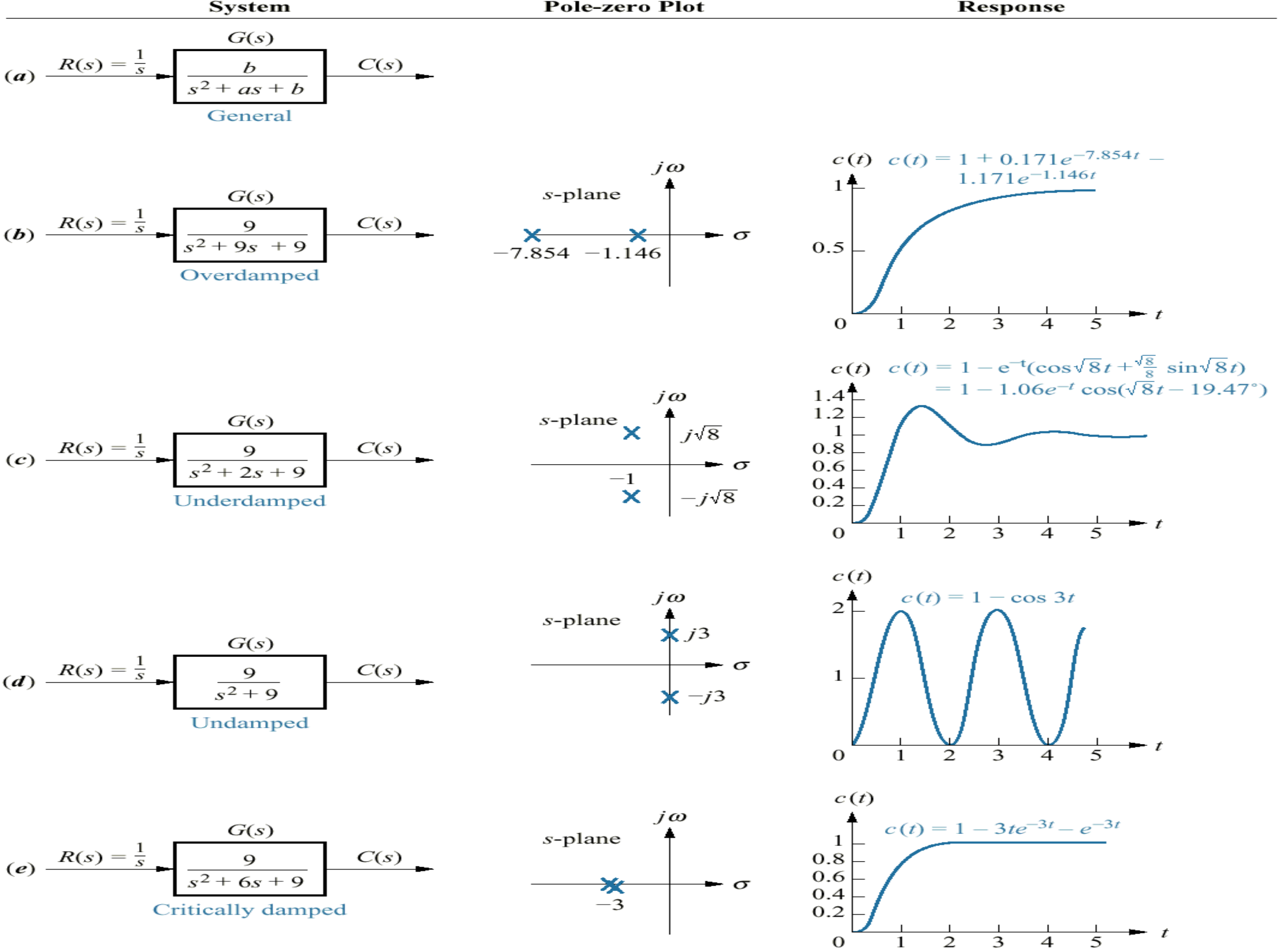




CRITICALLY DAMPED RESPONSE !!!

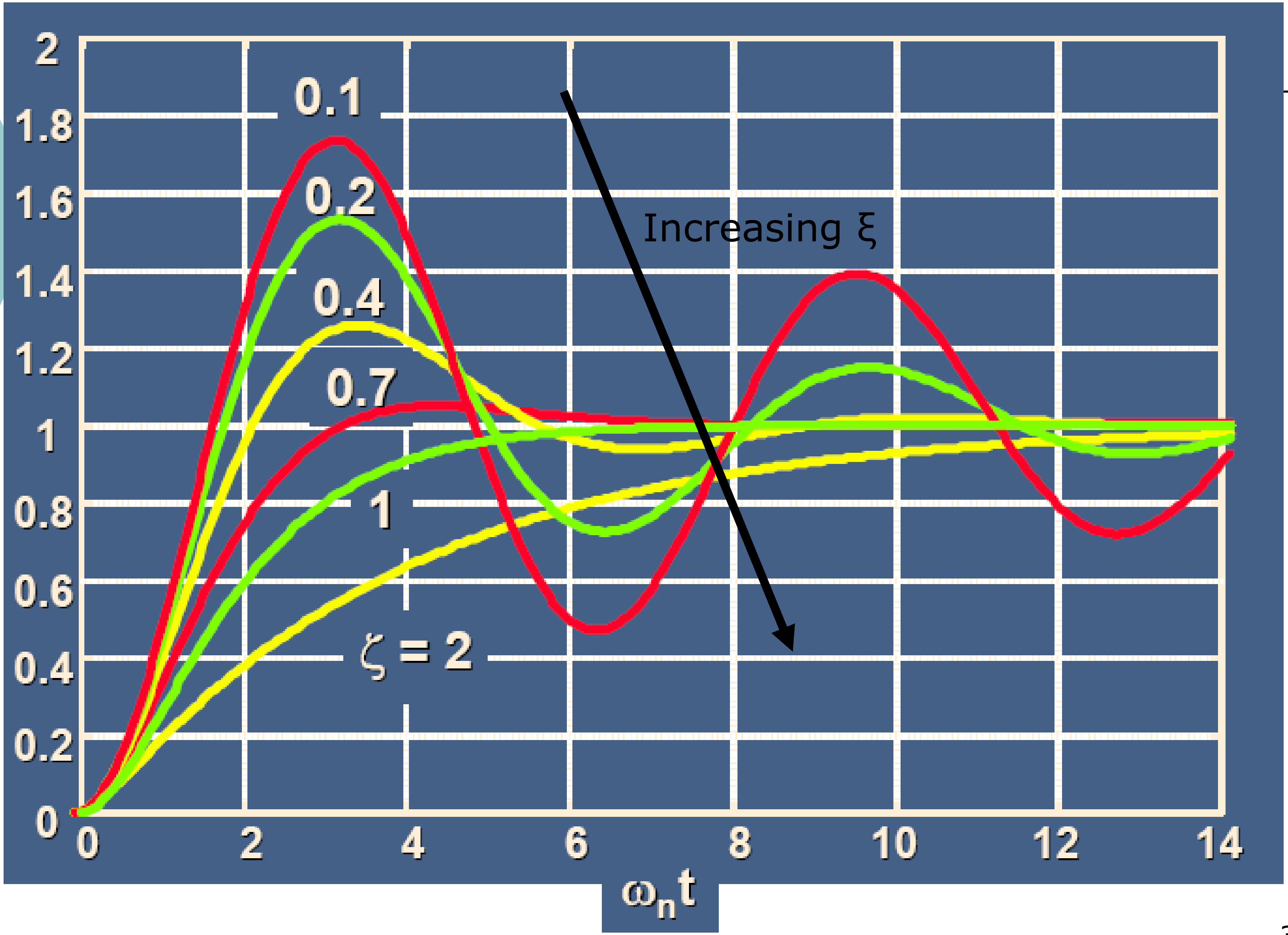
Second – Order System





System
Pole-zero Plot
Response
(a)
R(s) = 1/s
G(s)
b / (s^2 + as + b)
C(s)
General
(b)
R(s) = 1/s
G(s)
9 / (s^2 + 9s + 9)
C(s)
Overdamped
jω
s-plane
σ
−7.854
−1.146
c(t)
c(t) = 1 + 0.171e^{−7.854t} − 1.171e^{−1.146t}
(c)
R(s) = 1/s
G(s)
9 / (s^2 + 2s + 9)
C(s)
Underdamped
jω
s-plane
σ
j√8
−1
−j√8
c(t)
c(t) = 1 − e^{−t}(cos√8t + (√8/8) sin√8t)
= 1 − 1.06e^{−t} cos(√8t − 19.47°)
(d)
R(s) = 1/s
G(s)
9 / (s^2 + 9)
C(s)
Undamped
jω
s-plane
σ
j3
−j3
c(t)
c(t) = 1 − cos 3t
(e)
R(s) = 1/s
G(s)
9 / (s^2 + 6s + 9)
C(s)
Critically damped
jω
s-plane
σ
−3
c(t)
c(t) = 1 − 3te^{−3t} − e^{−3t}
t

Effect of different damping ratio, ξ



Second – Order System

Example 4: Describe the nature of the second-order system response via the value of the damping ratio for the systems with transfer function

1. $$G(s) = \frac{12}{s^2 + 8s + 12}$$

2. $$G(s) = \frac{16}{s^2 + 8s + 16}$$

3. $$G(s) = \frac{20}{s^2 + 8s + 20}$$

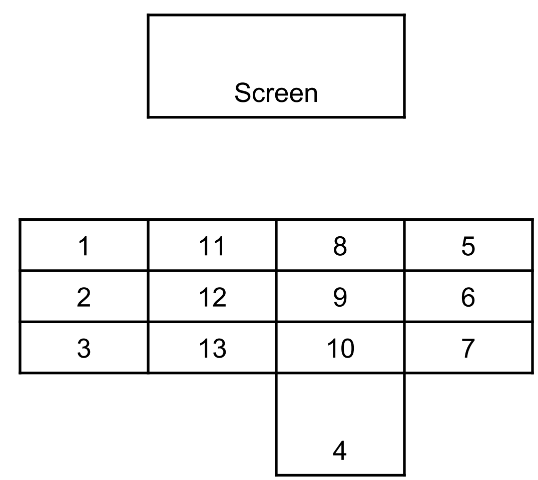
Do them as your own revision

Chapter 4

Transient & Steady State Response Analysis

Announcements!!!

- Textbooks –2 copies available
- Price of textbooks : Old :-RM 75.70, New:-RM 71.50. Your balance will be returned.
- Test 1 result: Insya Allah by Thursday. SMS me for confirmation
- Today's arrangement



Previous Class

- Chapter 4:
  - First Order System
  - Second Order System

Today's class

- Routh-Hurtwitz Criterion
- Steady-state error

Routh-Hurwitz Criterion

To check for stability of a system

Stability

- in order to know the location of the poles, we need to find the roots of the closed-loop characteristic equation.

- It turned out, however, that in order to judge a system's stability we don't need to know the actual location of the poles, just their sign. that is whether the poles are in the right-half or left-half plane.

- The Hurwitz criterion can be used to indicate that a characteristic polynomial with negative or missing coefficients is unstable.

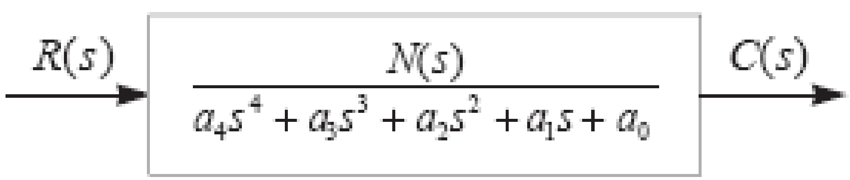
- The *Routh-Hurwitz Criterion* is called a *necessary* and *sufficient test* of stability because a polynomial that satisfies the criterion is guaranteed to stable. The criterion can also tell us how many poles are in the right-half plane or on the imaginary axis.

Stability

- need to construct a Routh array.

Consider the system shown in the Figure. The closed-loop characteristic equation is:

$$a_4 s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0 = 0.$$



- The Routh array is simply a rectangular matrix with one row for each power of *s* in the closed-loop characteristic polynomial

Stability

s^4	a_4	a_2	a_0
s^3	a_3	a_1	0
s^2	–	–	–
s^1	–	–	–
s^0	–	–	–

Table 1: Starting layout for Routh array

Stability

s^4	a_4	a_2	a_0
s^3	a_3	a_1	0
s^2	$b_1 = \dfrac{a_2 a_3 - a_4 a_1}{a_3}$	$b_2 = \dfrac{a_0 a_3 - a_4 \times 0}{a_3} = a_0$	$b_3 = \dfrac{0 \times a_3 - a_4 \times 0}{a_3} = 0$
s^1	$c_1 = \dfrac{a_1 b_1 - a_3 b_2}{b_1}$	$c_2 = \dfrac{0 \times b_1 - a_3 \times 0}{b_1} = 0$	$c_3 = \dfrac{0 \times b_1 - a_3 \times 0}{b_1} = 0$
s^0	$d_1 = \dfrac{b_2 \times c_1 - b_1 \times 0}{c_1} = b_2$	$d_2 = \dfrac{0 \times c_1 - b_1 \times 0}{c_1} = 0$	$d_3 = \dfrac{0 \times c_1 - b_1 \times 0}{c_1} = 0$

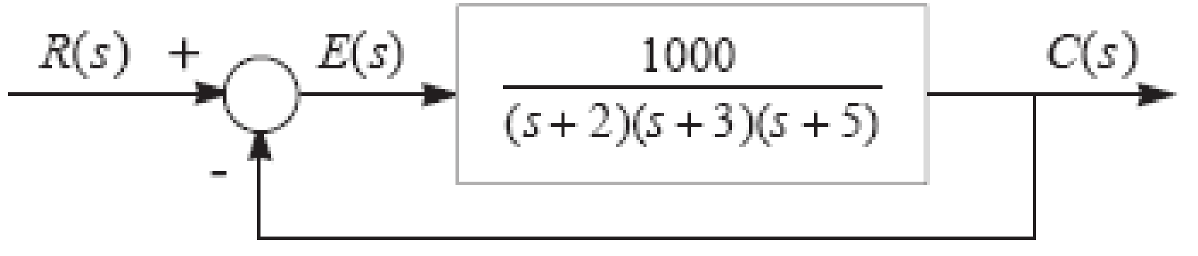
Table 2: Completed Routh array

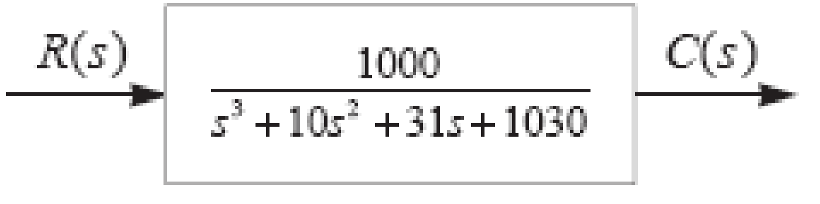
Stability

The Routh-Hurwitz Criterion: The number of roots of the characteristic polynomial that are in the right-half plane is equal to the number of sign changes in the first column of the *Routh Array*. If there are no sign changes, the system is stable.

Example:

Test the stability of the closed-loop system





Stability

Solution: Since all the coefficients of the closed-loop characteristic equation $s^3 + 10s^2 + 31s + 1030$ are present, the system passes the Hurwitz test. So we must construct the Routh array in order to test the stability further.

s^3	1	31	0
s^2	10	1030	0
s^1	–	–	–
s^0	–	–	–

s^3	1	31	0
s^2	1	103	0
s^1	–	–	–
s^0	–	–	–

Stability

s^3	1	31
s^2	1	103
s^1	$\frac{31 \times 1 - 1 \times 103}{1} = -72$	$\frac{0 \times 1 - 1 \times 0}{1} = 0$
s^0	$\frac{-72 \times 103 - 1 \times 0}{-72} = 103$	$\frac{-72 \times 0 - 1 \times 0}{-72} = 0$

For clarity, we can rewrite the array:

$$\begin{pmatrix} 1 & 31 & 0 \\ 1 & 103 & 0 \\ -72 & 0 & 0 \\ 103 & 0 & 0 \end{pmatrix}$$

Stability

and now it is clear that column 1 of the Routh array is:

First sign changes

Second sign changes

$$\begin{pmatrix} 1 \\ 1 \\ -72 \\ 103 \end{pmatrix}$$

- and it has two sign changes (from 1 to -72 and from -72 to 103). Hence the system is *unstable* with *two poles in the right-half plane*.

Any Q's so far ?

Stability

Special Case:

1. a zero may appear in the first column of the array
   - Zero Only in the First Column

2. a complete row can become zero
   - Entire Row Is Zero

Stability (Special Case 1)

Consider the control system with closed-loop transfer function:

$$G_c(s) = \frac{10}{s^5 + 2s^4 + 3s^3 + 6s^2 + 5s + 3}.$$

Routh array will be:

s^5	1	3	5
s^4	2	6	3
s^3	$0 \rightarrow \epsilon$	$7/2$	0
s^2	$\frac{6\epsilon - 7}{\epsilon}$	3	0
s^1	$\frac{42\epsilon - 49 - 6\epsilon^2}{12\epsilon - 14}$	0	0
s^0	3	0	0

Considering just the sign changes in column 1:

Label	First column	$\epsilon \rightarrow 0^+$	$\epsilon \rightarrow 0^-$
s^5	1	$+$	$+$
s^4	2	$+$	$+$
s^3	ϵ	$+$	$-$
s^2	$\frac{6\epsilon - 7}{\epsilon}$	$-$	$+$
s^1	$\frac{42\epsilon - 49 - 6\epsilon^2}{12\epsilon - 14}$	$+$	$+$
s^0	3	$+$	$+$

- If is chosen *positive* there are *two sign changes*. If is chosen *negative* there are also *two sign changes*. Hence the system has two poles in the right-half plane and it doesn't matter whether we chose to approach zero from the positive or the negative side.

Stability (Special Case 2)

Consider the control system with closed-loop transfer function:

$$G_c(s) = \frac{10}{s^5 + 7s^4 + 6s^3 + 42s^2 + 8s + 56}.$$

Routh array will be:

s^5	1	6	8
s^4	$7 \to 1$	$42 \to 6$	$56 \to 8$
s^3	0	0	0
s^2	–	–	–
s^1	–	–	–
s^0	–	–	–

divide by '7' for convenience

replace the zero row with a row formed from the coefficients of the derivative:

s^5	1	6	8
s^4	1	6	8
s^3	$0 \to 4 \to 1$	$0 \to 12 \to 3$	0
s^2	–	–	–
s^1	–	–	–
s^0	–	–	–

divide by '4' for convenience

$$Q(s) = s^4 + 6s + 8$$

Differentiate

$$\frac{dQ(s)}{ds} = 4s^3 + 12s + 0$$

s^5	1	6	8
s^4	1	6	8
s^3	1	3	0
s^2	3	8	0
s^1	1/3	0	0
s^0	8	0	0

There are no sign changes in the completed Routh array, hence the system is *stable*.

Example 1:

Construct a Routh table and determine the number of roots with *positive real parts* for the equation;

$$2s^3 + 4s^2 + 4s + 12 = 0$$

Solution:

✓ Since there are two changes of sign in the first columm of Routh table, the equation above have two roots at right side (positive real parts).

Example 2:

The characteristic equation of a given system is:

$$s^4 + 6s^3 + 11s^2 + 6s + K = 0$$

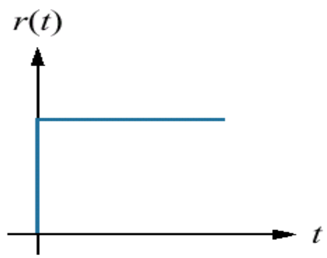
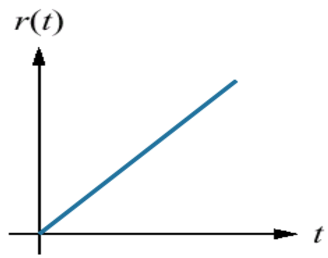
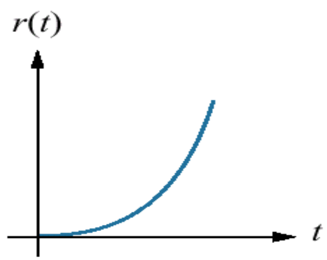
What restrictions must be placed upon the parameter *K* in order to ensure that the system is stable?

Solution:

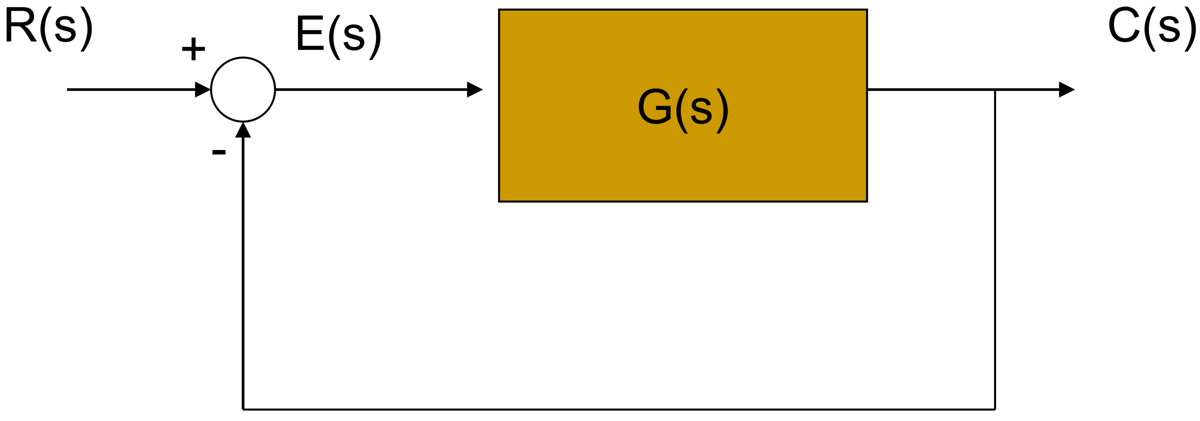
For the system to be stable, $60 - 6K < 0$, or $k < 10$, and $K > 0$. Thus $0 < K < 10$

Steady State Error Analysis

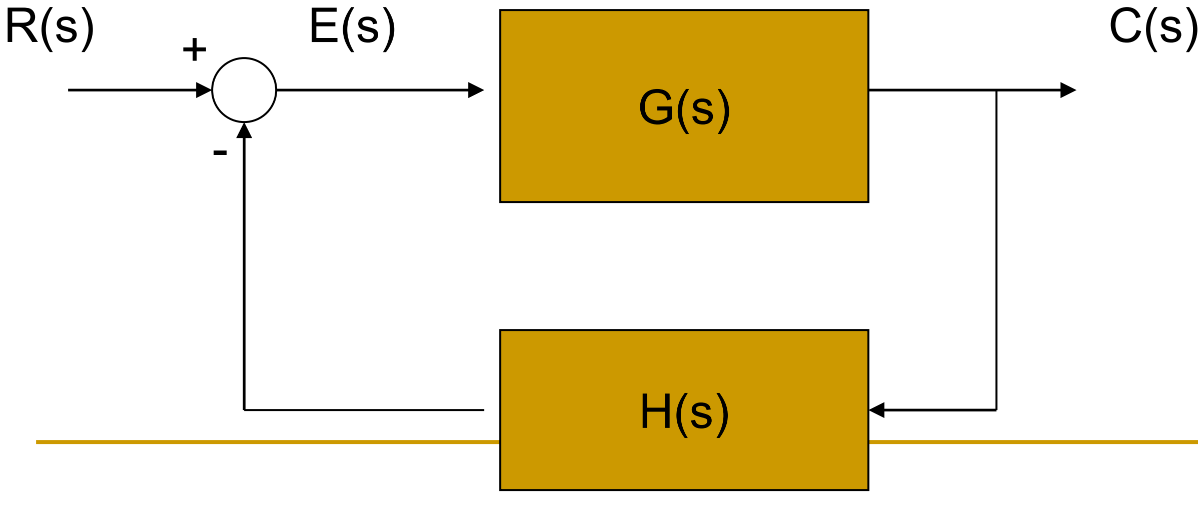
Test Waveform for evaluating steady-state error

Waveform	Name	Physical interpretation	Time function	Laplace transform
	Step	Constant position	1	$\frac{1}{s}$
	Ramp	Constant velocity	t	$\frac{1}{s^2}$
	Parabola	Constant acceleration	$\frac{1}{2}t^2$	$\frac{1}{s^3}$

Steady-state error analysis



Unity feedback
$H(s)=1$



Non-unity feedback
$H(s)\neq 1$

Steady-state error analysis

For unity feedback system:

$$E(s) = R(s) - C(s) \quad \rightarrow \qquad \text{System error}$$

For a non-unity feedback system:

$$E(s) = R(s) - H(s)C(s) \quad \rightarrow \qquad \text{Actuating error}$$

Steady-state error analysis

Consider a unity feedback system, if the inputs are step response, ramp & parabolic (no sinusoidal input). We want to find the steady-state error

$$e_{ss} = \lim_{t \to \infty} e(t)$$

Where, $$e(t) = r(t) - c(t)$$

By Final Value Theorem:

$$\boxed{e_{ss} = \lim_{t \to \infty} e(t) \cong \lim_{s \to 0} sE(s)}$$

Steady-state error analysis

Consider Unity Feedback System

$$E(s) = R(s) - C(s) \longrightarrow \quad (1)$$

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)} \longrightarrow \quad (2)$$

Substitute (2) into (1)

$$\therefore E(s) = R(s) - \frac{G(s)}{1+G(s)}R(s) = \frac{1}{1+G(s)}R(s) \longrightarrow \quad (3)$$

Steady-state error analysis

Based on equation (3), it can be seen that E(s) depends on:

(a) Input signal, R(s)

(b) G(s), open loop transfer function

Now, assume:

$$G(s) = \frac{K \prod_{i=1}^{M} (s + z_i)}{S^{N} \prod_{j=1}^{\theta} (s + p_j)}$$

type N

Cases to be considered:

$$(A) R(s) = \frac{1}{s}$$

$$(B) R(s) = \frac{1}{s^2}$$

$$(C) R(s) = \frac{1}{s^3}$$

Case (A): Input is a unit step R(s)=1/s

$$E(s)=\frac{1}{1+G(s)}R(s)=\frac{1/s}{1+G(s)}$$

$$e_{ss}=Steady-State\_Error\cong\lim_{s\to 0}sE(s)$$

$$e_{ss}=\lim_{s\to 0}s\left[\frac{1/s}{1+G(s)}\right]=\lim_{s\to 0}\left[\frac{1}{1+G(s)}\right]$$

$$=\left[\frac{1}{1+\lim_{s\to 0}G(s)}\right]=\left[\frac{1}{1+K_p}\right]$$

where $\boxed{K_p=\lim_{s\to 0}G(s)}$ → "Static Position Error Constant"

If N = 0, K_p = constant

$$e_{ss} = \frac{1}{1+K_p} = finite$$

If N ≥ 1, K_p = infinite

$$e_{ss} = \frac{1}{1+K_p} = \frac{1}{1+\infty} = 0$$

For unit step response, as the type of system increases (N ≥ 1), the steady state error goes to zero

Case (B): Input is a unit ramp R(s)=1/s^2

$$E(s)=\frac{1}{1+G(s)}R(s)=\frac{1/s^2}{1+G(s)}$$

$$e_{ss}=Steady-State\_Error\cong\lim_{s\to 0}sE(s)$$

$$e_{ss}=\lim_{s\to 0}s\left[\frac{1/s^2}{1+G(s)}\right]=\lim_{s\to 0}\left[\frac{1}{s+sG(s)}\right]$$

$$=\left[\frac{1}{0+\lim_{s\to 0}sG(s)}\right]=\left[\frac{1}{\lim_{s\to 0}sG(s)}\right]\cong\frac{1}{K_V}$$

where $\boxed{K_v=\lim_{s\to 0}sG(s)}$ → "Static Velocity Error Constant"

If N = 0, $K_v = s\dfrac{\pi(s+z_i)}{\pi(s+p_j)} = 0,$ $\qquad e_{ss} = \dfrac{1}{K_v} = \infty$

If N =1, K_v = finite $\qquad e_{ss} = \dfrac{1}{K_v} = finite$

If N ≥2, K_v = infinite $\qquad e_{ss} = \dfrac{1}{K_v} = \dfrac{1}{\infty} = 0$

For unit ramp response, the steady state error in infinite for system of type zero, finite steady state error for system of type 1, and zero steady state error for systems with type greater or equal to 2.

Case (C): Input is a parabolic, R(s)=1/s^3

$$E(s) = \frac{1}{1+G(s)} R(s) = \frac{1/s^3}{1+G(s)}$$

$$e_{ss} = Steady - State\_Error \cong \lim_{s\to 0} sE(s)$$

$$e_{ss} = \lim_{s\to 0} s\left[\frac{1/s^3}{1+G(s)}\right] = \lim_{s\to 0}\left[\frac{1}{s^2 + s^2 G(s)}\right]$$

$$= \left[\frac{1}{0+\lim_{s\to 0} s^2 G(s)}\right] = \left[\frac{1}{\lim_{s\to 0} s^2 G(s)}\right] \cong \frac{1}{K_a}$$

where $\boxed{K_a = \lim_{s\to 0} s^2 G(s)}$ → “Static Acceleration Error Constant”

If N = 0, $K_a = s^2 \dfrac{\pi(s+z_i)}{\pi(s+p_j)} = 0,$ $e_{ss} = \dfrac{1}{K_a} = \infty$

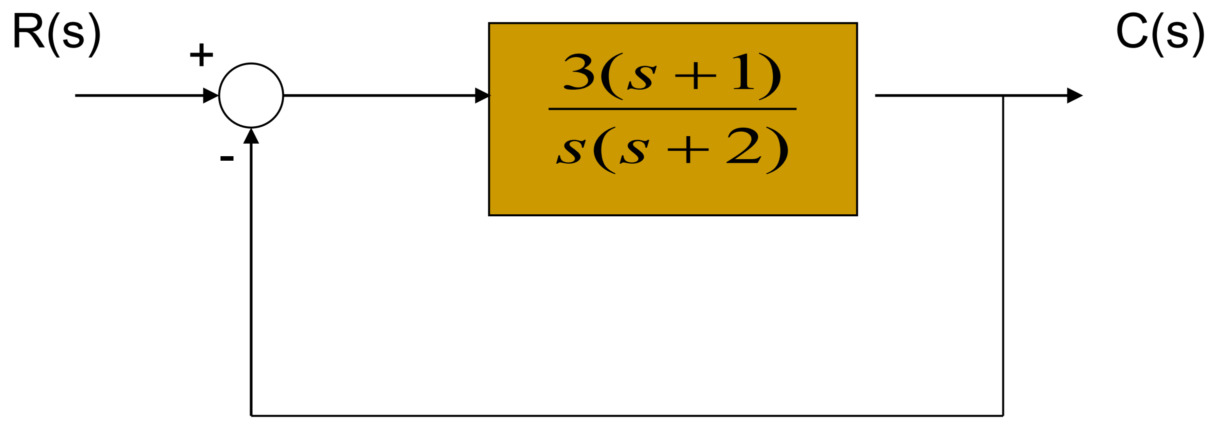
If N =1, $K_a = 0$ $e_{ss} = \dfrac{1}{K_a} = \infty$

If N = 2, K_a = constant $e_{ss} = \dfrac{1}{K_a} = finite$

If N ≥3 , K_a = infinite $e_{ss} = \dfrac{1}{K_a} = \dfrac{1}{\infty} = 0$

→ Increasing system type (N) will accommodate more different inputs.

Example 3



If r(t) = (2+3t)u(t), find the steady state error (e_{ss}) for the given system.

Solution:

$$K_p = \lim_{s\to 0} G(s) = \infty$$

$$K_v = \lim_{s\to 0} sG(s) = 3/2$$

$$e_{ss} = \frac{2}{1+K_p} + \frac{3}{K_v} = \frac{2}{1+\infty} + \frac{3}{3/2} = 2$$

Any Q's?

E5.4 A feedback system with negative unity feedback has a loop transfer function

$$L(s) = G_c(s)G(s) = \frac{2(s+8)}{s(s+4)}.$$

(a) Determine the closed-loop transfer function $T(s) = Y(s)/R(s)$. (b) Find the time response, $y(t)$, for a step input $r(t) = A$ for $t > 0$. (c) Using Figure 5.13(a), determine the overshoot of the response. (d) Using the final-value theorem, determine the steady-state value of $y(t)$.

Answer: (b) $y(t) = 1 - 1.07e^{-3t}\sin\left(\sqrt{7}t + 1.2\right)$

SOLVE HERE

E5.8 A control system for positioning the head of a floppy disk drive has the closed-loop transfer function

$$T(s) = \frac{11.1(s + 18)}{(s + 20)(s^2 + 4s + 10)}.$$

Plot the poles and zeros of this system and discuss the dominance of the complex poles. What overshoot for a step input do you expect?

SOLVE HERE

E5.9 A unity negative feedback control system has the loop transfer function

$$L(s) = G_c(s)G(s) = \frac{K}{s\left(s + \sqrt{2K}\right)}.$$

(a) Determine the percent overshoot and settling time (using a 2% settling criterion) due to a unit step input.

(b) For what range of K is the settling time less than 1 second?

SOLVE HERE

E5.13 For the system with unity feedback shown in Figure E5.11, determine the steady-state error for a step and a ramp input when

$$G(s) = \frac{20}{s^2 + 14s + 50}.$$

Answer: $e_{ss} = 0.71$ for a step and $e_{ss} = \infty$ for a ramp.

SOLVE HERE

E5.20 Consider the closed-loop system in Figure E5.19, where

$$G_c(s)G(s) = \frac{s+1}{s^2 + 03s} \text{ and } H(s) = K_a.$$

(a) Determine the closed-loop transfer function $T(s) = Y(s)/R(s)$.

(b) Determine the steady-state error of the closed-loop system response to a unit ramp input, $R(s) = 1/s^2$.

(c) Select a value for K_a so that the steady-state error of the system response to a unit step input, $R(s) = 1/s$, is zero.

SOLVE HERE

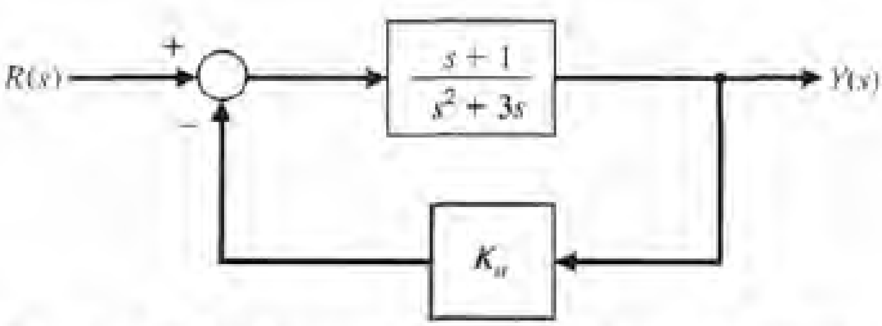


FIGURE E5.20 Nonunity closed-loop feedback control system with parameter K_a.

P5.20 A system is shown in Figure P5.20.

(a) Determine the steady-state error for a unit step input in terms of K and K_1, where $E(s) = R(s) - Y(s)$.

(b) Select K_1 so that the steady-state error is zero.

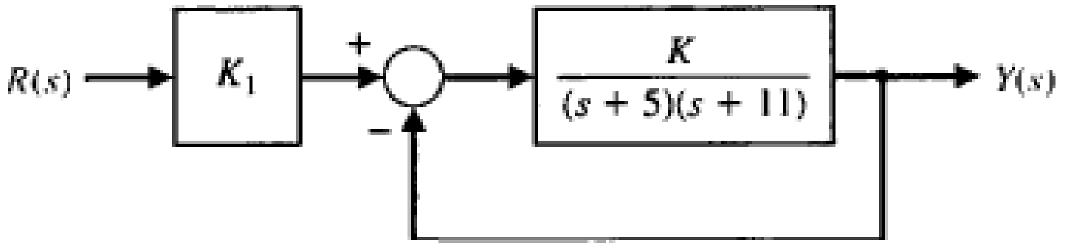


FIGURE P5.20 System with pregain, K_1.

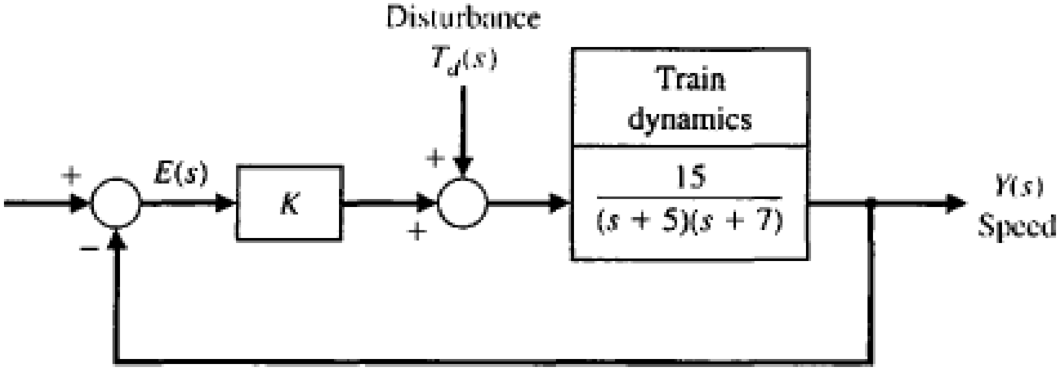
SOLVE HERE

AP5.4 The speed control of a high-speed train is represented by the system shown in Figure AP5.4 [17]. Determine the equation for steady-state error for K for a unit step input $r(t)$. Consider the three values for K equal to 1, 10, and 100.

(a) Determine the steady-state error.

(b) Determine and plot the response $y(t)$ for (i) a unit step input $R(s) = 1/s$ and (ii) a unit step disturbance input $T_d(s) = 1/s$.

(c) Create a table showing overshoot, settling time (with a 2% criterion), e_{ss} for $r(t)$, and $|y/t_d|_{max}$ for the three values of K. Select the best compromise value.



SOLVE HERE

Modern Control Systems (MCS)

Root Locus

Lecture Outline

- Construction of root loci
  - Angle and Magnitude Conditions
  - Illustrative Examples
- Closed loop stability via root locus
- Example of Root Locus
  - Root Locus of 1st order systems
  - Root Locus of 2nd order systems
  - Root Locus of Higher order systems

Construction of Root Loci

- Finding the roots of the characteristic equation of degree higher than 3 is laborious and will need computer solution.

- A simple method for finding the roots of the characteristic equation has been developed by W. R. Evans and used extensively in control engineering.

- This method, called the *root-locus method,* is one in which the roots of the characteristic equation are plotted for all values of a system parameter.

Construction of Root Loci

- The roots corresponding to a particular value of this parameter can then be located on the resulting graph.

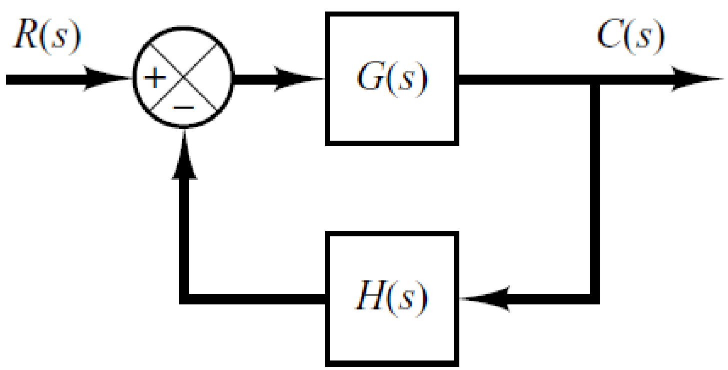
- Note that the parameter is usually the gain, but any other variable of the open-loop transfer function may be used.

- By using the root-locus method the designer can predict the effects on the location of the closed-loop poles of varying the gain value or adding open-loop poles and/or open-loop zeros.

Angle & Magnitude Conditions

- In constructing the root loci angle and magnitude conditions are important.

- Consider the system shown in following figure.



- The closed loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}$$

Construction of Root Loci

- The characteristic equation is obtained by setting the denominator polynomial equal to zero.

$$1 + G(s)H(s) = 0$$

- Or

$$G(s)H(s) = -1$$

- Where *G(s)H(s)* is a ratio of polynomial in s.

- Since *G(s)H(s)* is a complex quantity it can be split into angle and magnitude part.

Angle & Magnitude Conditions

- The angle of *G(s)H(s)=-1* is

$$\angle G(s)H(s) = \angle -1$$

$$\angle G(s)H(s) = \pm 180^\circ (2k+1)$$

- Where *k*=1,2,3…

- The magnitude of *G(s)H(s)=-1* is

$$|G(s)H(s)| = |-1|$$

$$|G(s)H(s)| = 1$$

Angle & Magnitude Conditions

- Angle Condition

$$\angle G(s)H(s) = \pm 180^\circ (2k+1) \qquad (k = 1,2,3\ldots)$$

- Magnitude Condition

$$\left| G(s)H(s) \right| = 1$$

- The values of s that fulfill both the angle and magnitude conditions are the roots of the characteristic equation, or the closed-loop poles.

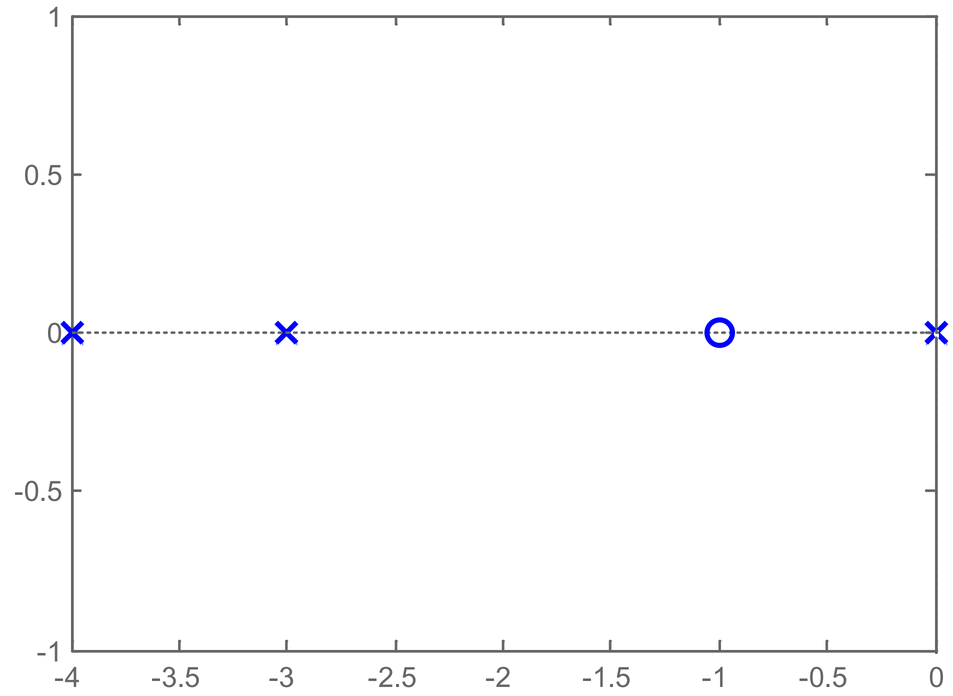
- A locus of the points in the complex plane satisfying the angle condition alone is the root locus.

Angle and Magnitude Conditions (Graphically)

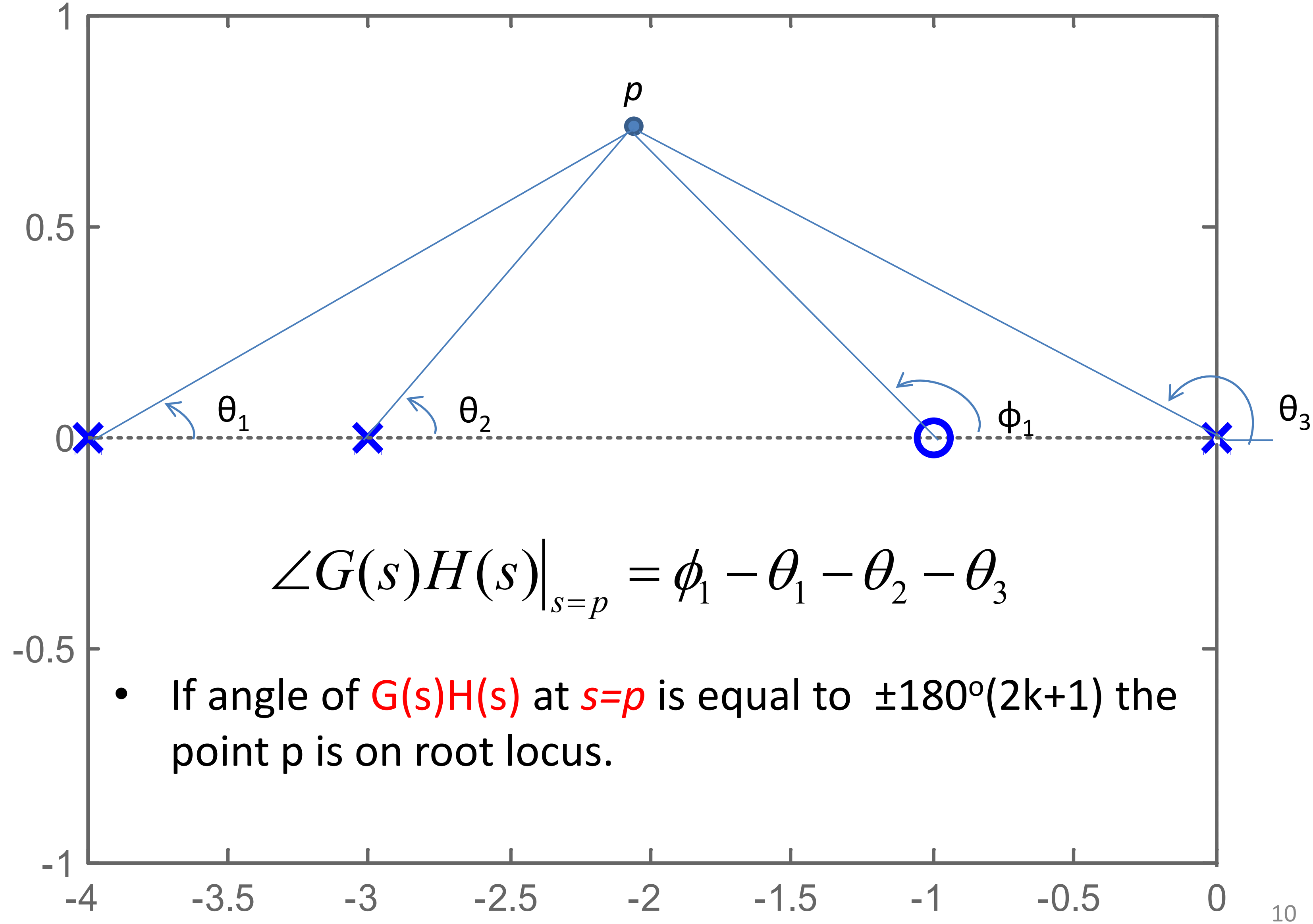
- To apply Angle and magnitude conditions graphically we must first draw the poles and zeros of G(s)H(s) in s-plane.

- For example if G(s)H(s) is given by

$$G(s)H(s) = \frac{s+1}{s(s+3)(s+4)}$$



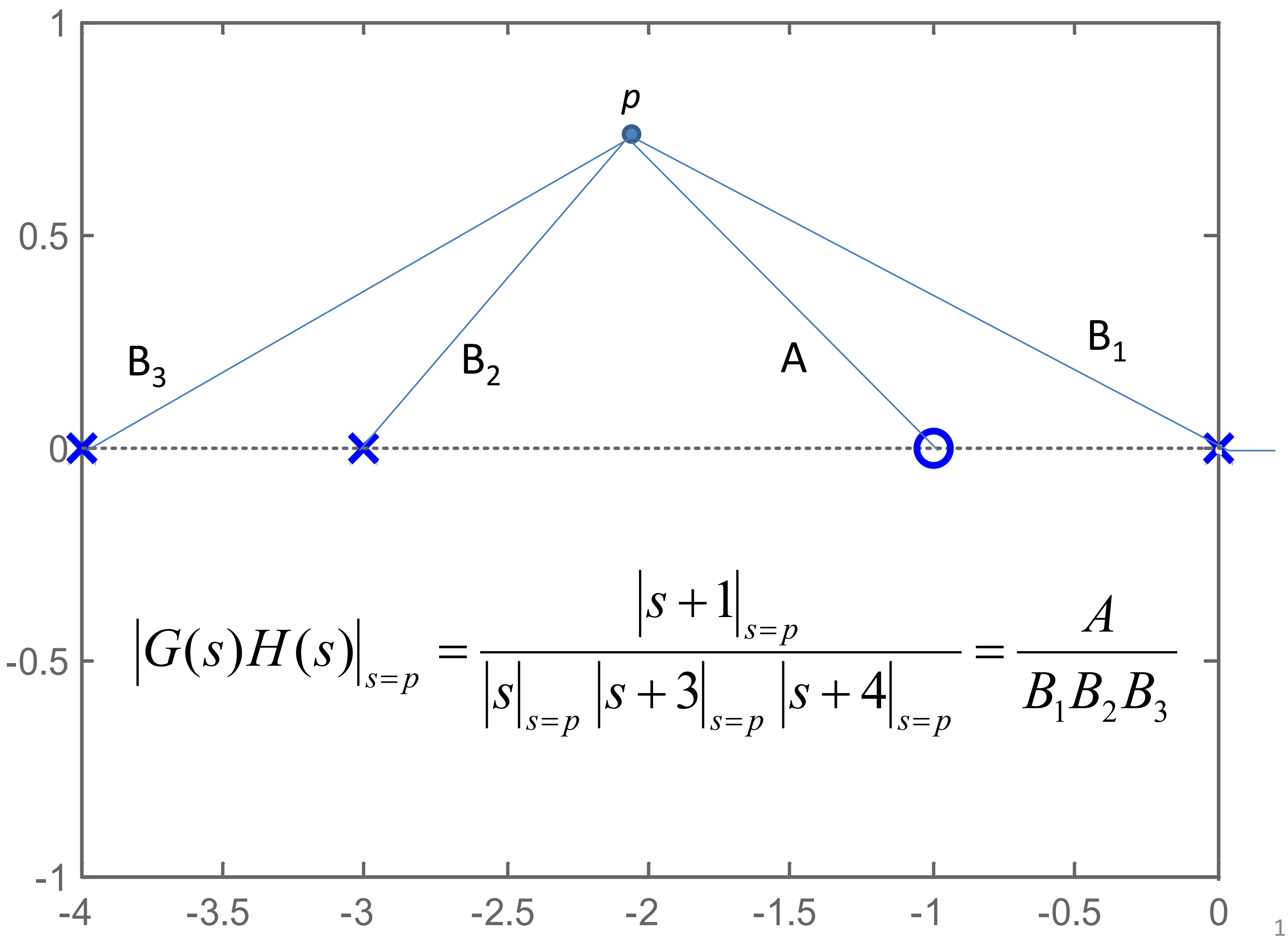
Angle and Magnitude Conditions (Graphically)



$$\angle G(s)H(s)\Big|_{s=p} = \phi_1 - \theta_1 - \theta_2 - \theta_3$$

- If angle of G(s)H(s) at *s=p* is equal to $\pm 180^o(2k+1)$ the point p is on root locus.

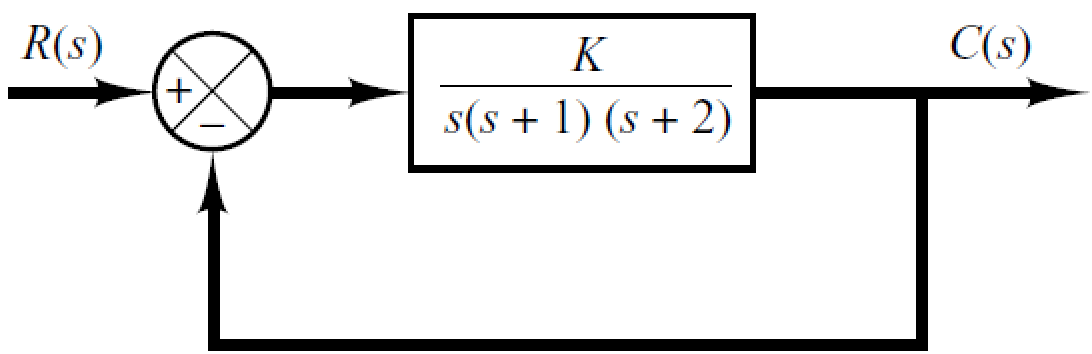
Angle and Magnitude Conditions graphically



$$\left|G(s)H(s)\right|_{s=p} = \frac{\left|s+1\right|_{s=p}}{\left|s\right|_{s=p}\left|s+3\right|_{s=p}\left|s+4\right|_{s=p}} = \frac{A}{B_1B_2B_3}$$

Illustrative Example#1

- Apply angle and magnitude conditions (Analytically as well as graphically) on following unity feedback system.



Illustrative Example#1

- Here $$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$

- For the given system the angle condition becomes

$$\angle G(s)H(s) = \angle \frac{K}{s(s+1)(s+2)}$$

$$\angle G(s)H(s) = \angle K - \angle s - \angle(s+1) - \angle(s+2)$$

$$\angle K - \angle s - \angle(s+1) - \angle(s+2) = \pm 180°(2k+1)$$

Illustrative Example#1

- For example to check whether *s=-0.25* is on the root locus or not we can apply angle condition as follows.

$$\angle G(s)H(s)\big|_{s=-0.25} = \angle K\big|_{s=-0.25} - \angle s\big|_{s=-0.25} - \angle(s+1)\big|_{s=-0.25} - \angle(s+2)\big|_{s=-0.25}$$

$$\angle G(s)H(s)\big|_{s=-0.25} = -\angle(-0.25) - \angle(0.75) - \angle(1.75)$$

$$\angle G(s)H(s)\big|_{s=-0.25} = -180^\circ - 0^\circ - 0^\circ$$

$$\angle G(s)H(s)\big|_{s=-0.25} = \pm 180^\circ(2k+1)$$

Illustrative Example#1

- Here $$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$

- And the Magnitude condition becomes

$$\left|G(s)H(s)\right| = \left|\frac{K}{s(s+1)(s+2)}\right| = 1$$

Illustrative Example#1

- Now we know from angle condition that the point s=-0.25 is on the rot locus. But we do not know the value of gain K at that specific point.

- We can use magnitude condition to determine the value of gain at any point on the root locus.

$$\left| \frac{K}{s(s+1)(s+2)} \right|_{s=-0.25} = 1$$

$$\left| \frac{K}{(-0.25)(-0.25+1)(-0.25+2)} \right|_{s=-0.25} = 1$$

Illustrative Example#1

$$\left|\frac{K}{(-0.25)(-0.25+1)(-0.25+2)}\right|_{s=-0.25} = 1$$

$$\left|\frac{K}{(-0.25)(0.75)(1.75)}\right| = 1$$

$$\left|\frac{K}{-0.3285}\right| = 1$$

$$\frac{K}{0.328} = 1$$

$$K = 0.328$$

Illustrative Example#1

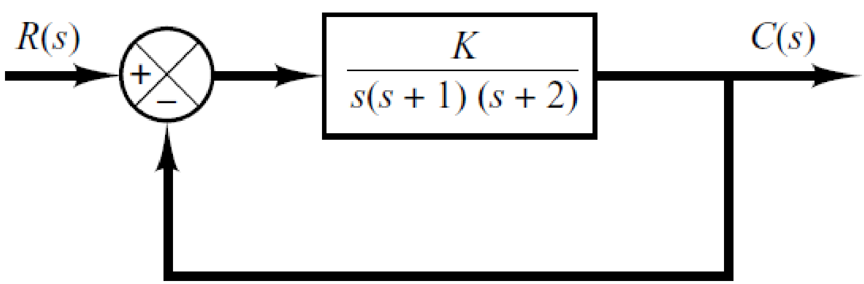
- **Home work:**
  - check whether $s=-0.2+j0.937$ is on the root locus or not (Graphically as well as analytically) ?
  - check whether $s=-1+j2$ is on the root locus or not (Graphically as well as analytically) ?

Illustrative Example#1

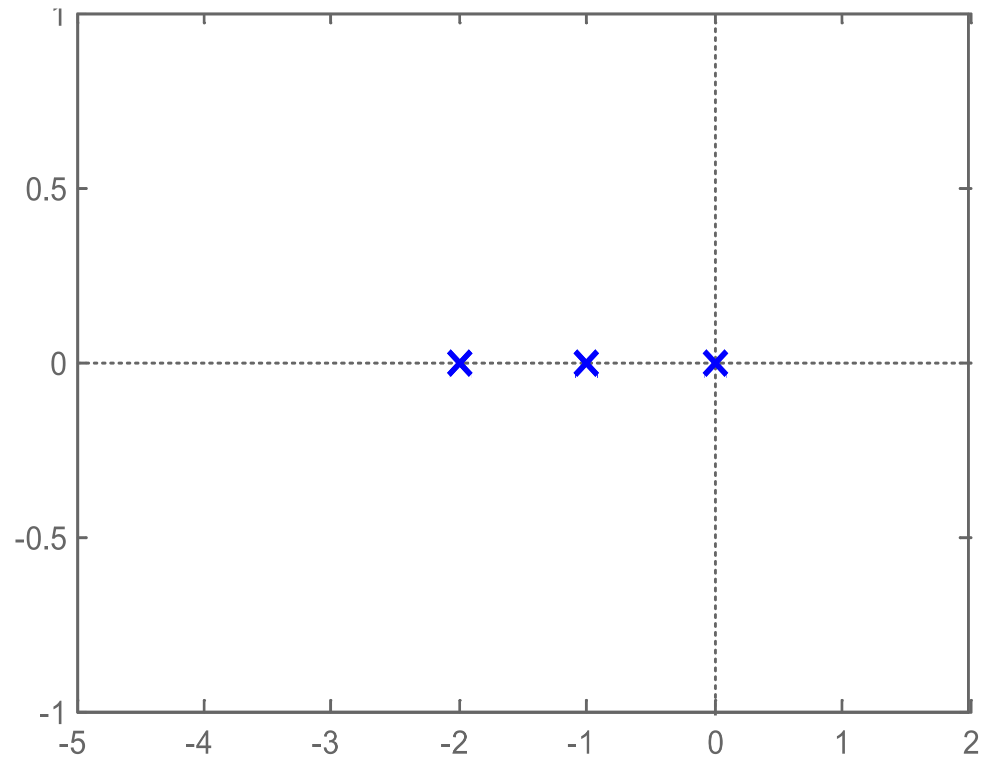
- **Home work:**
  - If $s=-0.2+j0.937$ is on the root locus determine the value of gain K at that point.
  - If $s=-1+j2$ is on the root locus determine the value of gain K at that point.

Construction of root loci

- Step-1: The first step in constructing a root-locus plot is to locate the open-loop poles and zeros in s-plane.



$$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$



Construction of root loci

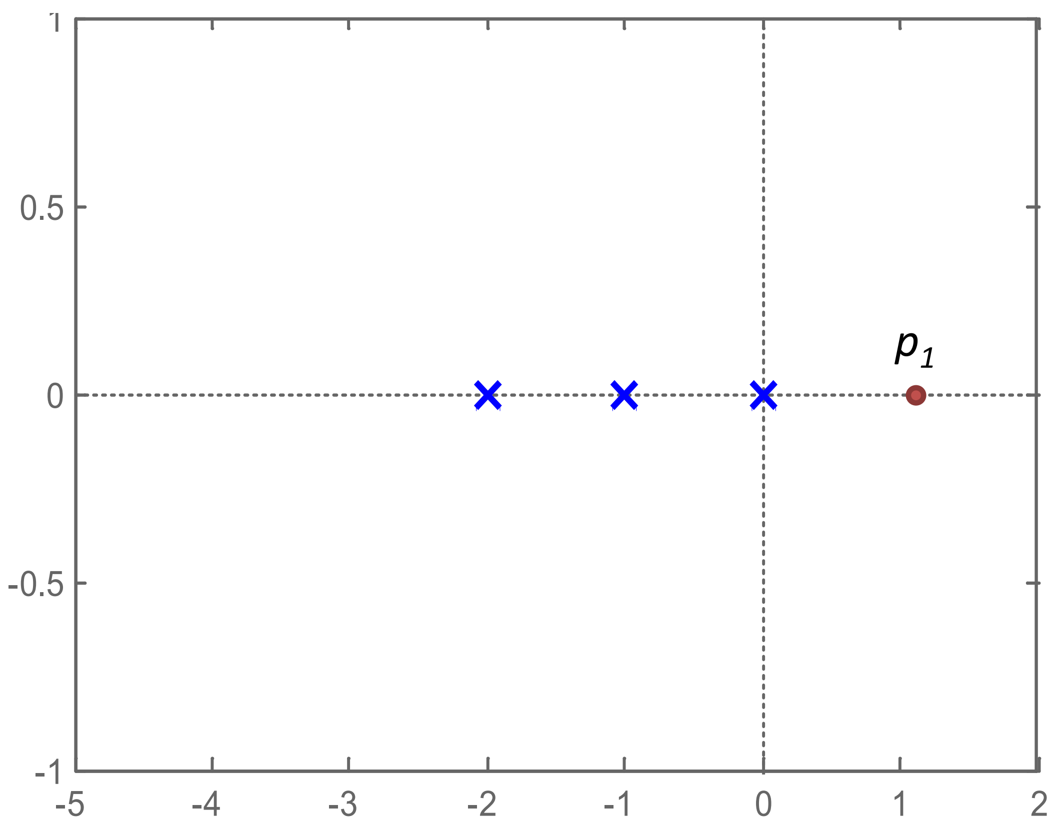
- Step-2: Determine the root loci on the real axis.

- To determine the root loci on real axis we select some test points.
- e.g: p_1 (on positive real axis).

$$\angle s = \angle s+1 = \angle s+2 = 0^\circ$$

- The angle condition is not satisfied.

- Hence, there is no root locus on the positive real axis.



Construction of root loci

- Step-2: Determine the root loci on the real axis.

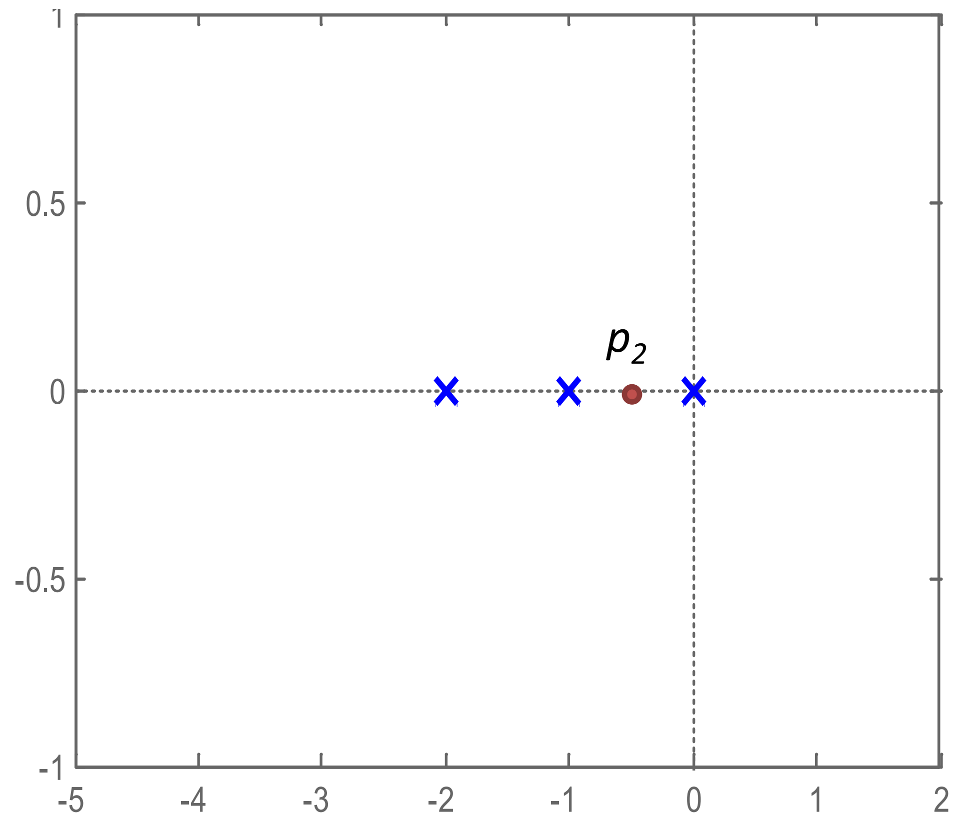
- Next, select a test point on the negative real axis between 0 and –1.
- Then

$$\underline{/s} = 180^\circ, \qquad \underline{/s+1} = \underline{/s+2} = 0^\circ$$

- Thus

$$-\underline{/s} - \underline{/s+1} - \underline{/s+2} = -180^\circ$$

- The angle condition is satisfied. Therefore, the portion of the negative real axis between 0 and –1 forms a portion of the root locus.



Construction of root loci

- Step-2: Determine the root loci on the real axis.

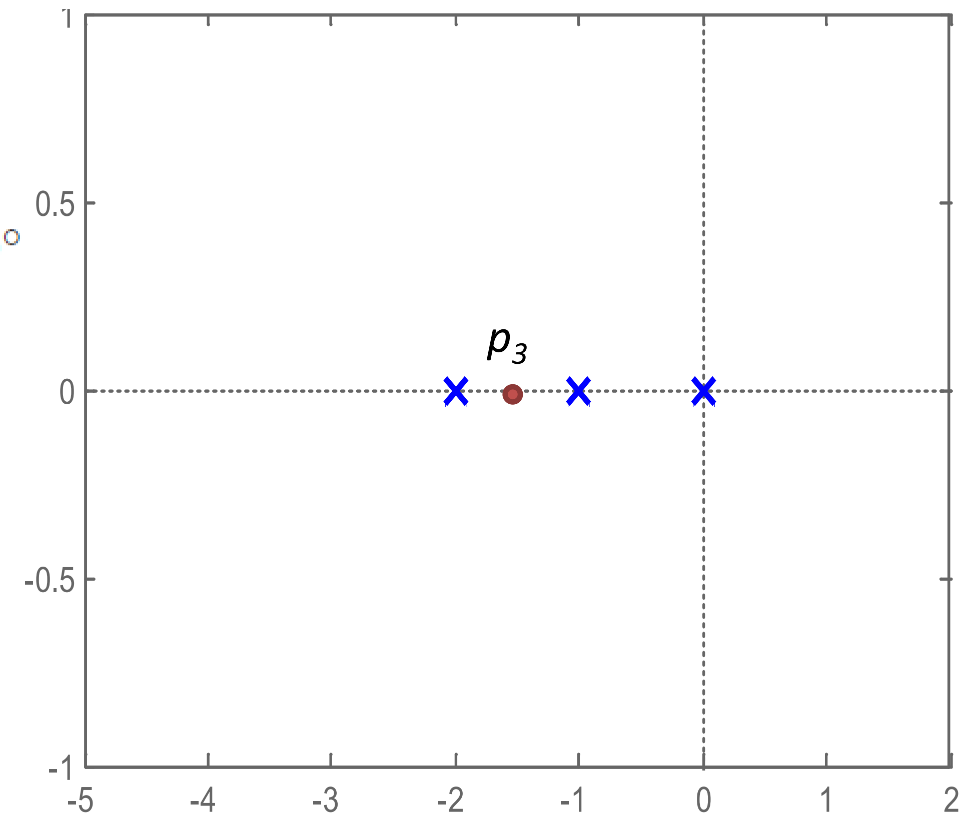
- Now, select a test point on the negative real axis between -1 and –2.
- Then

$$\underline{/s} = \underline{/s+1} = 180^\circ, \qquad \underline{/s+2} = 0^\circ$$

- Thus

$$-\underline{/s} - \underline{/s+1} - \underline{/s+2} = -360^\circ$$

- The angle condition is not satisfied. Therefore, the negative real axis between -1 and –2 is not a part of the root locus.

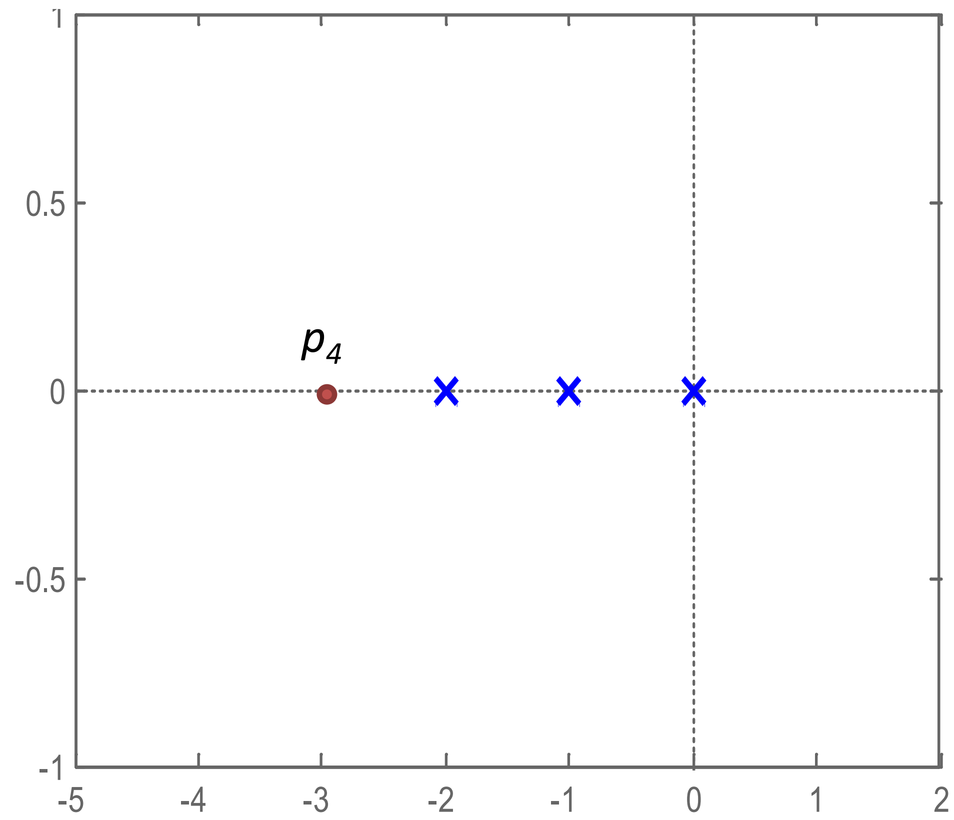


Construction of root loci

- Step-2: Determine the root loci on the real axis.

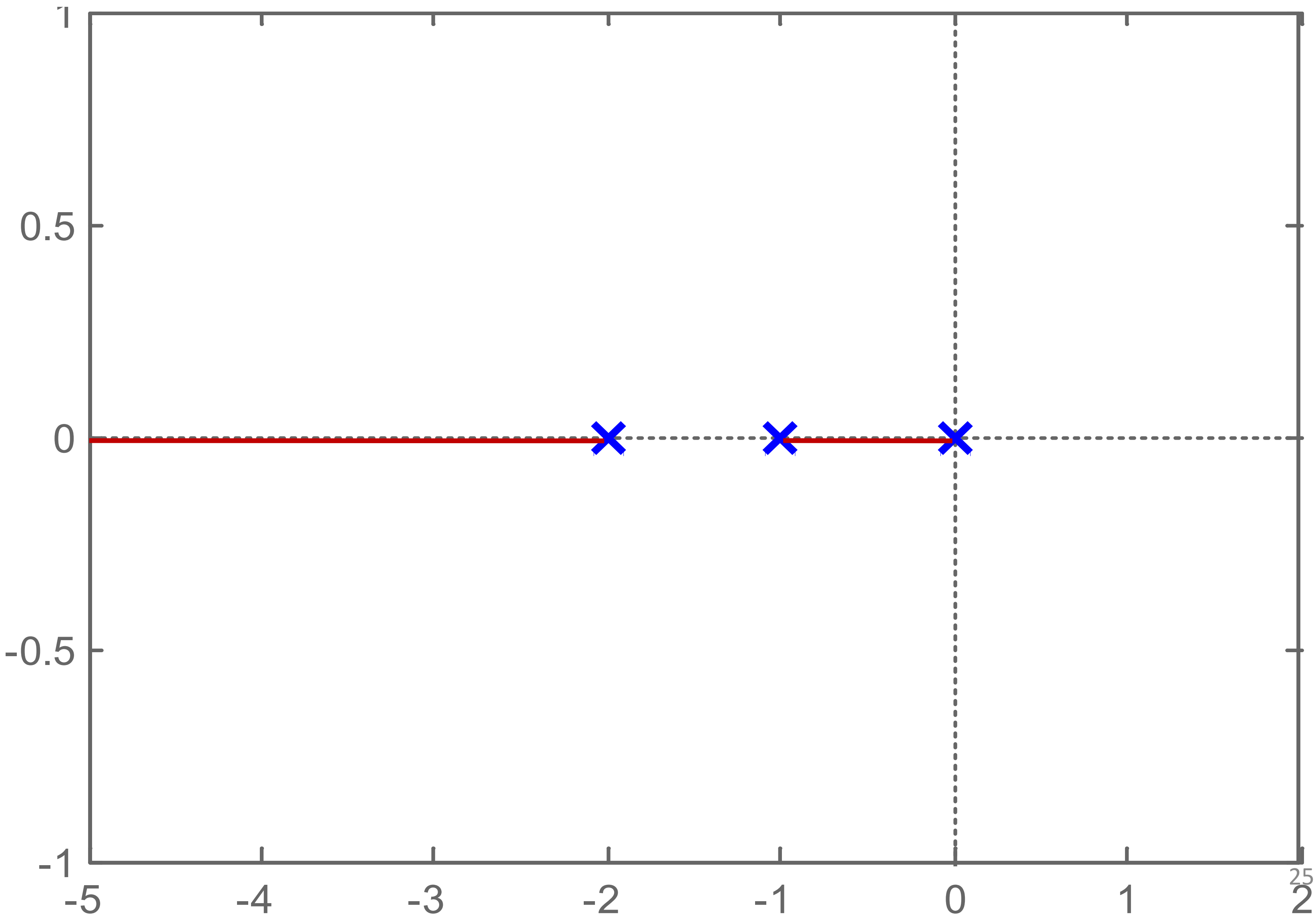
- Similarly, test point on the negative real axis between -3 and $-\infty$ satisfies the angle condition.

- Therefore, the negative real axis between -3 and $-\infty$ is part of the root locus.



Construction of root loci

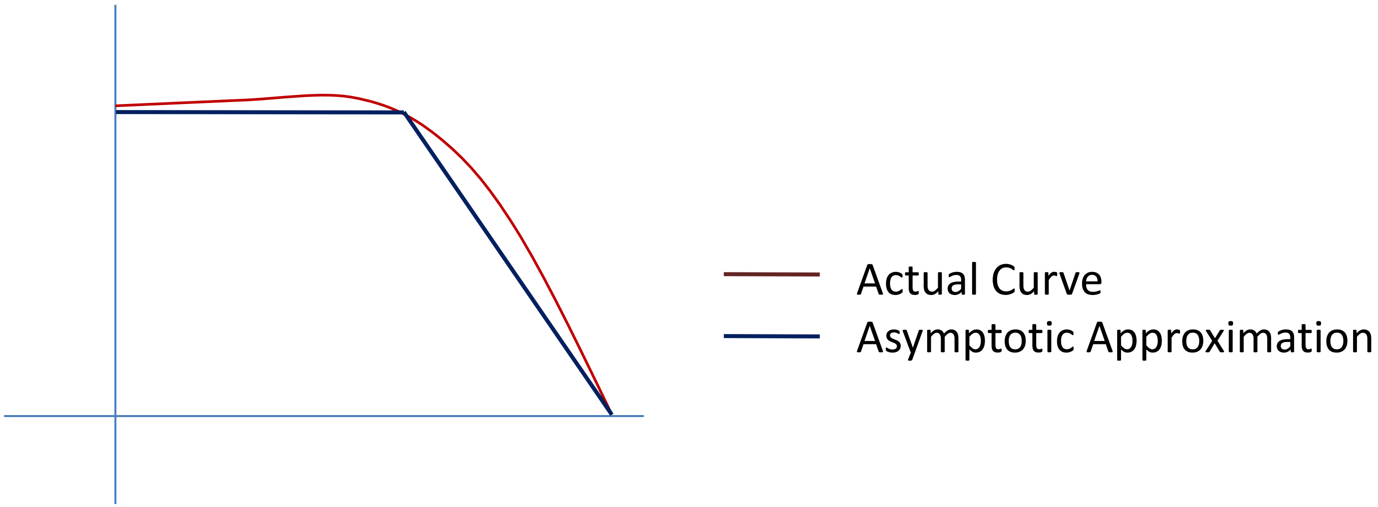
- Step-2: Determine the root loci on the real axis.



Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

Asymptote is the straight line approximation of a curve



Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

$$Angle\ of\ asymptotes = \psi = \frac{\pm 180°(2k+1)}{n-m}$$

- where
- n-----> number of poles
- m-----> number of zeros
- For this Transfer Function $G(s)H(s) = \dfrac{K}{s(s+1)(s+2)}$

$$\psi = \frac{\pm 180°(2k+1)}{3-0}$$

Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

$$
\begin{aligned}
\psi &= \pm 60^\circ && \text{when } k = 0 \\
&= \pm 180^\circ && \text{when } k = 1 \\
&= \pm 300^\circ && \text{when } k = 2 \\
&= \pm 420^\circ && \text{when } k = 3
\end{aligned}
$$

- Since the angle repeats itself as k is varied, the distinct angles for the asymptotes are determined as 60°, –60°, -180°and 180°.
- Thus, there are three asymptotes having angles 60°, –60°, 180°.

Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

- Before we can draw these asymptotes in the complex plane, we must find the point where they intersect the real axis.

- Point of intersection of asymptotes on real axis (or centroid of asymptotes) can be find as out

$$\sigma = \frac{\sum poles - \sum zeros}{n - m}$$

Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

- For $G(s)H(s) = \dfrac{K}{s(s+1)(s+2)}$

$$\sigma = \frac{(0-1-2)-0}{3-0}$$

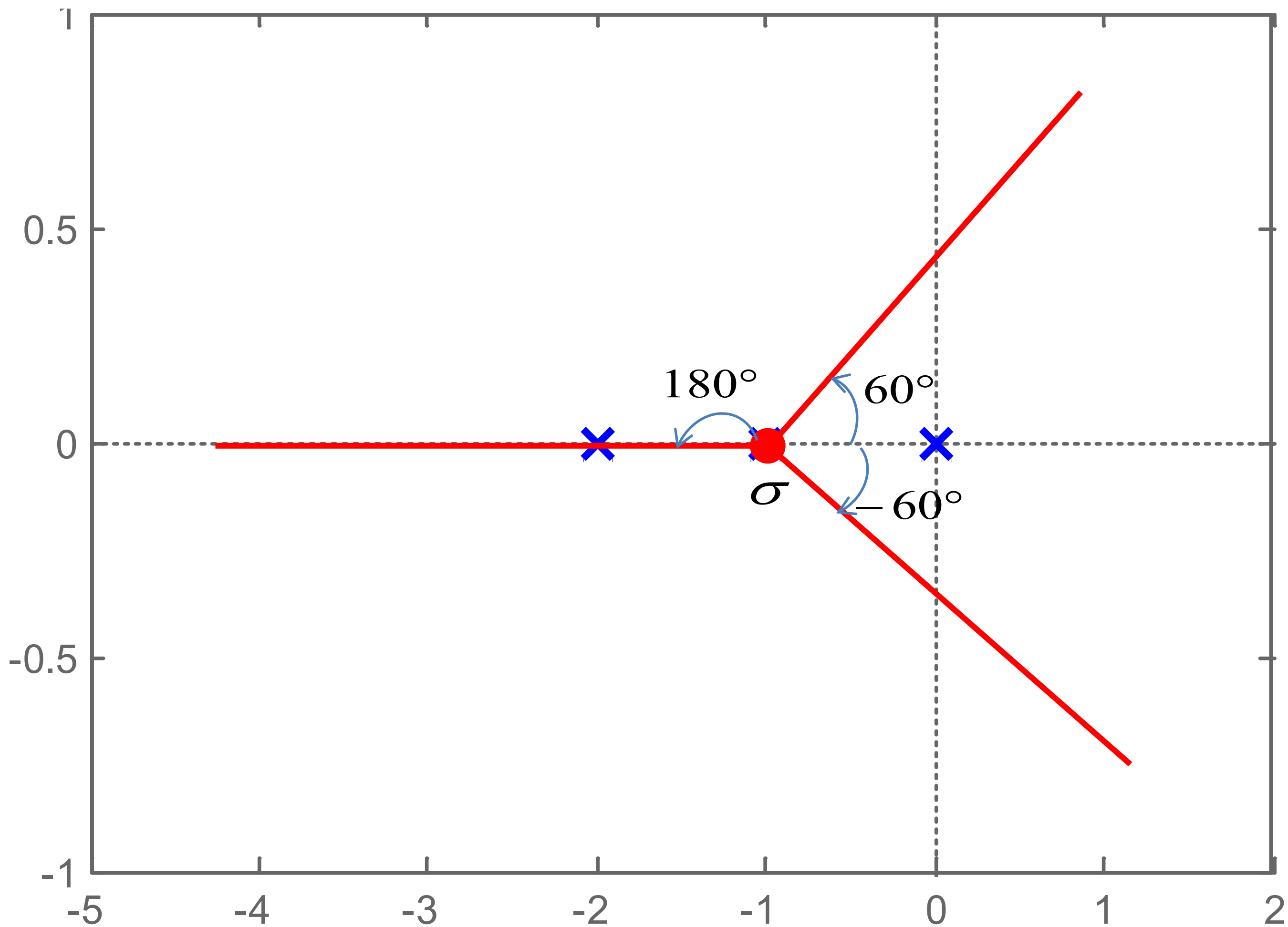
$$\sigma = \frac{-3}{3} = -1$$

Construction of root loci

- Step-3: Determine the *asymptotes* of the root loci.

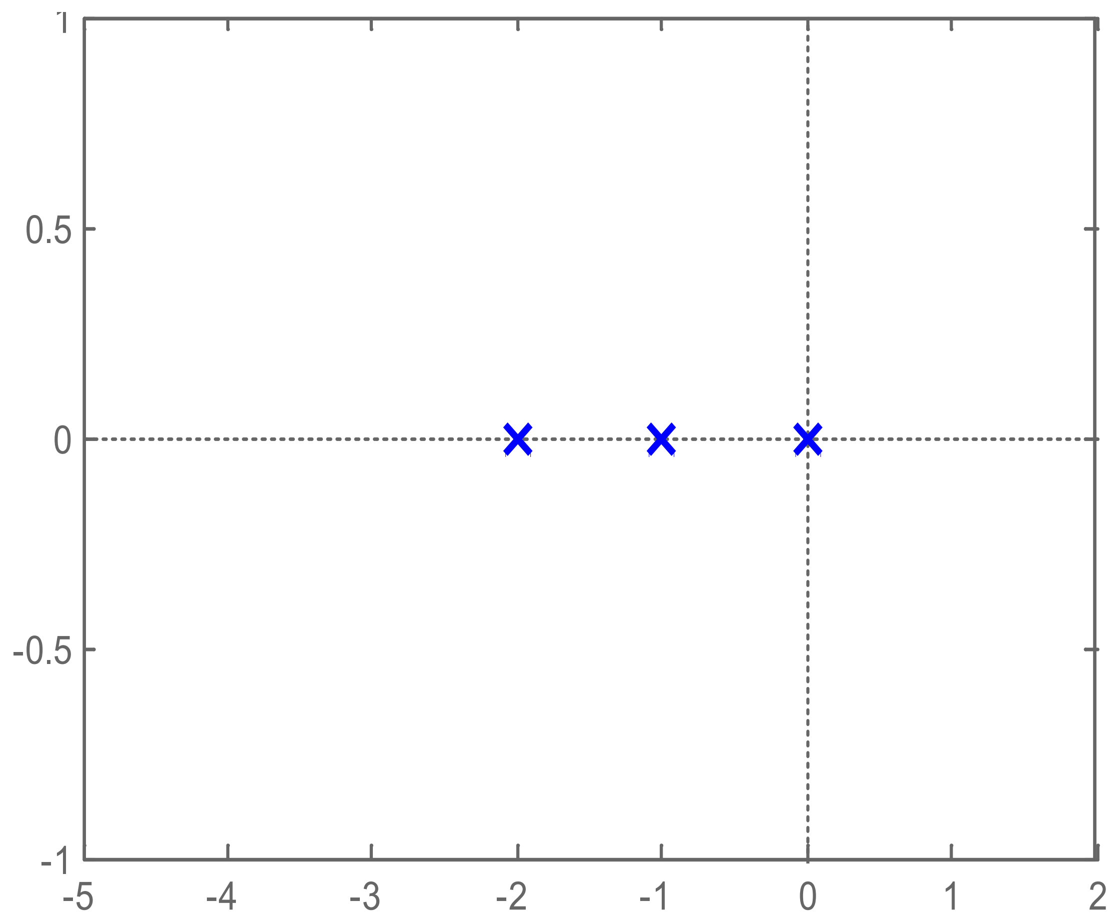
$$\psi = 60^\circ, -60^\circ, 180^\circ$$

$$\sigma = -1$$



Construction of root loci

- Step-4: Determine the *breakaway point*.
  - The breakaway point corresponds to a point in the s plane where multiple roots of the characteristic equation occur.
  - It is the point from which the root locus branches leaves real axis and enter in complex plane.

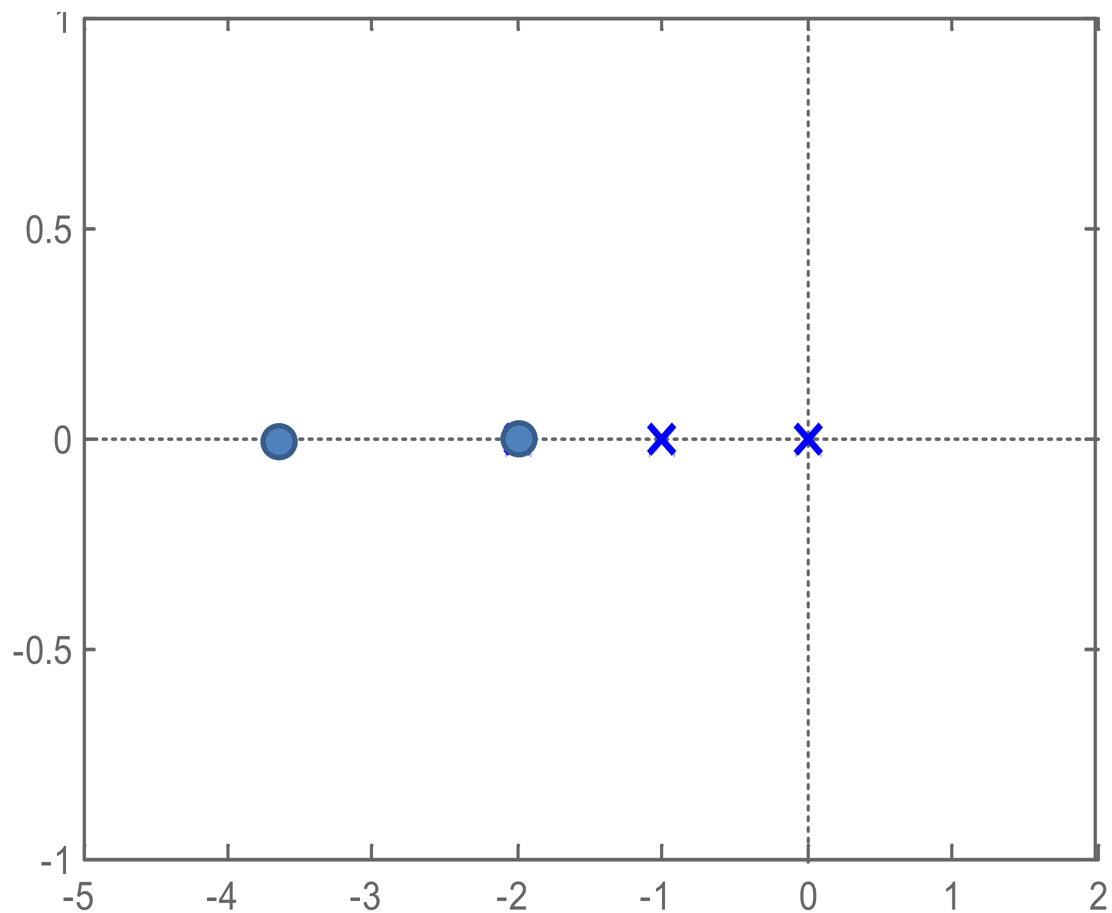


Construction of root loci

- Step-4: Determine the *break-in point*.

- The break-in point corresponds to a point in the s plane where multiple roots of the characteristic equation occur.

- It is the point where the root locus branches arrives at real axis.



Construction of root loci

- Step-4: Determine the *breakaway point* or *break-in point*.

  - The breakaway or break-in points can be determined from the roots of

$$\frac{dK}{ds} = 0$$

  - It should be noted that not all the solutions of dK/ds=0 correspond to actual breakaway points.

  - If a point at which dK/ds=0 is on a root locus, it is an actual breakaway or break-in point.

  - Stated differently, if at a point at which dK/ds=0 the value of K takes a real positive value, then that point is an actual breakaway or break-in point.

