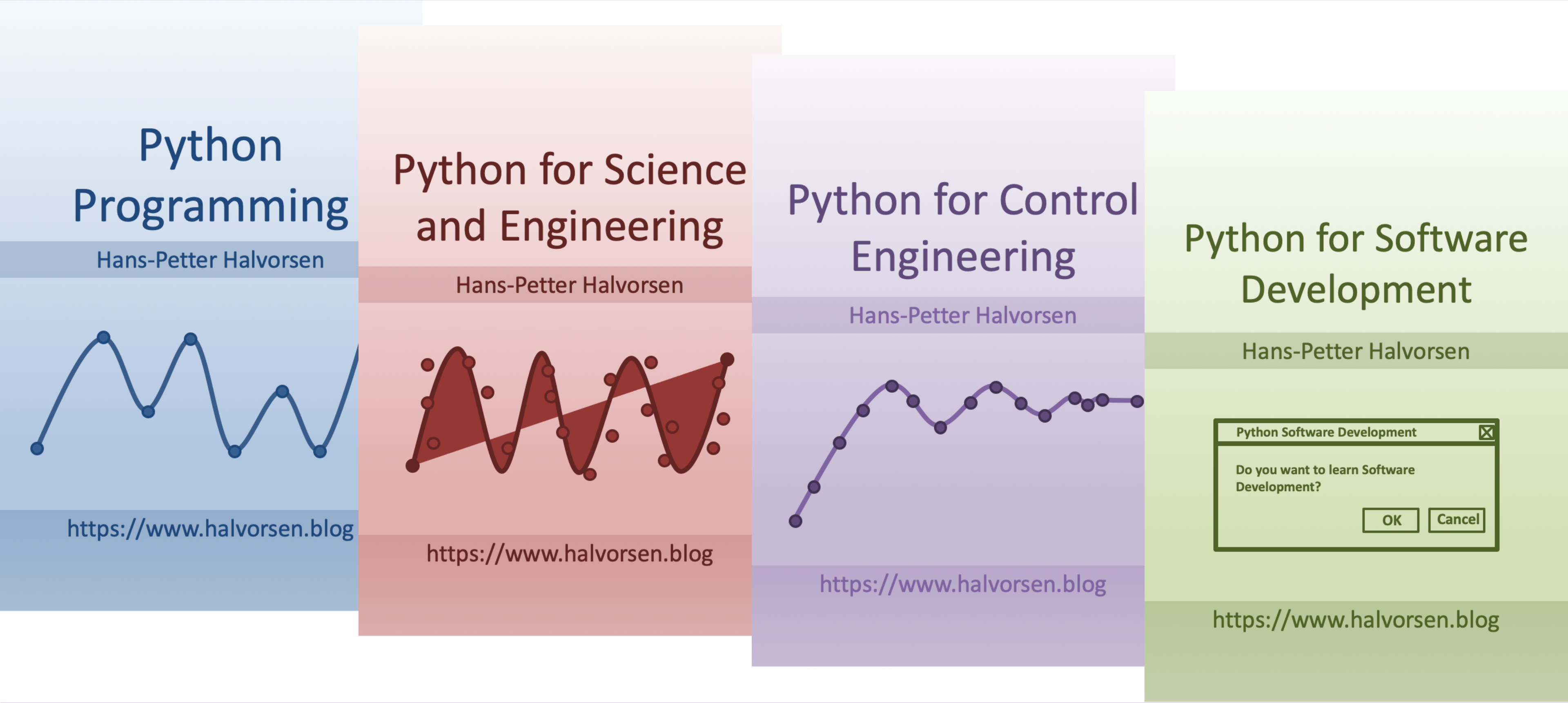
Stability Analysis of Control Systems with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

https://www.halvorsen.blog/documents/programming/python/

Additional Python Resources



https://www.halvorsen.blog/documents/programming/python/

Contents

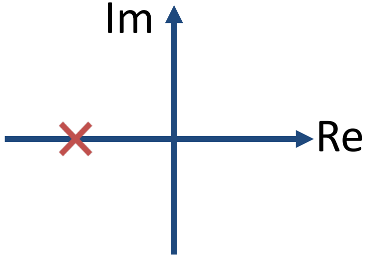
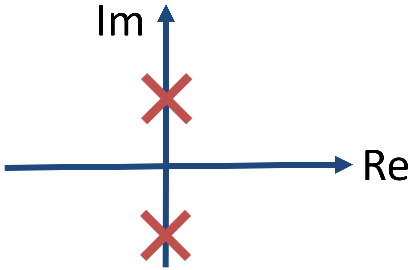
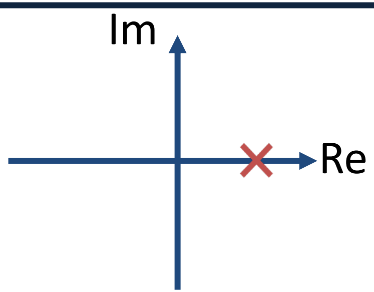
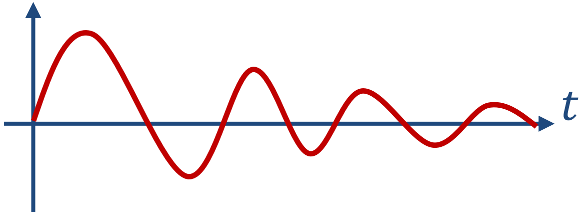
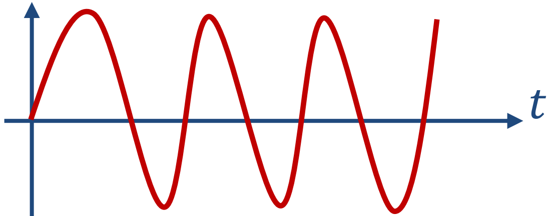
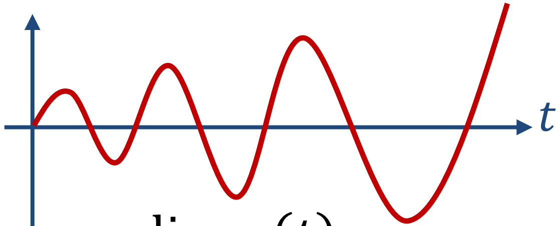
- Stability Analysis
- Poles
- Frequency Response
- Python Examples
  - SciPy (SciPy.signal)
  - Python Control Systems Library (control)

Stability Analysis

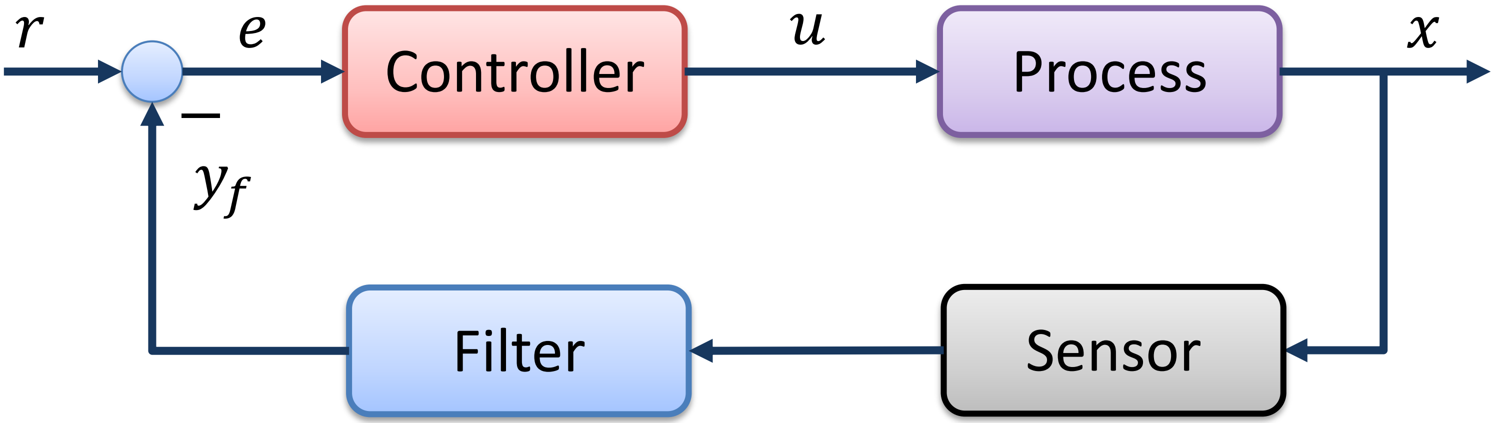
It is important to check the Stability properties of a given Control System and perform simulations before applied to the real process

- In the complex domain we can check the stability of the control system by the placements of the poles
- In the time domain we can simulate the system, e.g., performing a simple step response
- In the frequency domain we can check stability properties using, e.g., a Bode diagram

Stability Analysis

	Asymptotically Stable System	Marginally Stable System	Unstable System
Poles:			
Step Response:	 $\lim_{t\to\infty} y(t) = k$	 $0 < \lim_{t\to\infty} y(t) < \infty$	 $\lim_{t\to\infty} y(t) = \infty$
Frequency Response:	$\omega_c < \omega_{180}$	$\omega_c = \omega_{180}$	$\omega_c > \omega_{180}$

Control System



Transfer Functions

- In Stability Analysis and Control System design we typically use Transfer Functions.
- Typically we need to find a mathematical model of the process in form of a Transfer Function like this:

$$H_p(s) = \frac{y(s)}{u(s)}$$

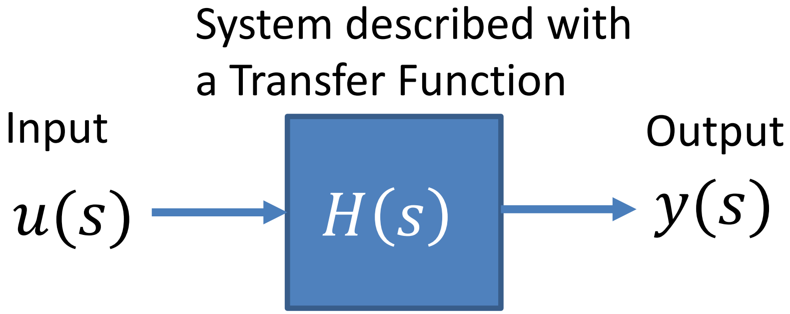
- Transfer functions are a model form based on the Laplace transform
- You can find the Transfer function(s) from the differential equation(s) or from logged data from the real process

Transfer Functions

A general Transfer function is on the form:

$$H(s) = \frac{y(s)}{u(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1\, s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1\, s + a_0} = \frac{(T_{n1}s + 1)(T_{n2}s + 1)\cdots(T_{nk}s + 1)}{(T_{d1}s + 1)(T_{d2}s + 1)\cdots(T_{dm}s + 1)}$$

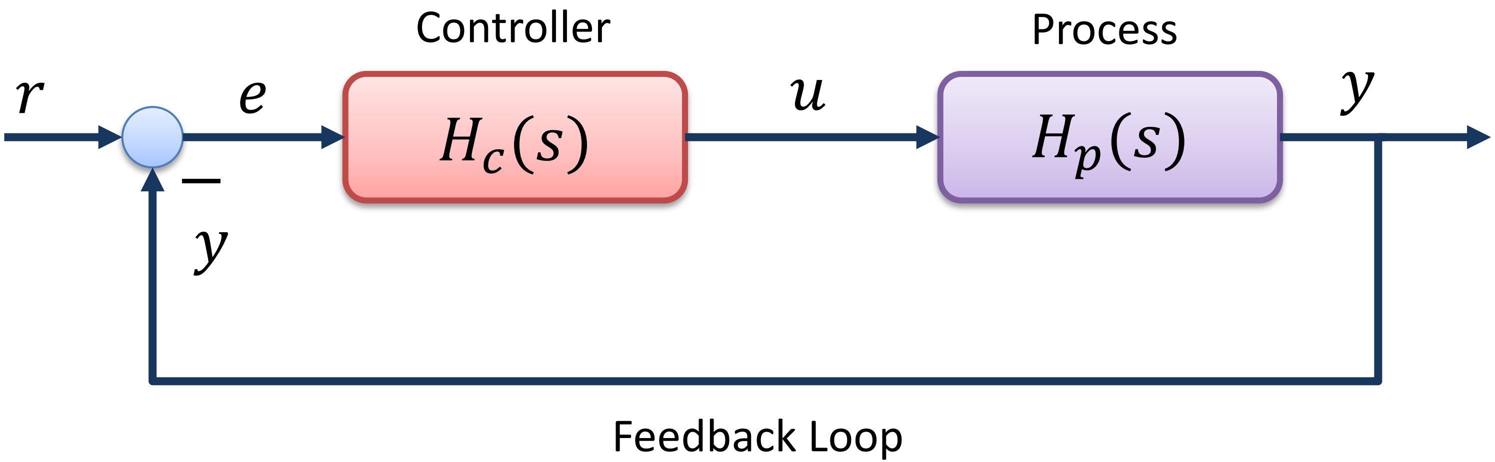
Where y is the output and u is the input. The symbol “s” is the Laplace operator.