Construction of root loci

- Step-4: Determine the *breakaway point* or *break-in point*.

$$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$

- The characteristic equation of the system is

$$1 + G(s)H(s) = 1 + \frac{K}{s(s+1)(s+2)} = 0$$

$$\frac{K}{s(s+1)(s+2)} = -1$$

$$K = -\left[s(s+1)(s+2)\right]$$

- The breakaway point can now be determined as

$$\frac{dK}{ds} = -\frac{d}{ds}\left[s(s+1)(s+2)\right]$$

Construction of root loci

- Step-4: Determine the *breakaway point* or *break-in point*.

$$\frac{dK}{ds} = -\frac{d}{ds}\left[s(s+1)(s+2)\right]$$

$$\frac{dK}{ds} = -\frac{d}{ds}\left[s^3 + 3s^2 + 2s\right]$$

$$\frac{dK}{ds} = -3s^2 - 6s - 2$$

- Set *dK/ds=0* in order to determine breakaway point.

$$-3s^2 - 6s - 2 = 0$$

$$3s^2 + 6s + 2 = 0$$

$$s = -0.4226$$

$$= -1.5774$$

Construction of root loci

- Step-4: Determine the *breakaway point* or *break-in point*.

$$s = -0.4226$$
$$= -1.5774$$

- Since the breakaway point must lie on a root locus between 0 and –1, it is clear that *s=–0.4226* corresponds to the actual breakaway point.

- Point s=–1.5774 is not on the root locus. Hence, this point is not an actual breakaway or break-in point.

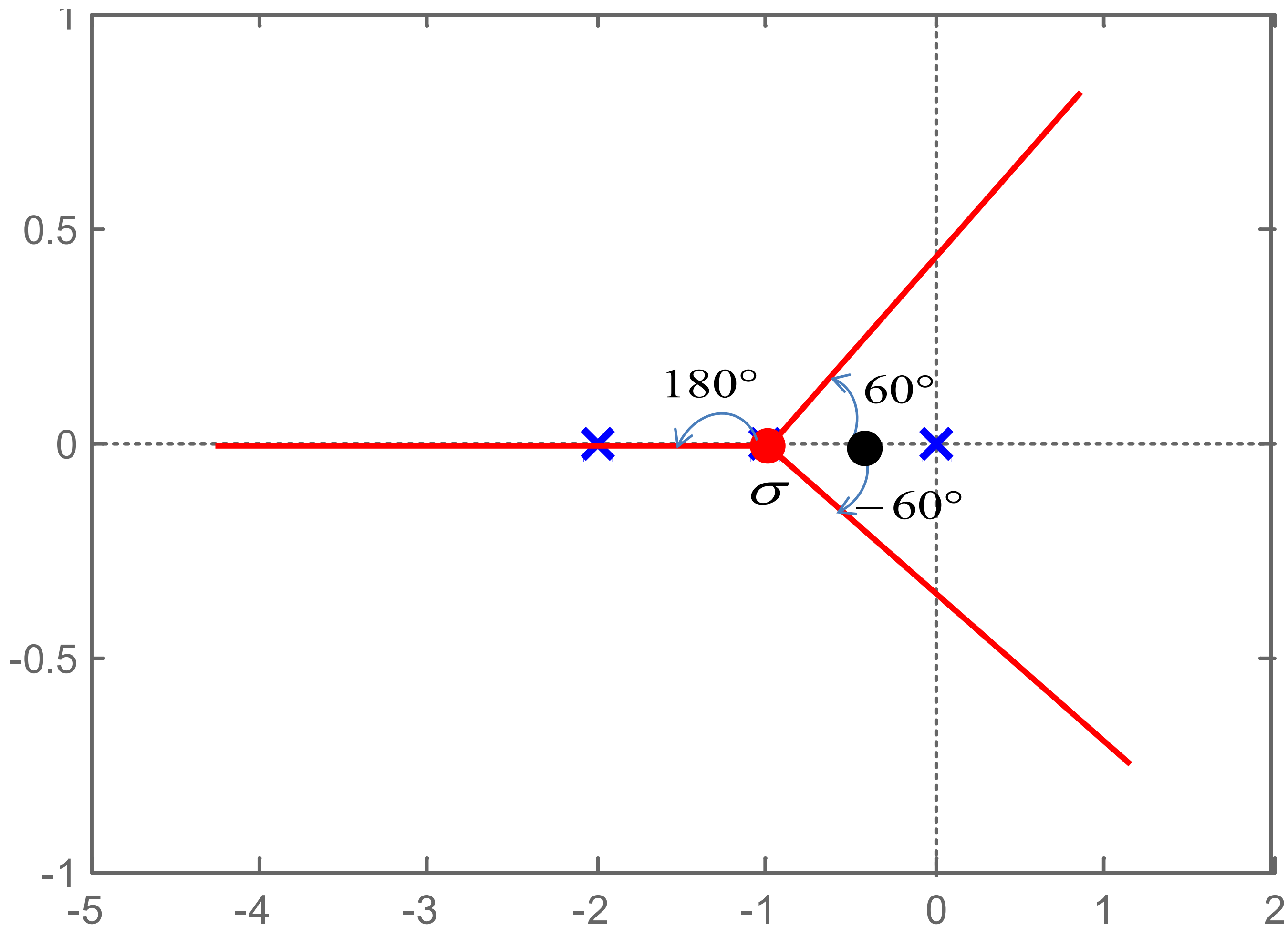
- In fact, evaluation of the values of K corresponding to s=–0.4226 and s=–1.5774 yields

$$K = 0.3849, \qquad \text{for } s = -0.4226$$
$$K = -0.3849, \qquad \text{for } s = -1.5774$$

Construction of root loci

- Step-4: Determine the *breakaway point*.

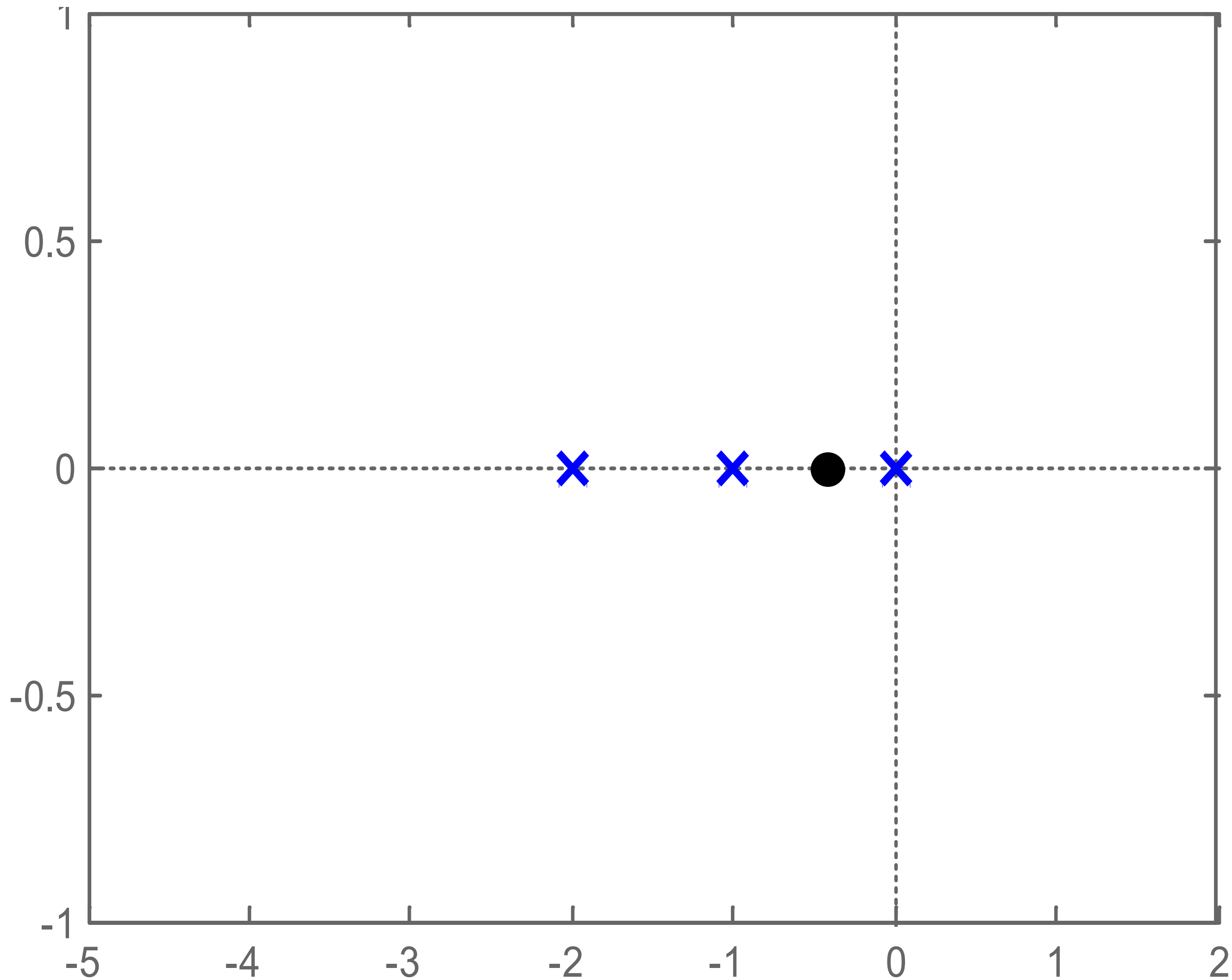
$$s = -0.4226$$



Construction of root loci

- Step-4: Determine the *breakaway point*.

$$s = -0.4226$$



Home Work

- Determine the Breakaway and break in points

$$KG(s)H(s) = \frac{K(s-3)(s-5)}{(s+1)(s+2)}$$

Solution

$$KG(s)H(s) = \frac{K(s-3)(s-5)}{(s+1)(s+2)} = \frac{K(s^2-8s+15)}{(s^2+3s+2)}$$

$$\frac{K(s^2-8s+15)}{s^2+3s+2} = -1$$

$$K = -\frac{(s^2+3s+2)}{(s^2-8s+15)}$$

- Differentiating ***K*** with respect to **s** and setting the derivative equal to zero yields;

$$\frac{dK}{ds} = -\frac{[(s^2-8s+15)(2s+3)-(s^2+3s+2)(2s-8)]}{(s^2-8s+15)^2} = 0$$

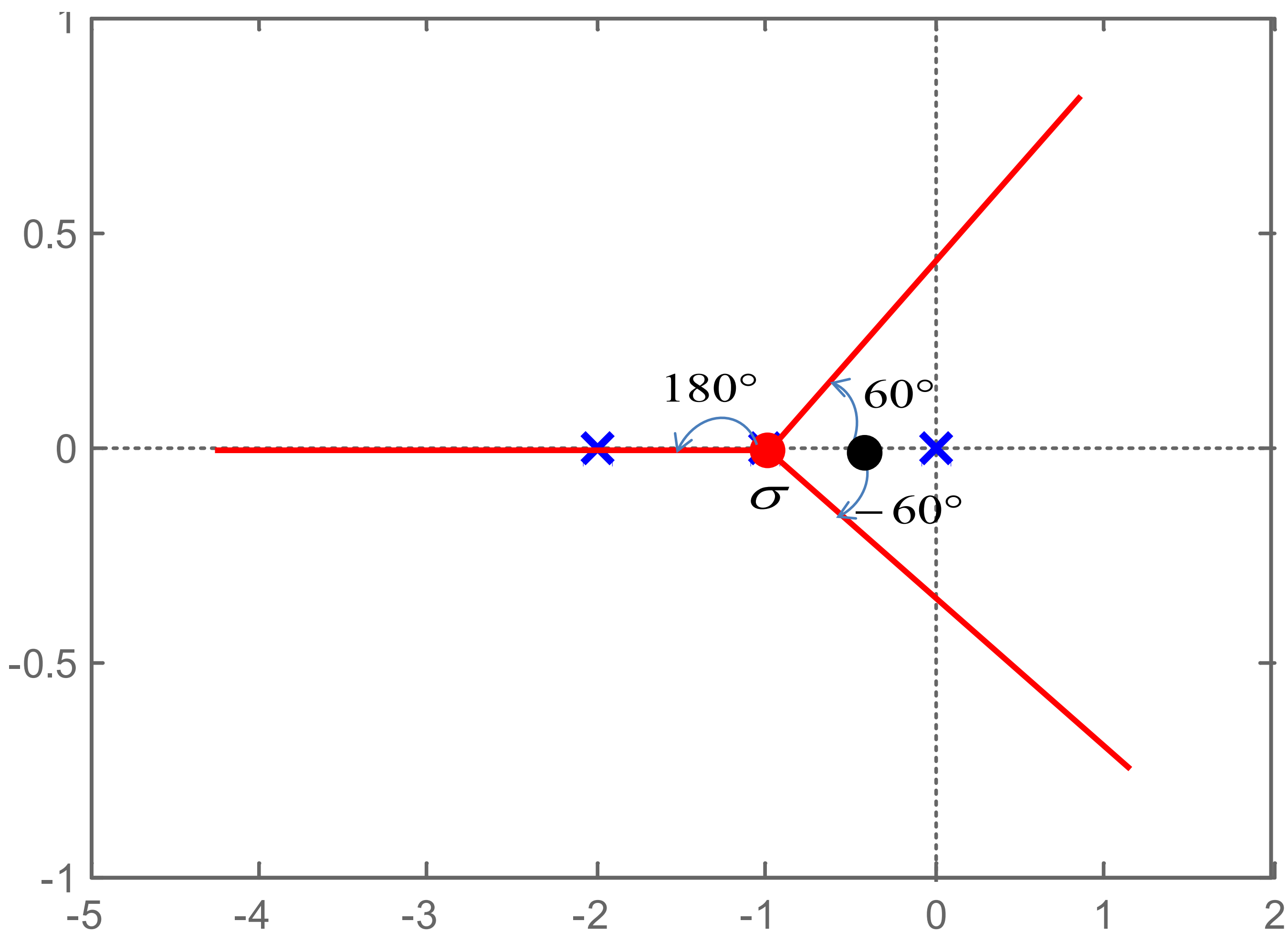
$$11s^2 - 26s - 61 = 0$$

Hence, solving for s, we find the break-away and break-in points;

$\boxed{s = -1.45 \text{ and } 3.82}$

Construction of root loci

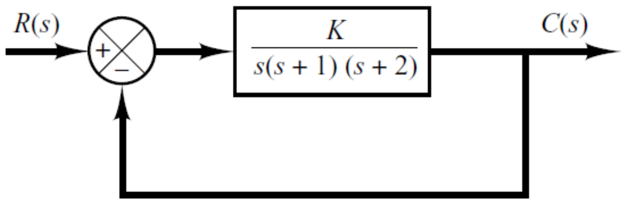
- Step-5: Determine the points where root loci cross the imaginary axis.



Construction of root loci

- Step-5: Determine the points where root loci cross the imaginary axis.
  - These points can be found by use of Routh's stability criterion.
  - Since the characteristic equation for the present system is

$$s^3 + 3s^2 + 2s + K = 0$$



  - The Routh Array Becomes

$$\begin{array}{lcc} s^3 & 1 & 2 \\ s^2 & 3 & K \\ s^1 & \dfrac{6-K}{3} & \\ s^0 & K & \end{array}$$

Construction of root loci

- Step-5: Determine the points where root loci cross the imaginary axis.

- The value(s) of K that makes the system marginally stable is 6.

- The crossing points on the imaginary axis can then be found by solving the auxiliary equation obtained from the s^2 row, that is,

$$3s^2 + K = 3s^2 + 6 = 0$$

- Which yields

$$s = \pm j\sqrt{2}$$

s^3	1	2
s^2	3	K
s^1	$\dfrac{6-K}{3}$	
s^0	K	

Construction of root loci

- Step-5: Determine the points where root loci cross the imaginary axis.

- An alternative approach is to let s=jω in the characteristic equation, equate both the real part and the imaginary part to zero, and then solve for ω and K.

- For present system the characteristic equation is

$$s^3 + 3s^2 + 2s + K = 0$$

$$(j\omega)^3 + 3(j\omega)^2 + 2j\omega + K = 0$$

$$(K - 3\omega^2) + j(2\omega - \omega^3) = 0$$

Construction of root loci

- Step-5: Determine the points where root loci cross the imaginary axis.

$$(K-3\omega^2)+j(2\omega-\omega^3)=0$$

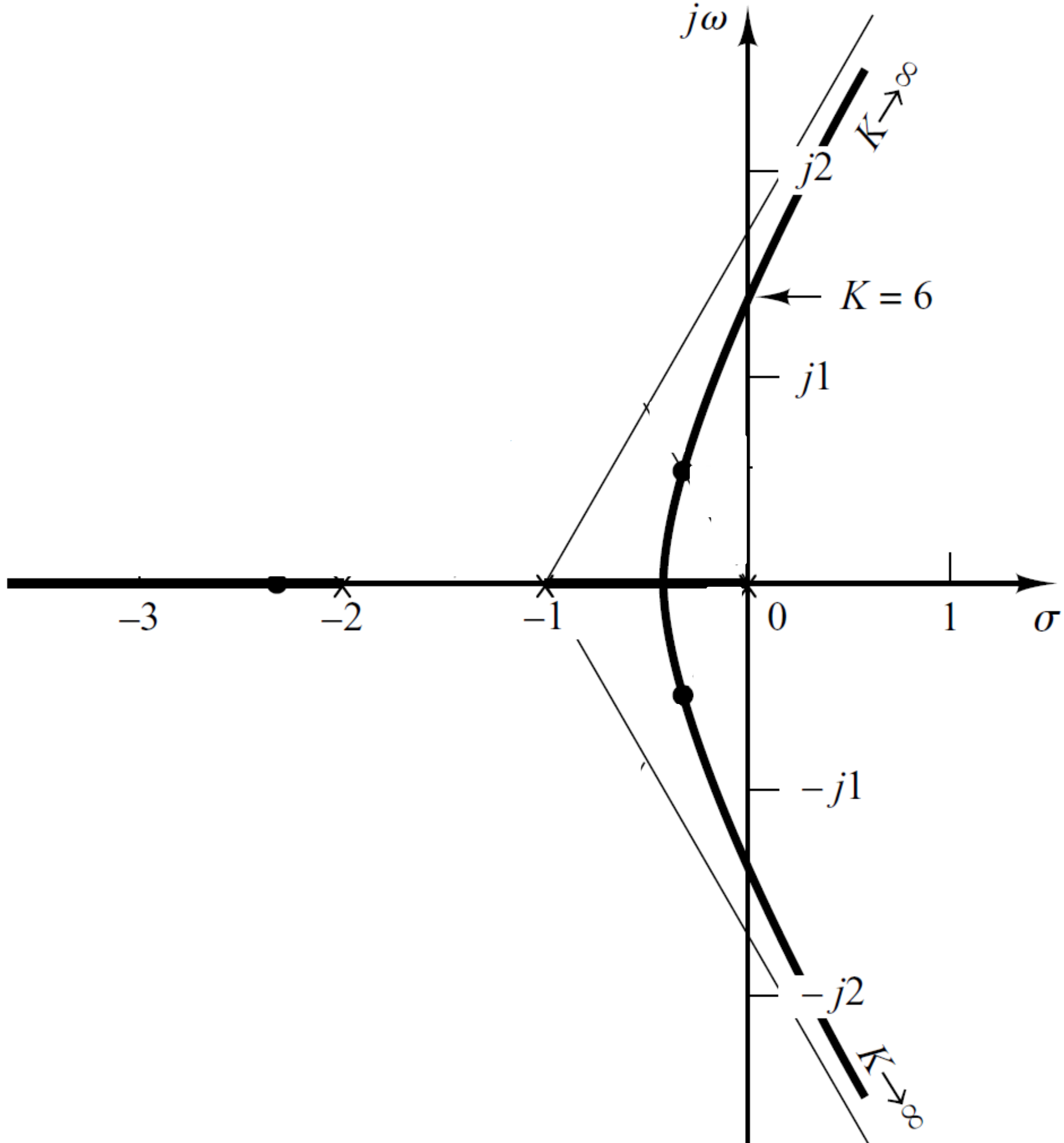
- Equating both real and imaginary parts of this equation to zero

$$(2\omega-\omega^3)=0$$

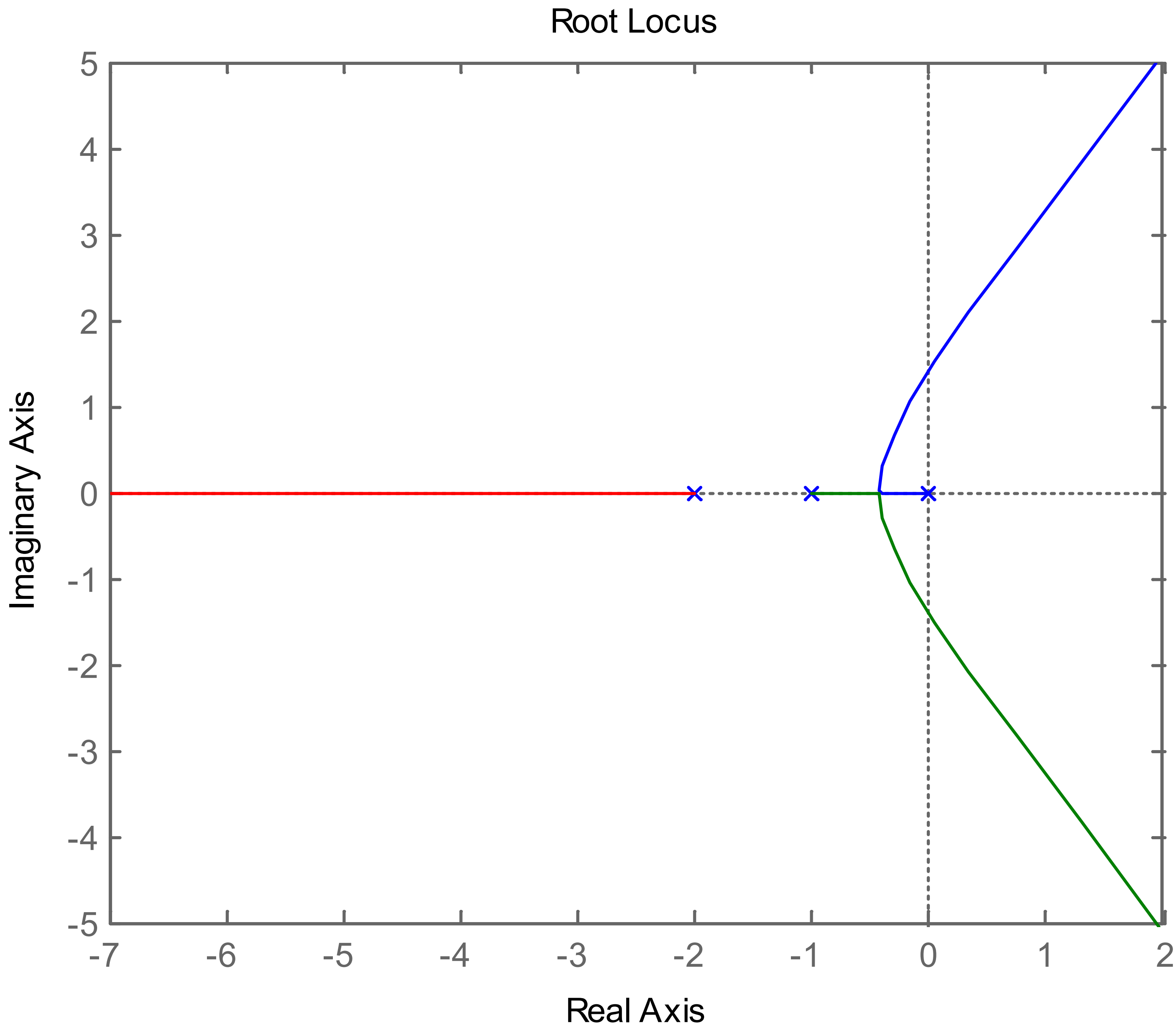
$$(K-3\omega^2)=0$$

- Which yields

$$\omega=\pm\sqrt{2}, \qquad K=6 \qquad \text{or} \qquad \omega=0, \qquad K=0$$

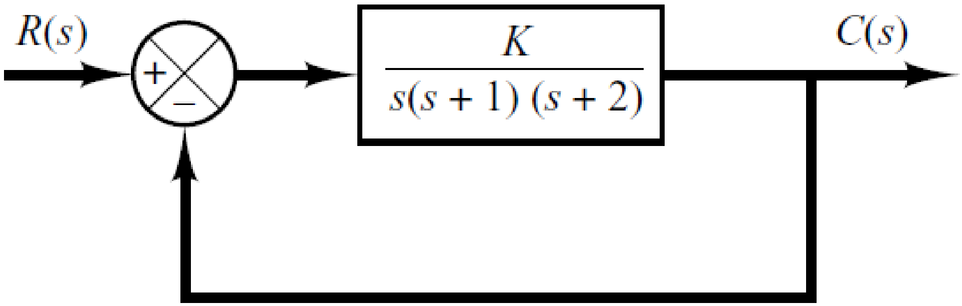


$j\omega$
$K \to \infty$
$j2$
$K = 6$
$j1$
-3
-2
-1
0
1
σ
$-j1$
$-j2$
$K \to \infty$


Root Locus
Imaginary Axis
Real Axis
5
4
3
2
1
0
-1
-2
-3
-4
-5
-7
-6
-5
-4
-3
-2
-1
0
1
2

Example#1

- Consider following unity feedback system.



- Determine the value of K such that the damping ratio of a pair of dominant complex-conjugate closed-loop poles is 0.5.

$$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$

Example#1

- The damping ratio of 0.5 corresponds to

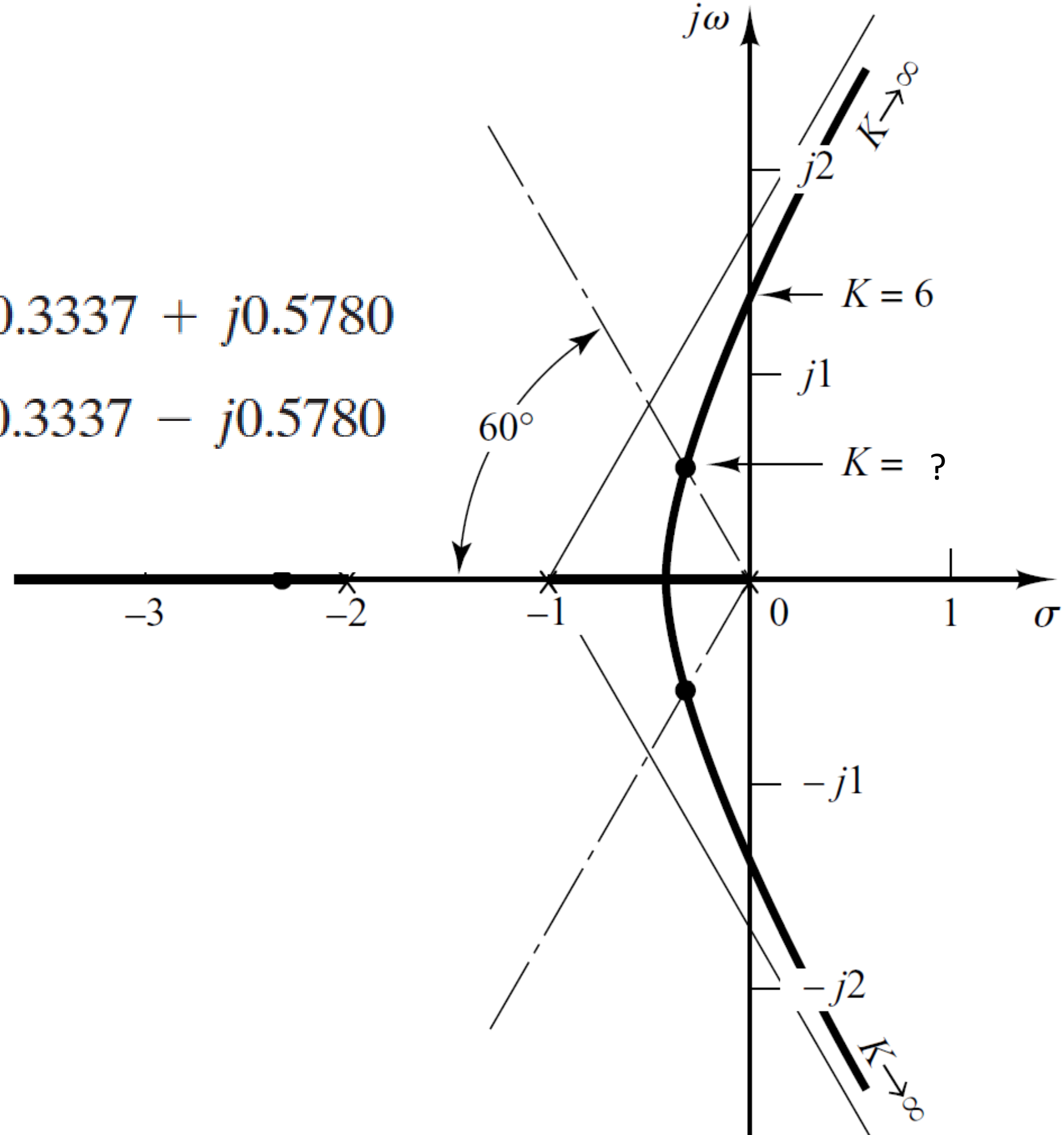
$$\zeta = \cos\theta$$

$$\theta = \cos^{-1}\zeta$$

$$\theta = \cos^{-1}(0.5) = 60^\circ$$

$$s_1 = -0.3337 + j0.5780$$

$$s_2 = -0.3337 - j0.5780$$

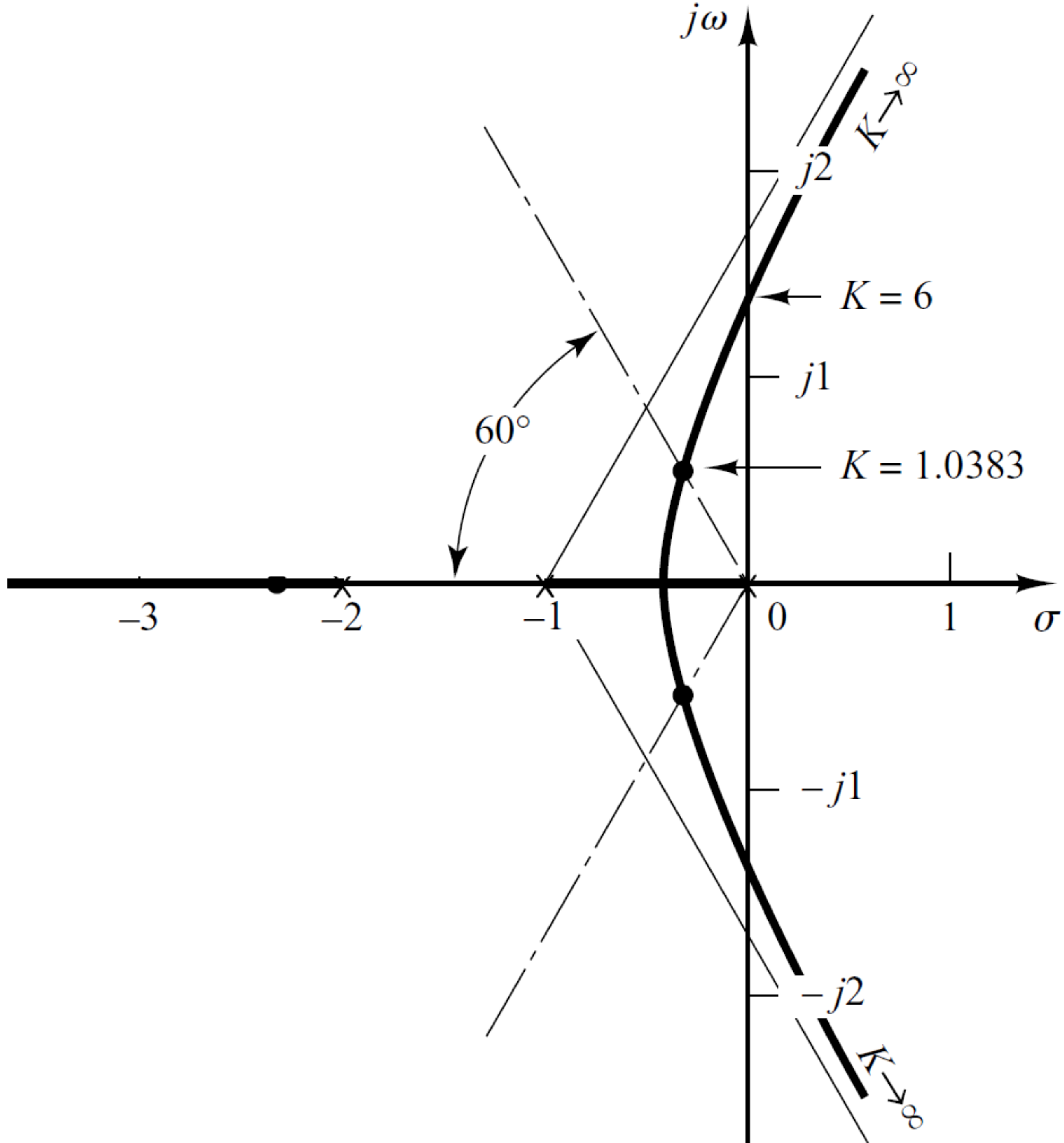


Example#1

- The value of K that yields such poles is found from the magnitude condition

$$\left|\frac{K}{s(s+1)(s+2)}\right|_{s=-0.3337+j0.5780} = 1$$

$$\begin{aligned} K &= |s(s+1)(s+2)|_{s=-0.3337+j0.5780} \\ &= 1.0383 \end{aligned}$$



$j\omega$
$K \to \infty$
$j2$
$K = 6$
$j1$
60°
$K = 1.0383$
–3
–2
–1
0
1
σ
$-j1$
$-j2$
$K \to \infty$

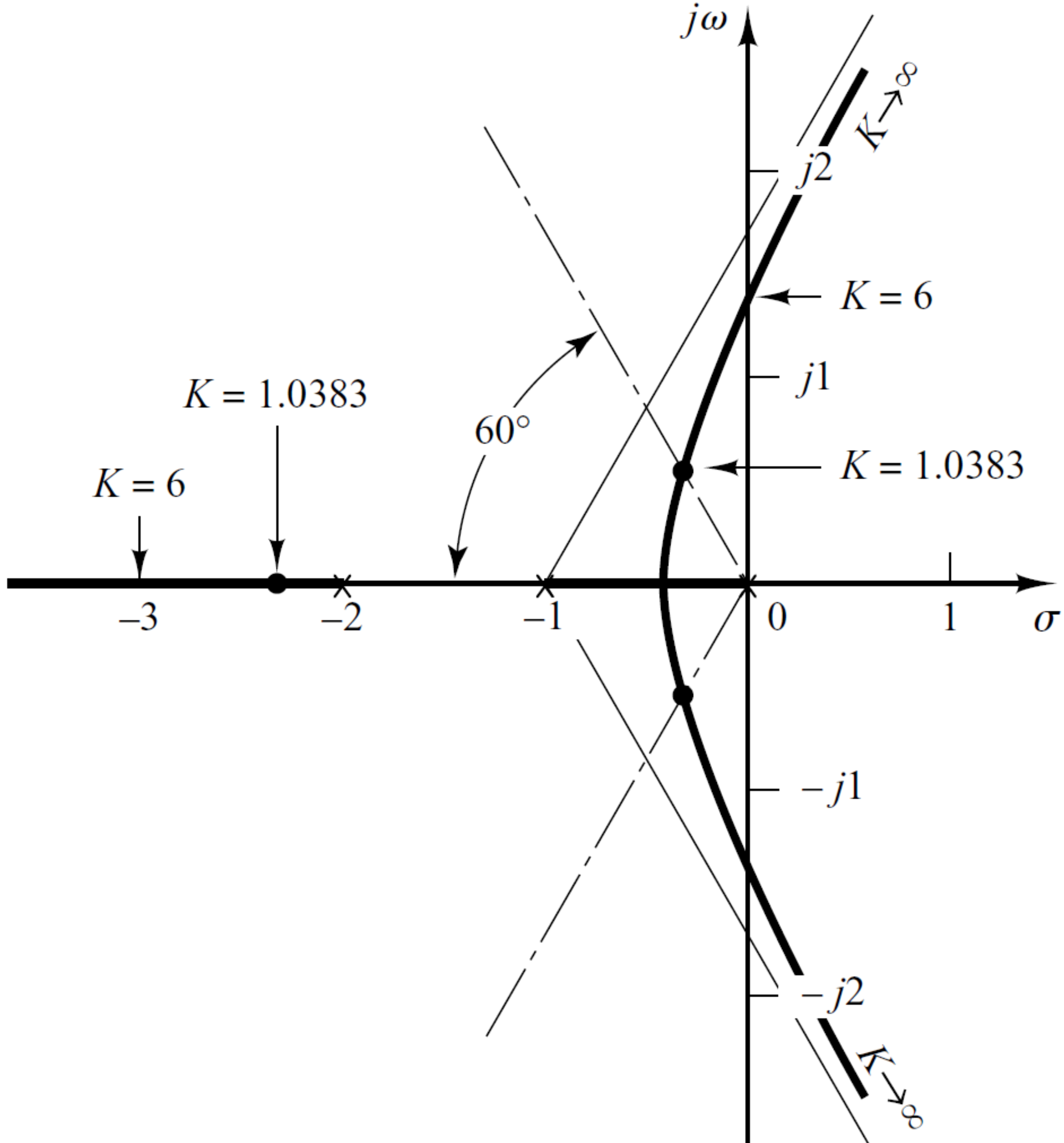
Example#1

- The third closed loop pole at K=1.0383 can be obtained as

$$1+G(s)H(s)=1+\frac{K}{s(s+1)(s+2)}=0$$

$$1+\frac{1.0383}{s(s+1)(s+2)}=0$$

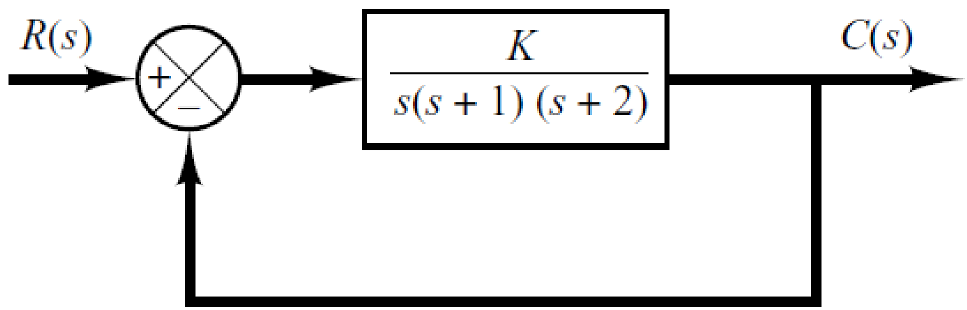
$$s(s+1)(s+2)+1.0383=0$$



jω
K→∞
j2
K = 6
j1
K = 1.0383
60°
K = 1.0383
K = 6
–3
–2
–1
0
1
σ
–j1
–j2
K→∞

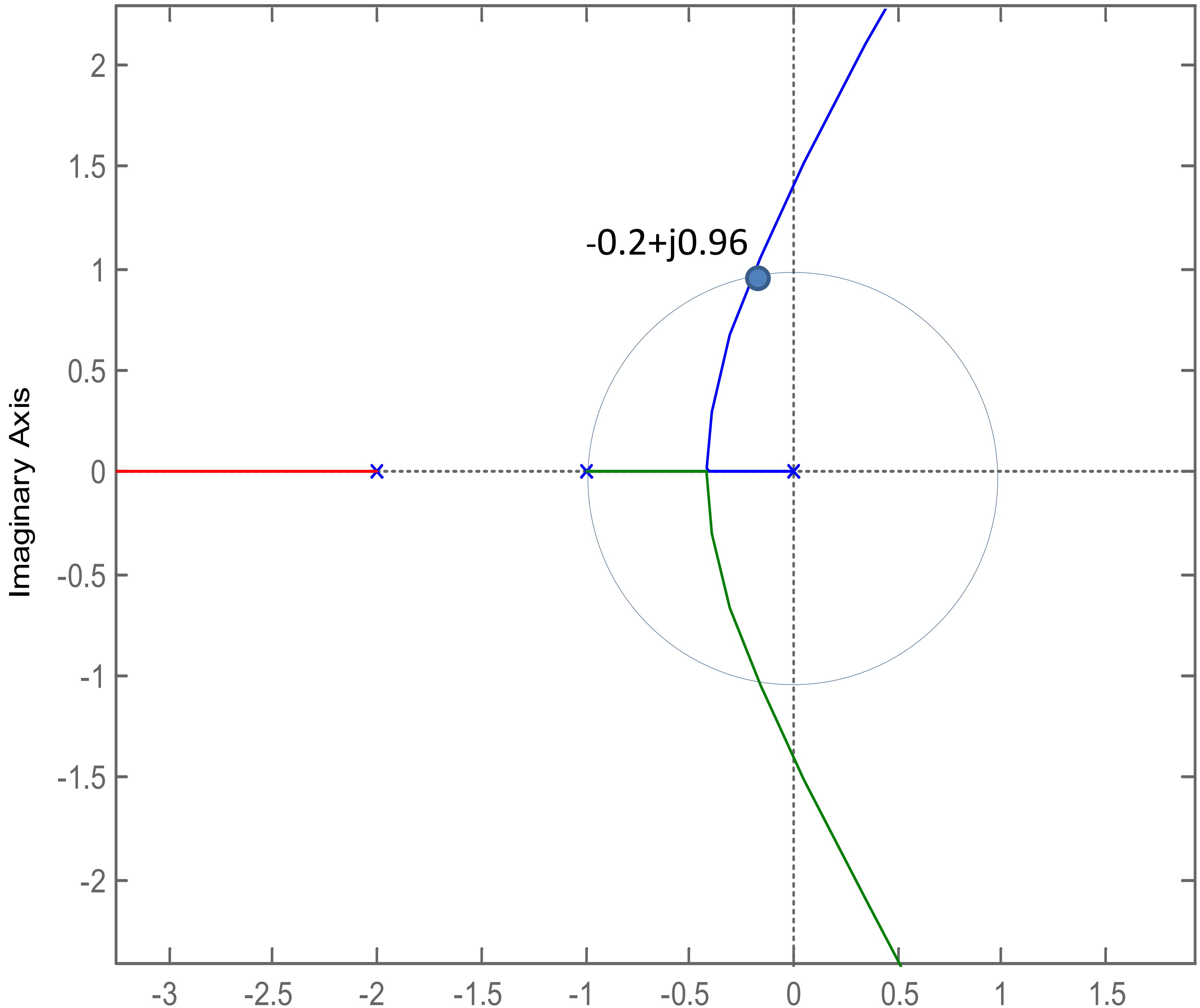
Home Work

- Consider following unity feedback system.



- Determine the value of K such that the natural undamped frequency of dominant complex-conjugate closed-loop poles is 1 rad/sec.

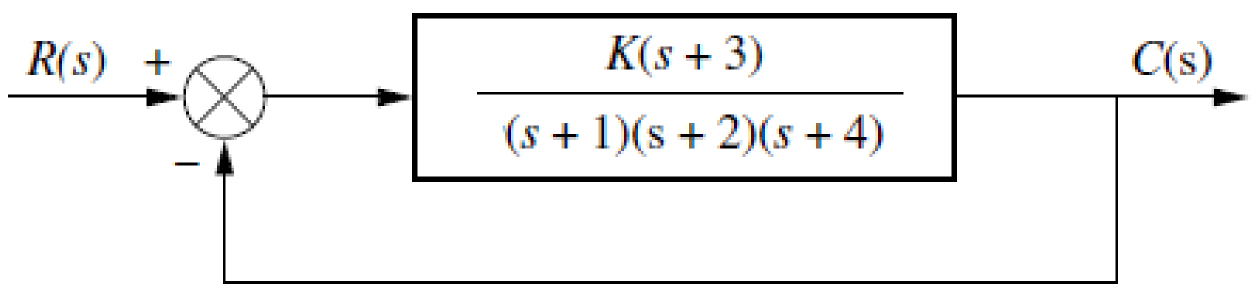
$$G(s)H(s) = \frac{K}{s(s+1)(s+2)}$$



-0.2+j0.96
Imaginary Axis
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5

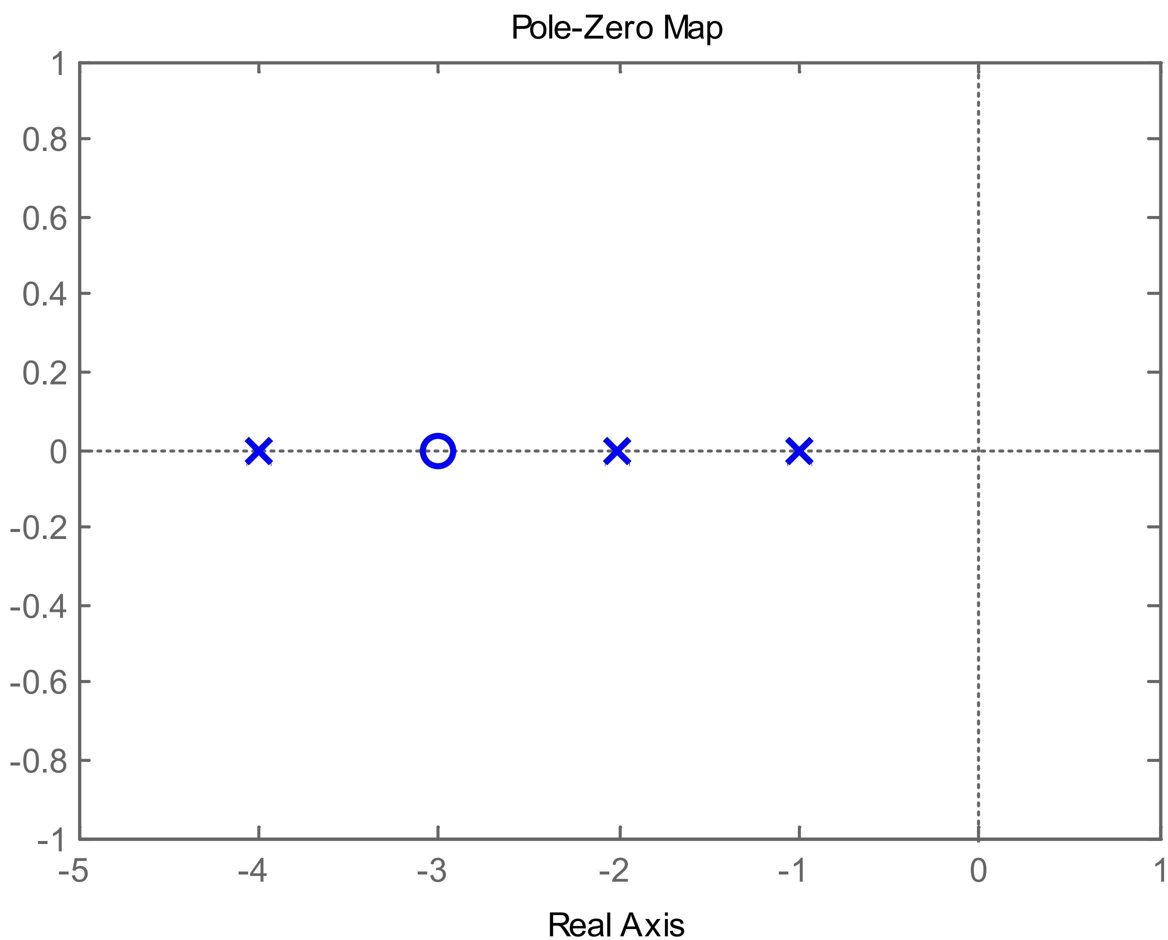
Example#2

- Sketch the root locus of following system and determine the location of dominant closed loop poles to yield maximum overshoot in the step response less than 30%.



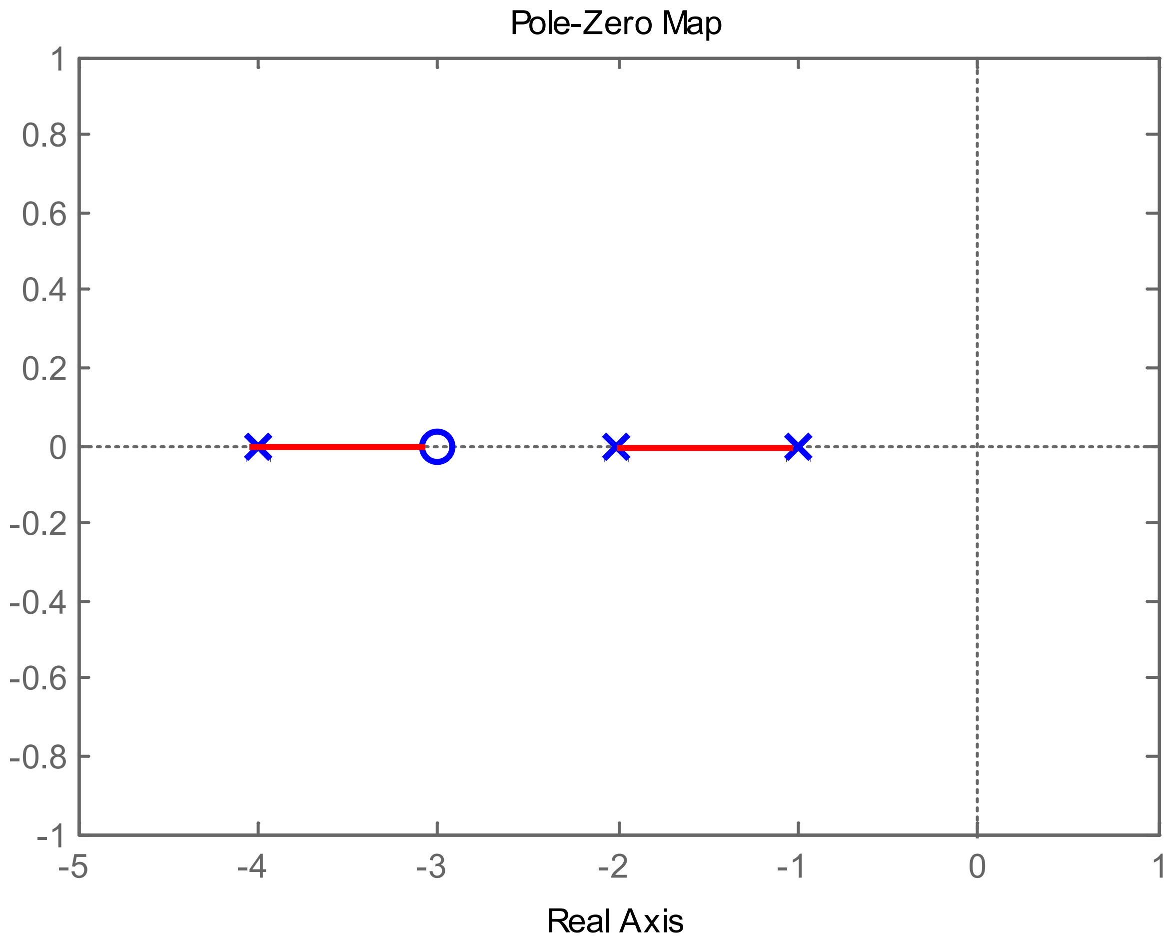
Example#2

- Step-1: Pole-Zero Map



Example#2

- Step-2: Root Loci on Real axis

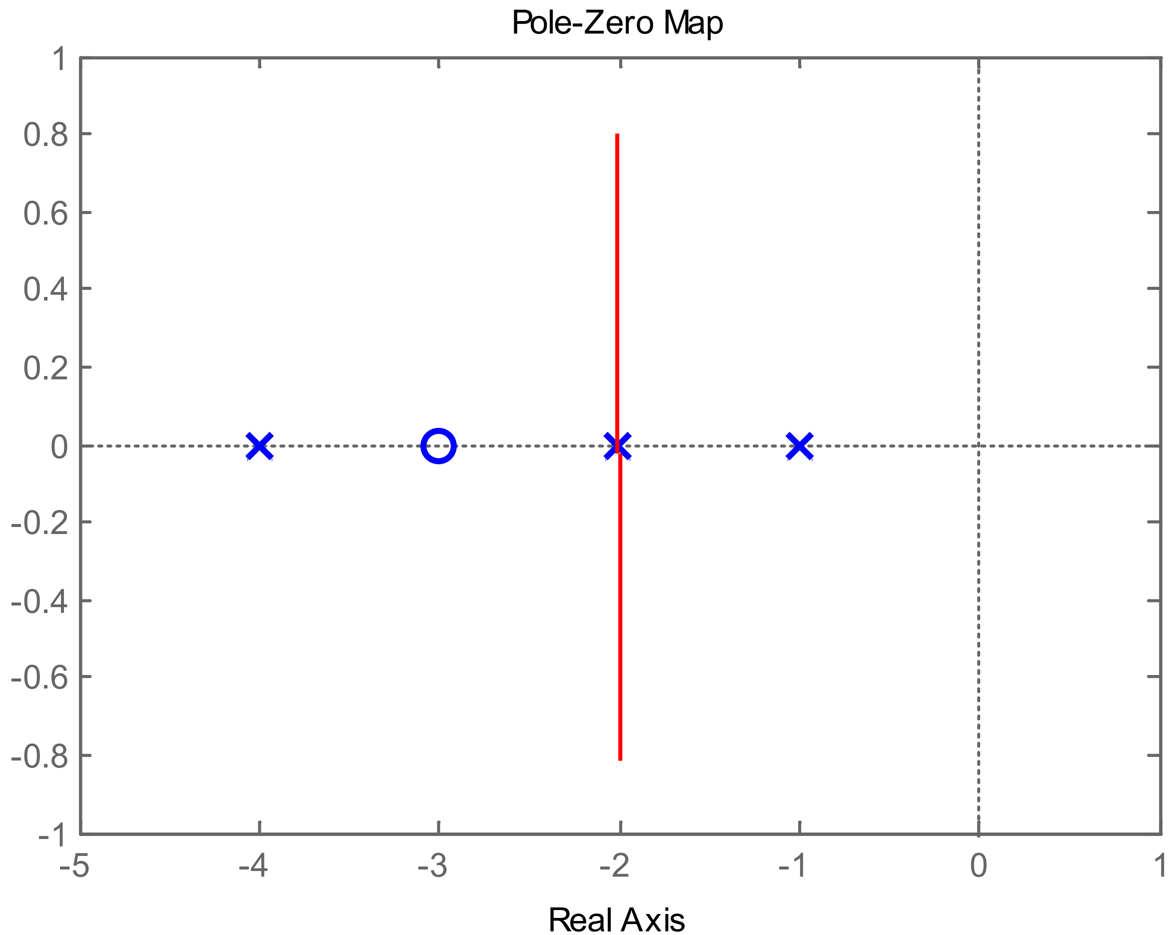


Example#2

- Step-3: Asymptotes

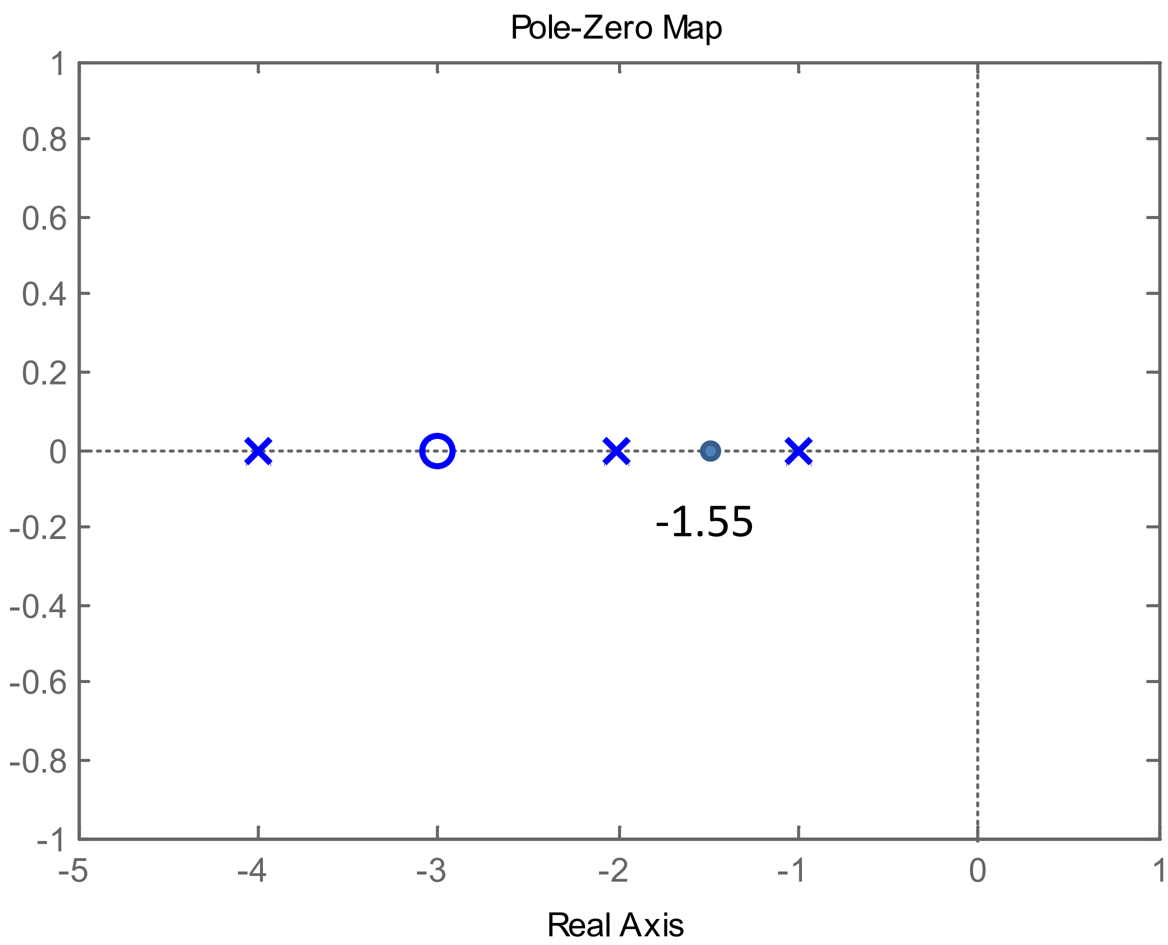
$$\psi = \pm 90^\circ$$

$$\sigma = -2$$

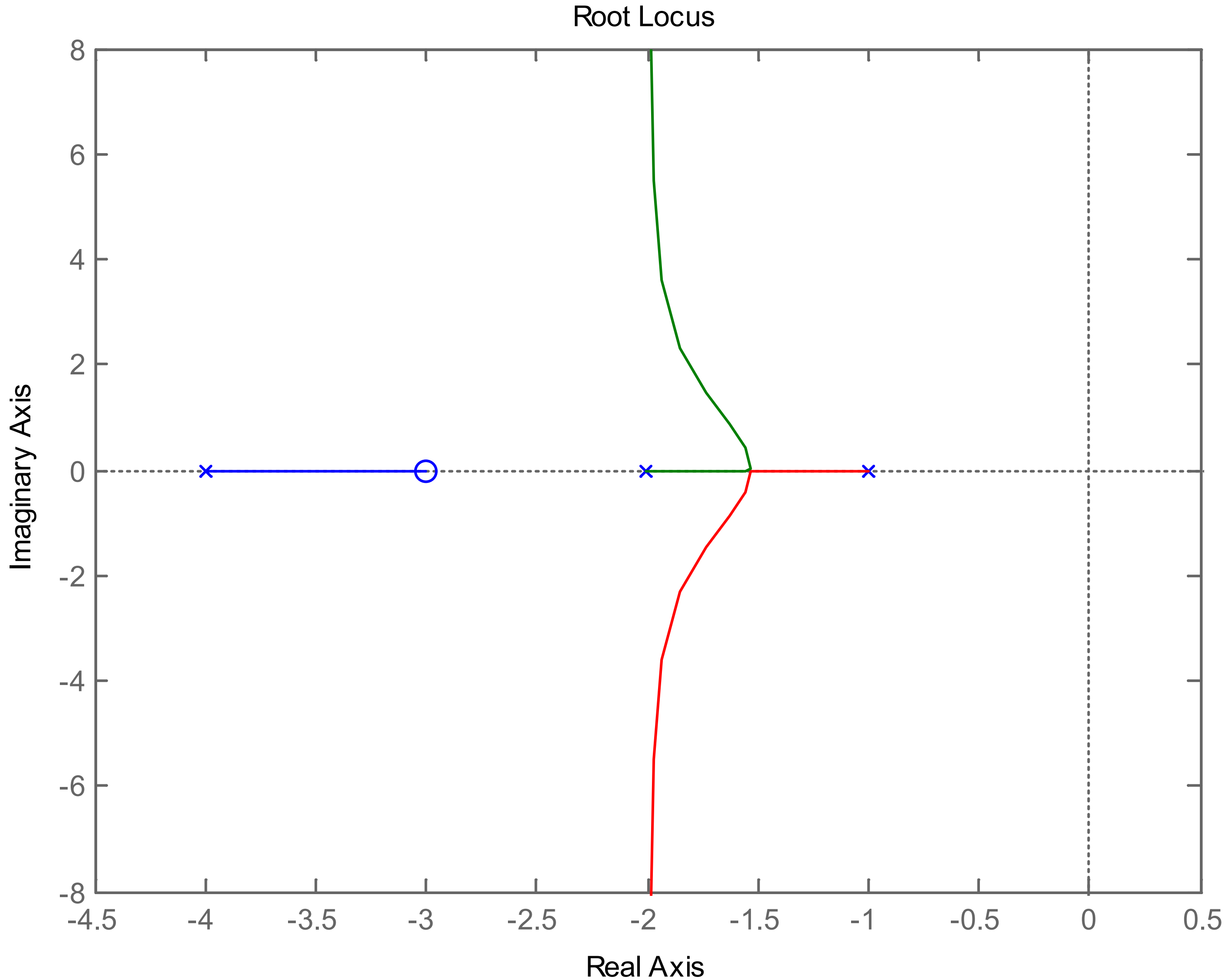


Example#2

- Step-4: breakaway point



Example#2



Root Locus
8
6
4
2
0
-2
-4
-6
-8
Imaginary Axis
-4.5
-4
-3.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
Real Axis

Example#2

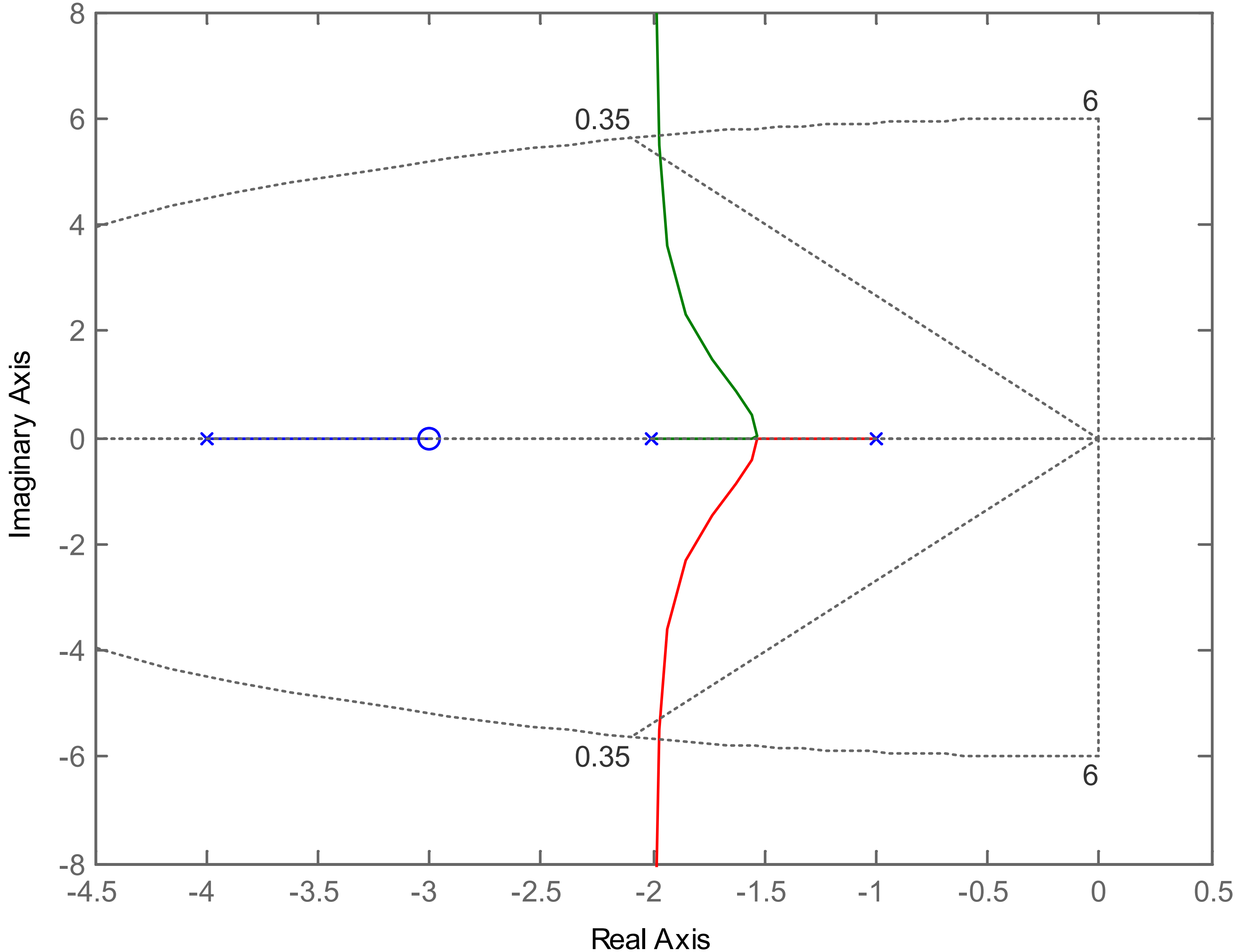
- Mp<30% corresponds to

$$M_p = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} \times 100$$

$$30\% = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} \times 100$$

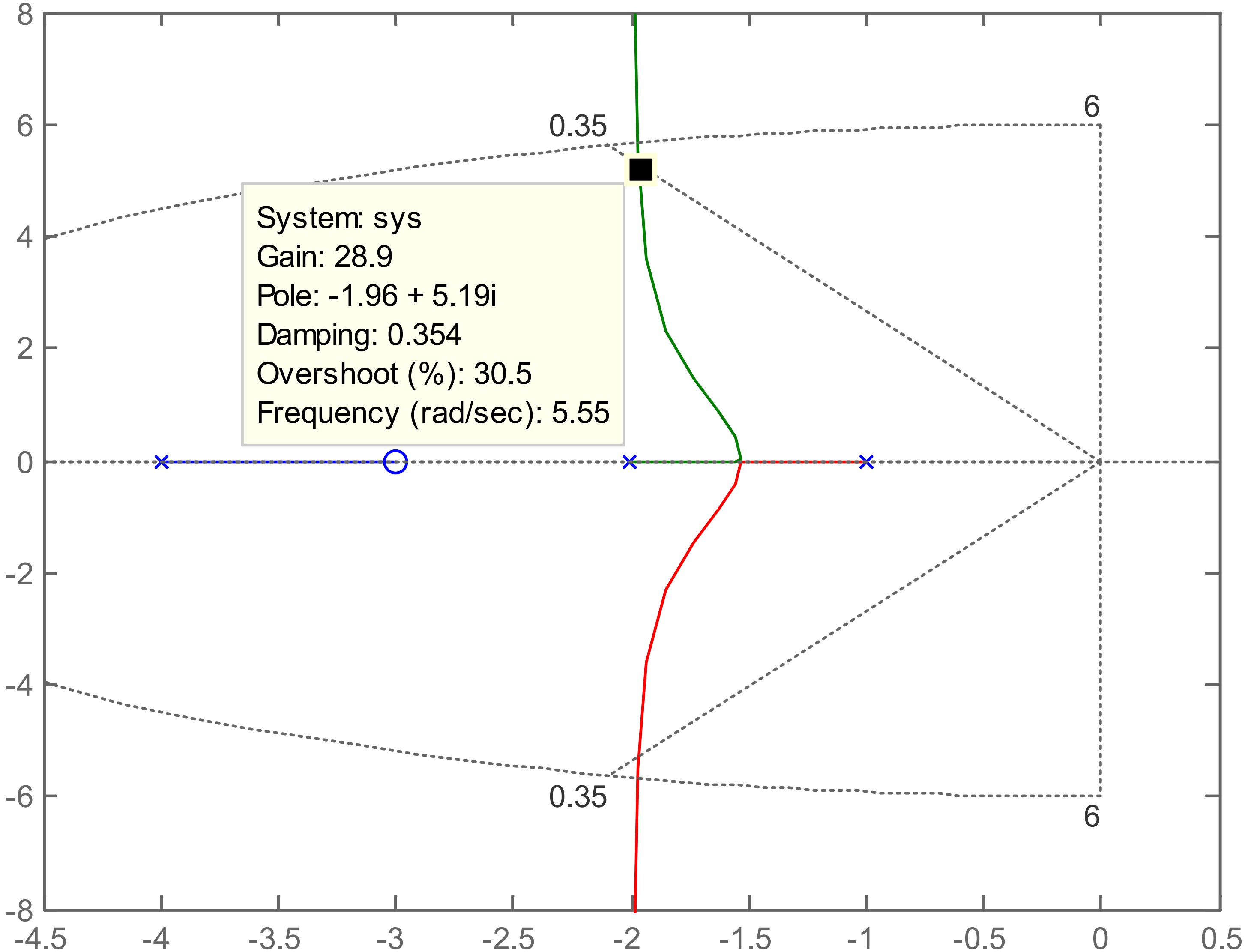
$$\zeta > 0.35$$

Example#2



Imaginary Axis
Real Axis
8
6
4
2
0
-2
-4
-6
-8
-4.5
-4
-3.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
0.35
0.35
6
6

Example#2



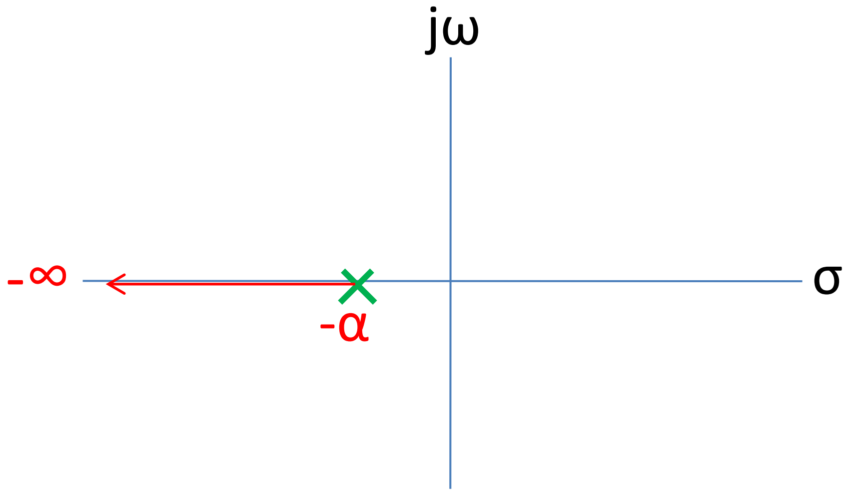
8
6
4
2
0
-2
-4
-6
-8
-4.5
-4
-3.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
0.35
6
0.35
6
System: sys
Gain: 28.9
Pole: -1.96 + 5.19i
Damping: 0.354
Overshoot (%): 30.5
Frequency (rad/sec): 5.55

Root Locus of 1st Order System

- 1st order systems (without zero) are represented by following transfer function.

$$G(s)H(s) = \frac{K}{s+\alpha}$$

- Root locus of such systems is a horizontal line starting from -α and moves towards -∞ as K reaches infinity.



Home Work

- Draw the Root Locus of the following systems.

1) $$G(s)H(s) = \frac{K}{s+2}$$

2) $$G(s)H(s) = \frac{K}{s-1}$$

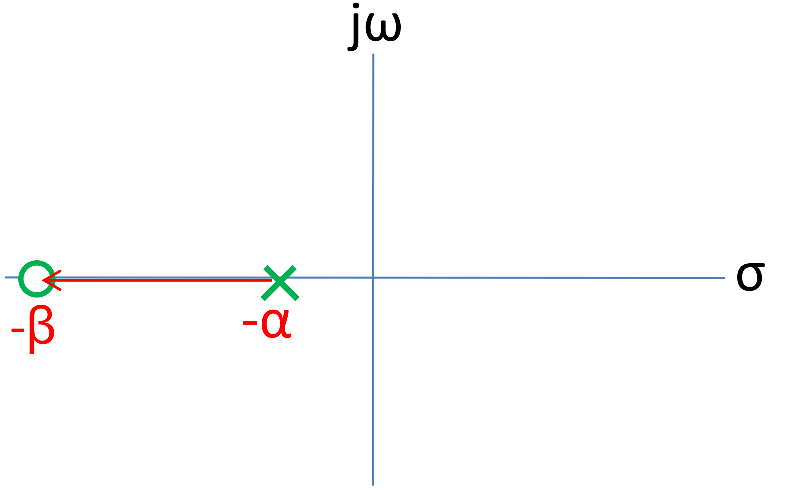
3) $$G(s)H(s) = \frac{K}{s}$$

Root Locus of 1st Order System

- 1st order systems with zero are represented by following transfer function.

$$G(s)H(s) = \frac{K(s+\beta)}{s+\alpha}$$

- Root locus of such systems is a horizontal line starting from -α and moves towards -β as K reaches infinity.



Home Work

- Draw the Root Locus of the following systems.

1) $$G(s)H(s) = \frac{Ks}{s+2}$$

2) $$G(s)H(s) = \frac{K(s+5)}{s-1}$$

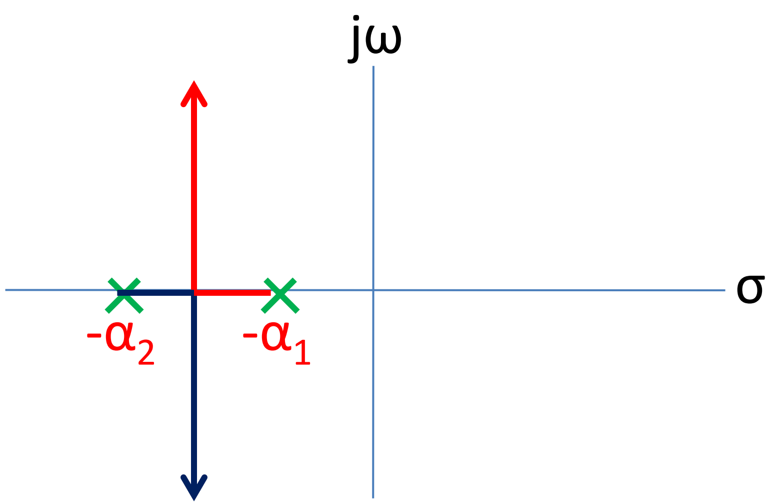
3) $$G(s)H(s) = \frac{K(s+3)}{s}$$

Root Locus of 2nd Order System

- Second order systems (without zeros) have two poles and the transfer function is given

$$G(s)H(s) = \frac{K}{(s+\alpha_1)(s+\alpha_2)}$$

- Root loci of such systems are vertical lines.



Home Work

- Draw the Root Locus of the following systems.

1) $$G(s)H(s) = \frac{K}{s(s+2)}$$

2) $$G(s)H(s) = \frac{K}{s^2}$$

3) $$G(s)H(s) = \frac{K}{(s+1)(s-3)}$$

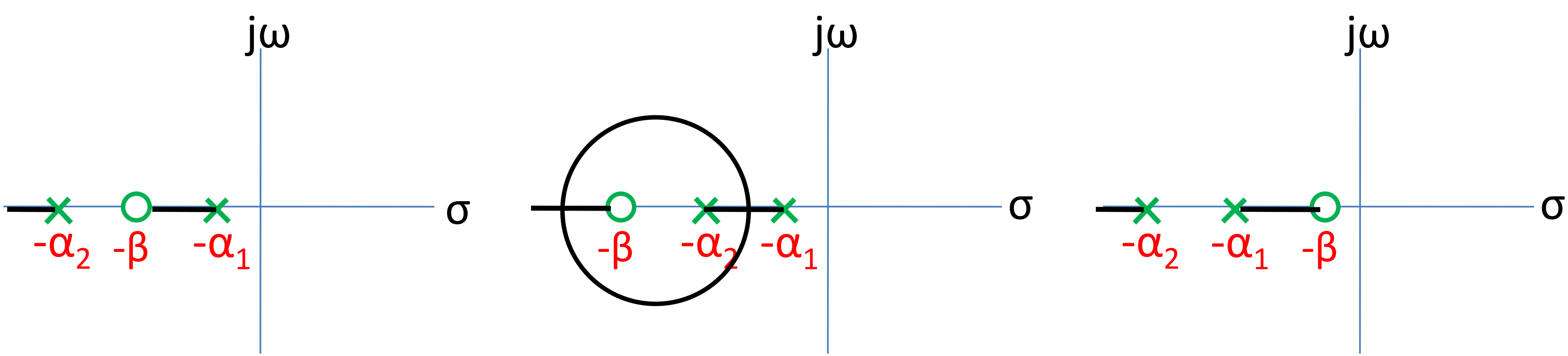
4) $$G(s)H(s) = \frac{K}{s^2 + 3s + 10}$$

Root Locus of 2nd Order System

- Second order systems (with one zero) have two poles and the transfer function is given

$$G(s)H(s) = \frac{K(s+\beta)}{(s+\alpha_1)(s+\alpha_2)}$$

- Root loci of such systems are either horizontal lines or circular depending upon pole-zero configuration.



Home Work

- Draw the Root Locus of the following systems.

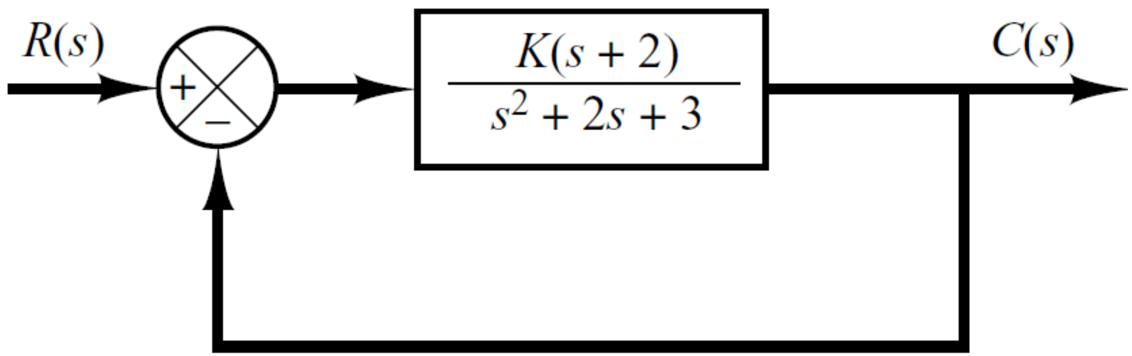
1) $$G(s)H(s) = \frac{K(s+1)}{s(s+2)}$$

2) $$G(s)H(s) = \frac{K(s-2)}{s^2}$$

3) $$G(s)H(s) = \frac{K(s+5)}{(s+1)(s-3)}$$

Example

- Sketch the root-locus plot of following system with complex-conjugate open loop poles.



$$G(s) = \frac{K(s+2)}{s^2+2s+3}, \qquad H(s) = 1$$

$G(s)$ has a pair of complex-conjugate poles at

$$s = -1 + j\sqrt{2}, \qquad s = -1 - j\sqrt{2}$$

Example

- ***Step-1:*** Pole-Zero Mao

- ***Step-2:*** Determine the root loci on real axis

- ***Step-3***: Asymptotes

Example

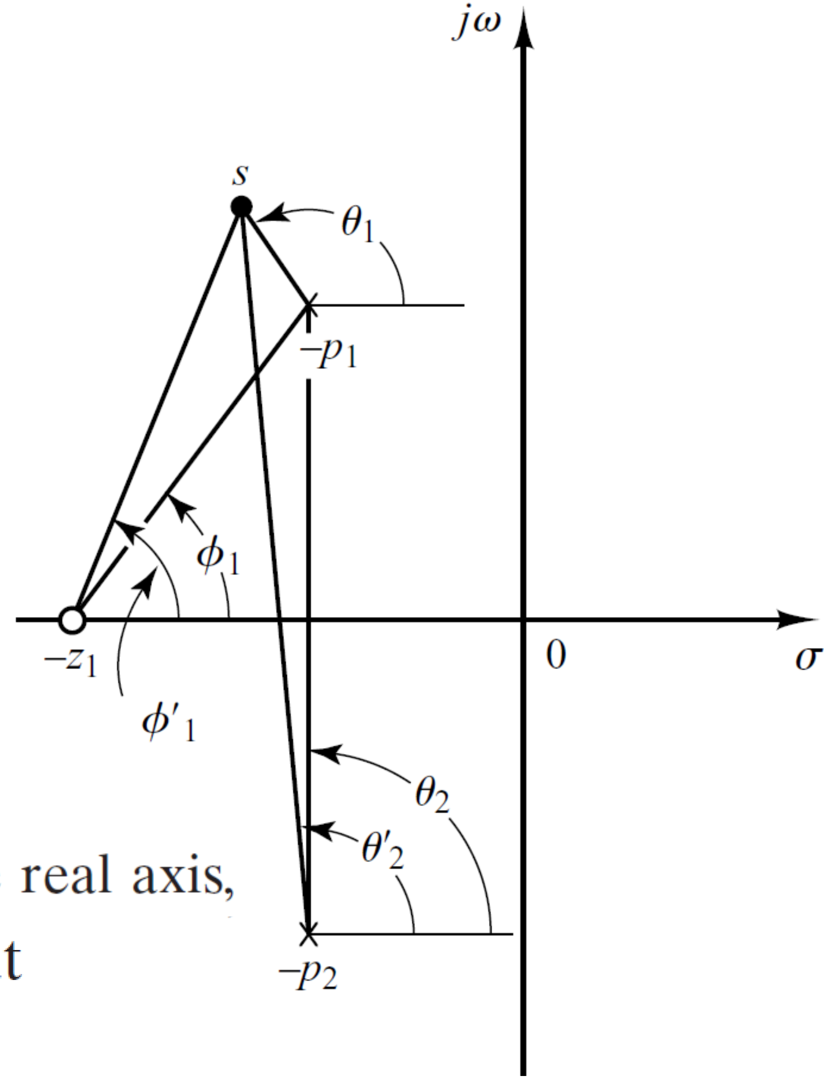
- *Step-4:* Determine the angle of departure from the complex-conjugate open-loop poles.

  – The presence of a pair of complex-conjugate open-loop poles requires the determination of the angle of departure from these poles.

  – Knowledge of this angle is important, since the root locus near a complex pole yields information as to whether the locus originating from the complex pole migrates toward the real axis or extends toward the asymptote.

Example

- ***Step-4:*** Determine the angle of departure from the complex-conjugate open-loop poles.



The angle of departure is then

$$\theta_1 = 180^\circ - \theta_2 + \phi_1$$
$$= 180^\circ - 90^\circ + 55^\circ = 145^\circ$$

Since the root locus is symmetric about the real axis, the angle of departure from the pole at

$$s = -p_2 \text{ is } -145^\circ$$

Example

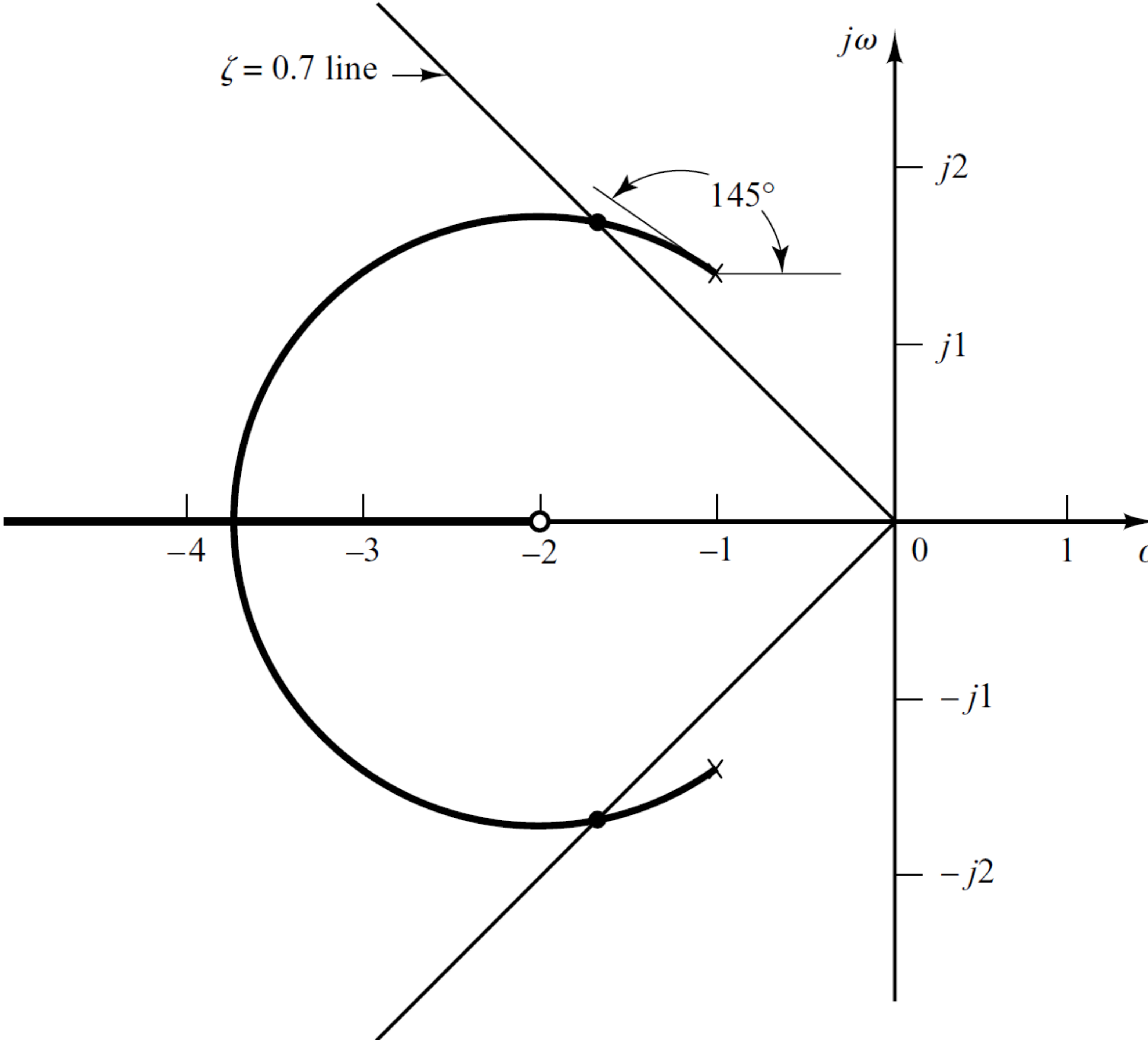
- ***Step-5:*** Break-in point

$$K = -\frac{s^2 + 2s + 3}{s + 2}$$

$$\frac{dK}{ds} = -\frac{(2s + 2)(s + 2) - (s^2 + 2s + 3)}{(s + 2)^2} = 0$$

$$s^2 + 4s + 1 = 0$$

$$s = -3.7320 \qquad \text{or} \qquad s = -0.2680$$


$\zeta = 0.7$ line
145°
$j\omega$
$j2$
$j1$
$-j1$
$-j2$
-4
-3
-2
-1
0
1
σ

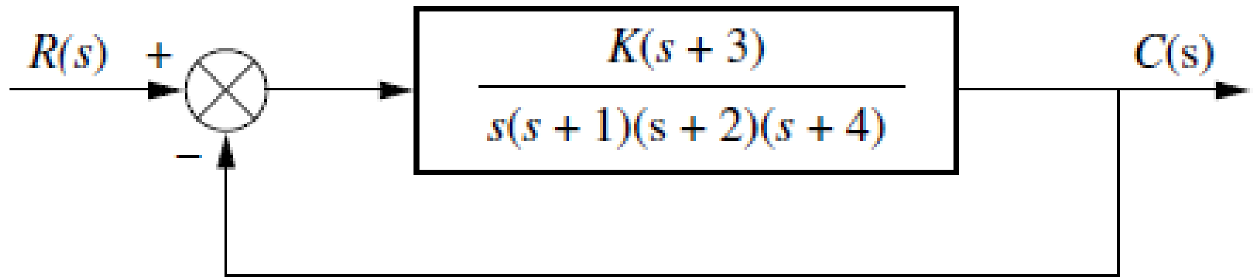
Root Locus of Higher Order System

- Third order System without zero

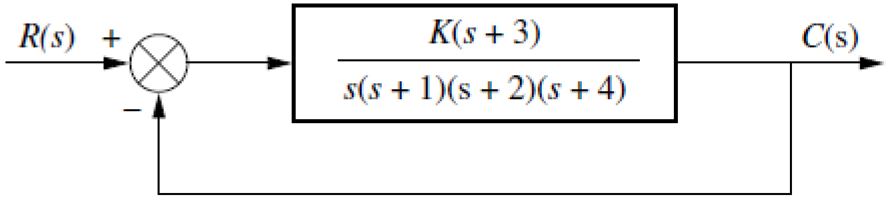
$$G(s)H(s) = \frac{K}{(s+\alpha_1)(s+\alpha_2)(s+\alpha_3)}$$

Root Locus of Higher Order System

- Sketch the Root Loci of following unity feedback system



$$G(s)H(s) = \frac{K(s+3)}{s(s+1)(s+2)(s+4)}$$



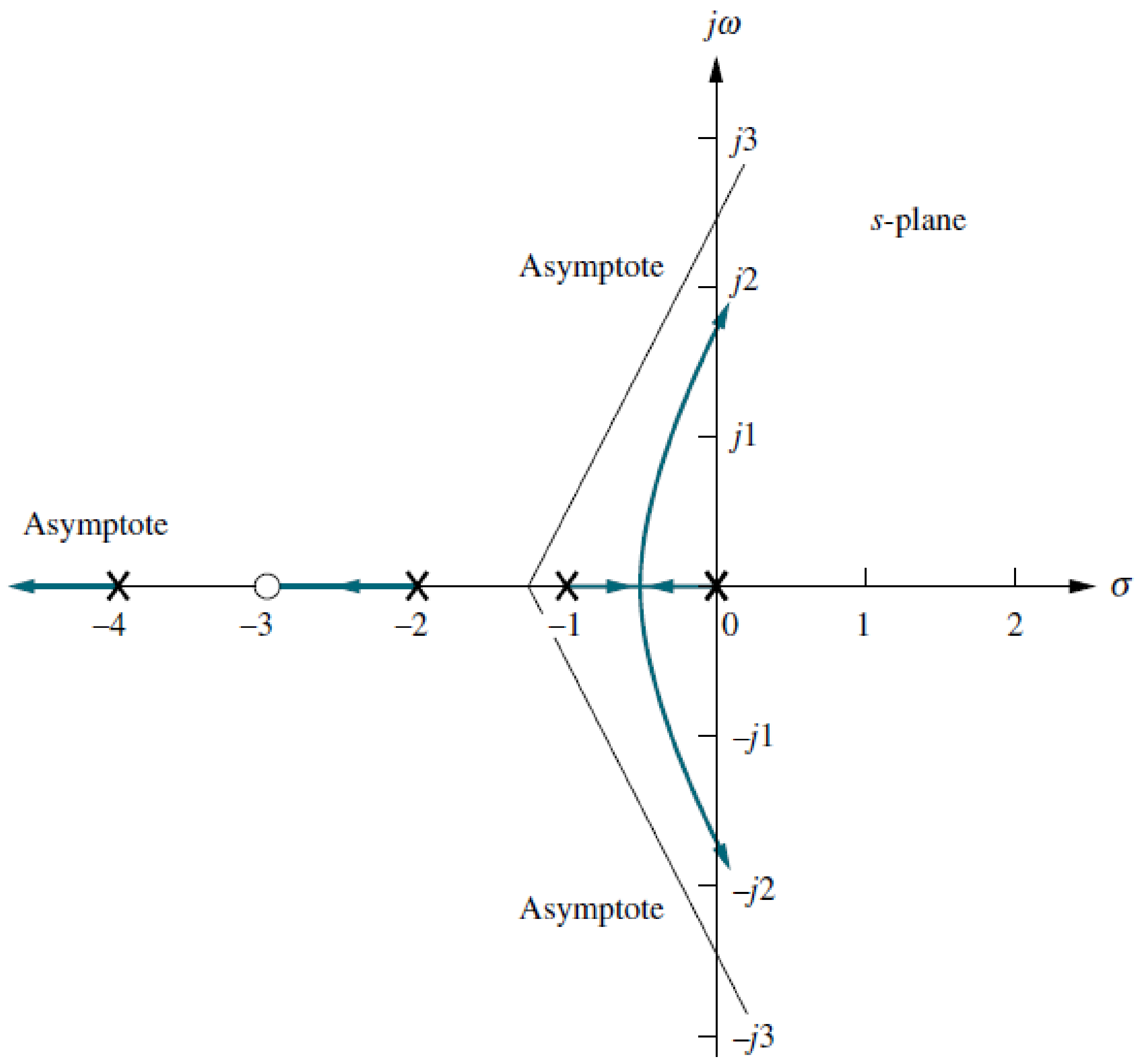
- Let us begin by calculating the asymptotes. The real-axis intercept is evaluated as;

$$\sigma_a = \frac{(-1-2-4)-(-3)}{4-1} = -\frac{4}{3}$$

- The angles of the lines that intersect at - 4/3, given by

$$\begin{aligned}\theta_a &= \frac{(2k+1)\pi}{\#\text{finite poles} - \#\text{finite zeros}} \\ &= \pi/3 \qquad \text{for } k = 0 \\ &= \pi \qquad \text{for } k = 1 \\ &= 5\pi/3 \qquad \text{for } k = 2\end{aligned}$$

- The Figure shows the complete root locus as well as the asymptotes that were just calculated.



Example: Sketch the root locus for the system with the characteristic equation of;

$$1 + GH(s) = 1 + \frac{K(s+1)}{s(s+2)(s+4)^2}$$

a) Number of finite poles = n = 4.
b) Number of finite zeros = m = 1.
c) Number of asymptotes = n - m = 3.
d) Number of branches or loci equals to the number of finite poles (n) = 4.
e) The portion of the real-axis between, 0 and -2, and between, -4 and -∞, lie on the root locus for *K* > 0.

- Using Eq. (v), the real-axis asymptotes intercept is evaluated as;

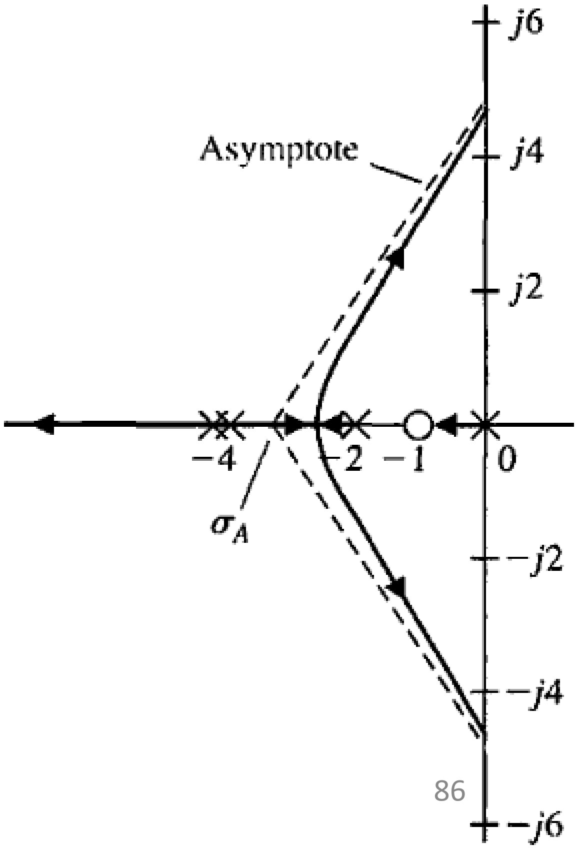
$$\sigma_a = \frac{(-2) + 2(-4) - (-1)}{n - m} = \frac{-10 + 1}{4 - 1} = -3$$

- The angles of the asymptotes that intersect at - 3, given by Eq. (vi), are;

$$\theta_a = \frac{(2k+1)\pi}{n - m} = \frac{(2k+1)\pi}{4 - 1}$$

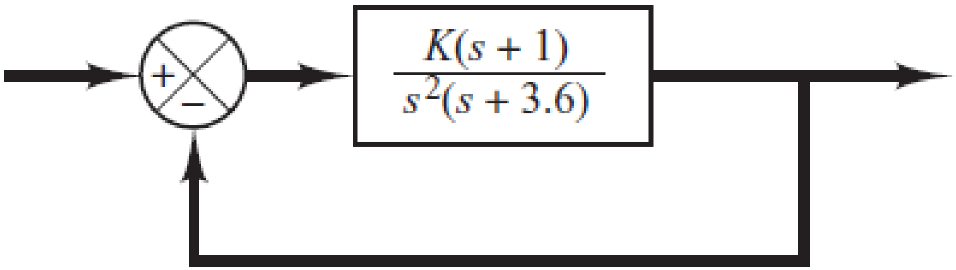
For *K* = 0, $\theta a = 60^o$
For *K* = 1, $\theta a = 180^o$
For *K* = 2, $\theta a = 300^o$

- The root-locus plot of the system is shown in the figure below.
- It is noted that there are three asymptotes. Since n – m = 3.
- The root loci must begin at the poles; two loci (or branches) must leave the double pole at s = -4.
- Using Eq. (vii), the breakaway point, σ, can be determine as;



- The solution of the above equation is $\sigma = -2.59$.

Example: Sketch the root loci for the system.



- A root locus exists on the real axis between points **s = –1** and **s = –3.6**.
- The intersection of the asymptotes and the real axis is determined as,

$$\sigma_a = \frac{0+0+3.6-1}{n\ -m} = \frac{2.6}{3\ -1} = -1.3$$

- The angles of the asymptotes that intersect at – 1.3, given by Eq. (vi), are;

$$\theta_a = \frac{(2k+1)\pi}{n\ -m} = \frac{(2k+1)\pi}{3-1}$$

For K = 0, θa = 90°
For K = 1, θa = -90° or 270°

- Since the characteristic equation is $s^3 + 3.6s^2 + K(s+1) = 0$
- We have $K = -\frac{s^3 + 3.6s^2}{s+1} \longrightarrow$ (a)

- The breakaway and break-in points are found from Eq. (a) as,

$$\frac{dK}{ds} = -\frac{(3s^2 + 7.2s)(s + 1) - (s^3 + 3.6s^2)}{(s + 1)^2} = 0$$

$$\text{or} \qquad s^3 + 3.3s^2 + 3.6s = 0$$

From which we get,

$$s = 0, \qquad s = -1.65 + j0.9367, \qquad s = -1.65 - j0.9367$$

- Point $s = 0$ corresponds to the actual breakaway point. But points $s = 1.65 \pm j0.9367$ neither breakaway nor break-in points, because the corresponding gain values K become complex quantities.

- To check the points where root-locus branches may cross the imaginary axis, substitute $s = j\omega$ into the characteristic equation, yielding.

$$(j\omega)^3 + 3.6(j\omega)^2 + Kj\omega + K = 0$$

or

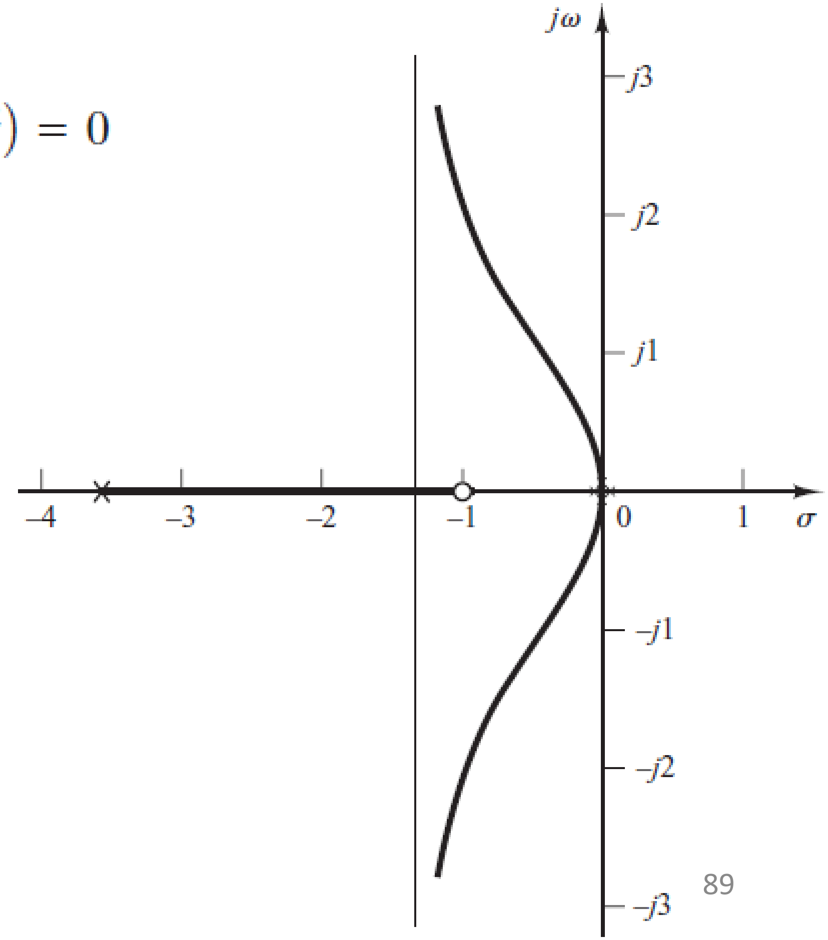
$$\left(K - 3.6\omega^2\right) + j\omega\left(K - \omega^2\right) = 0$$

- Notice that this equation can be satisfied only if $\omega = 0, K = 0$.

- Because of the presence of a double pole at the origin, the root locus is tangent to the $j\omega$axis at $k = 0$.

- The root-locus branches do not cross the $j\omega$axis.

- The root loci of this system is shown in the Figure.



Control System Toolbox

Transfer Function

$$H(s) = \frac{p_1 s^n + p_2 s^{n-1} + \ldots + p_{n+1}}{q_1 s^m + q_1 s^{m-1} + \ldots + q_{m+1}}$$

where

$p_1, p_2 \ldots p_{n+1}$ numerator coefficients

$q_1, q_1 \ldots q_{m+1}$ denominator coefficients

Control System Toolbox

Transfer Function

- Consider a linear time invariant (LTI) single-input/single-output system

$$y'' + 6y' + 5y = 4u' + 3u$$

- Applying Laplace Transform to both sides with zero initial conditions

$$G(s) = \frac{Y(s)}{U(s)} = \frac{4s+3}{s^2+6s+5}$$

Control System Toolbox

Transfer Function

```
>> num = [4 3];
>> den = [1 6 5];
>> sys = tf(num,den)
    Transfer function:
      4 s + 3
    ----------------
    s^2 + 6 s + 5
```

```
>> [num,den] =
      tfdata(sys,'v')
num =
     0   4   3
den =
     1   6   5
```

Control System Toolbox

Zero-pole-gain model (ZPK)

$$H(s) = K\,\frac{(s-p_1)(s-p_2)+\ldots+(s-p_n)}{(s-q_1)(s-q_2)+\ldots+(s-q_m)}$$

where

$p_1, p_2 \ldots p_{n+1}$ the zeros of H(s)

$q_1, q_1 \ldots q_{m+1}$ the poles of H(s)

Control System Toolbox

Zero-pole-gain model (ZPK)

- Consider a Linear time invariant (LTI) single-input/single-output system

$$y'' + 6y' + 5y = 4u' + 3u$$

- Applying Laplace Transform to both sides with zero initial conditions

$$G(s) = \frac{Y(s)}{U(s)} = \frac{4s+3}{s^2+6s+5} = \frac{4(s+0.75)}{(s+1)(s+5)}$$

Control System Toolbox

Zero-pole-gain model (ZPK)

```
>> sys1 =
 zpk(-0.75,[-1 -5],4)

Zero/pole/gain:
4 (s+0.75)
-----------
(s+1) (s+5)
```

```
>> [ze,po,k] = zpkdata(sys1,'v')
ze =
   -0.7500
po =
    -1
    -5
k =
     4
```

Control System Toolbox

State-Space Model (SS)

$$\dot{x} = A x + B u$$

$$y = C x + D u$$

where

x	state vector
u and y	input and output vectors
A, B, C and D	state-space matrices

Control System Toolbox

State-Space Models

- Consider a Linear time invariant (LTI) single-input/single-output system

$$y''+6y'+5y=4u''+3u$$

- State-space model for this system is

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -5 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \qquad \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$y = \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Control System Toolbox

State-Space Models

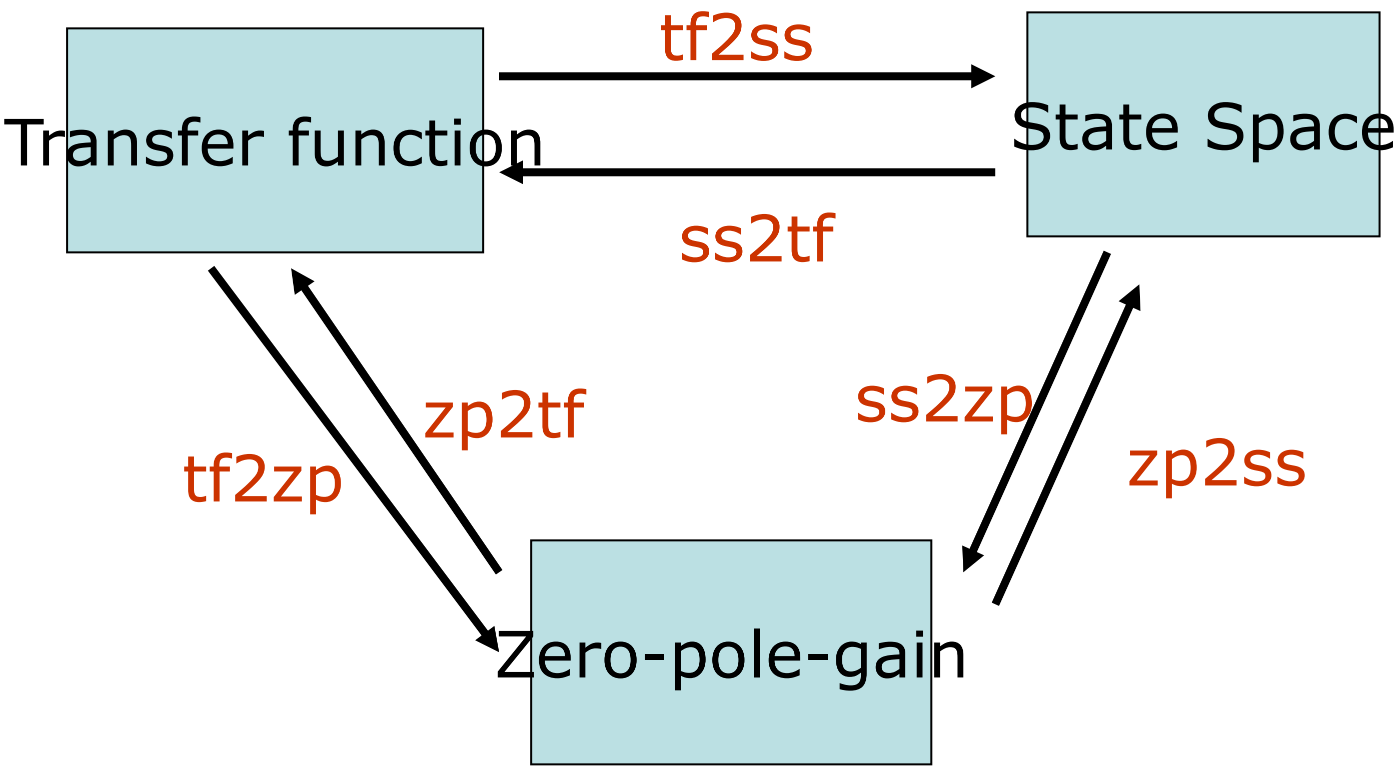
```
>> sys = ss([0 1; -5 -6],[0;1],[3,4],0)
 a =
        x1  x2
   x1   0   1
   x2  -5  -6

 b =
        u1
   x1   0
   x2   1

 c =
        x1  x2
   y1   3   4

 d =
        u1
   y1   0
```

Control System Toolbox

State Space Models

- **rss, drss** - **Random stable state-space models.**
- **ss2ss** - **State coordinate transformation.**
- **canon** - **State-space canonical forms.**
- **ctrb** - **Controllability matrix.**
- **obsv** - **Observability matrix.**
- **gram** - **Controllability and observability gramians.**
- **ssbal** - **Diagonal balancing of state-space realizations.**
- **balreal** - **Gramian-based input/output balancing.**
- **modred** - **Model state reduction.**
- **minreal** - **Minimal realization and pole/zero cancellation.**
- **sminreal** - **Structurally minimal realization.**

Conversion between different models



Model Dynamics

- **pzmap**: Pole-zero map of LTI models.
- **pole, eig** - System poles
- **zero** - System (transmission) zeros.
- **dcgain**: DC gain of LTI models.
- **bandwidth** - System bandwidth.
- **iopzmap** - Input/Output Pole-zero map.
- **damp** - Natural frequency and damping of system
- **esort** - Sort continuous poles by real part.
- **dsort** - Sort discrete poles by magnitude.
- **covar** - Covariance of response to white noise.

Control System Toolbox

Time Response of Systems

- Impulse Response *(impulse)*
- Step Response *(step)*
- General Time Response *(lsim)*
- Polynomial multiplication (*conv*)
- Polynomial division *(deconv)*
- Partial Fraction Expansion *(residue)*
- gensig - Generate input signal for lsim.

Control System Toolbox

Time Response of Systems

- The impulse *response* of a system is its output when the *input is a unit impulse*.
- The step *response* of a system is its output when the *input is a unit step*.
- The general *response* of a system to *any input* can be computed using the lsim command.

Control System Toolbox

Time Response of Systems

Problem Given the LTI system

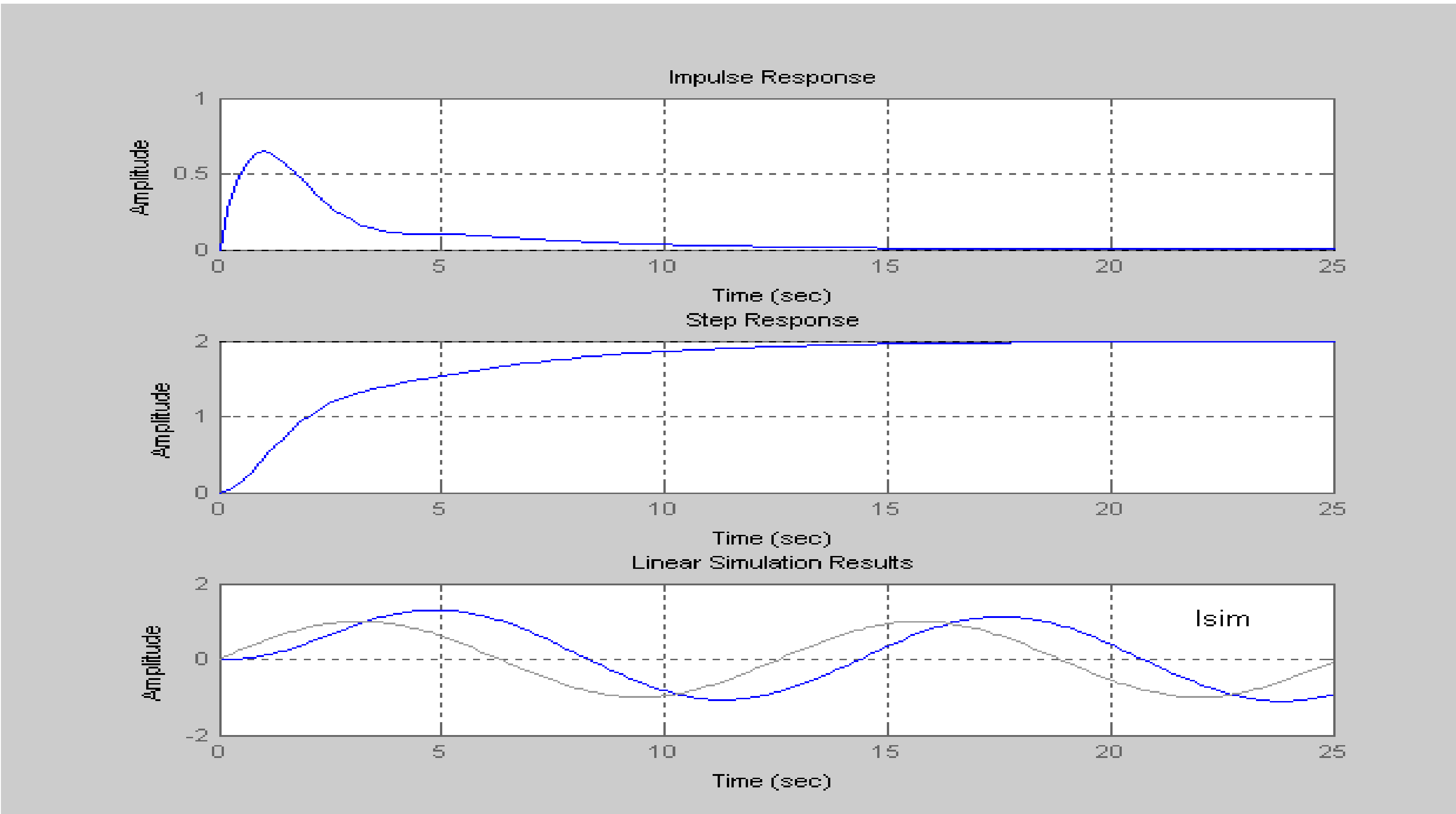
$$G(s) = \frac{3s+2}{2s^3+4s^2+5s+1}$$

Plot the following responses for:

- The impulse response using the `impulse` command.
- The step response using the `step` command.
- The response to the input $u(t) = sin(0.5t)$ calculated using both the `lsim` commands

Control System Toolbox

Time Response of Systems



Frequency Domain Analysis and Design

Root Locus

- Plot the root locus of the following system

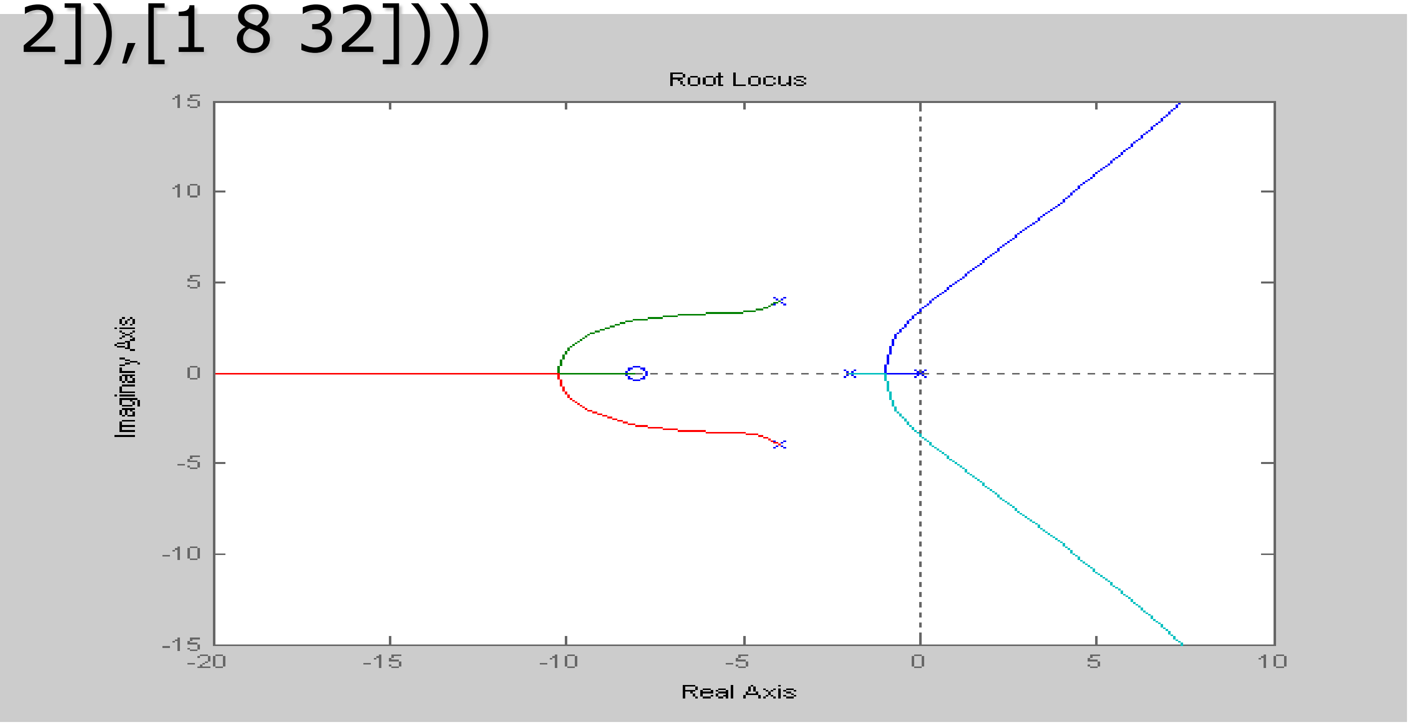
$$G(s) = \frac{K(s+8)}{s(s+2)(s^2+8s+32)}$$

MATLAB Control Toolbo

Frequency Domain Analysis and Design

Root Locus

```
>> rlocus(tf([1 8], conv(conv([1 0],[1 2]),[1 8 32])))
```



PEARSON

Frequency Response: Bode and Nyquist Plots

- Typically, the analysis and design of a control system requires an examination of its frequency response over a range of frequencies of interest.

- The MATLAB Control System Toolbox provides functions to generate two of the most common frequency response plots: Bode Plot (bode command) and Nyquist Plot (nyquist command).

Control System Toolbox

Frequency Response: Bode Plot

Problem

- Given the LTI system

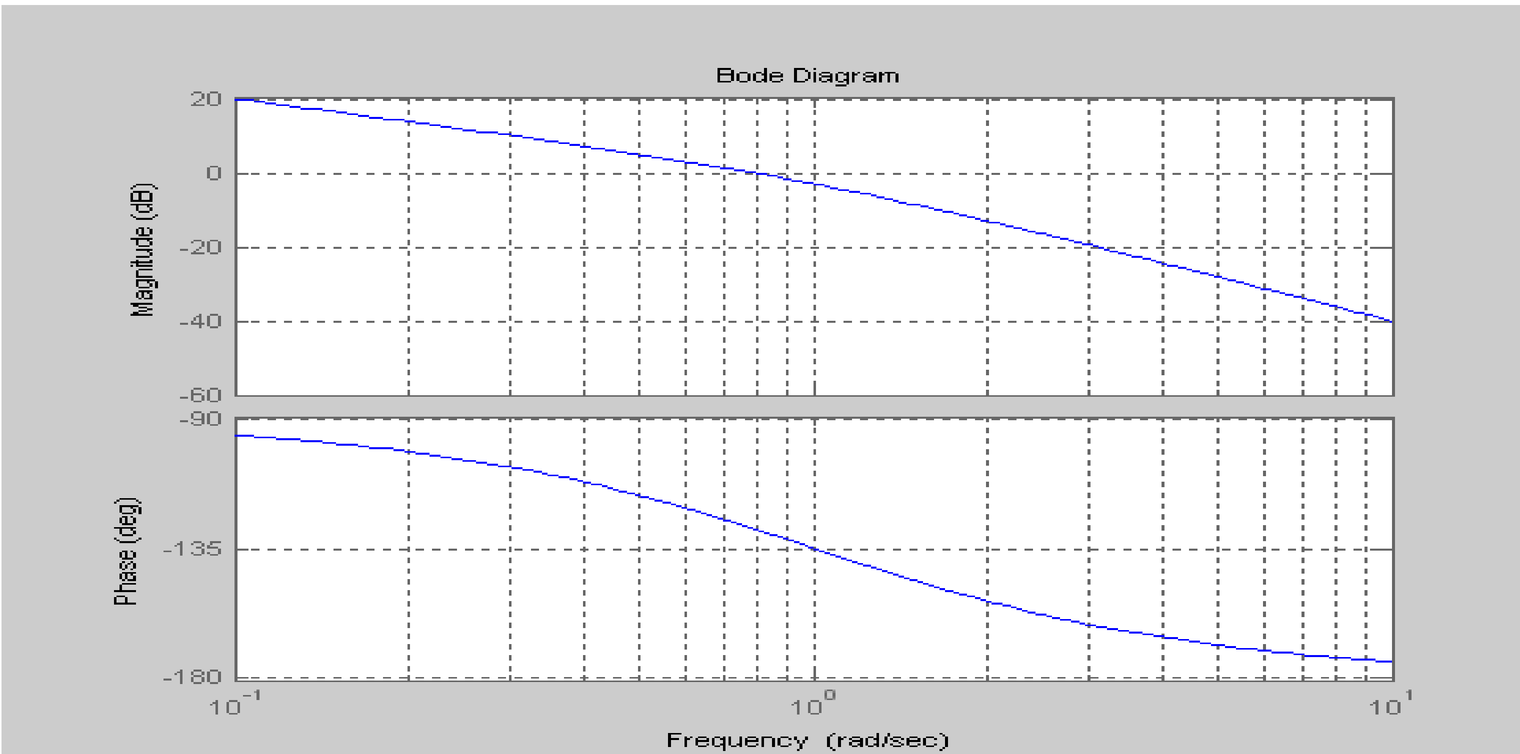
$$G(s) = \frac{1}{s(s+1)}$$

Draw the Bode diagram for 100 values of frequency in the interval $\left[10^{-1} \quad 10\right]$.

Control System Toolbox

Frequency Response: Bode Plot

```
>>bode(tf(1, [1 1 0]), logspace(-1,1,100));
```



Control System Toolbox

Frequency Response: Nyquist Plot

- The loop gain Transfer function *G(s)*
- The **gain margin** is defined as the multiplicative amount that the magnitude of *G(s)* can be increased before the closed loop system goes unstable
- **Phase margin** is defined as the amount of additional phase lag that can be associated with *G(s)* before the closed-loop system goes unstable

Control System Toolbox

Frequency Response: Nyquist Plot

Problem

Given the LTI system

$$G(s) = \frac{1280s + 640}{s^4 + 24.2s^3 + 1604.81s^2 + 320.24s + 16}$$

Draw the bode and nyquist plots for 100 values of frequencies in the interval $\left[10^{-4} \quad 10^{3}\right]$. In addition, find the gain and phase margins.

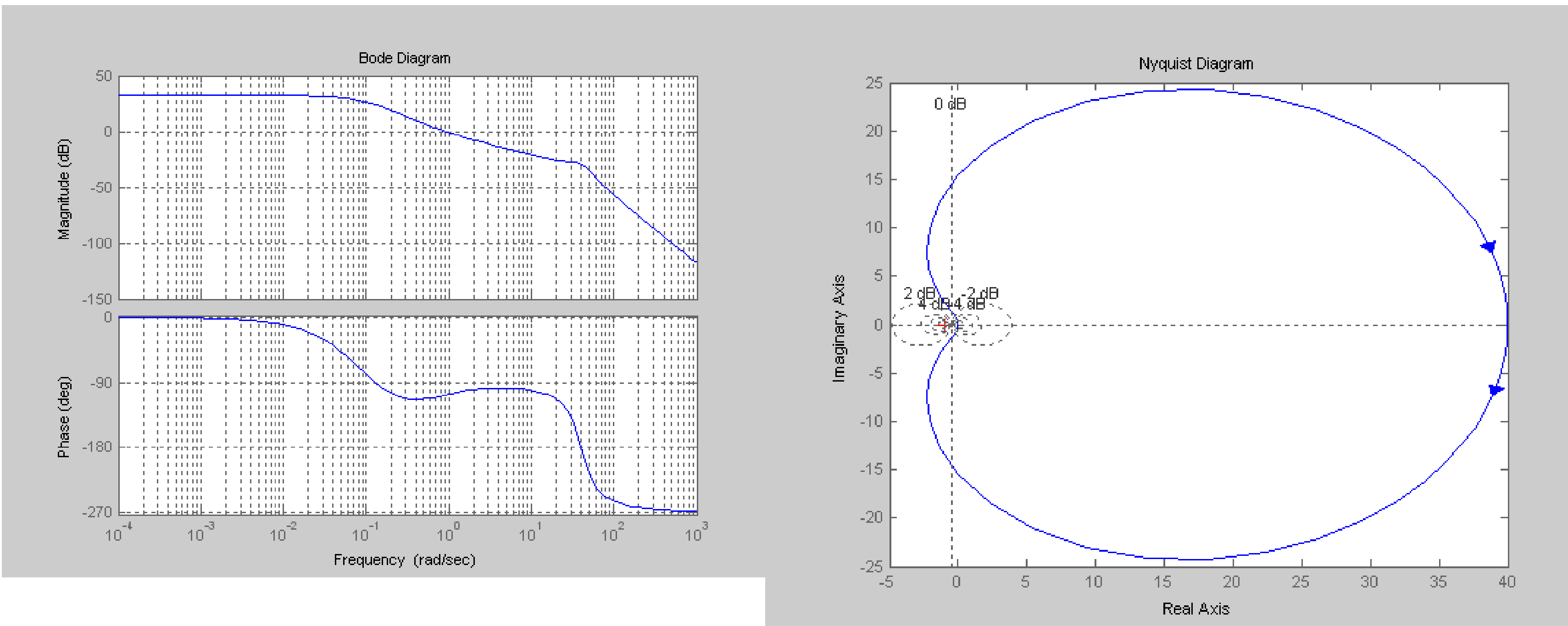
Control System Toolbox

Frequency Response: Nyquist Plot

```
w=logspace(-4,3,100);
sys=tf([1280 640], [1 24.2 1604.81 320.24 16]);
bode(sys,w)
[Gm,Pm,Wcg,Wcp]=margin(sys)
%Nyquist plot
figure
nyquist(sys,w)
```

Control System Toolbox

Frequency Response: Nyquist Plot

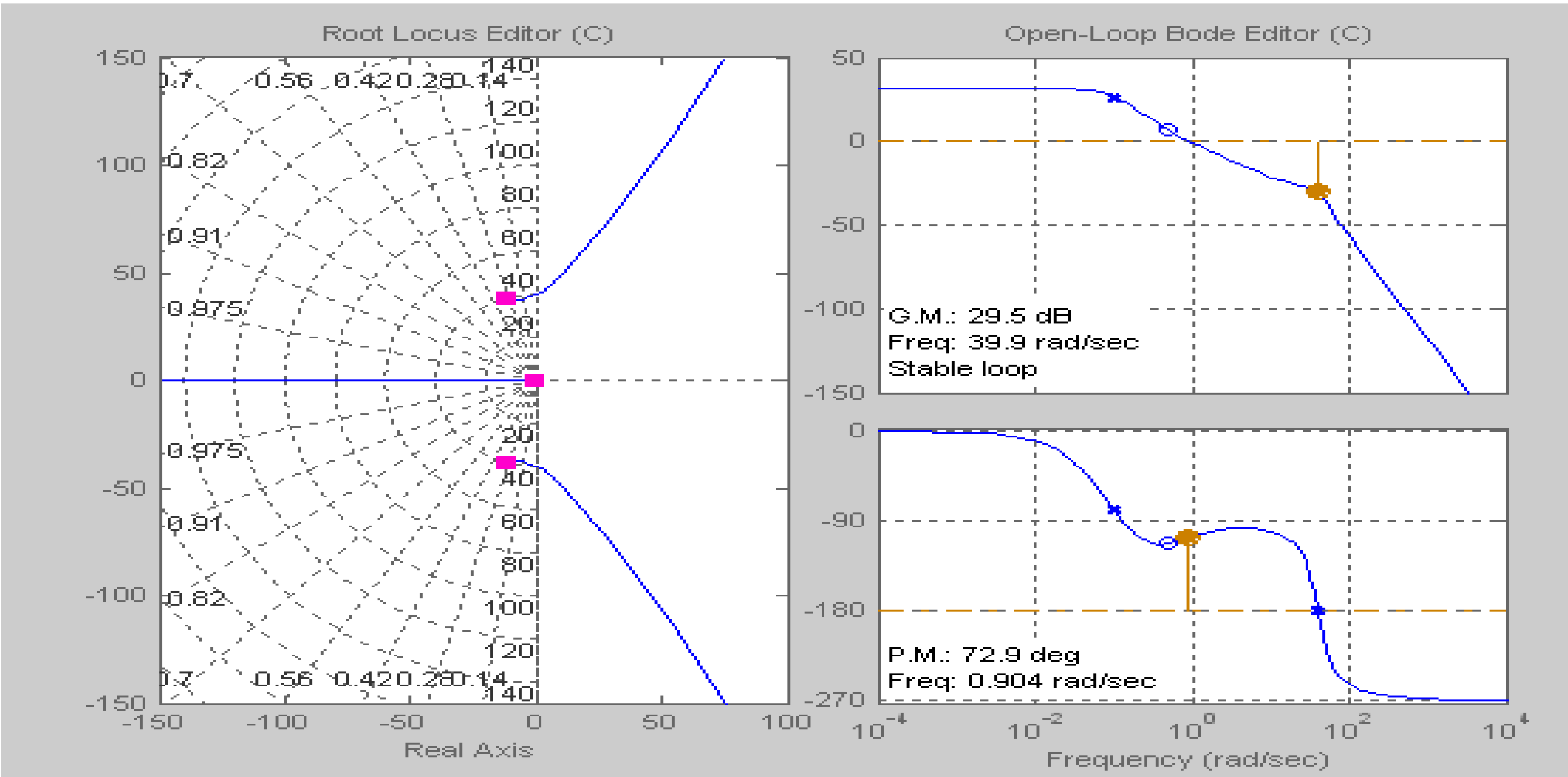


The values of gain and phase margin and corresponding frequencies are

Gm = 29.8637 Pm = 72.8960 Wcg = 39.9099 Wcp = 0.9036

Control System Toolbox

Design Tool: sisotool



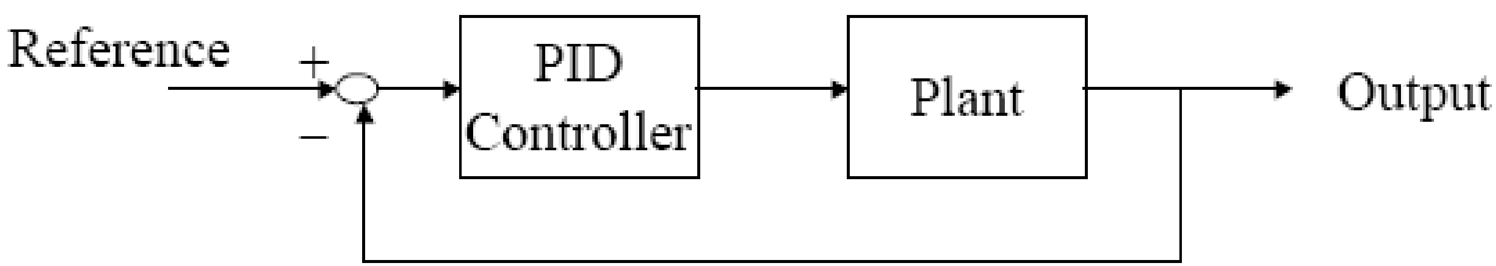
Design with root locus, Bode, and Nichols plots of the open-loop system.
Cannot handle continuous models with time delay.

Lecture Outline

- Introduction to PID
- Modes of Control
  - On-Off Control
  - Proportional Control
  - Proportional + Integral Control
  - Proportional + Derivative Control
  - Proportional + Integral + Derivative Control
- PID Tuning Rules
  - Zeigler-Nichol's Tuning Rules
    - 1st Method
    - 2nd Method

Introduction

- PID Stands for
  - P → Proportional
  - I → Integral
  - D → Derivative



Introduction

- The usefulness of PID controls lies in their general applicability to most control systems.
- In particular, when the mathematical model of the plant is not known and therefore analytical design methods cannot be used, PID controls prove to be most useful.
- In the field of process control systems, it is well known that the basic and modified PID control schemes have proved their usefulness in providing satisfactory control, although in many given situations they may not provide optimal control.

Introduction

- It is interesting to note that more than half of the industrial controllers in use today are PID controllers or modified PID controllers.
- Because most PID controllers are adjusted on-site, many different types of tuning rules have been proposed in the literature.
- Using these tuning rules, delicate and fine tuning of PID controllers can be made on-site.

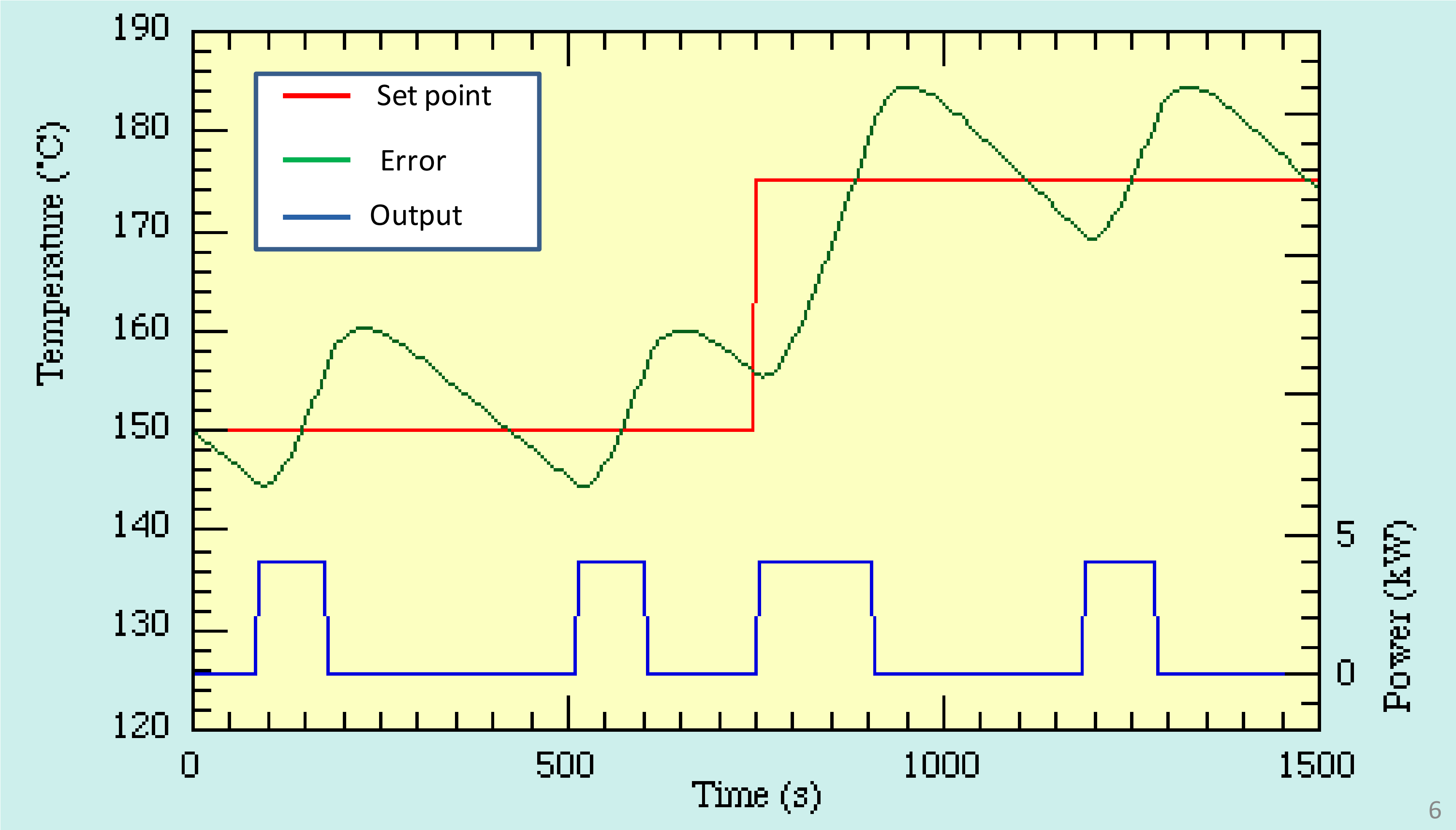
Four Modes of Controllers

- Each mode of control has specific advantages and limitations.

  - On-Off (Bang Bang) Control
  - Proportional (P)
  - Proportional plus Integral (PI)
  - Proportional plus Derivative (PD)
  - Proportional plus Integral plus Derivative (PID)

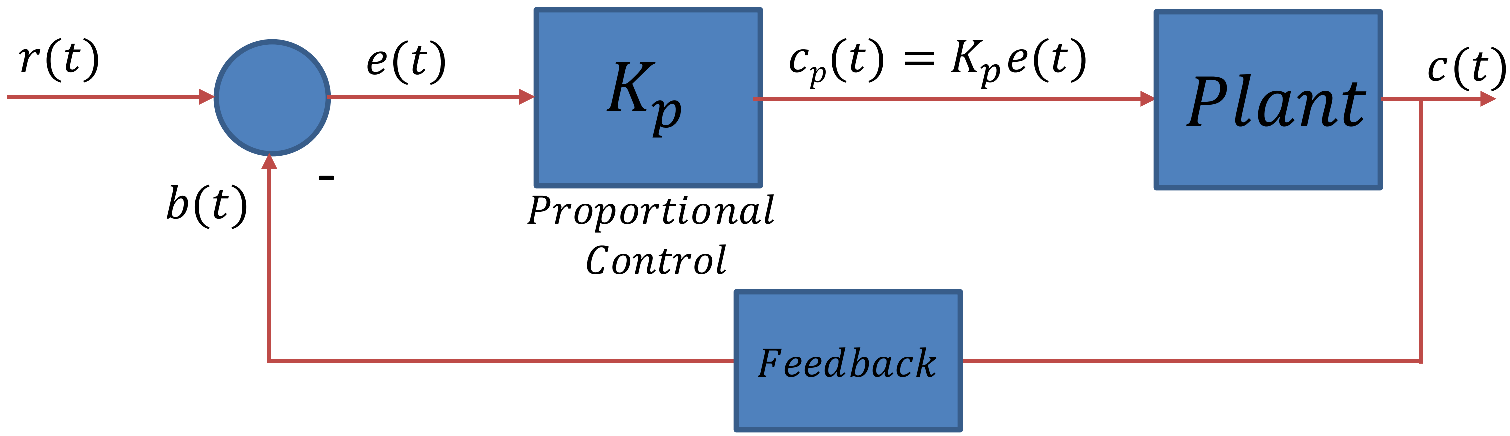
On-Off Control

- This is the simplest form of control.



Proportional Control (P)

- In *proportional mode*, there is a continuous linear relation between value of the controlled variable and position of the final control element.



- Output of proportional controller is

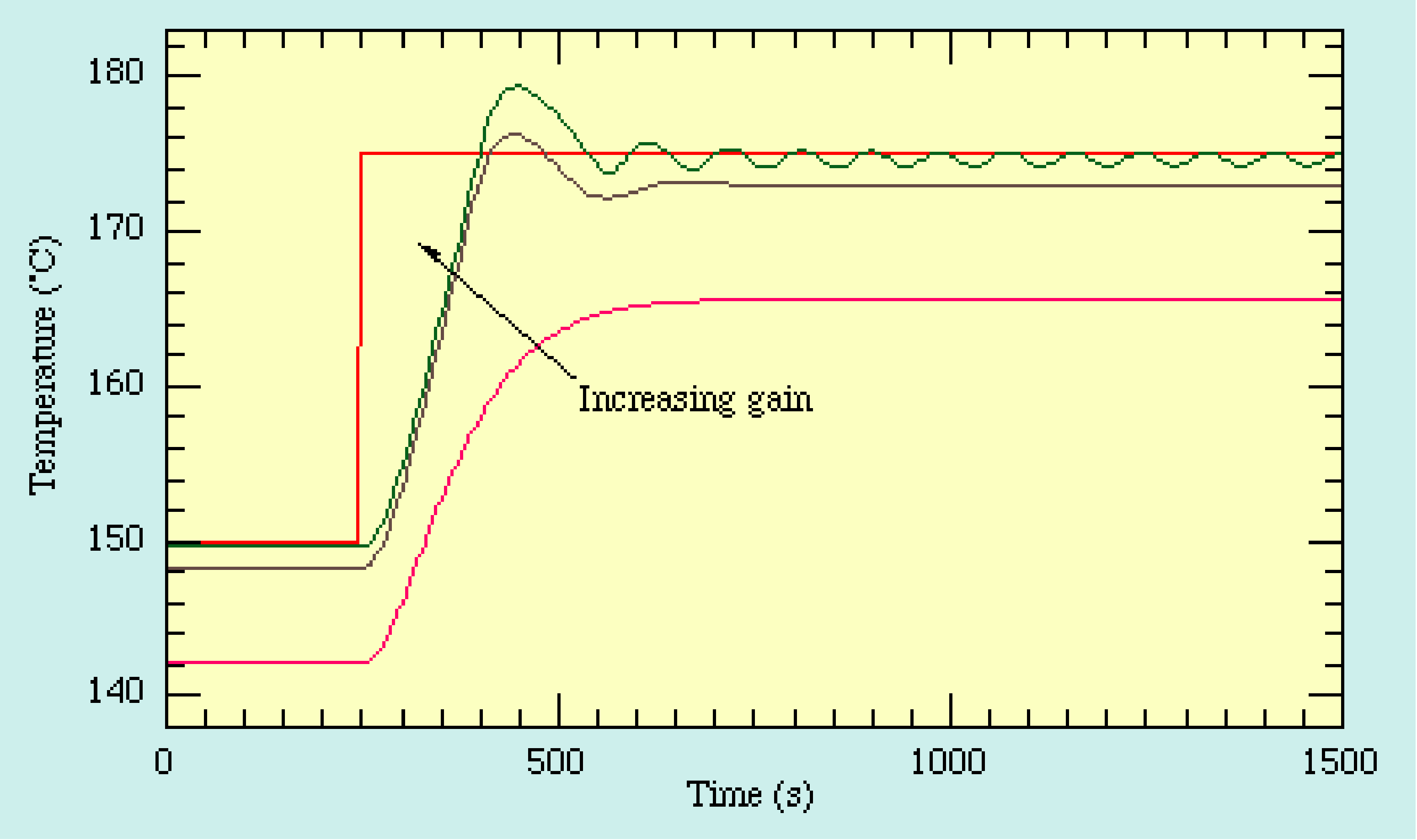
$$c_p(t) = K_p e(t)$$

- The transfer function can be written as

$$\frac{C_p(s)}{E(s)} = K_p$$

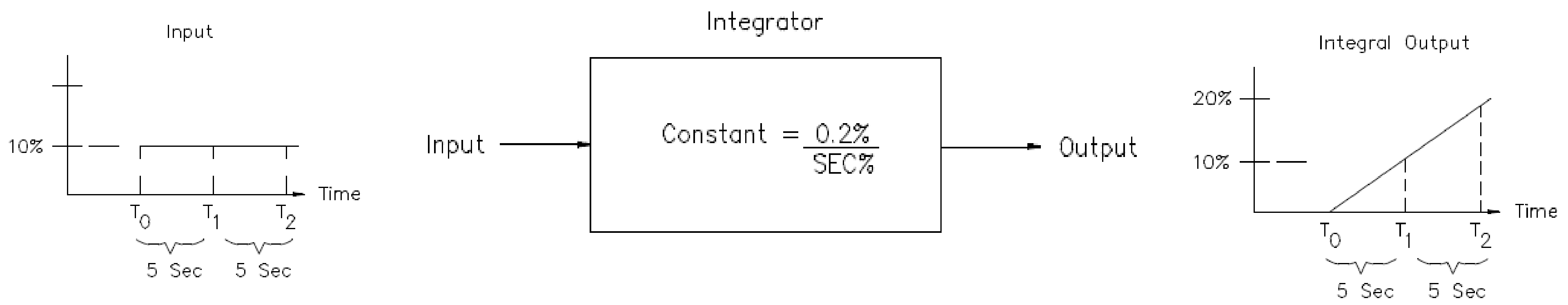
Proportional Controllers (P)

- As the gain is increased the system responds faster to changes in set-point but becomes progressively underdamped and eventually unstable.



Proportional Plus Integral Controllers (PI)

- Integral control describes a controller in which the output rate of change is dependent on the magnitude of the input.
- Specifically, a smaller amplitude input causes a slower rate of change of the output.



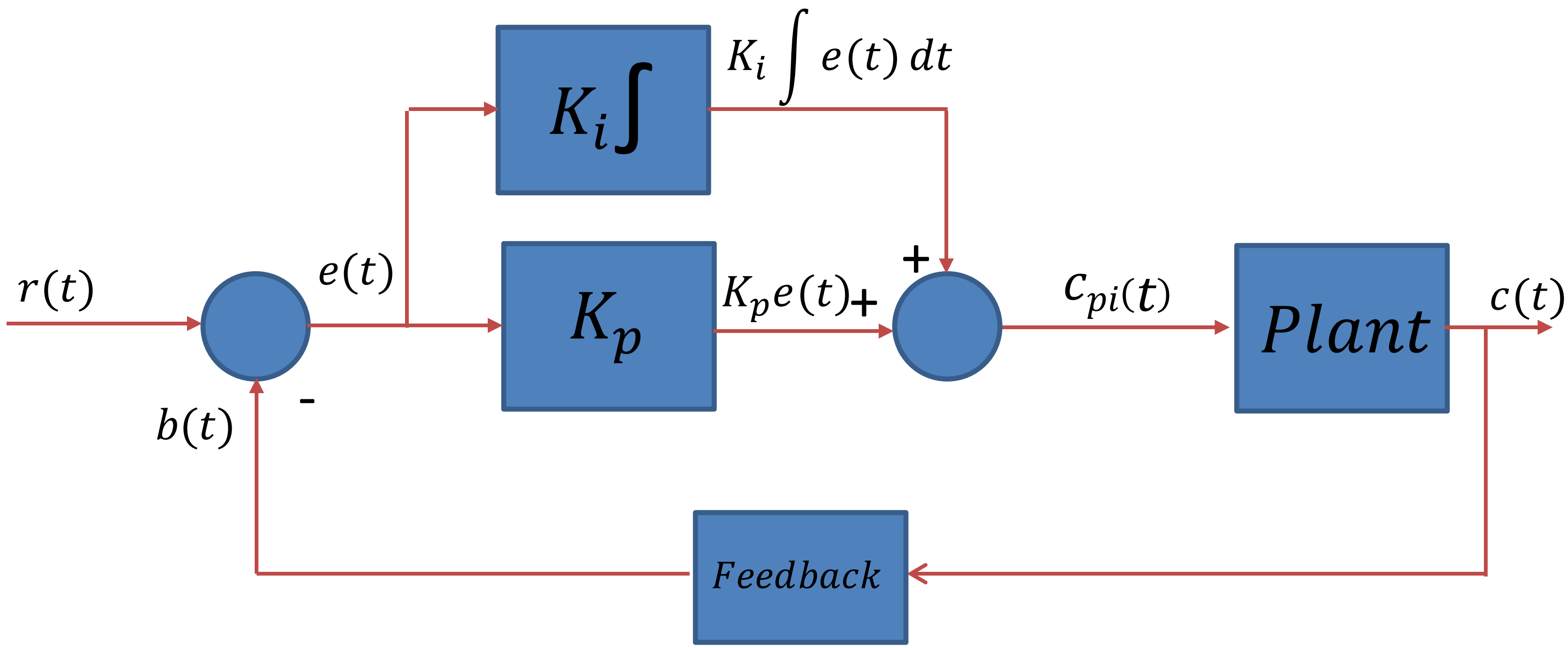
Proportional Plus Integral Controllers (PI)

- The major advantage of integral controllers is that they have the unique ability to return the controlled variable back to the exact set point following a disturbance.

- Disadvantages of the integral control mode are that it responds relatively slowly to an error signal and that it can initially allow a large deviation at the instant the error is produced.

- This can lead to system instability and cyclic operation. For this reason, the integral control mode is not normally used alone, but is combined with another control mode.

Proportional Plus Integral Control (PI)



$$c_{pi}(t) = K_p e(t) + K_i \int e(t)dt$$

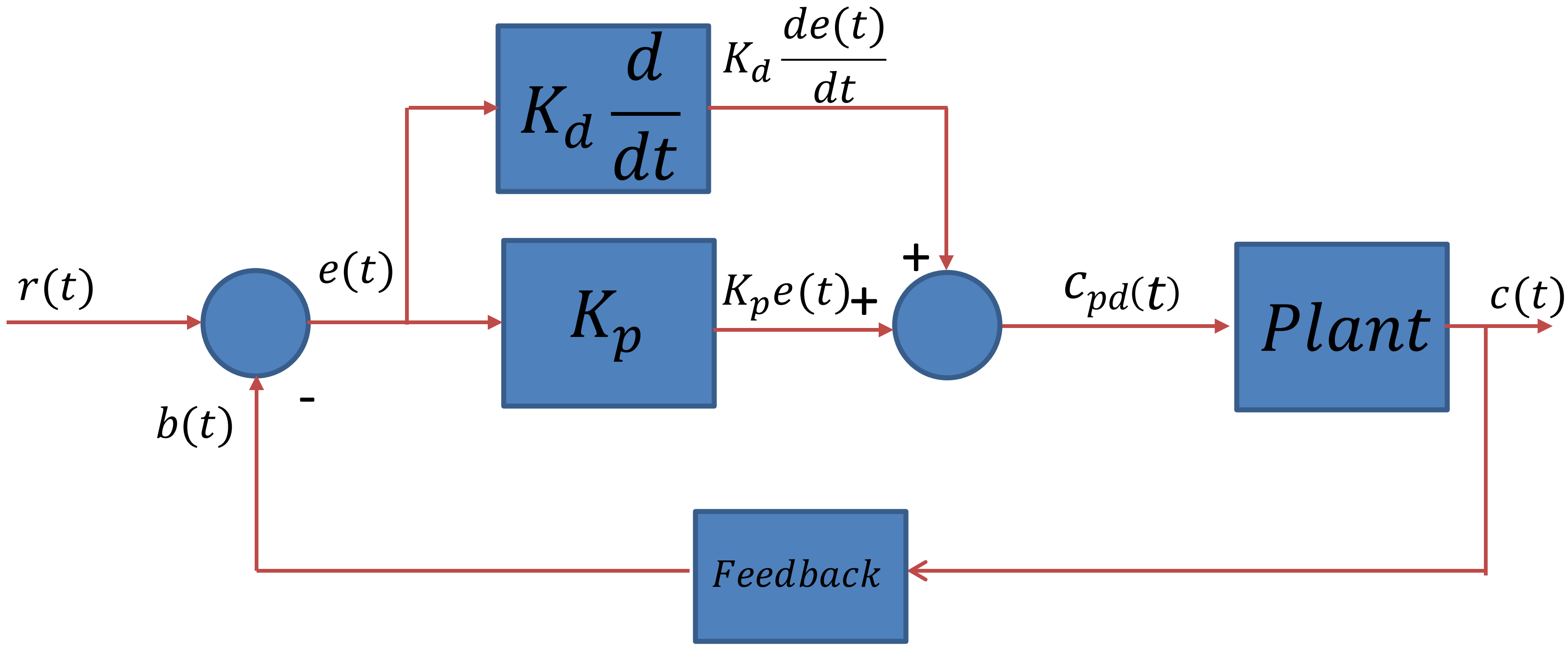
Proportional Plus Integral Control (PI)

$$c_{pi(t)} = K_p e(t) + K_i \int e(t)dt$$

- The transfer function can be written as

$$\frac{C_{pi}(s)}{E(s)} = K_p + K_i \frac{1}{s}$$

Proportional Plus derivative Control (PD)



$$c_{pd(t)} = K_p e(t) + K_d \frac{de(t)}{dt}$$

Proportional Plus derivative Control (PD)

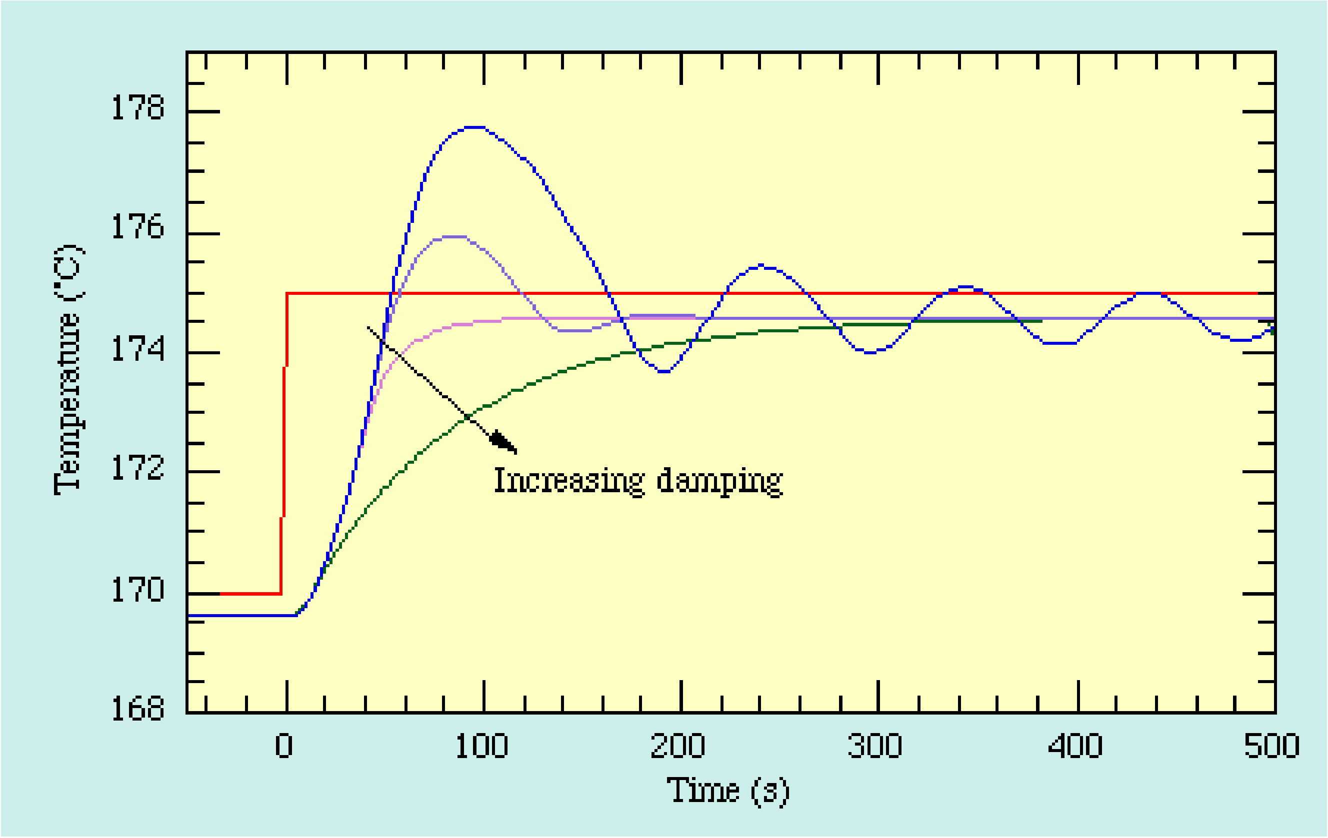
$$c_{pd(t)} = K_p e(t) + K_d \frac{de(t)}{dt}$$

- The transfer function can be written as

$$\frac{C_{pd}(s)}{E(s)} = K_p + K_d s$$

Proportional Plus derivative Control (PD)

- The stability and overshoot problems that arise when a proportional controller is used at high gain can be mitigated by adding a term proportional to the time-derivative of the error signal. The value of the damping can be adjusted to achieve a critically damped response.

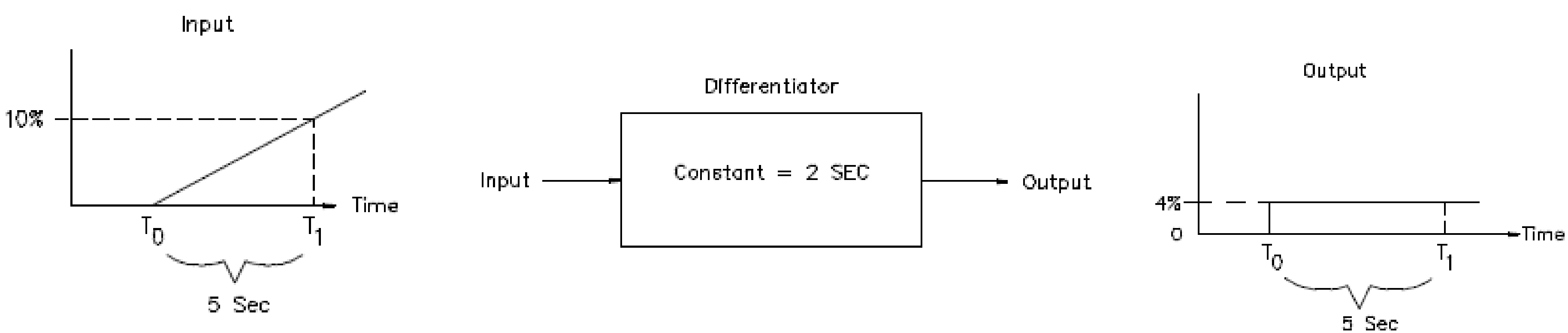


Proportional Plus derivative Control (PD)

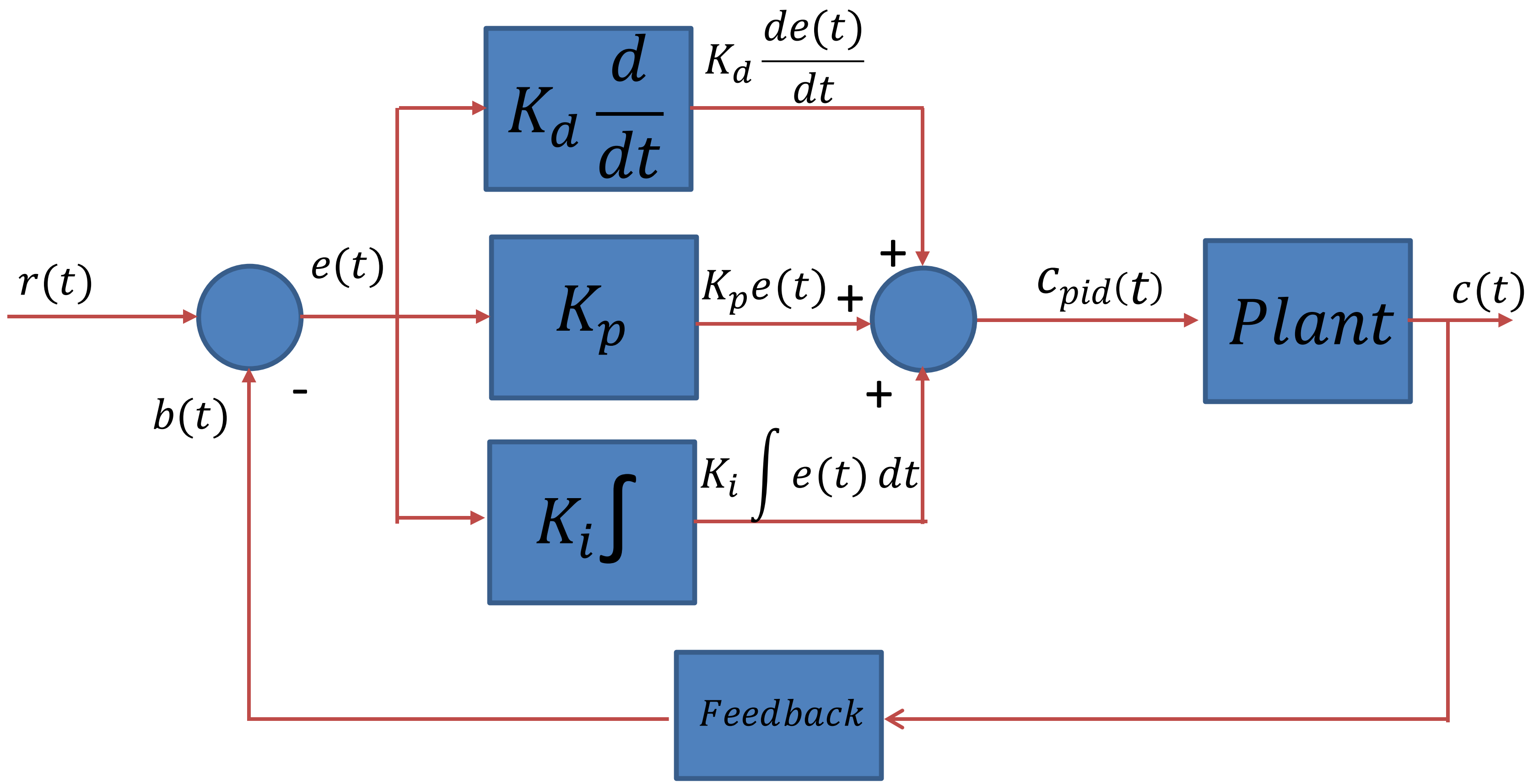
- The higher the error signal rate of change, the sooner the final control element is positioned to the desired value.

- The added derivative action reduces initial overshoot of the measured variable, and therefore aids in stabilizing the process sooner.

- This control mode is called proportional plus derivative (PD) control because the derivative section responds to the rate of change of the error signal



Proportional Plus Integral Plus Derivative Control (PID)



$$c_{pid(t)} = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt}$$

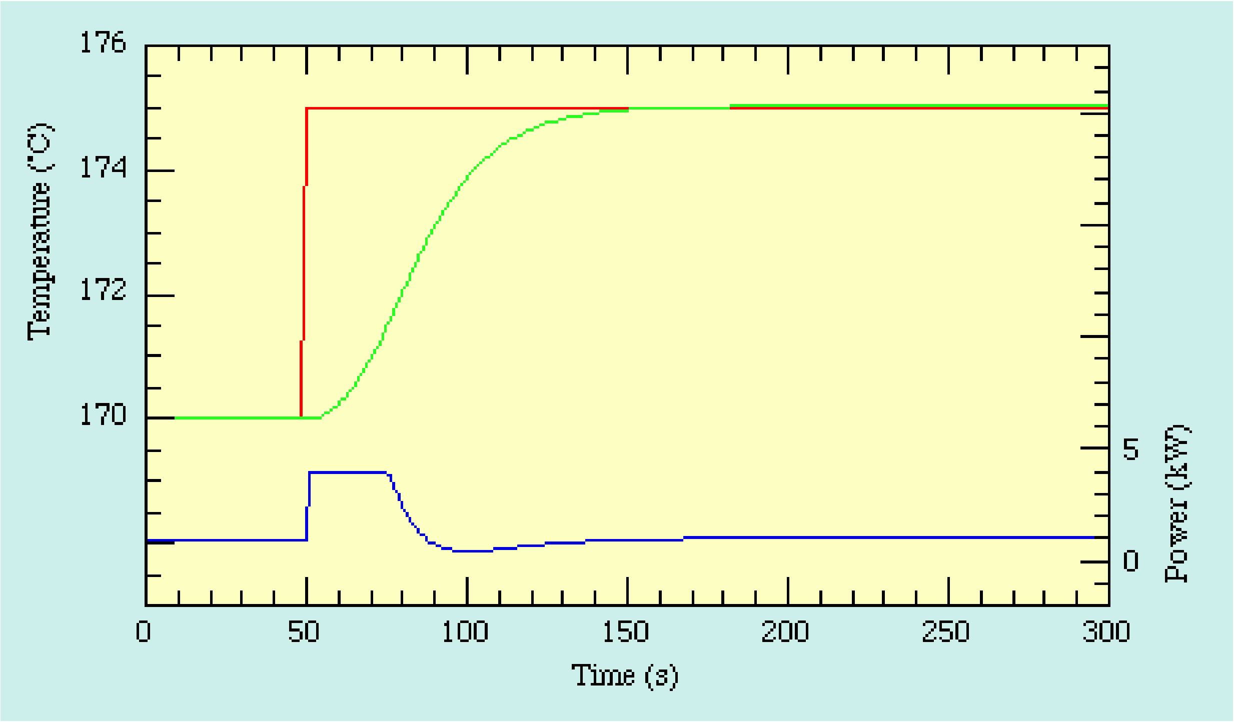
Proportional Plus Integral Plus Derivative Control (PID)

$$c_{pid(t)} = K_p e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt}$$

$$\frac{C_{pid}(s)}{E(s)} = K_p + K_i \frac{1}{s} + K_d\, s$$

Proportional Plus Integral Plus Derivative Control (PID)

- Although PD control deals neatly with the overshoot and ringing problems associated with proportional control it does not cure the problem with the steady-state error. Fortunately it is possible to eliminate this while using relatively low gain by adding an integral term to the control function which becomes



The Characteristics of P, I, and D controllers

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp ↑	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

Tips for Designing a PID Controller

1. Obtain an open-loop response and determine what needs to be improved
2. Add a proportional control to improve the rise time
3. Add a derivative control to improve the overshoot
4. Add an integral control to eliminate the steady-state error
5. Adjust each of K_p, K_i, and K_d until you obtain a desired overall response.

- Lastly, please keep in mind that you do not need to implement all three controllers (proportional, derivative, and integral) into a single system, if not necessary. For example, if a PI controller gives a good enough response (like the above example), then you don't need to implement derivative controller to the system. Keep the controller as simple as possible.

Part-II

PID TUNING RULES

PID Tuning

- The transfer function of PID controller is given as

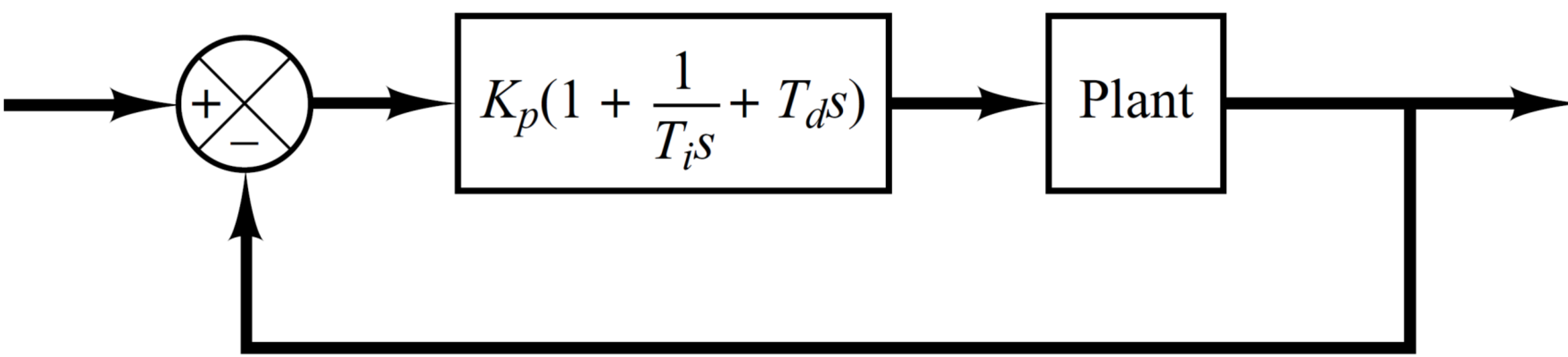
$$\frac{C_{pid}(s)}{E(s)} = K_p + K_i \frac{1}{s} + K_d\, s$$

- It can be simplified as

$$\frac{C_{pid(s)}}{E(s)} = K_p(1 + \frac{1}{T_i s} + T_d\, s)$$

- Where

$$T_i = \frac{K_p}{K_i} \qquad T_d = \frac{K_d}{K_p}$$



PID Tuning

- The process of selecting the controller parameters (K_p, T_i and T_d) to meet given performance specifications is known as controller tuning.

- Ziegler and Nichols suggested rules for tuning PID controllers experimentally.

- Which are useful when mathematical models of plants are not known.

- These rules can, of course, be applied to the design of systems with known mathematical models.

PID Tuning

- Such rules suggest a set of values of K_p, T_i and T_d that will give a stable operation of the system.

- However, the resulting system may exhibit a large maximum overshoot in the step response, which is unacceptable.

- In such a case we need series of fine tunings until an acceptable result is obtained.

- In fact, the Ziegler–Nichols tuning rules give an educated guess for the parameter values and provide a starting point for fine tuning, rather than giving the final settings for K_p, T_i and T_d in a single shot.

Zeigler-Nichol's PID Tuning Methods

- Ziegler and Nichols proposed rules for determining values of the K_p, T_i and T_d based on the transient response characteristics of a given plant.

- Such determination of the parameters of PID controllers or tuning of PID controllers can be made by engineers on-site by experiments on the plant.

- There are two methods called Ziegler–Nichols tuning rules:
  - First method (open loop Method)
  - Second method (Closed Loop Method)

First Method Ziegler Nichols

A linearized quantitative version of a simple plant can be obtained with an open loop experiment, using the following procedure:

1. With the plant in open loop, take the plant manually to a normal operating point. Say that the plant output settles at $y(t) = y_0$ for a constant plant input $u(t) = u_0$.
2. At an initial time, t_0, apply a step change to the plant input, from u_0 to u_∞ (*this should be in the range of 10 to 20% of full scale*).

Cont/...

3. Record the plant output until it settles to the new operating point. Assume you obtain the curve shown on the next slide. This curve is known as the *process reaction curve*.

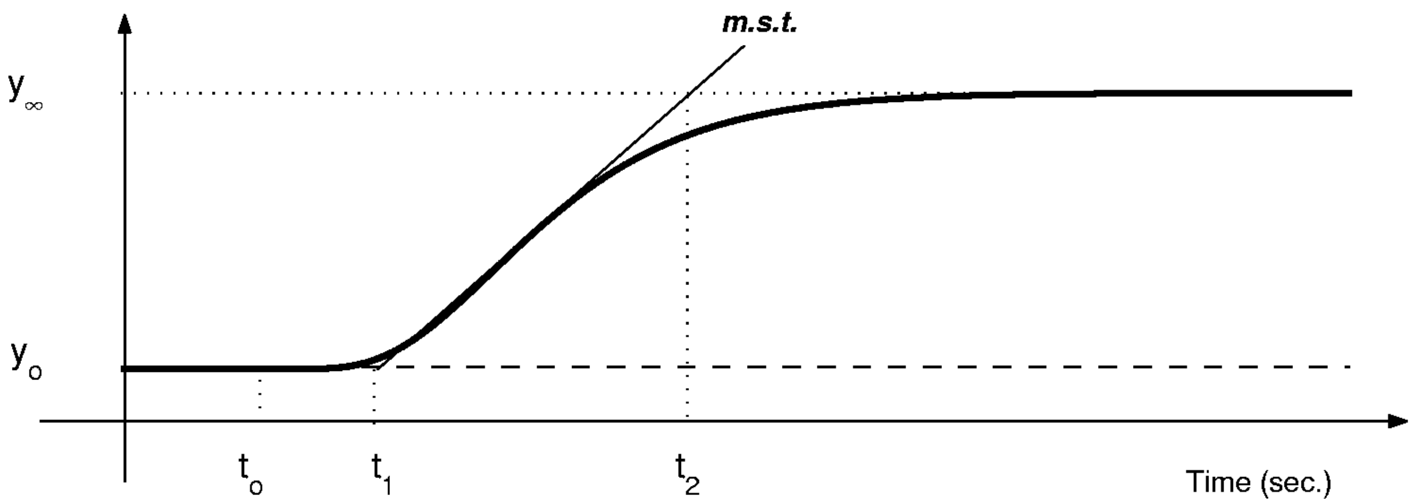
   In Figure 6.6, m.s.t. stands for *maximum slope tangent*.

4. Compute the parameter model as follows

$$K_o = \frac{y_\infty - y_o}{u_\infty - u_o}; \qquad \tau_o = t_1 - t_o; \qquad \nu_o = t_2 - t_1$$

Figure 6.6: *Plant step response*

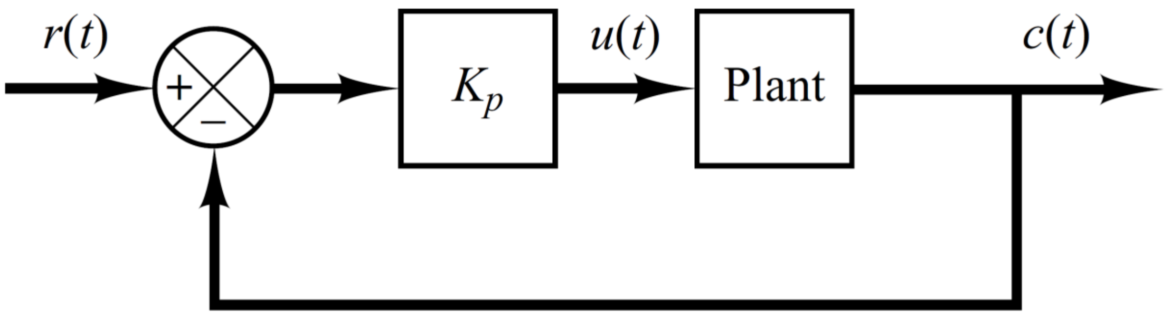
The suggested parameters are shown in Table 6.2.



Zeigler-Nichol's Second Method

- In the second method, we first set $T_i = \infty$ and $T_d = 0$.

- Using the proportional control action only (as shown in figure), increase K_p from 0 to a critical value K_{cr} at which the output first exhibits sustained oscillations.



- If the output does not exhibit sustained oscillations for whatever value K_p may take, then this method does not apply.

Zeigler-Nichol's Second Method

- Thus, the critical gain K_{cr} and the corresponding period P_{cr} are determined.