A Transfer Function can also easily be implemented in Python

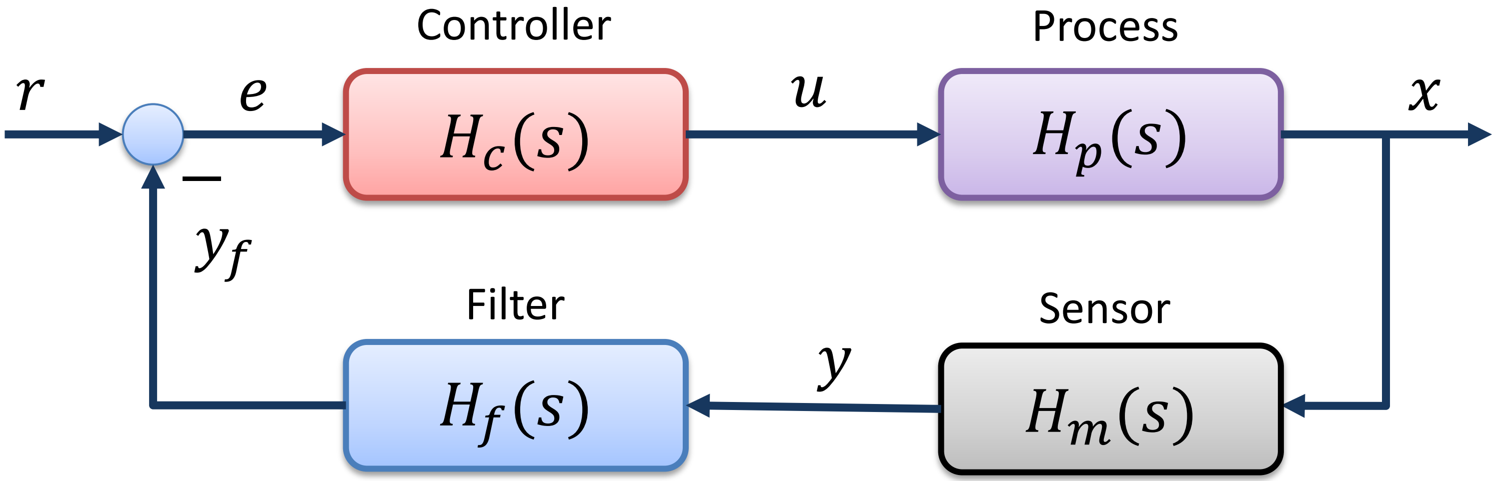
Basic Control System

Below we see a basic Control System consisting of a Process and a Controller:



Control System

The Control Loop basically consists of a set of Transfer Function. This is a more sophisticated example:



A Lowpass Filter that remove/ reduce Noise in the Measurements

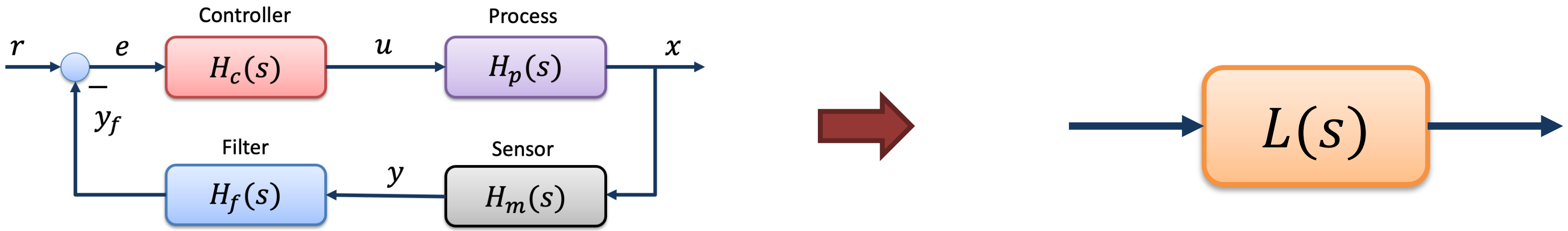
Sensor(s) the measure(s) one or more process variables

Sometimes you just include the sensor as part of the process and sometimes you don’t need a Filter

Loop Transfer Function

The Loop Transfer Function $L(s)$ is defined as follows:

$$\boxed{L(s) = H_c(s)H_p(s)H_m(s)H_f(s)\cdots}$$



The Loop Transfer Function is the <u>product of all the transfer functions in the loop</u>

Tracking Transfer Function

The Tracking Transfer Function $T(s)$ is defined as follows:

$$\boxed{T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)}}$$

The Tracking Transfer Function $T(s)$ is the transfer function from the reference/setpoint (r) to the process output variable (y)

$$r(s) \longrightarrow \boxed{T(s)} \longrightarrow y(s)$$

The Tracking Property is good if the tracking function T has value equal to or close to 1:

$$|T| \approx 1$$

Python Libraries

NumPy, Matplotlib

- In addition to Python itself, the Python libraries NumPy, Matplotlib is typically needed in all kind of applications
- If you have installed Python using the Anaconda distribution, these are already installed

SciPy.signal

The Signal Module in the SciPy Library

https://docs.scipy.org/doc/scipy/reference/signal.html

With SciPy.signal you can create Transfer Functions, State-space Models, you can simulate dynamic systems, do Frequency Response Analysis, including Bode plot, etc.

Continuous-time linear systems

lti(*system)	Continuous-time linear time invariant system base class.
StateSpace(*system, **kwargs)	Linear Time Invariant system in state-space form.
TransferFunction(*system, **kwargs)	Linear Time Invariant system class in transfer function form.
ZerosPolesGain(*system, **kwargs)	Linear Time Invariant system class in zeros, poles, gain form.
lsim(system, U, T[, X0, interp])	Simulate output of a continuous-time linear system.
lsim2(system[, U, T, X0])	Simulate output of a continuous-time linear system, by using the ODE solver **scipy.integrate.odeint**.
impulse(system[, X0, T, N])	Impulse response of continuous-time system.
impulse2(system[, X0, T, N])	Impulse response of a single-input, continuous-time linear system.
step(system[, X0, T, N])	Step response of continuous-time system.
step2(system[, X0, T, N])	Step response of continuous-time system.
freqresp(system[, w, n])	Calculate the frequency response of a continuous-time system.
bode(system[, w, n])	Calculate Bode magnitude and phase data of a continuous-time system.

Python Control Systems Library

- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: https://pypi.org/project/control
- Python Control Systems Library Documentation: https://python-control.readthedocs.io

Transfer Function in Python

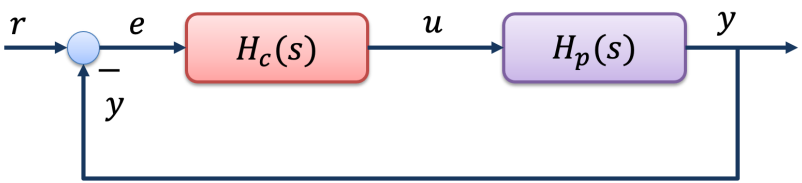
Transfer Function Example:

$$H(s) = \frac{y(s)}{u(s)} = \frac{2}{3s + 1}$$

```
import numpy as np
import control

# Define Transfer Function
num = np.array([2])
den = np.array([3 , 1])

H = control.tf(num , den)
print ('H(s) =', H)
```

Loop Transfer Function

Control System:



PI Controller:

$$H_c(s) = \frac{K_p(T_i s + 1)}{T_i s}$$

Process (random example):

$$H_p(s) = \frac{2}{3s + 1}$$

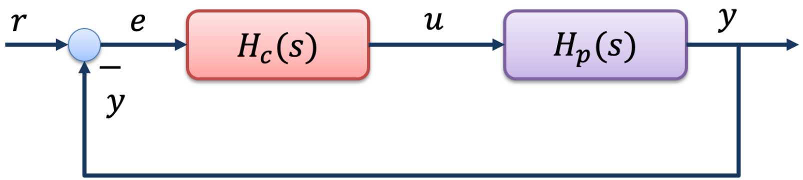
Loop Transfer Function:

$$L(s) = H_c(s)H_p(s)$$

```
import numpy as np
import control

# Controller
Kp = 0.4
Ti = 2
num = np.array ([Kp*Ti, Kp])
den = np.array ([Ti , 0])
Hc = control.tf(num , den)

# Process
num = np.array ([2])
den = np.array ([3 , 1])
Hp = control.tf(num , den)

L = control.series(Hc, Hp)
print(L)
```

Tracking Transfer Function

Control System:



PI Controller:

$$H_c(s) = \frac{K_p(T_i s + 1)}{T_i s}$$

Process (random example):

$$H_p(s) = \frac{2}{3s + 1}$$

Loop Transfer Function:

$$L(s) = H_c(s)H_p(s)$$

Tracking Transfer Function:

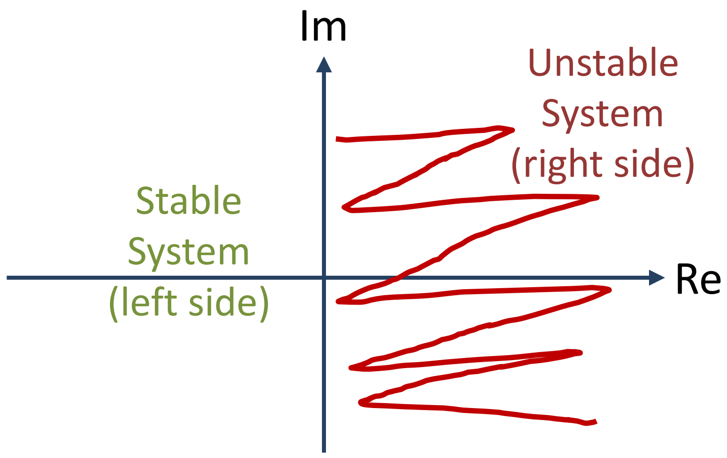
$$T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)}$$

```
import numpy as np
import control

# Controller
Kp = 0.4
Ti = 2
num = np.array ([Kp*Ti, Kp])
den = np.array ([Ti , 0])
Hc = control.tf(num , den)

# Process
num = np.array ([2])
den = np.array ([3 , 1])
Hp = control.tf(num , den)

L = control.series(Hc, Hp)
print(L)

T = control.feedback(L,1)
print(T)
```

Poles

Poles and Stability of the System

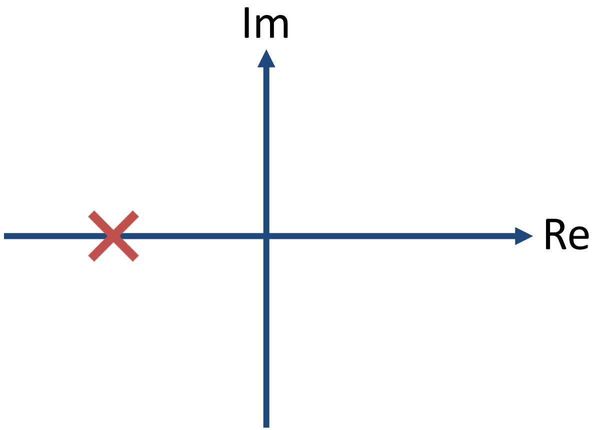
The poles are important when analyzing the stability of a system. The Figure below gives an overview of the poles impact on the stability of a system.



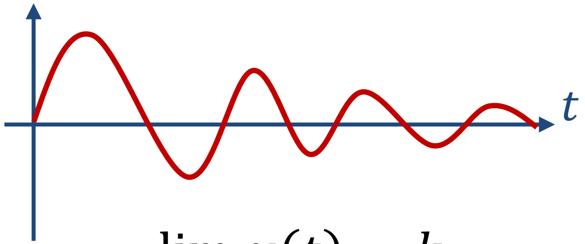
We have 3 different Alternatives:

1. Asymptotically Stable System
2. Marginally Stable System
3. Unstable System

Asymptotically Stable System



Each of the poles of the transfer function lies strictly in the left half plane (has strictly negative real part)



$$\lim_{t \to \infty} y(t) = k$$

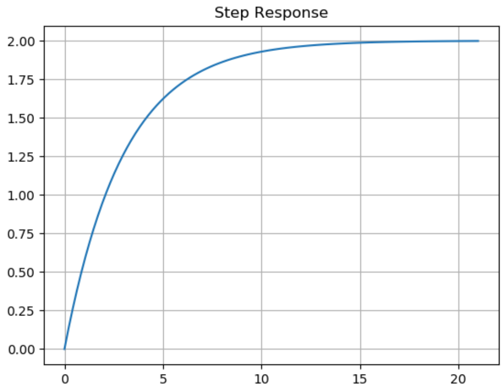
Python

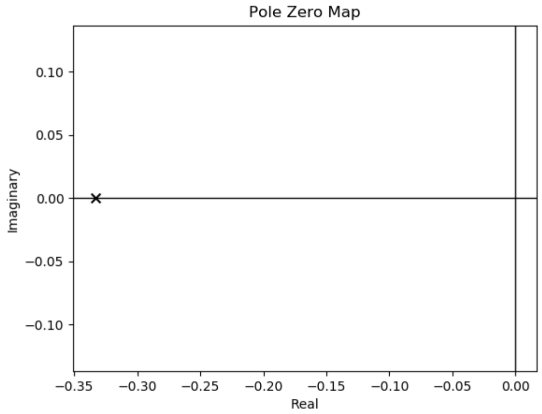
Transfer Function:

Asymptotically Stable System

$$H(s) = \frac{y(s)}{u(s)} = \frac{2}{3s+1}$$

```
p = [-0.33333333]
```

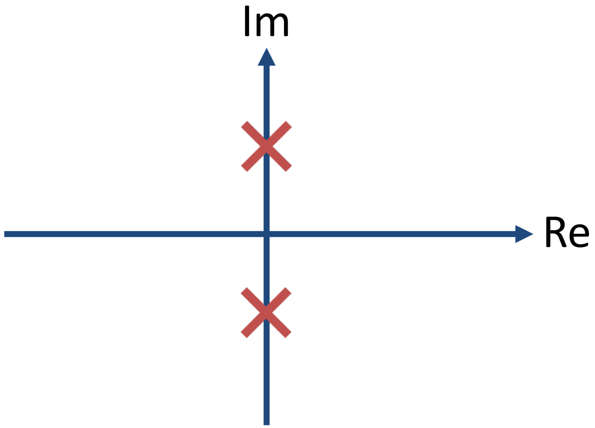




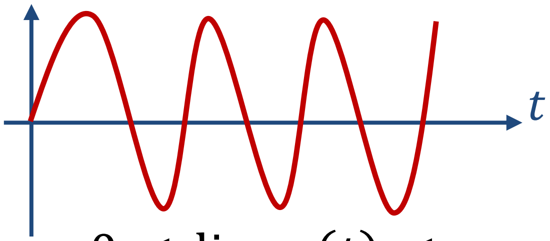
$$\lim_{t\to\infty} y(t) = k$$

```
import control
import numpy as np
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([2])
den = np.array([3 , 1])

H = control.tf(num , den)
print ('H(s) =', H)

# Poles
p = control.pole(H)
print ('p =', p)

# Step Response
t, y = control.step_response(H)

plt.plot(t,y)
plt.title("Step Response")
plt.grid()

control.pzmap(H)
```

Marginally Stable System



One or more poles lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.



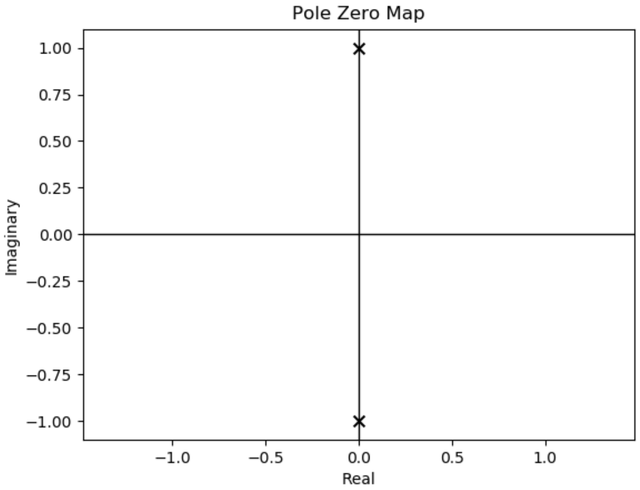
$$0 < \lim_{t \to \infty} y(t) < \infty$$

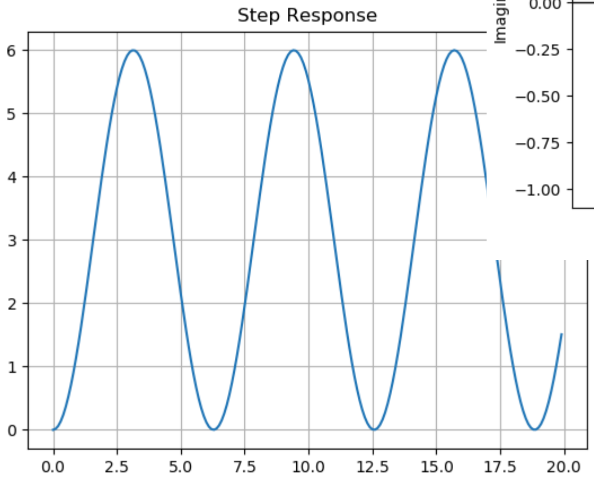
Python

Transfer Function:

Marginally Stable System

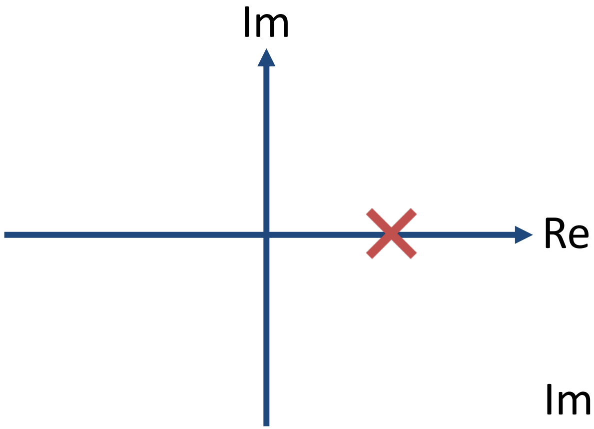
$$H(s) = \frac{3}{s^2 + 1}$$



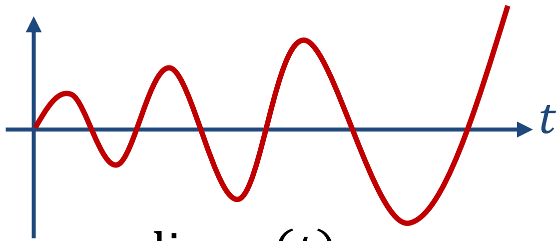



```python
import control
import numpy as np
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([3])
den = np.array([1, 0, 1])

H = control.tf(num , den)
print ('H(s) =', H)

# Poles
p = control.pole(H)
print ('p =', p)

# Step Response
tstart = 0; tstop = 20; tstep = 0.1
t = np.arange(tstart, tstop, tstep)

t, y = control.step_response(H, t)

plt.plot(t,y)
plt.title("Step Response")
plt.grid()

control.pzmap(H)
```

Unstable System

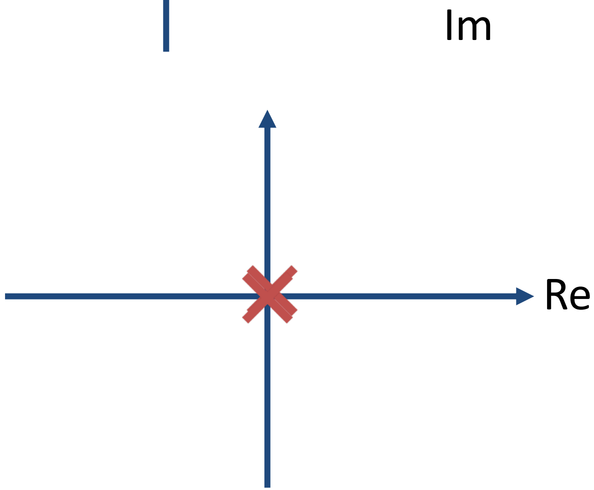


At least one pole lies in the right half plane (has real part greater than zero).



$$\lim_{t\to\infty} y(t) = \infty$$

or



Or: There are multiple and coincident poles on the imaginary axis.