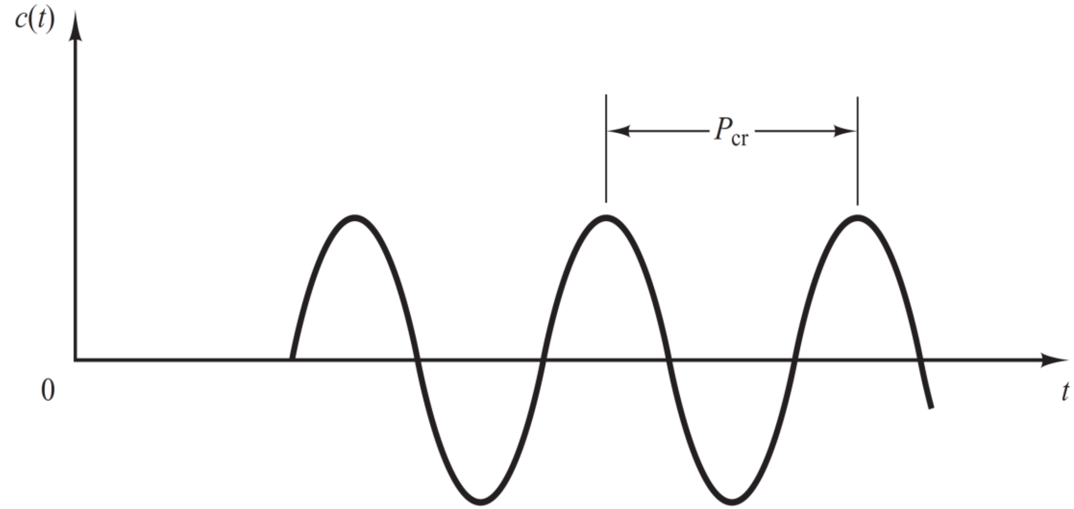
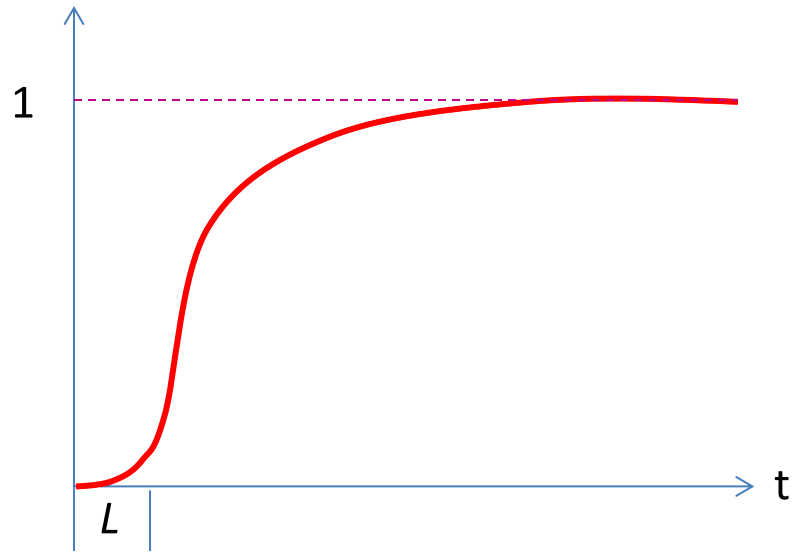


Table-2

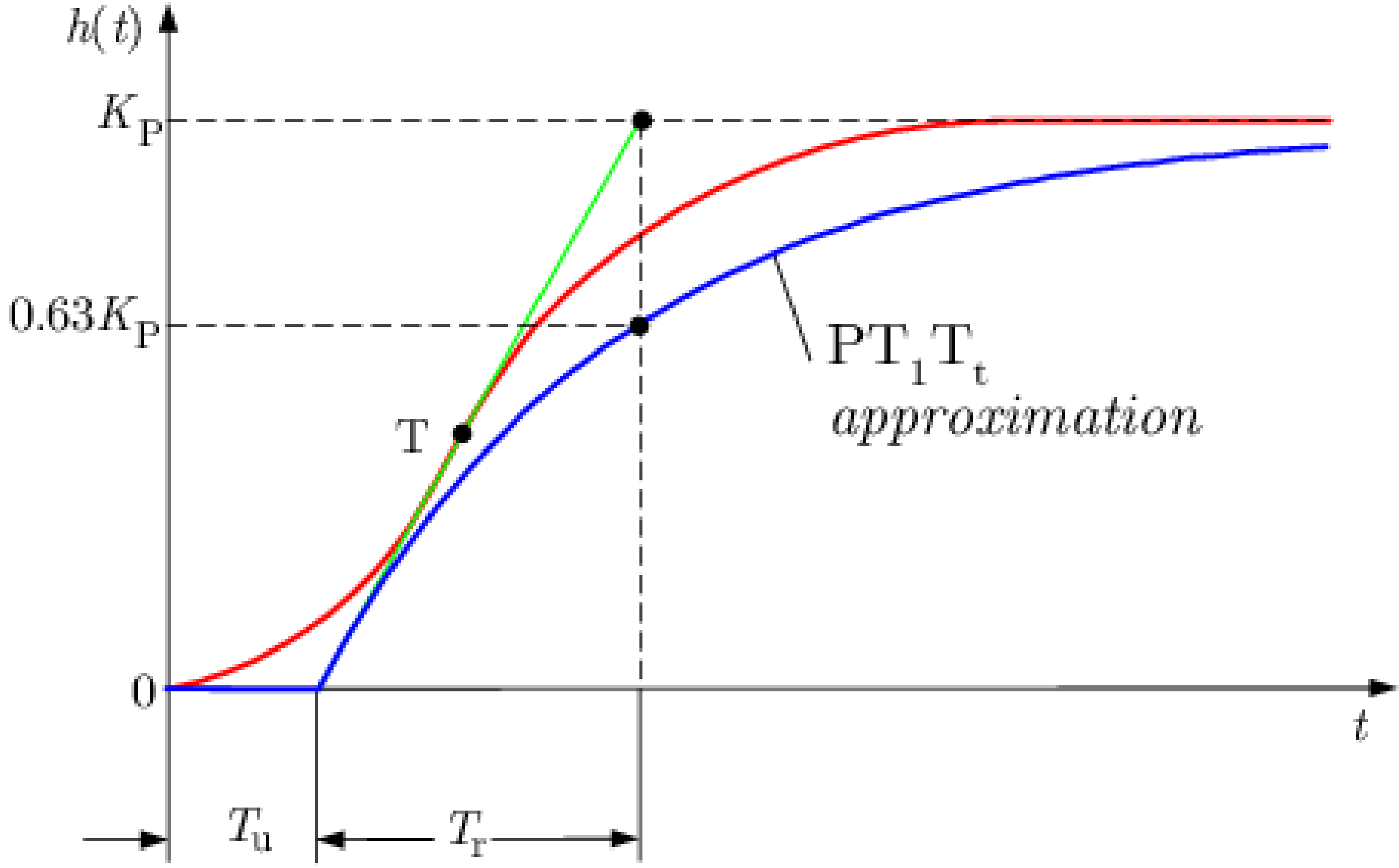
Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Example-1

$$\frac{C(s)}{R(s)} = \frac{K}{Ts+1} e^{-sL}$$

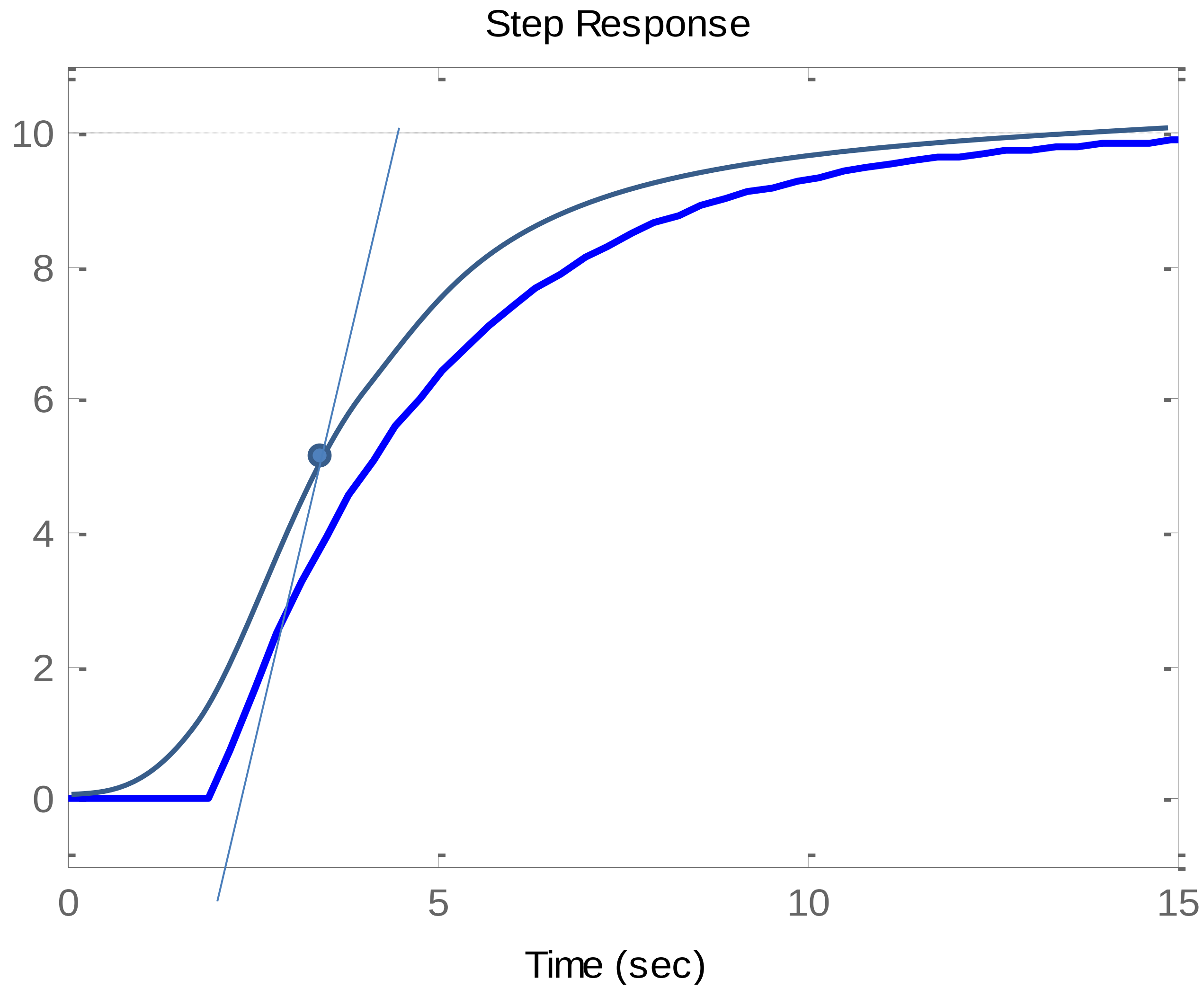


Example-1



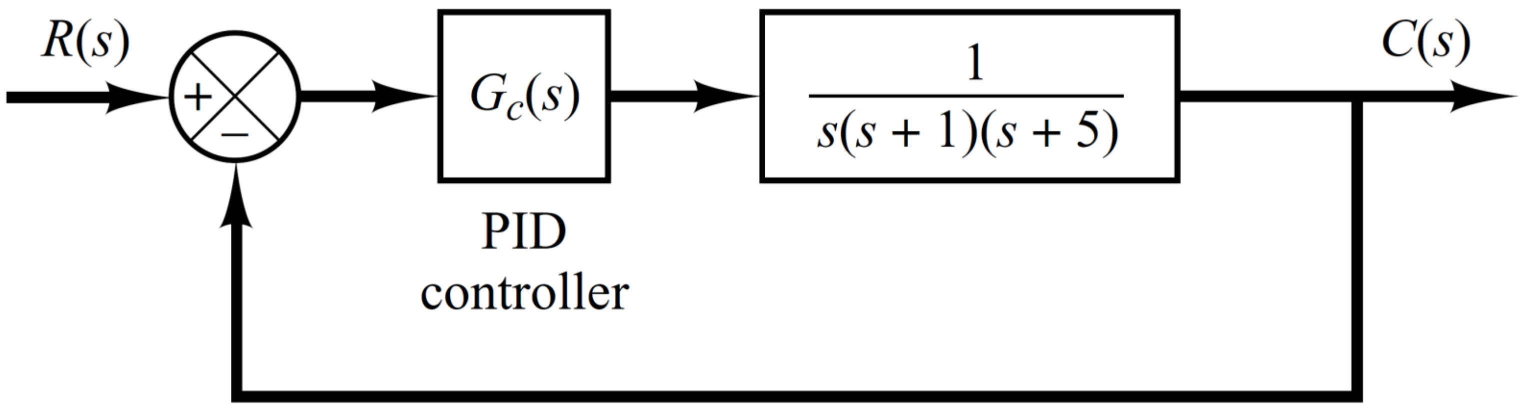
Example-1

$$\frac{C(s)}{R(s)} = \frac{10}{3s+1} e^{-2s}$$



Example-2

- Consider the control system shown in following figure.



- Apply a Ziegler–Nichols tuning rule for the determination of the values of parameters K_p, T_i and T_d.

Example-2

- Transfer function of the plant is

$$G(s) = \frac{1}{s(s+1)(s+5)}$$

- Since plant has an integrator therefore Ziegler-Nichol's first method is not applicable.

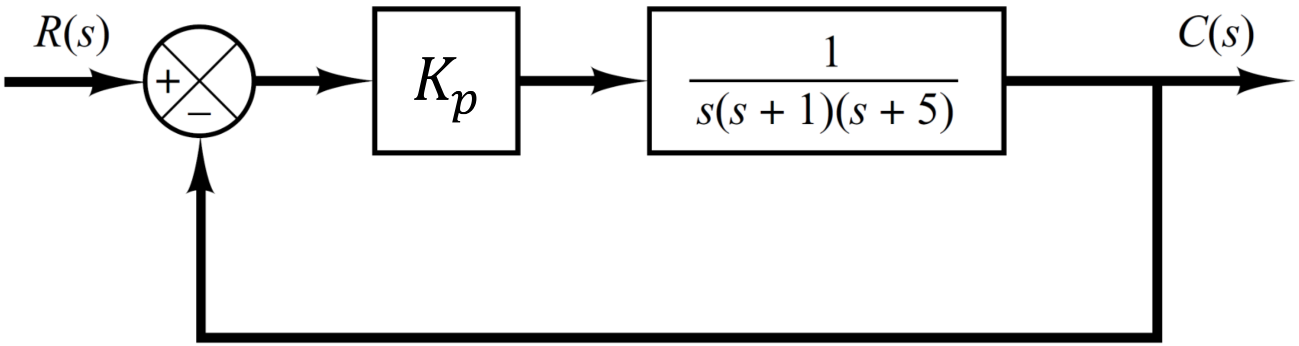
- According to second method proportional gain is varied till sustained oscillations are produced.

- That value of K_c is referred as K_{cr}.

Example-2

- Here, since the transfer function of the plant is known we can find K_{cr} using
  - Root Locus
  - Routh-Herwitz Stability Criterion

- By setting $T_i = \infty$ and $T_d = 0$ closed loop transfer function is obtained as follows.



$$\frac{C(s)}{R(s)} = \frac{K_p}{s(s+1)(s+5)+K_p}$$

Example-2

- The value of K_p that makes the system marginally unstable so that sustained oscillation occurs can be obtained as

$$s^3 + 6s^2 + 5s + K_p = 0$$

- The Routh array is obtained as

$$\begin{array}{lcc} s^3 & 1 & 5 \\ s^2 & 6 & K_p \\ s^1 & \dfrac{30 - K_p}{6} & \\ s^0 & K_p & \end{array}$$

- Examining the coefficients of first column of the Routh array we find that sustained oscillations will occur if $K_p = 30$.

- Thus the critical gain K_{cr} is

$$K_{cr} = 30$$

Example-2

$$\omega = \sqrt{5}\, rad/sec$$

- Hence the period of sustained oscillations P_{cr} is

$$P_{cr} = \frac{2\pi}{\omega}$$

$$P_{cr} = \frac{2\pi}{\sqrt{5}} = 2.8099\ sec$$

- Referring to Table-2

$$K_p = 0.6K_{cr} = 18$$

$$T_i = 0.5P_{cr} = 1.405$$

$$T_d = 0.125P_{cr} = 0.35124$$

Example-2

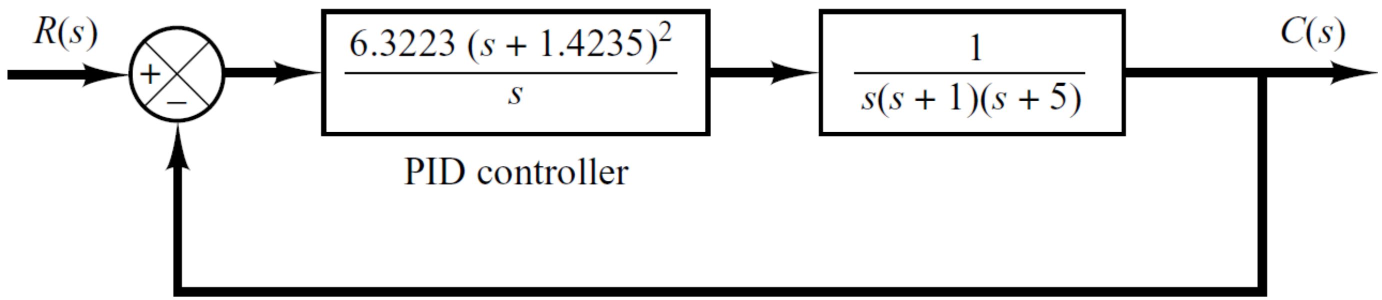
$K_p = 18$ $\quad T_i = 1.405$ $\quad T_d = 0.35124$

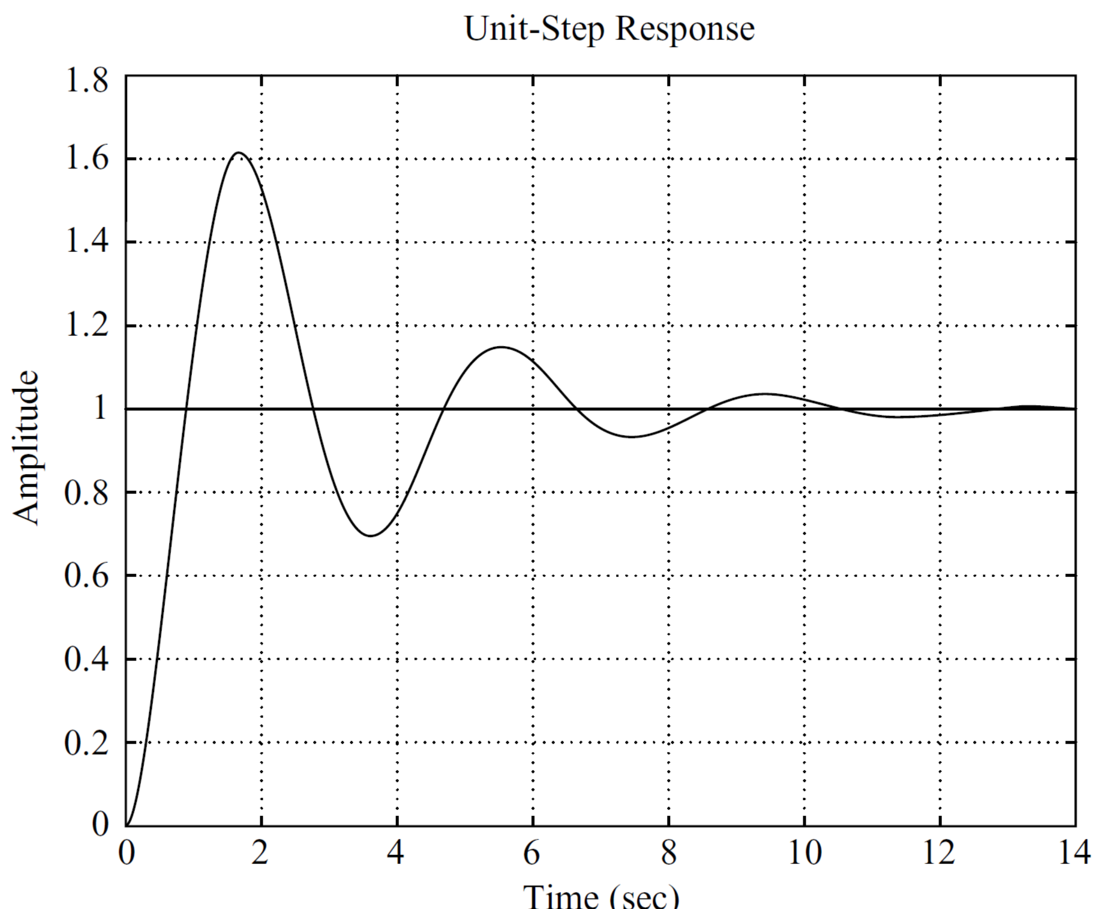
- Transfer function of PID controller is thus obtained as

$$G_c(s) = K_p(1 + \frac{1}{T_i s} + T_d\, s)$$

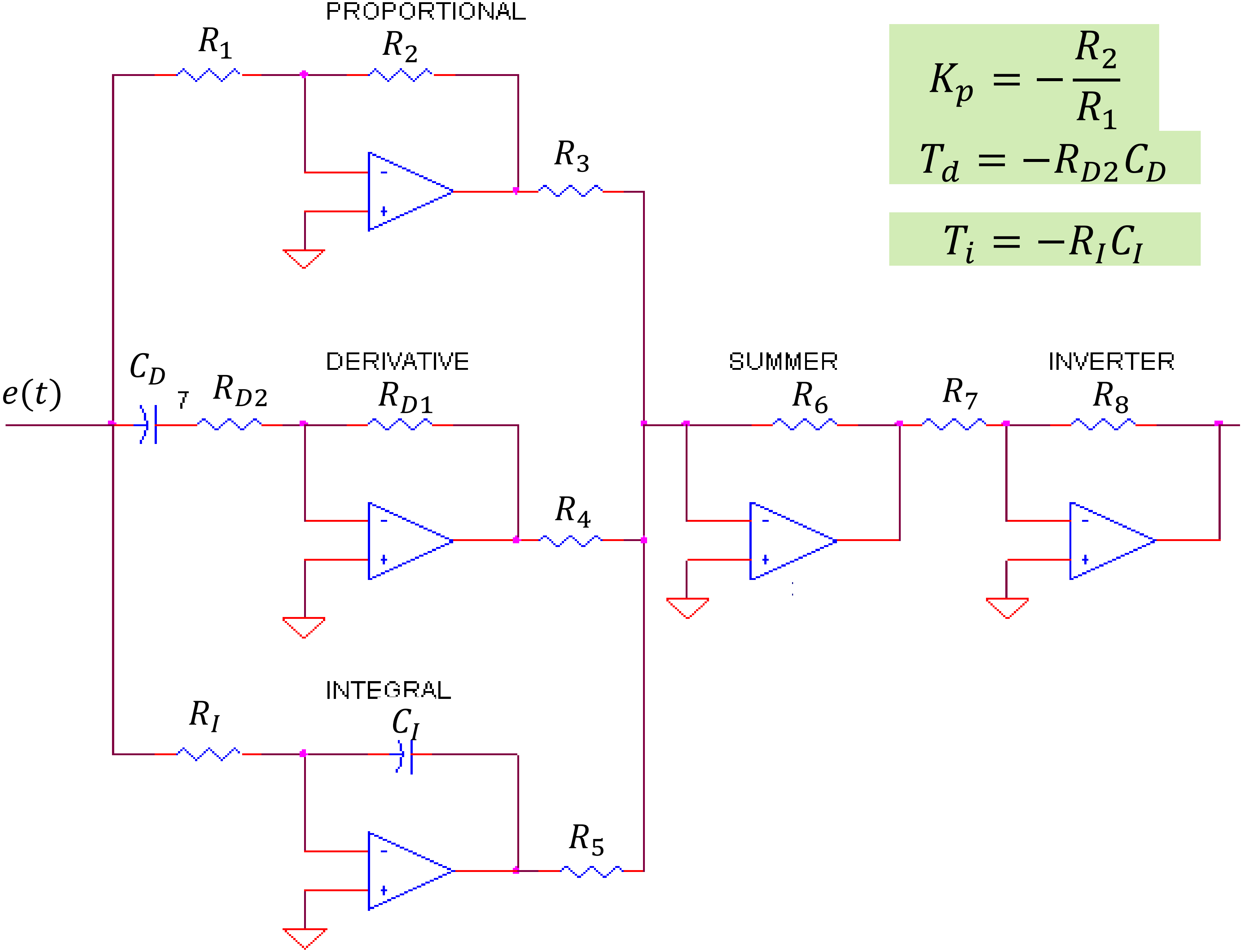
$$G_c(s) = 18(1 + \frac{1}{1.405s} + 0.35124s)$$

Example-2


R(s)
+
−
6.3223 (s + 1.4235)²
s
1
s(s + 1)(s + 5)
C(s)
PID controller


Unit-Step Response
Amplitude
Time (sec)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14

Electronic PID Controller

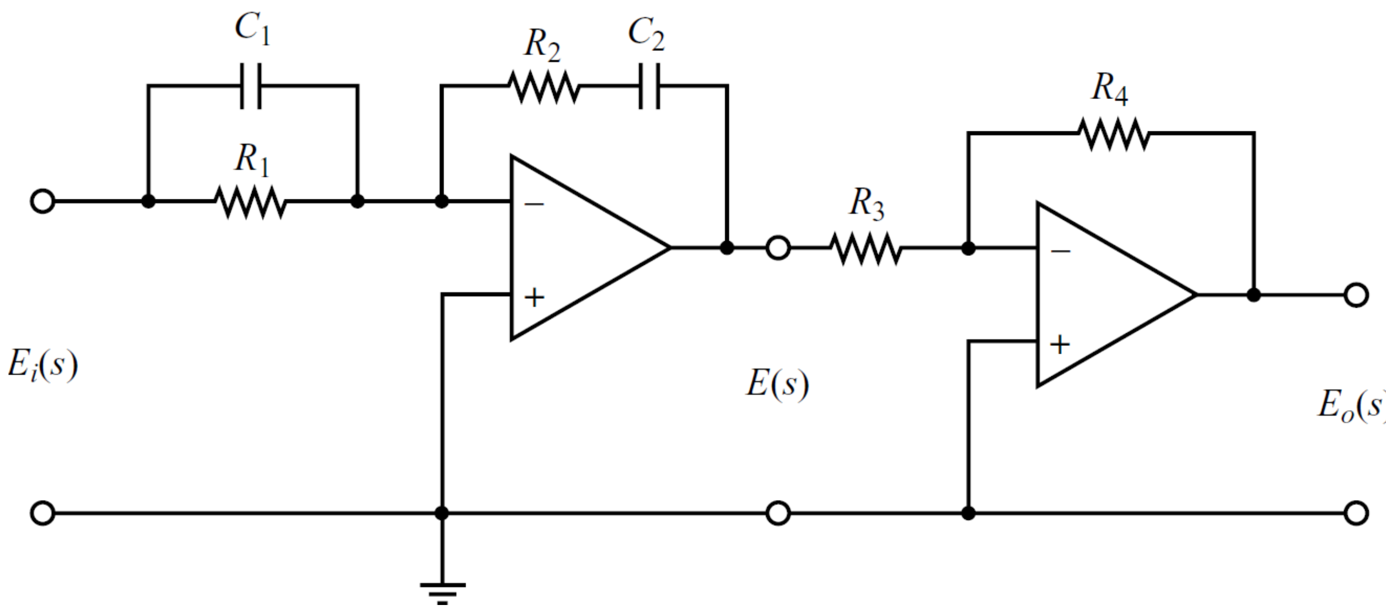


$$K_p = -\frac{R_2}{R_1}$$

$$T_d = -R_{D2}C_D$$

$$T_i = -R_I C_I$$

Electronic PID Controller



$$\frac{E_o(s)}{E_i(s)} = \frac{R_4}{R_3}\,\frac{(R_1C_1s+1)(R_2C_2s+1)}{R_2C_2s}$$

$$\frac{E_o(s)}{E_i(s)} = \frac{R_4R_2}{R_3R_1}\left(\frac{R_1C_1+R_2C_2}{R_2C_2} + \frac{1}{R_2C_2s} + R_1C_1s\right)$$

Electronic PID Controller

$$\frac{E_o(s)}{E_i(s)} = \frac{R_4 R_2}{R_3 R_1}\left(\frac{R_1 C_1 + R_2 C_2}{R_2 C_2} + \frac{1}{R_2 C_2 s} + R_1 C_1 s\right)$$

$$\frac{E_o(s)}{E_i(s)} = \frac{R_4(R_1 C_1 + R_2 C_2)}{R_3 R_1 C_2}\left[1 + \frac{1}{(R_1 C_1 + R_2 C_2)s} + \frac{R_1 C_1 R_2 C_2}{R_1 C_1 + R_2 C_2} s\right]$$

$$K_p = \frac{R_4(R_1 C_1 + R_2 C_2)}{R_3 R_1 C_2} \qquad T_i = R_1 C_1 + R_2 C_2 \qquad T_d = \frac{R_1 C_1 R_2 C_2}{R_1 C_1 + R_2 C_2}$$

- In terms of K_p, K_i, K_d we have

$$K_p = \frac{R_4(R_1 C_1 + R_2 C_2)}{R_3 R_1 C_2} \qquad K_i = \frac{R_4}{R_3 R_1 C_2} \qquad K_d = \frac{R_4 R_2 C_1}{R_3}$$

PID implementation using Arduino: Method 1

In the s-domain the PID controller has the following form

$$U(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right)E(s) \tag{1}$$

where $U(s)$ is the control action that is sent to the actuator, E(s) is the control error defined by

$$E(s) = Y_r(s) - Y(s) \tag{2}$$

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\dot{e}(t)\right) \tag{3}$$

$$e(t) = y_r(t) - y(t) \tag{4}$$

$$u(t) = K\big(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\dot{e}(t)\big) \quad (3)$$

Take derivative of both sides

$$\dot{u}(t) = K\dot{e}(t) + \frac{K}{T_i}e(t) + KT_d\ddot{e}(t) \quad (6)$$

$$\dot{u}(t) \approx \frac{u_k - u_{k-1}}{h} \quad (7)$$

Similarly, we approximate the first derivative of the control error

$$\dot{e}(t) \approx \frac{e_k - e_{k-1}}{h} \tag{8}$$

The second derivative of the control error is approximated as follows

$$\ddot{e}(t) \approx \frac{\dot{e}_k - \dot{e}_{k-1}}{h} \tag{9}$$

By substituting \eqref{firstDerivativeApproximationError} for the time indices k and $k-1$, we obntain

$$\ddot{e}(t) \approx \frac{e_k - 2e_{k-1} + e_{k-2}}{h^2} \tag{10}$$

$$u_k = u_{k-1} + K_0 e_k + K_1 e_{k-1} + K_2 e_{k-2} \tag{11}$$

where the constants K_0, K_1, and K_2 are determined as follows

$$\begin{aligned} K_0 &= K\left(1 + \frac{h}{T_i} + \frac{T_d}{h}\right) \\ K_1 &= -K(1 + \frac{2T_d}{h}) \\ K_2 &= \frac{KT_d}{h} \end{aligned} \tag{12}$$

```
//sensor parameters

int distanceSensorPin = A0;     // distance sensor pin
float Vr=5.0;                   // reference voltage for A/D conversion
float sensorValue = 0;          // raw sensor reading
float sensorVoltage = 0;        // sensor value converted to volts
float k1=16.7647563;            // sensor parameter fitted using the least-squar
float k2=-0.85803107;           // sensor parameter fitted using the least-squar
float distance=0;               // distance in cm
int noMeasurements=200;          // number of measurements for averaging the dis
float sumSensor;                // sum for computing the average raw sensor valu

// motor parameters
#include <Servo.h>
Servo servo_motor;
int servoMotorPin = 9;          // the servo motor is attached to the 9th Pulse


// control parameters
float desiredPosition=35;       // desired position of the ball
float errorK;                   // position error at the time instant k
float errorKm1=0;               // position error at the time instant k-1
float errorKm2=0;               // position error at the time instant k-2
float controlK=0;               // control signal at the time instant k
float controlKm1=0;             // control signal at the time instant k-1
int delayValue=0;               // additional delay in [ms]

float Kp=0.2;                          // proportional control
float Ki=10;                           // integral control
float Kd=0.4;                          // derivative control
float h=(delayValue+32)*0.001;         // discretization constant, that is equal

float keK=Kp*(1+h/Ki+Kd/h);                 // parameter that multiplies the err
float keKm1=-Kp*(1+2*Kd/h);                 // parameter that multiplies the err
float keKm2=Kp*Kd/h;                        // parameter that multiplies the err
```

```
void setup()
{
  Serial.begin(9600);
  servo_motor.attach(servoMotorPin);
}
```

```
void loop()
{
  unsigned long startTime = micros(); // this is used to measure the time it t
  // obtain the sensor measurements
  sumSensor=0;

  // this loop is used to average the measurement noise
  for (int i=0; i<noMeasurements; i++)
  {
    sumSensor=sumSensor+float(analogRead(distanceSensorPin));
  }
  sensorValue=sumSensor/noMeasurements;
  sensorVoltage=sensorValue*Vr/1024;
  distance = pow(sensorVoltage*(1/k1), 1/k2); // final value of the distance m


  errorK=desiredPosition-distance; // error at the time instant k;

  // compute the control signal
  controlK=controlKm1+keK*errorK+keKm1*errorKm1+keKm2*errorKm2;

  // update the values for the next iteration
  controlKm1=controlK;
  errorKm2=errorKm1;
  errorKm1=errorK;

  servo_motor.write(94+controlK); // the number 94 is the control action neces
 // Serial.println((String)"Control:"+controlK+(String)"---Error:"+errorK);

 // these three lines are used to plot the data using the Arduino serial plott
  Serial.print(errorK);
  Serial.print(" ");
  Serial.println(controlK);
  unsigned long endTime = micros();
  unsigned long deltaTime=endTime-startTime;
 // Serial.println(deltaTime);

 // delay(delayValue); // uncomment this to introduce an additional delay

}
```

Uncategorized

META

Log in

Entries feed

Comments feed

WordPress.org

Method II

Implementing PID controller using Arduino

Now, I'll be going over how to implement a PID controller in code on the Arduino. The mathematical equation written here is a controller expressed in continuous time or in the analog domain.

$$u = \underbrace{K_p e}_{\substack{\text{Proportional}\\ \text{Term}}} + \underbrace{K_i \int_0^t e\,dt}_{\substack{\text{Integral}\\ \text{Term}}} + \underbrace{K_d \frac{d}{dt} e}_{\substack{\text{Differential}\\ \text{Term}}}$$

Now studying the controller in the continuous or analog domain makes it easier for us to realize what is going on. But most controllers these days are implemented digitally or with microcontroller like Arduino in software. So we want to implement this PID controller on the Arduino. We are going to have to convert it to the discrete time or digital domain as we can see here.

$$u[n] = Kp^{*}e[n] + Ki^{*}\sum_{k=0}^{n} e[k]\,T + Kd^{*}\frac{(e[n] - e[n-1])}{T}$$

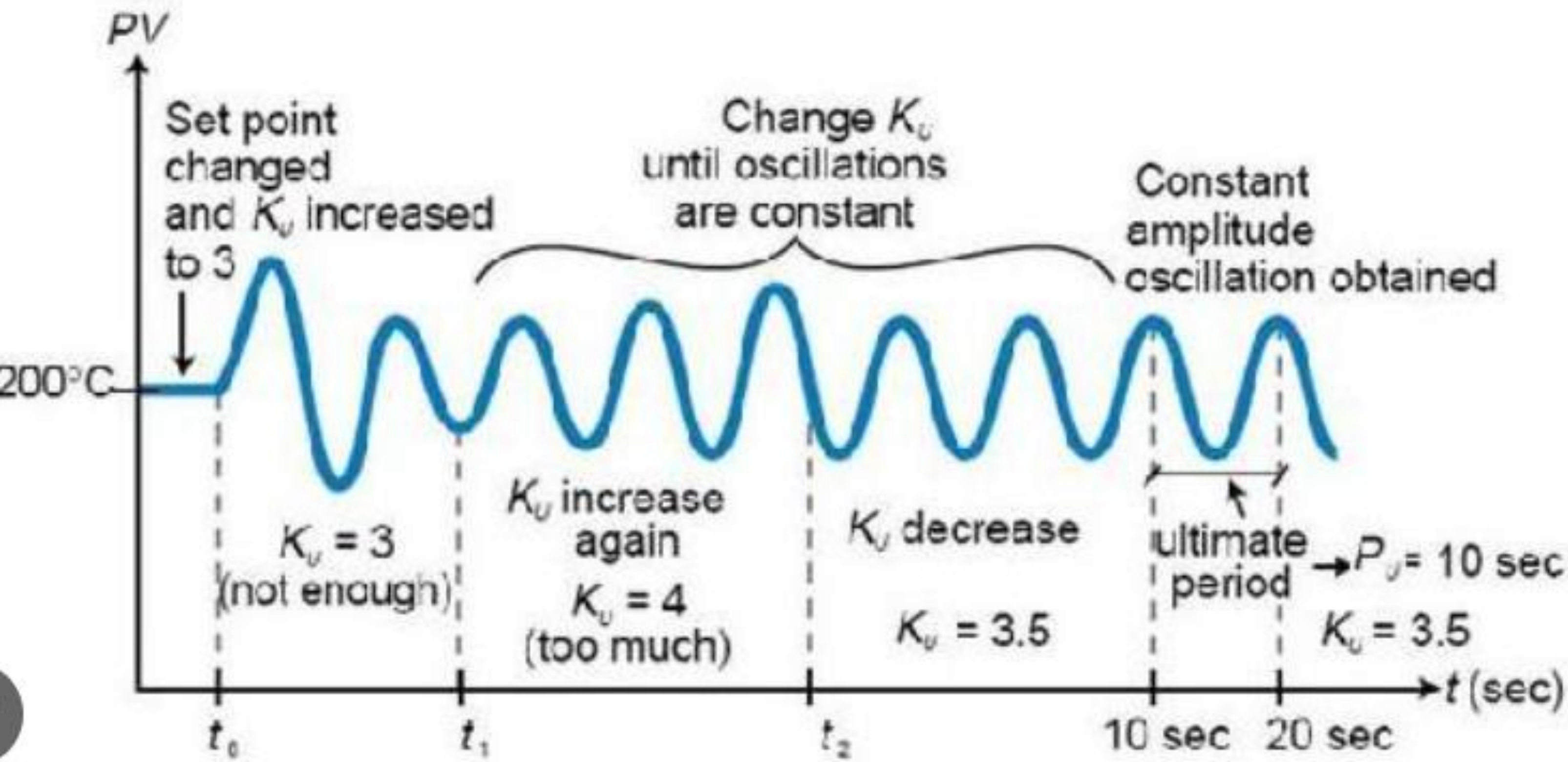
```
double sensed_output, control_signal;
double setpoint;
double Kp; //proportional gain
double Ki; //integral gain
double Kd; //derivative gain
int T; //sample time in milliseconds (m
unsigned long last_time;
double total_error, last_error;
int max_control;
int min_control;
```

```
void setup(){

}
```

```
void loop(){

PID_Control(); //calls the PID function every T interval and outputs a control signal
```

```
void PID_Control(){

unsigned long current_time = millis(); //returns the number of milliseconds passed since the

int delta_time = current_time - last_time; //delta time interval

if (delta_time >= T){

double error = setpoint - sensed_output;

total_error += error; //accumalates the error - integral term
if (total_error >= max_control) total_error = max_control;
else if (total_error <= min_control) total_error = min_control;

double delta_error = error - last_error; //difference of error for derivative term

control_signal = Kp*error + (Ki*T)*total_error + (Kd/T)*delta_error; //PID control compute
if (control_signal >= max_control) control_signal = max_control;
else if (control_signal <= min_control) control_signal = min_control;

last_error = error;
last_time = current_time;
}
}
```

Tuning example:



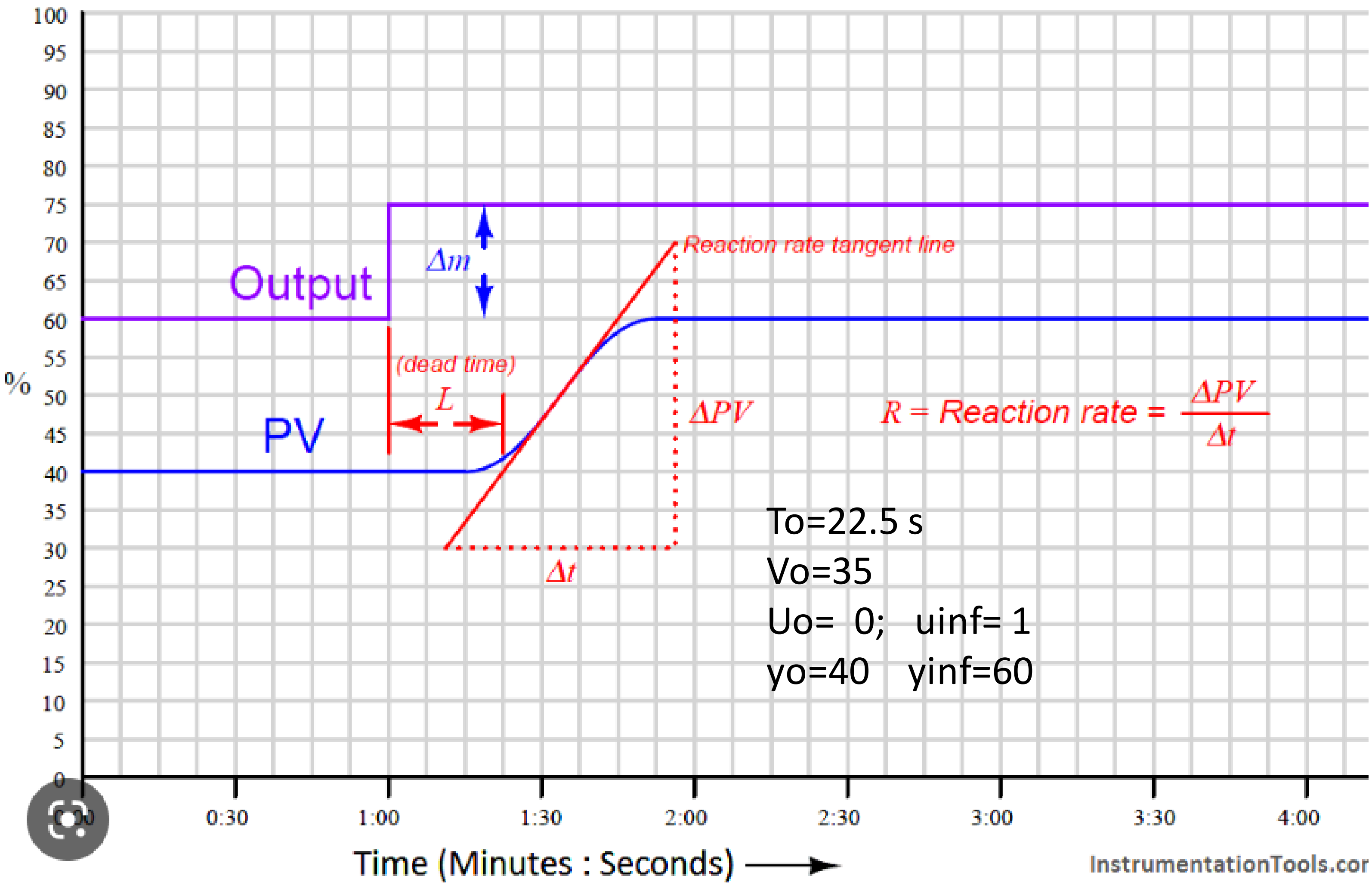
Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Now select the required controller from table based on the question.
For example if the required is PI then we select the second row

Kp=0.45 *Kcr =0.45 *3.5
Ti=1/1.2 * Pcr= 1/1.2 * 10

By yourself solve the same example if PID is required not PI

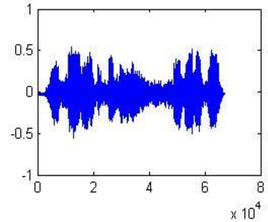
Tuning example II

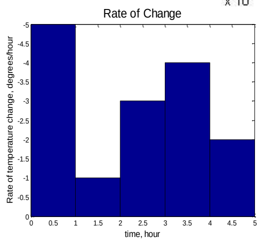


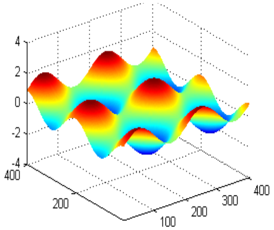
	$\mathbf{K_p}$	$\mathbf{T_r}$	$\mathbf{T_d}$
P	$\frac{\nu_o}{K_o\tau_o}$		
PI	$\frac{0.9\nu_o}{K_o\tau_o}$	$3\tau_o$	
PID	$\frac{1.2\nu_o}{K_o\tau_o}$	$2\tau_o$	$0.5\tau_o$

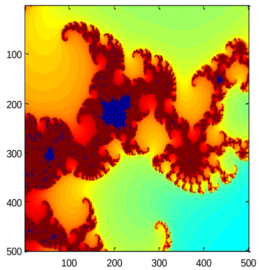
Ko=(60-40)/(1-0)=20

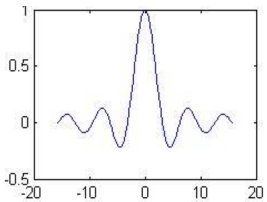
If we select PID to implement
Kp=1.2 * 35/(20*22.5) ; Tr=2 *22.5;
Td=0.5*22.5

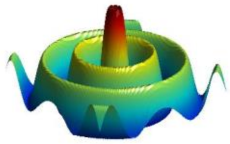






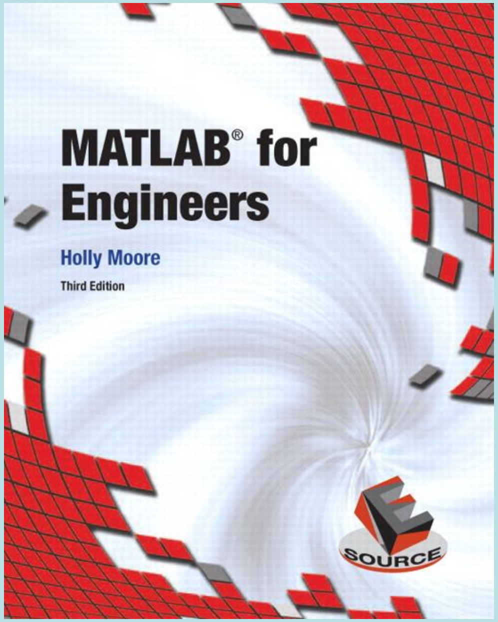


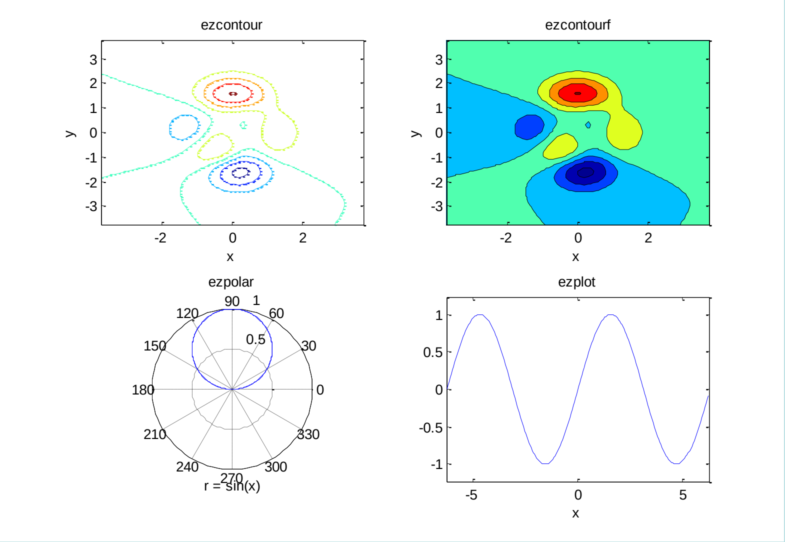


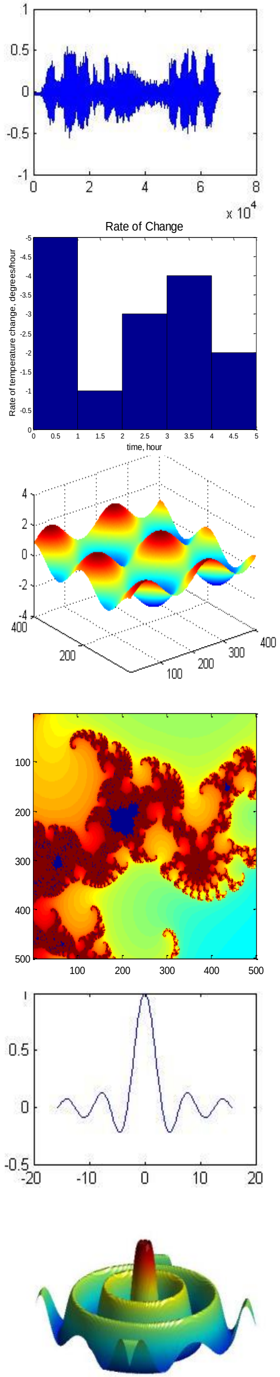


Symbolic Mathematics

Chapter 12

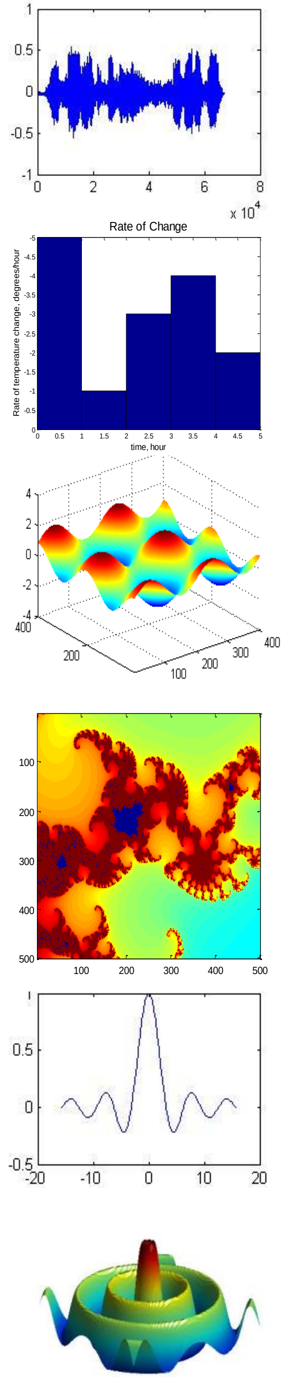


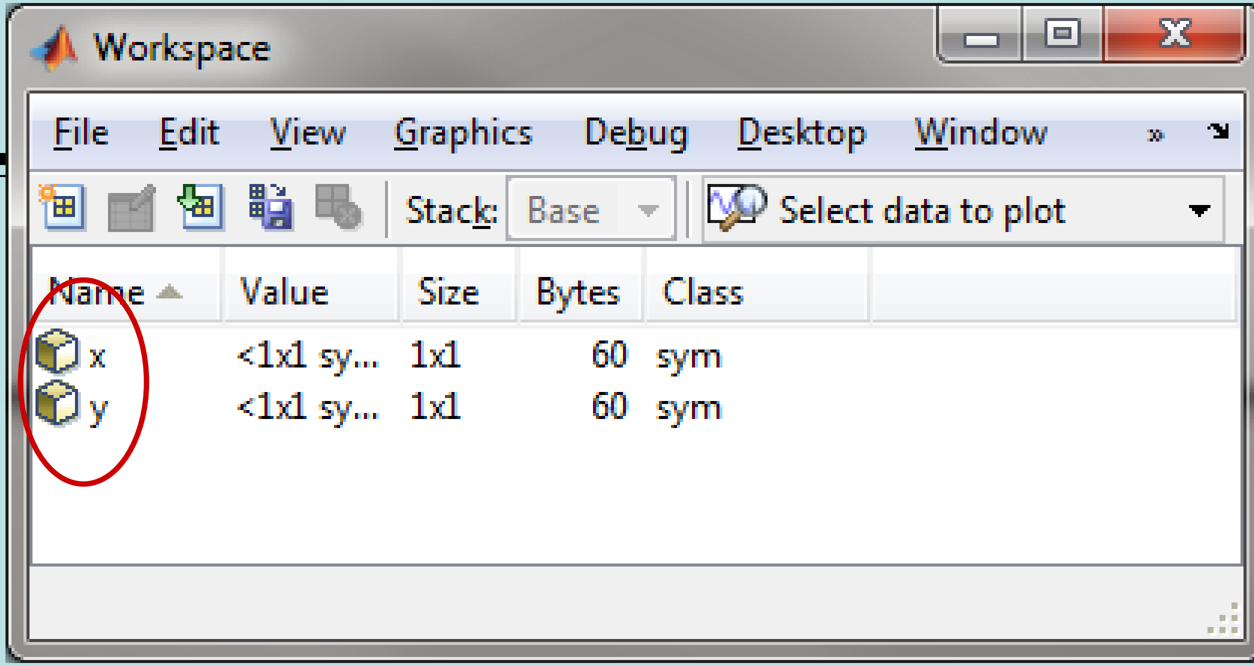




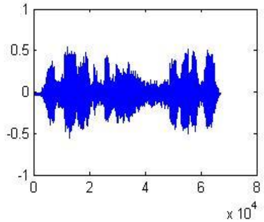
Defining variables

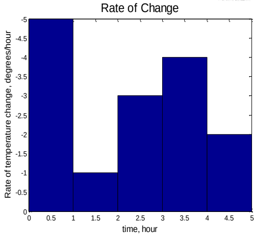
- Define x as a symbolic variable
  - **x=sym('x')** or
  - **syms x**
- Use x to create a more complicated expression
  - **y = 2*(x+3)^2/(x^2+6*x+9)**

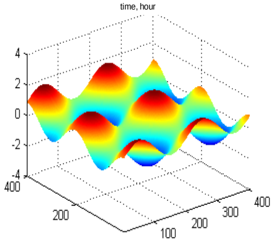


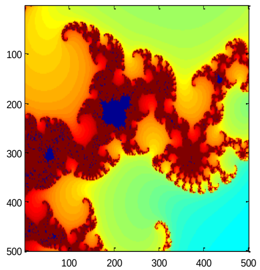


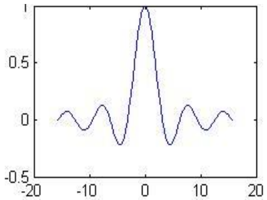
x and y are both symbolic variables

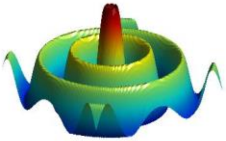








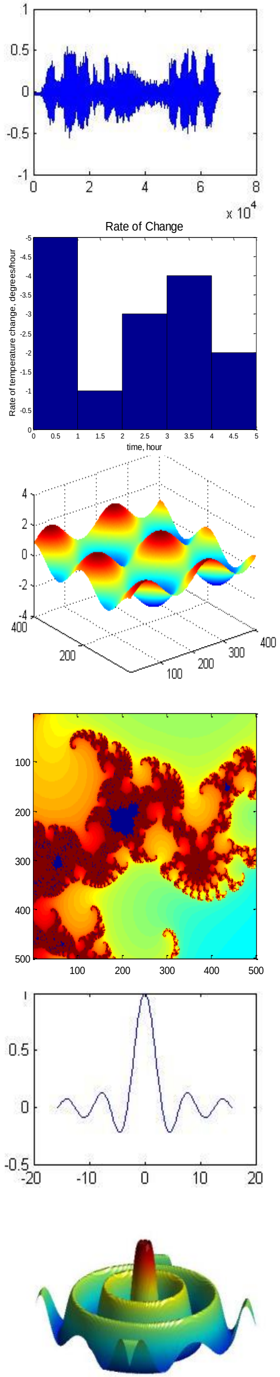




The syms command can create multiple variables

- **syms Q R T k0**
- Use these variables to create another symbolic variables

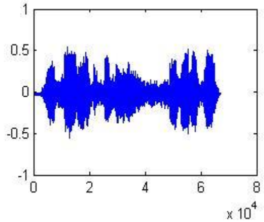
Notice that we used standard algebraic operators – the array operators (.*, ./ and .^) are not used in symbolic algebra

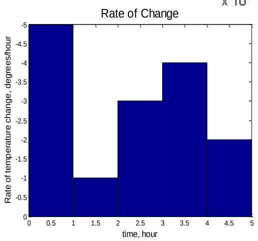


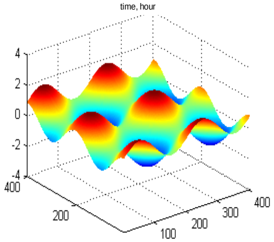
Reserved Variable Names

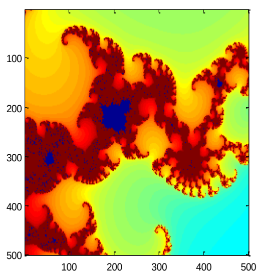
One idiosyncrasy of the implementation of MuPad inside MATLAB is that a number of commonly used variables are reserved. They can be overwritten, however it you try to use them inside expressions or equations you may run into problems.

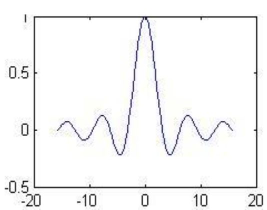
D, E, I, O, beta, zeta, theta, psi, gamma, Ci, Si, Ei

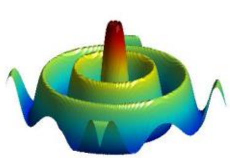










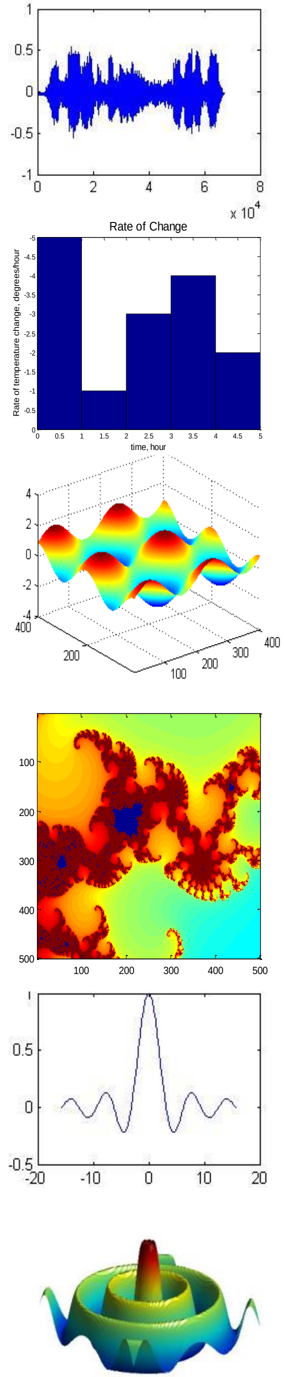


Simplifying equation

$$y = \frac{2*(x+3)^2}{x^2+6x+9}$$

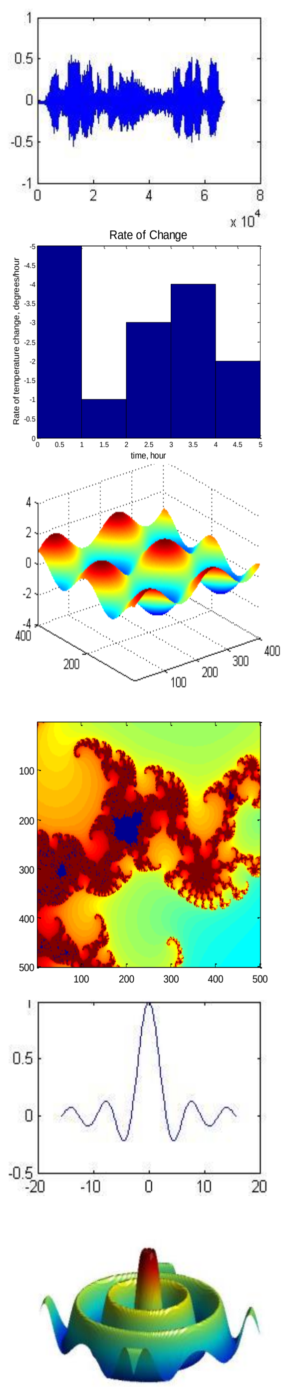
Consider this equation

$$y = \frac{2*(x+3)^2}{x^2+6x+9} = \frac{2*(x^2+6x+9)}{(x^2+6x+9)} = 2$$



In MATLAB

```
>> syms x
>> y = 2 * (x+3)^2 / (x^2 + 6 *x+ 9)

y =

(2*(x + 3)^2)/(x^2 + 6*x + 9)

>> simplify(y)

ans =

2
```



Inversing functions in MATLAB

$$k = k_0 e^{-Q/RT}$$

$$\ln(k) = \ln(k_0) - \frac{Q}{RT}$$

$$\ln\left(\frac{k}{k_0}\right) = -\frac{Q}{RT}$$

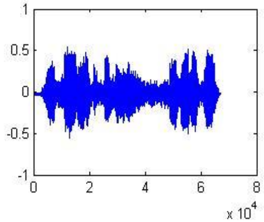
$$\ln\left(\frac{k_0}{k}\right) = \frac{Q}{RT}$$

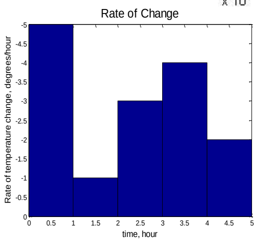
$$T = \frac{Q}{R\ln(k_0/k)}$$

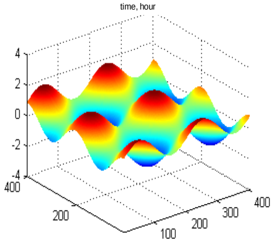
If we know
- k_0
- Q
- R
- T

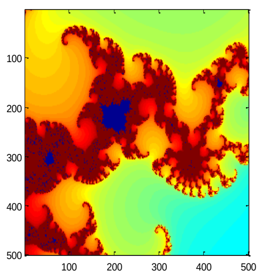
Its easy to solve for k

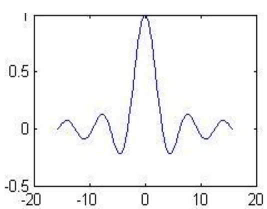
It's not easy to solve for T!

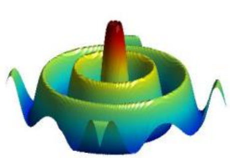












IN MATLAB

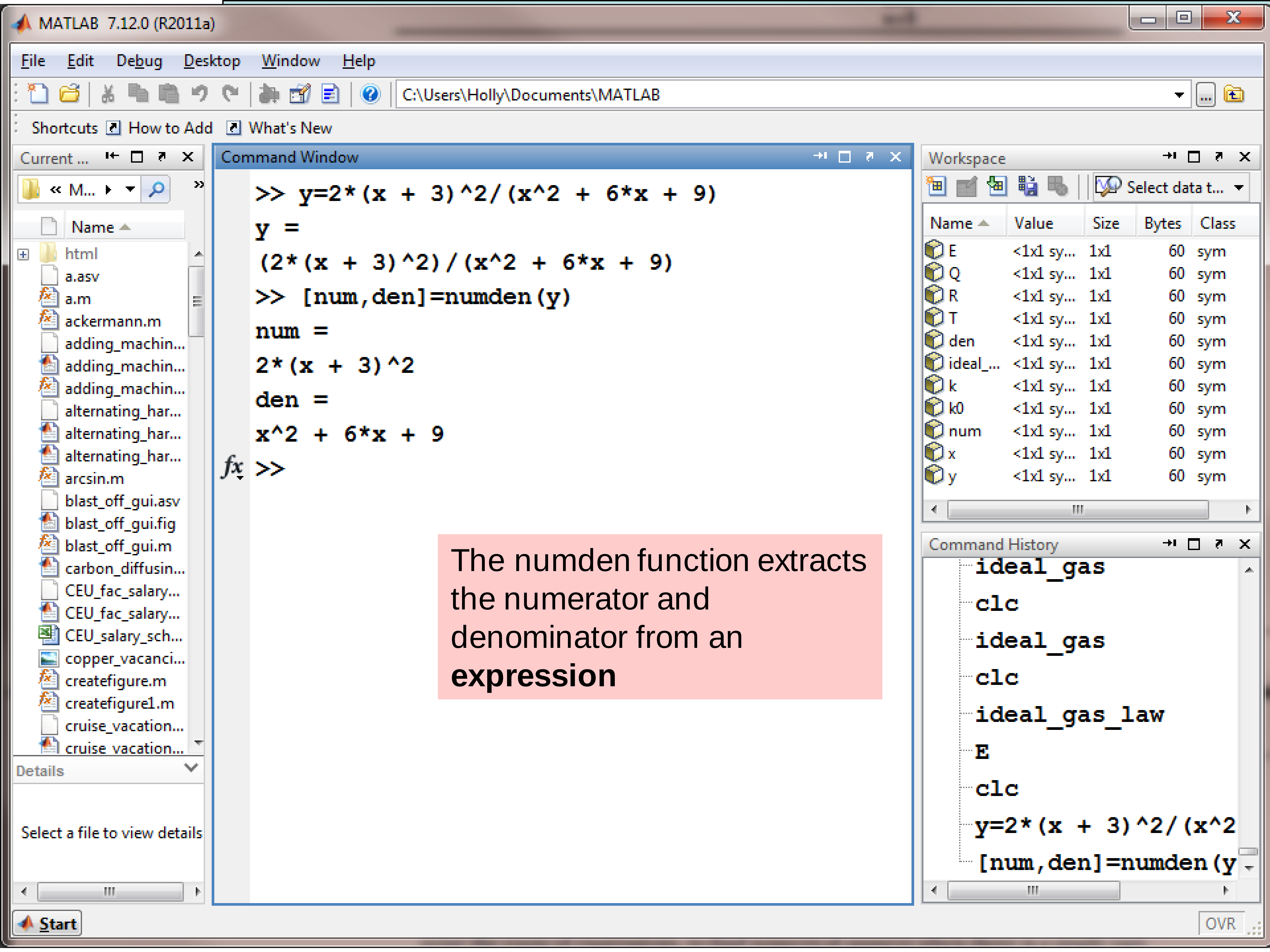
```
>> clear all
>> syms ko Q T R
>> k=ko * exp(-Q/(R *T))

k =

ko*exp(-Q/(R*T))

>> g=finverse(k,T)

g =

-Q/(R*log(T/ko))
```

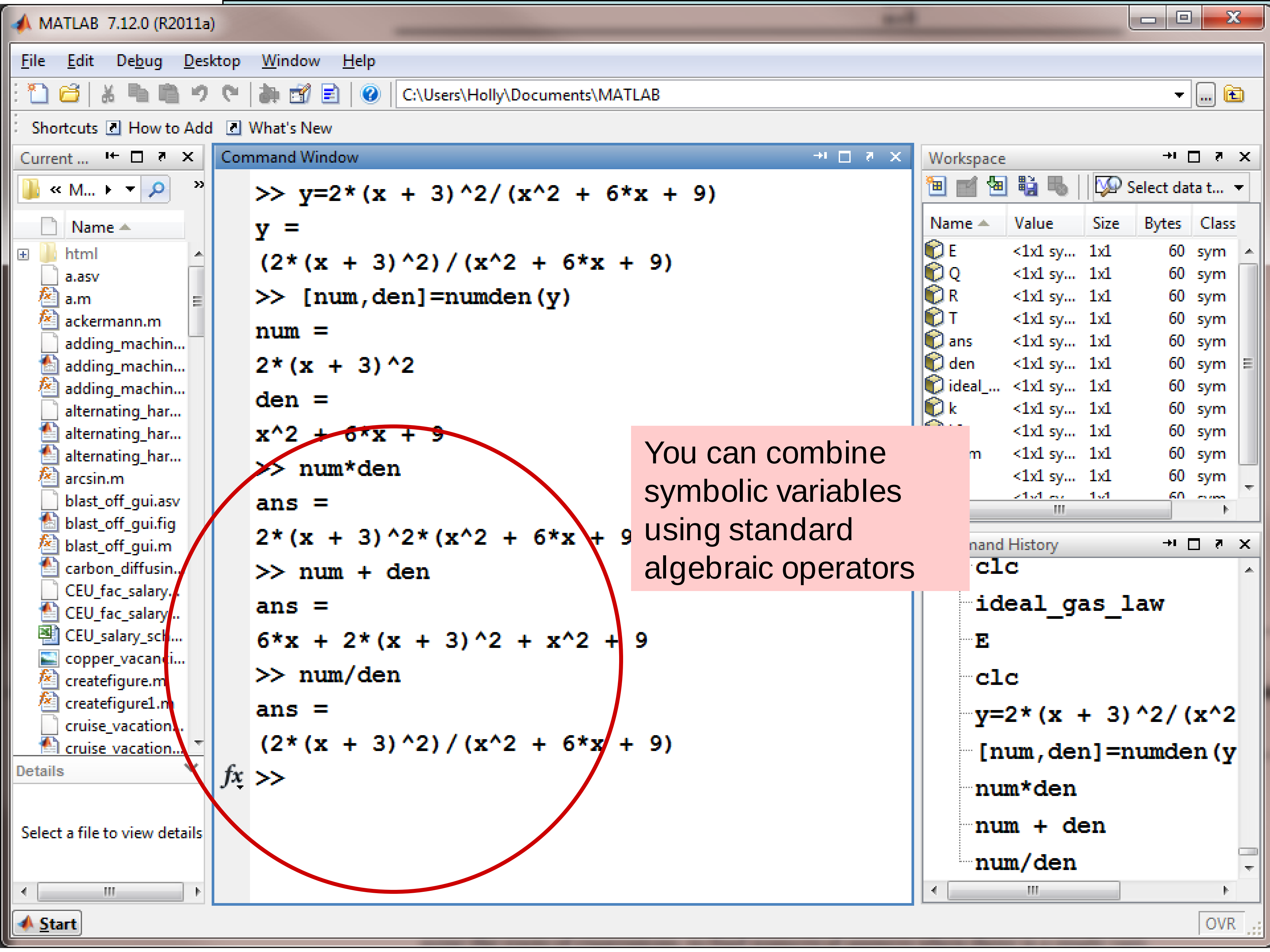
We use finverse(

To invert the function

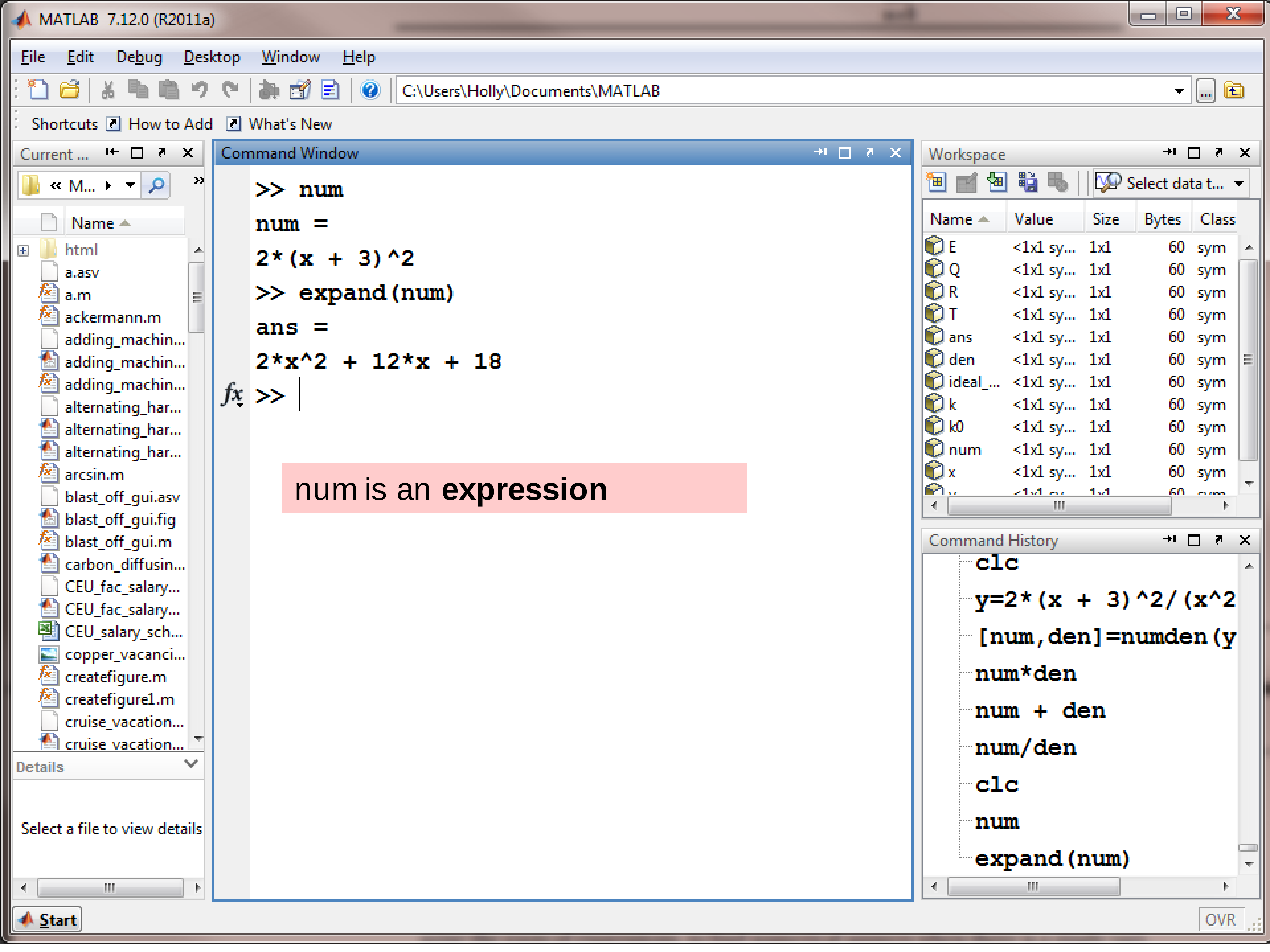
However you need to place k

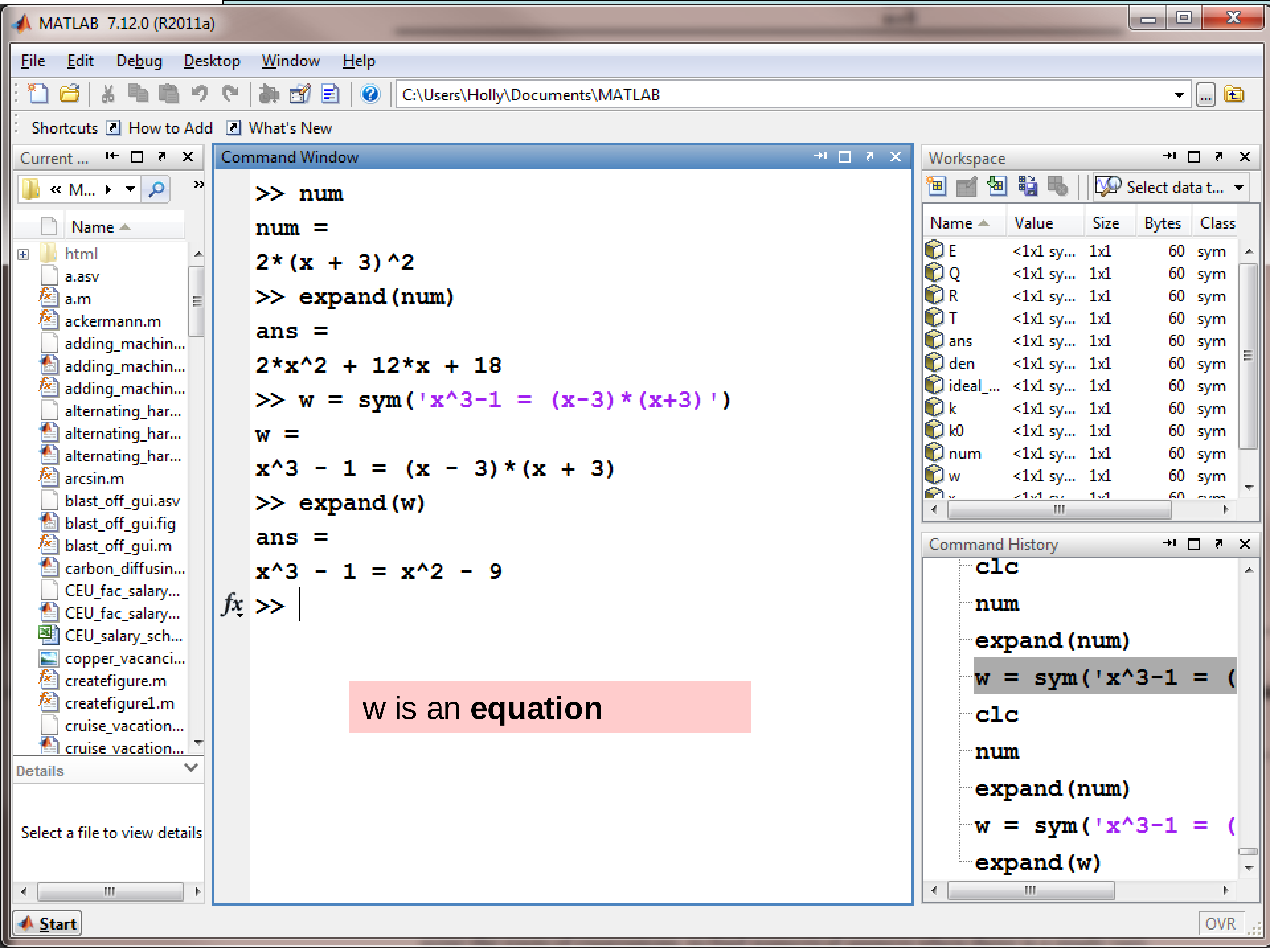
In T place

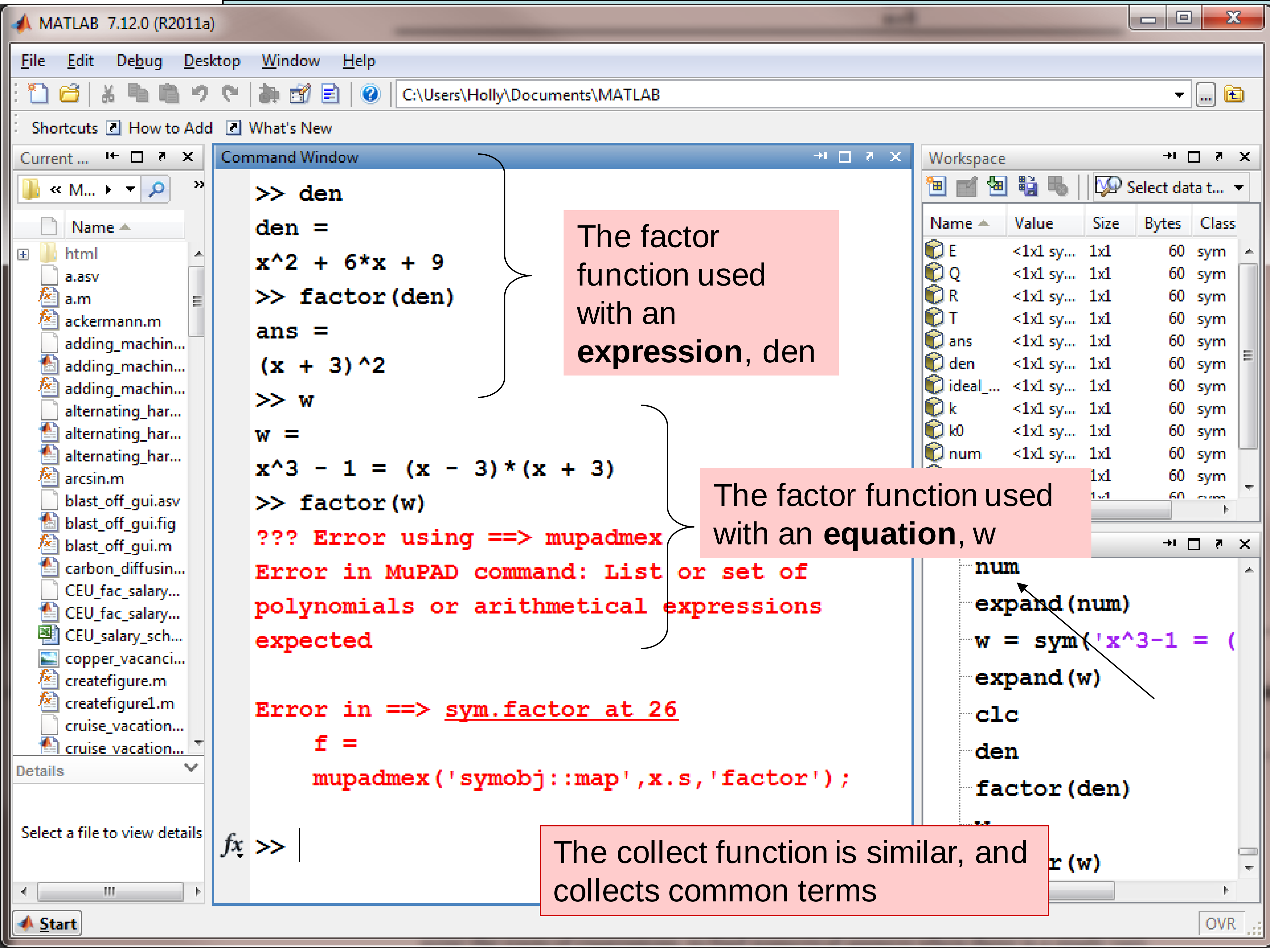

MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Command Window
>> y=2*(x + 3)^2/(x^2 + 6*x + 9)
y =
(2*(x + 3)^2)/(x^2 + 6*x + 9)
>> [num,den]=numden(y)
num =
2*(x + 3)^2
den =
x^2 + 6*x + 9
>>
The numden function extracts the numerator and denominator from an expression
Workspace
Name Value Size Bytes Class
E <1x1 sy... 1x1 60 sym
Q <1x1 sy... 1x1 60 sym
R <1x1 sy... 1x1 60 sym
T <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
k0 <1x1 sy... 1x1 60 sym
num <1x1 sy... 1x1 60 sym
x <1x1 sy... 1x1 60 sym
y <1x1 sy... 1x1 60 sym
Command History
ideal_gas
clc
ideal_gas
clc
ideal_gas_law
E
clc
y=2*(x + 3)^2/(x^2
[num,den]=numden(y
Current ...
Name
html
a.asv
a.m
ackermann.m
adding_machin...
adding_machin...
adding_machin...
alternating_har...
alternating_har...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
cruise_vacation...
Details
Select a file to view details
Start
OVR

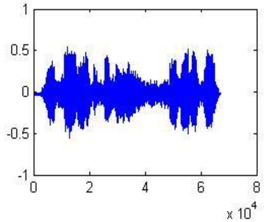


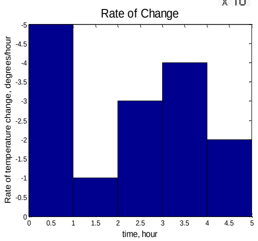
MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Current ...
Command Window
>> y=2*(x + 3)^2/(x^2 + 6*x + 9)
y =
(2*(x + 3)^2)/(x^2 + 6*x + 9)
>> [num,den]=numden(y)
num =
2*(x + 3)^2
den =
x^2 + 6*x + 9
>> num*den
ans =
2*(x + 3)^2*(x^2 + 6*x + 9
>> num + den
ans =
6*x + 2*(x + 3)^2 + x^2 + 9
>> num/den
ans =
(2*(x + 3)^2)/(x^2 + 6*x + 9)
>>
You can combine symbolic variables using standard algebraic operators
Workspace
Select data t...
Name Value Size Bytes Class
E <1x1 sy... 1x1 60 sym
Q <1x1 sy... 1x1 60 sym
R <1x1 sy... 1x1 60 sym
T <1x1 sy... 1x1 60 sym
ans <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
Command History
clc
ideal_gas_law
E
clc
y=2*(x + 3)^2/(x^2
[num,den]=numden(y
num*den
num + den
num/den
Name
html
a.asv
a.m
ackermann.m
adding_machin...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
Details
Select a file to view details
Start
OVR

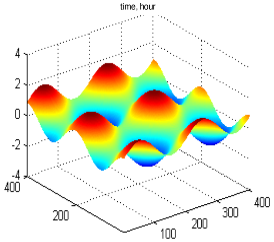

MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Current ...
Name
html
a.asv
a.m
ackermann.m
adding_machin...
adding_machin...
adding_machin...
alternating_har...
alternating_har...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
cruise_vacation...
Details
Select a file to view details
Command Window
>> num
num =
2*(x + 3)^2
>> expand(num)
ans =
2*x^2 + 12*x + 18
>>
num is an **expression**
Workspace
Select data t...
Name Value Size Bytes Class
E <1x1 sy... 1x1 60 sym
Q <1x1 sy... 1x1 60 sym
R <1x1 sy... 1x1 60 sym
T <1x1 sy... 1x1 60 sym
ans <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
k0 <1x1 sy... 1x1 60 sym
num <1x1 sy... 1x1 60 sym
x <1x1 sy... 1x1 60 sym
Command History
clc
y=2*(x + 3)^2/(x^2
[num,den]=numden(y
num*den
num + den
num/den
clc
num
expand(num)
Start
OVR

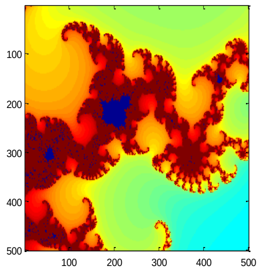

MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Current ...
Name
html
a.asv
a.m
ackermann.m
adding_machin...
adding_machin...
adding_machin...
alternating_har...
alternating_har...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
cruise_vacation...
Details
Select a file to view details
Command Window
>> num
num =
2*(x + 3)^2
>> expand(num)
ans =
2*x^2 + 12*x + 18
>> w = sym('x^3-1 = (x-3)*(x+3)')
w =
x^3 - 1 = (x - 3)*(x + 3)
>> expand(w)
ans =
x^3 - 1 = x^2 - 9
>>
w is an equation
Workspace
Select data t...
Name Value Size Bytes Class
E <1x1 sy... 1x1 60 sym
Q <1x1 sy... 1x1 60 sym
R <1x1 sy... 1x1 60 sym
T <1x1 sy... 1x1 60 sym
ans <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
k0 <1x1 sy... 1x1 60 sym
num <1x1 sy... 1x1 60 sym
w <1x1 sy... 1x1 60 sym
Command History
clc
num
expand(num)
w = sym('x^3-1 = (
clc
num
expand(num)
w = sym('x^3-1 = (
expand(w)
Start
OVR

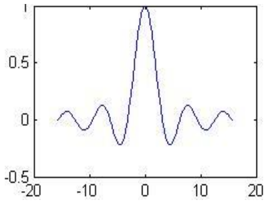

MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Current ...
Command Window
>> den
den =
x^2 + 6*x + 9
>> factor(den)
ans =
(x + 3)^2
>> w
w =
x^3 - 1 = (x - 3)*(x + 3)
>> factor(w)
??? Error using ==> mupadmex
Error in MuPAD command: List or set of polynomials or arithmetical expressions expected
Error in ==> sym.factor at 26
f =
mupadmex('symobj::map',x.s,'factor');
>>
The factor function used with an expression, den
The factor function used with an equation, w
The collect function is similar, and collects common terms
Workspace
Select data t...
Name Value Size Bytes Class
E <1x1 sy... 1x1 60 sym
Q <1x1 sy... 1x1 60 sym
R <1x1 sy... 1x1 60 sym
T <1x1 sy... 1x1 60 sym
ans <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
k0 <1x1 sy... 1x1 60 sym
num <1x1 sy... 1x1 60 sym
num
expand(num)
w = sym('x^3-1 = (
expand(w)
clc
den
factor(den)
r(w)
Name
html
a.asv
a.m
ackermann.m
adding_machin...
adding_machin...
adding_machin...
alternating_har...
alternating_har...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
cruise_vacation...
Details
Select a file to view details
Start
OVR

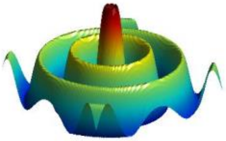






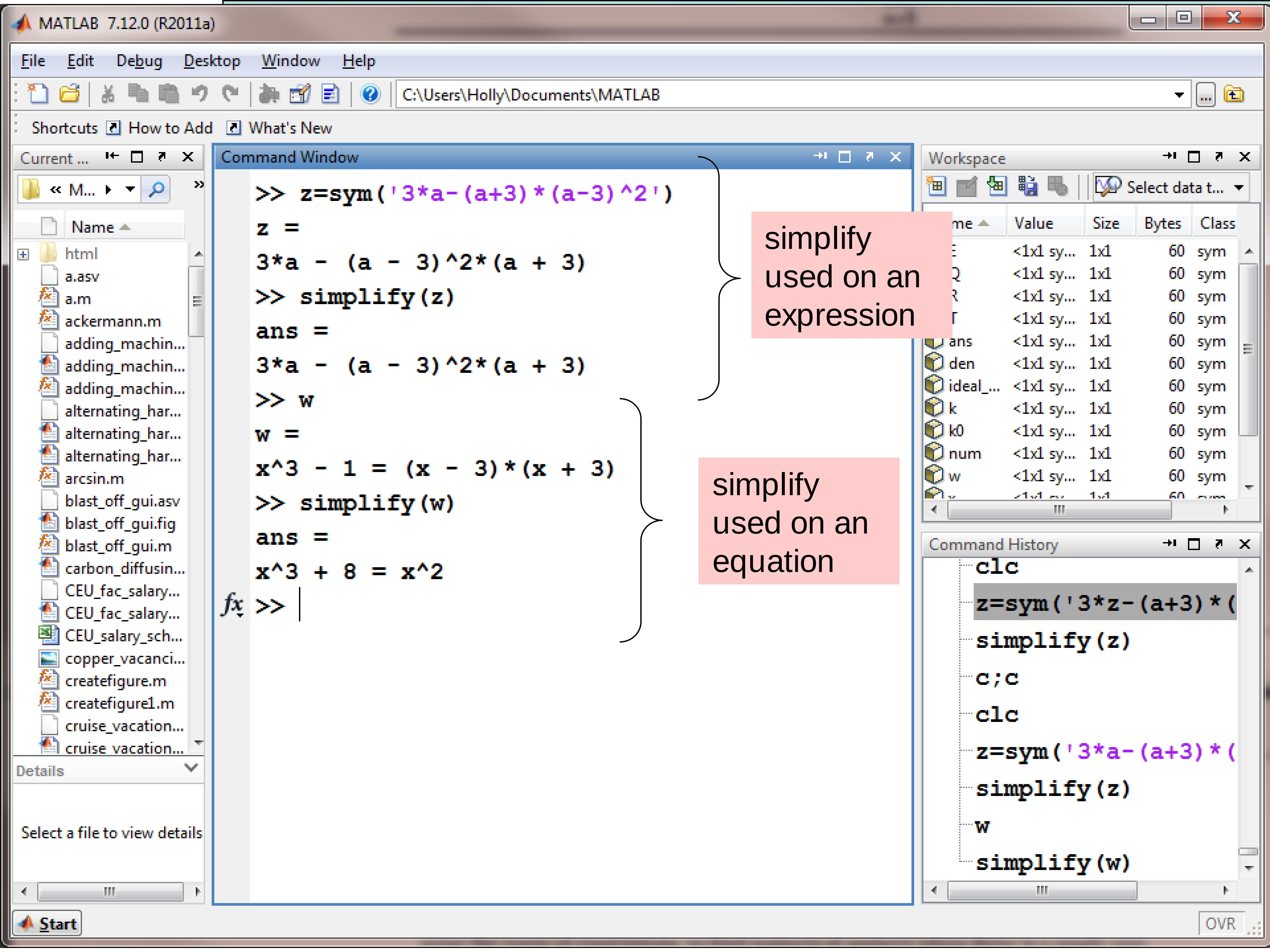


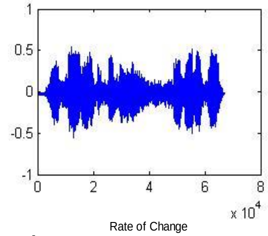


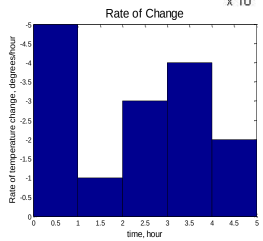


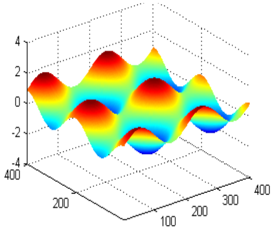
Simplifying

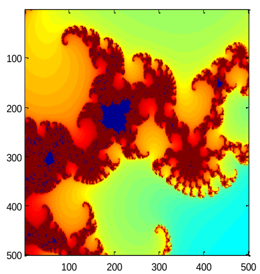
- The **expand**, **factor** and **collect** functions can be used to "simplify" an expression and sometimes an equation
- What constitutes a simplification is not always obvious
- The **simplify** function uses a set of built in rules

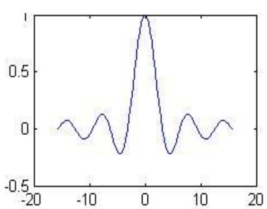

MATLAB 7.12.0 (R2011a)
File Edit Debug Desktop Window Help
C:\Users\Holly\Documents\MATLAB
Shortcuts How to Add What's New
Current ...
Name
html
a.asv
a.m
ackermann.m
adding_machin...
adding_machin...
adding_machin...
alternating_har...
alternating_har...
alternating_har...
arcsin.m
blast_off_gui.asv
blast_off_gui.fig
blast_off_gui.m
carbon_diffusin...
CEU_fac_salary...
CEU_fac_salary...
CEU_salary_sch...
copper_vacanci...
createfigure.m
createfigure1.m
cruise_vacation...
cruise_vacation...
Details
Select a file to view details
Command Window
>> z=sym('3*a-(a+3)*(a-3)^2')
z =
3*a - (a - 3)^2*(a + 3)
>> simplify(z)
ans =
3*a - (a - 3)^2*(a + 3)
>> w
w =
x^3 - 1 = (x - 3)*(x + 3)
>> simplify(w)
ans =
x^3 + 8 = x^2
fx >>
simplify used on an expression
simplify used on an equation
Workspace
Select data t...
Value Size Bytes Class
ans <1x1 sy... 1x1 60 sym
den <1x1 sy... 1x1 60 sym
ideal_... <1x1 sy... 1x1 60 sym
k <1x1 sy... 1x1 60 sym
k0 <1x1 sy... 1x1 60 sym
num <1x1 sy... 1x1 60 sym
w <1x1 sy... 1x1 60 sym
Command History
clc
z=sym('3*z-(a+3)*(
simplify(z)
c;c
clc
z=sym('3*a-(a+3)*(
simplify(z)
w
simplify(w)
Start
OVR