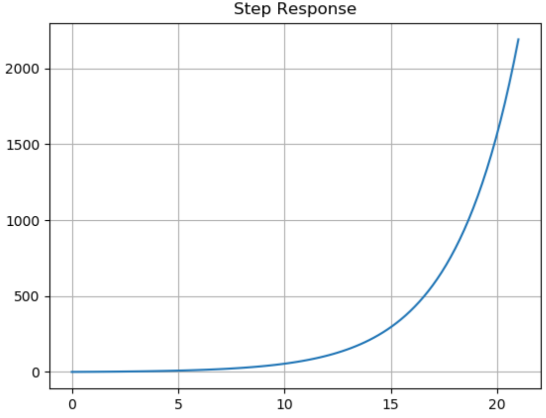
Example: double integrator $H(s) = \frac{1}{s^2}$

Python

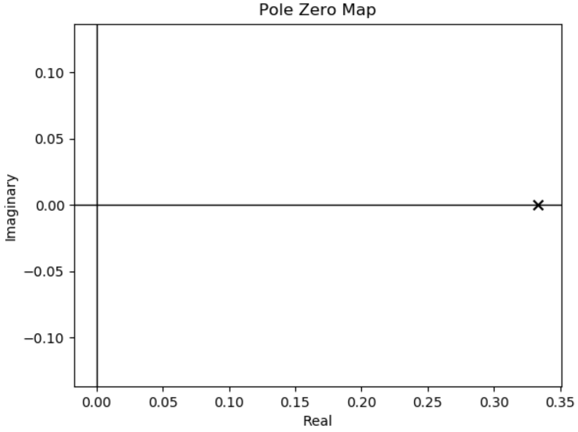
Transfer Function:

Unstable System

$$H(s) = \frac{2}{3s - 1}$$



$$\lim_{t \to \infty} y(t) = \infty$$



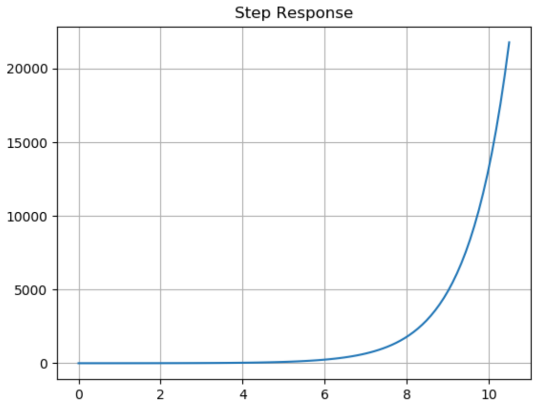

```python
import control
import numpy as np
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([2])
den = np.array([3 , -1])

H = control.tf(num , den)
print ('H(s) =', H)

# Poles
p = control.pole(H)
print ('p =', p)

# Step Response
t, y = control.step_response(H)

plt.plot(t,y)
plt.title("Step Response")
plt.grid()

control.pzmap(H)
```

Python

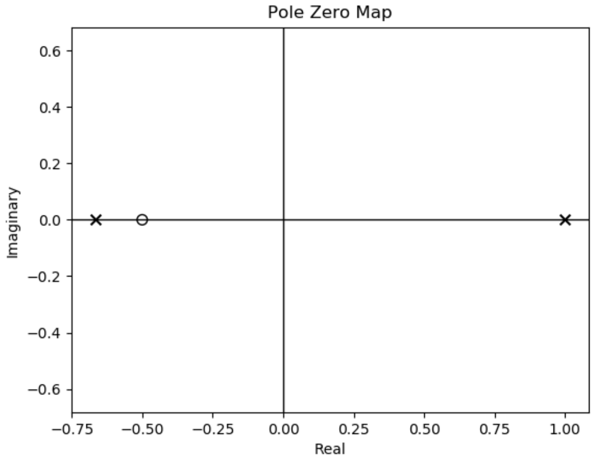
Transfer Function:

$$H(s) = \frac{2s + 1}{3s^2 - s - 2}$$

Unstable System



$$\lim_{t \to \infty} y(t) = \infty$$




```python
import control
import numpy as np
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([2, 1])
den = np.array([3 , -1, -2])

H = control.tf(num , den)
print ('H(s) =', H)

# Poles
p = control.pole(H)
print ('p =', p)

# Step Response
t, y = control.step_response(H)

plt.plot(t,y)
plt.title("Step Response")
plt.grid()

control.pzmap(H)
```

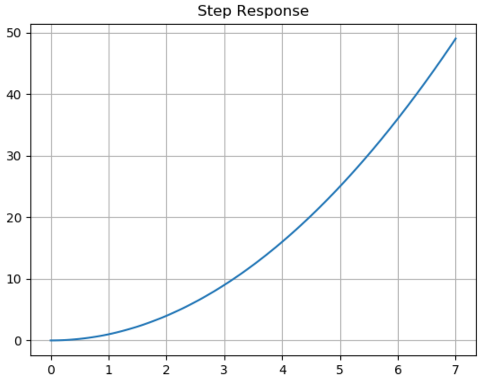
Python

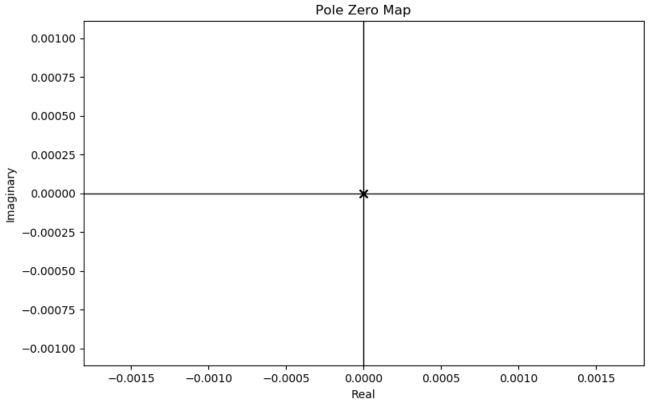
Transfer Function:

Unstable System

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

(This is a double Integrator)

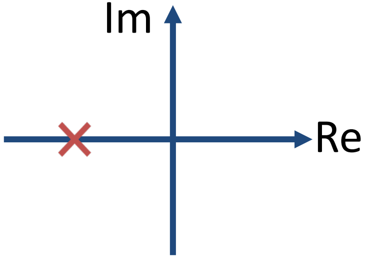
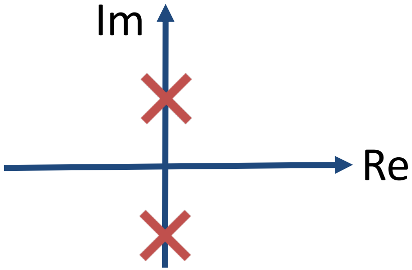
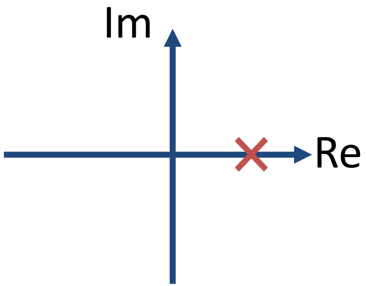
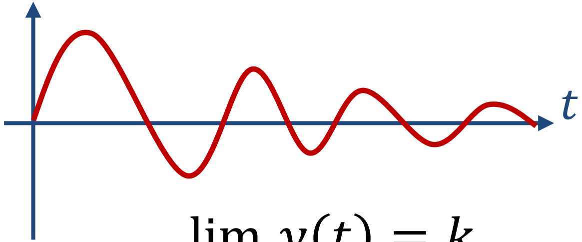
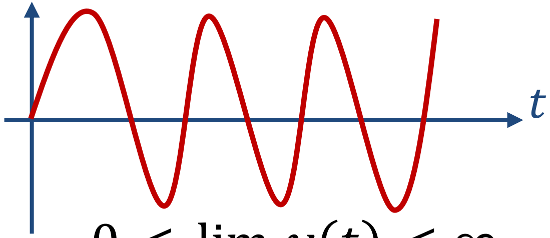
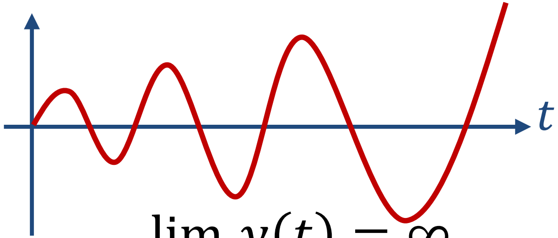




$$p_1 = 0, p_2 = 0$$

```
import control
import numpy as np
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([2])
den = np.array([1, 0, 0])

H = control.tf(num , den)
print ('H(s) =', H)

# Poles
p = control.pole(H)
print ('p =', p)

# Step Response
t, y = control.step_response(H)

plt.plot(t,y)
plt.title("Step Response")
plt.grid()

control.pzmap(H)
```

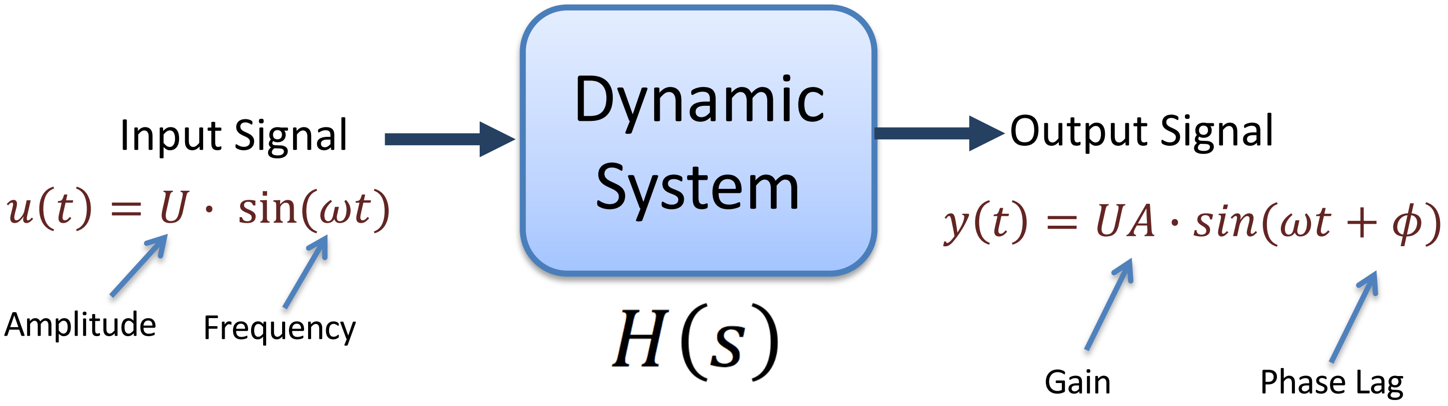
Poles and Stability

	Asymptotically Stable System	Marginally Stable System	Unstable System
Poles:			
Step Response:	 $\lim_{t\to\infty} y(t) = k$	 $0 < \lim_{t\to\infty} y(t) < \infty$	 $\lim_{t\to\infty} y(t) = \infty$

Frequency Response

Hans-Petter Halvorsen

Frequency Response



The frequency response of a system expresses how a **sinusoidal** signal of a given frequency on the system input is transferred through the system. <u>The only difference in the signal is the **gain** and the **phase lag**.</u>

Frequency Response

For a given Transfer Function:

$$H(s) = \frac{y(s)}{u(s)}$$

We have that:

$$H(j\omega) = |H(j\omega)|e^{j\angle H(j\omega)}$$

Where $H(j\omega)$ is the frequency response of the system, i.e., we may find the frequency response by setting $s = j\omega$ in the transfer function. Bode diagrams are useful in frequency response analysis. The Bode diagram consists of 2 diagrams, the Bode magnitude diagram, $A(\omega)$ and the Bode phase diagram, $\phi(\omega)$.