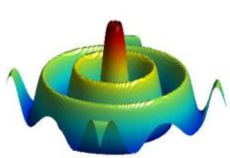






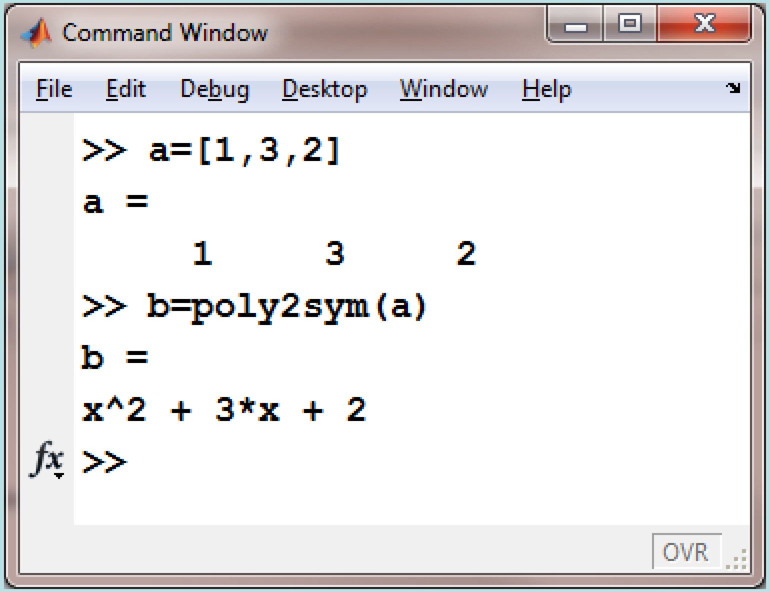


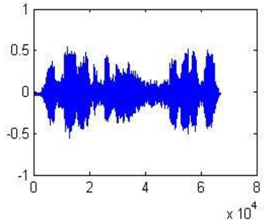


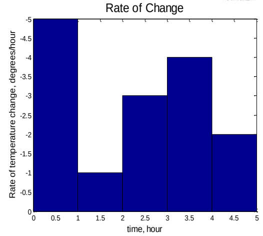


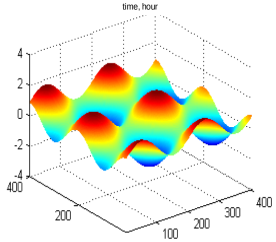
Hint

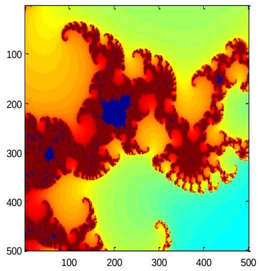
- Use the poly2sym function as a shortcut to create a polynomial

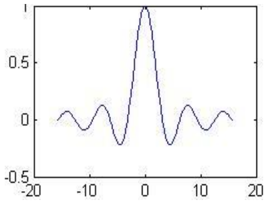


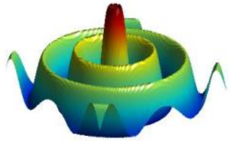






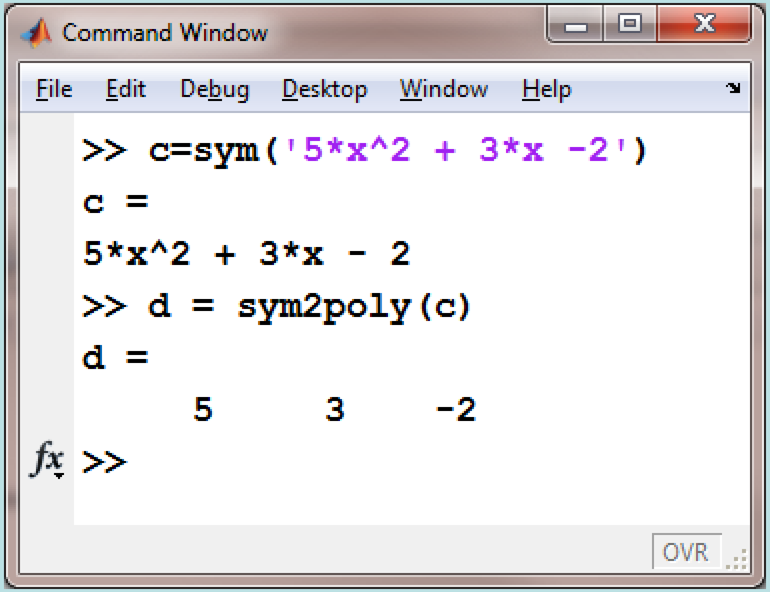


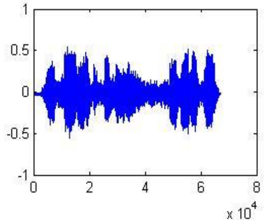


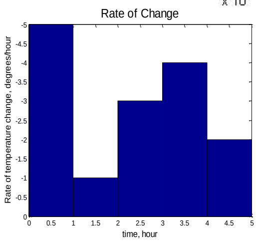


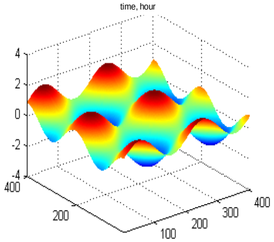
Hint

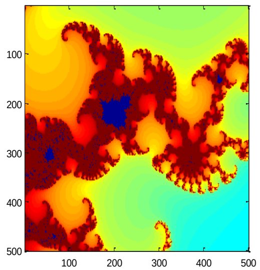
- Extract the coefficients from a polynomial, using the sym2poly function

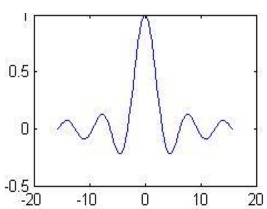


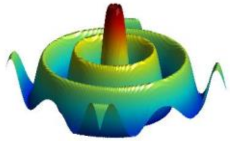






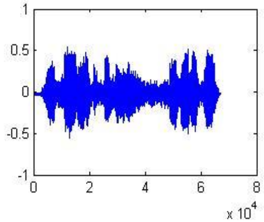


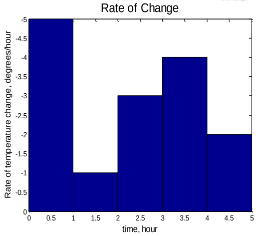


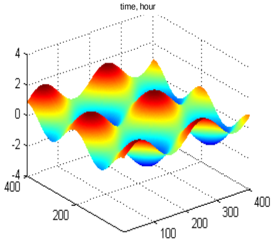


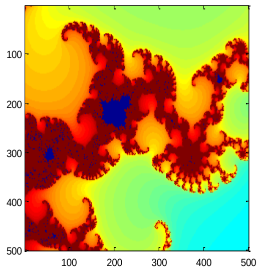
Section 12.2
Solving Equations and Expressions

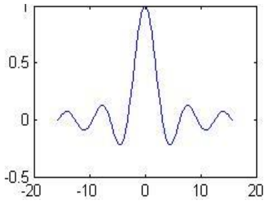
- Use the solve function
- Automatically sets expressions equal to 0 and solves for the roots
- Uses the equality specified in equations
- Solves for the variables in systems of equations

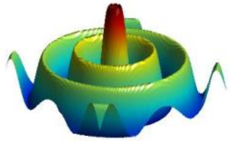












Solving quadratic equation

```
syms a b c x
eqn = a*x^2 + b*x + c == 0
```

eqn = $a\,x^2 + b\,x + c = 0$

```
S = solve(eqn)
```

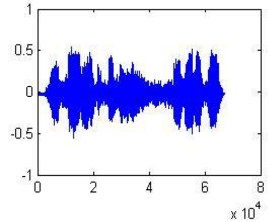
S =

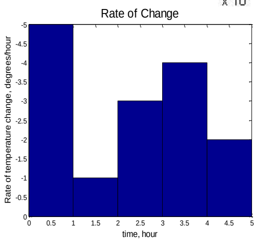
$$\begin{pmatrix} -\dfrac{b+\sqrt{b^2-4\,a\,c}}{2\,a} \\ -\dfrac{b-\sqrt{b^2-4\,a\,c}}{2\,a} \end{pmatrix}$$

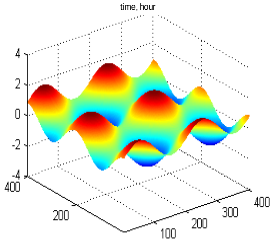
```
Sa = solve(eqn,a)
```

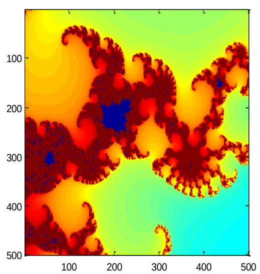
Sa =

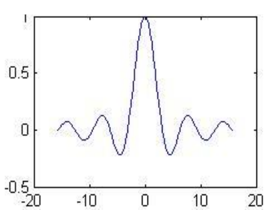
$$-\frac{c+b\,x}{x^2}$$

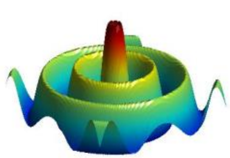












Solving fifth order equation

```
syms x
eqn = x^5 == 3125;
S = solve(eqn,x)
```

```
S =
```

$$\begin{pmatrix} 5 \\ -\sigma_1 - \frac{5}{4} - \frac{5\,\sqrt{2}\,\sqrt{5-\sqrt{5}}\,i}{4} \\ -\sigma_1 - \frac{5}{4} + \frac{5\,\sqrt{2}\,\sqrt{5-\sqrt{5}}\,i}{4} \\ \sigma_1 - \frac{5}{4} - \frac{5\,\sqrt{2}\,\sqrt{\sqrt{5}+5}\,i}{4} \\ \sigma_1 - \frac{5}{4} + \frac{5\,\sqrt{2}\,\sqrt{\sqrt{5}+5}\,i}{4} \end{pmatrix}$$

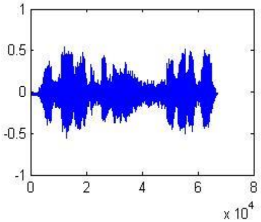
where

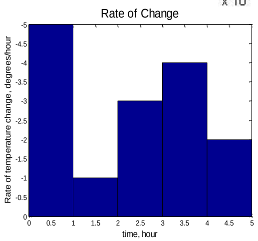
$$\sigma_1 = \frac{5\,\sqrt{5}}{4}$$

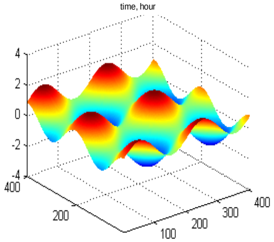
Only real part

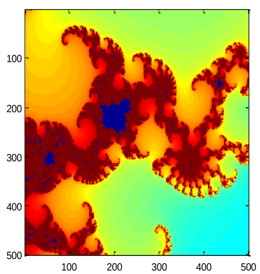
```
S = solve(eqn,x,'Real',true)
```

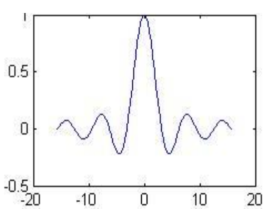
```
S = 5
```

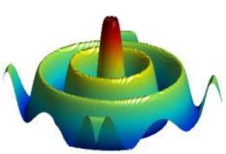






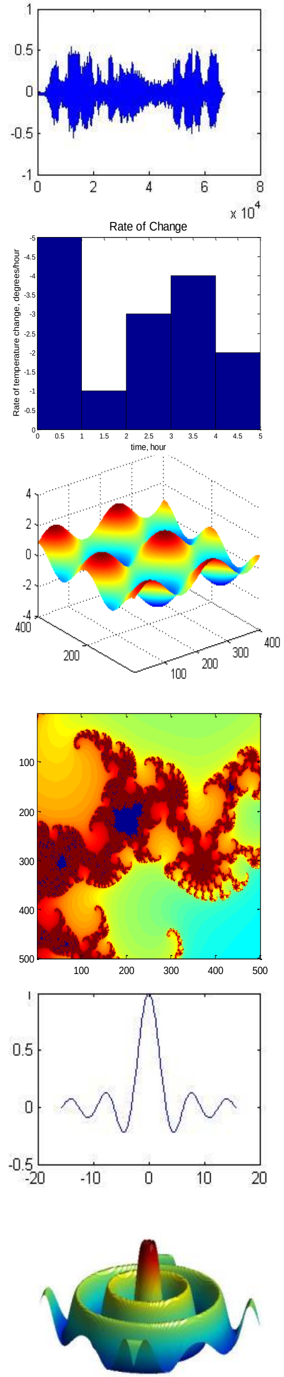






If not possible to have solution we get numerical

```
syms x
eqn = sin(x) == x^2 - 1;
S = solve(eqn,x)

Warning: Unable to solve symbolically. Ret

S = −0.63673265080528201088799090383828
```



Solving multi-variable

```
syms u v
eqns = [2*u + v == 0, u - v == 1];
S = solve(eqns,[u v])
```

```
S = struct with fields:
    u: 1/3
    v: -2/3
```

```
S.u
```

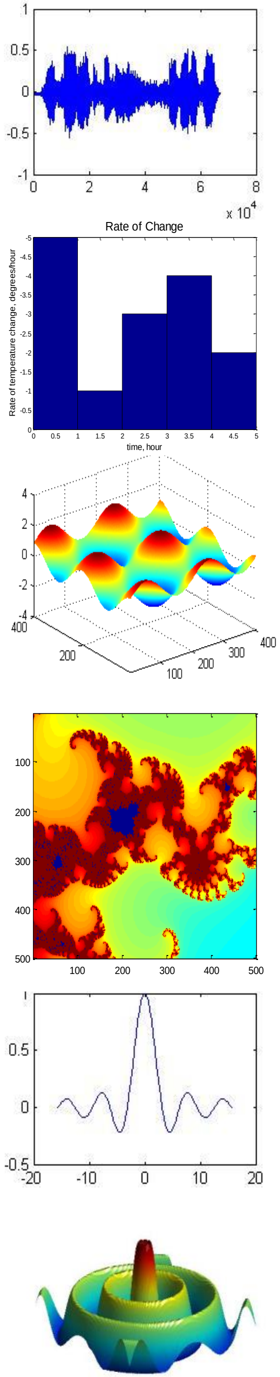
ans =

$$\frac{1}{3}$$

```
S.v
```

ans =

$$-\frac{2}{3}$$



Subs 1 substitutes the solution of the previous equation into expressions

```
expr1 = u^2;
e1 = subs(expr1,S)
```

e1 =

$$\frac{1}{9}$$

```
expr2 = 3*v + u;
e2 = subs(expr2,S)
```

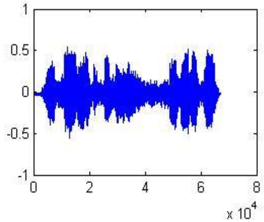
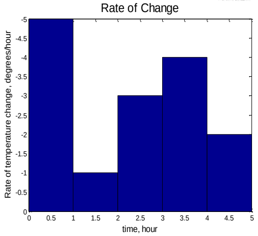
e2 =

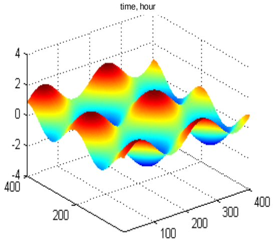
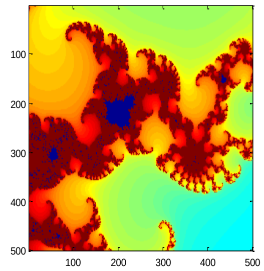
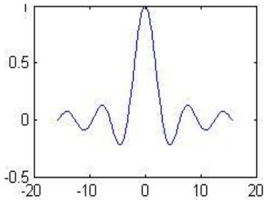
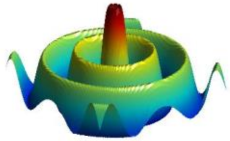
$$-\frac{5}{3}$$

If solve returns an empty object, then no solutions exist.

```
eqns = [3*u+2, 3*u+1];
S = solve(eqns,u)
```

```
S =

Empty sym: 0-by-1
```

To solve the equation with conditions we us 'ReturnCondition'

```
syms x y
eqn1 = x > 0;
eqn2 = y > 0;
eqn3 = x^2 + y^2 + x*y < 1;
eqns = [eqn1 eqn2 eqn3];

S = solve(eqns,[x y],'ReturnConditions',true);
```

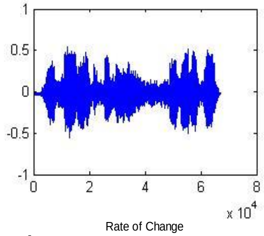
```
S.x
```

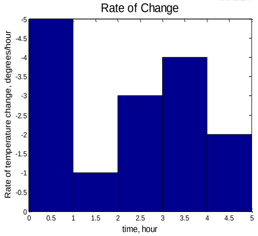
ans =

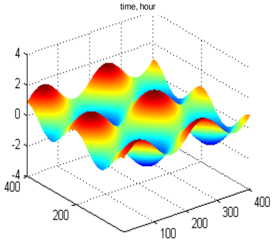
$$\frac{\sqrt{u - 3\,v^2}}{2} - \frac{v}{2}$$

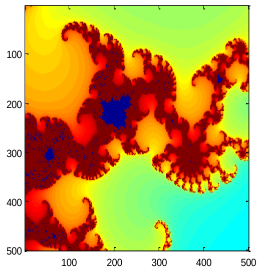
```
S.y
```

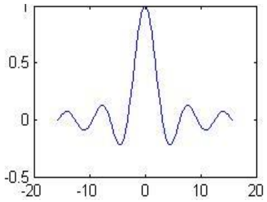
ans = v

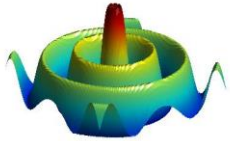












$$2u^2 + v^2 = 0$$

$$u - v = 1$$

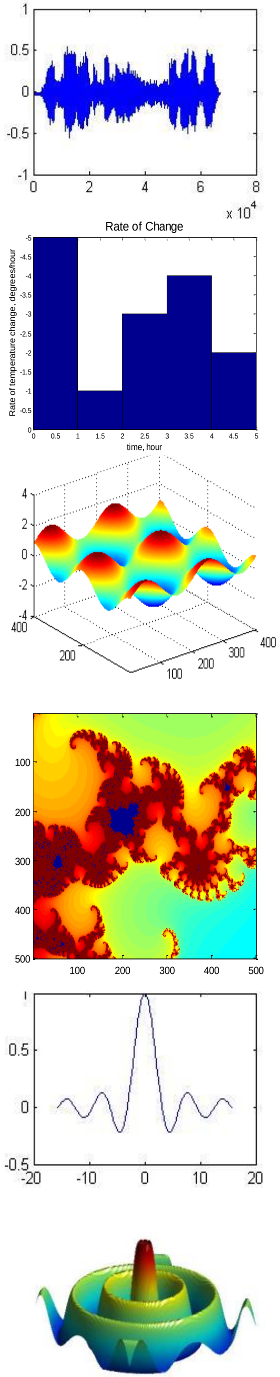
```
syms u v
eqns = [2*u^2 + v^2 == 0, u - v == 1];
vars = [v u];
[solv, solu] = solve(eqns,vars)
```

solv =

$$\begin{pmatrix} -\frac{2}{3} - \frac{\sqrt{2}\,\mathrm{i}}{3} \\ -\frac{2}{3} + \frac{\sqrt{2}\,\mathrm{i}}{3} \end{pmatrix}$$

solu =

$$\begin{pmatrix} \frac{1}{3} - \frac{\sqrt{2}\,\mathrm{i}}{3} \\ \frac{1}{3} + \frac{\sqrt{2}\,\mathrm{i}}{3} \end{pmatrix}$$



```
syms x
eqn = sin(x) == 0;
[solx,parameters,conditions] = solve(eqn,x,'ReturnConditions',true)
```

solx = πk

parameters = k

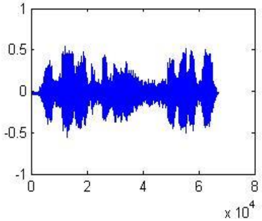
conditions = $k \in \mathbb{Z}$

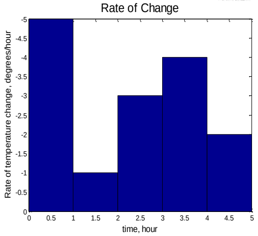
```
assume(conditions)
restriction = [solx > 0, solx < 2*pi];
solk = solve(restriction,parameters)
```

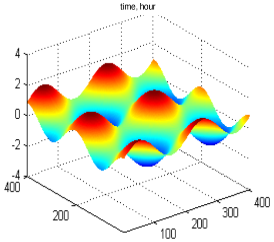
solk = 1

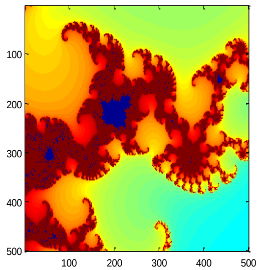
```
valx = subs(solx,parameters,solk)
```

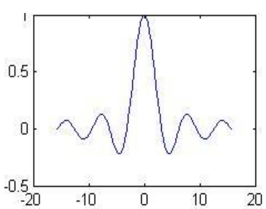
valx = π

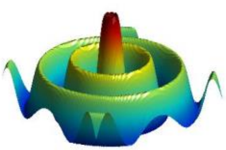












```
syms x
eqn = exp(log(x)*log(3*x)) == 4;
S = solve(eqn,x)
```

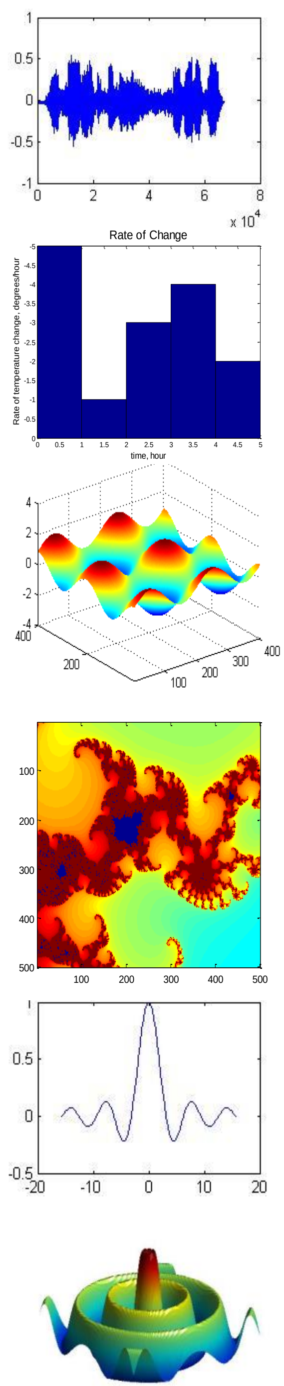
```
Warning: Unable to solve symbolically. Returning a numeric solution using <a h
```

S = $-14.009379055223370038369334703094 - 2.9255310052111190366687179887 69\,\mathrm{i}$

```
S = solve(eqn,x,'IgnoreAnalyticConstraints',true)
```

S =

$$\begin{pmatrix} \dfrac{\sqrt{3}\, e^{-\frac{\sqrt{\log(256)+\log(3)^2}}{2}}}{3} \\[2ex] \dfrac{\sqrt{3}\, e^{\frac{\sqrt{\log(256)+\log(3)^2}}{2}}}{3} \end{pmatrix}$$



```
syms x positive
```

When you solve an equation for a variable under assumptions, the solver only returns solution

```
eqn = x^2 + 5*x - 6 == 0;
S = solve(eqn,x)
```

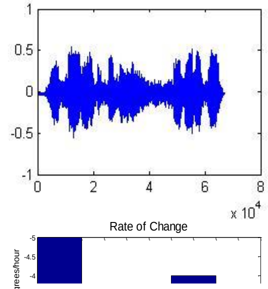
S = 1

Allow solutions that do not satisfy the assumptions by setting `'IgnoreProperties'` to `true`.

```
S = solve(eqn,x,'IgnoreProperties',true)
```

S =

$$\begin{pmatrix} -6 \\ 1 \end{pmatrix}$$



MaxDegree

```
syms x a
eqn = x^3 + x^2 + a == 0;
solve(eqn, x)
```

```
ans =
```

$$\begin{pmatrix} \mathrm{root}(z^3 + z^2 + a, z, 1) \\ \mathrm{root}(z^3 + z^2 + a, z, 2) \\ \mathrm{root}(z^3 + z^2 + a, z, 3) \end{pmatrix}$$

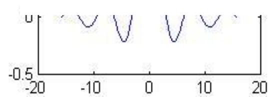
```
S = solve(eqn, x, 'MaxDegree', 3)
```

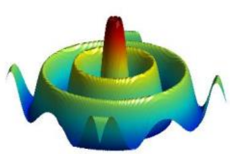
```
S =
```

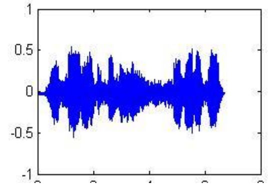
$$\begin{pmatrix} \frac{1}{9\,\sigma_1} + \sigma_1 - \frac{1}{3} \\ -\frac{1}{18\,\sigma_1} - \frac{\sigma_1}{2} - \frac{1}{3} - \frac{\sqrt{3}\left(\frac{1}{9\,\sigma_1} - \sigma_1\right)\mathrm{i}}{2} \\ -\frac{1}{18\,\sigma_1} - \frac{\sigma_1}{2} - \frac{1}{3} + \frac{\sqrt{3}\left(\frac{1}{9\,\sigma_1} - \sigma_1\right)\mathrm{i}}{2} \end{pmatrix}$$

where

$$\sigma_1 = \left(\sqrt{\left(\frac{a}{2} + \frac{1}{27}\right)^2 - \frac{1}{729}} - \frac{a}{2} - \frac{1}{27}\right)^{1/3}$$







Principle value

```
syms x
eqn = sin(x) + cos(2*x) == 1;
S = solve(eqn,x)
```

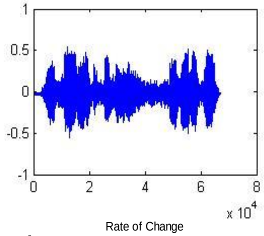
S =

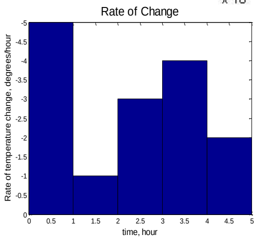
$$\begin{pmatrix} 0 \\ \frac{\pi}{6} \\ \frac{5\,\pi}{6} \end{pmatrix}$$

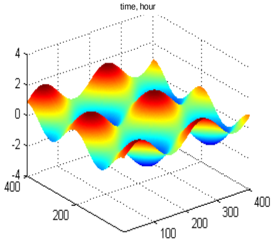
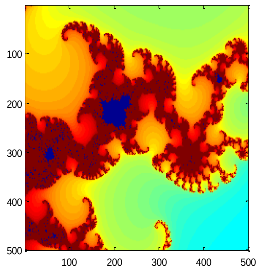
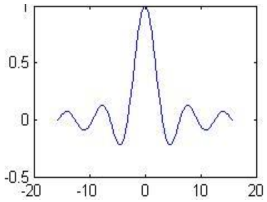
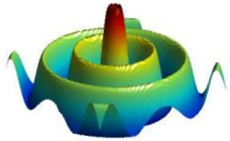
Choose only one solution by setting `'PrincipalValue'` to `true`.

```
S1 = solve(eqn,x,'PrincipalValue',true)
```

S1 = 0





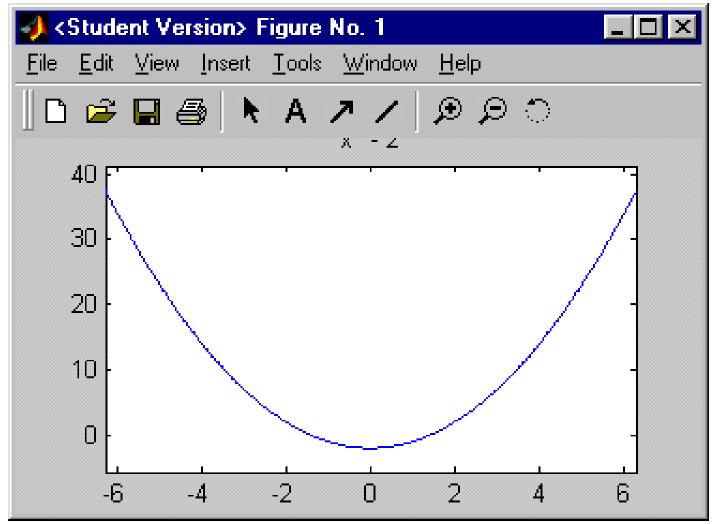





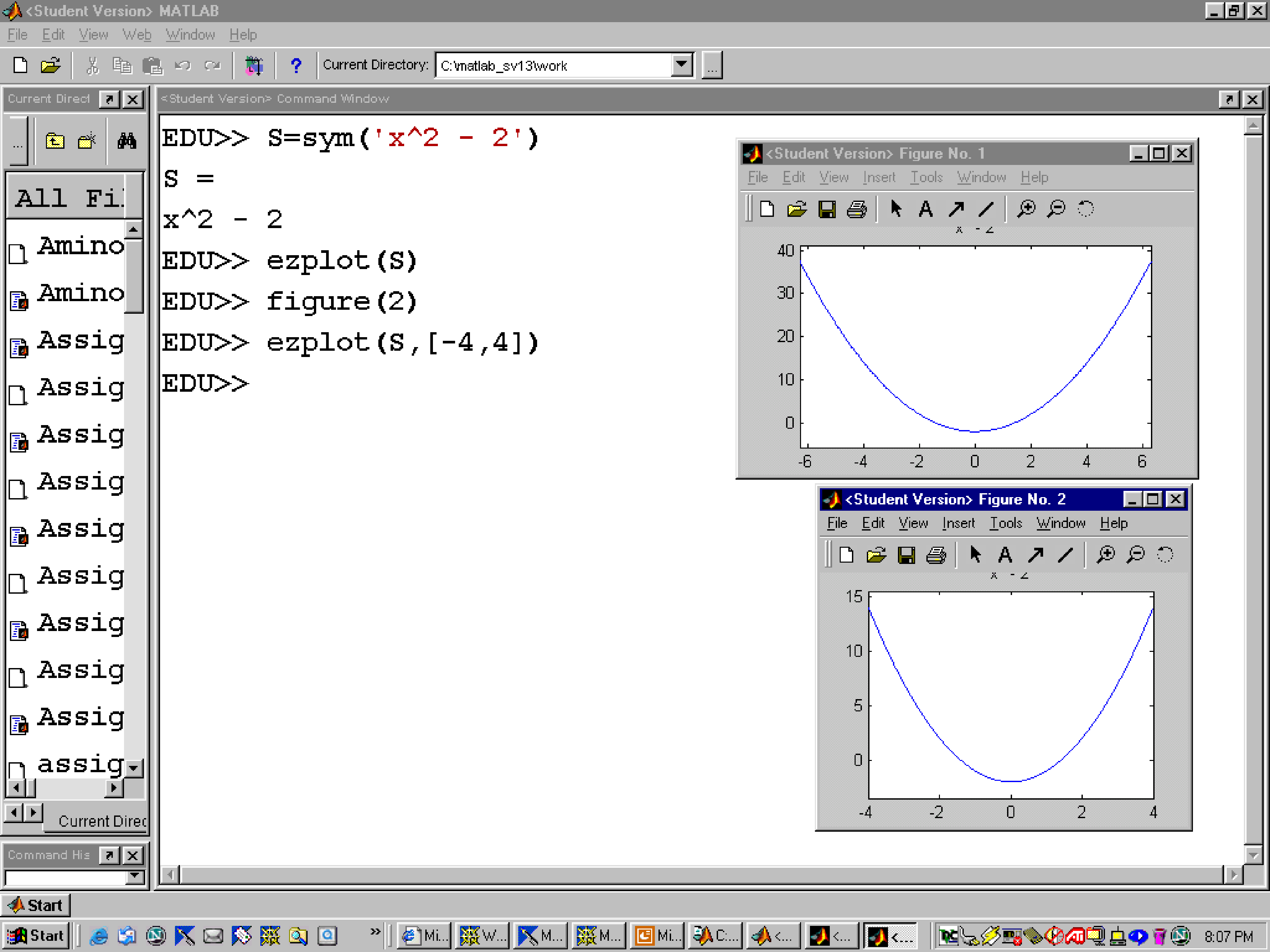
EZPlot

- Allows you to plot symbolic expressions
- ezplot(S)
  - Defaults to a range of -2π to $+2\pi$
- ezplot(S, [xmax, xmin])

```
EDU>> S=sym('x^2 - 2')
S =
x^2 - 2
EDU>> ezplot(S)
EDU>>
```

Note this plot was created with the student version
–The symbolic functionality is included in the student version

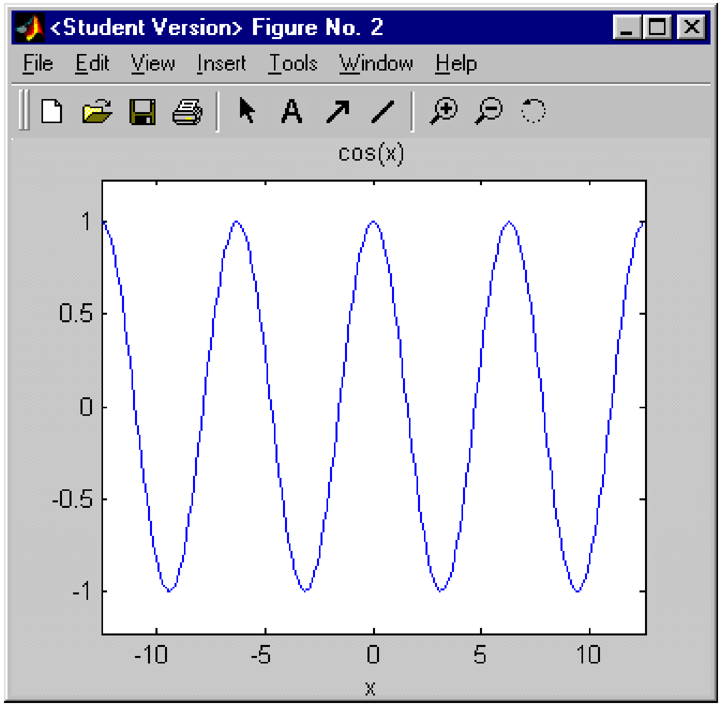


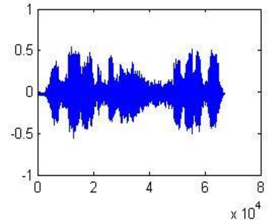

<Student Version> MATLAB
File Edit View Web Window Help
Current Directory: C:\matlab_sv13\work
Current Direct
<Student Version> Command Window
EDU>> S=sym('x^2 - 2')
S =
x^2 - 2
EDU>> ezplot(S)
EDU>> figure(2)
EDU>> ezplot(S,[-4,4])
EDU>>
All Fi
Amino
Amino
Assig
Assig
Assig
Assig
Assig
Assig
Assig
Assig
Assig
assig
Current Direc
Command His
<Student Version> Figure No. 1
File Edit View Insert Tools Window Help
40
30
20
10
0
-6 -4 -2 0 2 4 6
<Student Version> Figure No. 2
File Edit View Insert Tools Window Help
15
10
5
0
-4 -2 0 2 4
Start
Start
8:07 PM

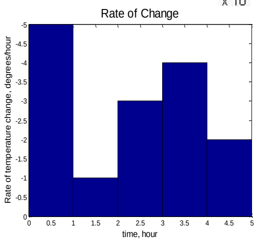
```
EDU>> ezplot('cos(x)',[-4*pi,4*pi])
EDU>>
```

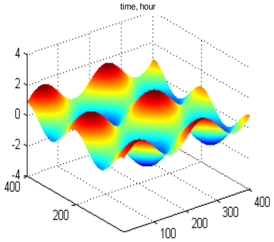
Notice that ezplot creates a title and axis labels automatically

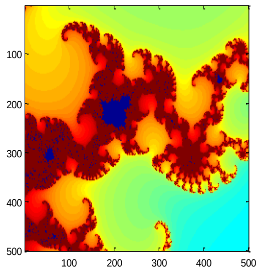
Add your own titles, axis labels and other annotations using the same functions described for numeric plotting in Chapter 5

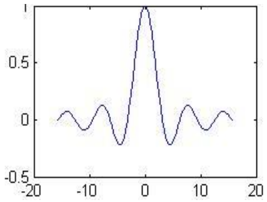


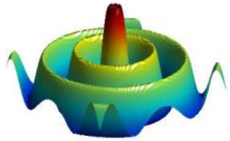






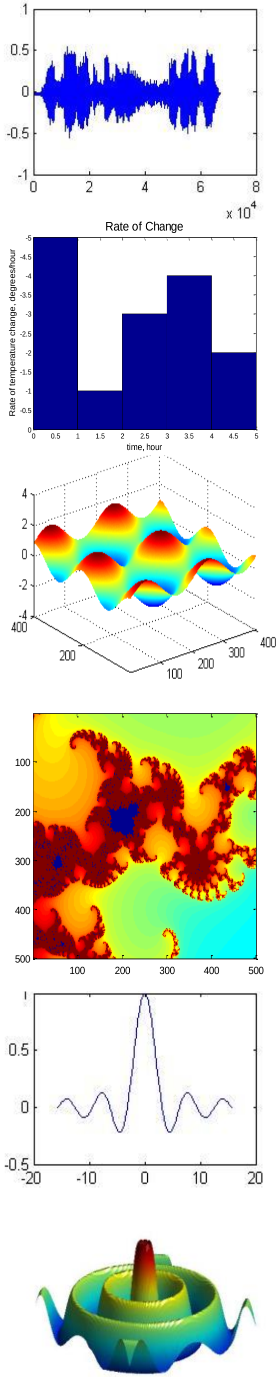






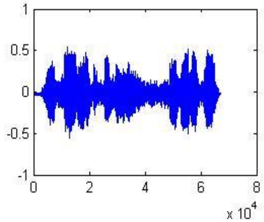
ezplot supports implicit plotting

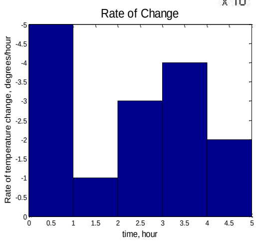
- The equation for a circle can be expressed implicitly as:
  - $x^2 + y^2 = 1$
- You could solve for y, but it's not necessary with ezplot
- **ezplot('x^2 + y^2 =1',[-1.5,1.5])**

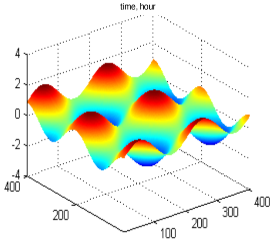


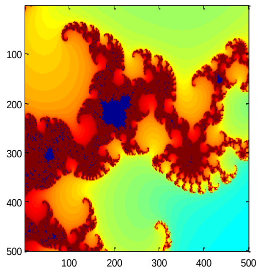
Ezplot supports parametric equation graphs

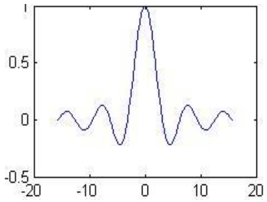
- The equation for a circle can be expressed parametrically as:
  - x=sin(t)
  - y=cos(t)
- To create the graph use…
  - ezplot('sin(x)','cos(x)')

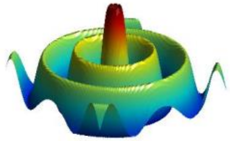




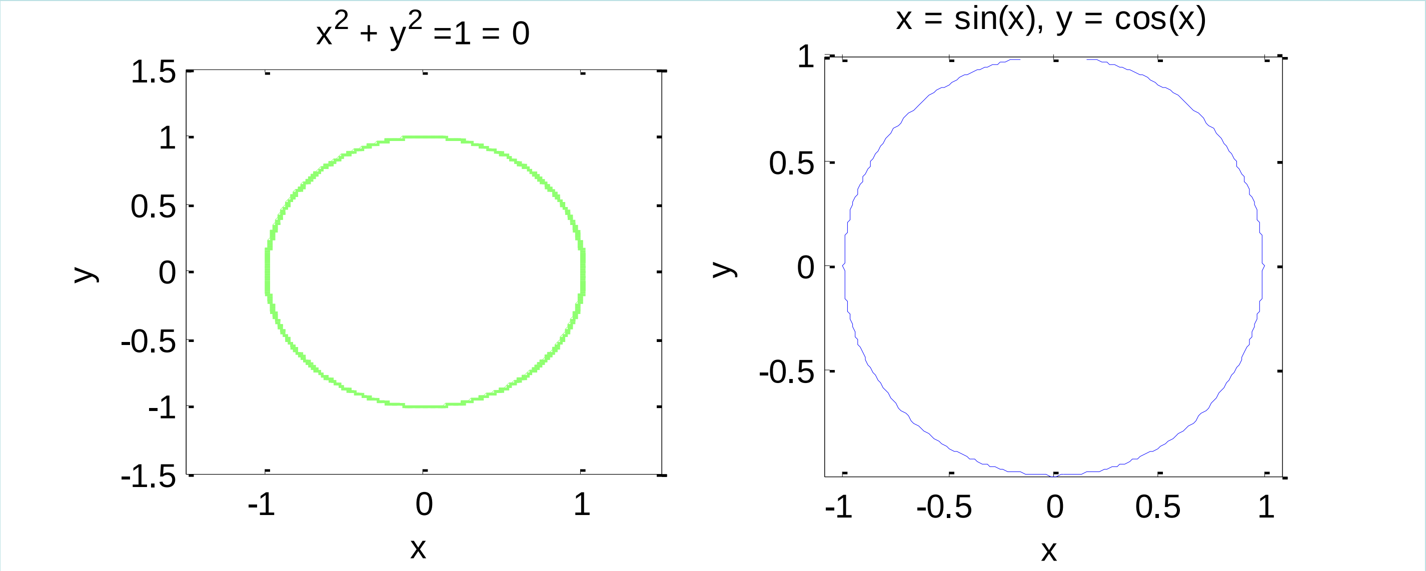


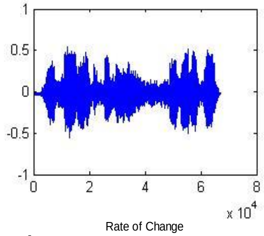
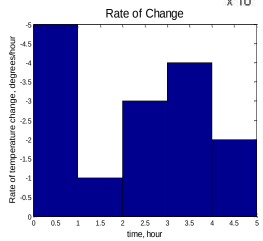


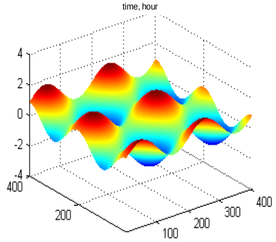
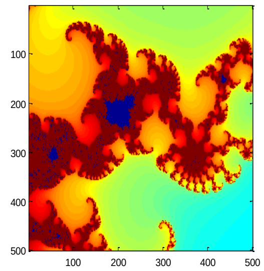
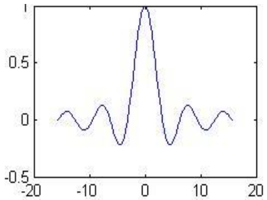
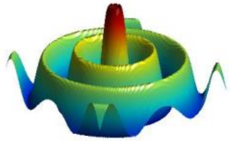




Implicit and Parametric plots of a circle



Hint

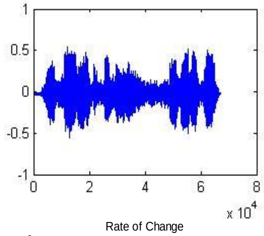
Most symbolic functions will allow you to either enter a symbolic variable that represents a function, or to enter the function itself enclosed in single quotes. For example

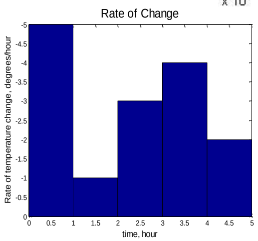
y=sym('x^2-1')

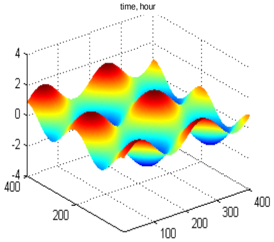
ezplot(y)

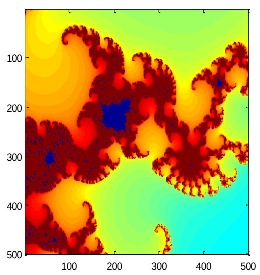
is equivalent to

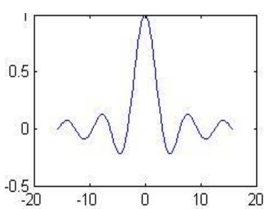
ezplot('x^2-1')

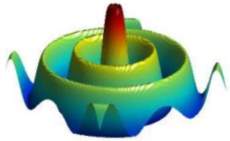






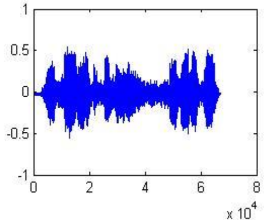


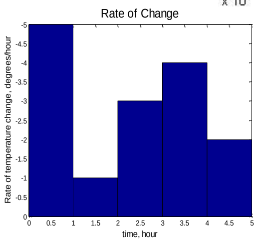


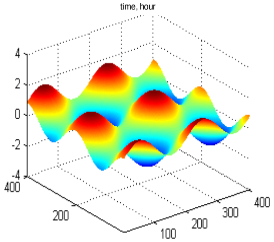


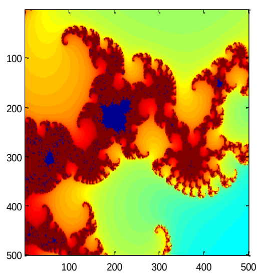
Other Symbolic Plots

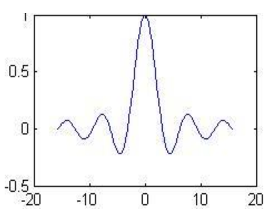
- Additional symbolic plotting functions are available, which mirror the functions used in numeric MATLAB plotting options

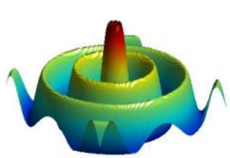






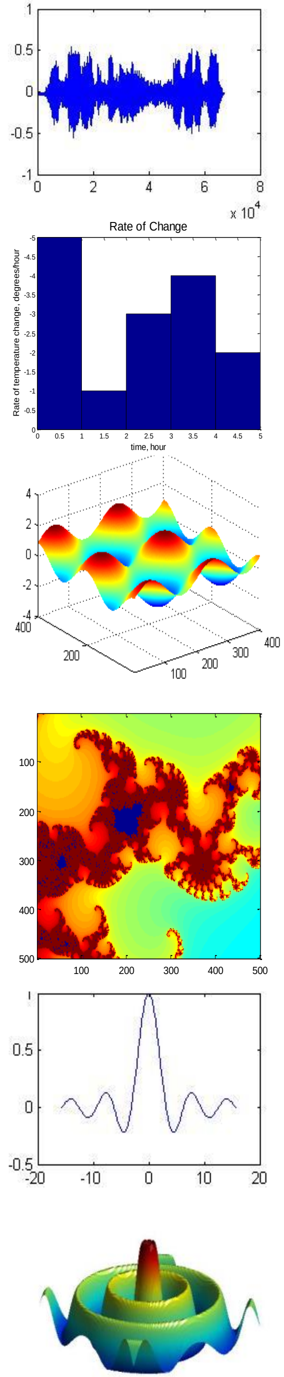




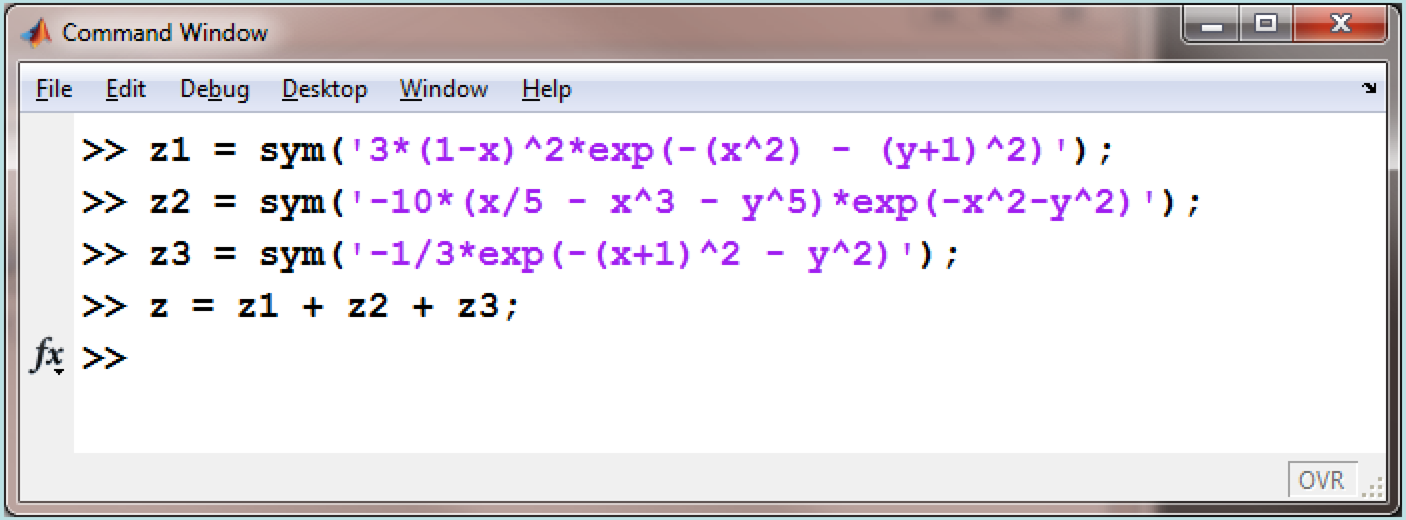


Symbolic Plot Types

ezplot	Function plotter	if z is a function of x **ezplot(z)**
ezmesh	Mesh plotter	if z is a function of x and y **ezmesh(z)**
ezmeshc	Combined mesh and contour plotter	if z is a function of x and y **ezmeshc(z)**
ezsurf	Surface plotter	if z is a function of x and y **ezsurf(z)**
ezsurfc	Combined surface and contour plotter	if z is a function of x and y **ezsurfc(z)**
ezcontour	Contour plotter	if z is a function of x and y **ezcontour(z)**
ezcontourf	Filled contour plotter	if z is a function of x and y **ezcontourf(z)**
ezplot3	3-D parametric curve plotter	if x is a function of t if y is a function of t if z is a function of t ezplot3(x,y,z)
ezpolar	Polar Coordinate plotter	if r is a function of θ ezpolar(r)



To demonstrate these plot types create a symbolic version of “peaks”

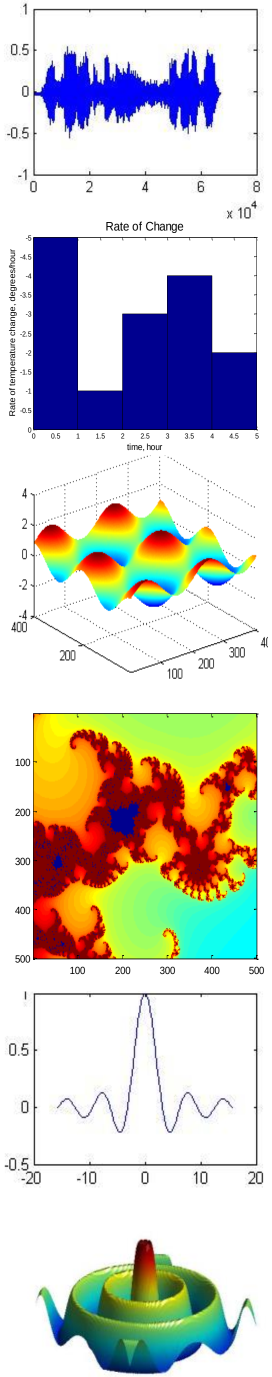


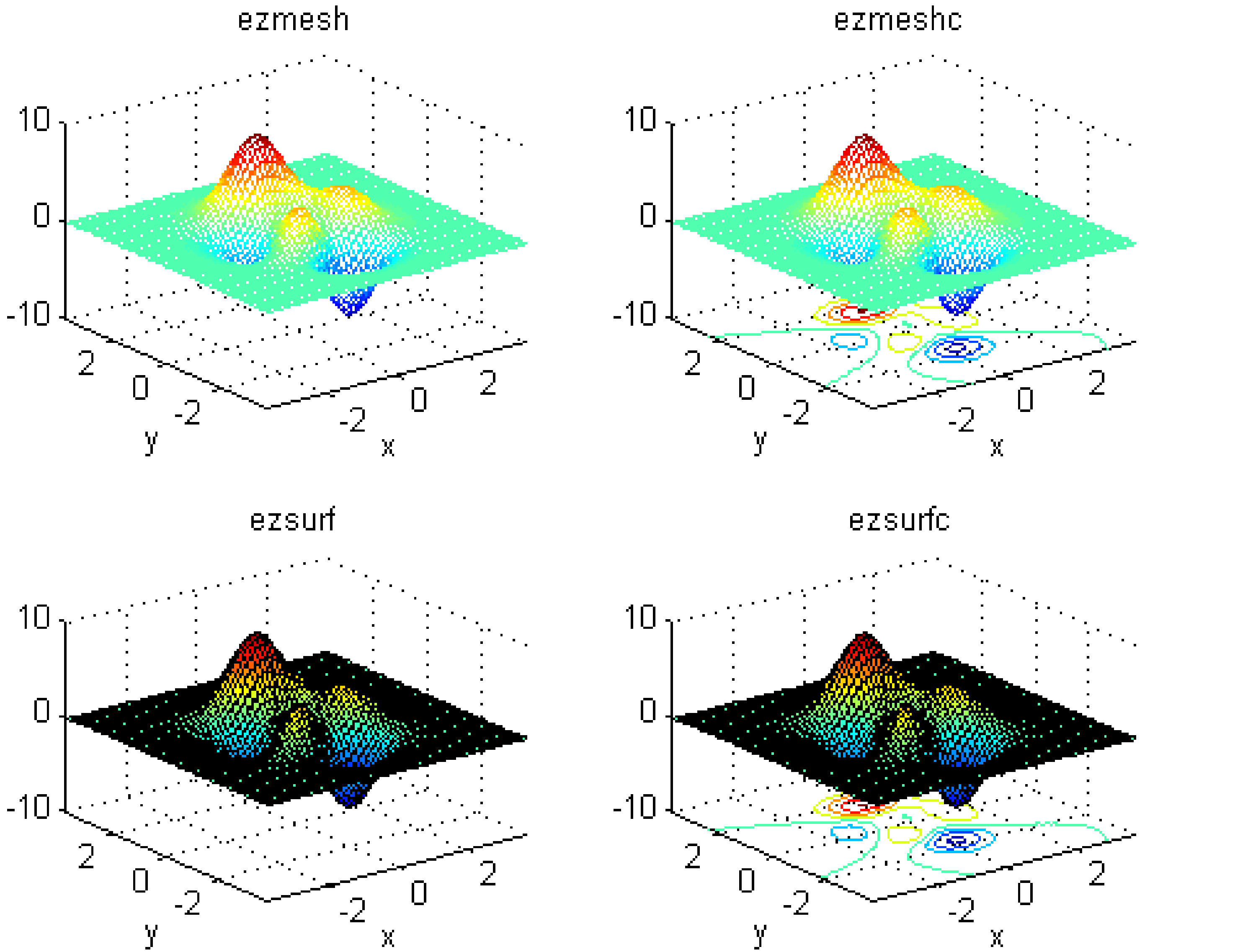
We broke this function up into three parts to make it easier to enter into the computer. Notice that there are no “dot” operators used in these expressions, since they are all symbolic.

```
subplot(2,2,1)
ezmesh(z)
title('ezmesh')

subplot(2,2,2)
ezmeshc(z)
title('ezmeshc')

subplot(2,2,3)
ezsurf(z)
title('ezsurf')

subplot(2,2,4)
ezsurfc(z)
title('ezsurfc')
```

When we created the same plots using a standard MATLAB approach it was necessary to define an array of both x and y values, mesh them together, and calculate the values of z based on the two dimensional arrays. The symbolic plotting capability contained in the symbolic toolbox makes creating these graphs much easier.

All of these graphs can be annotated using the standard MATLAB functions such as **title**, **xlabel**, **text**, etc.





ezmesh
ezmeshc
ezsurf
ezsurfc
10
0
-10
2
0
-2
y
-2
0
2
x

Editor - Untitled*

File Edit Text Go Cell Tools Debug Desktop Window Help

Stack: Base

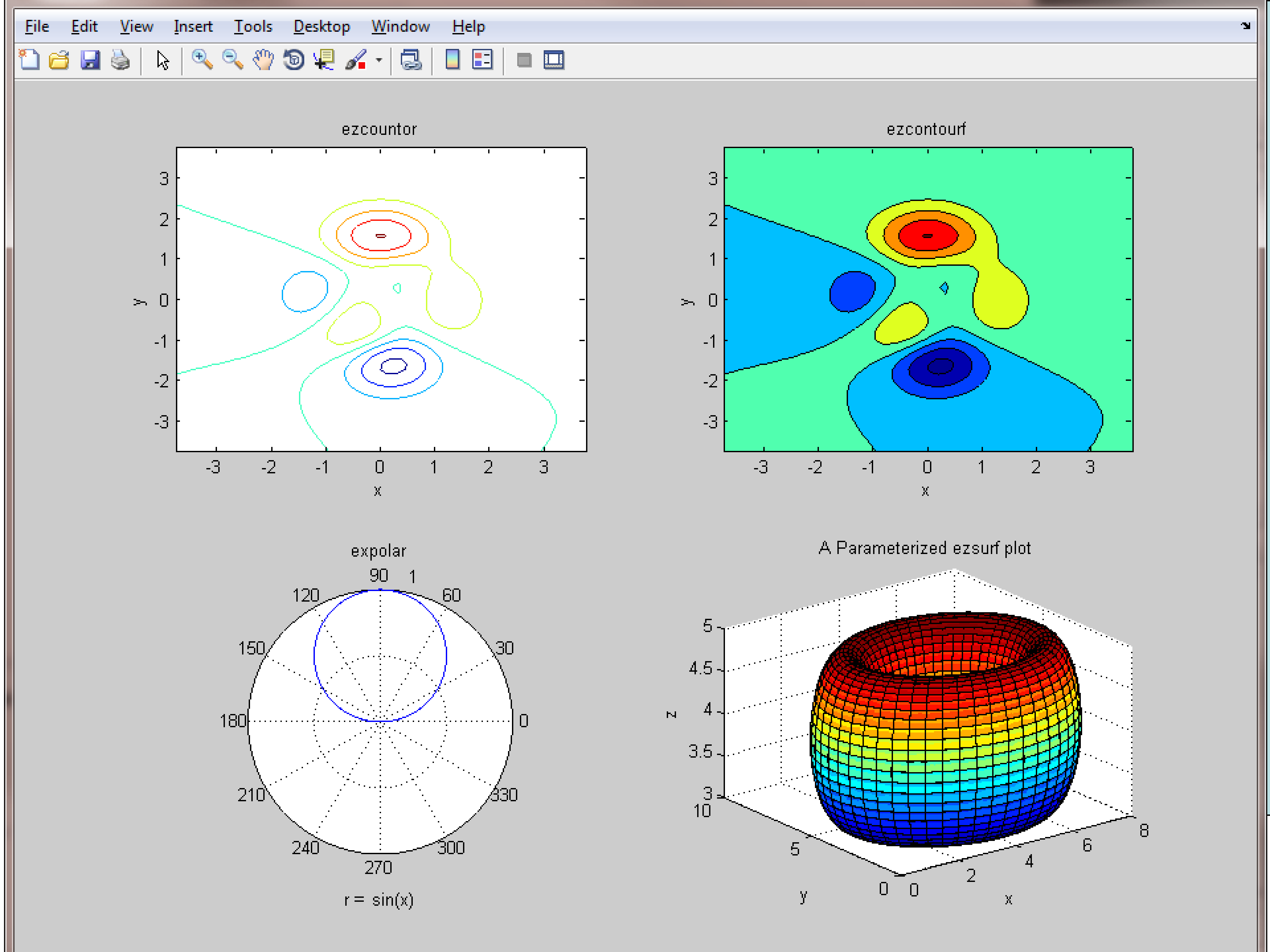
```
subplot(2,2,1)
ezcontour(z)
title('ezcountor')

subplot(2,2,2)
ezcontourf(z)
title('ezcontourf')

subplot(2,2,3)
z = sym('sin(x)')
ezpolar(z)
title('expolar')

subplot(2,2,4)
x = sym('4 + (3 + cos(v))*sin(u)')
y = sym('4 + (3 + cos(v))*cos(u)')
z = sym('4 + sin(v)')
ezsurf(x,y,z)
title('A Parameterized ezsurf plot')
```

These contour plots are a two-dimensional representation of the three-dimensional peaks function

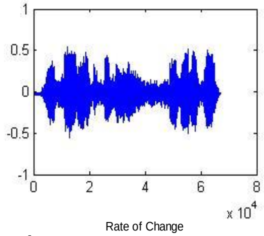
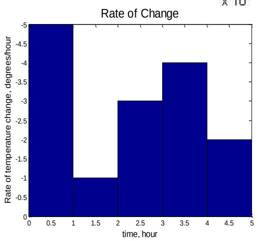
The polar graph requires us to define a new function

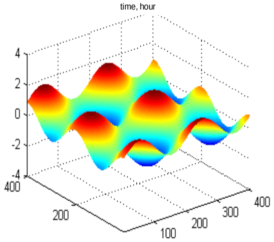
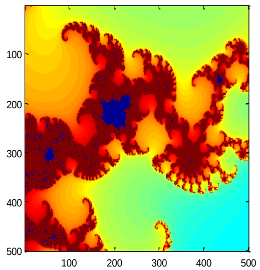
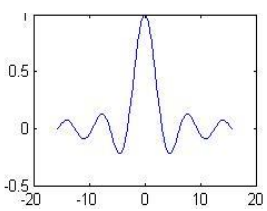
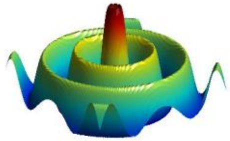
Any of these ezplot graphs can handle parameterized equations

script Ln 8 Col 19 OVR



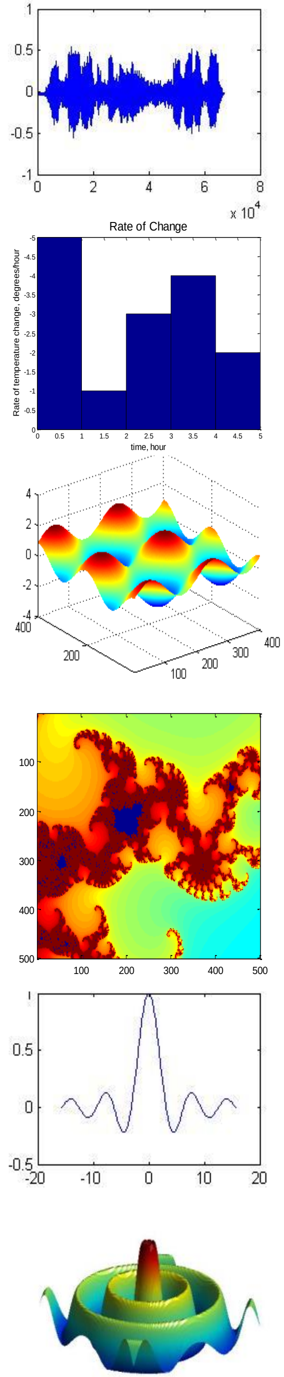
File Edit View Insert Tools Desktop Window Help
ezcountor
ezcontourf
x
y
-3 -2 -1 0 1 2 3
expolar
90
1
120
60
150
30
180
0
210
330
240
300
270
r = sin(x)
A Parameterized ezsurf plot
z
5
4.5
4
3.5
3
10
5
0
y
0
2
4
6
8
x

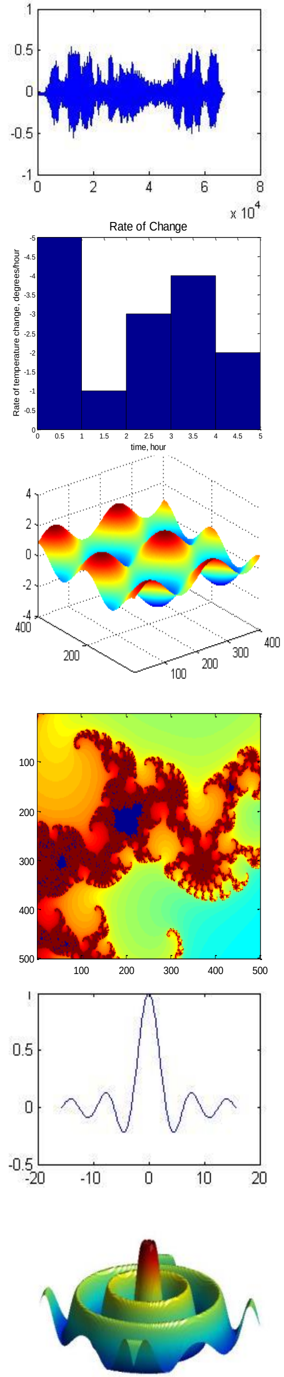
Differentiation

- Concept introduced in Calculus I
- However… a derivative is really just the slope of an equation
- A common application of derivatives is to find velocities and accelerations



Consider a race car…

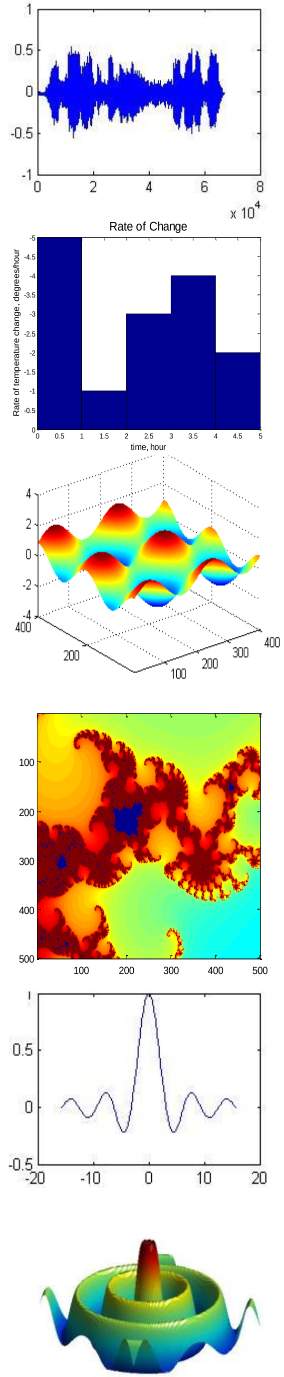
- Assume that during a race the car starts out slowly, and reaches its fastest speed at the finish line
- To avoid running into the stands, the car must then slow down until it finally stops



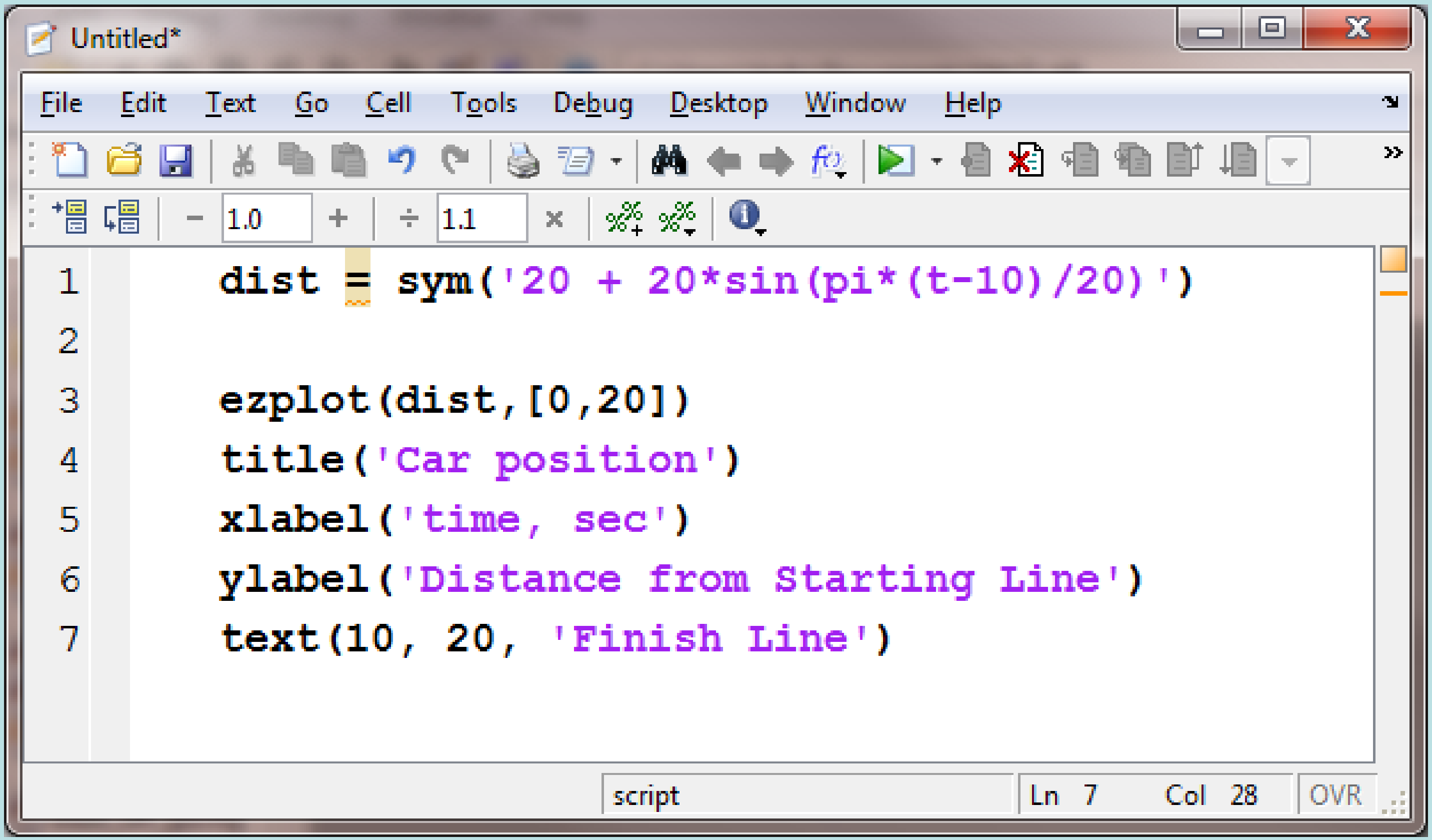
Model

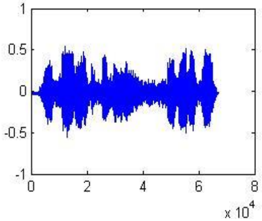
- We might model the position of the car using a sine wave

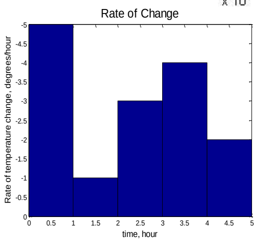
$$dist = 20 + 20 * \sin(\pi * (t - 10) / 20)$$

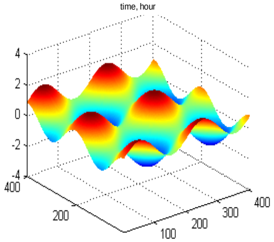


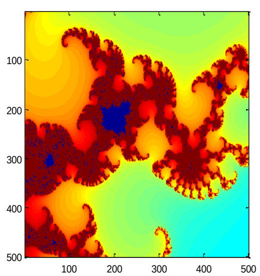
Create a plot of position vs time using ezplot

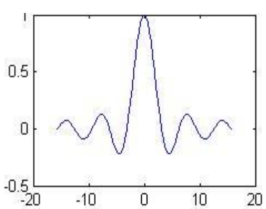


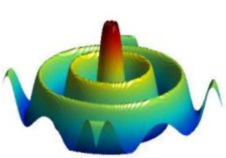




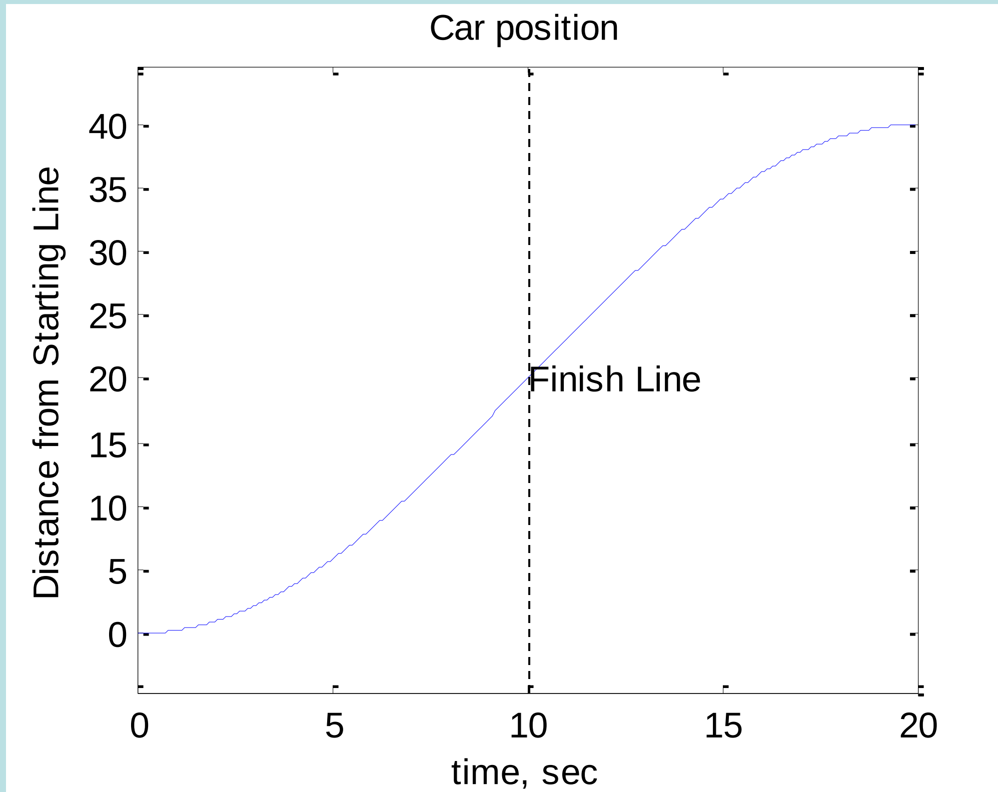


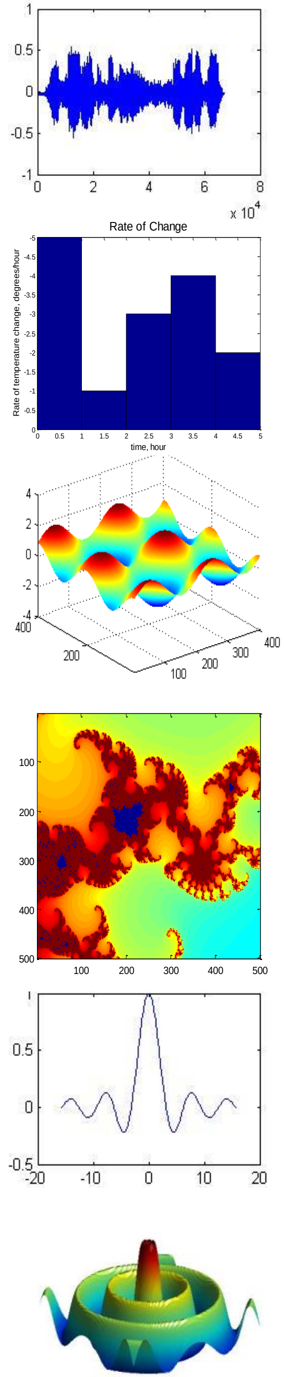






ezplot of position





diff function

- The diff function finds a symbolic derivative
- The velocity is the derivative of the position, so to find the equation of the velocity of the car we’ll use the **diff** function, then plot the result

Untitled*

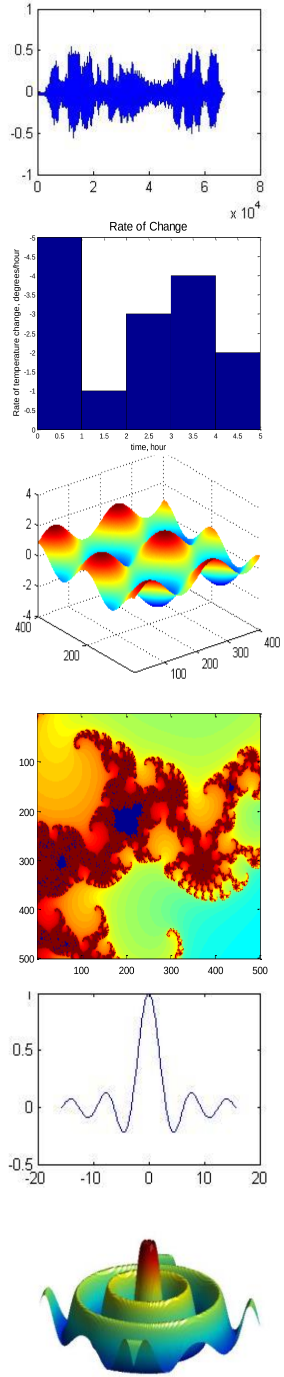
File Edit Text Go Cell Tools Debug Desktop Window Help

```
dist = sym('20 + 20*sin(pi*(t-10)/20)')

ezplot(dist,[0,20])
title('Car position')
xlabel('time, sec')
ylabel('Distance from Starting Line')
text(10, 20, 'Finish Line')

velocity = diff(dist)

ezplot(velocity,[0,20])
title('Race Car Velocity')
xlabel('time, sec')
ylabel('velocity, distance/time')
text(10,3,'Finish Line')
```

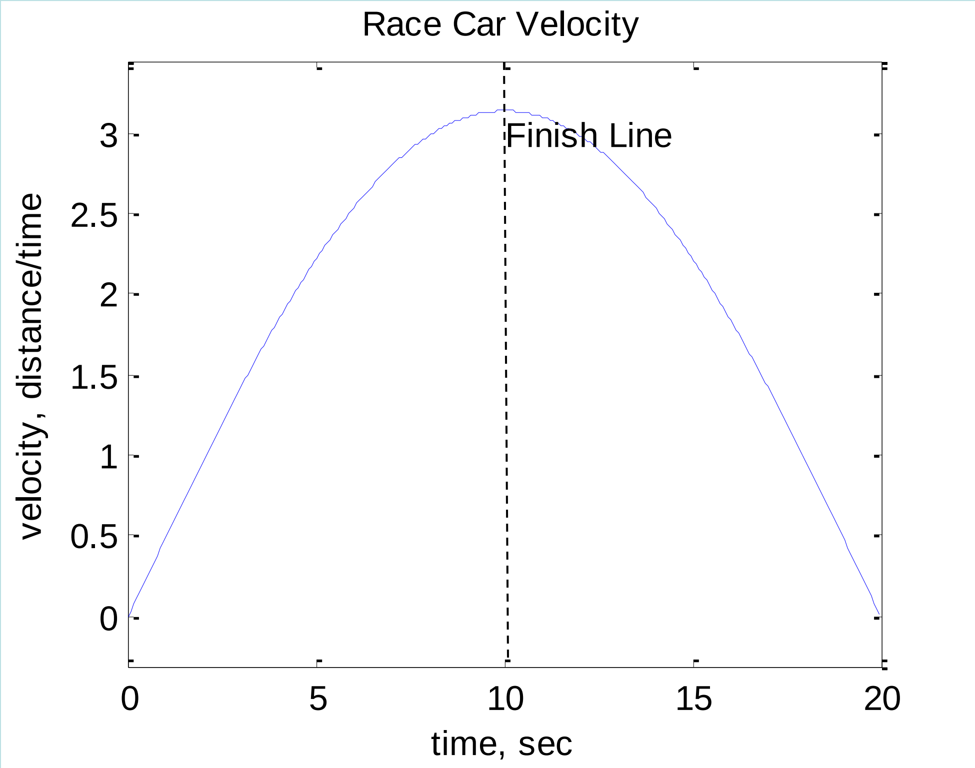
Find the symbolic derivative, which corresponds to the velocity

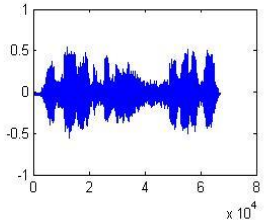
Create a plot of velocity and time

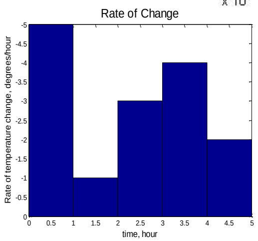
script | Ln 15 | Col 25 | OVR

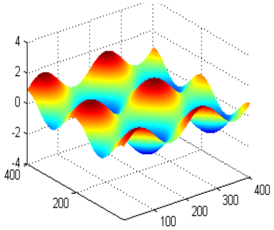


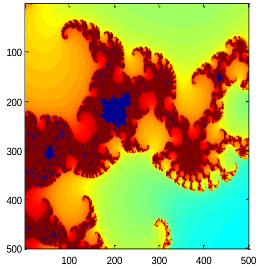
The velocity is the derivative of the position with respect to time

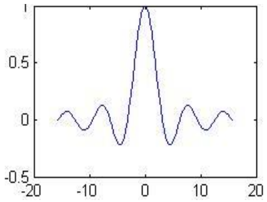


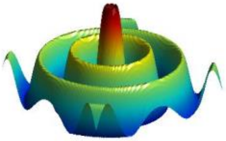










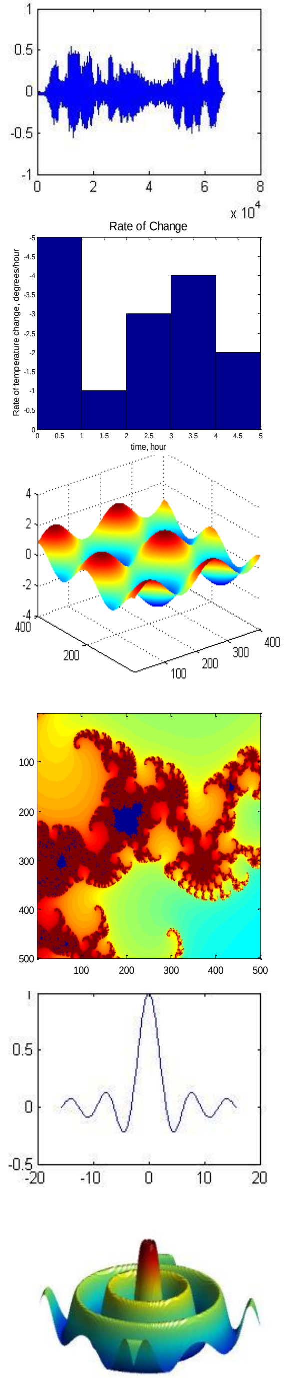


Acceleration

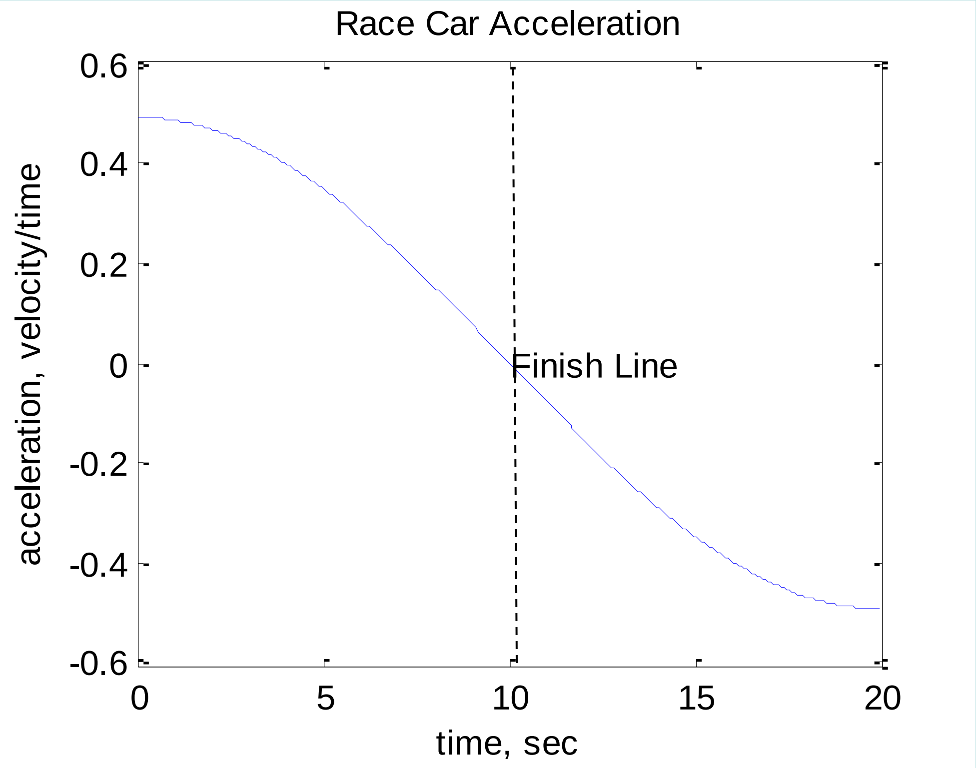
- The acceleration is the derivative of the velocity, so to find the equation of the acceleration of the car we'll use the **diff** function, then plot the result

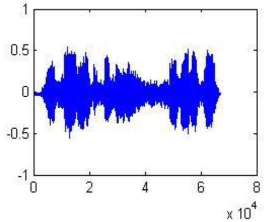
```
dist = sym('20 + 20*sin(pi*(t-10)/20)')

ezplot(dist,[0,20])
title('Car position')
xlabel('time, sec'), ylabel('Distance from Starting Line')
text(10, 20, 'Finish Line')

velocity = diff(dist)

ezplot(velocity,[0,20])
title('Race Car Velocity')
xlabel('time, sec'), ylabel('velocity, distance/time')
text(10,3,'Finish Line')

acceleration = diff(velocity)
ezplot(acceleration, [0,20])
title('Race Car Acceleration')
xlabel('time,sec'), ylabel('acceleration, velocity/time')
text(10,0,'Finish Line')
```

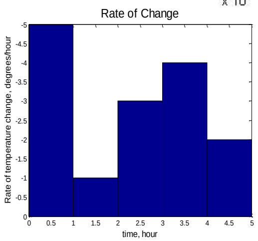
Determine the equation for the acceleration

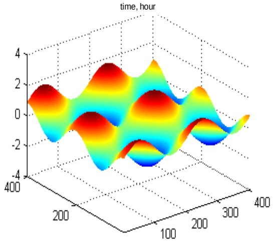


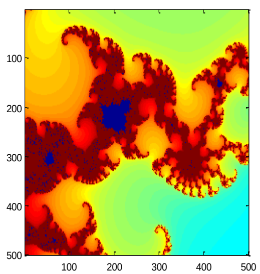
Acceleration is the derivative of the velocity

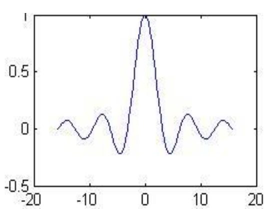


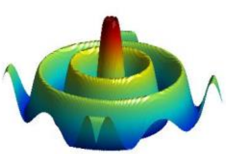






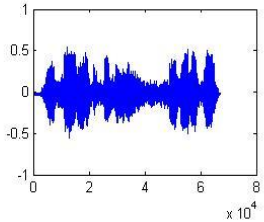
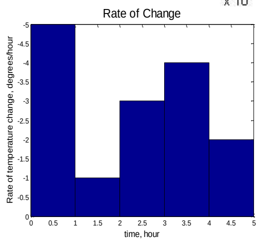


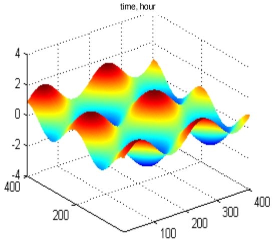
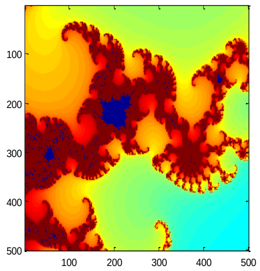
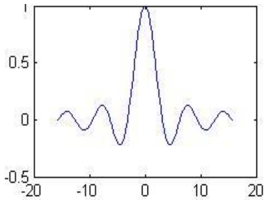
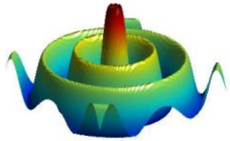




Symbolic Differentiation

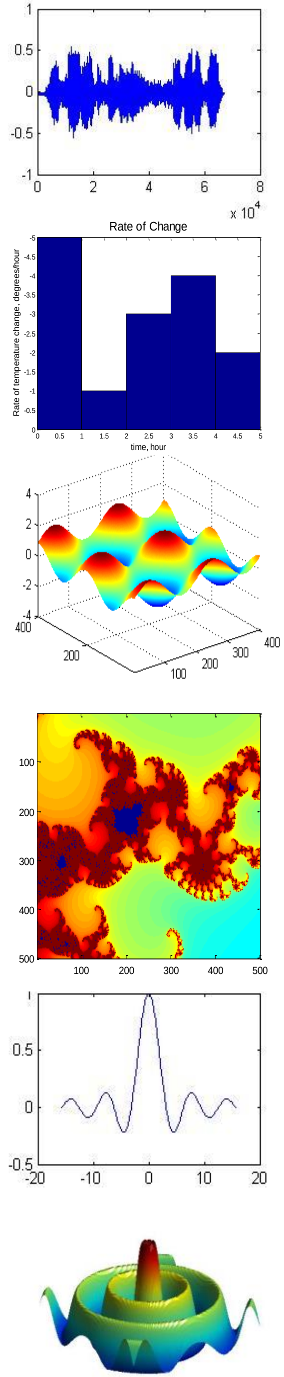
`diff(f)`	Returns the derivative of the expression **`f`** with respect to the default independent variable	**y=sym('x^3+z^2')** **diff(y)** **ans =** **3*x^2**
`diff(f,'t')`	Returns the derivative of the expression **`f`** with respect to the variable **`t`**.	**y=sym('x^3+z^2')** **diff(y,'z')** **ans =** **2*z**
`diff(f,n)`	Returns the **n**th derivative of the expression **`f`** with respect to the default independent variable	**y=sym('x^3+z^2')** **diff(y,2)** **ans =** **6*x**
`diff(f,'t',n)`	Returns the **n**th derivative of the expression **`f`** with respect to the variable **`t`**.	**y=sym('x^3+z^2')** **diff(y,'z',2)** **ans =** **2**

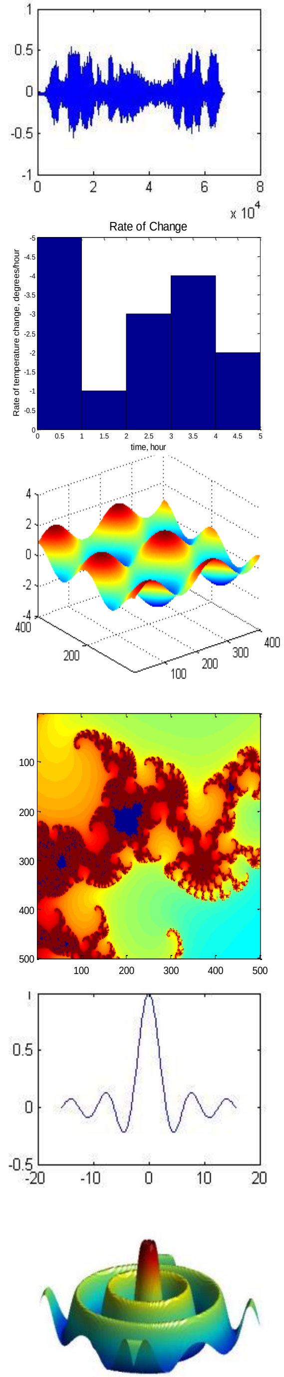





Partial Derivatives

- If you have multiple variables, MATLAB takes the derivative with respect to x – unless you specify otherwise
- All the other variables are kept constant



```
>> y = sym('x^2 + t - 3*z^3')
y =
x^2 - 3*z^3 + t
>> diff(y)
ans =
2*x
>>
```



Command Window

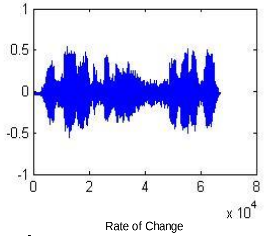
File Edit Debug Desktop Window Help

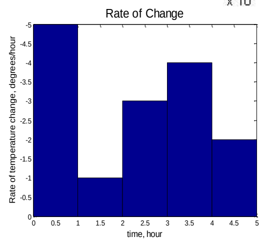
```
>> y = sym('x^2 + t - 3*z^3')
y =
x^2 - 3*z^3 + t
>> diff(y)
ans =
2*x
>> diff(y,'t')
ans =
1
>>
```

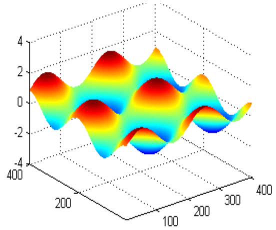
To find the derivative with respect to some variable other than x, you must specify it in the diff function

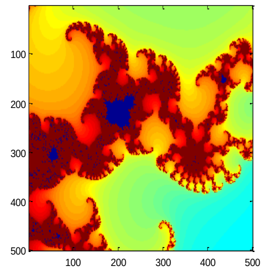
Notice that t is enclosed in single quotes, since we haven't specified it as a symbolic variable

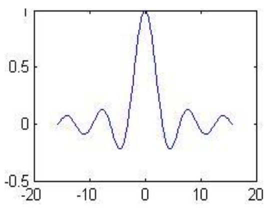
OVR

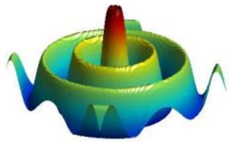






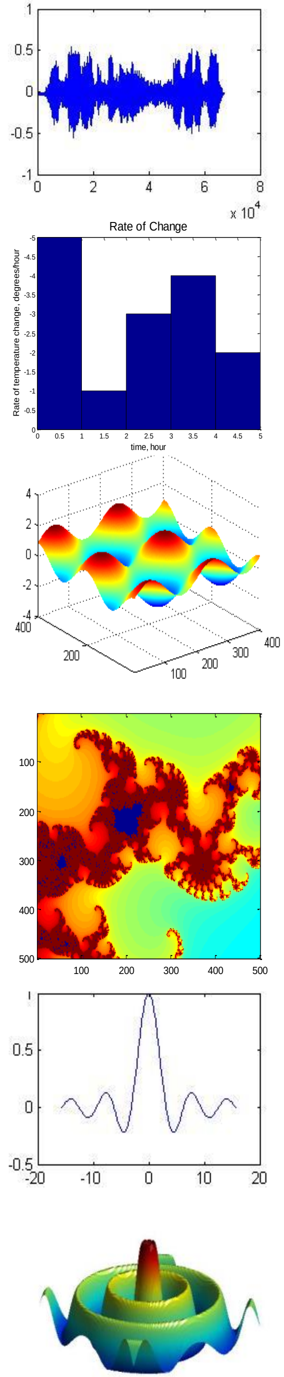






Integration

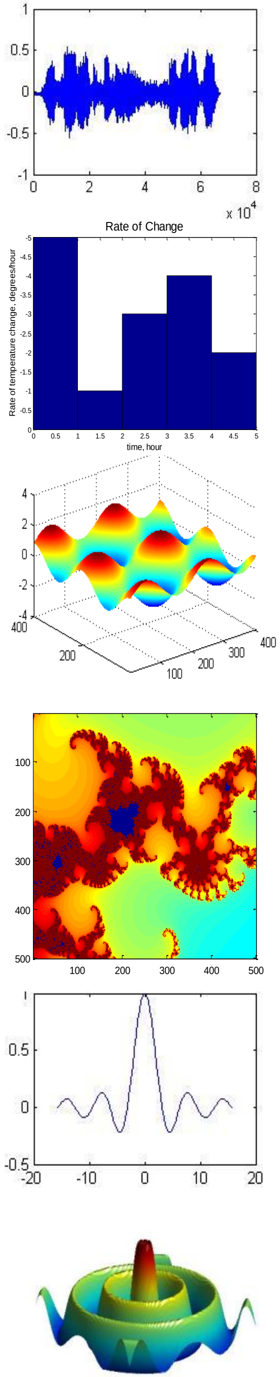
- Usually introduced in Calculus II
- Often visualized as the area under a curve
- MATLAB has built in symbolic integration capability.