The **Gain** (Magnitude) function:

$$A(\omega) = |H(j\omega)|$$

The **Phase** function:

$$\phi(\omega) = \angle H(j\omega)$$

Bode Diagram

- The Bode diagram gives a simple Graphical overview of the Frequency Response for a given system.
- The Bode Diagram is tool for Analyzing the Stability properties of the Control System.
- You can find the Bode diagram from experiments on the physical process or from the transfer function (the model of the system). We will use the Transfer Function

Bode Diagram Explained

Below we see a Bode Diagram for a given Transfer Function

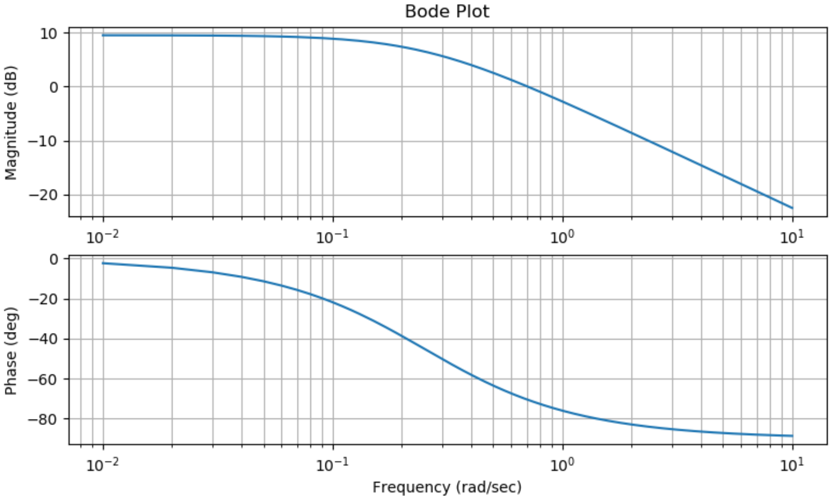
The y-scale is in $[dB]$

Magnitude/ Gain Plot

$$A(\omega) = |H(j\omega)|$$

Phase Plot

$$\phi(\omega) = \angle H(j\omega)$$



Normally, the unit for frequency is Hertz [Hz], but in frequency response and Bode diagrams we use radians ω $[rad/s]$.

The y-scale is in $[degrees]$

The x-scale is in radians ω $[rad/s]$

The x-scale is logarithmic

Conversion Formulas

The y-scale is in $[dB]$. So we typically need to use the following formula:

$$\boxed{x\ [dB] = 20log_{10}x}$$

The y-scale is in $[degrees]$

We know that the relationship between radians and degrees are:

$$\boxed{2\pi\ rad = 360°}$$

The x-scale should be in radians $\omega\ [rad/s]$

The relationship between the frequency f in $Hertz\ [Hz]$ and the frequency ω in radians $[rad/s]$ is:

$$\boxed{\omega = 2\pi f}$$

This gives the following conversion formulas:

$$\boxed{d[degrees] = r[radians] \cdot \frac{180}{\pi}}$$

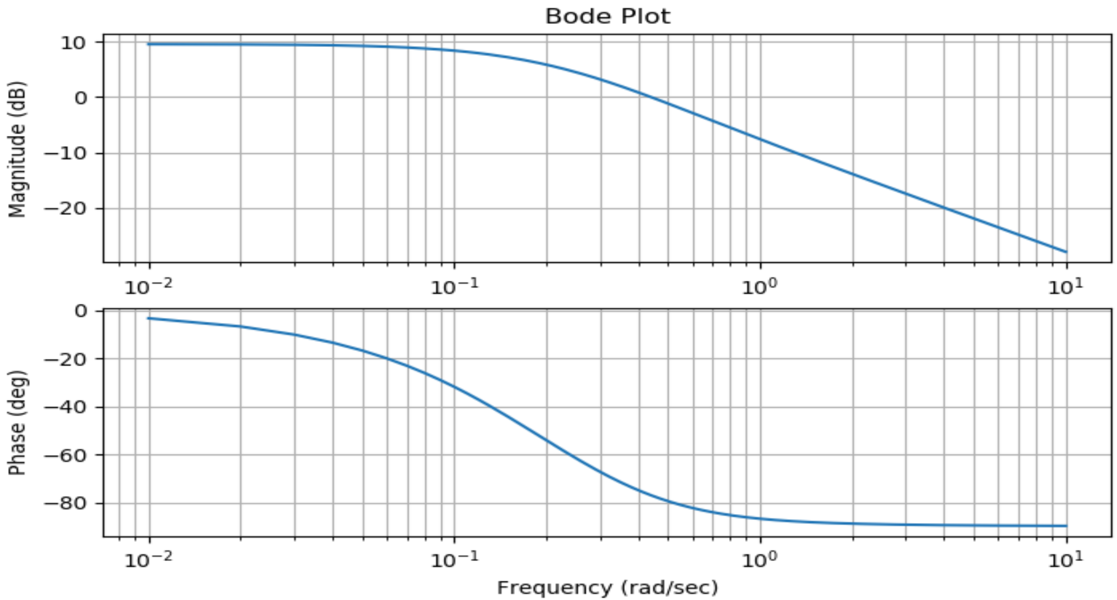
$$\boxed{r[radians] = d[degrees] \cdot \frac{\pi}{180}}$$

Python

SciPy.signal

Transfer Function Example:

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



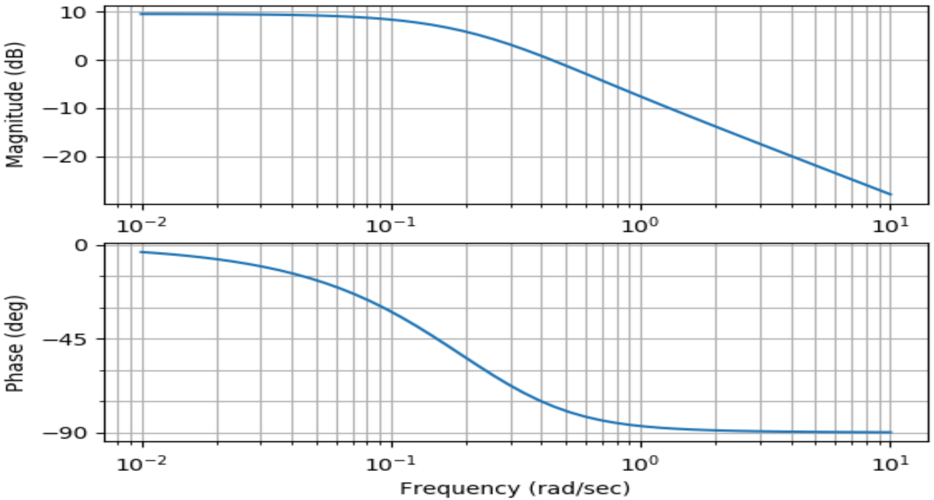

```python
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define Transfer Function
num1 = np.array([3])
num2 = np.array([2, 1])
num = np.convolve(num1, num2)

den1 = np.array([3, 1])
den2 = np.array([5, 1])
den = np.convolve(den1, den2)

H = signal.TransferFunction(num, den)
print ('H(s) =', H)

# Frequencies
w_start = 0.01
w_stop = 10
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

# Bode Plot
w, mag, phase = signal.bode(H, w)

plt.figure()
plt.subplot (2, 1, 1)
plt.semilogx(w, mag)    # Bode Magnitude Plot
plt.title("Bode Plot")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")

plt.subplot (2, 1, 2)
plt.semilogx(w, phase)  # Bode Phase plot
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Phase (deg)")
plt.xlabel("Frequency (rad/sec)")
plt.show()
```

Python

Python Control Systems Library

Transfer Function Example:

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



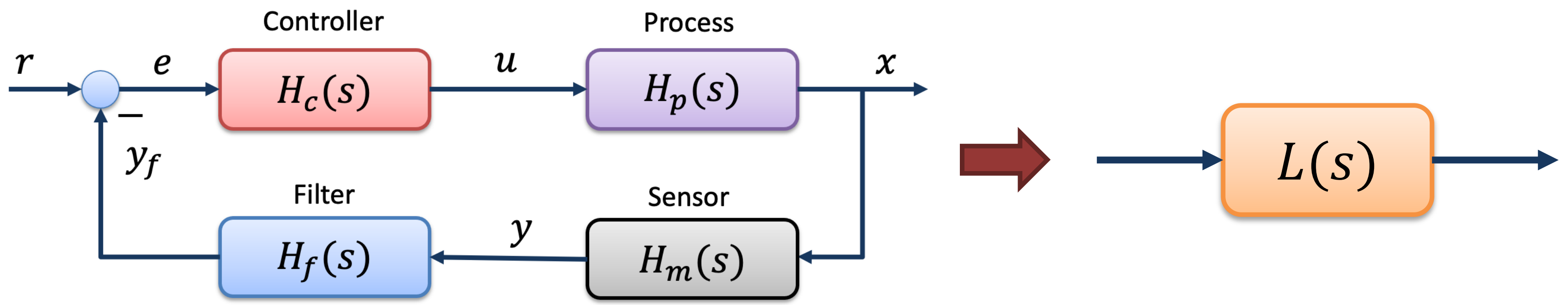

```
import numpy as np
import control

# Define Transfer Function
num1 = np.array([3])
num2 = np.array([2, 1])
num = np.convolve(num1, num2)

den1 = np.array([3, 1])
den2 = np.array([5, 1])
den = np.convolve(den1, den2)

H = control.tf(num, den)
print ('H(s) =', H)

# Bode Plot
control.bode(H, dB=True)
```

Frequency Response Stability Analysis

We use the Loop Transfer Function in Frequency Response Stability Analysis of a Control System

The Loop Transfer Function $L(s)$ is defined as follows:

$$\boxed{L(s) = H_c(s)H_p(s)H_m(s)H_f(s) \cdots}$$



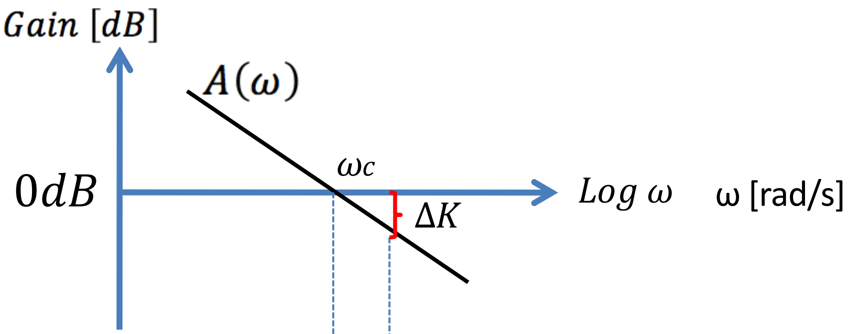
The Loop Transfer Function is the product of all the transfer functions in the loop

Bode and Stability Properties

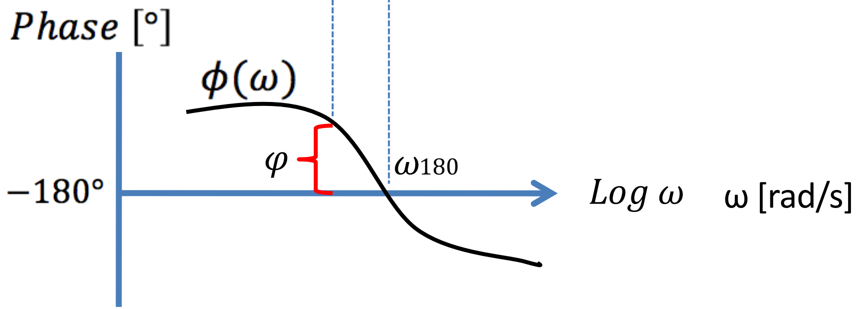
We use the Loop Transfer Function $L(s)$ as basis for the Bode Diagram

$L(j\omega)$

$A(\omega) = |L(j\omega)|$



$\phi(\omega) = \angle L(j\omega)$



- The Bode diagram gives a simple Graphical overview of the Frequency Response for a given system.
- A Tool for Analyzing the Stability properties of the Control System.
- With Python you can easily create Bode diagrams from the Transfer function model using the bode() function

Frequency Response Stability Analysis

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

Gain Margin - GM (ΔK) and Phase Margin – PM (ϕ) are important design criteria for analysis of feedback control systems.