Consider a piston cylinder device

- Work done by a piston cylinder device as it moves up or down, can be calculated by taking the integral of P with respect to V

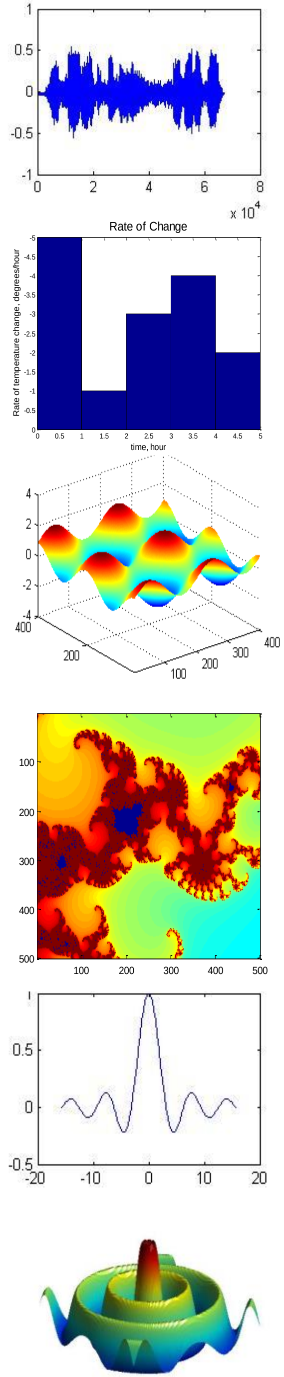
$$W = \int_1^2 P\,dV$$



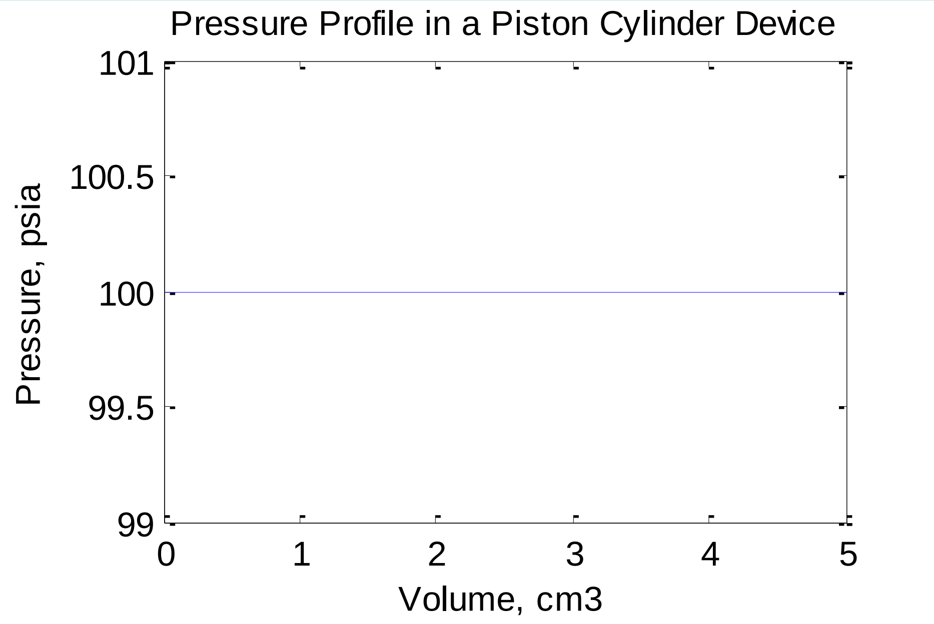
To perform the integration we need to know how P changes with V

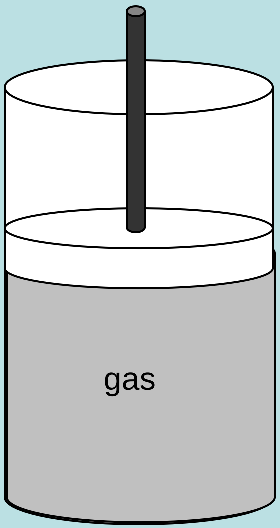
- If P is constant the problem becomes

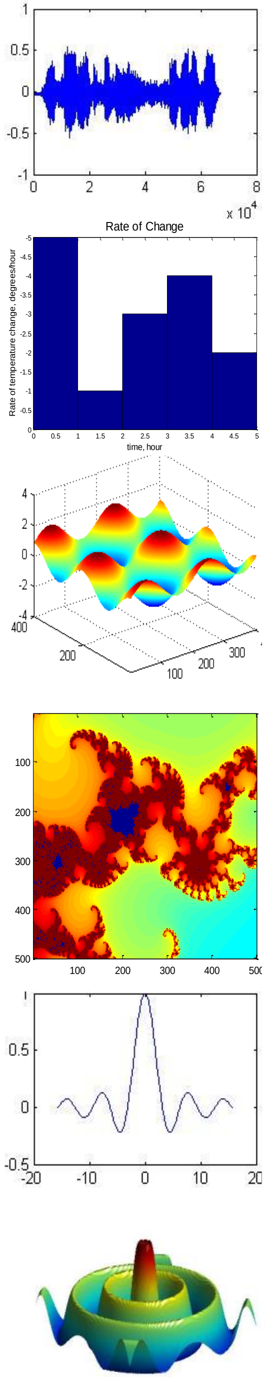
$$W = P\int_1^2 dV$$

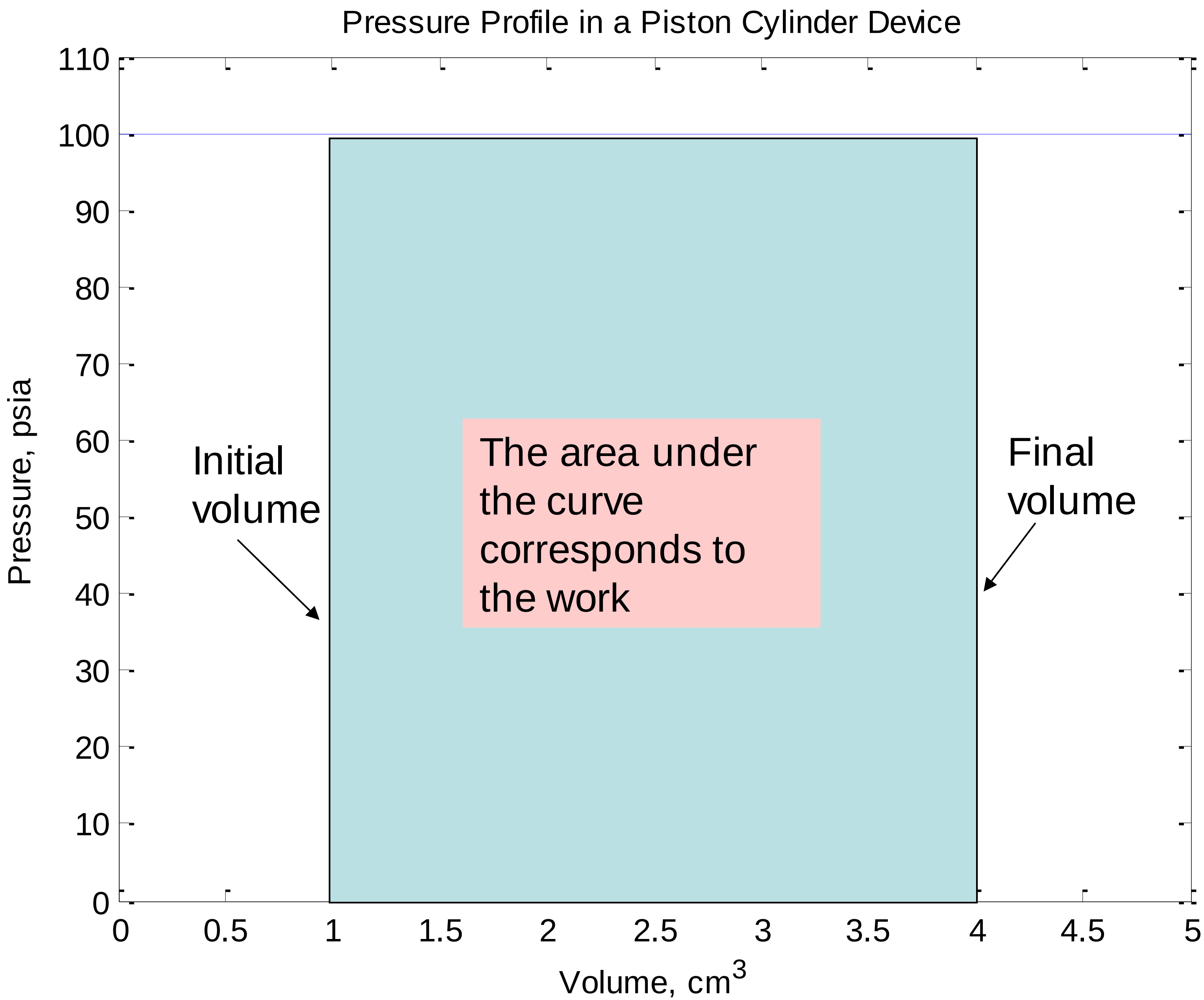


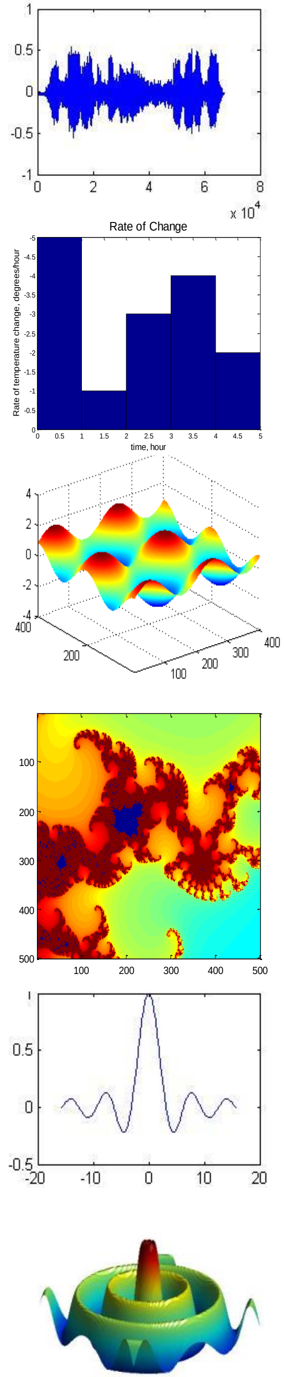
Model of the behavior of a piston cylinder device








Pressure Profile in a Piston Cylinder Device
Pressure, psia
Volume, cm3
Initial volume
Final volume
The area under the curve corresponds to the work
0
10
20
30
40
50
60
70
80
90
100
110
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



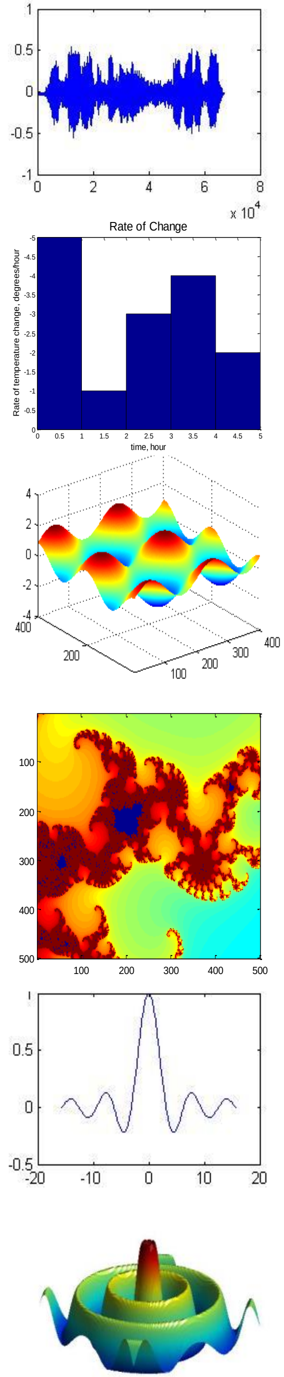
Hand Calculation

$$W = \int_1^4 P dV = P\int_1^4 dV = PV\Big|_1^4 = PV_4 - PV_1 = P\Delta V$$

$$if \quad P = 100\, psia$$

$$W = 3cm^3 * 100\, psia$$

Read this as: Work is equal to the integral of P with respect to V, from V=1 to V=4



MATLAB Solution

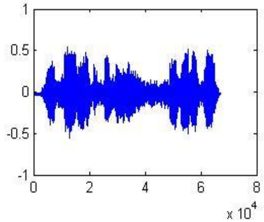
Command Window

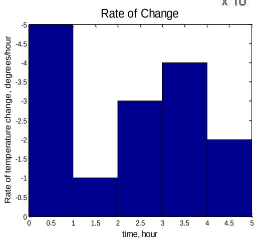
File Edit Debug Desktop Window Help

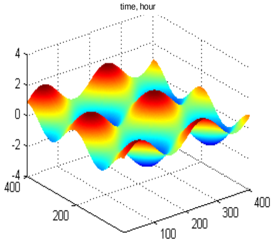
```
>> syms P V
>> W = int(P,V,1,4)
W =
3*P
>> subs(W,P,100)
ans =
   300
>> 
```

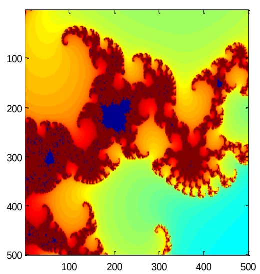
Work is equal to the integral of P with respect to V, from V=1 to V=4

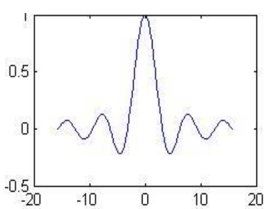
Substitute in 100 as the value of P

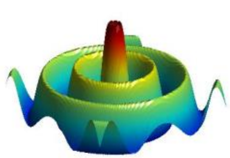






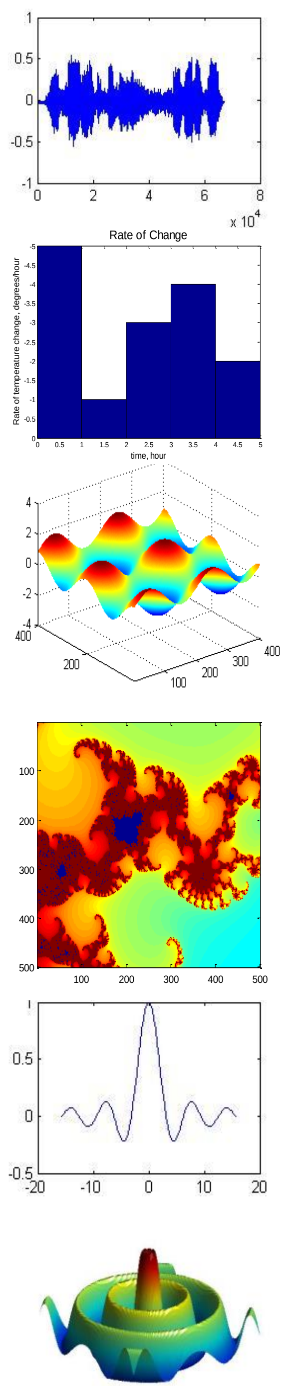






Symbolic Integration

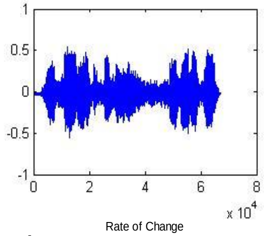
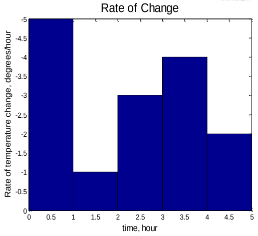
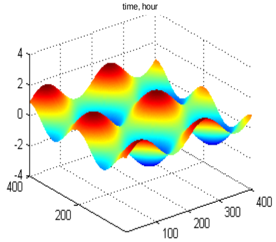
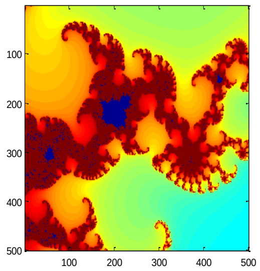
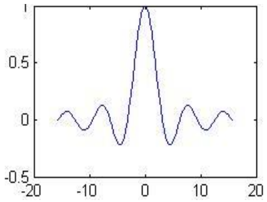
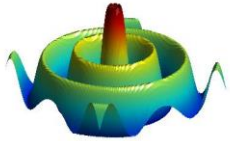
`int(f)`	Returns the integral of the expression **`f`** with respect to the default independent variable	**y=sym('x^3+z^2')** **int(y)** **ans =** **1/4*x^4+z^2*x**
`int(f,'t')`	Returns the integral of the expression **`f`** with respect to the variable **`t`**.	**y=sym('x^3+z^2')** **int(y,'z')** **ans =** **x^3*z+1/3*z^3**
`int(f,a,b)`	Returns the integral with respect to the default variable, of the expression **`f`** between the numeric bounds, a and b.	**y=sym('x^3+z^2')** **int(y,2,3)** **ans =** **65/4+z^2**
`int(f,'t',a,b)`	Returns the integral with respect to the variable **t**, of the expression **`f`** between the numeric bounds, a and b.	**y=sym('x^3+z^2')** **int(y,'z',2,3)** **ans =** **x^3+19/3**
`int(f,'t',a,b)`	Returns the integral with respect to the variable **t**, of the expression **`f`** between the symbolic bounds, a and b.	**y=sym('x^3+z^2')** **int(y,'z','a','b')** **ans =** **x^3*(b-a)+1/3*b^3-1/3*a^3**



Symbolic solution of differential equation

```
syms y(t) a
eqn = diff(y,t) == a*y;
S = dsolve(eqn)
```

$$S = C_1 e^{a t}$$

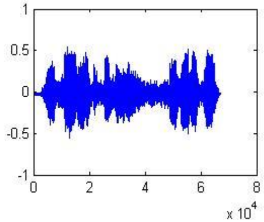
Second Order

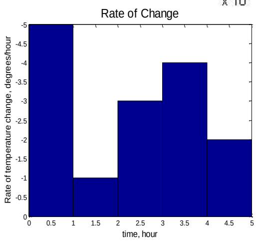
Solve the second-order differential equation $\frac{d^2y}{dt^2} = ay$.

Specify the second-order derivative of y by using `diff(y,t,2)` `dsolve`.

```
syms y(t) a
eqn = diff(y,t,2) == a*y;
ySol(t) = dsolve(eqn)
```

ySol(t) = $C_1 e^{-\sqrt{a}\,t} + C_2 e^{\sqrt{a}\,t}$





With initial conditions

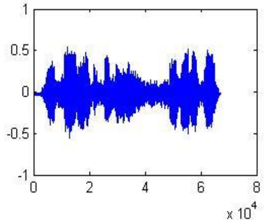
Next, solve the second-order differential equation $\frac{d^2y}{dt^2} = a^2 y$ with the initial conditions $y(0) = b$ and $y'(0) = 1$.

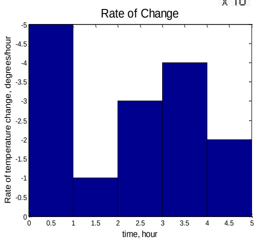
Specify the second initial condition by assigning `diff(y,t)` to `Dy` and then using `Dy(0) == 1`.

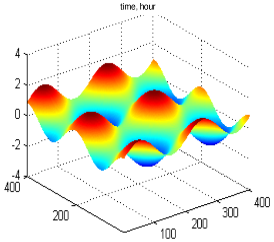
```
syms y(t) a b
eqn = diff(y,t,2) == a^2*y;
Dy = diff(y,t);
cond = [y(0)==b, Dy(0)==1];
ySol(t) = dsolve(eqn,cond)
```

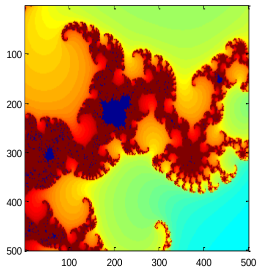
ySol(t) =

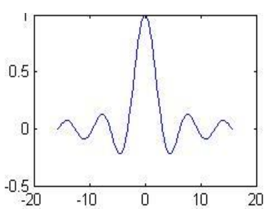
$$\frac{e^{a\,t}\,(a\,b+1)}{2\,a} + \frac{e^{-a\,t}\,(a\,b-1)}{2\,a}$$

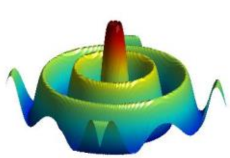












System of differential equations

$$\frac{dy}{dt} = z$$

$$\frac{dz}{dt} = -y.$$

```
syms y(t) z(t)
eqns = [diff(y,t) == z, diff(z,t) == -y];
S = dsolve(eqns)
```

```
S = struct with fields:
    z: C2*cos(t) - C1*sin(t)
    y: C1*cos(t) + C2*sin(t)
```

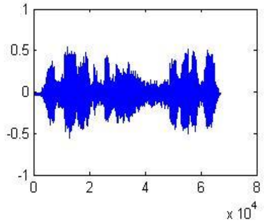
Without assignment

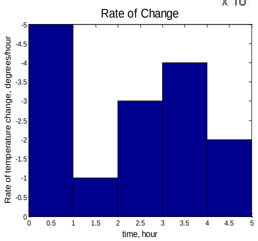
```
syms y(t) z(t)
eqns = [diff(y,t)==z, diff(z,t)==-y];
[ySol(t),zSol(t)] = dsolve(eqns)
```

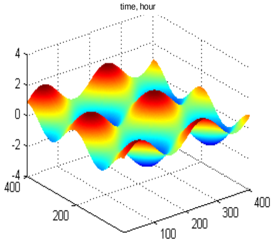
$$\mathtt{ySol(t)} = C_1\cos(t) + C_2\sin(t)$$

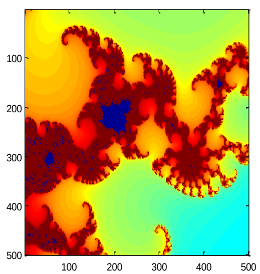
With Assignment

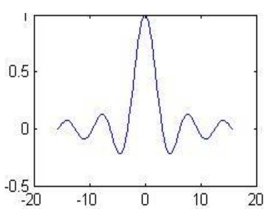
$$\mathtt{zSol(t)} = C_2\cos(t) - C_1\sin(t)$$

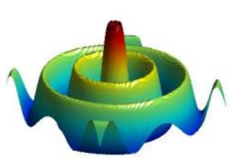












Solving the differential equations

```
syms y(t)
eqn = diff(y) == y+exp(-y)
```

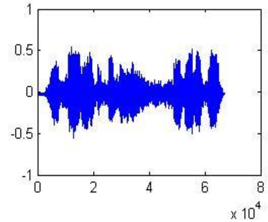
eqn(t) =

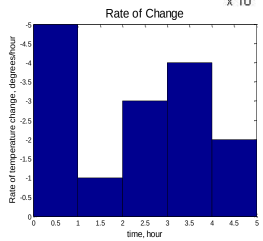
$$\frac{\partial}{\partial t}\, y(t) = e^{-y(t)} + y(t)$$

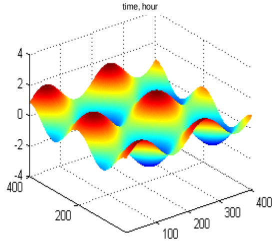
```
sol = dsolve(eqn)
```

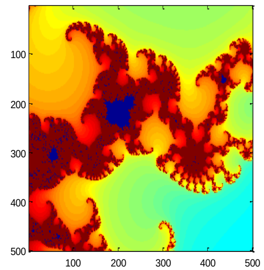
sol = $W_0(-1)$

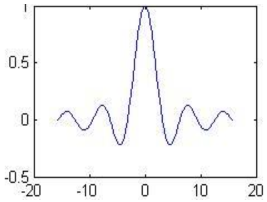
ev al
rite to:

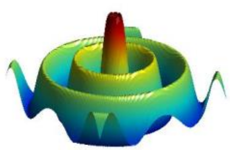












Solve the differential equation $\dfrac{dy}{dx} = \dfrac{1}{x^2} e^{-\frac{1}{x}}$ without specifying the initial condition.

```
syms y(x)
eqn = diff(y) == exp(-1/x)/x^2;
ySol(x) = dsolve(eqn)
```

```
ySol(x) =
```

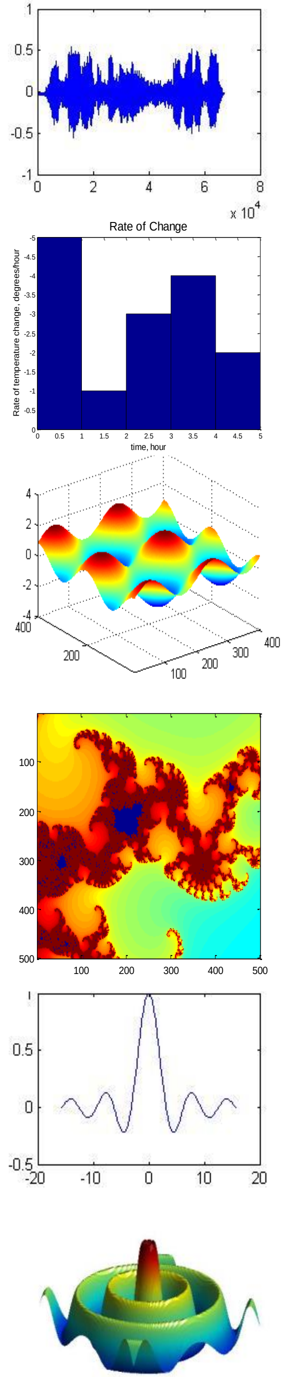
$$C_1 + e^{-\frac{1}{x}}$$

To eliminate constants from the solution, specify the initial condition $y(0) = 1$.

```
cond = y(0) == 1;
S = dsolve(eqn,cond)
```

```
S =
```

$$e^{-\frac{1}{x}} + 1$$

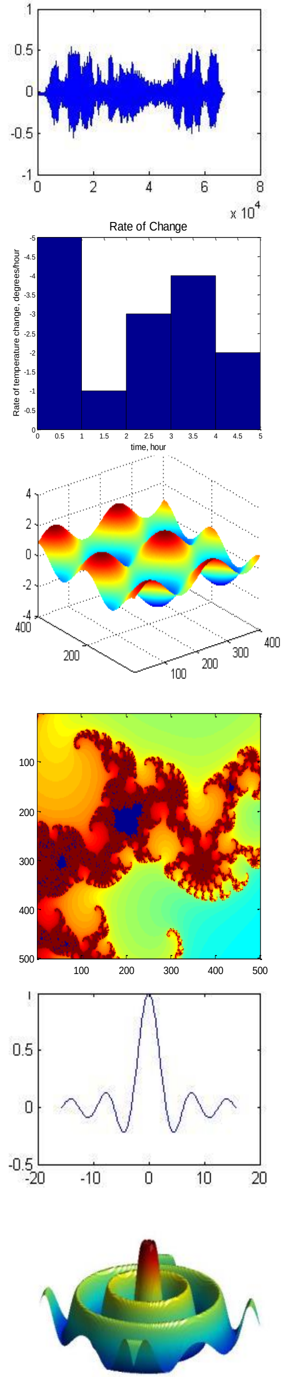


12.5 Differential Equations

- Differential equations contain both
  - the derivative of the dependent variable with respect to the independent variable
  - the dependent variable

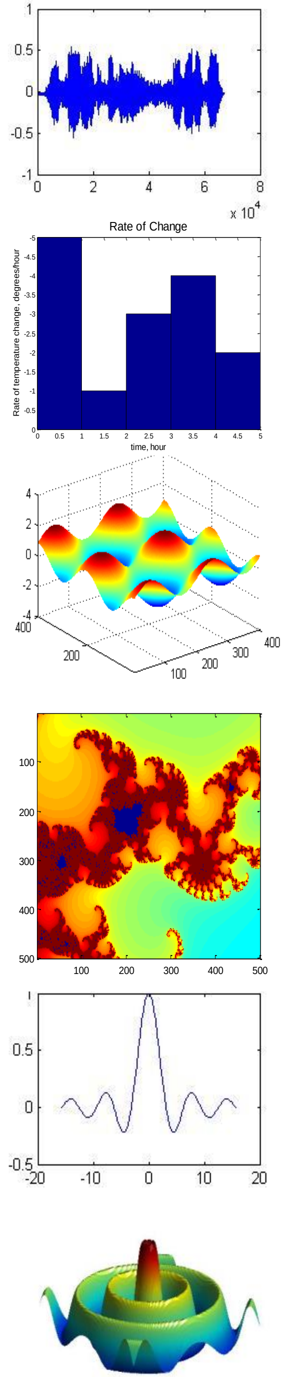
$$\frac{dy}{dt} = y$$

is a differential equation



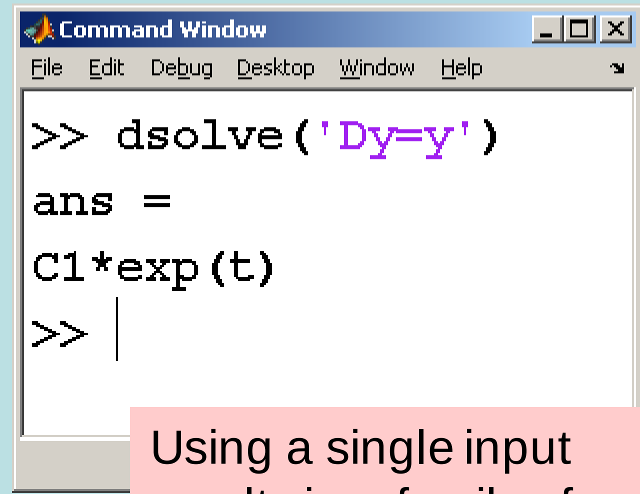
Default variable

- Although any symbol can be used for either the independent or the dependent variable, the default independent variable is t in MATLAB (and is the usual choice for most ordinary differential equation formulations.)



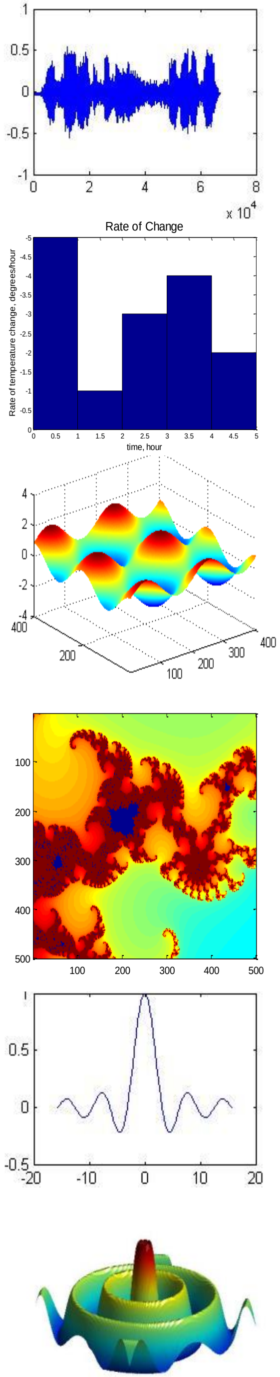
dsolve

- When we solve a differential equation, we are looking for an expression for y in terms of t
- dsolve requires the differential equation as input
  - use the symbol D to specify derivatives with respect to the independent variable



Using a single input results in a family of results

dsolve is a “function function”

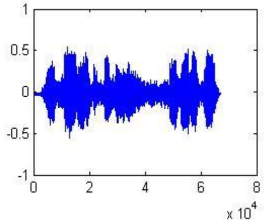


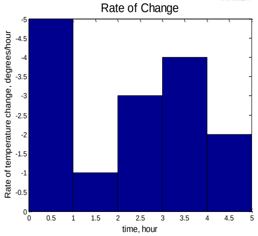
Command Window

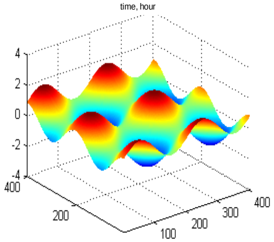
File Edit Debug Desktop Window Help

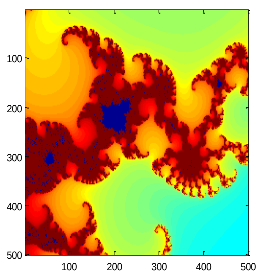
```
>> dsolve('Dy = y')
ans =
C2*exp(t)
>> dsolve('Dy = y','y(0)=1')
ans =
exp(t)
>>
```

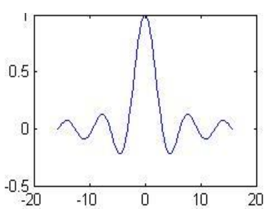
Specify an initial or boundary condition in the second field

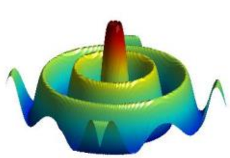




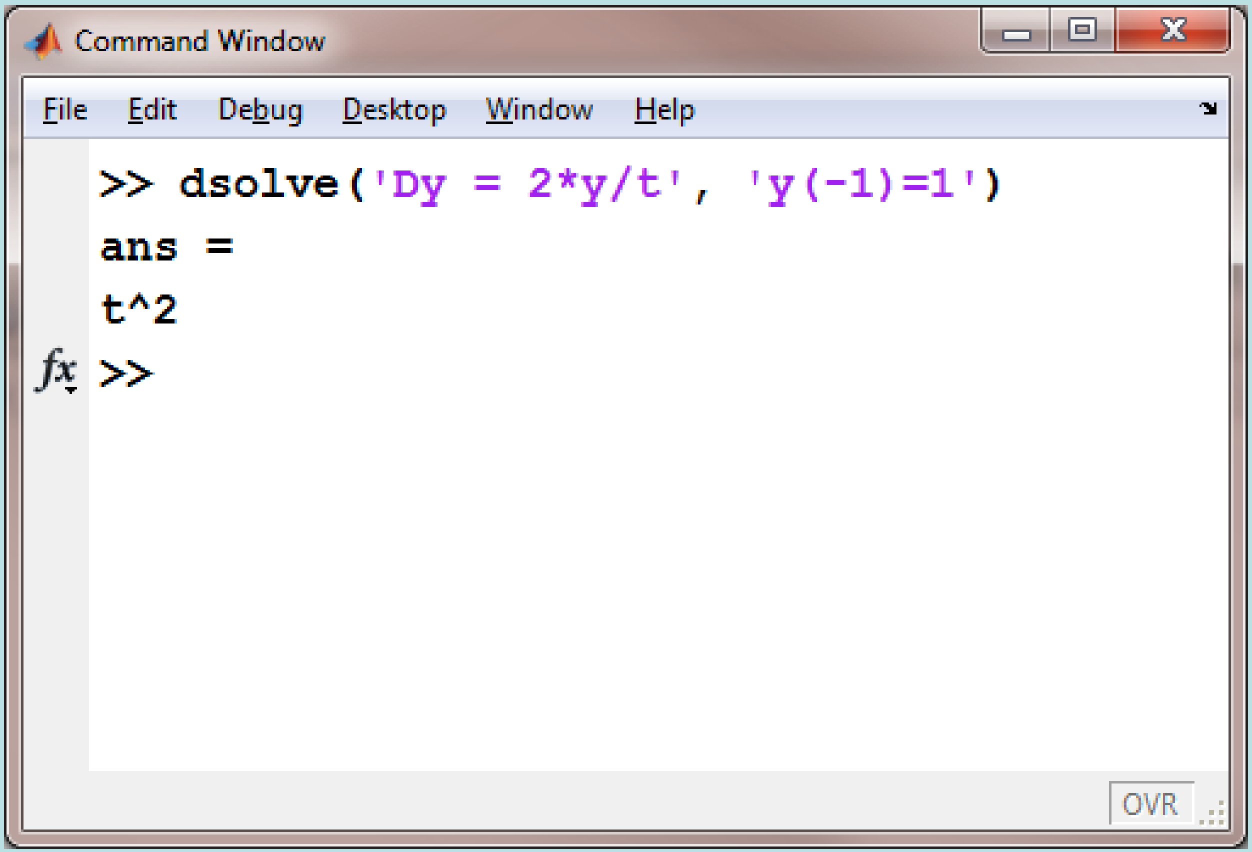


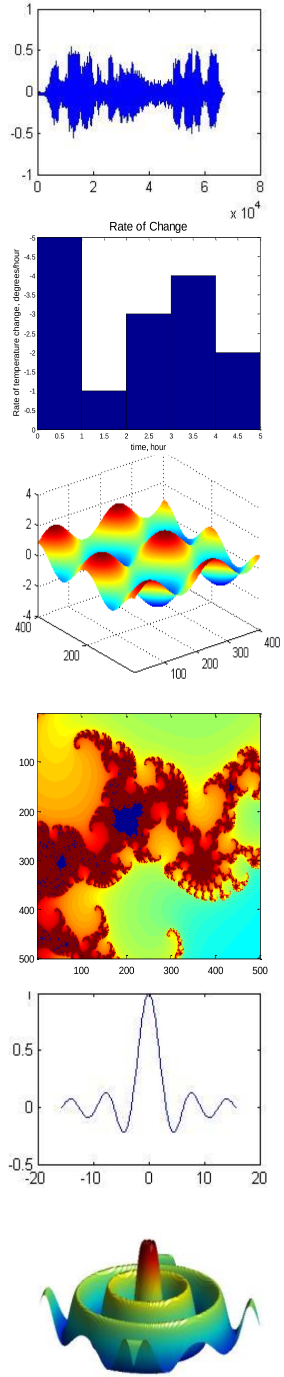




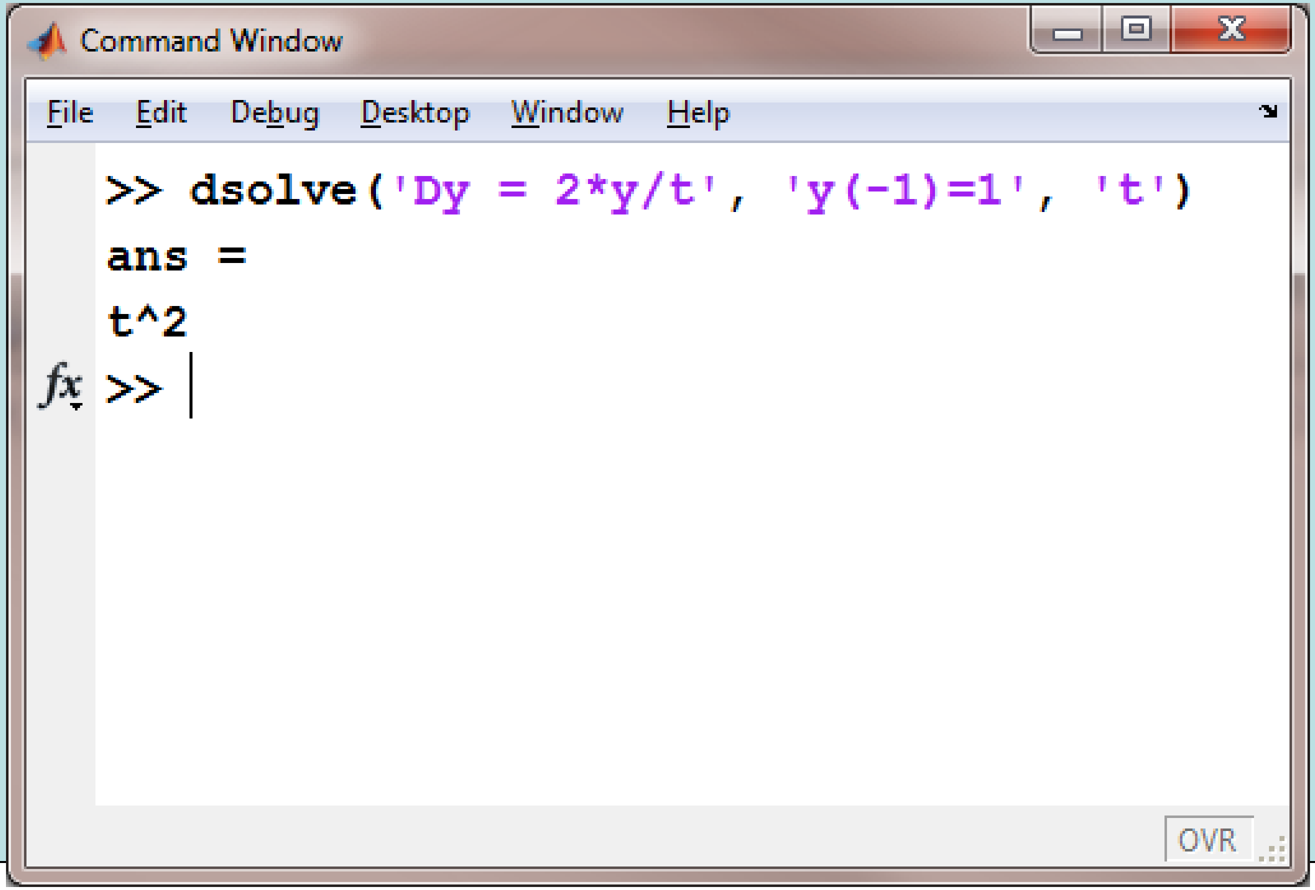


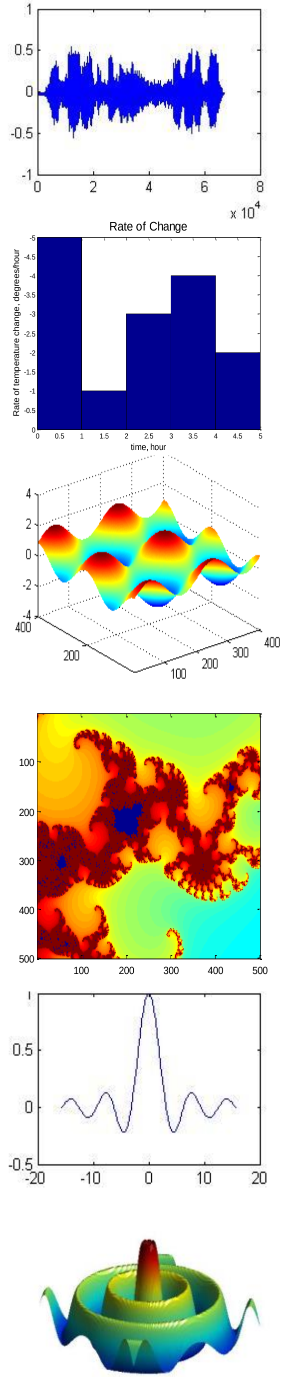
Here's a more complicated example



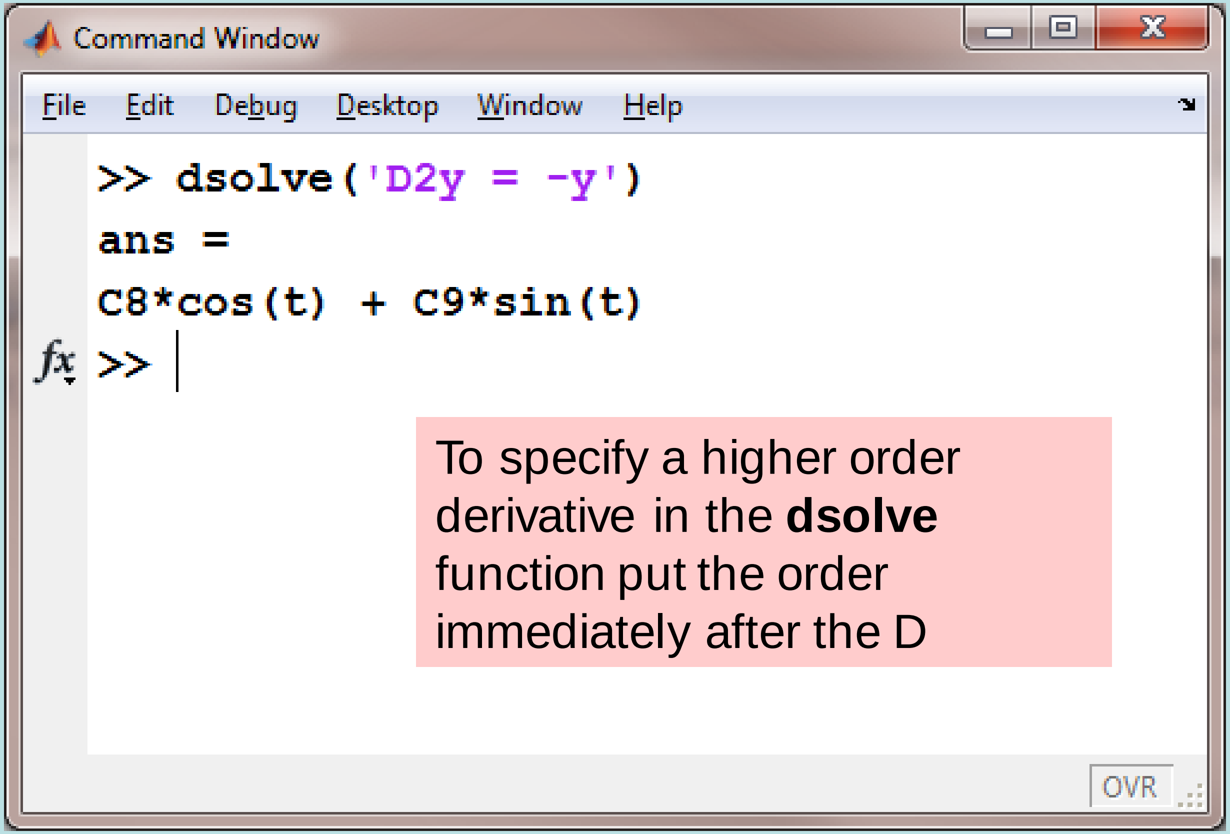


You can specify the independent variable in the third field

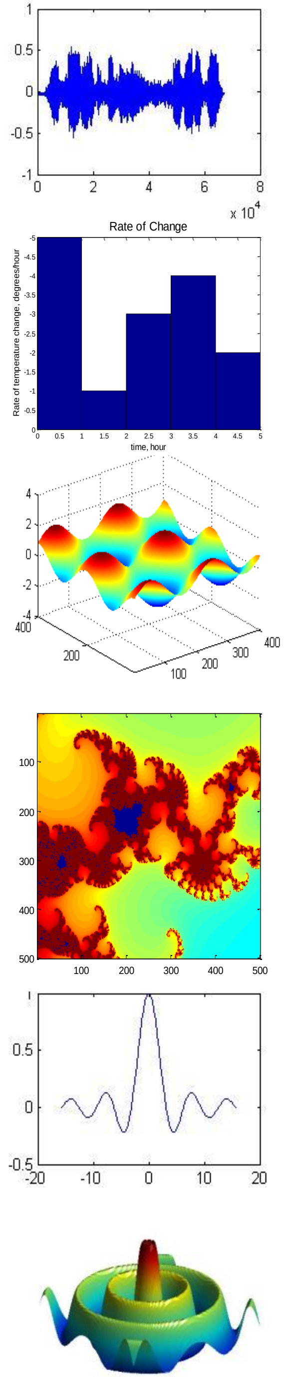




Higher Order Derivatives

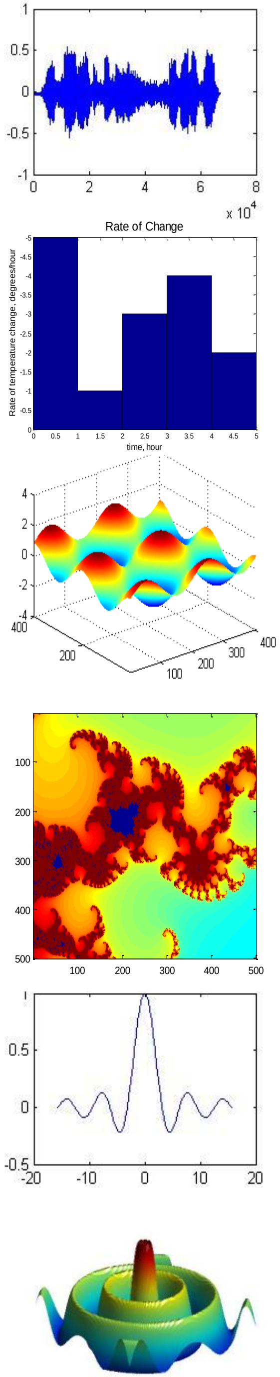


To specify a higher order derivative in the **dsolve** function put the order immediately after the D



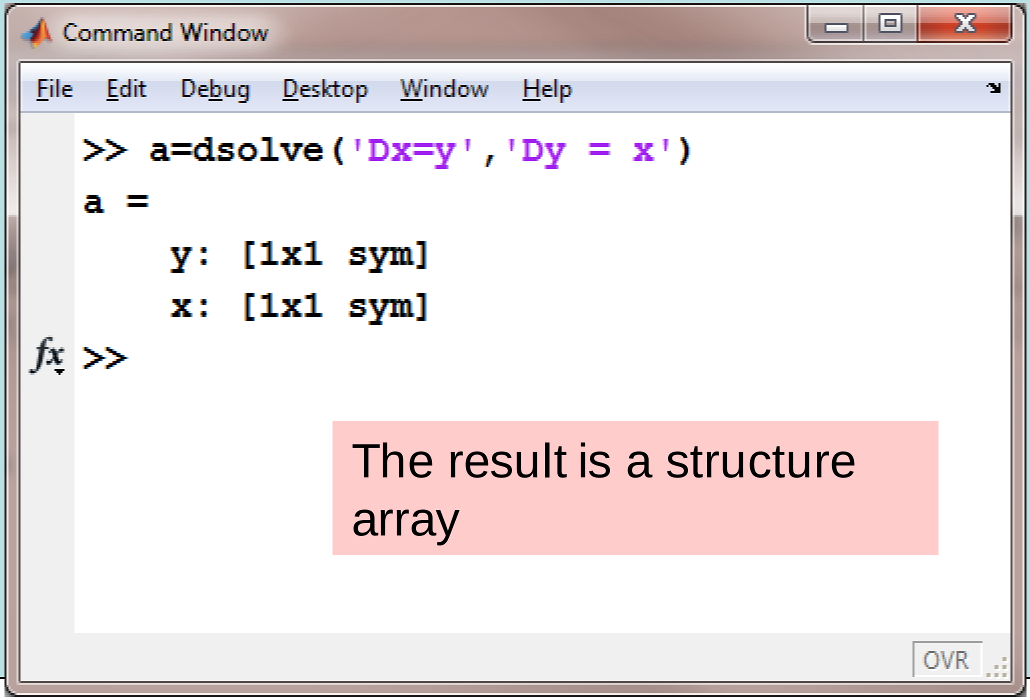
Hint

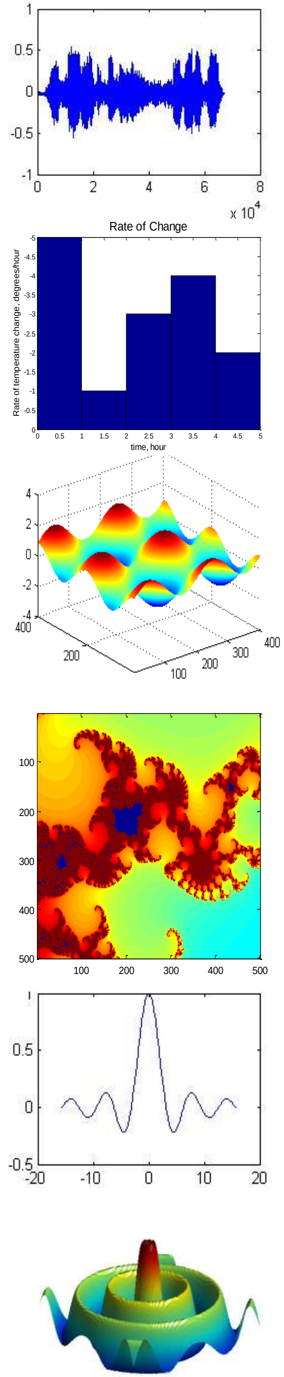
- Don’t use the letter D in your variable names in differential equations.
- It will confuse the function into thinking you are trying to specify a derivative



Use the dsolve function to solve systems of equations

- **dsolve('eq1,eq2,...', 'cond1,cond2,...', 'v')**



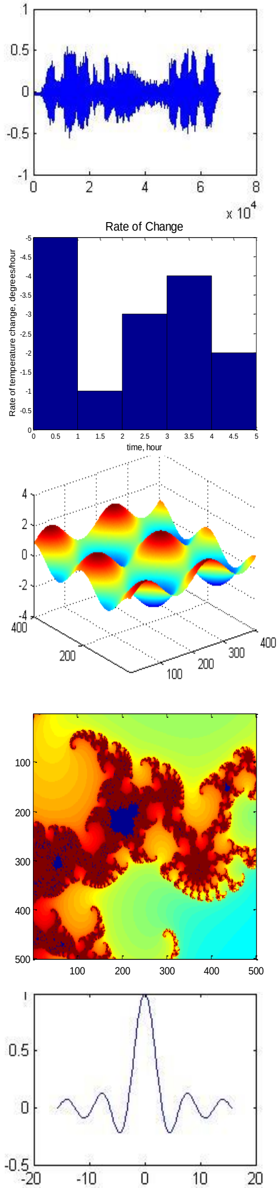


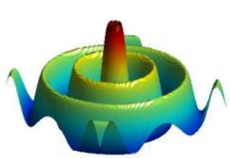
Command Window

File Edit Debug Desktop Window Help

```
>> a=dsolve('Dx=y','Dy = x')
a =
     y: [1x1 sym]
     x: [1x1 sym]
>> a.x
ans =
C10*exp(t) - C11/exp(t)
>> a.y
ans =
C10*exp(t) + C11/exp(t)
>>
```

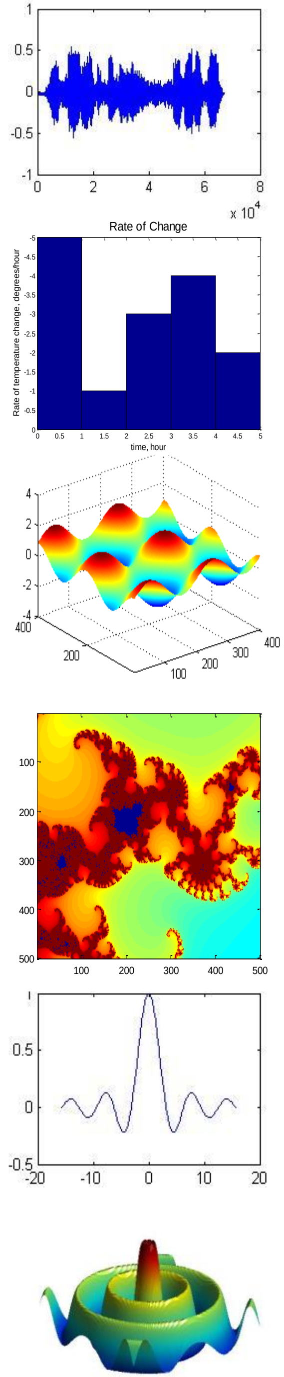
OVR





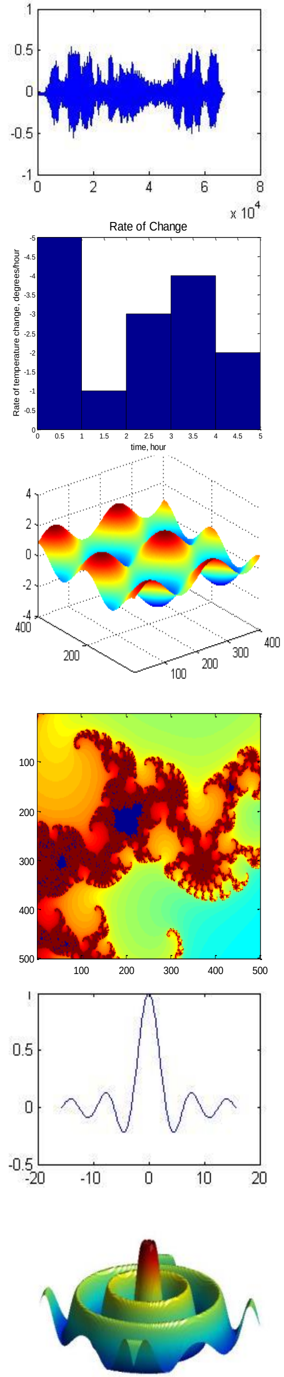
MATLAB can not solve every differential equation symbolically.

- For complicated (or ill behaved) systems of equations you may find it easier to use MuPad
  - Remember that MATLAB’s symbolic capability is based on the MuPad engine
- There are many differential equations that can’t be solved analytically at all
  - The numerical techniques described in Chapter 13 can be used to solve many of these equations.

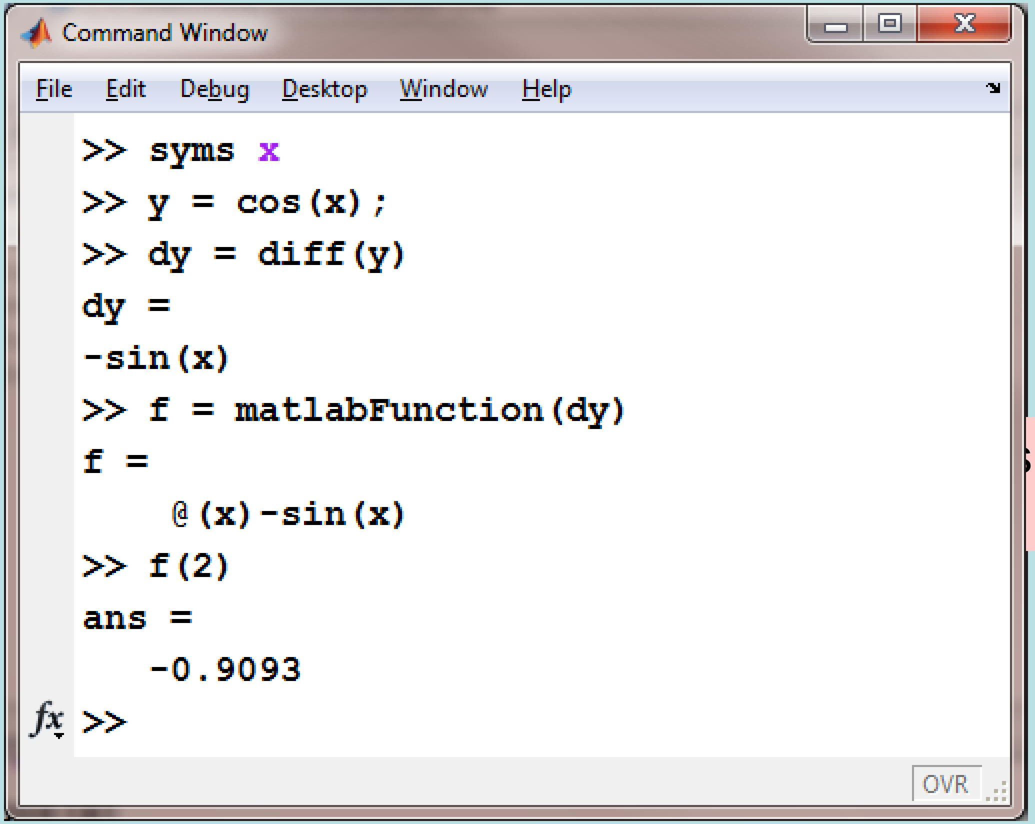


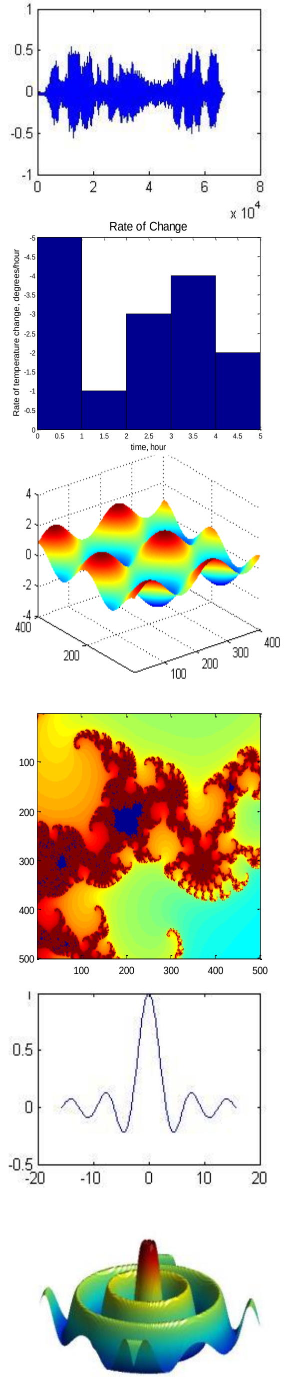
12.6 Converting Symbolic Expressions to MATLAB functions

- It is often useful to manipulate expressions symbolically … but then to perform numeric calculations using more traditional MATLAB functions
- matlabFunction converts a symbolic expression to an anonymous function



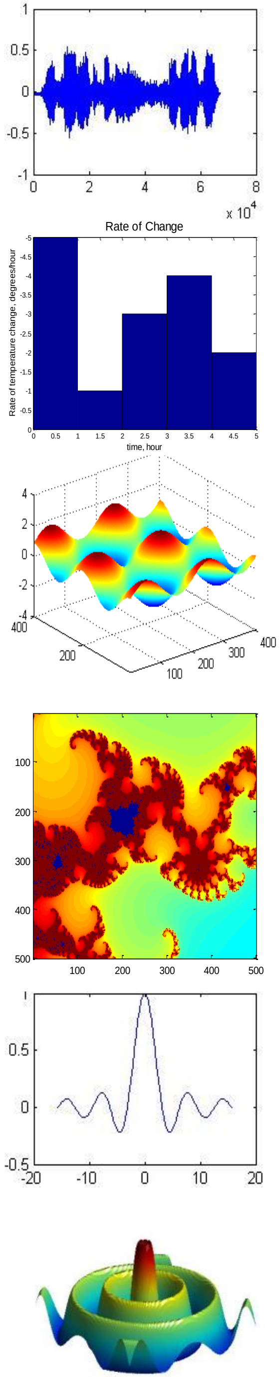
matlabFunction





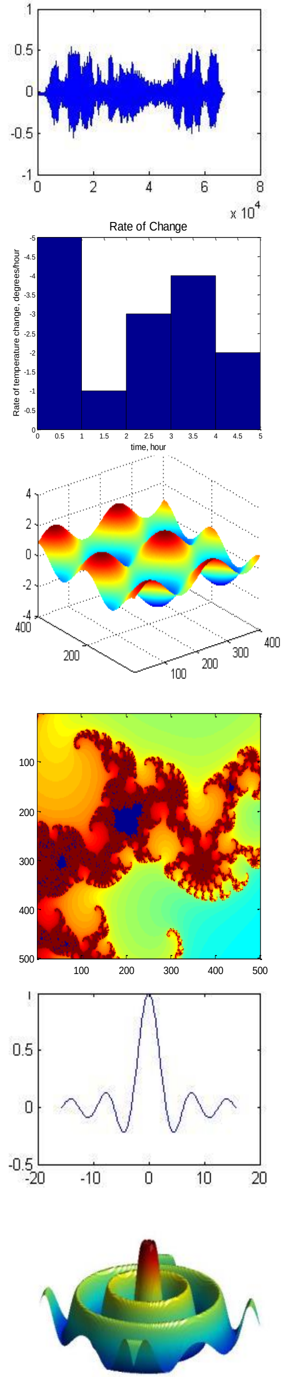
Summary

- MATLAB uses MuPad as its symbolic engine
- The symbolic toolbox is an optional component of the professional version
- A subset is included with the student version



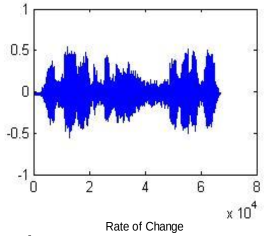
Summary – Variable Definition

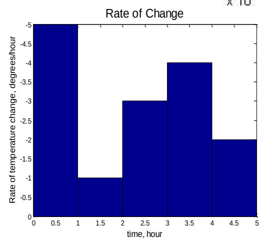
- Use either
  - sym
  - syms
- The sym command can be used to create symbolic expressions or equations
- The syms command can create multiple symbolic variables in one step

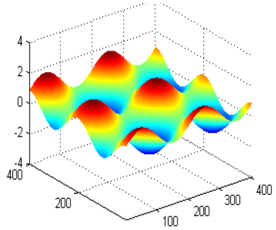


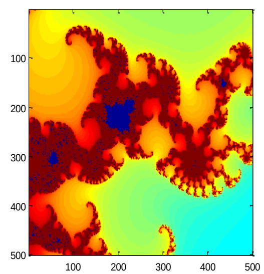
Summary – Composition of expressions

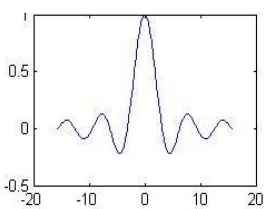
- Once symbolic variables have been created they can be used to create more complicated expression

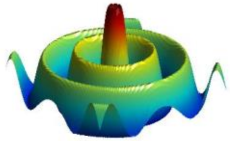






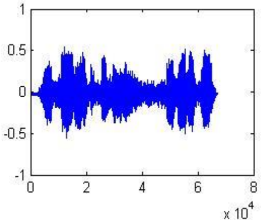
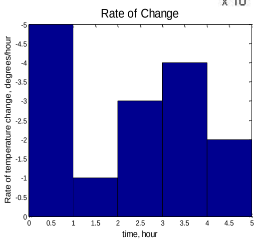


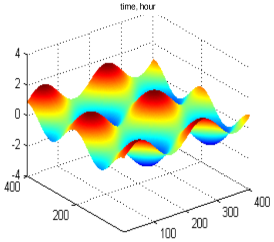
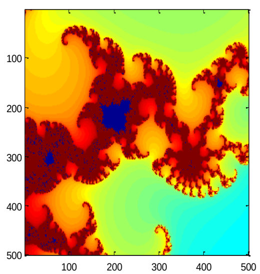
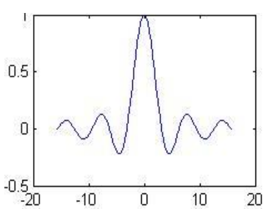
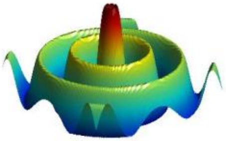




Summary
Equations vs Expressions

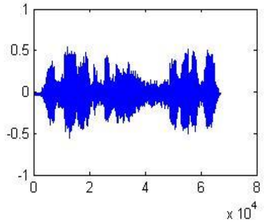
- Equations are set equal to something
- Expressions are not
- If you set one expression equal to another, you've created an equation

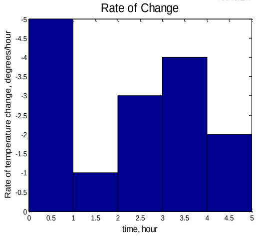



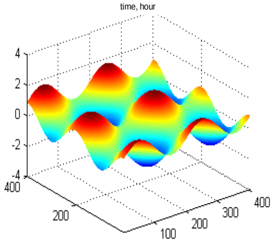





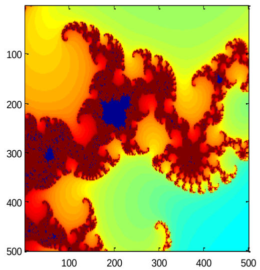
Summary – Symbolic functions

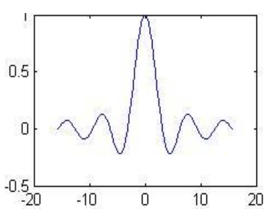
- numden
- expand
- factor
- collect
- simplify
- simple

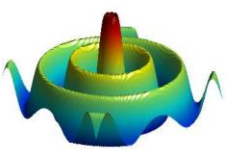






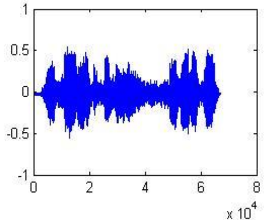


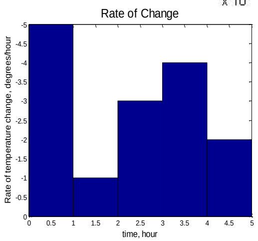


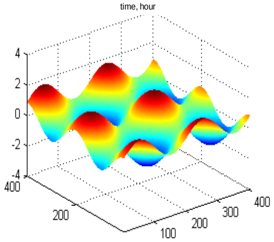


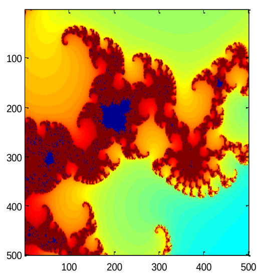
Summary – Solve

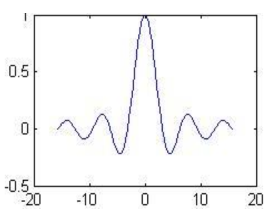
- If the input to solve is an expression MATLAB sets it equal to 0 and solves
- If the input is an equation, MATLAB solves the equation for either the default variable, or a user defined variable
- solve can also solve systems of equations

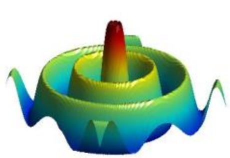






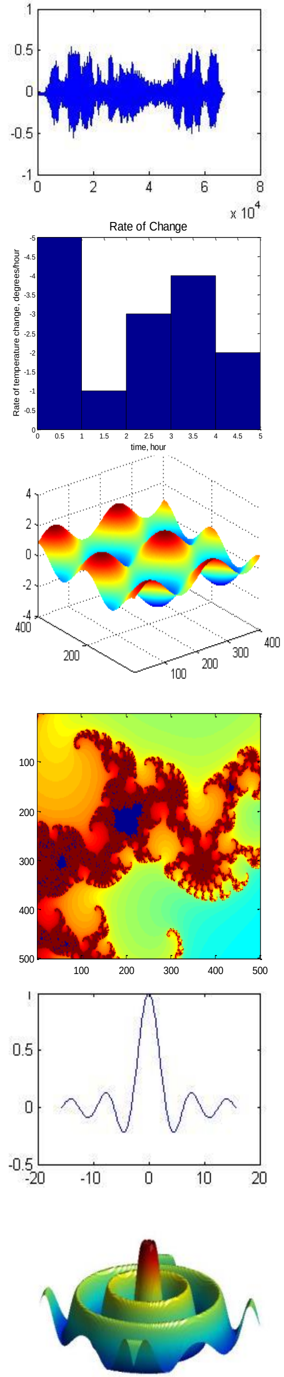






Summary - dsolve

- Used to solve differential equations
- D signifies a derivative
- Can be used to solve systems of equations
- Not all differential equations can be solved analytically



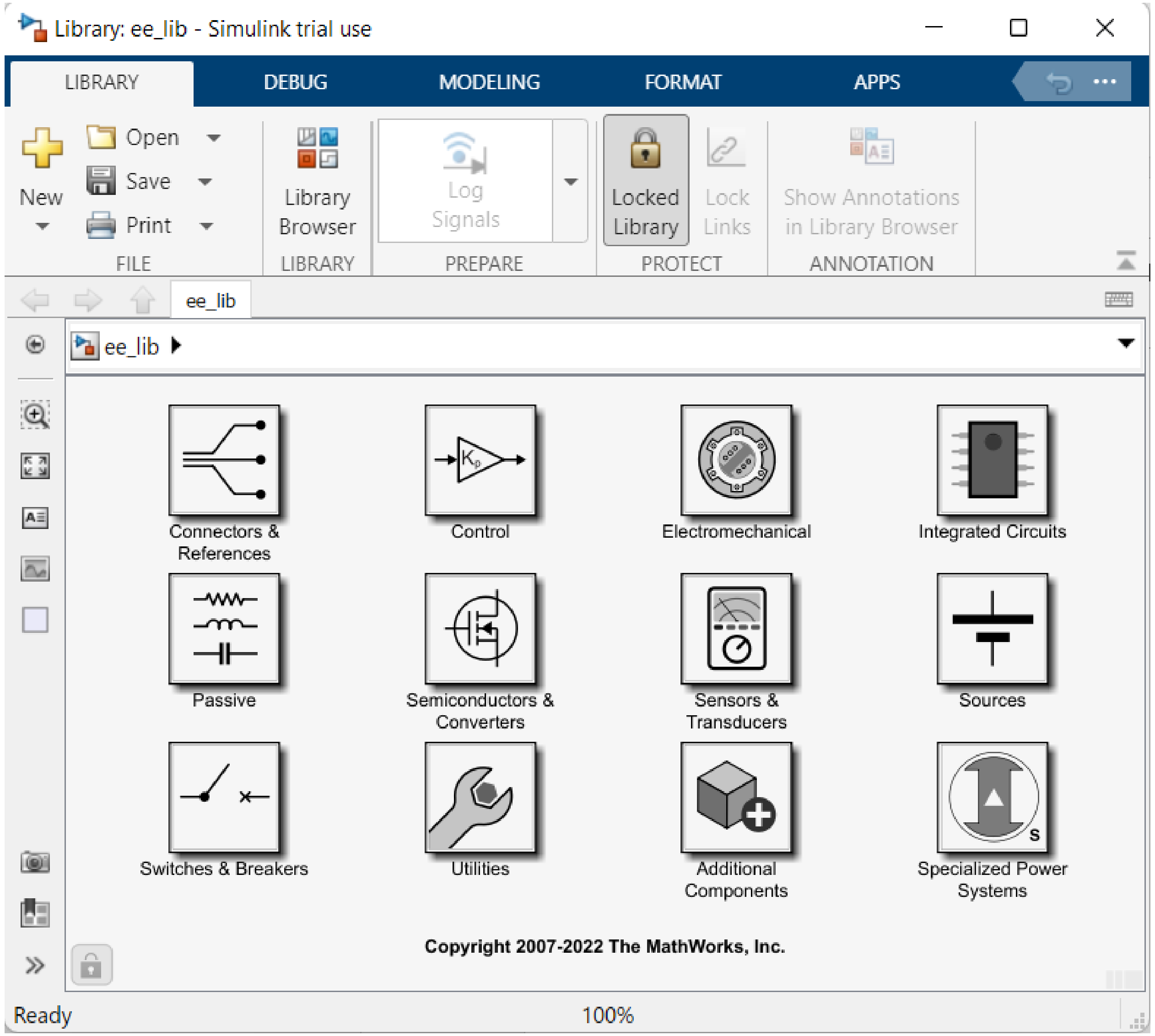
Summary - Calculus

- diff - finds the derivative
- int - takes the integral

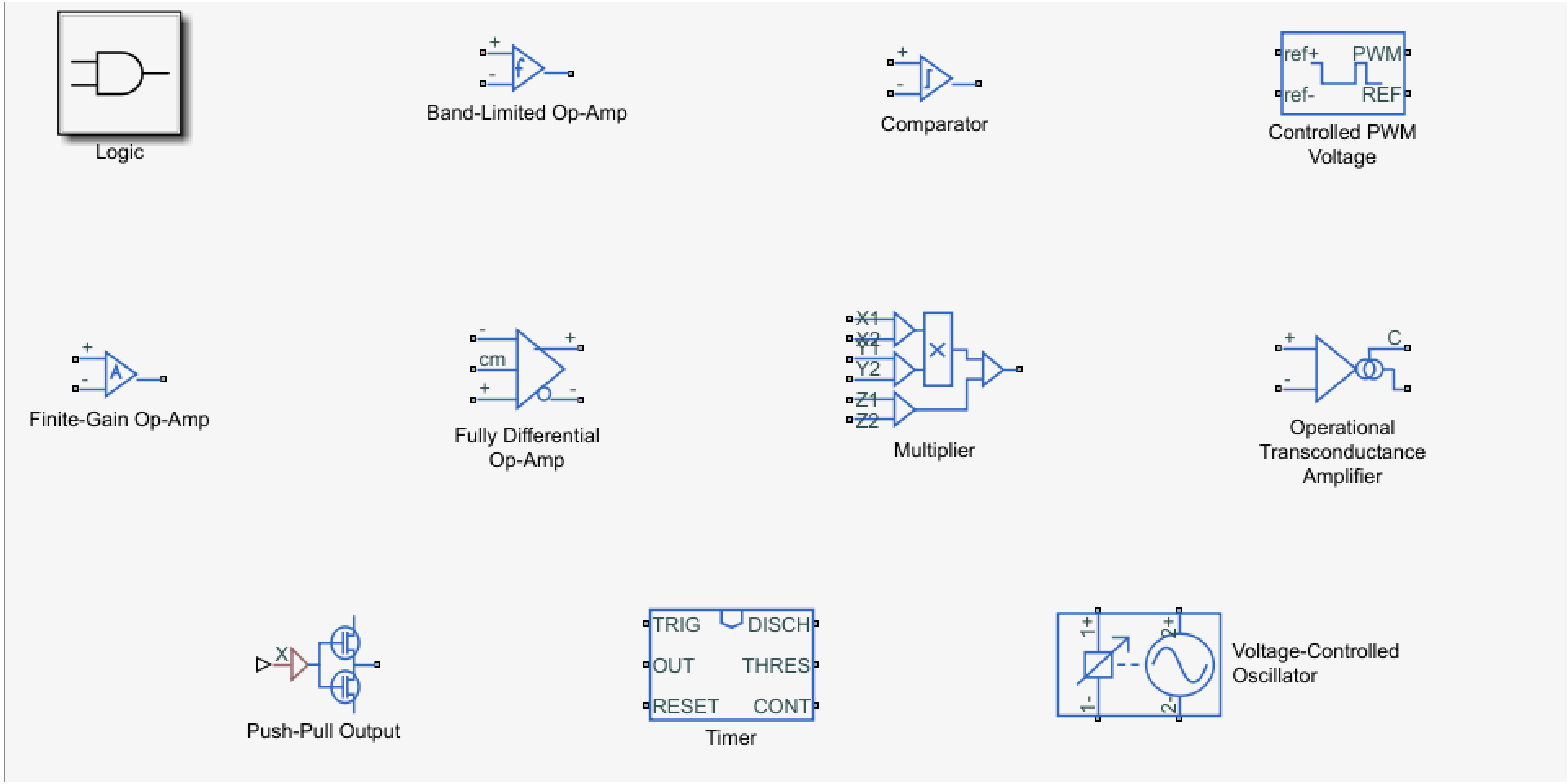
Simscape electrical

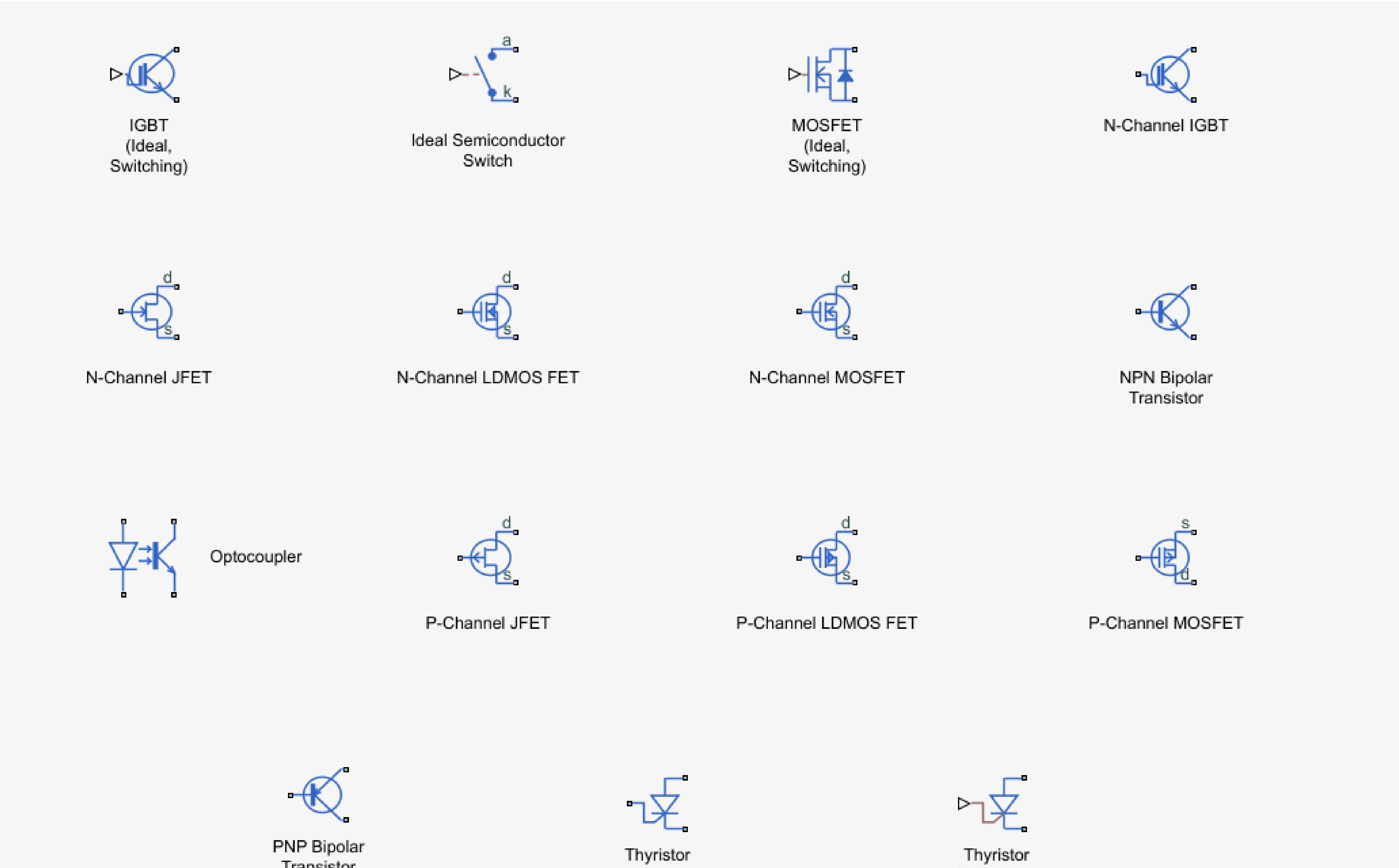
>>ee_lib

The following is the electrical engineering library >>ee_lib

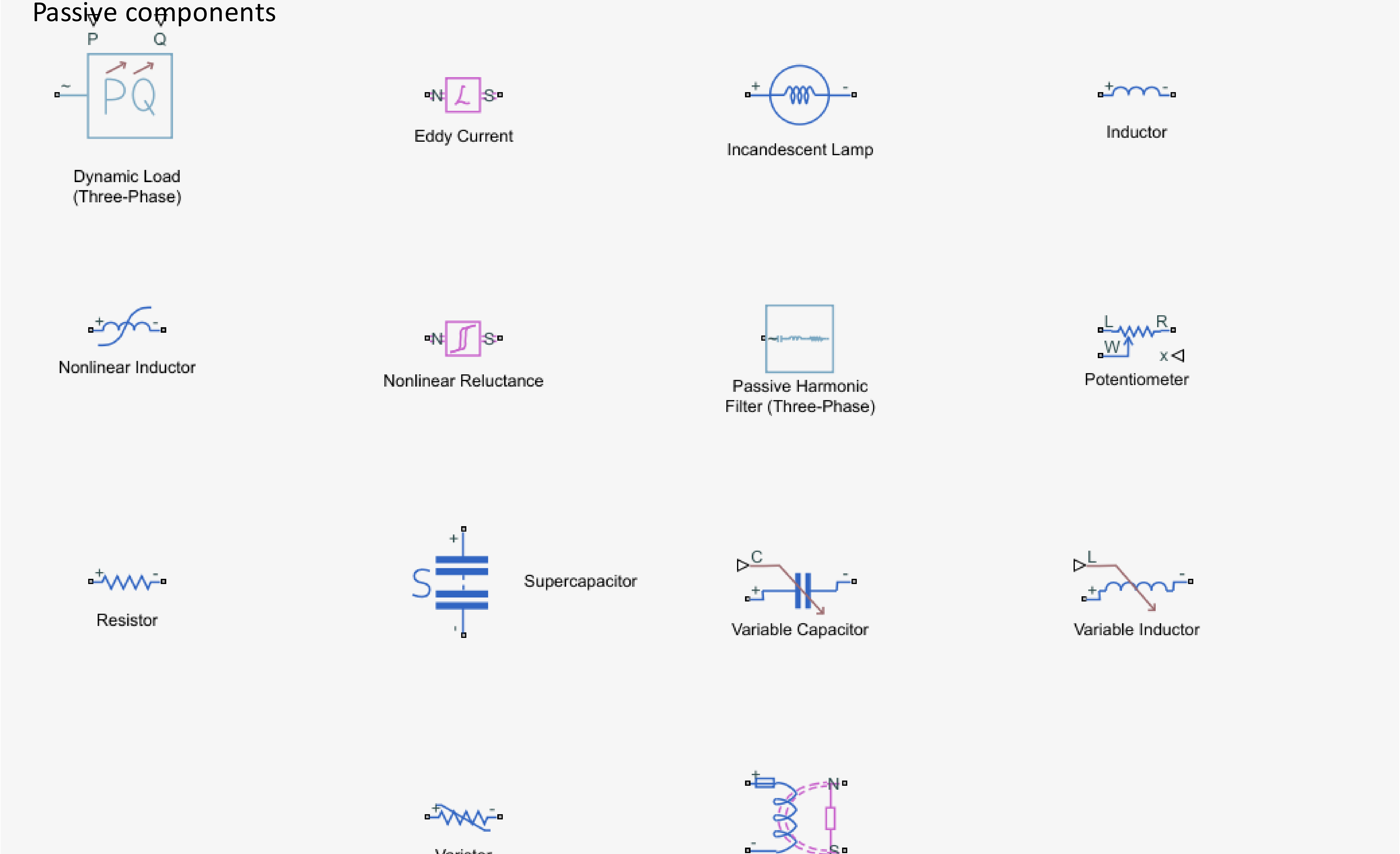


Integrated circuits



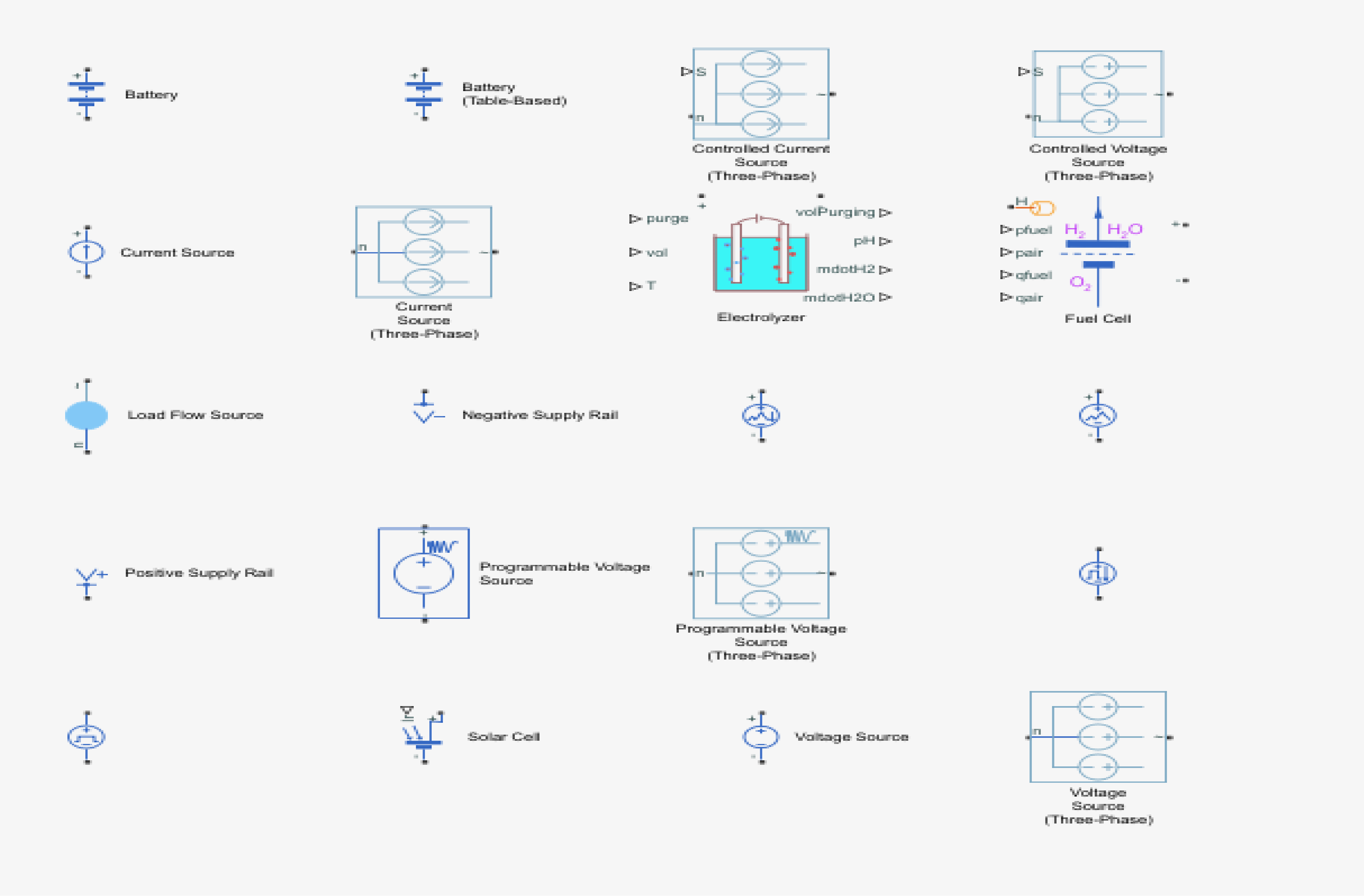

a
k
IGBT (Ideal, Switching)
Ideal Semiconductor Switch
MOSFET (Ideal, Switching)
N-Channel IGBT
N-Channel JFET
N-Channel LDMOS FET
N-Channel MOSFET
NPN Bipolar Transistor
Optocoupler
P-Channel JFET
P-Channel LDMOS FET
P-Channel MOSFET
PNP Bipolar
Thyristor
Thyristor

Passive components



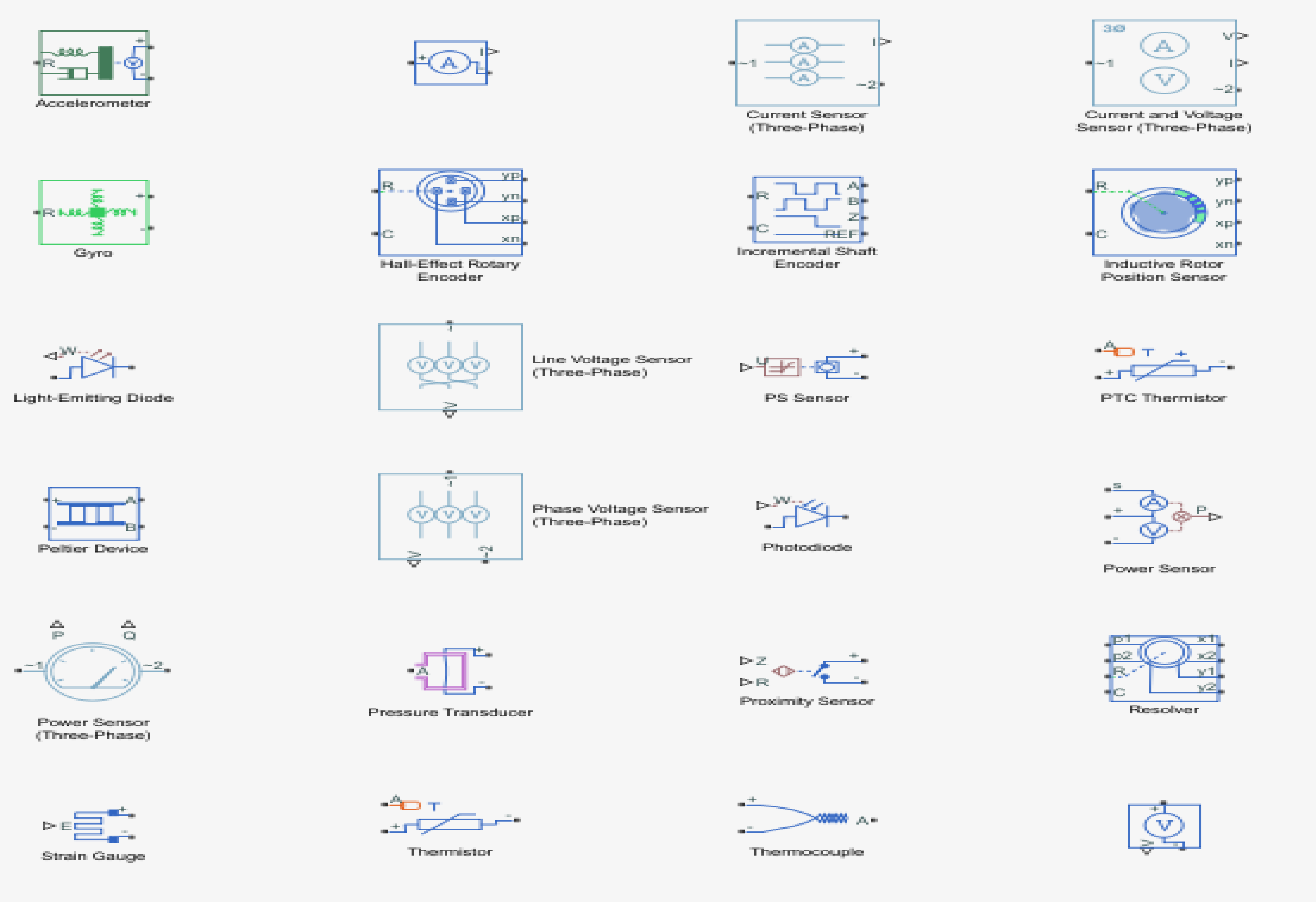
Electromechanical

Sources

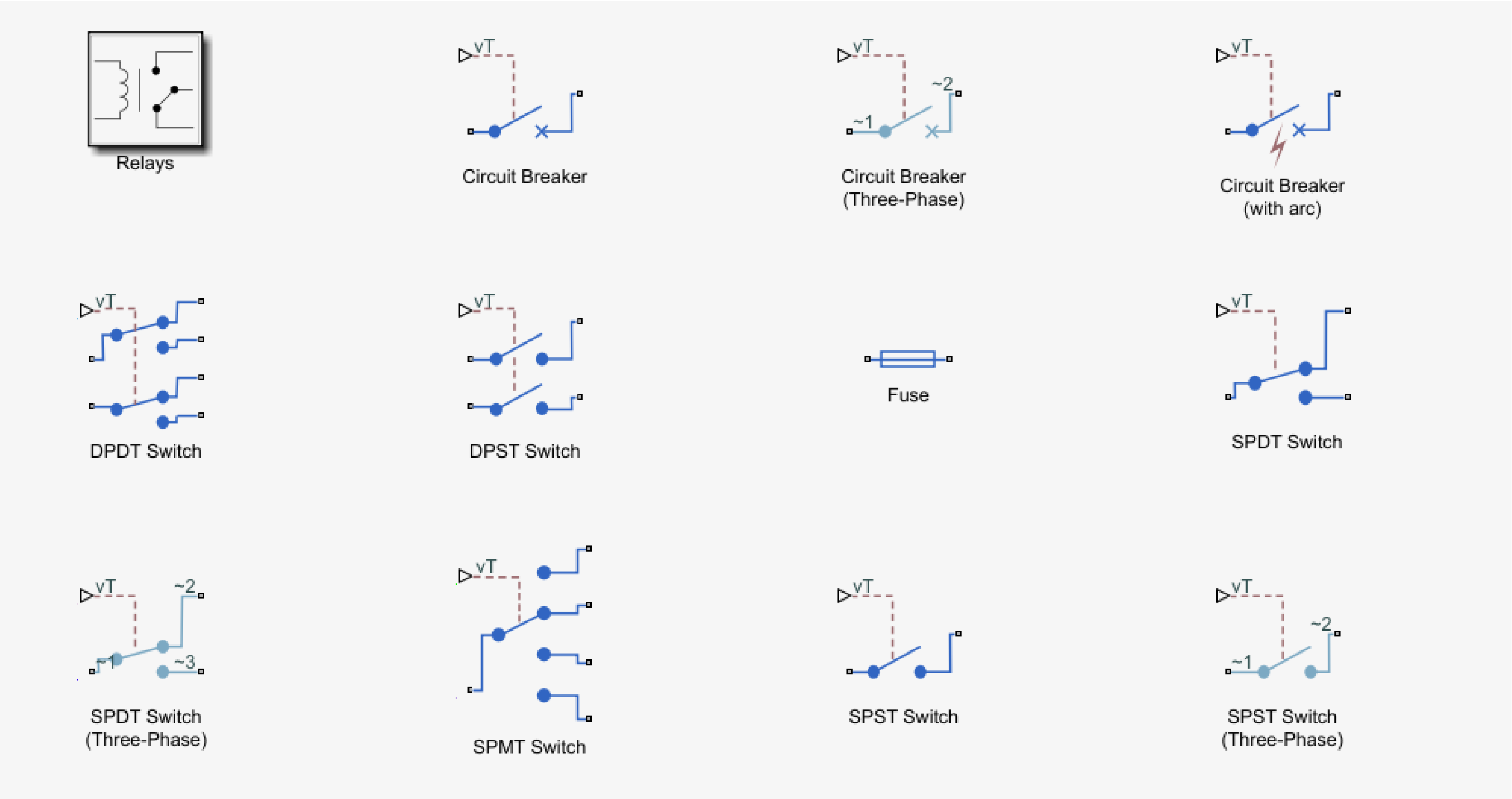


Battery
Battery (Table-Based)
Controlled Current Source (Three-Phase)
Controlled Voltage Source (Three-Phase)
Current Source
Current Source (Three-Phase)
purge
vol
T
volPurging
pH
mdotH2
mdotH2O
Electrolyzer
pfuel
pair
qfuel
qair
Fuel Cell
Load Flow Source
Negative Supply Rail
Positive Supply Rail
Programmable Voltage Source
Programmable Voltage Source (Three-Phase)
Solar Cell
Voltage Source
Voltage Source (Three-Phase)

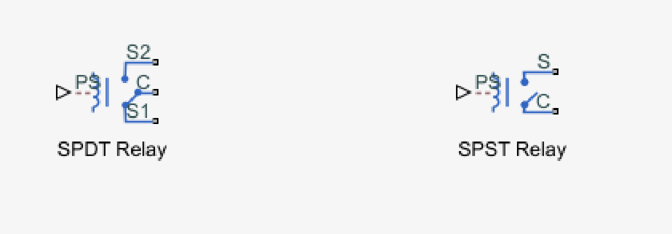
Sensors and transducers



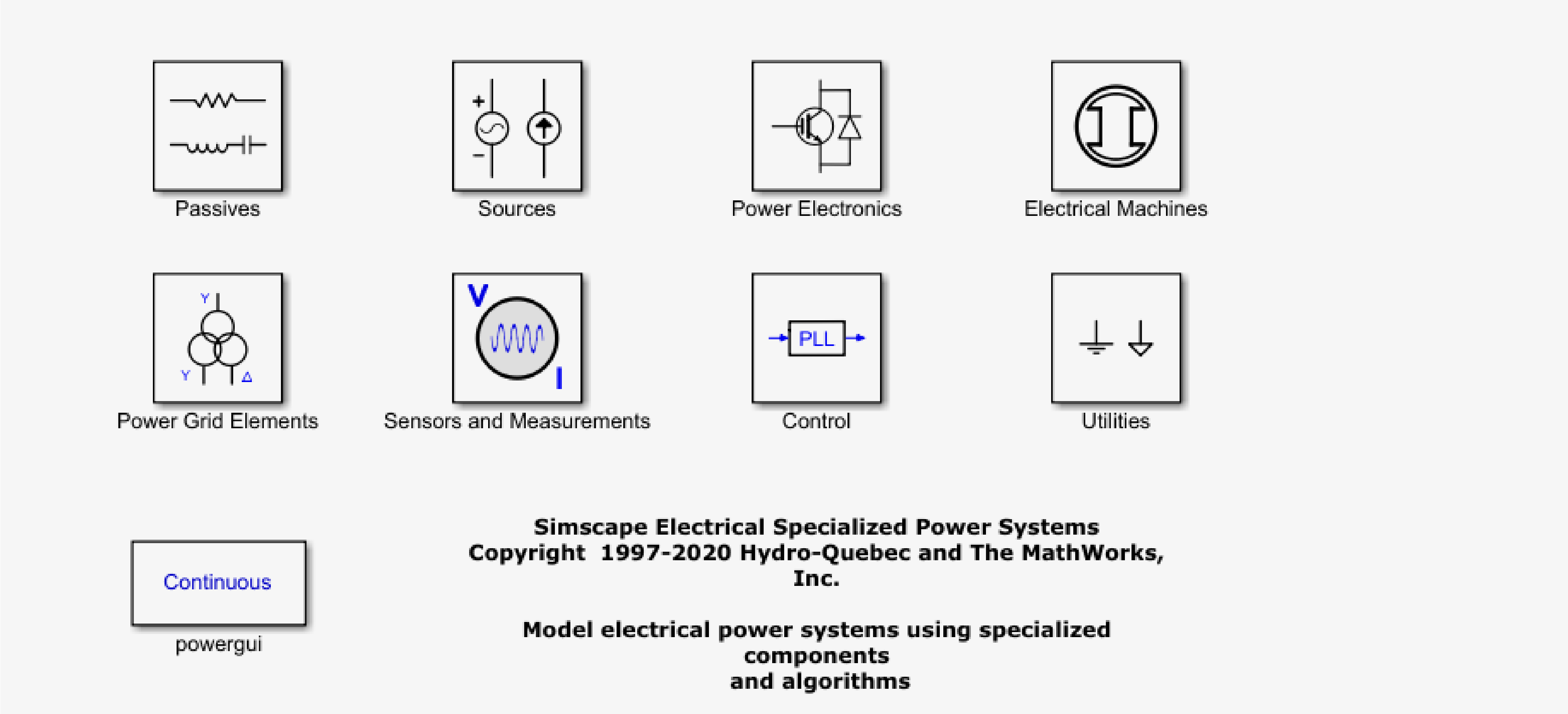
Switches and breakers



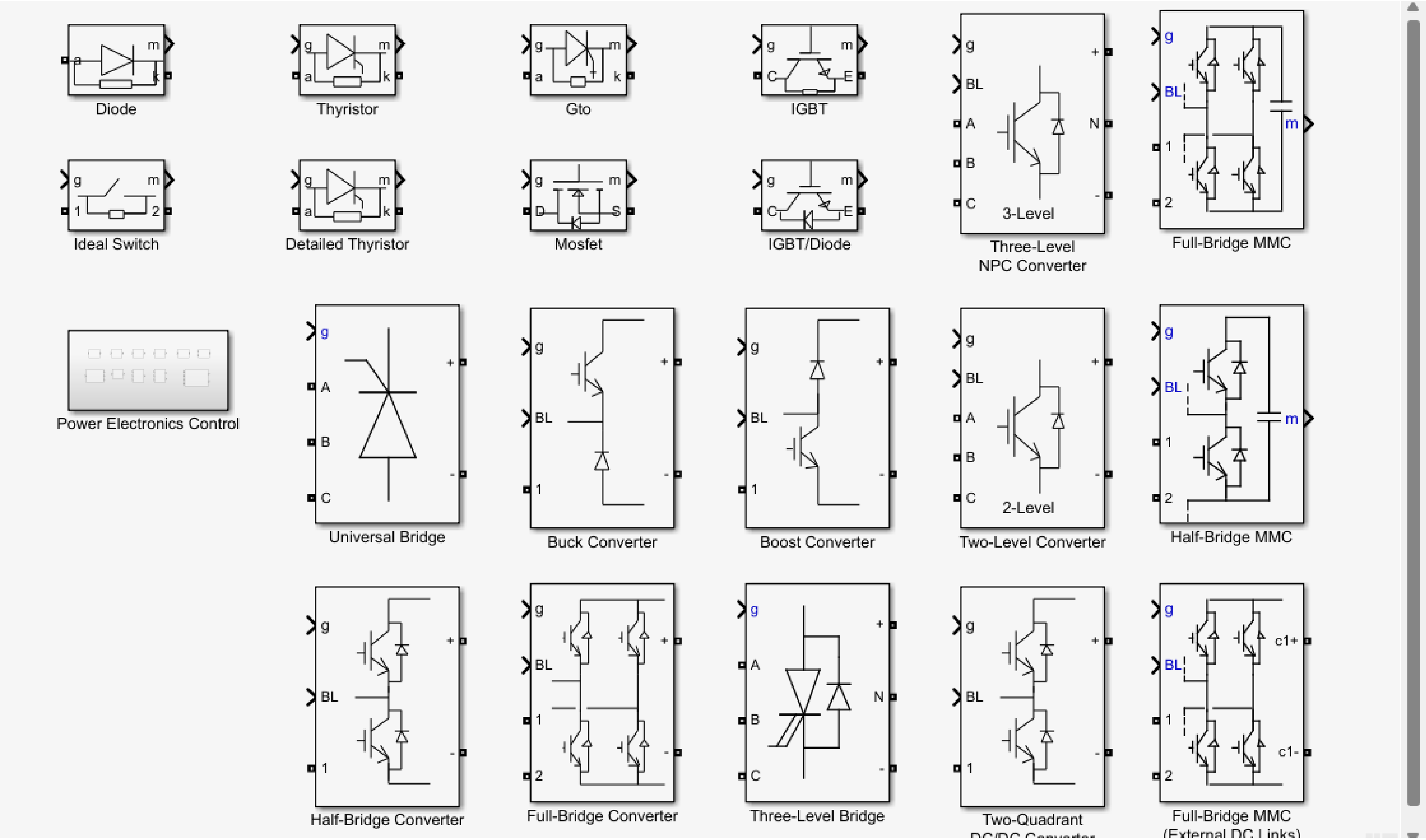
Relays

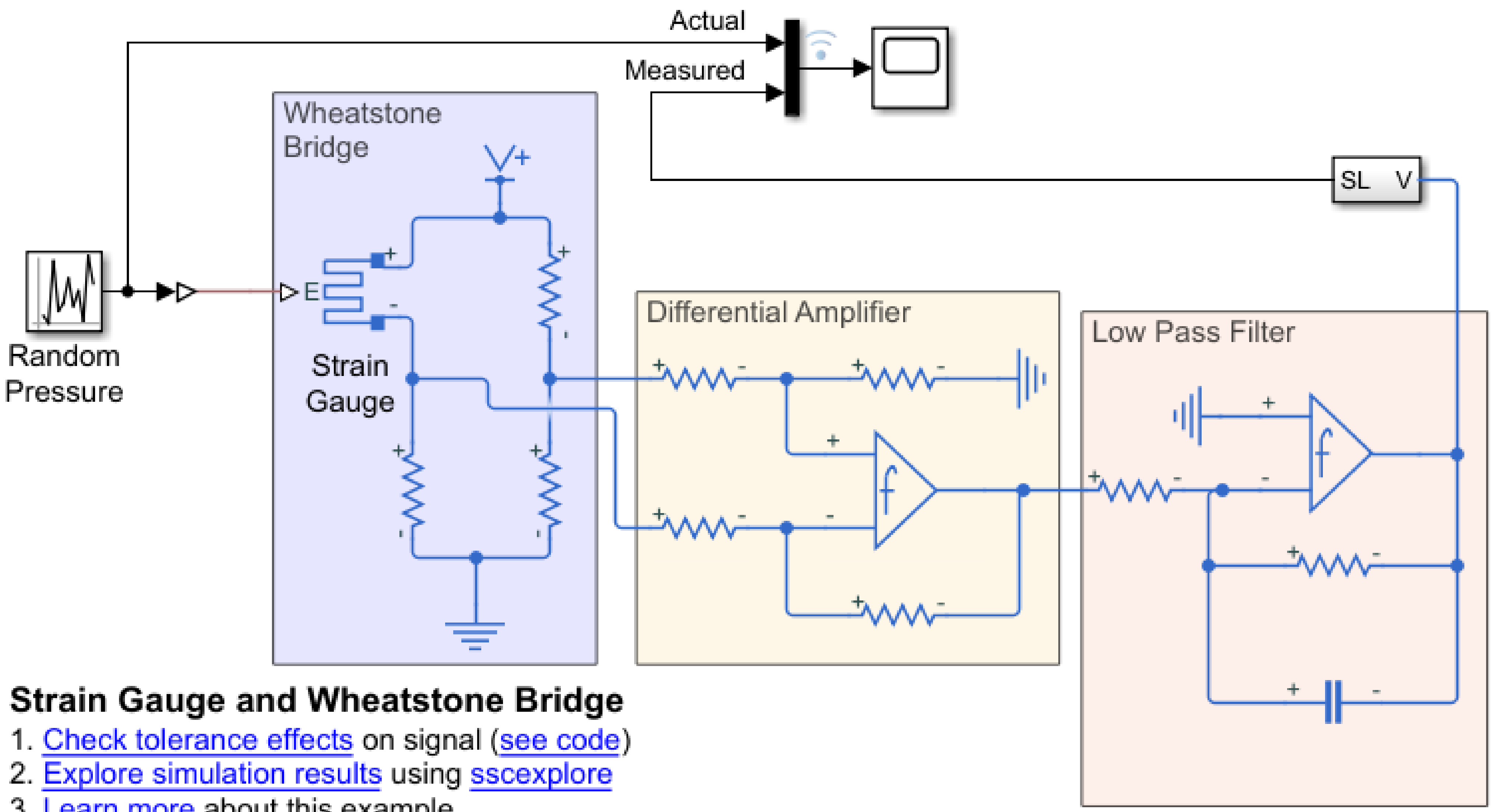


Specialised



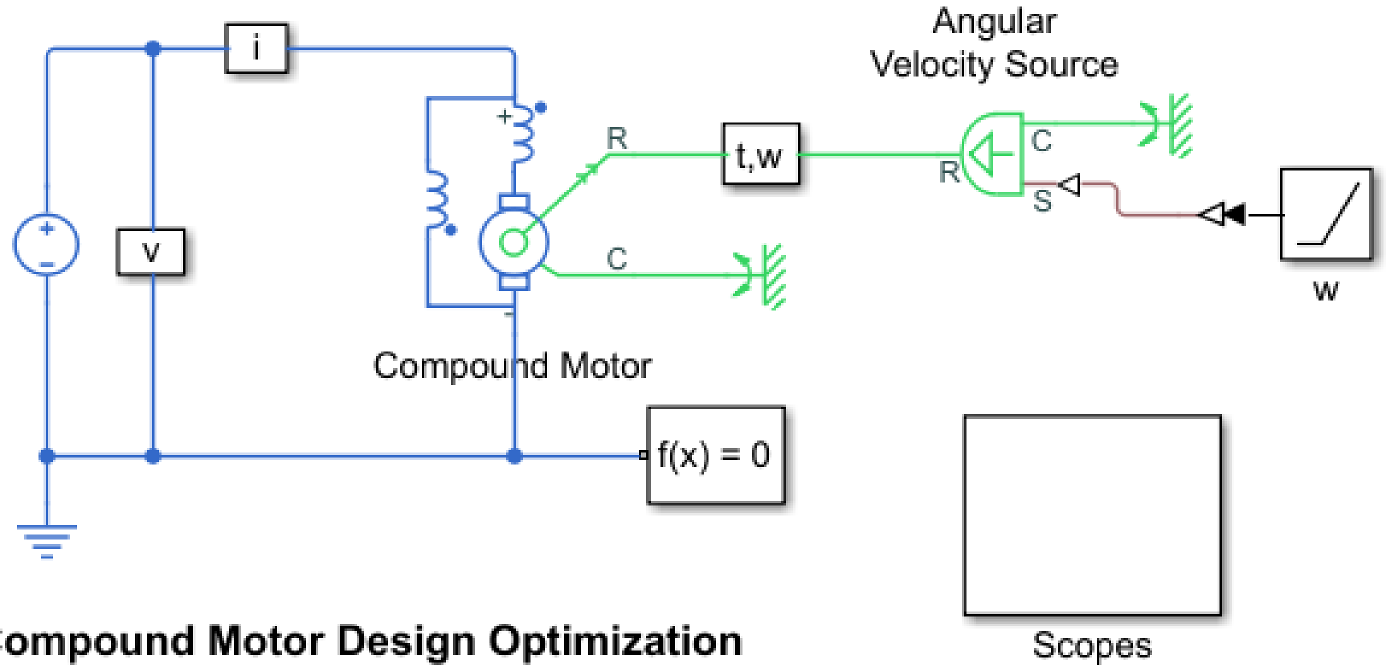
Specilised power electronics





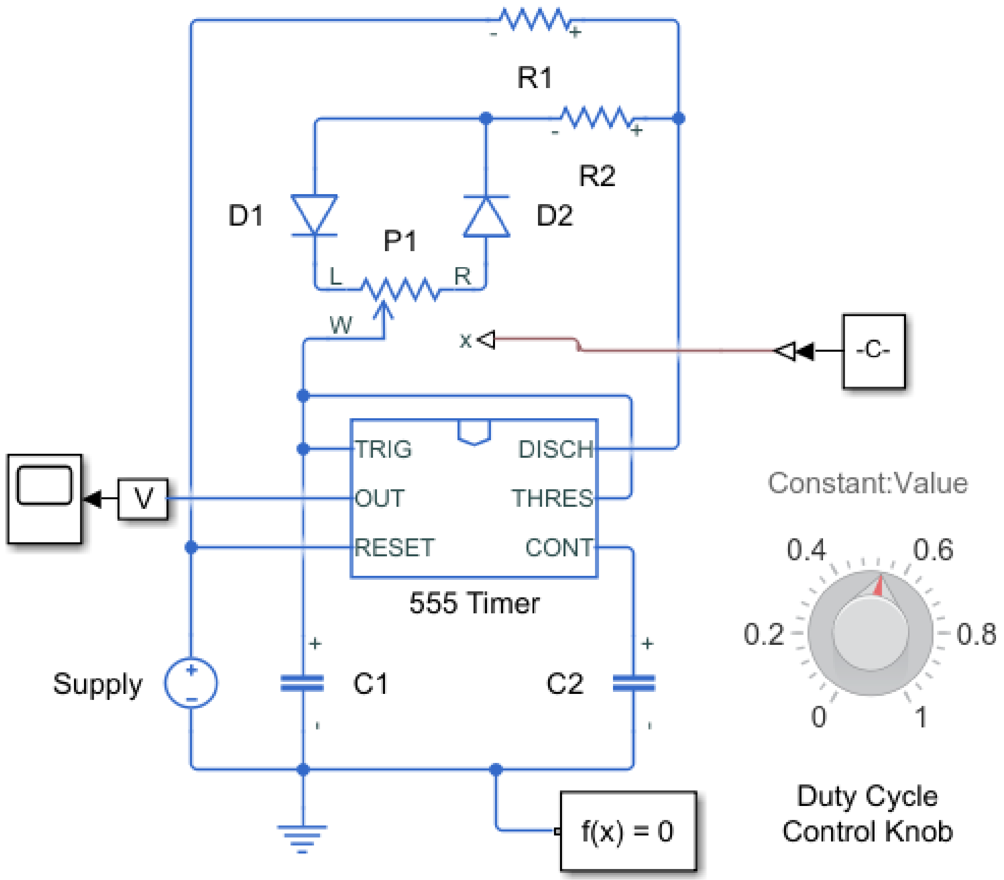
Strain Gauge and Wheatstone Bridge

1. Check tolerance effects on signal (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example



Compound Motor Design Optimization

1. Modify model parameters
2. Optimize motor torque-speed curve (see code)
3. Plot torque, power and efficiency curves (see code)
3. Explore simulation results using sscexplore
4. Learn more about this example

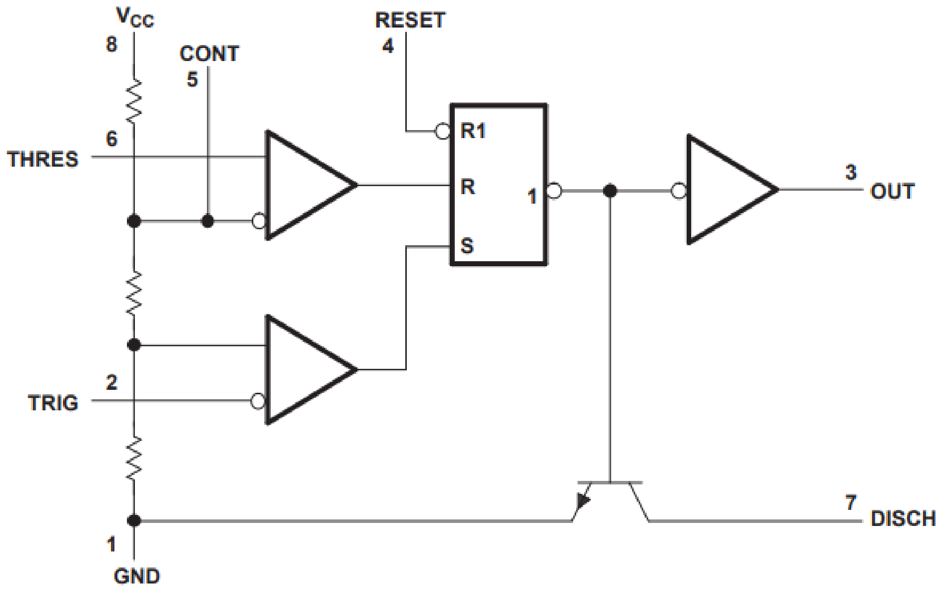


PWM Circuit Using 555 Timer

1. Explore simulation results using sscexplore
2. Learn more about this example

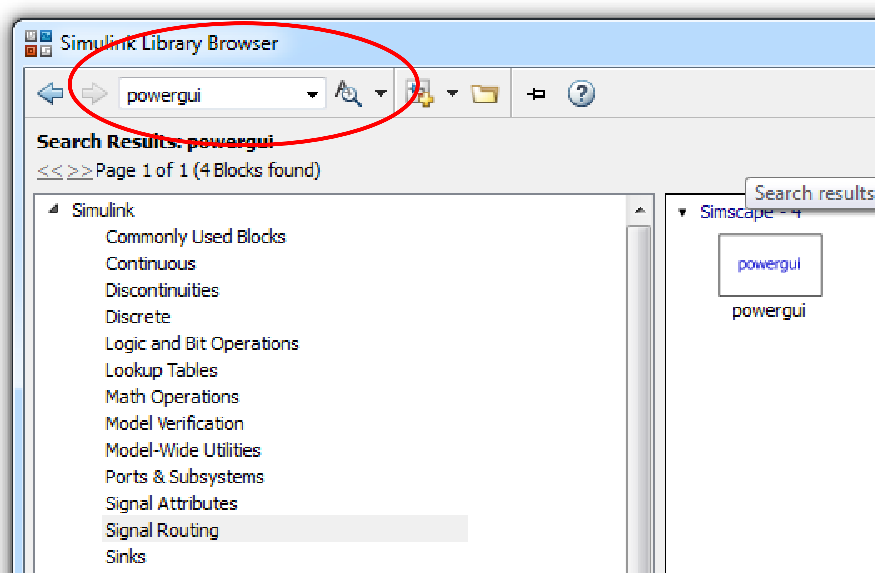
PIN			I/O	DESCRIPTION
NAME	D, P, PS, PW, JG	FK		
	NO.			
CONT	5	12	I/O	Controls comparator thresholds, Outputs 2/3 VCC, allows bypass capacitor connection
DISCH	7	17	O	Open collector output to discharge timing capacitor
GND	1	2	–	Ground
NC		1, 3, 4, 6, 8, 9, 11, 13, 14, 16, 18, 19	–	No internal connection
OUT	3	7	O	High current timer output signal
RESET	4	10	I	Active low reset input forces output and discharge low.
THRES	6	15	I	End of timing input. THRES > CONT sets output low and discharge low
TRIG	2	5	I	Start of timing input. TRIG < ½ CONT sets output high and discharge open
V_{CC}	8	20	–	Input supply voltage, 4.5 V to 16 V. (SE555 maximum is 18 V)

8.2 Functional Block Diagram



Important when working with PE !

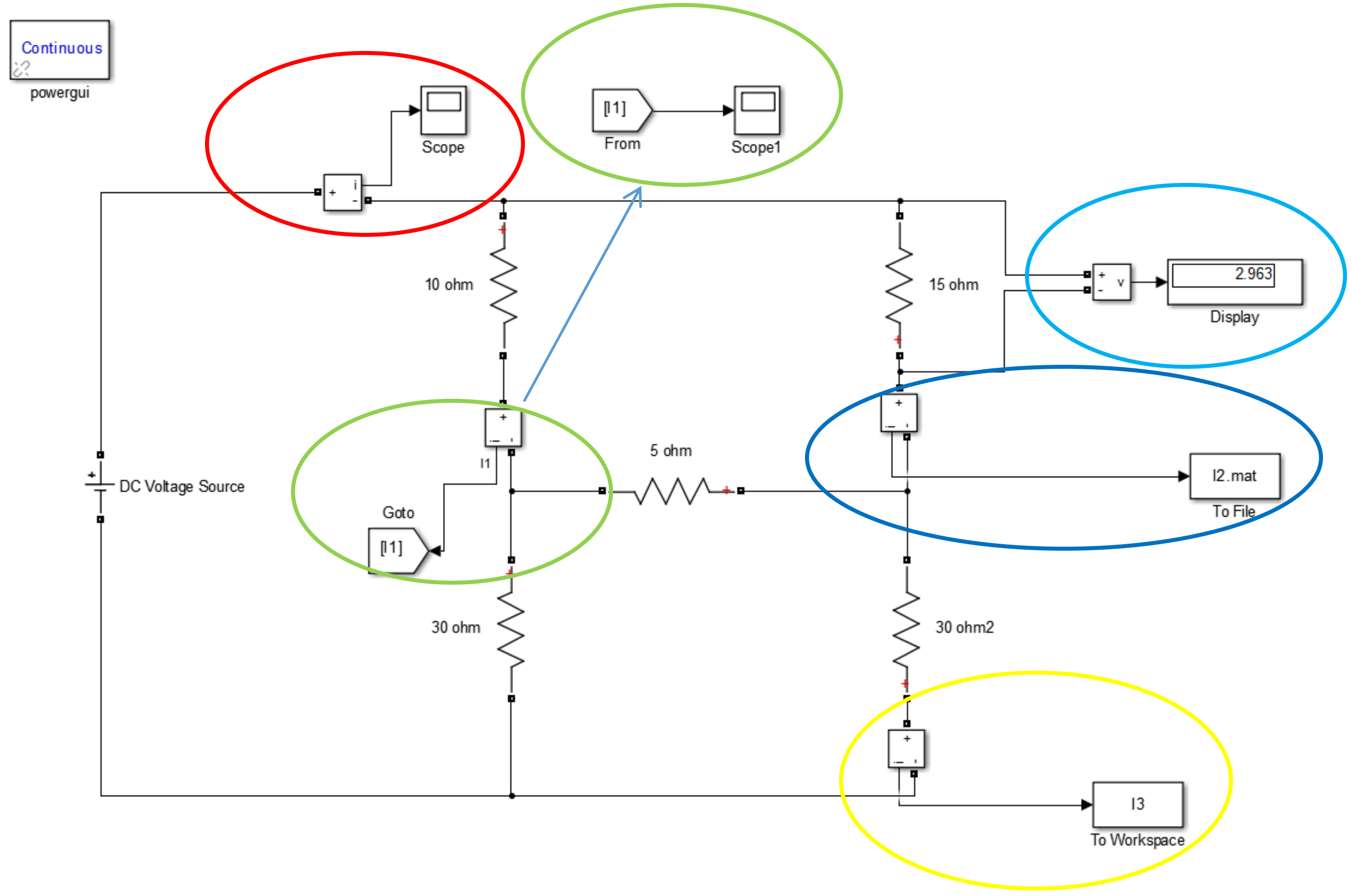
- You need a ***powergui*** block



The simplest way of finding it is by searching for it in the library.
We will discuss more about it later.

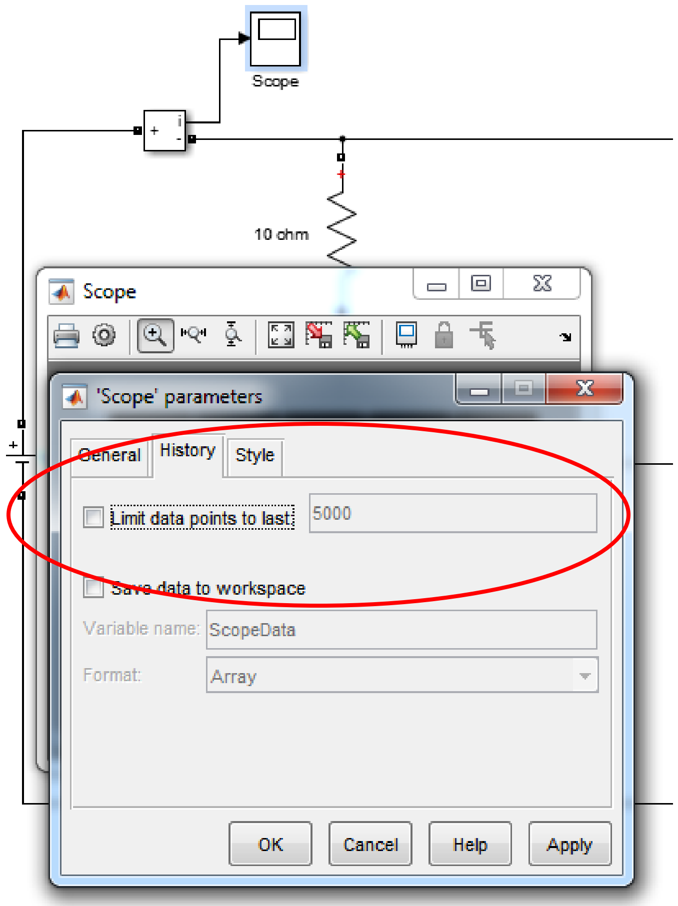
Measurements

- There are multiple ways in which we can measure different values in our models.

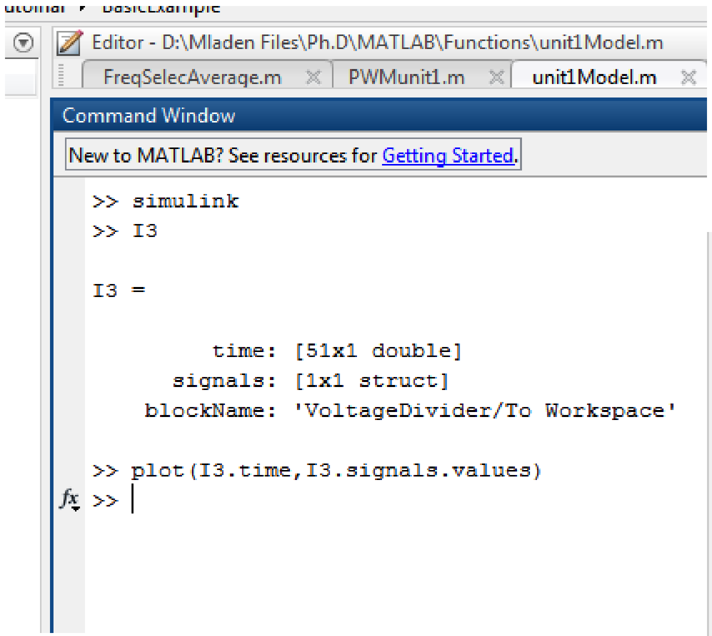


Important for Scopes

- Remove the **data limit** as soon as you copy the first scope to your workspace

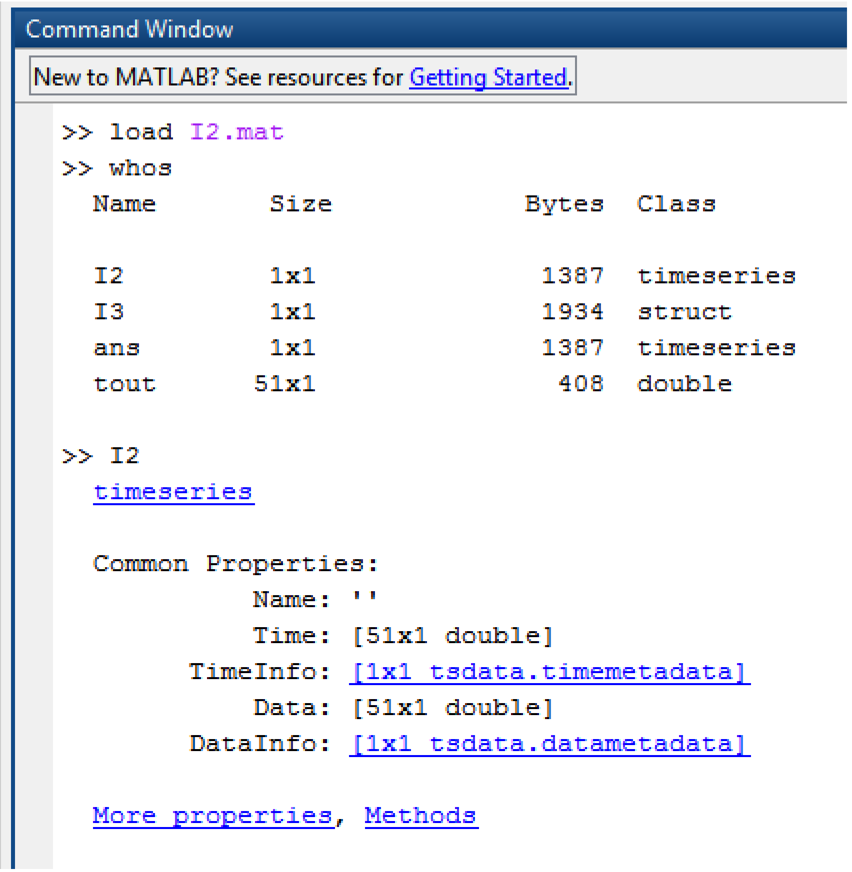


Extracting Data from the Simulation

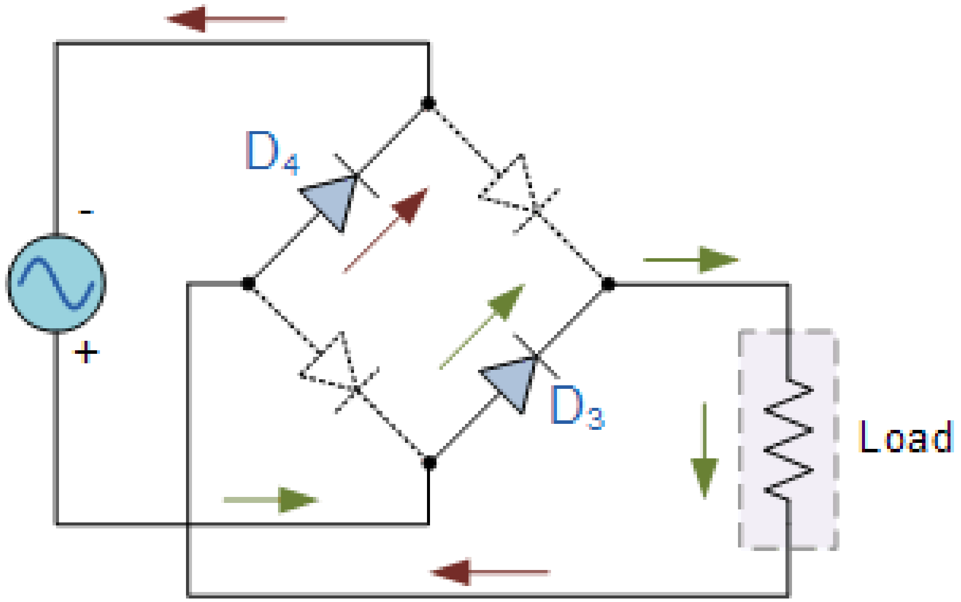


To workspace
In the block, specify to save the format as **structure with time**

From file:
In the block, specify to save the format as **array**



Using PE Devices- Uncontrolled Rectifier



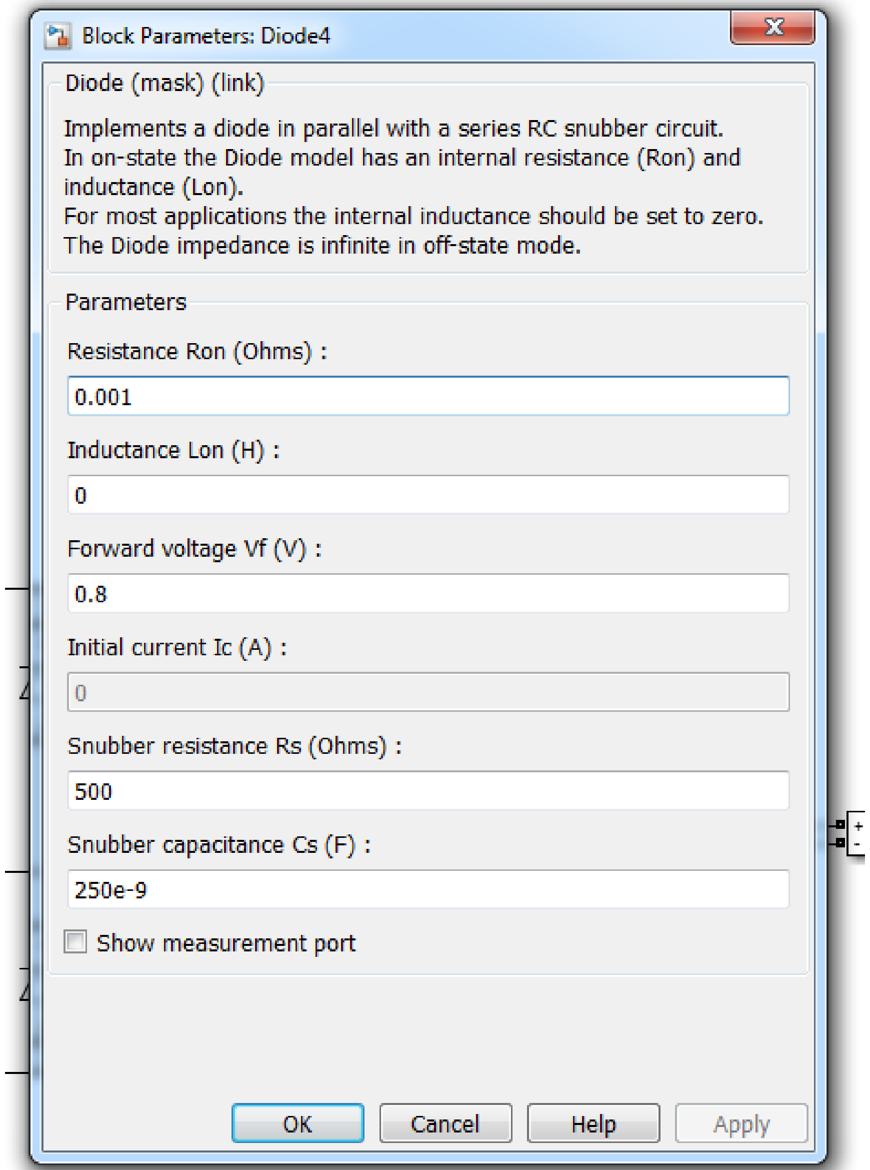
Where are the diodes *hidden* in SimPowerSystem ?

- Start with a diode bridge, AC voltage source and a 100 ohm resistor.
- Measure all relevant values.

Diode Parameters

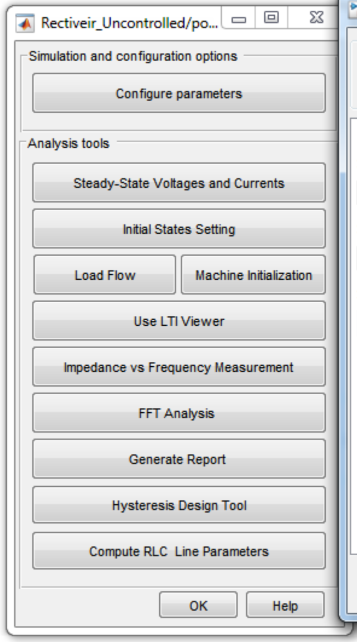
Keep the values od the diode parameters as-is.

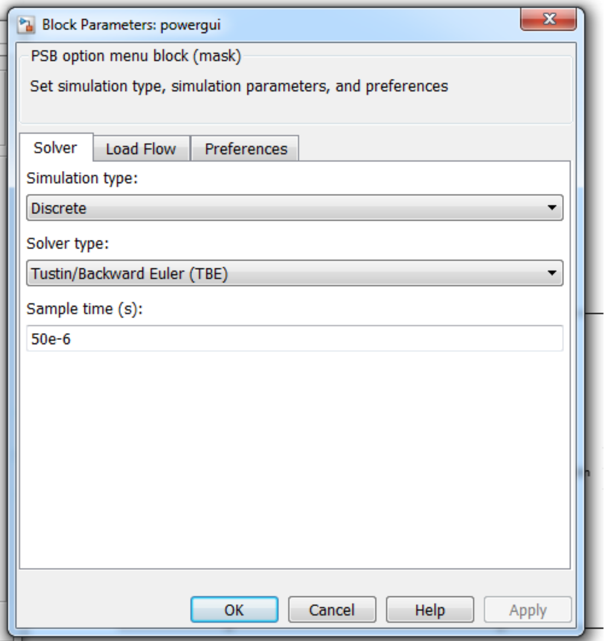
- Run the simulation, what do you notice?
- Add a capacitor in parallel with the resistance- now what happens ?
  - Simulation time ?
  - Initial current ?

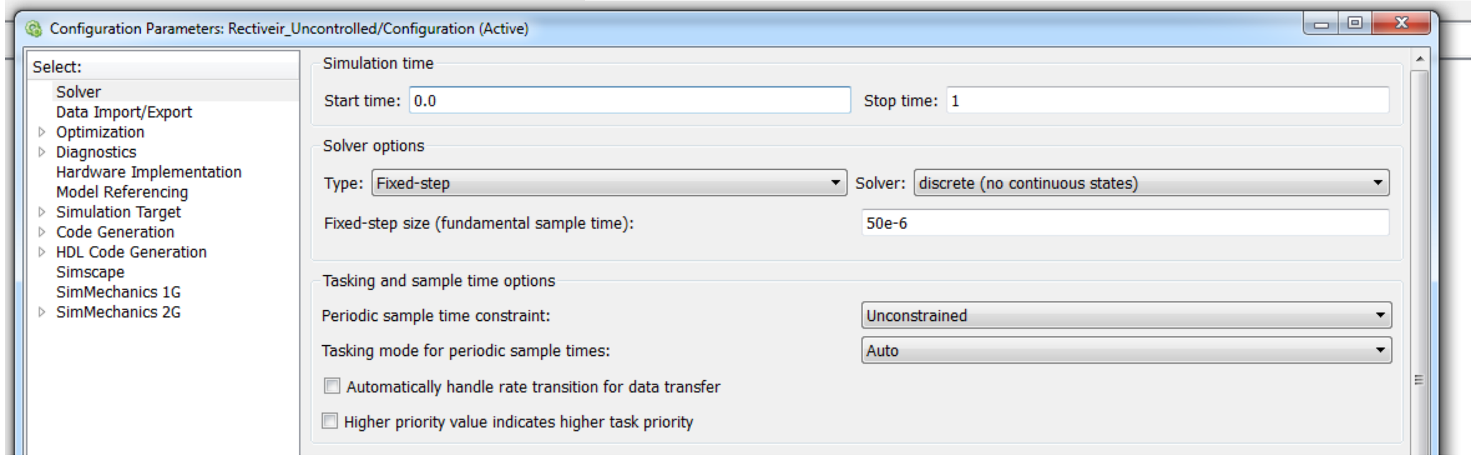


Time to Save Time and Switch the Switches in Digital

- Or in this case the diodes, and the simulation, to be more precise, in fixed-step (discrete).
- Modify the powergui and configuration parameters as such:

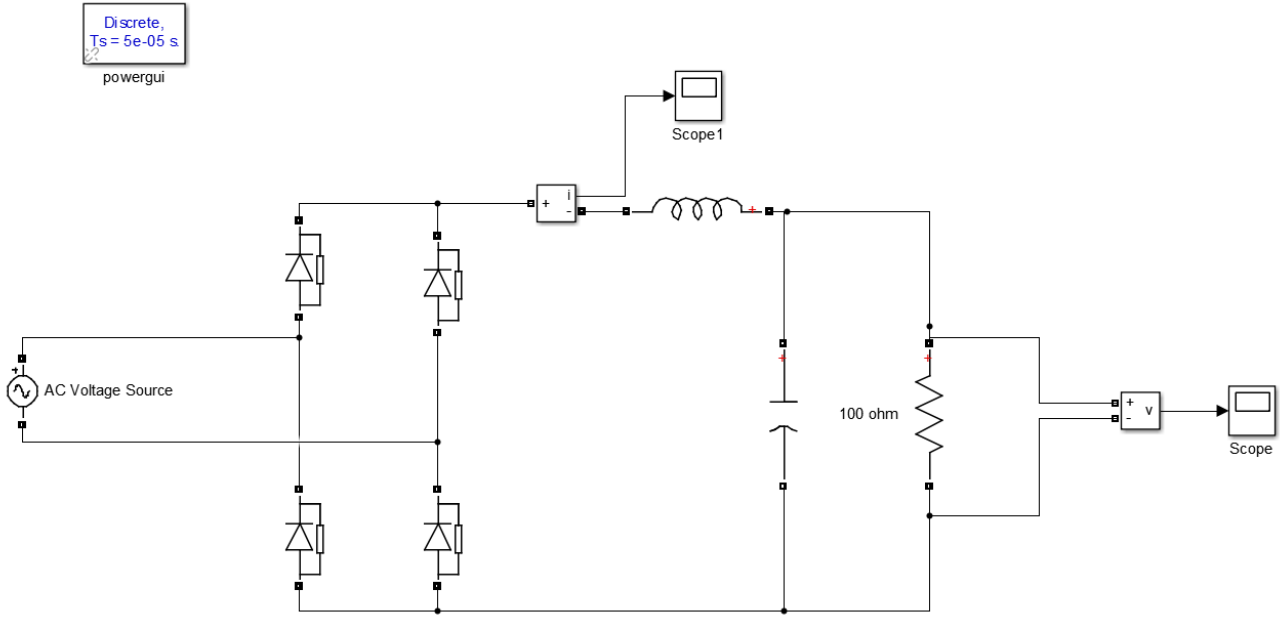






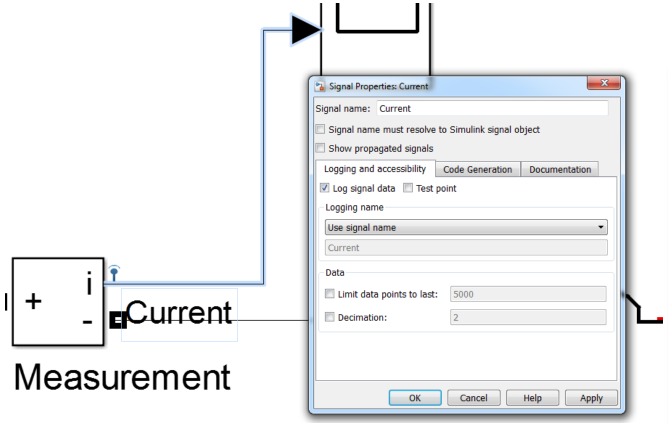
Finalising the Rectifier

- Add an inductor in series with the load and capacitor.
  - What happens to the initial current ?

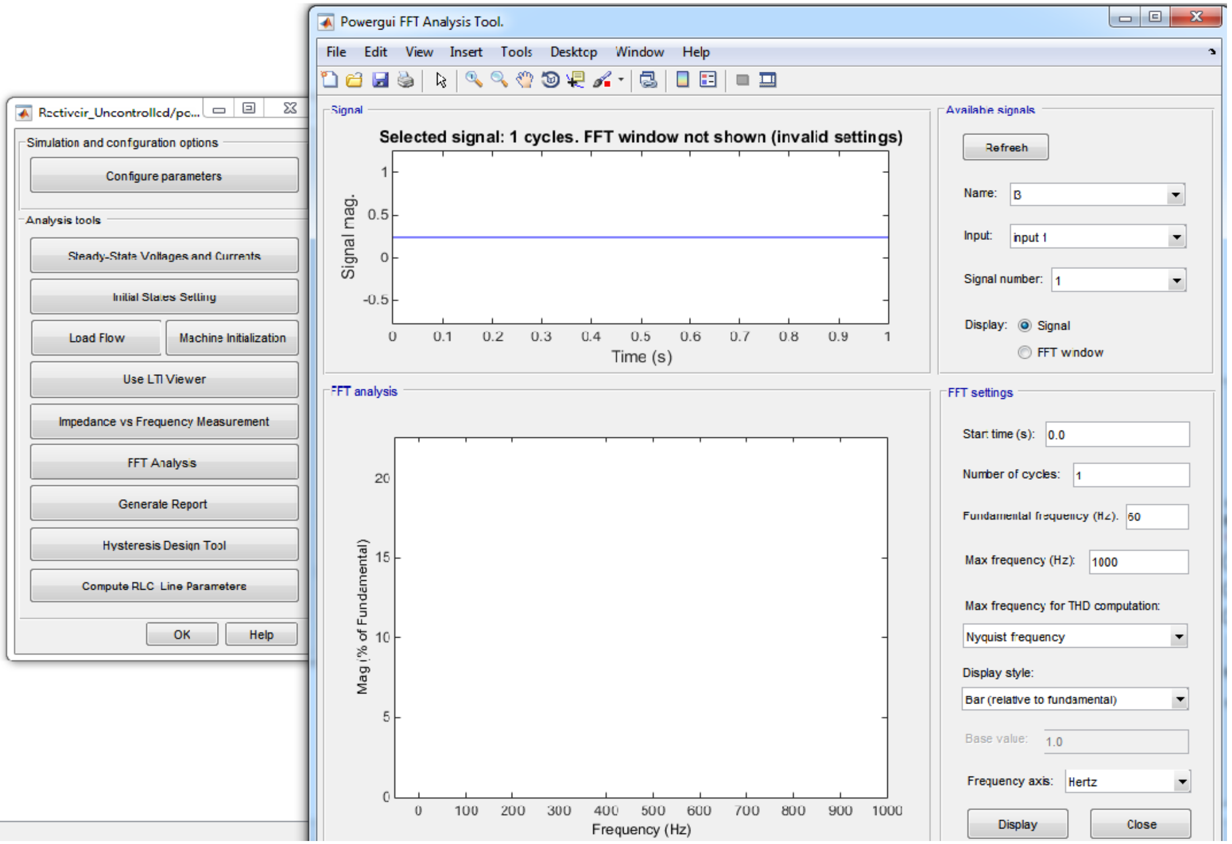


Analysing the Harmonics

- In the case of PE converters or various electrical systems, there is a great focus on evaluating the THD and harmonic spectre of electrical parameters.
- To achieve this in Simulink, we specify the values variables of interest and analyse them via the *powergui block.*

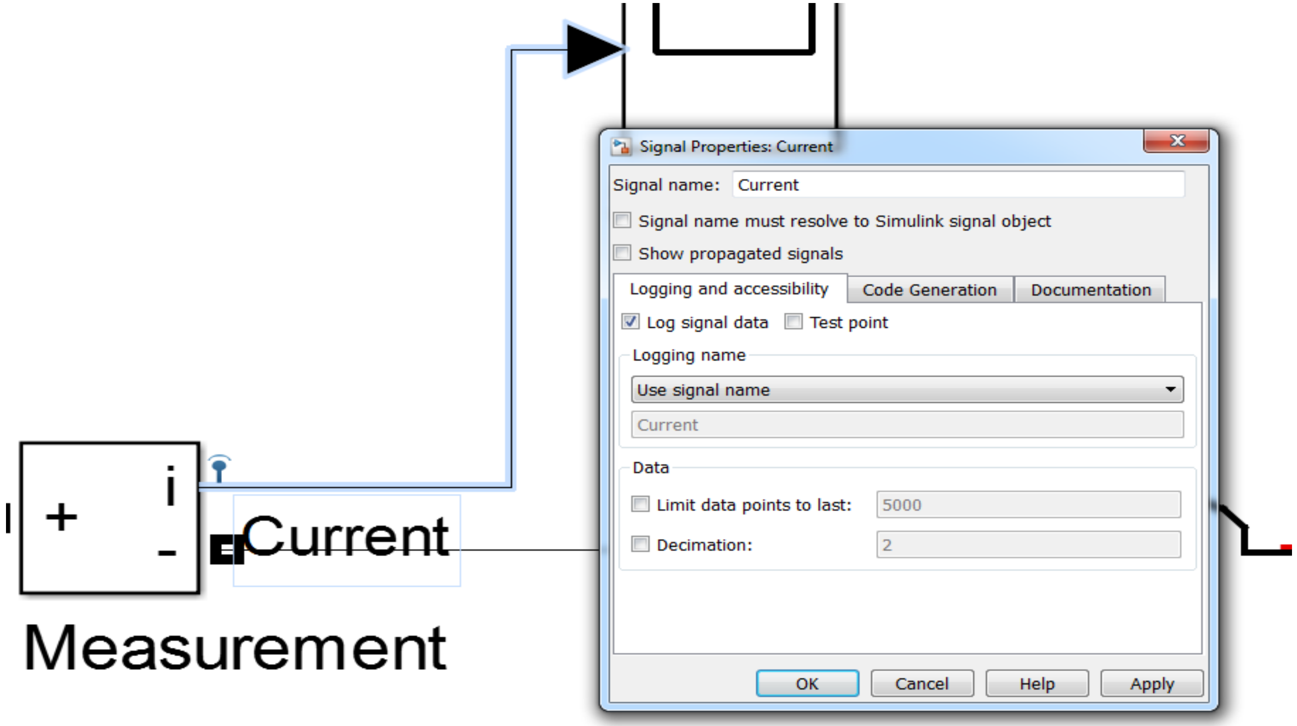


To assign a signal that you wish to analyse, right-click on the line and chose properties. Then, set the options as such:



Analysing the Harmonics

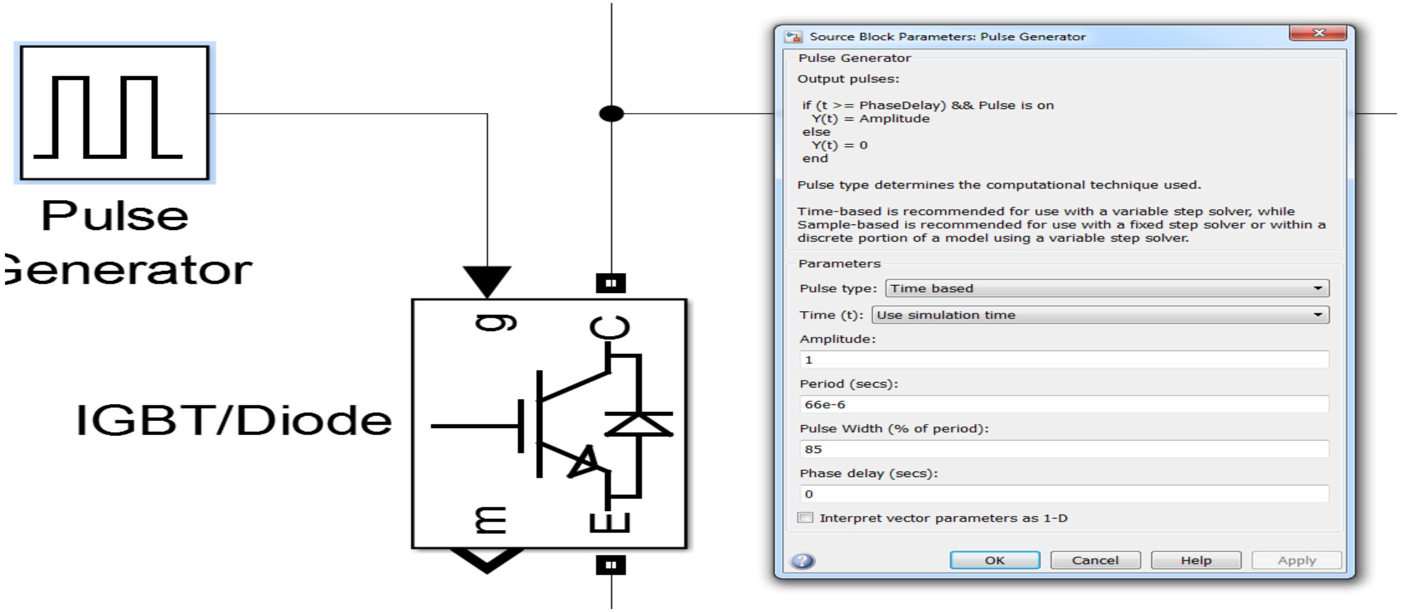
To assign a signal that you wish to analyse, right-click on the line and chose properties. Then, set the options as such:



Run the simulation, experiment with different parameter values and se how the THD changes.

Time to *Switch* to a More Controllable Device and *Boost* our Understanding

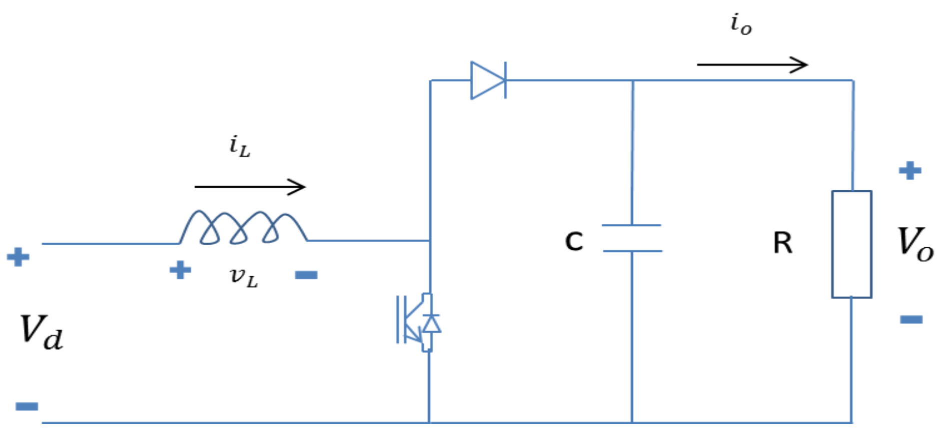
- In order to function properly, PE switches require signals that determine there state (on/off) and modulates the output signal.
- In Simulink, this signal represents a logical (1 or 0) value that we bring to the *gate* terminal of the block.
- The most simple form of modulation is a fixed duty-cycle periodic signal that can easily be achieved via the *Pulse Generator* block



Make sure that your simulation period is in sync with your switching period

Your task:

- Construct a Boost converter with one of the two main types of switches and their respective parameters:



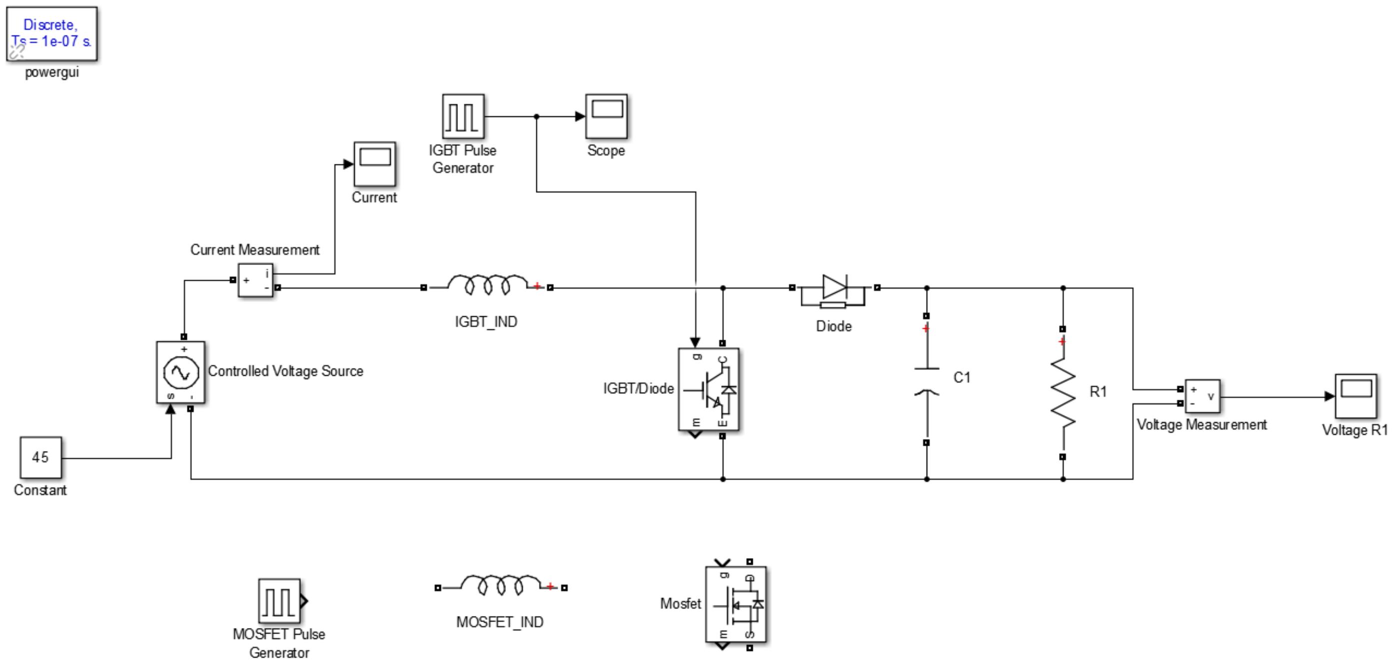
MOSFET version

D	0,7
f	80kHz
Vd	45V
Vo	150V
R	400 ohm
L	0,787mH
C	3,3uF
Δi	20%

IGBT version

D	0,85
f	15kHz
Vd	45V
Vo	300V
R	400 ohm
L	1,02mH
C	3,3uF
Δi	20%

Results ?

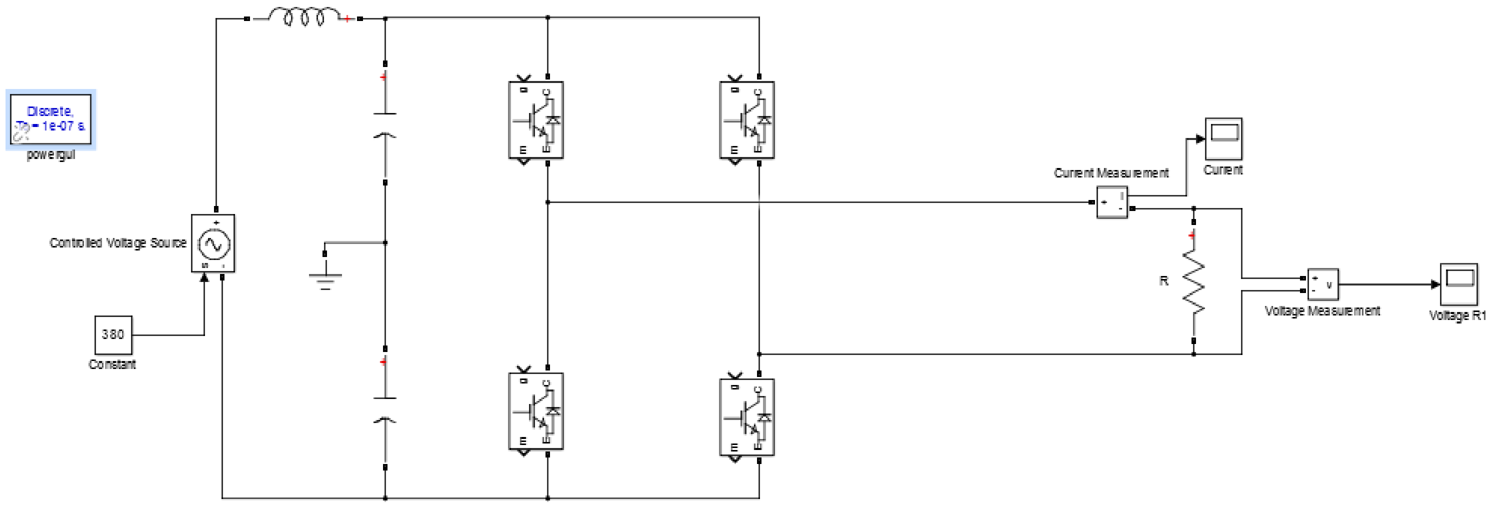


If you are done, try to make a Buck or Buck-Boost converter using these blocks.

Single Phase Inverter

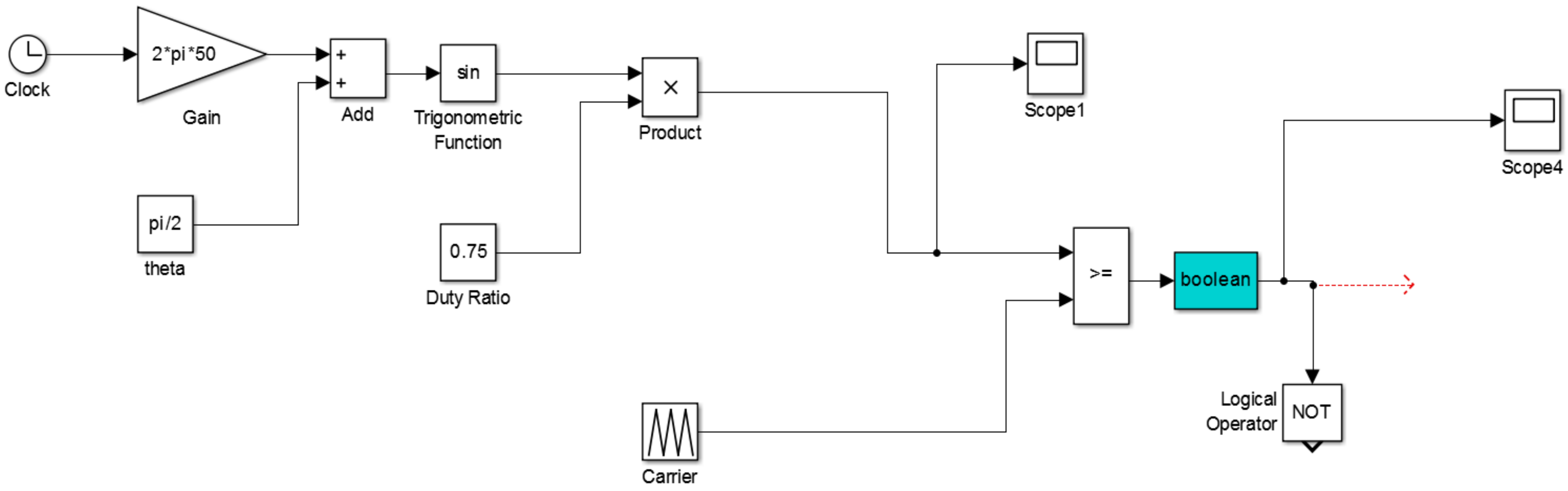
- In order to operate a full-bridge converter we need to implement a variable duty-cycle PWM.
- To generate this type of PWM, we need to construct a carrier and a modulation signal and compare them.

For now, construct a FB inverter using IGBTs blocks, a controlled voltage source on the DC bus and a 100 ohm resistor on the output.



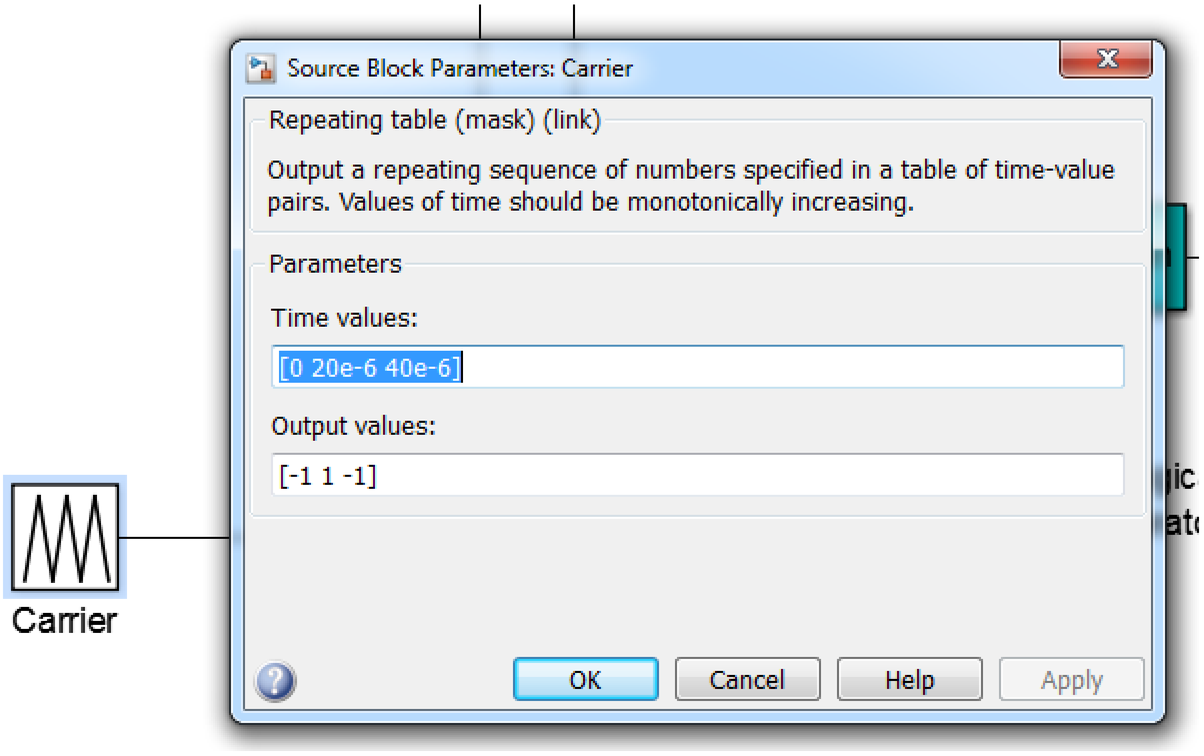
PWM Signal

- The PWM generator, as previously stated need to provide the neccesary type of signal to the gates of the switches.
- One way to construct it using block is as followed:



PWM Signal

- The carrier signal in this case is a triangular periodic signal with the following parameters:

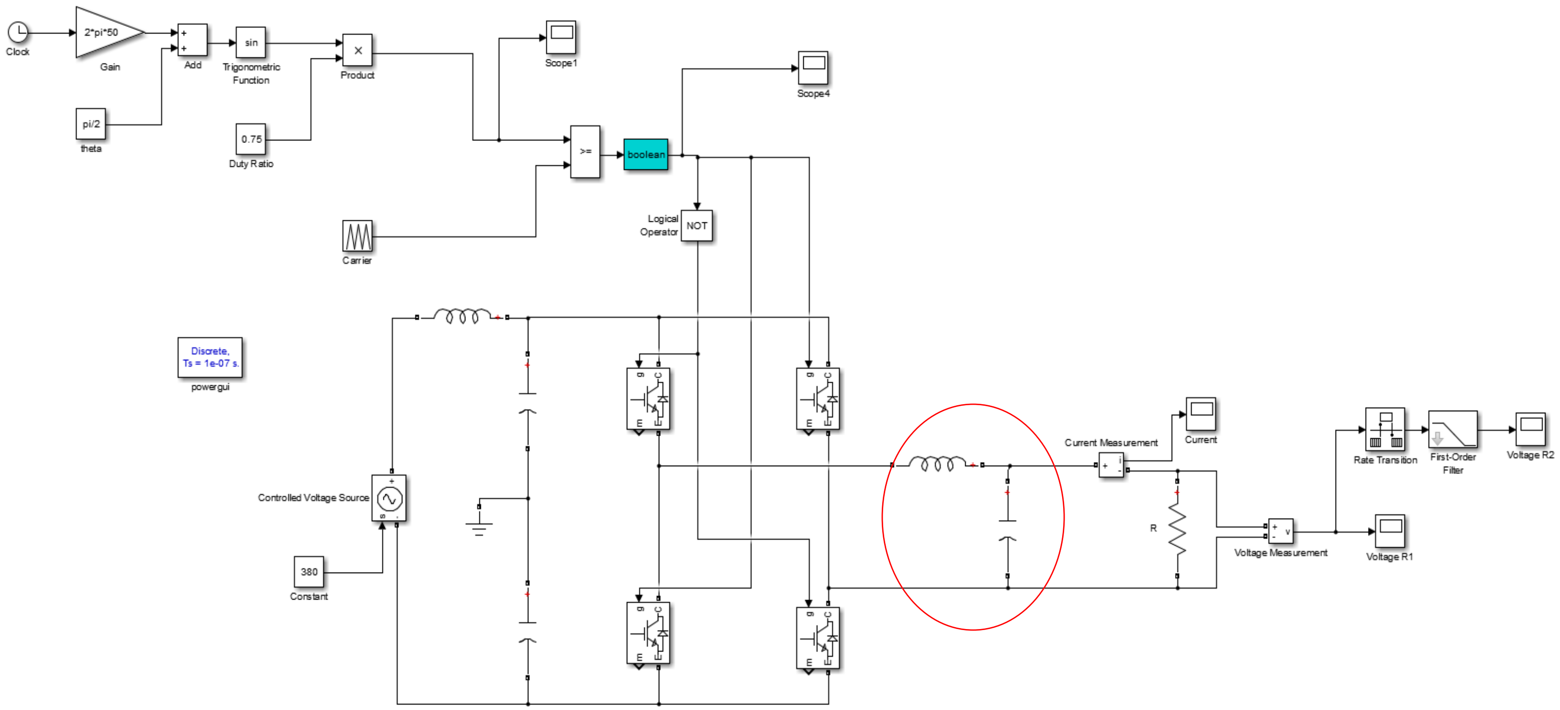


The modulation signal can also be changed by adjusting the parameters in the blocks.
Connect these signals to the respective gates and run the simulation.

Harmonic Filtering

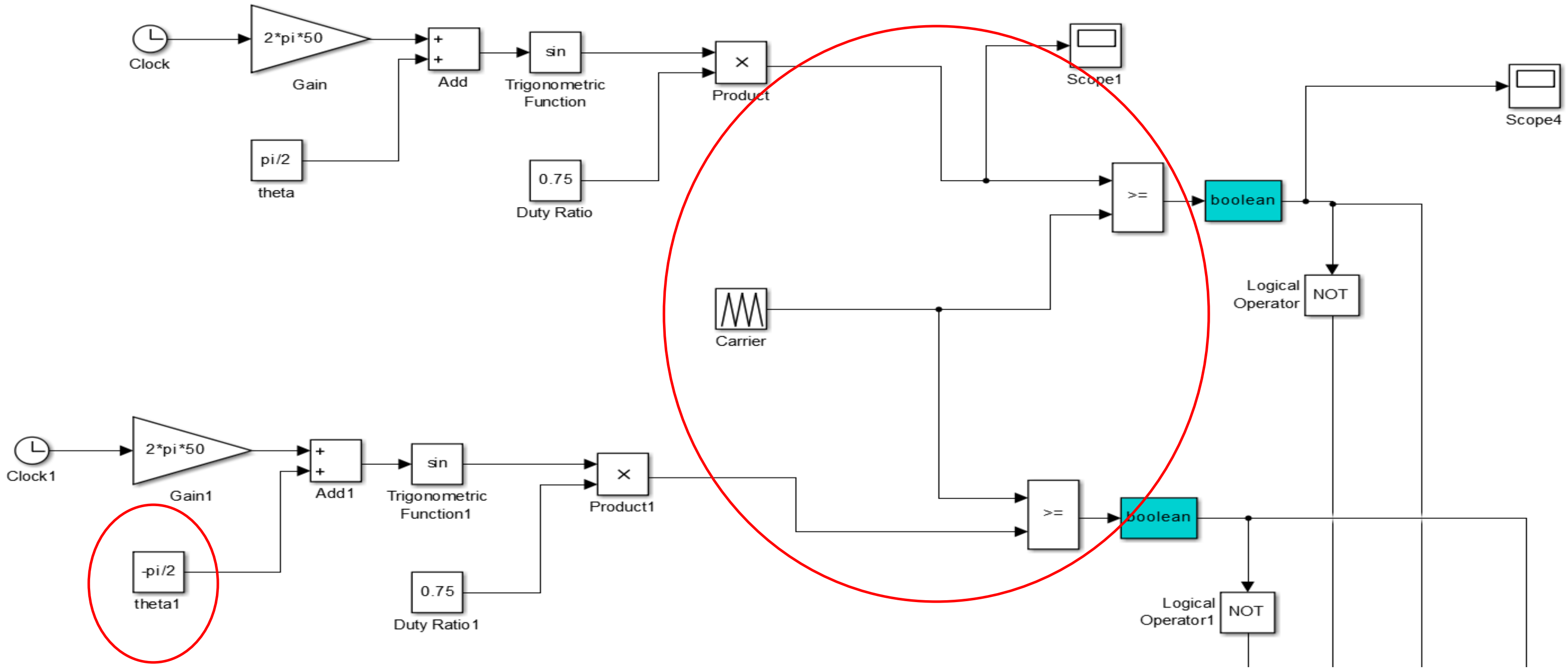
- How does the voltage/ current of the output look ?
- Measure the THD of these signals like we did in the previous examples.

- Add an output filter using a inductor and capacitor choose there values in order to mitigate the THD.

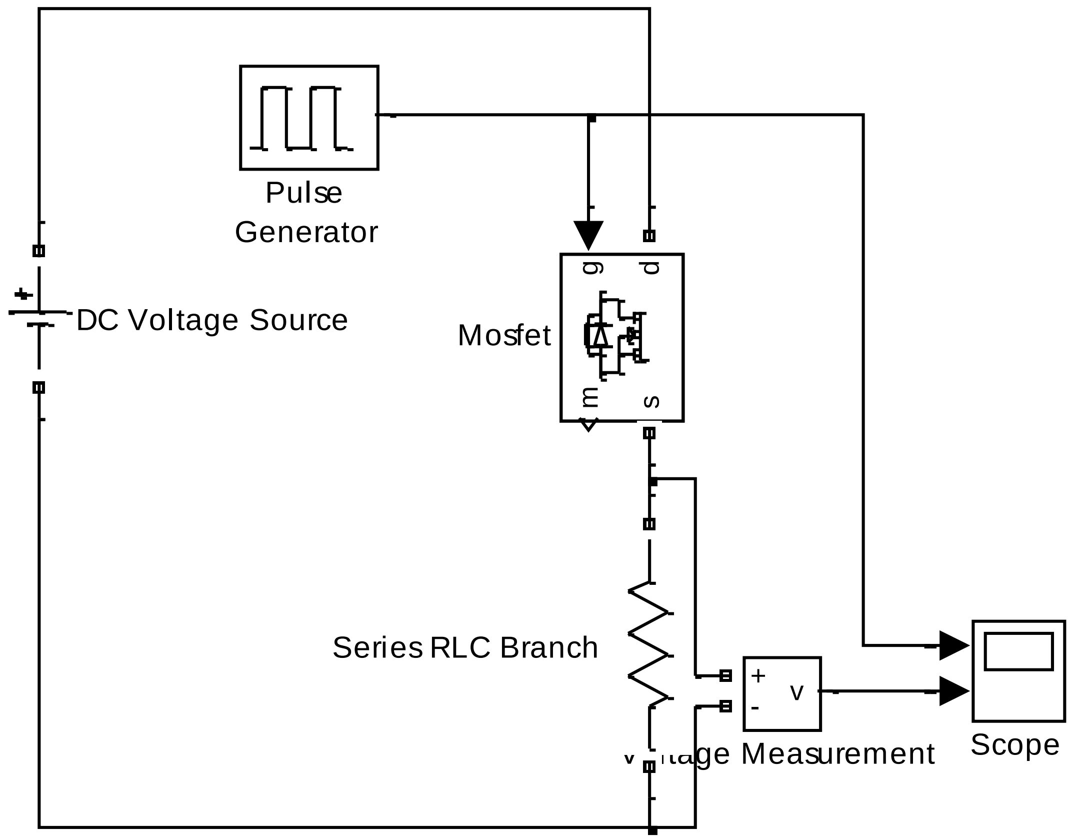


Unipolar Modulation

- Modify the model to operate with a unipolar PWM.
- What is the difference in THD in comparison with bipolar modulation ?



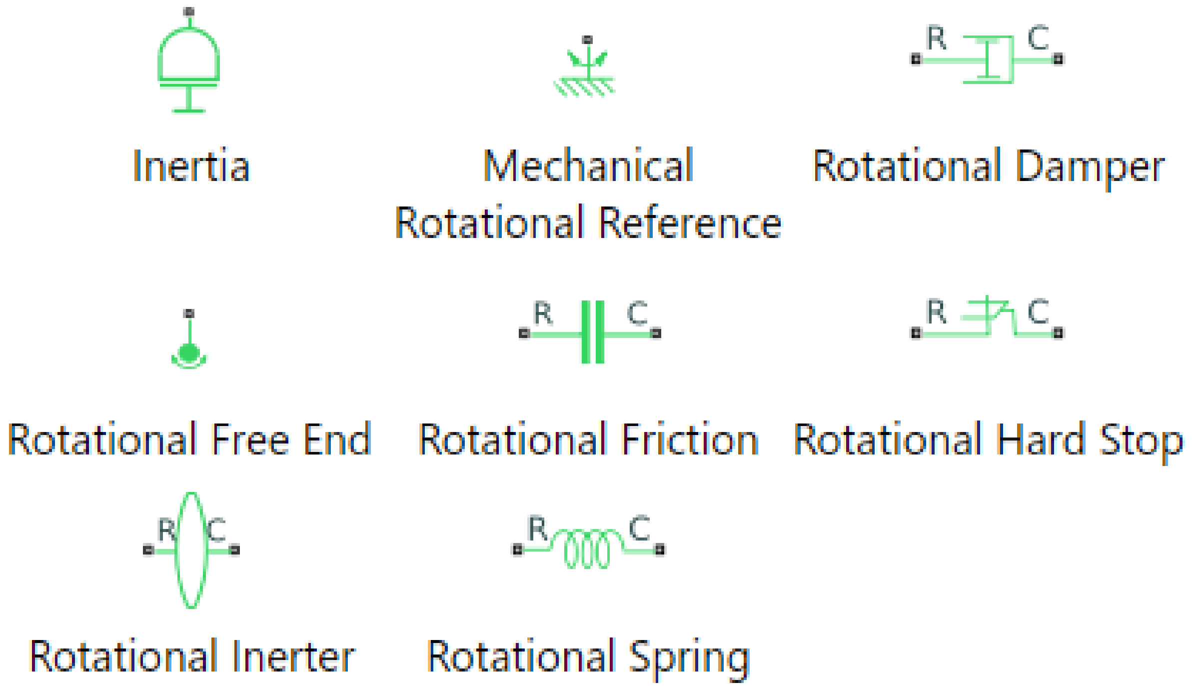
The odd carrier and associated sideband harmonic are eliminated

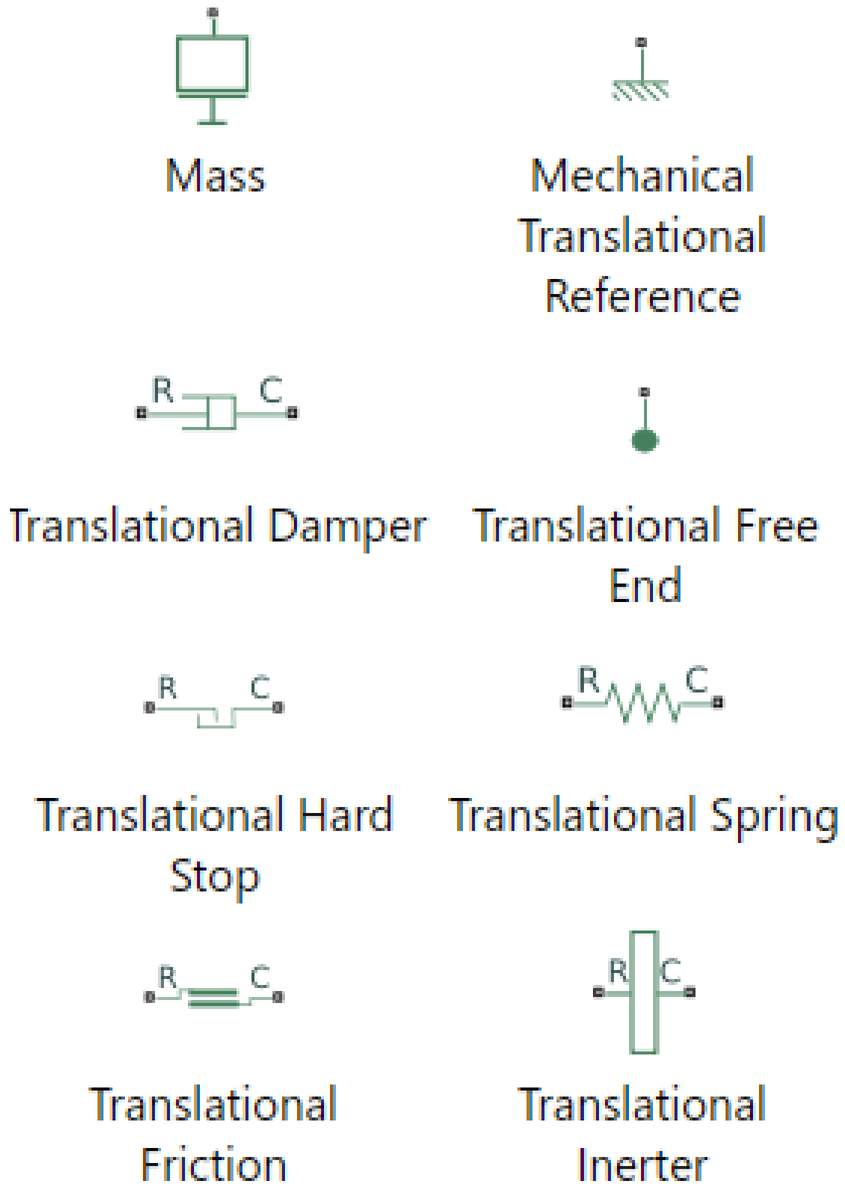


Pulse
Generator
DC Voltage Source
Mosfet
g
d
m
s
Series RLC Branch
v
Voltage Measurement
Scope

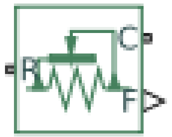
Simscape mechanical

Rotational and translational elements

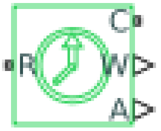




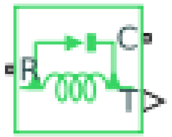
Sensors



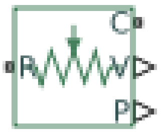
Ideal Force Sensor



Ideal Rotational Motion Sensor

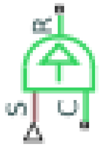


Ideal Torque Sensor

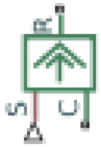


Ideal Translational Motion Sensor

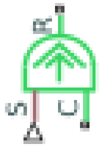
Sources



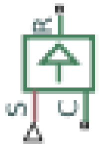
Ideal Angular Velocity Source



Ideal Force Source

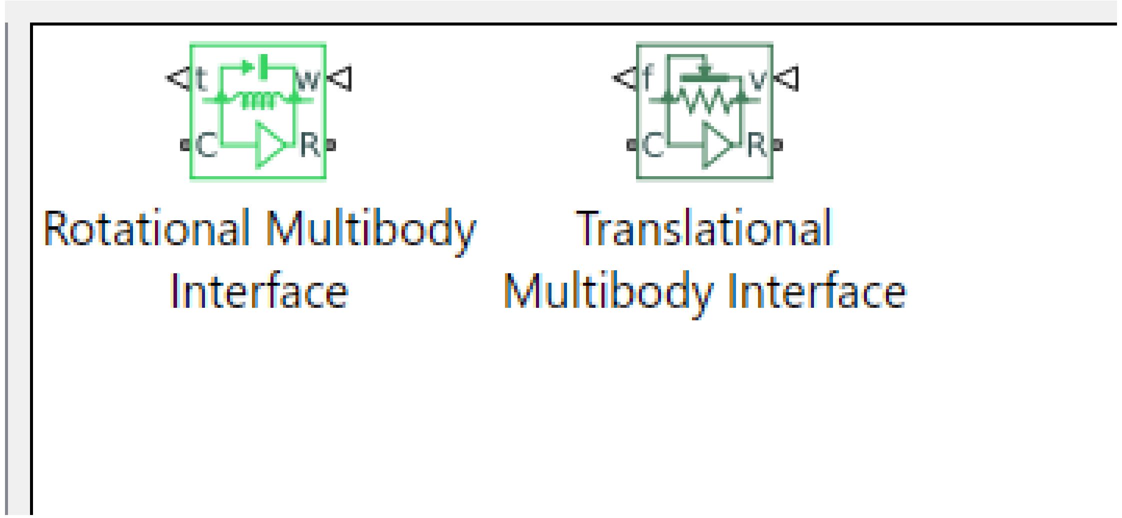


Ideal Torque Source

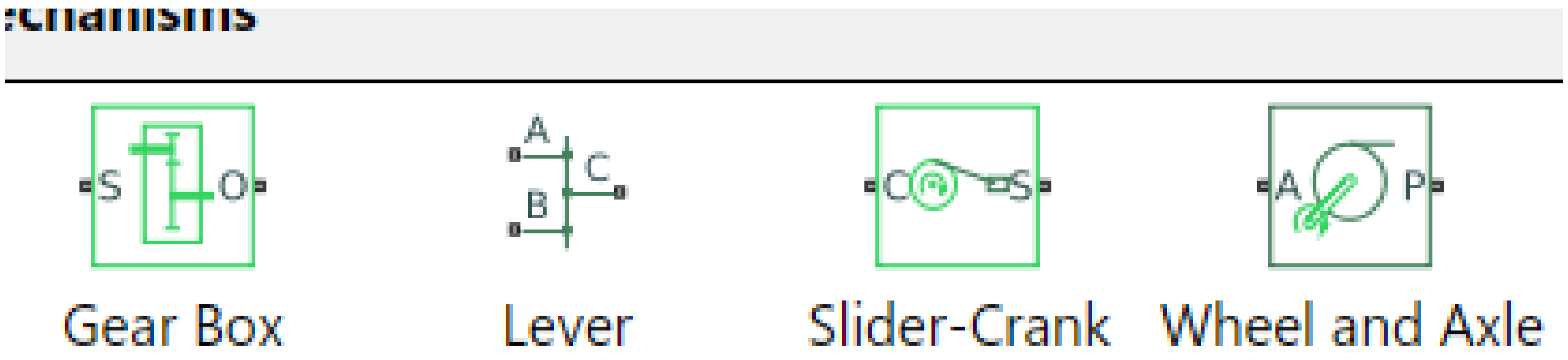


Ideal Translational Velocity Source

Multi-body interface



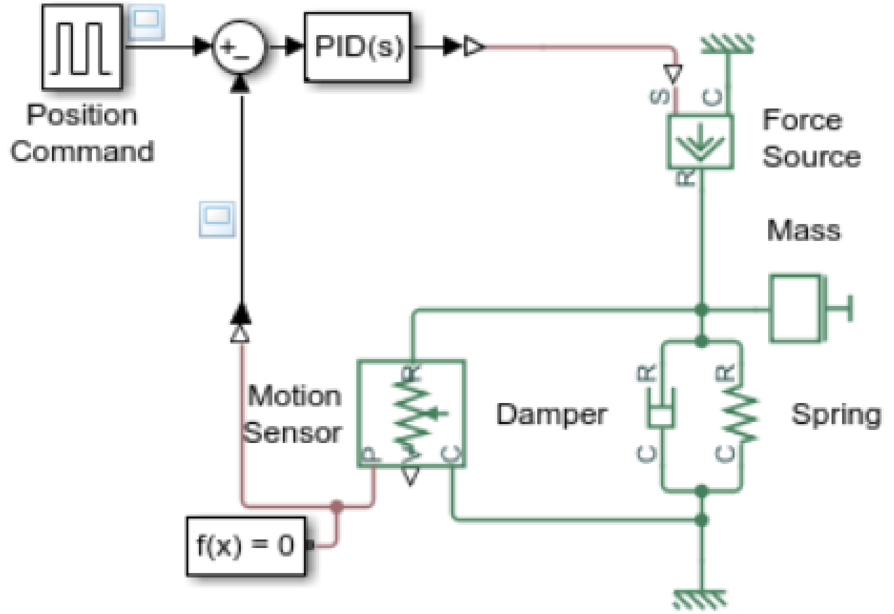
Mechanisms



Mass-Spring-Damper with Controller

This example shows a controlled mass-spring-damper. A controller adjusts the force on the mass to have its position track a command signal. The initial velocity for the mass is 10 meters per second. The controller adjusts the force applied by the Force Source to track the step changes to the input signal.

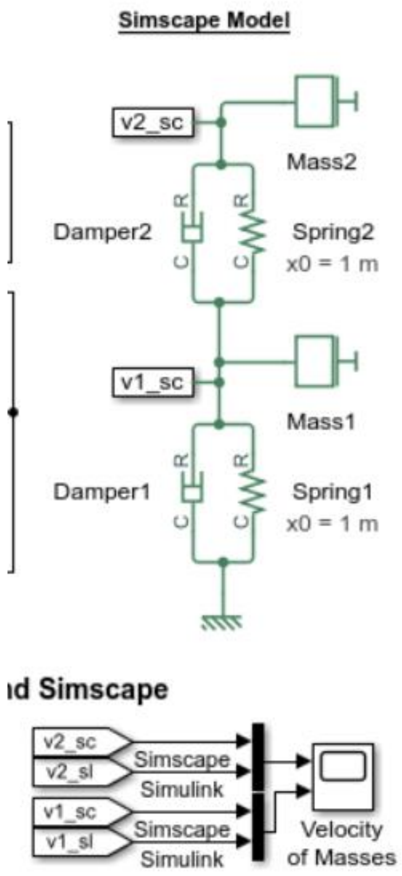
Model



Mass-Spring-Damper with Controller

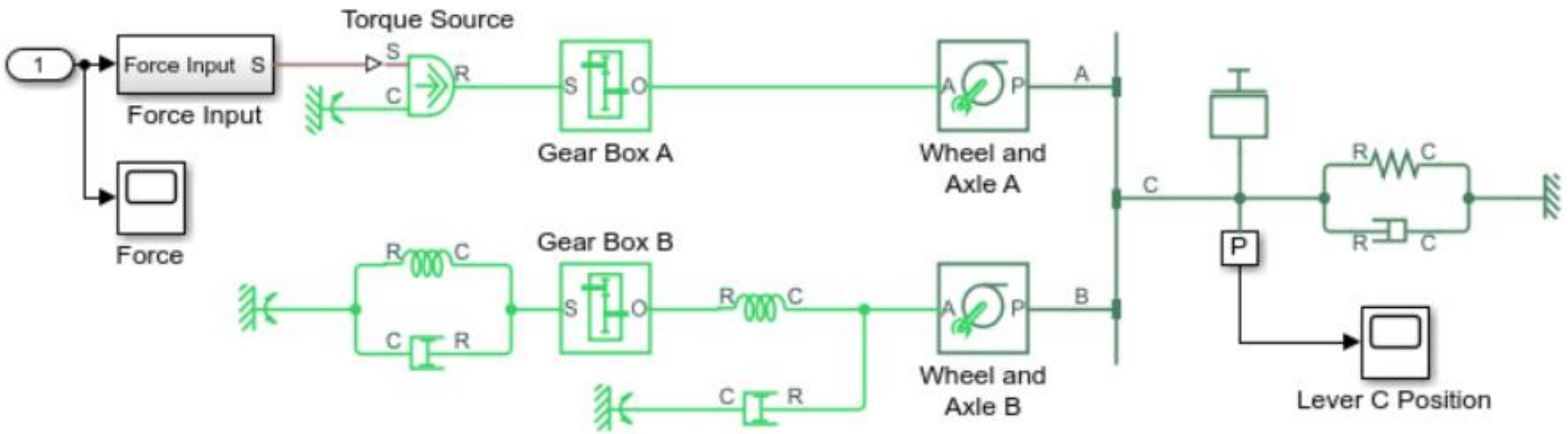
1. Plot forces in system and mass position (see code)
2. Explore simulation results using sscexplore
3. Learn more about this example

Double mass


Simscape Model
v2_sc
Mass2
Damper2
Spring2
x0 = 1 m
v1_sc
Mass1
Damper1
Spring1
x0 = 1 m
d Simscape
v2_sc
v2_sl
Simscape
Simulink
v1_sc
v1_sl
Simscape
Simulink
Velocity
of Masses

Simple Mechanical System

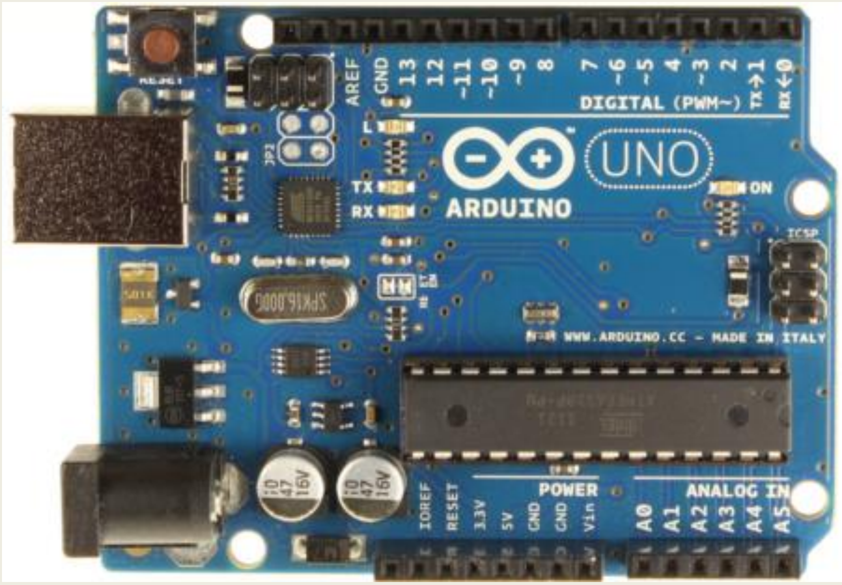
This example shows a model of a system that connects rotational and translational motion. A summing lever drives a load consisting of a mass, viscous friction, and a spring connected to its joint C. Joint B is suspended on two rotational springs connected to reference point through a wheel and axle and a gear box. Joint A is connected to a torque source through a gear box and a wheel and axle mechanism.



Simple Mechanical System

1. Explore simulation results using sscexplore
2. Learn more about this example

Arduino: This is our Brain in Phys120B



Arduino Uno

Arduino Nano

- Packaged Microcontroller (ATMega 328)
  - lots of varieties; we'll primarily use Uno and Nano
  - USB interface; breakout to pins for easy connections
  - Cross-platform, Java-based IDE, C-based language
  - Provides higher-level interface to guts of device

Every Arduino "Sketch"

- Each "sketch" (code) has these common elements

```
// variable declarations, like
const int LED 13;

void setup()
{
  // configuration of pins, etc.
}

void loop()
{
  // what the program does, in a continuous loop
}
```

- Other subroutines can be added, and the internals can get pretty big/complex

Rudimentary C Syntax

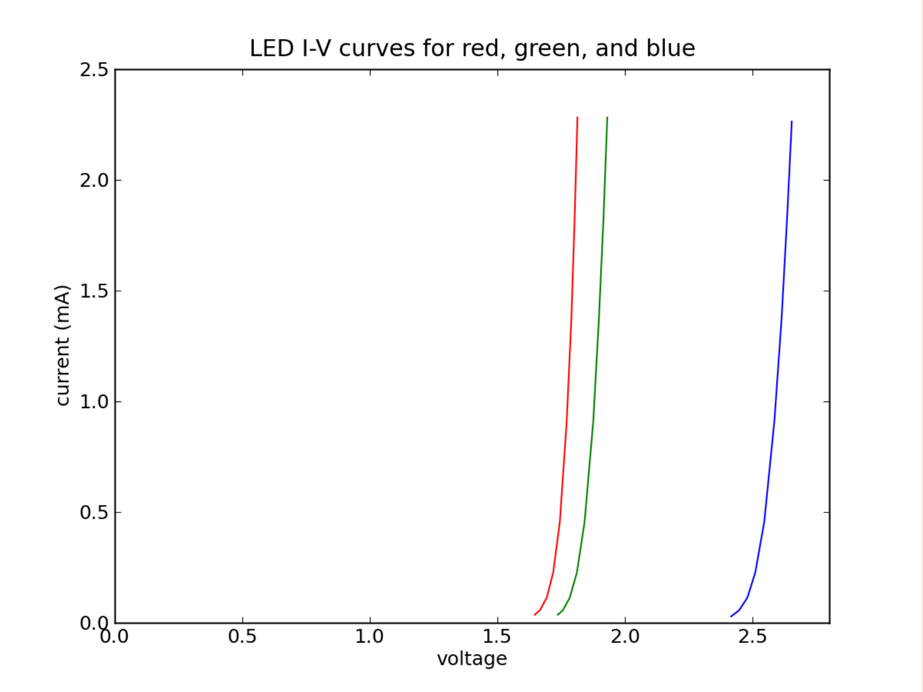
- Things to immediately know
  - anything after // on a line is ignored as a comment
  - braces { } encapsulate blocks
  - semicolons ; must appear after every command
    - exceptions are conditionals, loop invocations, subroutine titles, precompiler things like `#include`, `#define`, and a few others
  - every variable used in the program needs to be declared
    - common options are `int`, `float`, `char`, `long`, `unsigned long`, `void`
    - conventionally happens at the top of the program, or within subroutine if confined to { } block
  - Formatting (spaces, indentation) are irrelevant in C
    - but it is to your great benefit to adopt a rigid, readable format
    - much easier to read if indentation follows consistent rules

Example Arduino Code

```
// blink_LED. . . . . . . slow blink of LED on pin 13
const int LED = 13;      // LED connected to pin 13
                         // const: will not change in prog.

void setup()             // obligatory; void->returns nada
{
  pinMode(LED, OUTPUT); // pin 13 as output (Arduino cmd)
}

void loop()              // obligatory; returns nothing
{
  digitalWrite(LED, HIGH); // turn LED ON (Arduino cmd)
  delay(1000);             // wait 1000 ms (Arduino cmd)
  digitalWrite(LED, LOW);  // turn LED OFF
  delay(1000);             // wait another second
}
```

LED hookup

- The output of Arduino digital I/O pins will be either 0 or 5 volts
- An LED has a diode-like I-V curve
- Can't just put 5 V across
  - it'll blow, unless current is limited
- Put resistor in series, so ~2.5 V drop across each
  - 250 Ω would mean 10 mA
  - 10 mA is pretty bright



Comments on Code

- Good practice to start code with descriptive comment
  - include name of sketch so easy to relate print-out to source
- Most lines commented: also great practice
- Only one integer variable used, and does not vary
  - so can declare as `const`
- `pinMode()`, `digitalWrite()`, and `delay()` are Arduino commands
- `OUTPUT`, `HIGH`, `LOW` are Arduino-defined constants
  - just map to integers: 1, 1, 0, respectively
- Could have hard-coded `digitalWrite(13,1)`
  - but less human-readable than `digitalWrite(LED, HIGH)`
  - also makes harder to change output pins (have to hunt for each instance of 13 and replace, while maybe not every 13 should be)

Arduino-Specific Commands

- Command reference: http://arduino.cc/en/Reference/HomePage
  - Also abbr. version in Appendix C of *Getting Started* book (2nd ed.)
- In first week, we'll see:
  - `pinMode(`*pin*, [INPUT | OUTPUT]`)`
  - `digitalWrite(`*pin*, [LOW | HIGH]`)`
  - `digitalRead(`*pin*`)` → `int`
  - `analogWrite(`*pin*, [0…255]`)`
  - `analogRead(`*pin*`)` → `int` in range [0..1023]
  - `delay(`*integer milliseconds*`)`
  - `millis()` → `unsigned long` (ms elapsed since reset)

Arduino Serial Commands

- Also we'll use serial communications in week 1:
  - `Serial.begin(`*baud*`)`: in `setup`; 9600 is common choice
  - `Serial.print(`*string*`)`: *string* → `"example text "`
  - `Serial.print(`*data*`)`: prints *data* value (default encoding)
  - `Serial.print(`*data,encoding*`)`
    - *encoding* is DEC, HEX, OCT, BIN, BYTE for format
  - `Serial.println()`: just like `print`, but CR & LF (`\r\n`) appended
  - `Serial.available()` → `int` (how many bytes waiting)
  - `Serial.read()` → `char` (one byte of serial buffer)
  - `Serial.flush()`: empty out pending serial buffer

Types in C

- We are likely to deal with the following types

```
char c;              // single byte
int i;               // typical integer
unsigned long j;     // long positive integer
float x;             // floating point (single precision)
double y;            // double precision

c = 'A';
i = 356;
j = 230948935;
x = 3.1415927;
y = 3.14159265358979;
```

- Note that the variable `c='A'` is just an 8-bit value, which happens to be 65 in decimal, 0x41 in hex, 01000001
  - could say `c = 65`; or `c = 0x41`; with equivalent results
- Not much call for double precision in Arduino, but good to know about for other C endeavors

Changing Types (Casting)

- Don’t try to send float values to pins, and watch out when dividing integers for unexpected results
- Sometimes, we need to compute something as a floating point, then change it to an integer
  - `ival = (int) fval;`
  - `ival = int(fval);   // works in Arduino, anyhow`
- Beware of integer math:
  - 1/4 = 0; 8/9 = 0; 37/19 = 1
  - so sometimes want `fval = ((float) ival1)/ival2`
  - or `fval = float(ival1)/ival2 //okay in Arduino`

Conditionals

- The if statement is a workhorse of coding
  - `if (i < 2)`
  - `if (i <= 2)`
  - `if (i >= -1)`
  - `if (i == 4) // note difference between == and =`
  - `if (x == 1.0)`
  - `if (fabs(x) < 10.0)`
  - `if (i < 8 && i > -5)          // && = and`
  - `if (x > 10.0 || x < -10.0)   // || = or`
- Don't use assignment (=) in test clauses
  - Remember to double up ==, &&, ||
- Will execute single following command, or next { } block
  - wise to form { } block even if only one line, for readability/expansion
- Can combine with else statements for more complex behavior

If..else construction

- Snippet from code to switch LED ON/OFF by listening to a button

```
void loop()
{
  val = digitalRead(BUTTON);
  if (val == HIGH){
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }
}
```

- BUTTON and LED are simply constant integers defined at the program start
- Note the use of braces
  - exact placement/arrangement unnec., but be consistent

For loops

- Most common form of loop in C
  - also `while`, `do..while` loops
  - associated action encapsulated by braces

```
int k,count;

count = 0;
for (k=0; k < 10; k++)
{
  count += 1;
  count %= 4;
}
```

- `k` is iterated
  - assigned to zero at beginning
  - confined to be less than 10
  - incremented by one after each loop (could do `k += 1`)
- `for(;;)` makes infinite loop (no conditions)
- `x += 1` means `x = x + 1`; `x %= 4` means `x = x % 4`
  - `count` will go 1, 2, 3, 0, 1, 2, 3, 0, 1, 2 then end loop

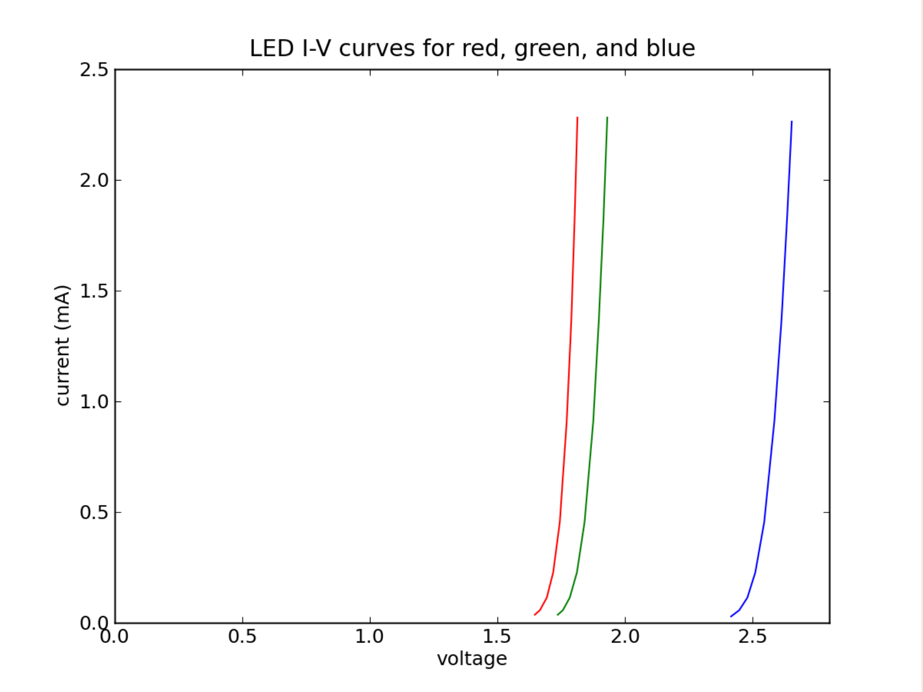
`#define` to ease the coding

```
#define NPOINTS 10
#define HIGHSTATE 1
```

- `#define` comes in the "preamble" of the code
  - note no semi-colons
  - just a text replacement process: any appearance of NPOINTS in the source code is replaced by 10
  - Convention to use all CAPs to differentiate from normal variables or commands
  - Now to change the number of points processed by that program, only have to modify one line
  - Arduino.h defines handy things like HIGH = 0x1, LOW = 0x0, INPUT = 0x0, OUTPUT = 0x1, INPUT_PULLUP = 0x2, PI, HALF_PI, TWO_PI, DEG_TO_RAD, RAD_TO_DEG, etc. to make programming easier to read/code

LED hookup

- The output of Arduino digital I/O pins will be either 0 or 5 volts
- An LED has a diode-like I-V curve
- Can't just put 5 V across
  - it'll blow, unless current is limited
- Put resistor in series, so ~2.5 V drop across each
  - 250 Ω would mean 10 mA
  - 10 mA is pretty bright



Blink Function (Subroutine)

- For complex blink patterns, it pays to consolidate blink operation into a function

```
void blink(int ontime, int offtime)
{
  // turns on LED (externally defined) for ontime ms
  // then off for offtime ms before returning
  digitalWrite(LED, HIGH);
  delay(ontime);
  digitalWrite(LED, LOW);
  delay(offtime);
}
```

- Now call with, e.g., `blink(600,300)`
- Note function definition expects two integer arguments
- LED is assumed to be global variable (defined outside of loop)

Blink Constructs

- For something like Morse Code, could imagine building functions on functions, like

```
void dot()
{ blink(200,200); }

void dash()
{ blink(600,200); }

void letterspace()
{ delay(400); }

void wordspace()
{ delay(800); }
```

- And then perhaps letter functions:

```
void morse_s()
{ dot(); dot(); dot(); letterspace(); }

void morse_o()
{ dash(); dash(); dash(); letterspace(); }
```

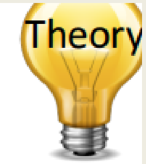
Morse, continued

- You could then spell out a word pretty easily like:

```
morse_s();
morse_o();
morse_s();
wordspace();
```

- Once you have a library of all the letters, it would be very simple to blink out anything you wanted

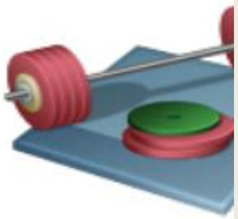
Temperature Measurements



Different methods for measuring the Tempertature:

- Thermocouples
- Thermistors
- RTD (Resistance Temperature Detector)
  - e.g. Pt100
- Infrared
- Thermometers

Temperature Sensors



Make the following Temperature Sensors work with Arduino:

NTC Thermistor

TMP36

Small-scale Temperature Sensors

TMP36

Technical data	
Temperature measurement range	-40...+125 °C
Accuracy	±2 °C (0...70 °C)
Power supply	2.3...5.5 V
Package	TO-92
Temperature sensitivity, voltage	10 mV/°C

https://www.sparkfun.com/products/10988

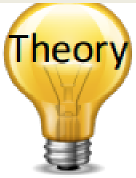
https://www.elfa.se/elfa3~eu_en/elfa/init.do?item=73-889-29&toc=0&q=73-889-29

NTC Thermistor

Technical data	
Resistance @ 25°C	10 kΩ
Temperature range	-40...+125 °C
Power max.	500 mW
Pitch	2.54 mm
Resistance tolerance	±5 %
$W_{25/100}$ value	3977 K
B value tolerance	±0.75 %
Thermal time constant	15 s

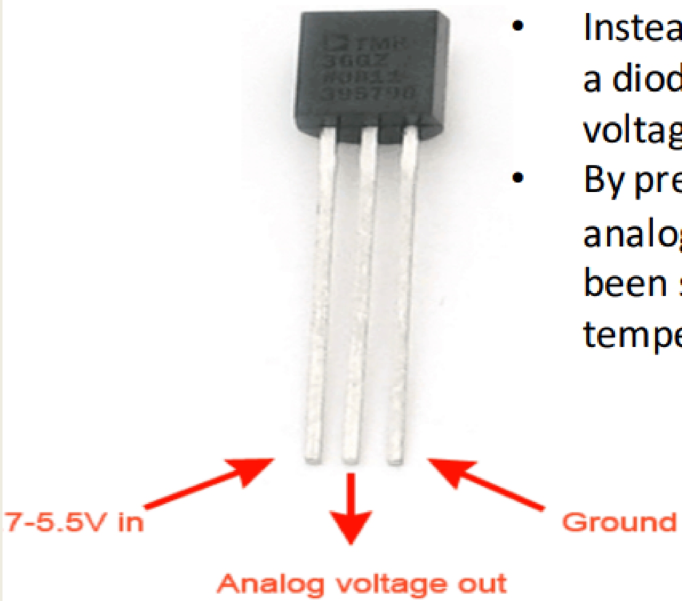
https://www.elfa.se/elfa3~eu_en/elfa/init.do?item=60-260-41&toc=0&q=60-260-41

Tutorial: http://garagelab.com/profiles/blogs/tutorial-using-ntc-thermistors-with-arduino



TMP36

- These sensors use a solid-state technique to determine the temperature. That is to say, they don't use mercury (like old thermometers), bimetalic strips (like in some home thermometers or stoves), nor do they use thermistors (temperature sensitive resistors).
- Instead, they use the fact as temperature increases, the voltage across a diode increases at a known rate. (Technically, this is actually the voltage drop between the base and emitter - the Vbe - of a transistor.)
- By precisely amplifying the voltage change, it is easy to generate an analog signal that is directly proportional to temperature. There have been some improvements on the technique but, essentially that is how temperature is measured.



Because these sensors have no moving parts, they are precise, never wear out, don't need calibration, work under many environmental conditions, and are consistant between sensors and readings. Moreover they are very inexpensive and quite easy to use.

ttps://learn.adafruit.com/tmp36-temperature-sensor

Datasheet Calculations

Theory

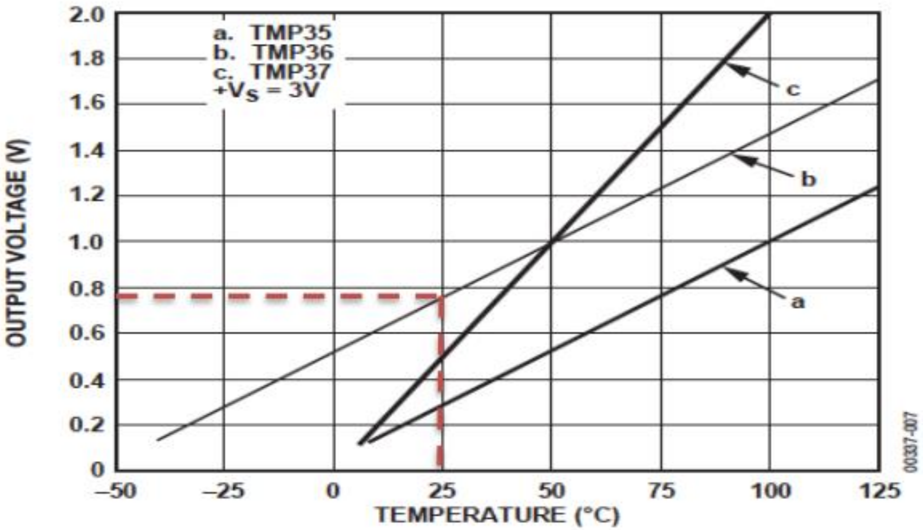


Figure 6. Output Voltage vs. Temperature

From the plot we have:

(x1, y1) = (750mV, 25°C)

(x2, y2) = (1000mV, 50°C)

Linear relationship: $y = ax + b$

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

You have to find a (slope) and b (intercept):

y-25°C = ((50°C-25°C)/(1000mV-750mV)) * (x-750mV)

This gives: y[°C] = (1/10)*x[mv]-50

25

Voltage-based Sensors

TMP36

According to the TMP36 datasheet, the relation of the output voltage to the actual temperature uses this equation:

$$y[°C] = (1/10)*x[mv]-50$$

Where the voltage value is specified in millivolts.

However, before you use that equation, you must convert the integer value that the analogRead function returns into a millivolt value.

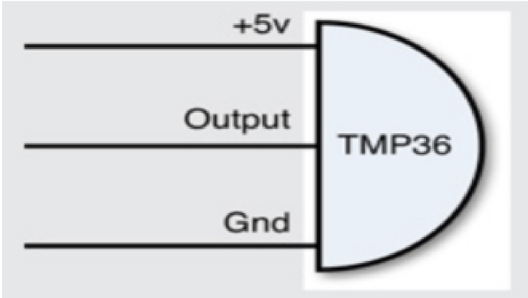
10-bit analog to digital converter

You know that for a 5000mV (5V) value span the analogRead function will return 1024 possible values:

$$\underbrace{\text{voltage}}_{\text{mV}} = (5000 / 1024) * \text{output}$$

Where

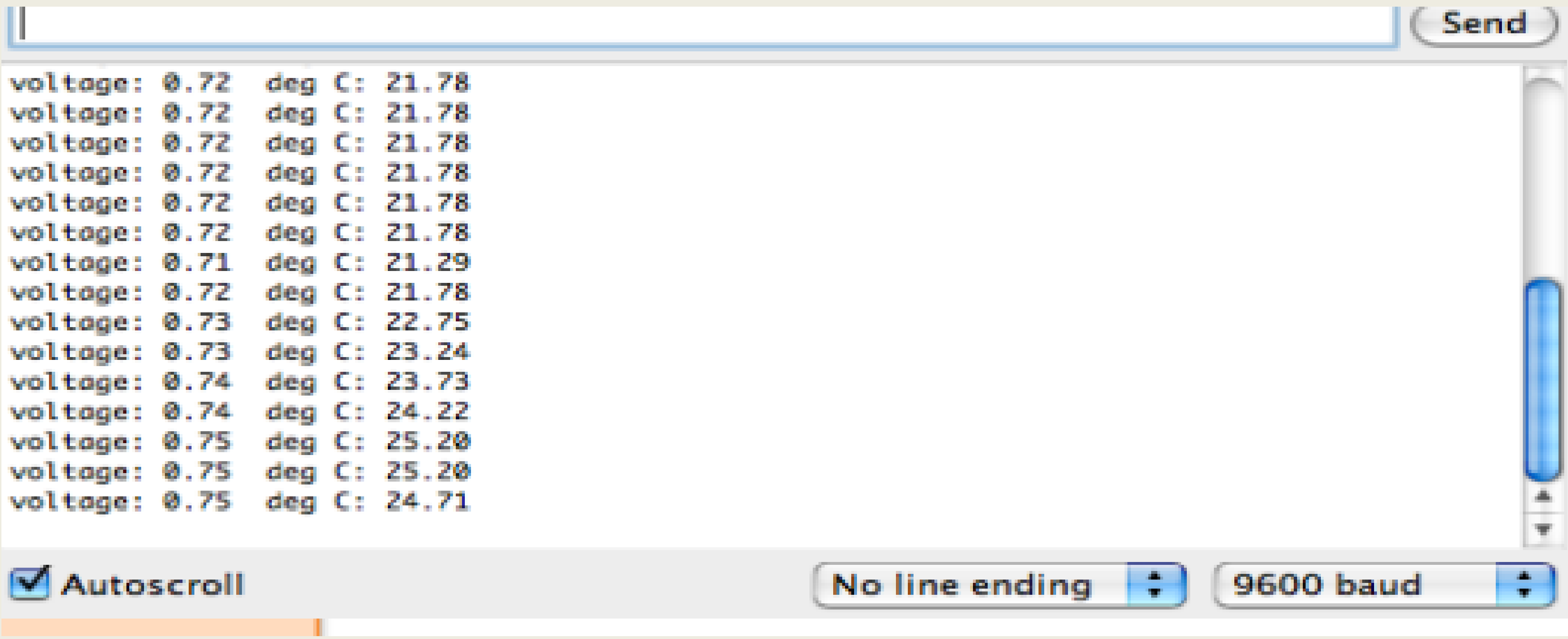
$$\underbrace{\text{output}}_{0\text{-}1023} = \text{analogRead}(\underbrace{\text{aichannel}}_{\text{A0-A5}})$$



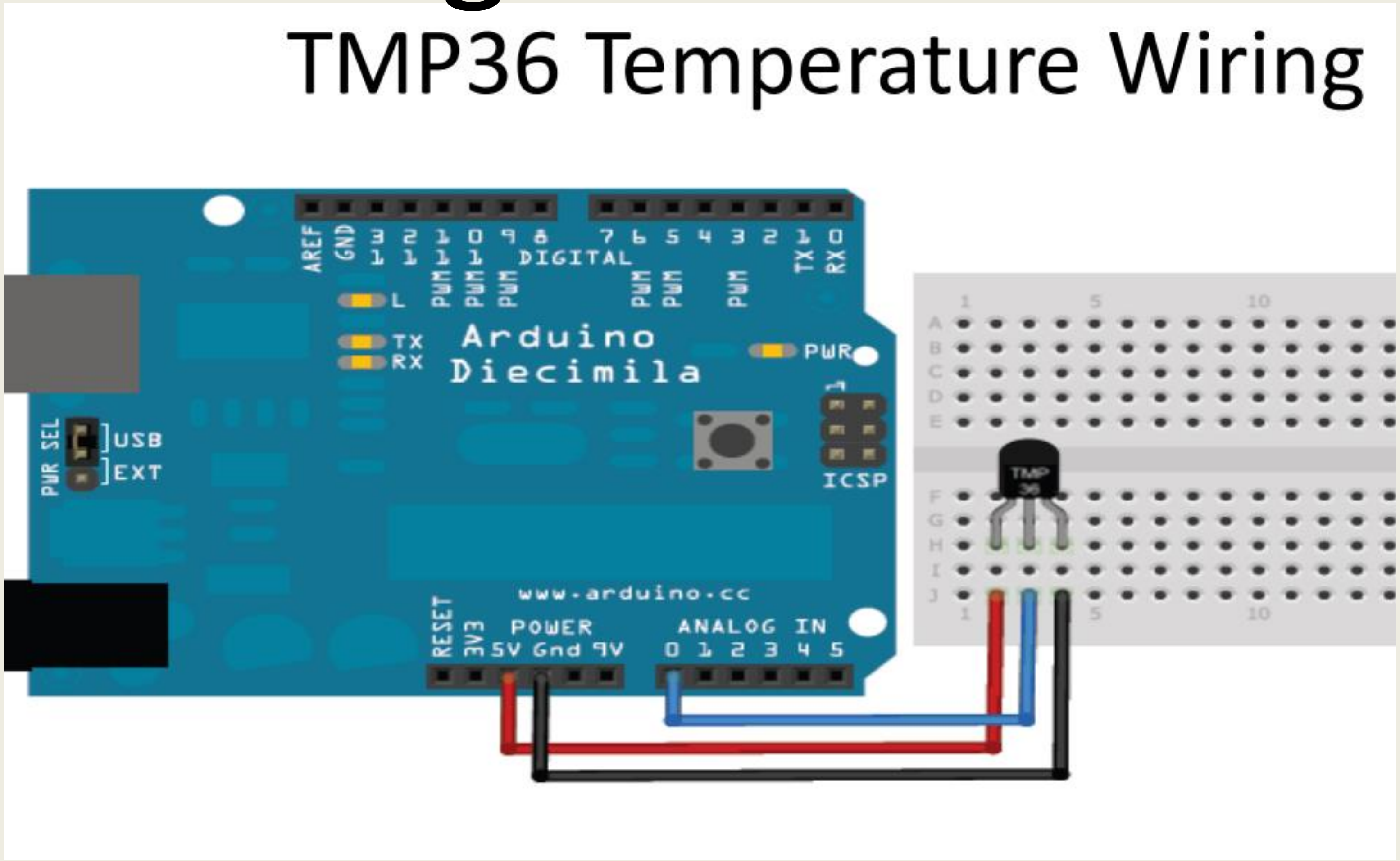
CODE For temperature measurement

```
// We'll use analog input 0 to read Temperature Data const int temperaturePin = 0;
void setup()
 { Serial.begin(9600); }
void loop()
{ float voltage, degreesC, degreesF;
voltage = getVoltage(temperaturePin);
 // Now we'll convert the voltage to degrees Celsius.
// This formula comes from the temperature sensor datasheet:
 degreesC = (voltage - 0.5) * 100.0;
 // Send data from the Arduino to the serial monitor window
 Serial.print("voltage: ");
 Serial.print(voltage);
Serial.print(" deg C: ");
Serial.println(degreesC);
 delay(1000);
// repeat once per second (change as you wish!) }
float getVoltage(int pin)
 { return (analogRead(pin) * 0.004882814); }
// This equation converts the 0 to 1023 value that analogRead()
// returns, into a 0.0 to 5.0 value that is the true voltage
 // being read at that pin.
```

In the Computer

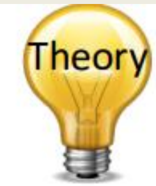


Wiring


TMP36 Temperature Wiring
AREF
GND
13 12 11 10 9 8 DIGITAL
7 6 5 4 3 2 1 0
TX
RX
PWM
L
TX
RX
Arduino
Diecimila
PWR
ICSP
USB
EXT
PWR SEL
www.arduino.cc
RESET
3V3
POWER
5V Gnd 9V
ANALOG IN
0 1 2 3 4 5
TMP
36
A
B
C
D
E
F
G
H
I
J
1
5
10

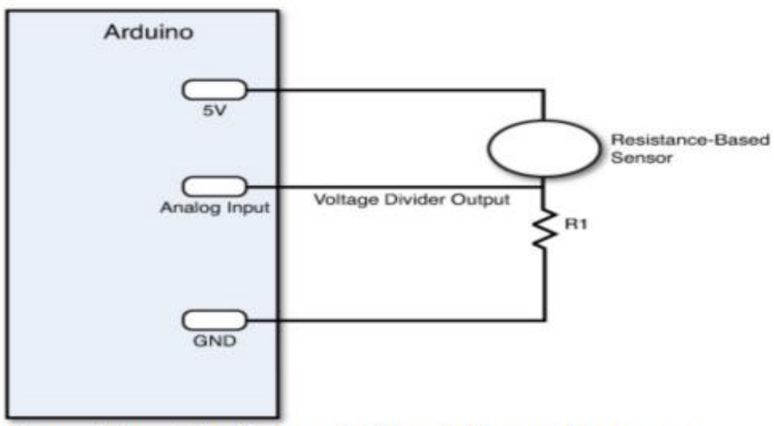
```
// We'll use analog input 0 to read Temperature Data const int t
void setup()
 { Serial.begin(9600); }
void loop()
{ float voltage, degreesC, degreesF;
voltage = getVoltage(temperaturePin);
 // Now we'll convert the voltage to degrees Celsius.
// This formula comes from the temperature sensor datasheet:
 degreesC = (voltage - 0.5) * 100.0;
 // Send data from the Arduino to the serial monitor window
Serial.print("voltage: ");
Serial.print(voltage);
Serial.print(" deg C: ");
Serial.println(degreesC);
delay(1000);
// repeat once per second (change as you wish!) }
float getVoltage(int pin)
 { return (analogRead(pin) * 0.004882814); }
// This equation converts the 0 to 1023 value that analogRead()
// returns, into a 0.0 to 5.0 value that is the true voltage
// being read at that pin.
```

Resistance-based Sensors



The problem with resistance sensors is that the Arduino analog interfaces can't directly detect resistance changes.
This will require some extra electronic components. The easiest way to detect a change in resistance is to convert that change to a voltage change. You do that using a **voltage divider**, as shown below.

Thermistor



By keeping the power source output constant, as the resistance of the sensor changes, the voltage divider circuit changes, and the output voltage changes. The size of resistor you need for the R1 resistor depends on the resistance range generated by the sensor and how sensitive you want the output voltage to change.

Generally, a value between 1K and 10K ohms works just fine to create a meaningful output voltage that you can detect in your Arduino analog input interface.

E.g., the Steinhart-Hart Equation can be used to find the Temperature:

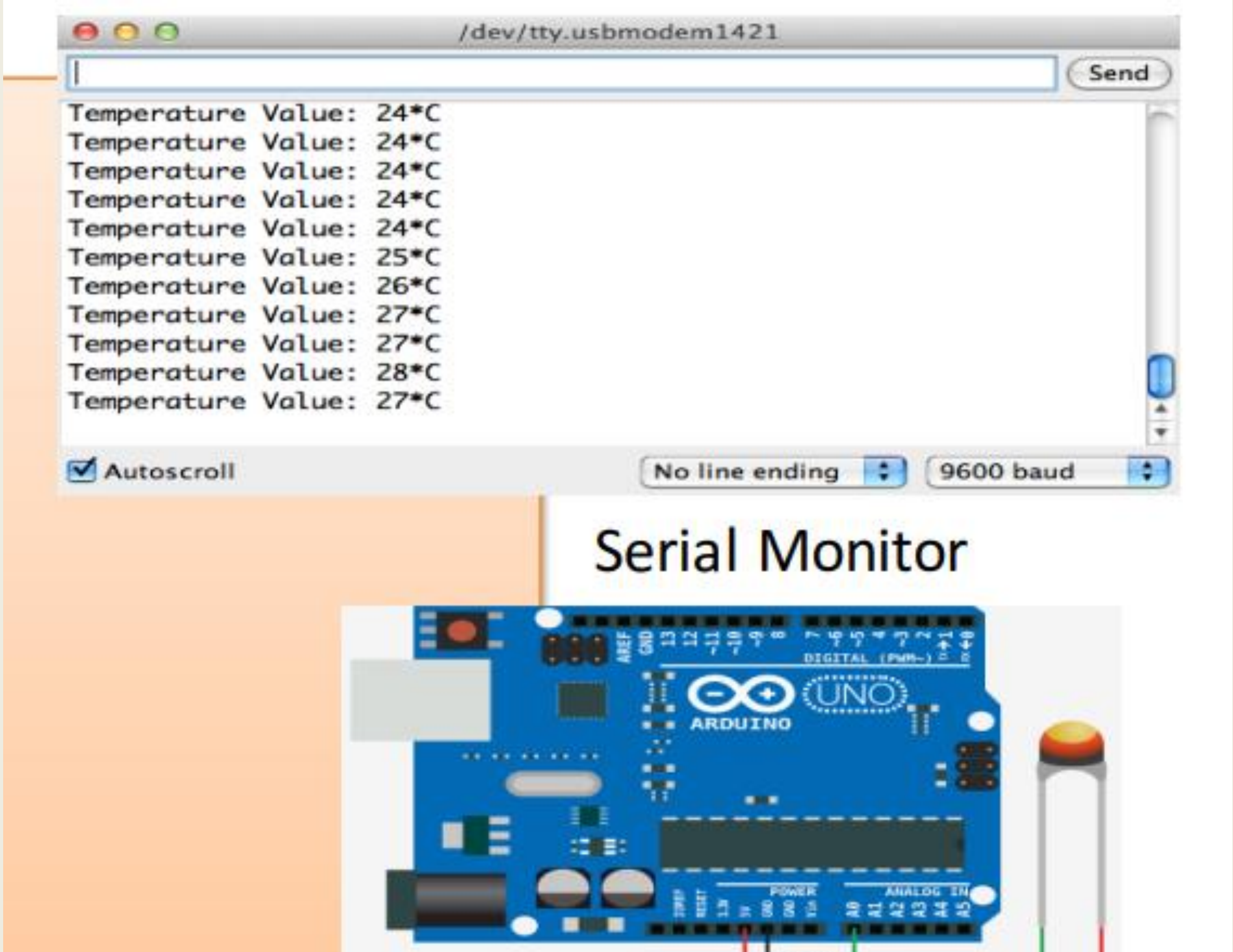
$$\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$$

```
// Read Temerature Values from NTC Thermistor
 const int temperaturePin = 0;
 void setup()
{ Serial.begin(9600); }
 void loop()
 { int temperature = getTemp();
Serial.print("Temperature Value: ");
Serial.print(temperature);
 Serial.println("*C");
delay(1000);
 }
 double getTemp()
{
// Inputs ADC Value from Thermistor and outputs Temperature in Celsius  int RawADC =
analogRead(temperaturePin);
long Resistance;
 double Temp;
 // Assuming a 10k Thermistor. Calculation is actually: Resistance = (1024/ADC)
Resistance=((10240000/RawADC) - 10000);
 // Utilizes the Steinhart-Hart Thermistor Equation:
// Temperature in Kelvin = 1 / {A + B[ln(R)] + C[ln(R)]^3}
// where A = 0.001129148, B = 0.000234125 and C = 8.76741E-08 Temp = log(Resistance);
 Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp *
Temp)); Temp = Temp - 273.15;
// Convert Kelvin to Celsius  return Temp;
 // Return the Temperature
 }
```

In Arduino

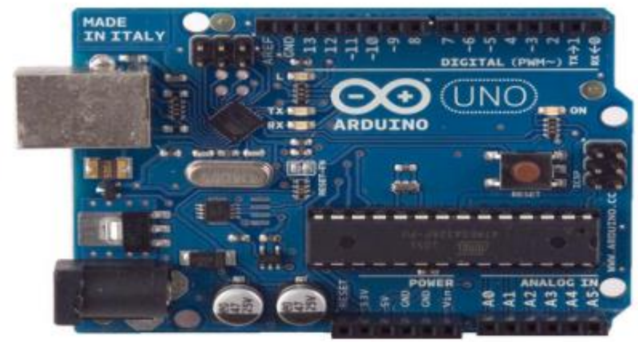
```
// Read Temerature Values from NTC Thermistor
 const int temperaturePin = 0;
 void setup()
{ Serial.begin(9600); }
 void loop()
 { int temperature = getTemp();
Serial.print("Temperature Value: ");
Serial.print(temperature);
 Serial.println("*C");
delay(1000);
 }
 double getTemp()
{
// Inputs ADC Value from Thermistor and outputs Temperature in Celsius int RawADC = analogRead(temperaturePin);
long Resistance;
 double Temp;
 // Assuming a 10k Thermistor. Calculation is actually: Resistance = (1024/ADC) Resistance=((10240000/RawADC) - 10000);
 // Utilizes the Steinhart-Hart Thermistor Equation:
// Temperature in Kelvin = 1 / {A + B[ln(R)] + C[ln(R)]^3}
// where A = 0.001129148, B = 0.000234125 and C = 8.76741E-08 Temp = log(Resistance);
 Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp * Temp)); Temp = Temp - 273.15;
// Convert Kelvin to Celsius return Temp;
 // Return the Temperature
 }
```

In Computer + wiring



Temperature Data Logger/Embedded DAQ System

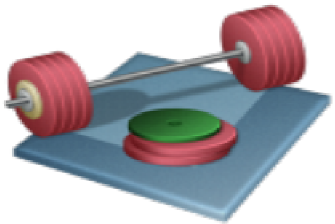
You use the PC when creating the software, then you download the software to the Arduino and disconnect the USB cable. Use e.g., a 9V battery or an external Power Supply.



NTC Thermistor

Use different Temperature sensors for comparison, i.e log data from 2 different sensors at the same time.

Temperature Data Logger/ Embedded DAQ System

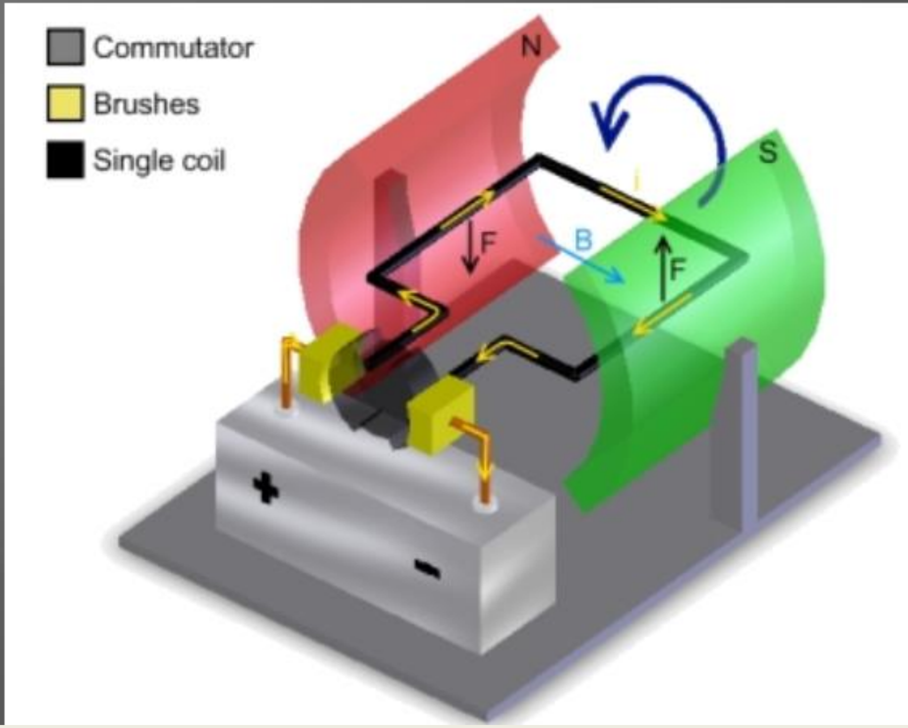


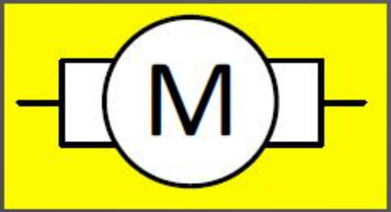
Create a **Temperature Logger**/Embedded DAQ System. Suggested Tasks:

- Create and use a **Lowpass Filter/Average Filter**
- **Alarm** functionality: Use LEDs with different colors when Temperature is above/below the Limits
- Use e.g., Arduino **Wi- Fi/Ethernet Shield** for Communication over a network - or use the microSD card on these Shields
- Save the data to a microSD card located on the Wi- Fi/Ethernet Shield - or connect e.g., to **xively.com** or **temboo.com** - which are free datalogging sites.
- Log Temperature Data for e.g., 24 hours and import Data into Excel, LabVIEW or MATLAB for Analysis and Visualization
- Use e.g. a 9V battery or an external power source to make it portable and small

DC Motor

- DC motors spin when a steady voltage is applied
  - Can draw significant current (~ 1A or more)
- Fixed permanent magnet
- Rotating coil
- Brushes



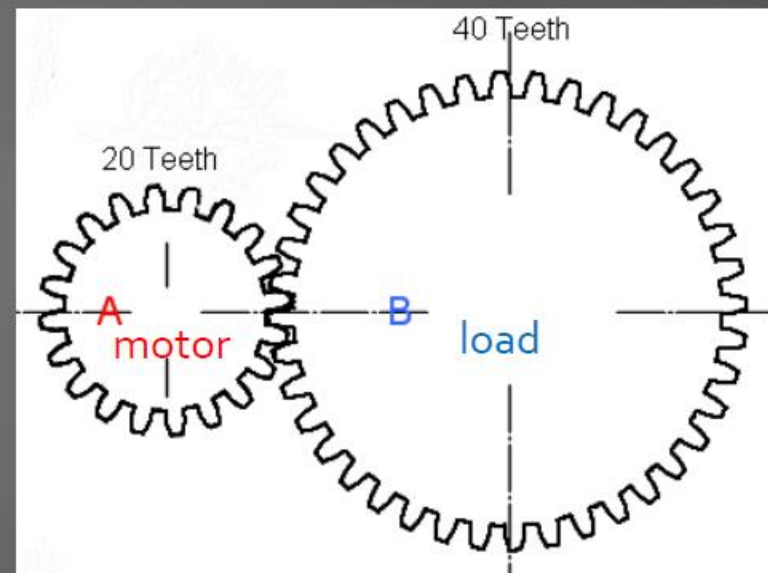


E11 Motors

- Operating Voltage: 3-12 V
- At 6 V operation:
  - Free run speed: 11,500 RPM
  - Unloaded current: 70 mA
  - Stall current: 800 mA
  - ~0.5 oz-in torque

Gearing

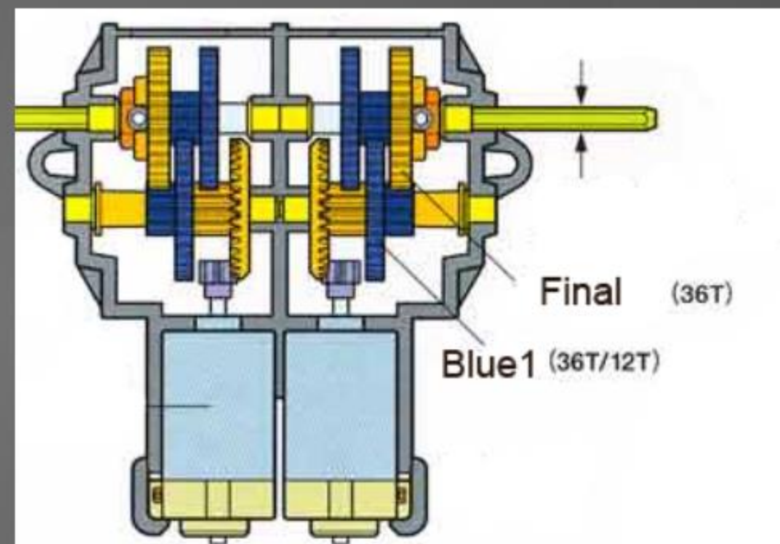
- DC motors spin too fast
  - And too little torque
- Gears slow the load rotation
  - Also increase torque
- In this example, load spins at half the speed of the driver

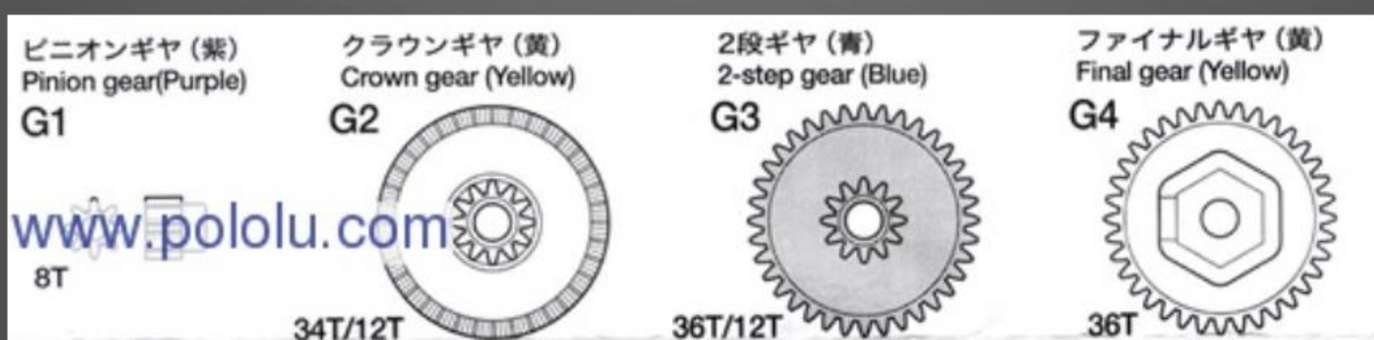


- Gear ratio: $\omega_B / \omega_A = N_A / N_B$

Example: Tamiya Gear Box

- Gear Ratio:
  - Final to Blue1
  - Blue1 to Blue2
  - Blue2 to Crown
  - Crown to Pinion
  - Total:





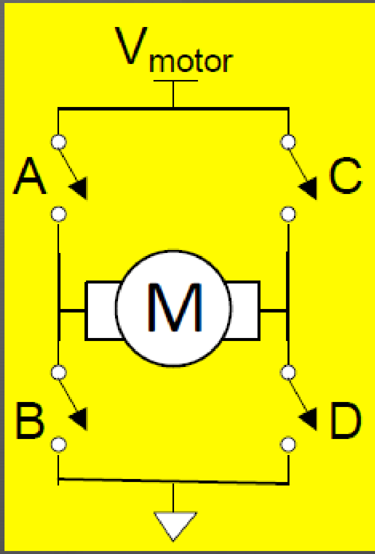
pololu.com

7

H-Bridge

- Motors require large current to operate
  - But Arduino outputs only offer 40 mA
- H-Bridges are used to drive the large current

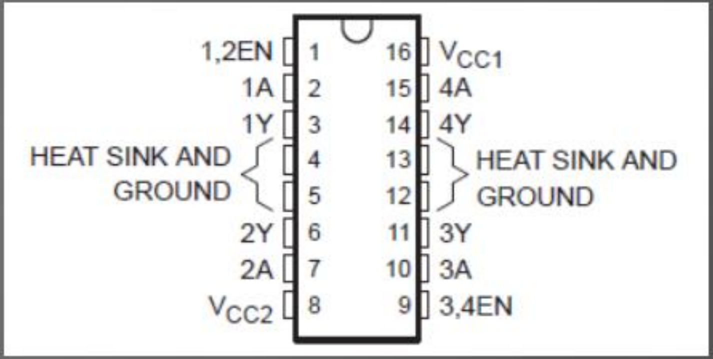
A	B	C	D	Motor
ON	OFF	OFF	ON	
OFF	ON	ON	OFF	
ON	OFF	ON	OFF	
OFF	OFF	OFF	OFF	
ON	ON	OFF	OFF	

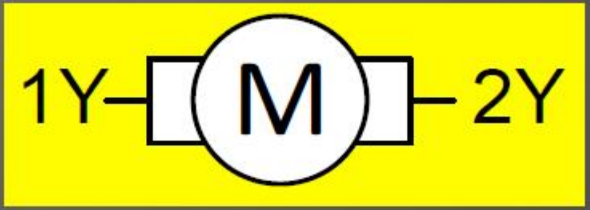


SN754410 H-Bridge

- 754410 Dual H-Bridge is easy to control with digital logic
  - V_{CC1} = Logic Supply (5V)
  - V_{CC2} = Motor Supply (4.5-36 V)

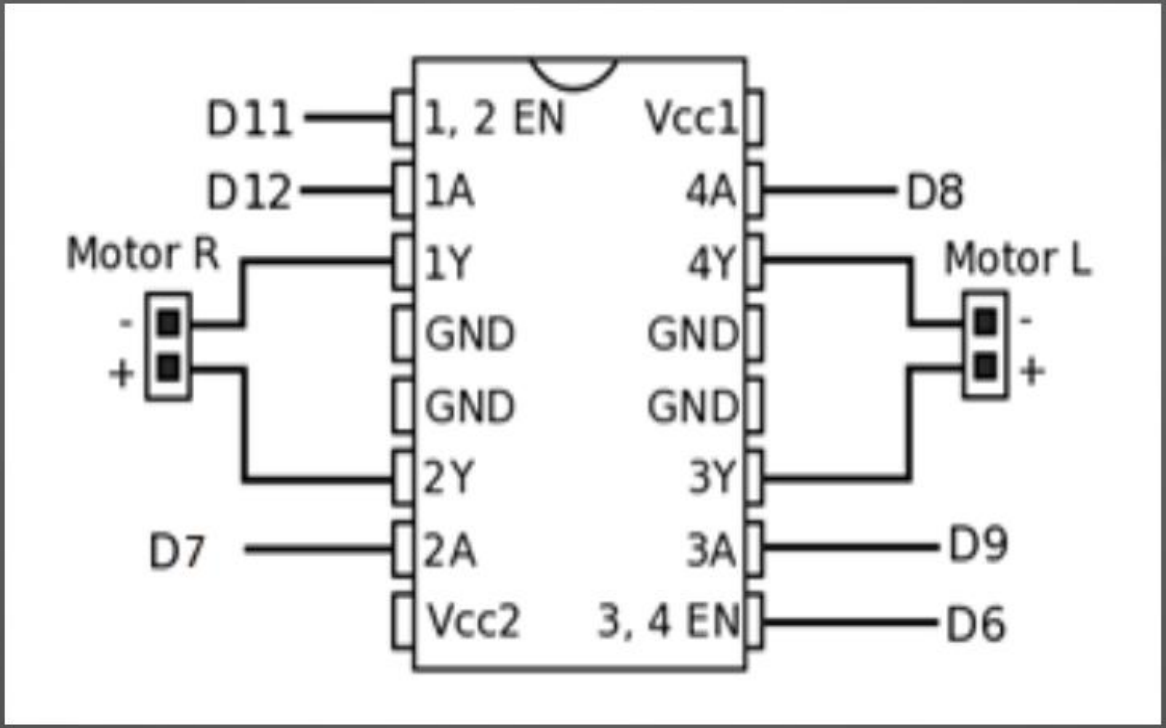
12En	1A	2A	Motor
0	X	X	
1	0	0	
1	0	1	
1	1	0	
1	1	1	





- Contains two H-Bridges to drive two motors

Mudduino H-Bridge Interface



Motor Driver Software

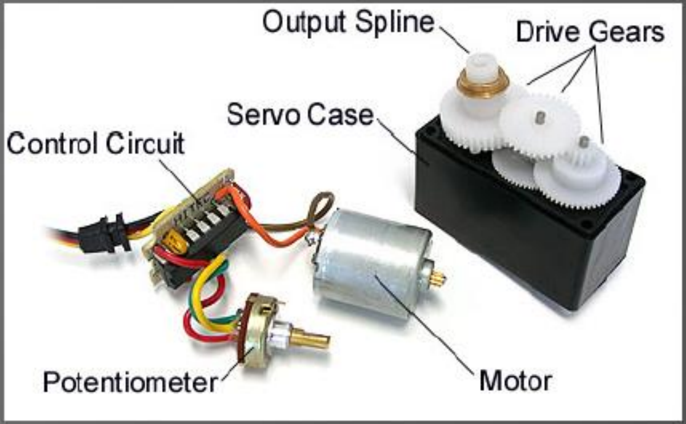
```
#define LEN 6
#define LPLUS 9
#define LMINUS 8

void forward(void)
{
      digitalWrite(LEN,    1);
      digitalWrite(LPLUS,  1);
      digitalWrite(LMINUS, 0);
      // similar for right motor…
}
```

Shaft Encoding

- Sometimes it helps to know the position of the motor
- Optical shaft encoder
  - Disk with slits attached to motor shaft
  - Light and optical sensor on opposite sides of disk
  - Count light pulses as the disk rotates
- Analog shaft encoder
  - Connect potentiometer (variable resistor) to shaft
  - Resistance varies as shaft turns
- Our DC motors don't have shaft encoders built in

Servo Motor

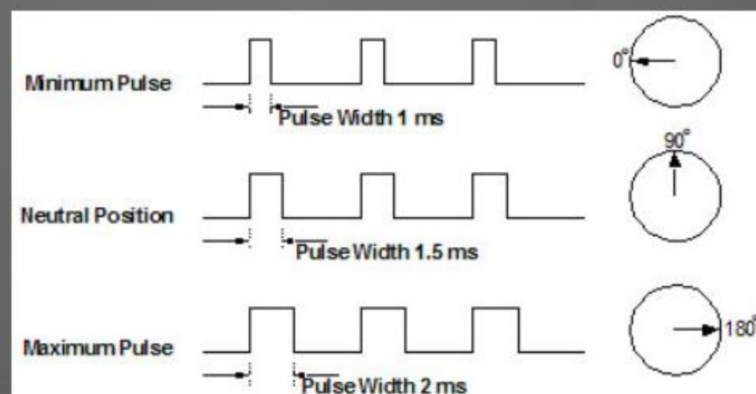
- Servo motors are designed to be easy to use
  - DC motor
  - Gearing
  - Analog shaft encoder
  - Control circuitry
  - High-current driver
- Three wires: 5V, GND, Control
- Turn from 0 to 180 degrees
  - Position determined by pulses on control wire



servocity.com

Servo Pulse Width Modulation

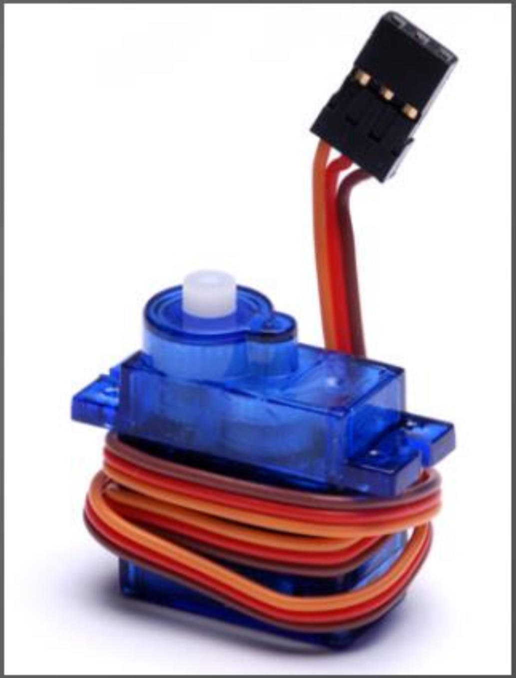
- Control position with 50 Hz (20 ms) pulses
- Pulse width modulation (PWM)
  - 1 ms = 0°
  - 1.5 ms = 90°
  - 2 ms = 180°



servocity.com

SG90 Servo

- 4.0 – 7.2 V Operation
- At 4.8 V
  - Speed: 0.12 sec / 60 degrees (83 RPM)
  - Stall Torque: 16.7 oz-in



hobbypartz.com

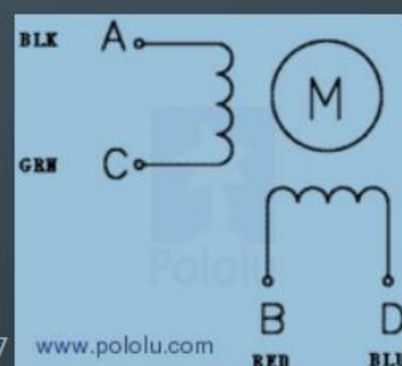
Arduino Servo Library

- Arduino offers a servo library for controlling servos

```
// servotest.pde
// David_Harris@hmc.edu 1 October 2011

#include <Servo.h>

// pins
#define SERVOPIN 10

// Global variable for the servo information
Servo servo;

void testServo()
{
  initServo();
  servo.write(90);  // set angle between 0 and 180 degrees
}

void initServo()
{
  pinMode(SERVOPIN, OUTPUT);
  servo.attach(SERVOPIN);
}
```

Stepper Motor

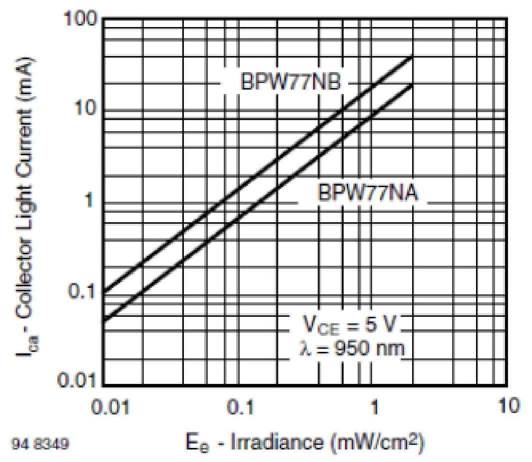
- Stepper motors are also popular
  - Motor advances in discrete steps
  - Input pulses indicate when to advance
- Example: Pololu 1207 Stepper Motor
  - 1.8° steps (200 steps/revolution)
  - 280 mA @ 7.4 V
  - 9 oz-in holding torque
  - Needs H-Bridge driver
  - Ground C and D
  - Alternate pulses to A and B

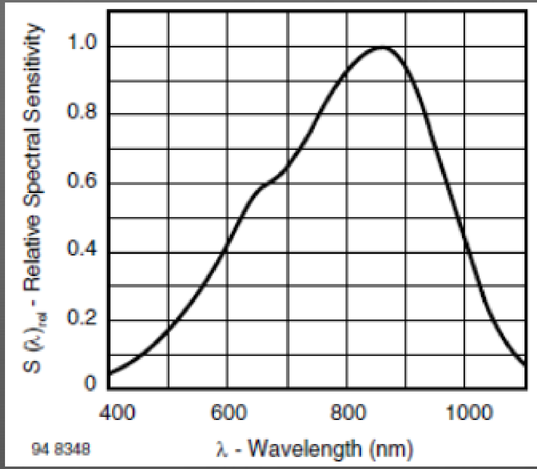


17

Phototransistor

- Converts light to electrical current
- Vishay BPW77NA NPN Phototransistor
  - Dark current: 1 – 100 nA
  - Angle of half sensitivity: ±10°



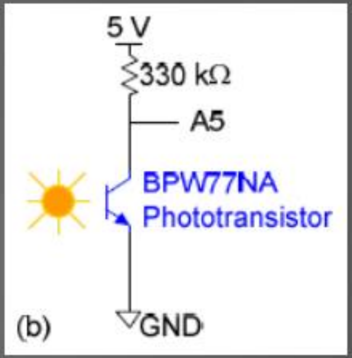


vishay.com 18

jameco.com

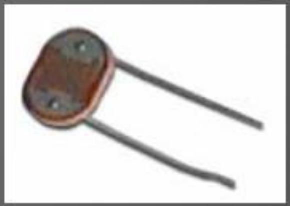
Phototransistor Circuit

- Leave base terminal unconnected
- $V_{out} = 5 - I_{photo} \times 330\ k\Omega$
  - In dark, $V_{out} \approx 5$ V
  - For $I_{photo} > 15\ \mu A$, V_{out} drops to ~0
- Large resistor gives sensitivity to weak light



Other Light Sensors

- Photodiodes
  - Similar to phototransistors
  - Lower sensitivity
- Cadmium Sulfide (CDS) Cell
  - Resistance changes with light
    - From > 1 MΩ in dark to 200 Ω in full light
  - Slow response time



goldmine-elec-products.com

Sensor Read Code

```
#define PHOTO_TRANS 19

void setup()
{
  Serial.begin(9600);

  // configure sensors
  pinMode(PHOTO_TRANS, INPUT);
}

void loop()
{
  int sensor;

  // test sensors
  sensor = analogRead(PHOTO_TRANS-14); // analogRead uses analog port #
  Serial.print("Reflectance sensor: "); Serial.println(sensor);
  delay(500);
}
```

Sensor Averaging

- Sensors are subject to noise
- Average multiple readings for more stable results

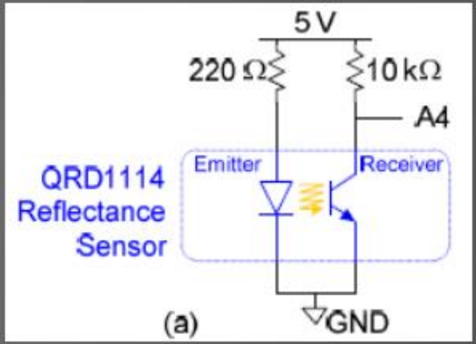
Reflectance Sensor

- Infrared LED and phototransistor pair
  - LED illuminates surface
  - Phototransistor receives reflected light
  - Daylight filter on sensor reduces interference
  - Sensitive to distance, color, reflectivity
- Fairchild QRD1114 Reflectance Sensor
  - ~20 mA LED current
  - 1.7 V LED ON voltage
  - 940 nm wavelength (near infrared)

fairchild.com

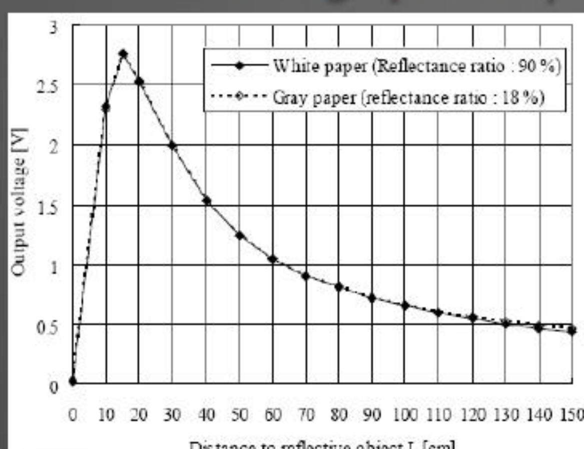
Reflectance Sensor Circuit

- $I_{LED} = (5\text{-}1.7\text{ V}) / 220\ \Omega = 15\text{ mA}$
- $V_{out} = 5 - I_{photo} \times 10\text{ k}\Omega$
- Resistor was selected to give a good range of response



IR Distance Sensor

- Sharp GP2Y0A21YK0F
- Range of 8 to 60"
- Triangulates with linear CCD array
- Three terminals: 5V, GND, Signal



25

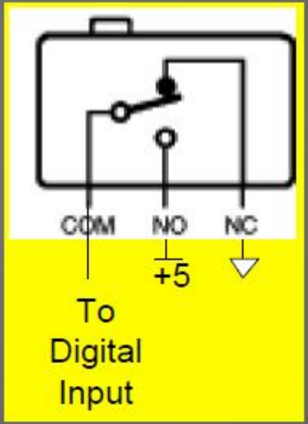
Ultrasonic Distance Sensor

- Measure flight time of ultrasonic pulse
  - Less sensitive to ambient light
  - More precise
  - More expensive
- Example: LV-MaxSonar-EZ
  - 42 KHz ultrasonic beam
  - Range of 254″ with resolution of 1″
  - 2.5 – 5.5 V operation
  - Analog voltage output

maxbotix.com

Switches

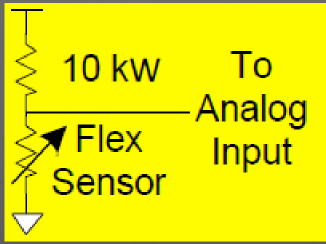
- Switches are useful for proximity detection
- Three terminals
  - COM: Common
  - NO: Normally Open
  - NC: Normally Closed
- Mounting issues
  - Good supporting surface
  - Gang 2 or more with plate between



sparkfun.com

Flex Sensors

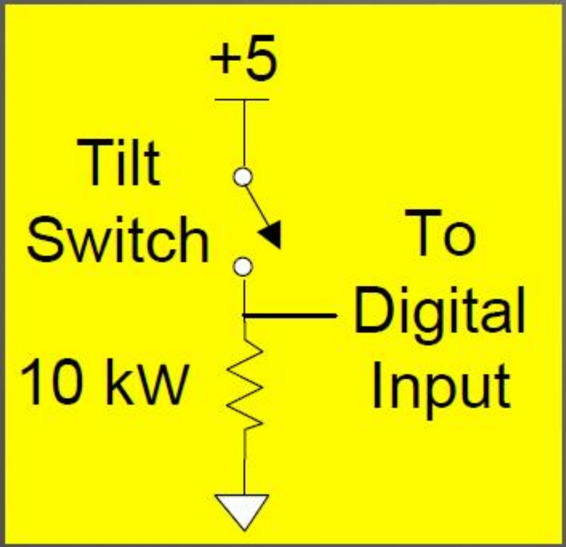
- Resistance changes with flex
- Example: Spectra Symbol Flex
  - 4.5" length
  - 10 KΩ ± 30% when flat
  - 60-110 KΩ when bent
- Sample Circuit
  - V_{out} = 2.5 V when flat
  - Increases when bent

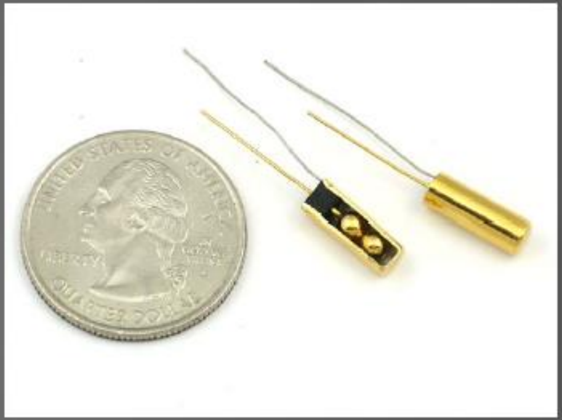


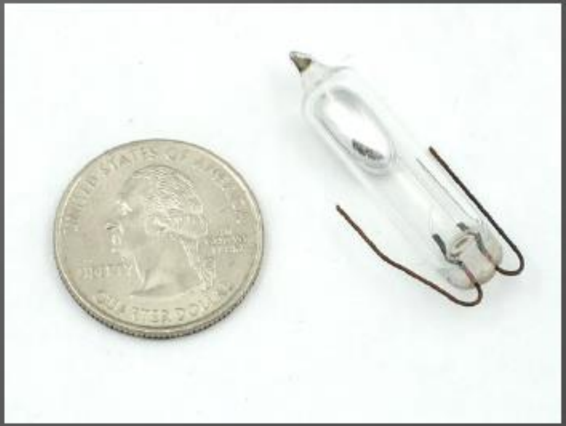
sparkfun.com

Tilt Switches

- **Mercury or Ball**
- **Warn if your bot is about to topple!**







Navigation Sensors

- Track your position
  - Watch for operating voltage and analog/digital interface
  - Some of these sensors are expensive!
- Sparkfun
  - HMC6352 Digital Compass
  - MLX90609 Single Axis Gyroscope
  - ITG-3200 Triple Axis Gyroscope
  - ADXL322 Dual Axis Accelerometer
  - Inertial Measurement Units

Mounting Sensors & Actuators

- Secure mounting is half the challenge
  - Poorly mounted sensors will fail at an inopportune time
  - Tangles of cables will catch on obstructions and pull loose
  - High center of gravity leads bots to topple in collisions
- Consider building a custom mount
  - Machine shop
  - 3D printer
- Use Breadboard to test electronics
  - Solder final electronics onto front of Mudduino for security

31

Adhesives

- Cynoacrylate (CA) Glue (aka Super Glue)
  - Fast drying, good for bonding plastic
  - Low shear strength
  - Don’t bond your fingers – wear gloves
- Hot Glue
- Electrical Tape
  - Insulator, low strength
- Gaffer’s Tape
  - Like duct tape, but stronger and removes cleanly