- **The Gain Margin – GM** (ΔK) is how much the loop gain can increase before the system become unstable.
- The **Phase Margin - PM** (ϕ) is how much the phase lag function of the loop can be reduced before the loop becomes unstable.

Crossover Frequencies

ω_{180} (gain margin frequency - gmf) is the gain margin frequency/frequencies, in radians/second. A gain margin frequency indicates where the model phase crosses -180 degrees.
ω_c (phase margin frequency - pmf) returns the phase margin frequency/frequencies, in radians/second. A phase margin frequency indicates where the model magnitude crosses 0 decibels.

Gain Crossover-frequency - ω_c Definition:

$$|L(j\omega_c)| = 1 = 0dB$$

Phase Crossover-frequency - ω_{180} Definition:

$$\angle L(j\omega_{180}) = -180^o$$

Note! Both ω_{180} and ω_c are called the crossover-frequencies

Gain Margin and Phase Margin

Gain Margin - GM (ΔK) and Phase Margin – PM (ϕ) are important design criteria for analysis of feedback control systems.

The Gain Margin – GM (ΔK) is how much the loop gain can increase before the system become unstable.
Definition:

$$GM = \frac{1}{|L(j\omega_{180})|}$$

or:

$$GM\ [dB] = -|L(j\omega_{180})|\ [dB]$$

The Phase Margin - PM (ϕ) is how much the phase lag function of the loop can be reduced before the loop becomes unstable.
Definition:

$$PM = 180^o + \angle L(j\omega_c)$$

Frequency Response Stability Analysis

We have the following:

- Asymptotically stable system: $\omega_c < \omega_{180}$
- Marginally stable system: $\omega_c = \omega_{180}$
- Unstable system: $\omega_c > \omega_{180}$

The Tracking Property is <u>good</u> if:

$$|L(j\omega)| \gg 1\ (0\ dB)$$

The Tracking Property is <u>poor</u> if:

$$|L(j\omega)| \ll 1(0\ dB)$$

Python

Assume the following Loop Transfer Function:

$$L(s) = \frac{0.1}{s(3s+1)(5s+1)}$$

$$= \frac{0.1}{15s^3 + 8s^2 + s}$$

```
import numpy as np
import control

# Define Transfer Function
num = np.array([0.1])


den1 = np.array([1, 0])
den2 = np.array([3, 1])
den3 = np.array([5, 1])
den = np.convolve(den1, np.convolve(den2, den3))

L = control.tf(num, den)
print ('HL(s) =', L)
```

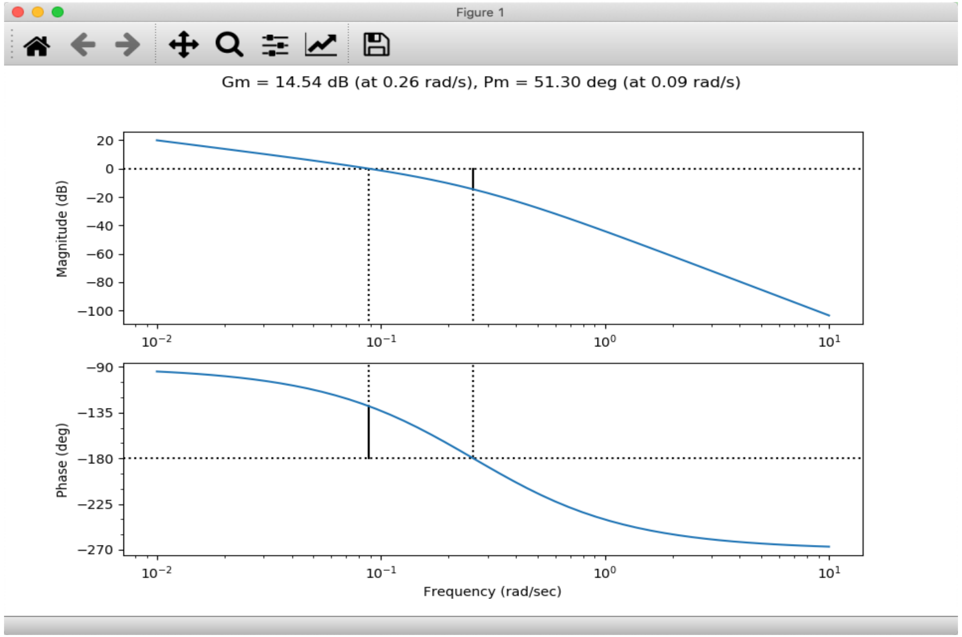
Or:

```
import numpy as np
import control

# Define Transfer Function
num = np.array([0.1])
den = np.array([15, 8, 1, 0])

L = control.tf(num, den)
print ('L(s) =', L)
```

Python

Loop Transfer Function:

$$L(s) = \frac{0.1}{15s^3 + 8s^2 + s}$$

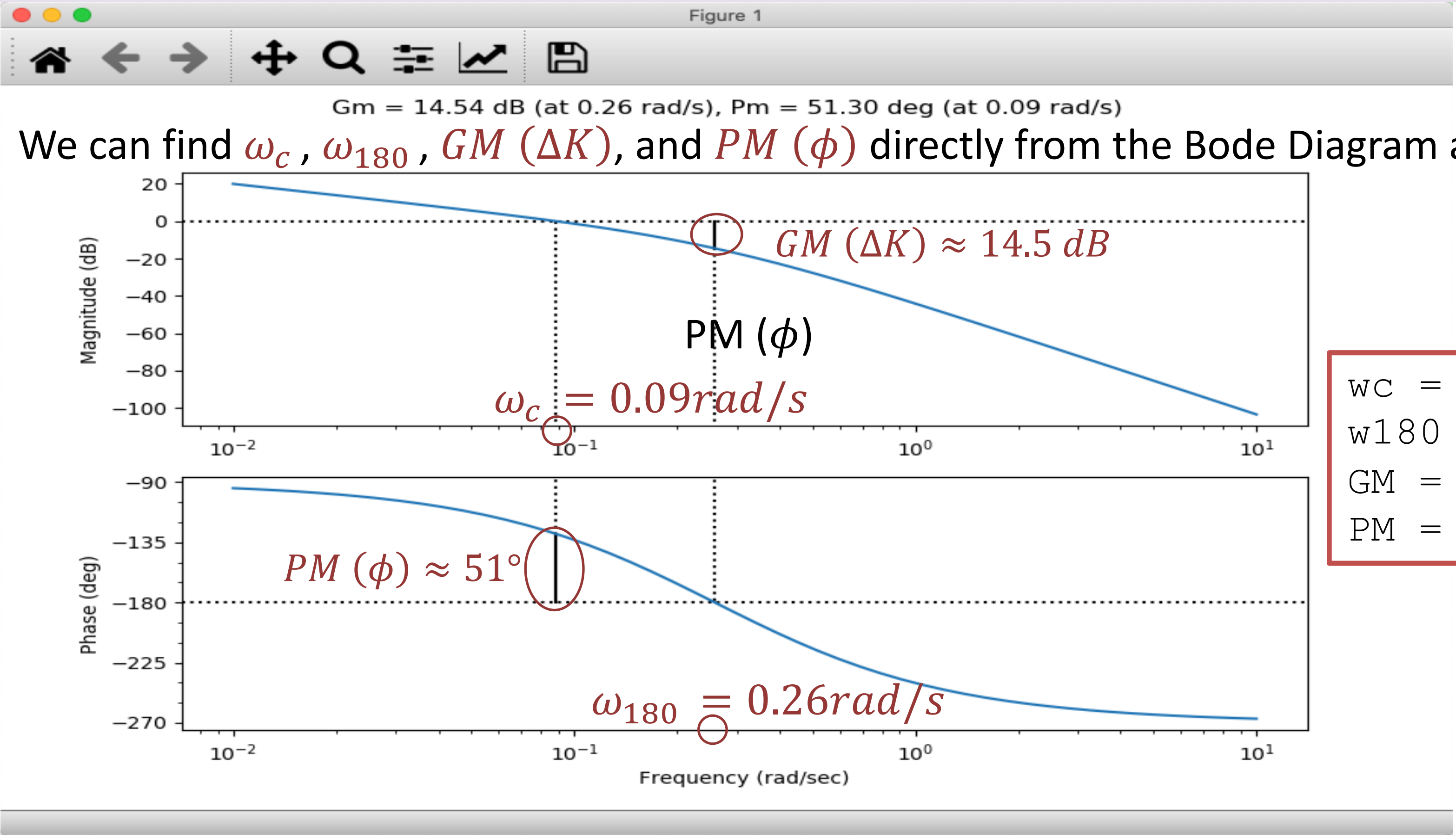



```
import numpy as np
import control

# Define Transfer Function
num = np.array([0.1])
den = np.array([15, 8, 1, 0])

L= control.tf(num, den)
print ('L(s) =', L)

control.bode(L, dB=True, deg=True, margins=True)

# Stability margins and crossover frequencies
gm , pm , w180 , wc = control.margin(L)

# Convert gm to Decibel
gmdb = 20 * np.log10(gm)

print("wc =", f'{wc:.2f}', "rad/s")
print("w180 =", f'{w180:.2f}', "rad/s")
print("GM =", f'{gm:.2f}')
print("GM =", f'{gmdb:.2f}', "dB")
print("PM =", f'{pm:.2f}', "deg")
```

```
wc = 0.09 rad/s
w180 = 0.26 rad/s
GM = 14.54 dB
PM = 51.30 deg
```

Bode Plot

We can find ω_c , ω_{180} , $GM\ (\Delta K)$, and $PM\ (\phi)$ directly from the Bode Diagram as shown




```
wc = 0.09 rad/s
w180 = 0.26 rad/s
GM = 14.54 dB
PM = 51.30 deg
```

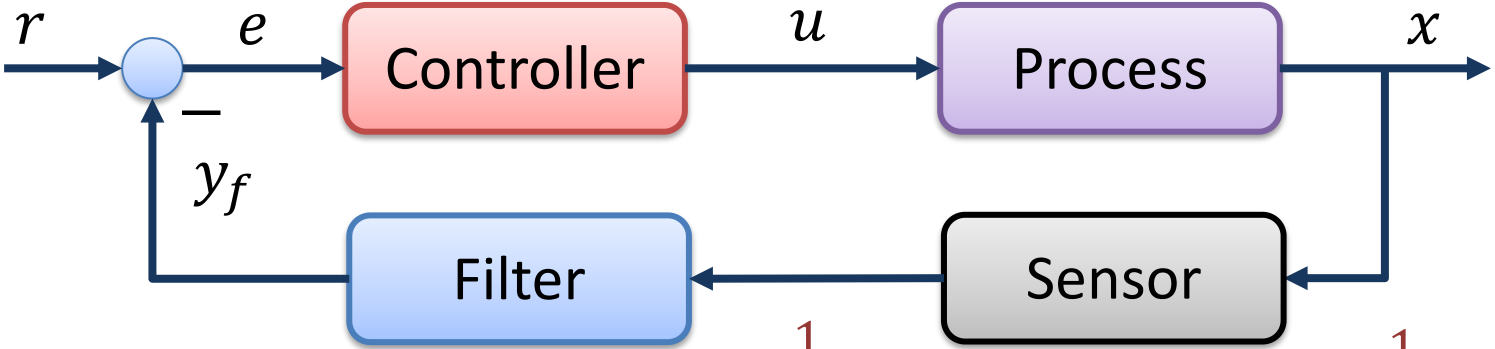
Python Example

Hans-Petter Halvorsen

Stability Analysis Example

$$H_c(s) = \frac{K_p(T_i s + 1)}{T_i s}$$

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



$$H_f(s) = \frac{1}{T_f s + 1}$$

$$H_m(s) = \frac{1}{T_m s + 1}$$

In Stability Analysis we use the following Transfer Functions:

Loop Transfer Function: $\boxed{L(s) = H_c(s)H_p(s)H_m(s)H_f(s)}$

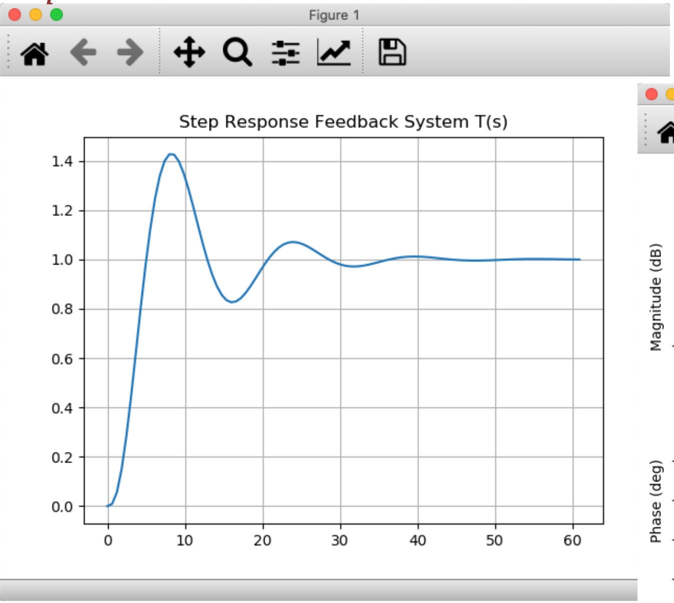
Tracking Transfer Function: $\boxed{T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1+L(s)}}$

```python
import numpy as np
import matplotlib.pyplot as plt
import control

# Transfer Function Process
K = 3; T = 4
num_p = np.array ([K])
den_p = np.array ([T , 1])
Hp = control.tf(num_p , den_p)
print ('Hp(s) =', Hp)

# Transfer Function PI Controller
Kp = 0.4
Ti = 2
num_c = np.array ([Kp*Ti, Kp])
den_c = np.array ([Ti , 0])
Hc = control.tf(num_c, den_c)
print ('Hc(s) =', Hc)

# Transfer Function Measurement
Tm = 1
num_m = np.array ([1])
den_m = np.array ([Tm , 1])
Hm = control.tf(num_m , den_m)
print ('Hm(s) =', Hm)

# Transfer Function Lowpass Filter
Tf = 1
num_f = np.array ([1])
den_f = np.array ([Tf , 1])
Hf = control.tf(num_f , den_f)
print ('Hf(s) =', Hf)

# The Loop Transfer function
L = control.series(Hc, Hp, Hf, Hm)
print ('L(s) =', L)
```

```python
# Tracking transfer function
T = control.feedback(L,1)
print ('T(s) =', T)

# Step Response Feedback System (Tracking System)
t, y = control.step_response(T)
plt.figure(1)
plt.plot(t,y)
plt.title("Step Response Feedback System T(s)")
plt.grid()

# Bode Diagram with Stability Margins
plt.figure(2)
control.bode(L, dB=True, deg=True, margins=True)

# Poles and Zeros
control.pzmap(T)
p = control.pole(T)
z = control.zero(T)
print("poles = ", p)

# Calculating stability margins and crossover frequencies
gm , pm , w180 , wc = control.margin(L)

# Convert gm to Decibel
gmdb = 20 * np.log10(gm)

print("wc =", f'{wc:.2f}', "rad/s")
print("w180 =", f'{w180:.2f}', "rad/s")

print("GM =", f'{gm:.2f}')
print("GM =", f'{gmdb:.2f}', "dB")
print("PM =", f'{pm:.2f}', "deg")

# Find when Sysem is Marginally Stable (Kritical Gain -  Kc)
Kc = Kp*gm
print("Kc =", f'{Kc:.2f}')
```

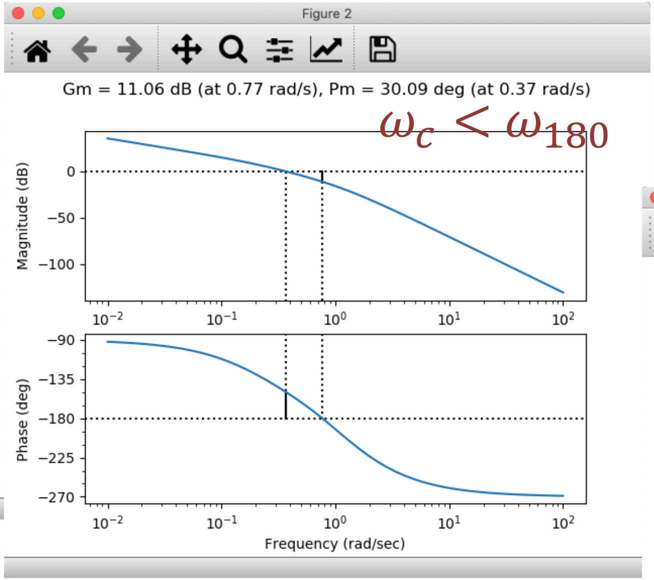
Asymptotically Stable System

$K_p \ = \ 0.4$



Step Response

Frequency Response



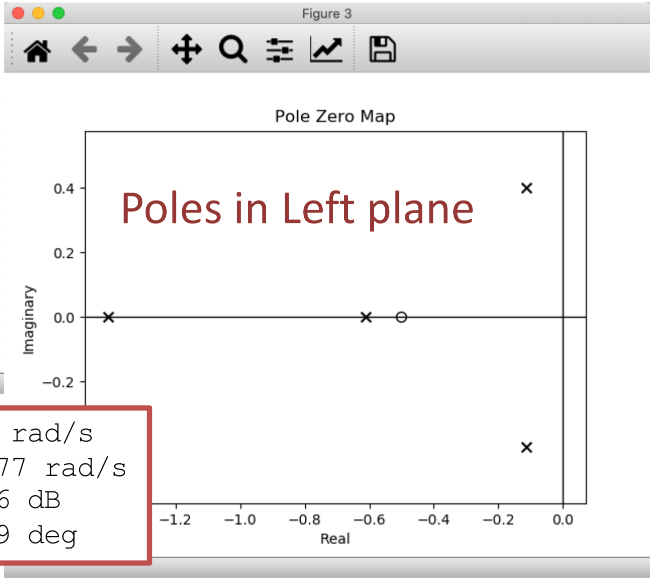
$\omega_c < \omega_{180}$

Gain Margin (GM): $\Delta K \ \approx \ 11.dB$

Phase Margin (PM): $\varphi \approx \ 30°$

This means that we can increase K_p a bit without problem

Poles



Poles in Left plane

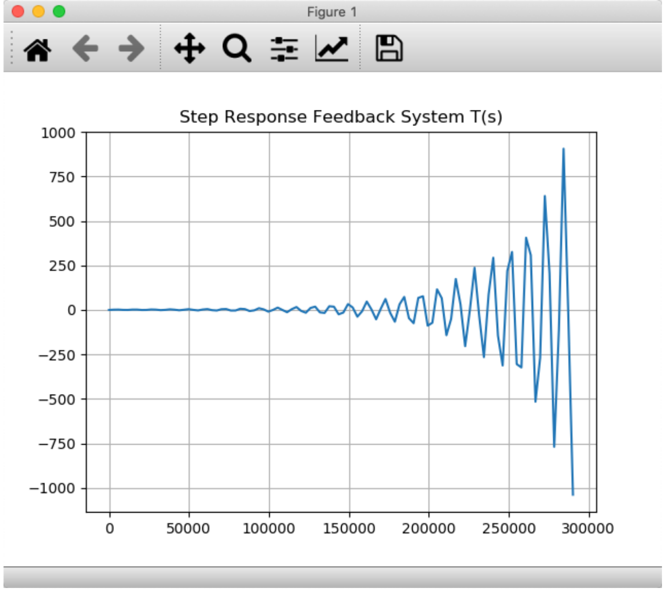
As you see we have an Asymptotically Stable System

The Critical Gain is $K_c \ = K_c \times \Delta K \approx 1.43$

```
wc = 0.37 rad/s
w180 = 0.77 rad/s
GM = 11.06 dB
PM = 30.09 deg
```

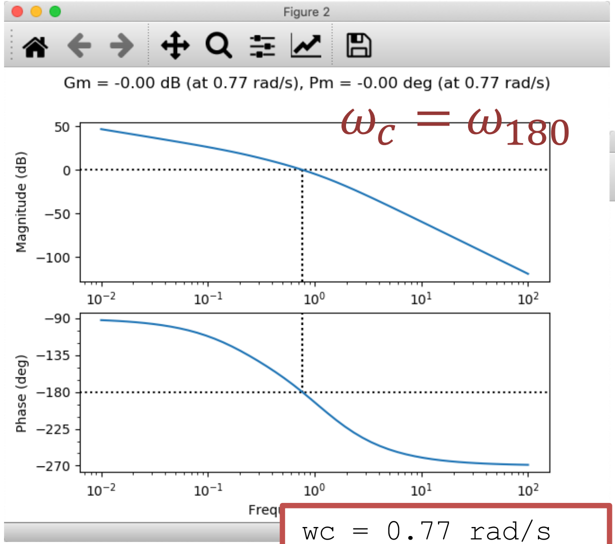
Marginally Stable System

$K_p = 1.43$



Step Response

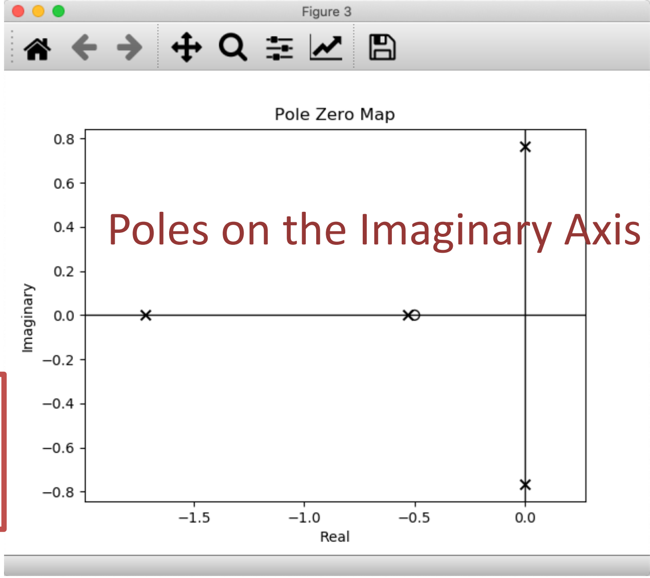
Frequency Response



$\omega_c = \omega_{180}$

```
wc = 0.77 rad/s
w180 = 0.77 rad/s
GM = 0.00 dB
PM = 0.00 deg
```

Poles

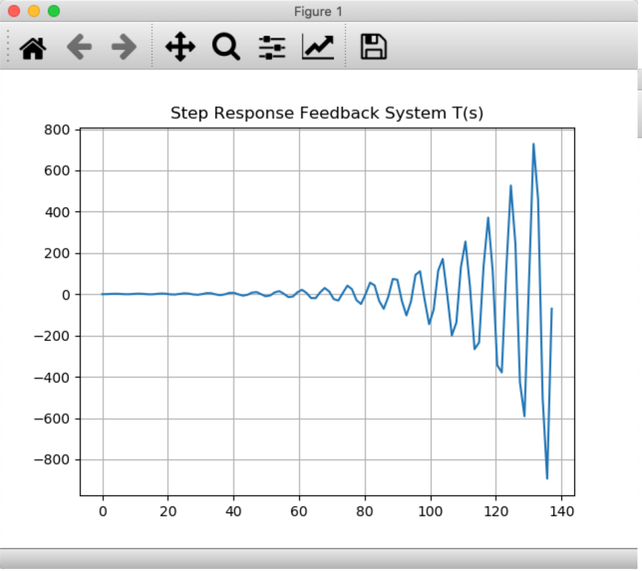


Poles on the Imaginary Axis

As you see we have a Marginally Stable System

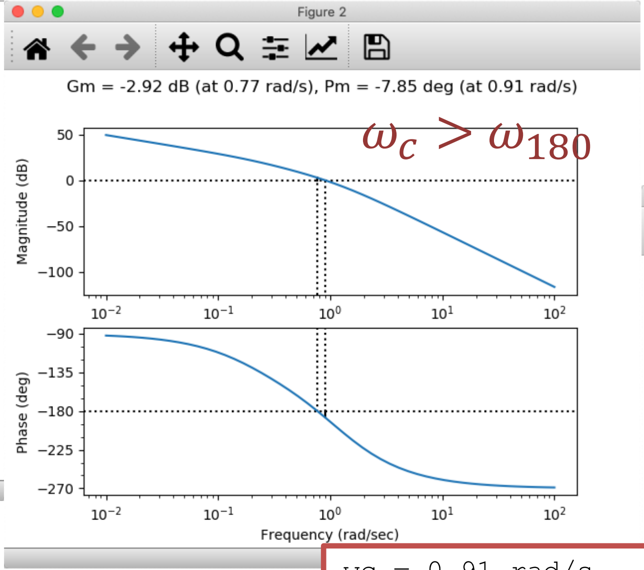
Unstable System

$K_p \; = 2$



Step Response

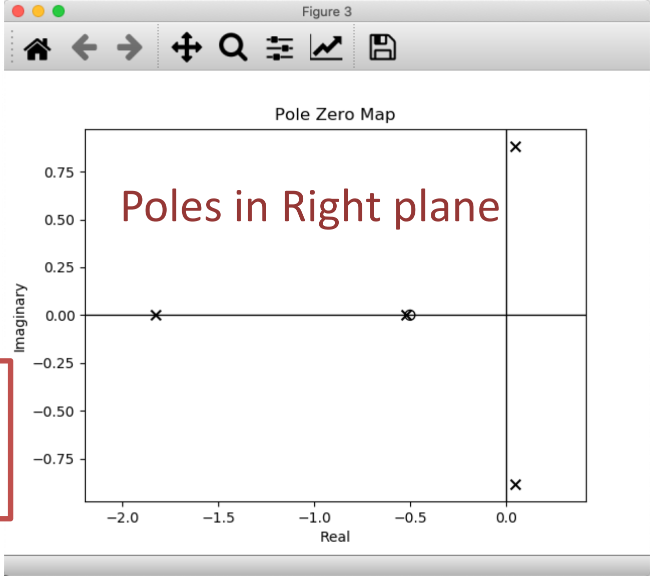
Frequency Response



$\omega_c > \omega_{180}$

```
wc = 0.91 rad/s
w180 = 0.77 rad/s
GM = -2.92 dB
PM = -7.85 deg
```

Poles



Poles in Right plane

As you see we have a Marginally Stable System

Conclusions

We have an Asymptotically Stable System when $K_p < K_c$

- We have Poles in the left half plane
- $\lim_{t \to \infty} y(t) = 1$ (Good Tracking)
- $\omega_c < \omega_{180}$

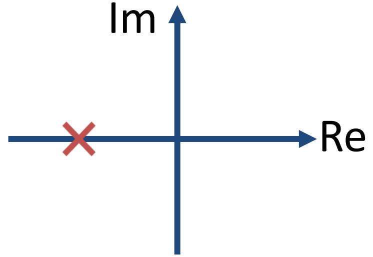
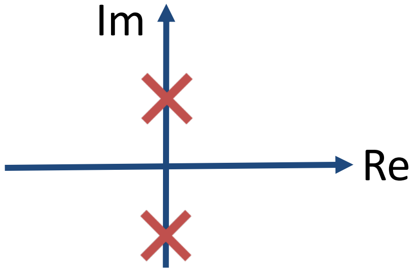
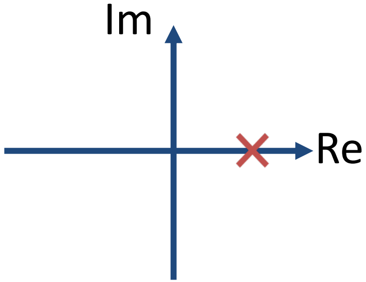
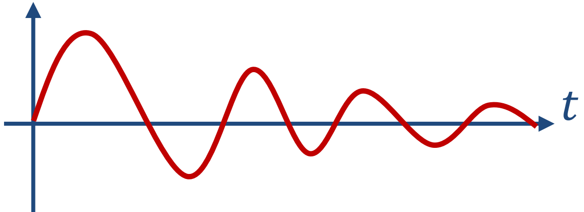
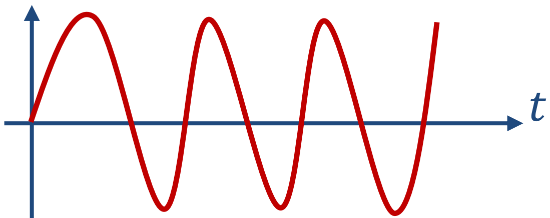
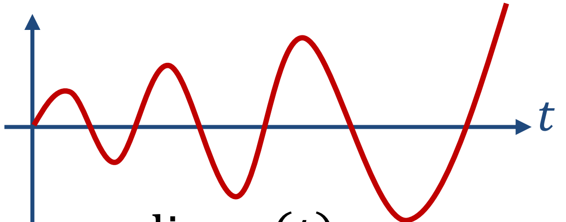
We have a Marginally Stable System when $K_p = K_c$

- We have Poles on the Imaginary Axis
- $0 < \lim_{t \to \infty} y(t) < \infty$
- $\omega_c = \omega_{180}$

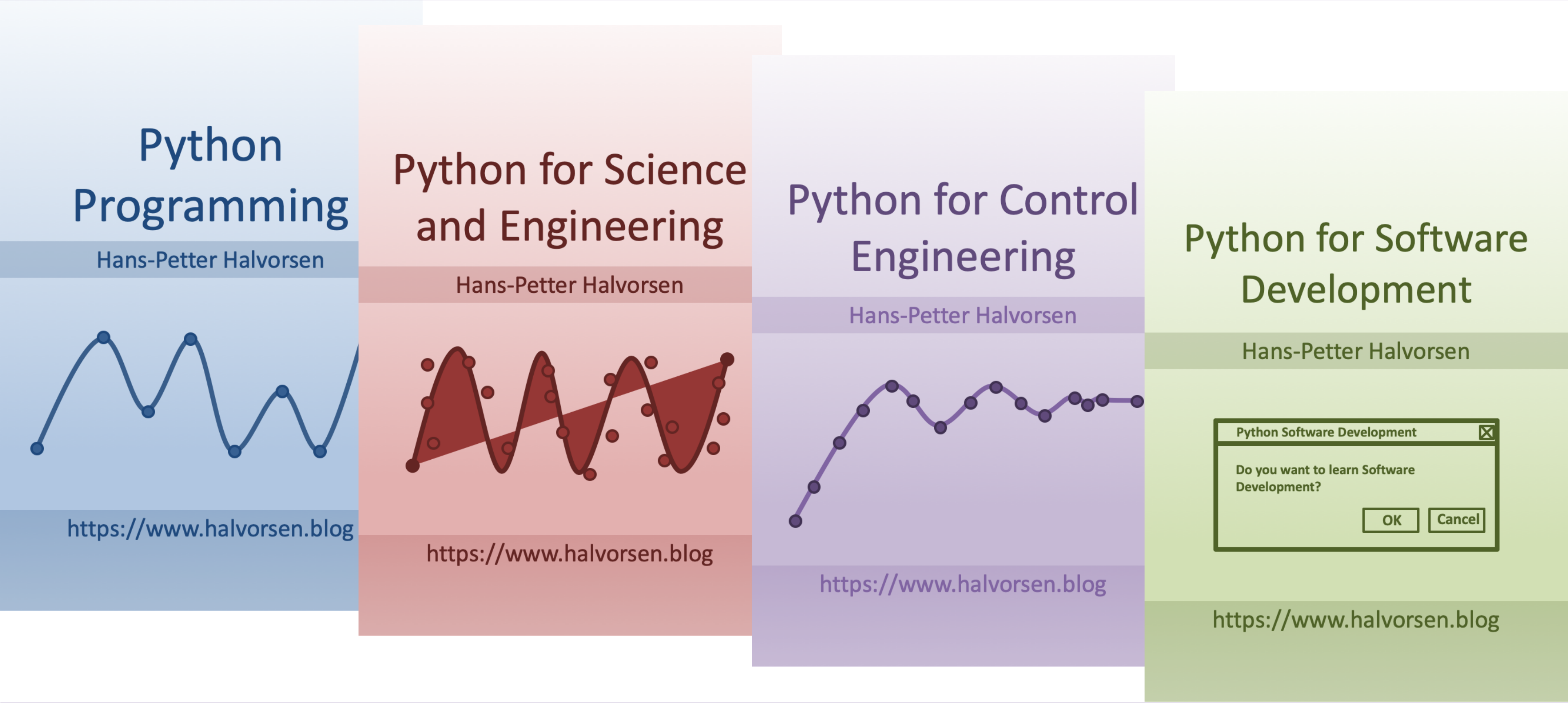
We have an Unstable System when $K_p > K_c$

- We have Poles in the right half plane
- $\lim_{t \to \infty} y(t) = \infty$
- $\omega_c > \omega_{180}$

Stability Analysis Summary

	Asymptotically Stable System	Marginally Stable System	Unstable System
Poles:			
Step Response:	 $\lim_{t\to\infty} y(t) = k$	 $0 < \lim_{t\to\infty} y(t) < \infty$	 $\lim_{t\to\infty} y(t) = \infty$
Frequency Response:	$\omega_c < \omega_{180}$	$\omega_c = \omega_{180}$	$\omega_c > \omega_{180}$

Additional Python Resources



https://www.halvorsen.blog/documents/programming/python/

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog