https://www.halvorsen.blog

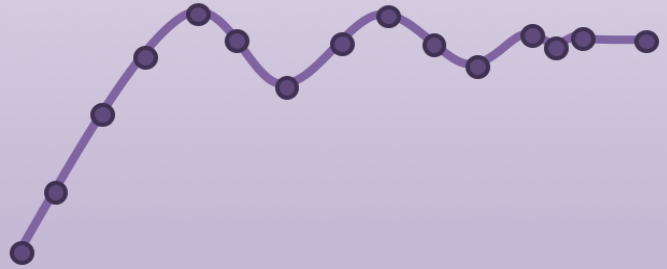
Frequency Response with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Control Engineering

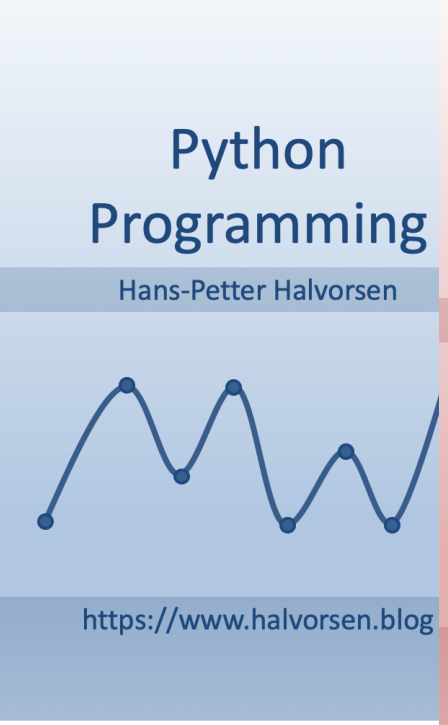
Hans-Petter Halvorsen

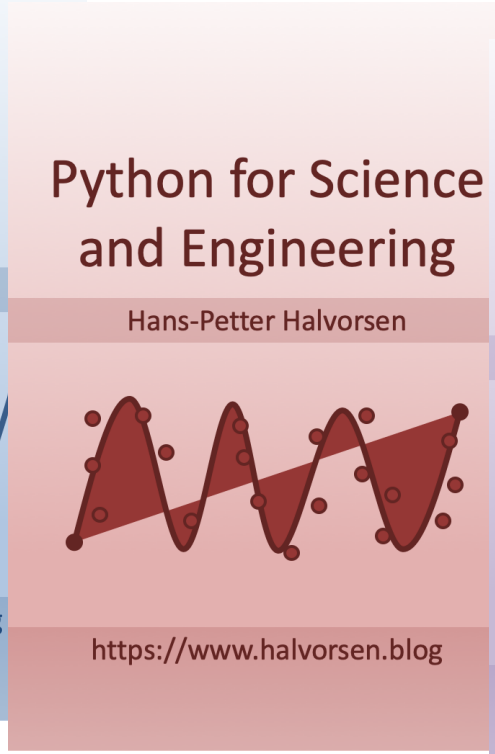


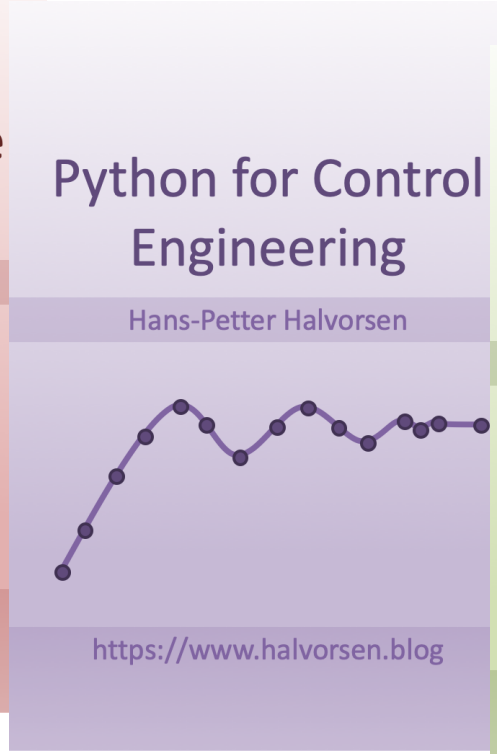
https://www.halvorsen.blog

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

Additional Python Resources







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

Contents

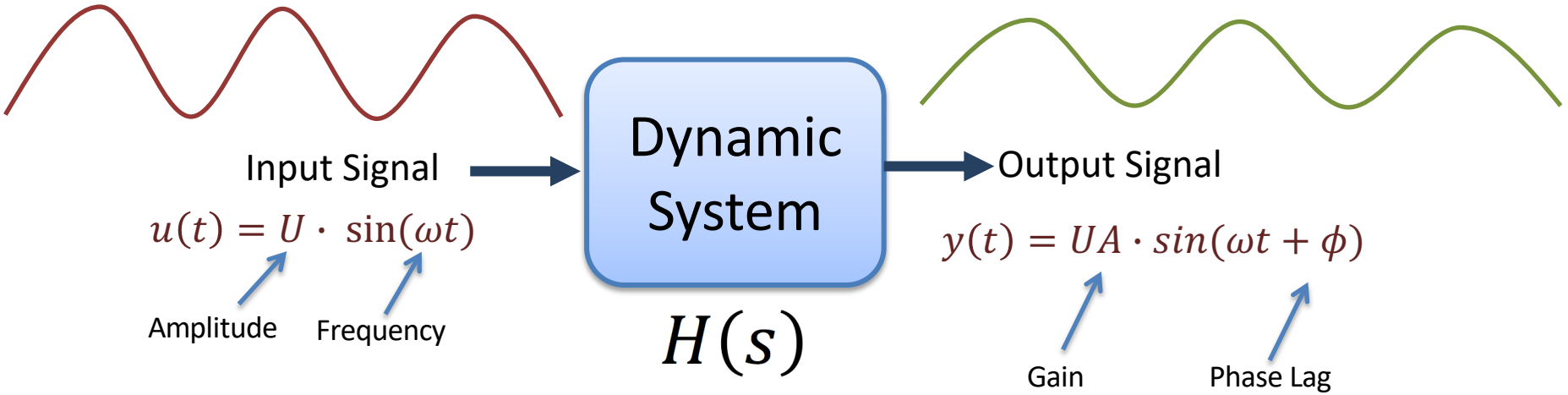
- Introduction to Frequency Response
- Python Examples:
  2 different Python libraries will be used:
  - SciPy (SciPy.Signal)
  - The Python Control Systems Library
- Introduction to Filter Design and Stability Analysis in Control Systems with Examples

It is recommended that you know about Transfer Functions. If not, take a closer look at my Tutorial “Transfer Functions with Python”

Frequency Response

- The frequency response of a system is a frequency dependent function which expresses how a sinusoidal signal of a given frequency on the system input is transferred through the system. Each frequency component is a sinusoidal signal having certain amplitude and a certain frequency.
- The frequency response is an important tool for analysis and design of **signal filters** and for analysis and design of **control systems**.
- The frequency response can be found experimentally or from a transfer function model.
- The frequency response of a system is defined as the steady-state response of the system to a sinusoidal input signal. When the system is in steady-state, it differs from the input signal only in amplitude/gain (A) and phase lag (ϕ).

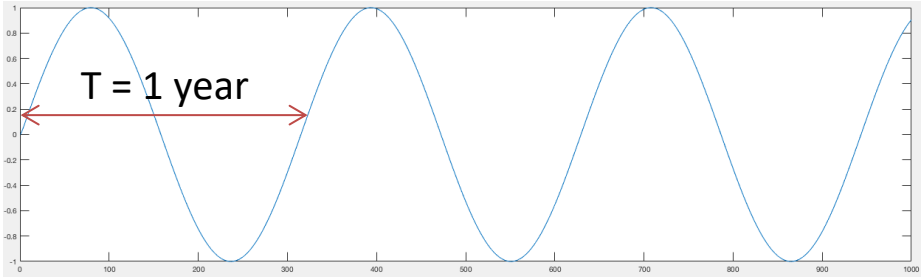
Frequency Response



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

Frequency Response - Example

Outside Temperature

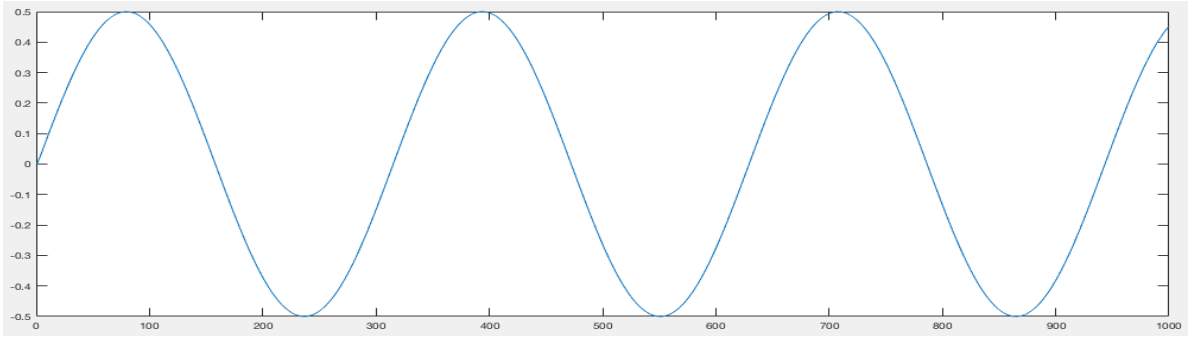


frequency 1 (year)

Dynamic System

frequency 1 (year)

Inside Temperature



Note! Only the gain and phase are different

Assume the outdoor temperature is varying like a sine function during a year (frequency 1) or for 24 hours (frequency 2). Then the indoor temperature will be a sine as well, but with different gain. In addition it will have a phase lag.

Frequency Response

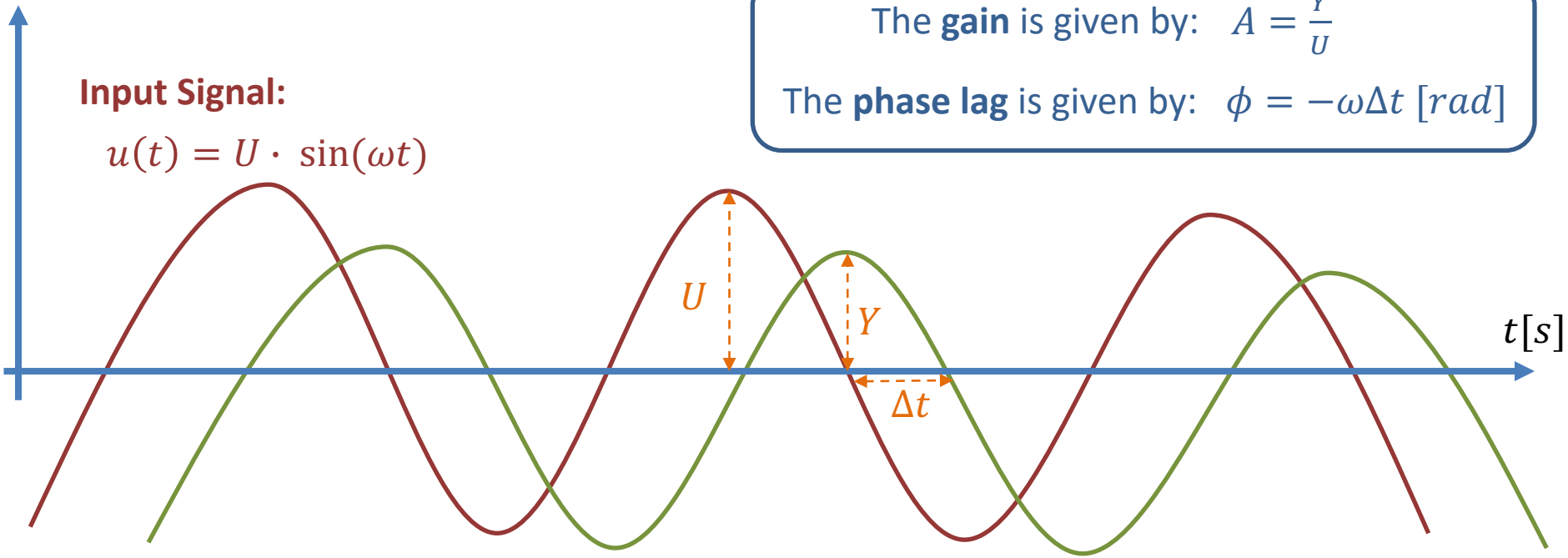
Below we see a plot for the input and output sine signals for a given frequency $\omega[rad/s]$:

The **gain** is given by: $A = \frac{Y}{U}$

The **phase lag** is given by: $\phi = -\omega\Delta t\ [rad]$

Input Signal:

$u(t) = U \cdot \sin(\omega t)$

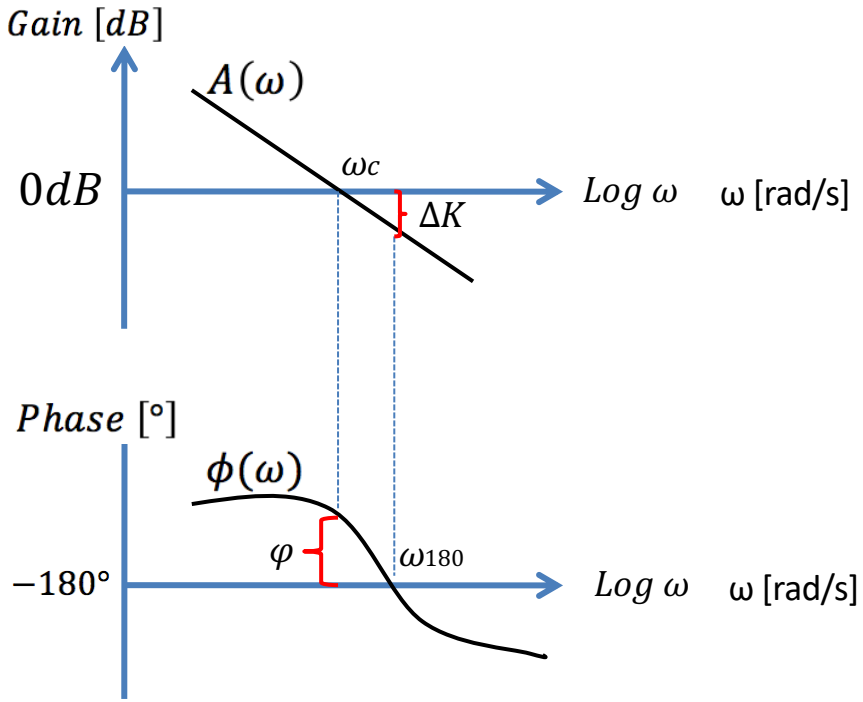


Output Signal: $y(t) = UA \cdot sin(\omega t + \phi) = Y \cdot sin(\omega t + \phi)$

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).

Bode Diagram

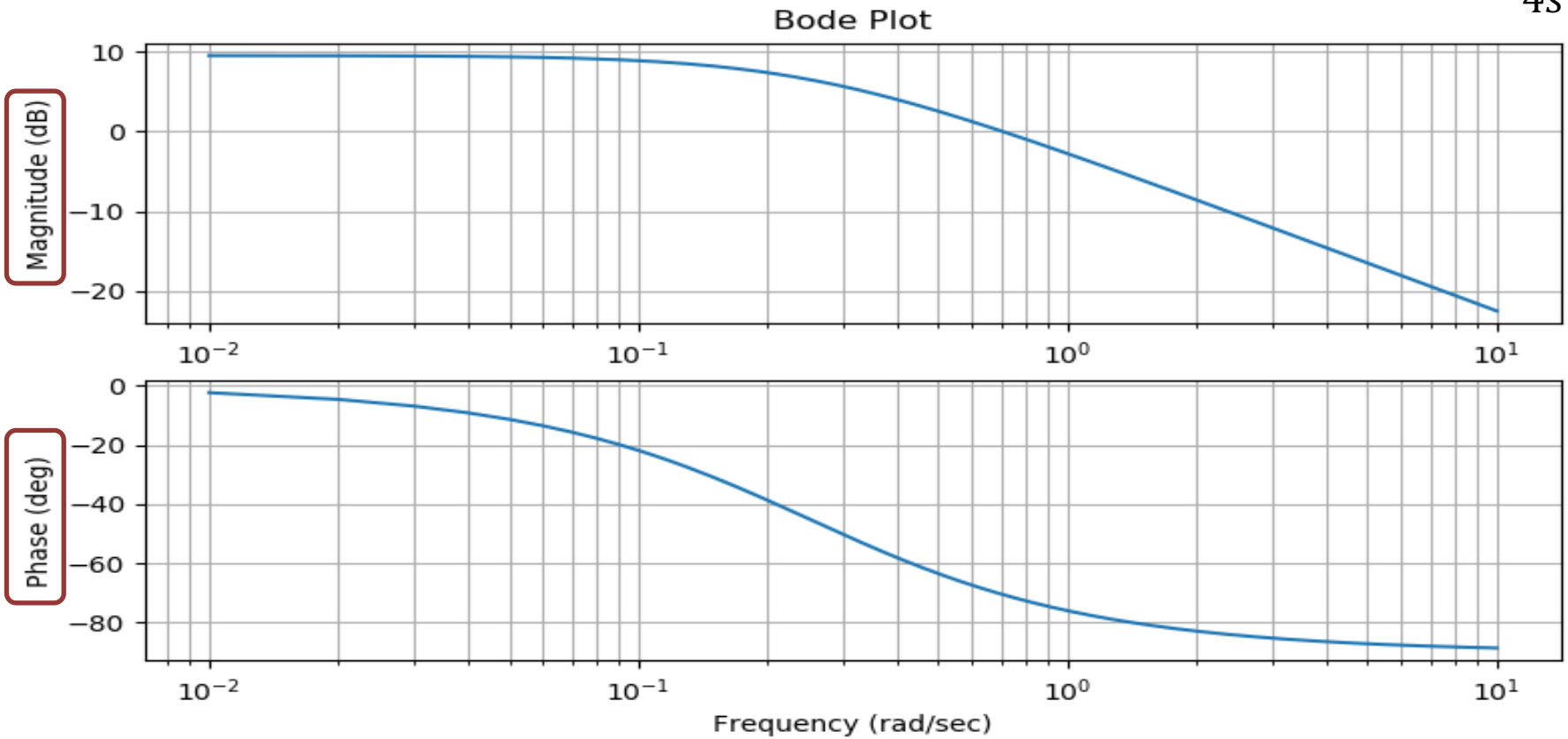


The Bode diagram gives a simple Graphical overview of the Frequency Response for a given system. A Tool for Analyzing the Stability properties of the Control System.

With Python you can easily create Bode diagrams from the Transfer function model using the bode() function

Bode Diagram Example

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



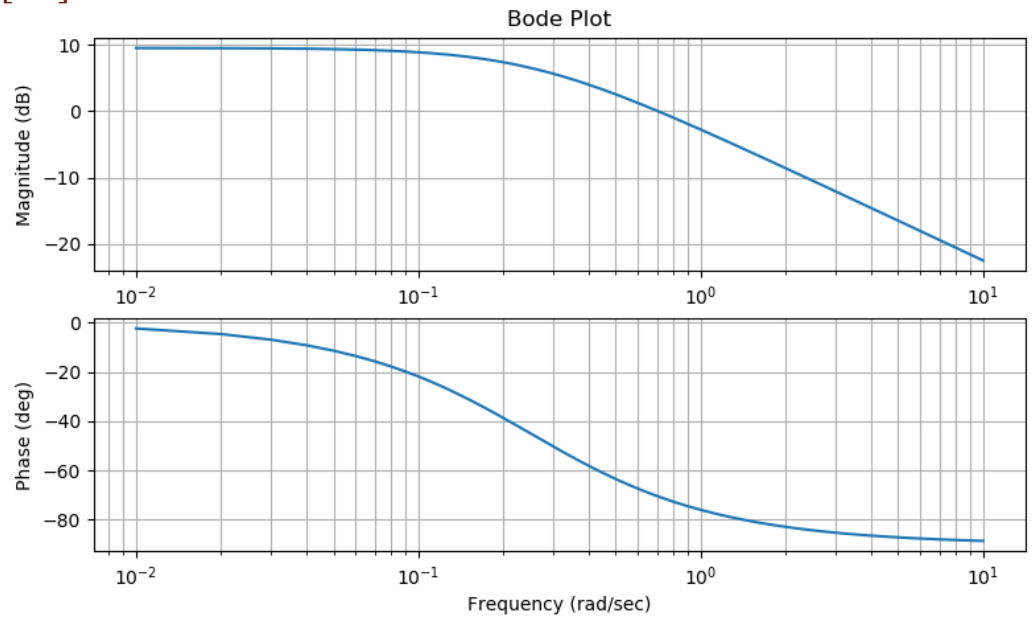
Bode Diagram Explained

The y-scale is in $[dB]$

Magnitude/
Gain Plot

Phase Plot

The y-scale is in $[degrees]$



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

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:

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

The y-scale is in $[degrees]$

We know that the relationship between radians and degrees are:

$$2\pi\,rad = 360°$$

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

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

$$\omega = 2\pi f$$

This gives the following conversion formulas:

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

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

Python Examples

Hans-Petter Halvorsen

Python Examples

2 different Python libraries will be used:

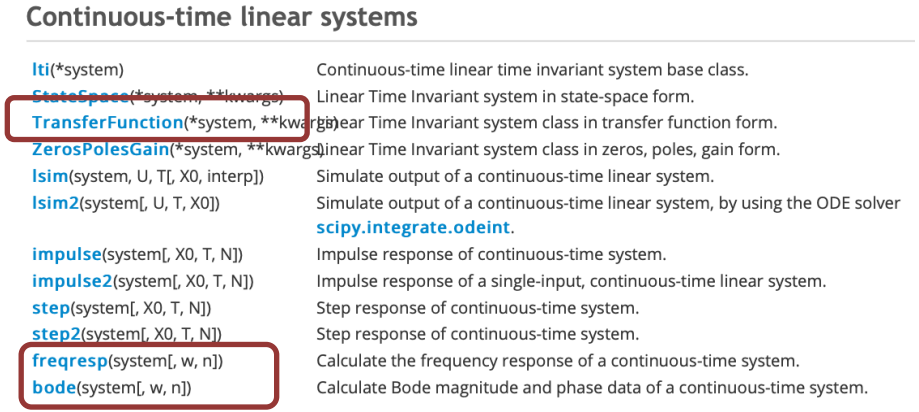
- SciPy (SciPy.Signal)
  - Included with Anaconda Distribution
- The Python Control Systems Library
  - You must install it using PIP

SciPy.signal

Hans-Petter Halvorsen

SciPy.signal

- The SciPy.signal contains Signal Processing functions
- SciPy is also included with the Anaconda distribution
- If you have installed Python using the Anaconda distribution, you don't need to install anything
- https://docs.scipy.org/doc/scipy/reference/signal.html



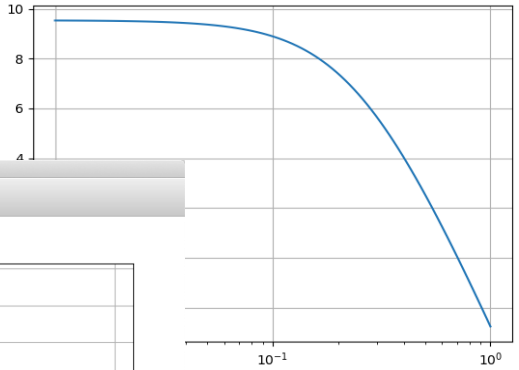
Continuous-time linear systems

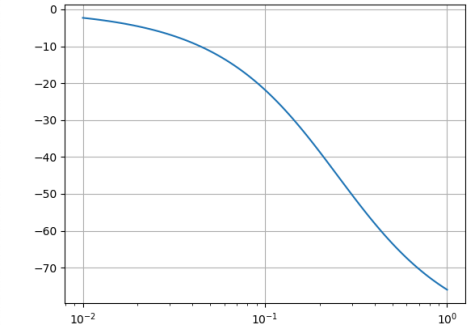
lti(*system)	Continuous-time linear time invariant system base class.
StateSpace(*system, **kwargs)	Linear Time Invariant system in state-space form.
TransferFunction(*system, **kwargs)	Linear Time Invariant system class in transfer function form.
ZerosPolesGain(*system, **kwargs)	Linear Time Invariant 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.

Basic Example

Transfer Function: $H(s) = \frac{3}{4s + 1}$

Magnitude Plot





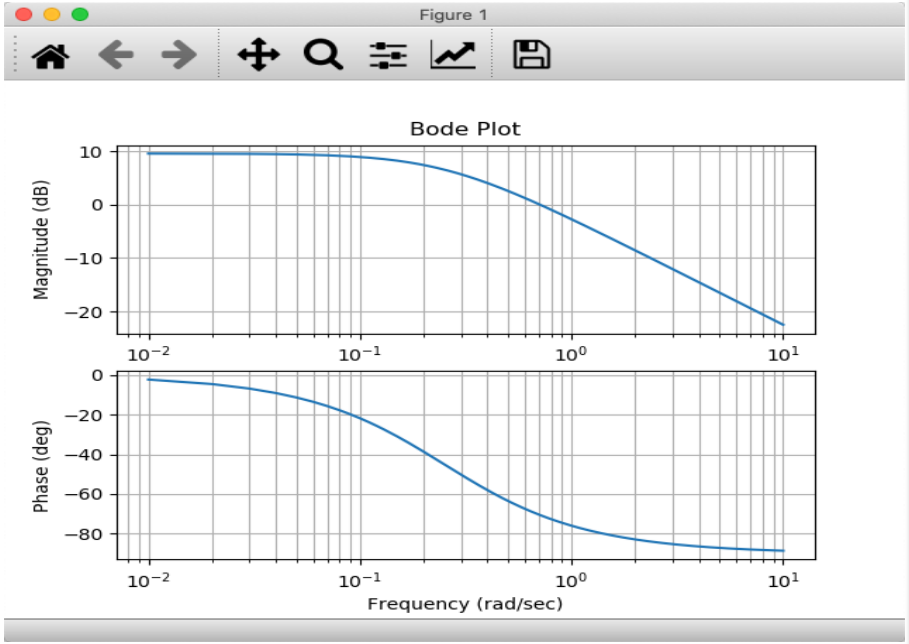
Phase Plot

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

num = np.array([3])
den = np.array([4 , 1])

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

w, mag, phase = signal.bode(H)

plt.figure()
plt.semilogx(w, mag) # Magnitude Plot
plt.grid()
plt.figure()
plt.semilogx(w, phase)  # Phase plot
plt.grid()
plt.show()
```

Modified

Transfer Function:

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

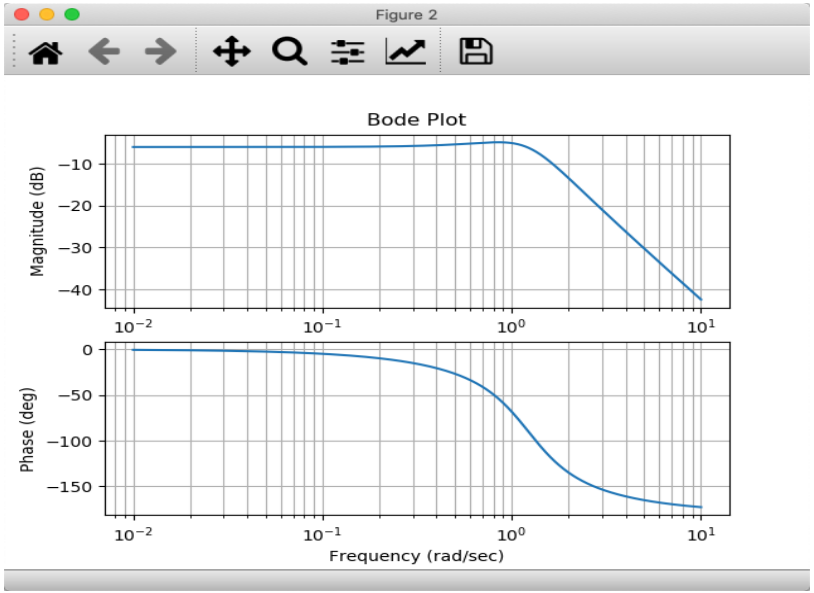



```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([3])
den = np.array([4 , 1])

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()
```

2.order System

Transfer Function

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




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

# Define Transfer Function
num = np.array([3])
den = np.array([4 , 5, 6])

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()
```

Phase Plot and Transfer Functions

Integrator:	$H(s) = \frac{K}{s}$	➡	The phase goes to $-90°$
1.order System:	$H(s) = \frac{K}{Ts + 1}$	➡	The phase goes to $-90°$
2.order System:	$H(s) = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$	➡	The phase goes to $-180°$
Derivator:	$H(s) = Ks$	➡	The phase goes to $+90°$
Zero Part:	$H(s) = K(Ts + 1)$	➡	The phase goes to $+90°$

K = Gain, T = Time Constant(s)

Phase Plot and Transfer Functions

Zeros in the Numerator

The phase goes to $+90°$ for each zero

$$H(s) = \frac{(T_{n1}s + 1)(T_{n2}s + 1)\cdots(T_{nk}s + 1)}{(T_{d1}s + 1)(T_{d2}s + 1)\cdots(T_{dm}s + 1)}$$

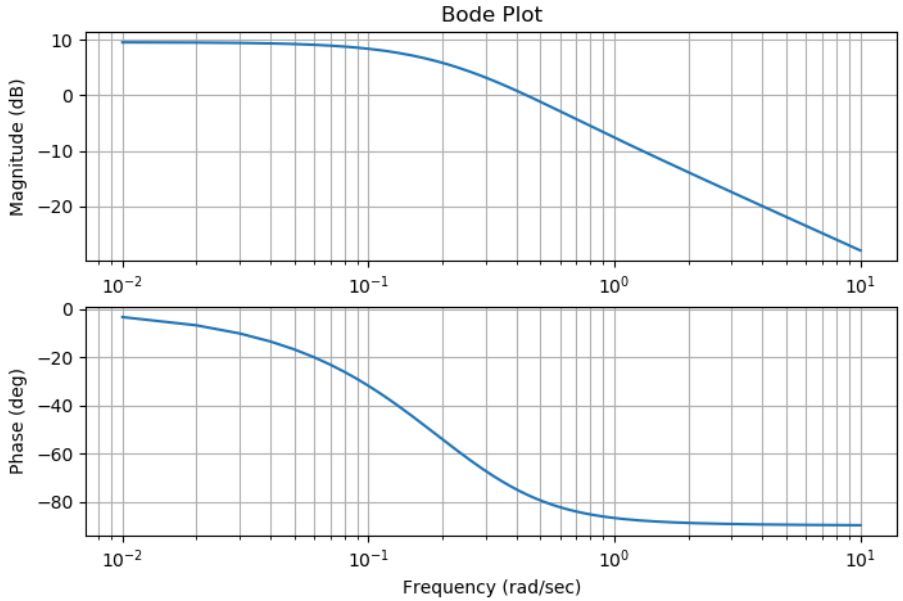
The phase goes to $-90°$ for each pole

Poles in the Denominator

Zeros/Poles

Transfer Function:

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



The phase goes to $-90°$ (There are 1 zero and 2 poles)

```
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()
```

https://www.halvorsen.blog

Python Control Systems Library

Hans-Petter Halvorsen

Python Control Systems Library

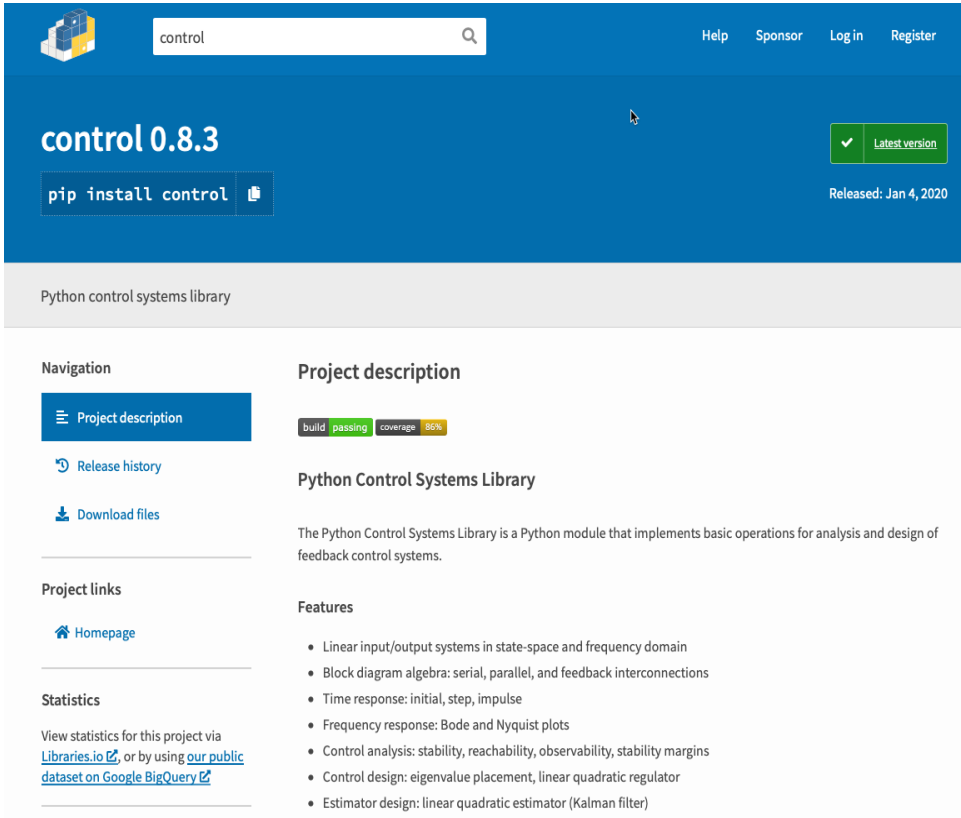
- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: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

Installation

The Python Control Systems Library package may be installed using pip:

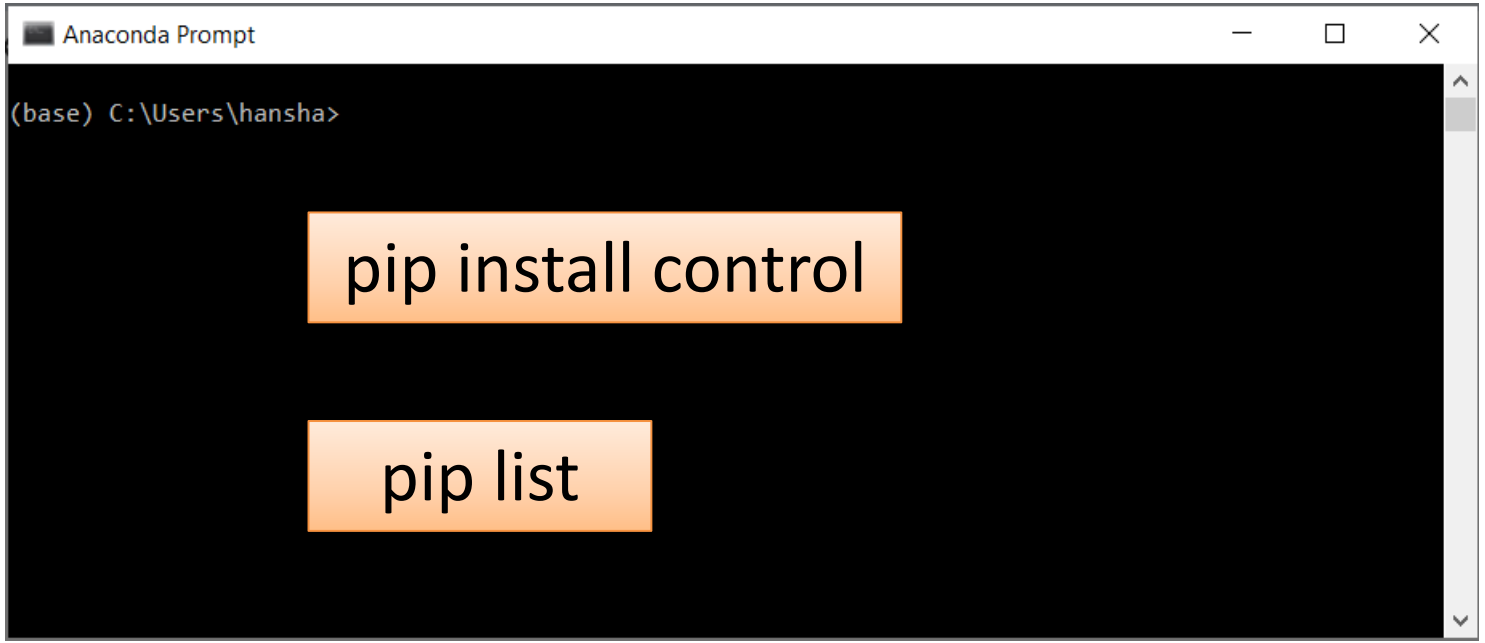
```
pip install control
```

- PIP is a **Package Manager** for Python packages/modules.
- You find more information here: https://pypi.org
- Search for “control“
- **The Python Package Index (PyPI)** is a repository of Python packages where you use PIP in order to install them

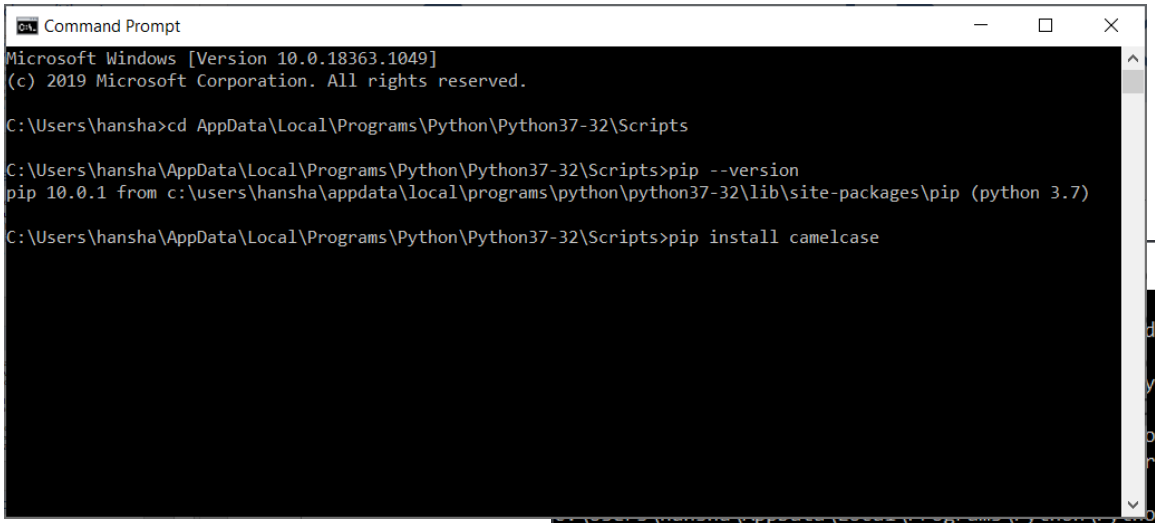


Anaconda Prompt

If you have installed Python with **Anaconda Distribution**, use the **Anaconda Prompt** in order to install it (just search for it using the Search field in Windows).



Command Prompt - PIP



pip install control

C:\Users\hansha\AppData\Local\Programs\Python\Python37-32\Scripts\pip install control

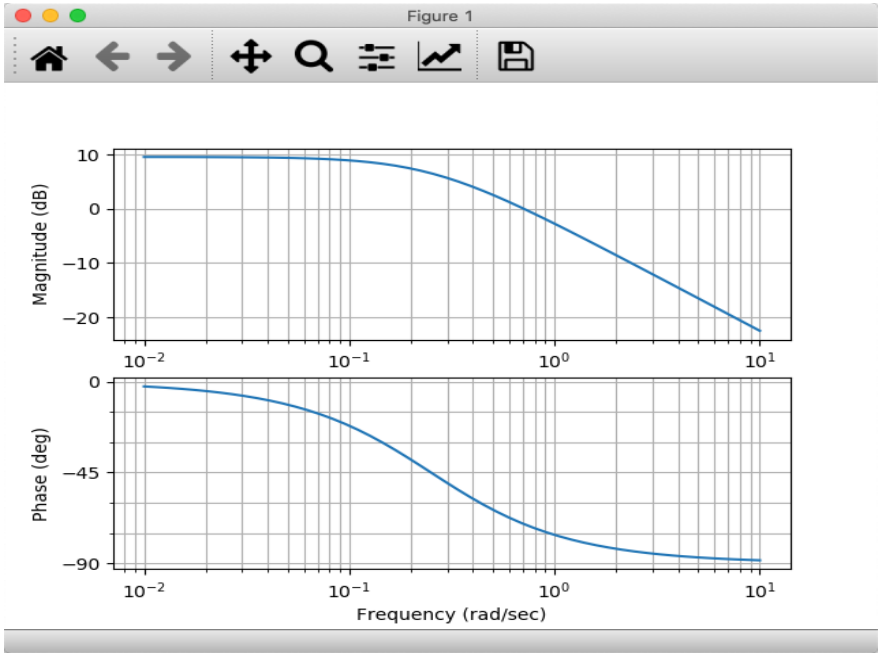
pip list

or “Python37_64“ for Python 64bits

Basic Example

Transfer Function

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




```
import numpy as np
import control

num = np.array([3])
den = np.array([4 , 1])

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

control.bode_plot(H, dB=True)
```

control.bode_plot()

control.bode_plot(*syslist, omega=None, Plot=True, omega_limits=None, omega_num=None, margins=None, *args, **kwargs*)

Bode plot for a system

Plots a Bode plot for the system over a (optional) frequency range.

Parameters:

- **syslist** (*linsys*) – List of linear input/output systems (single system is OK)
- **omega** (*list*) – List of frequencies in rad/sec to be used for frequency response
- **dB** (*bool*) – If True, plot result in dB. Default is false.
- **Hz** (*bool*) – If True, plot frequency in Hz (omega must be provided in rad/sec). Default value (False) set by config.defaults['bode.Hz']
- **deg** (*bool*) – If True, plot phase in degrees (else radians). Default value (True) config.defaults['bode.deg']
- **Plot** (*bool*) – If True (default), plot magnitude and phase
- **omega_limits** (*tuple, list, .. of two values*) – Limits of the to generate frequency vector. If Hz=True the limits are in Hz otherwise in rad/s.
- **omega_num** (*int*) – Number of samples to plot. Defaults to config.defaults['freqplot.number_of_samples'].
- **margins** (*bool*) – If True, plot gain and phase margin.
- ***args** – Additional arguments for `matplotlib.plot()` (color, linestyle, etc)
- ****kwargs** – Additional keywords (passed to *matplotlib*)

Returns:

- **mag** (*array (list if len(syslist) > 1)*) – magnitude
- **phase** (*array (list if len(syslist) > 1)*) – phase in radians
- **omega** (*array (list if len(syslist) > 1)*) – frequency in rad/sec

https://python-control.readthedocs.io

From “Scratch“

Transfer Function

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

In this Example we still use the `control.bode_plot()` function, but we don’t use the function for plotting.

We make our own Bode plot using the functions in the **Matplotlib library**

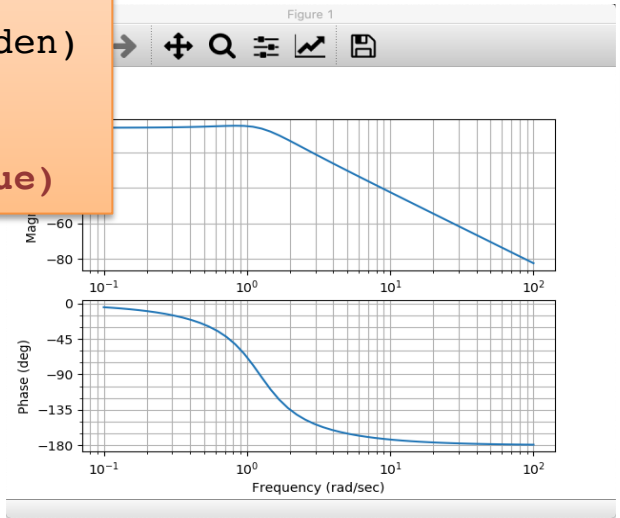
```
import numpy as np
import matplotlib.pyplot as plt
import control

num = np.array([3])
den = np.array([4 , 1])

H = control.tf(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)

# Get Array of of Magnitudes and Phases
mag , phase_rad , w = control.bode_plot(H, w, dB=True, Plot=False)

# Convert to dB
mag_db = 20 * np.log10(mag)

# Convert to Degrees
phase_deg = phase_rad * 180/np.pi;

# Plot Bode Plot
plt.figure()

plt.subplot (2, 1, 1)
plt.semilogx(w, mag_db)    # 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_deg)  # 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()
```

2.order System

Transfer Function: $H(s) = \frac{3}{4s^2 + 5s + 6}$

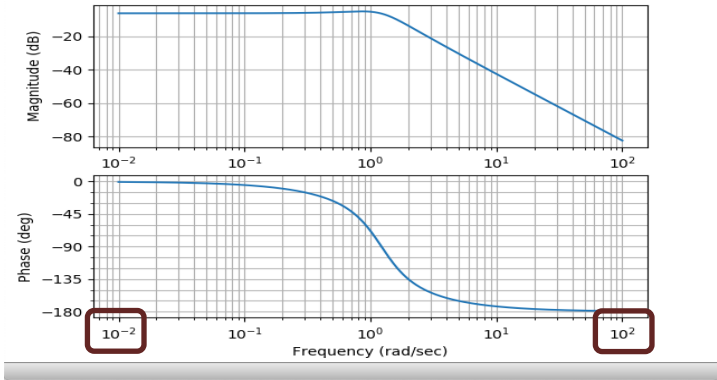
```
import numpy as np
import control

num = np.array([3])
den = np.array([4, 5, 6])

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

control.bode(H, dB=True)
```

Default



Modified

```
import numpy as np
import control

# Transfer Function
num = np.array([3])
den = np.array([4, 5, 6])

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

# Frequencies
w_start = 0.01
w_stop = 100
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

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



Filter Design

Hans-Petter Halvorsen

Lowpass Filter

The Transfer Function for a Low-pass filter is given by:

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

Where T_f is the Filter Time Constant

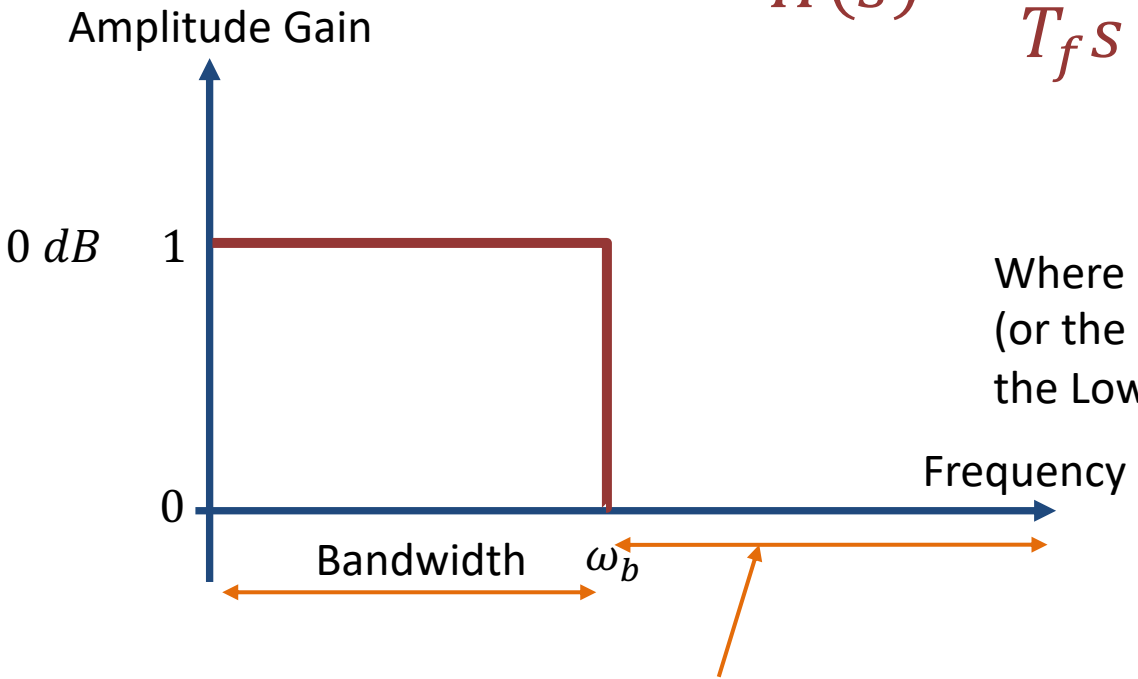
Why Lowpass Filter?

- In Measurement systems and Control Systems we typically need to deal with noise
- Noise is something we typically don't want
- Lowpass Filters are used to remove noise from the measured signals
- Noise is high-frequency signals
- A Lowpass Filter make sure the low frequencies pass and removes the high frequencies (the noise)

Lowpass Filter

Below we see an Ideal Lowpass Filter:

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



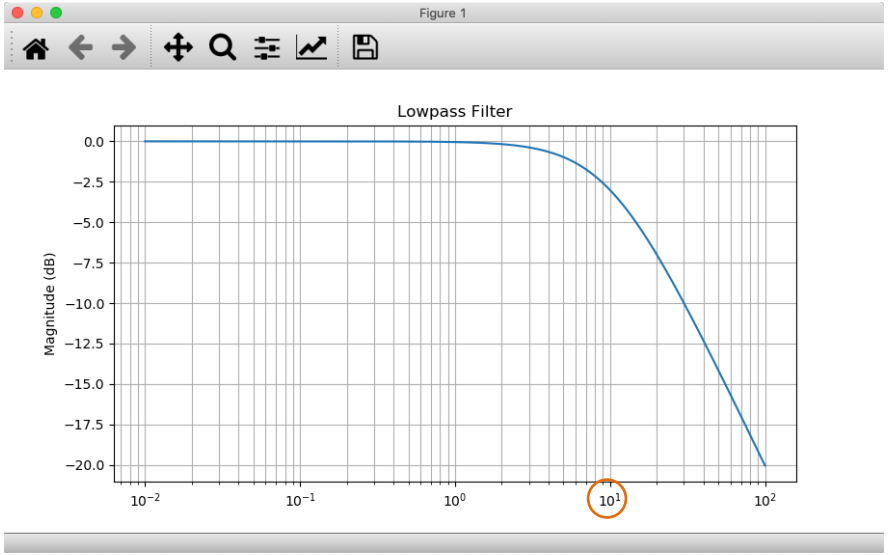
Where ω_b is the Bandwidth (or the cut-off frequency) of the Lowpass Filter

High frequencies (above ω_b) are removed (or reduced)

Python

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

We set $\omega_b = 10$ [rad/s] in this example



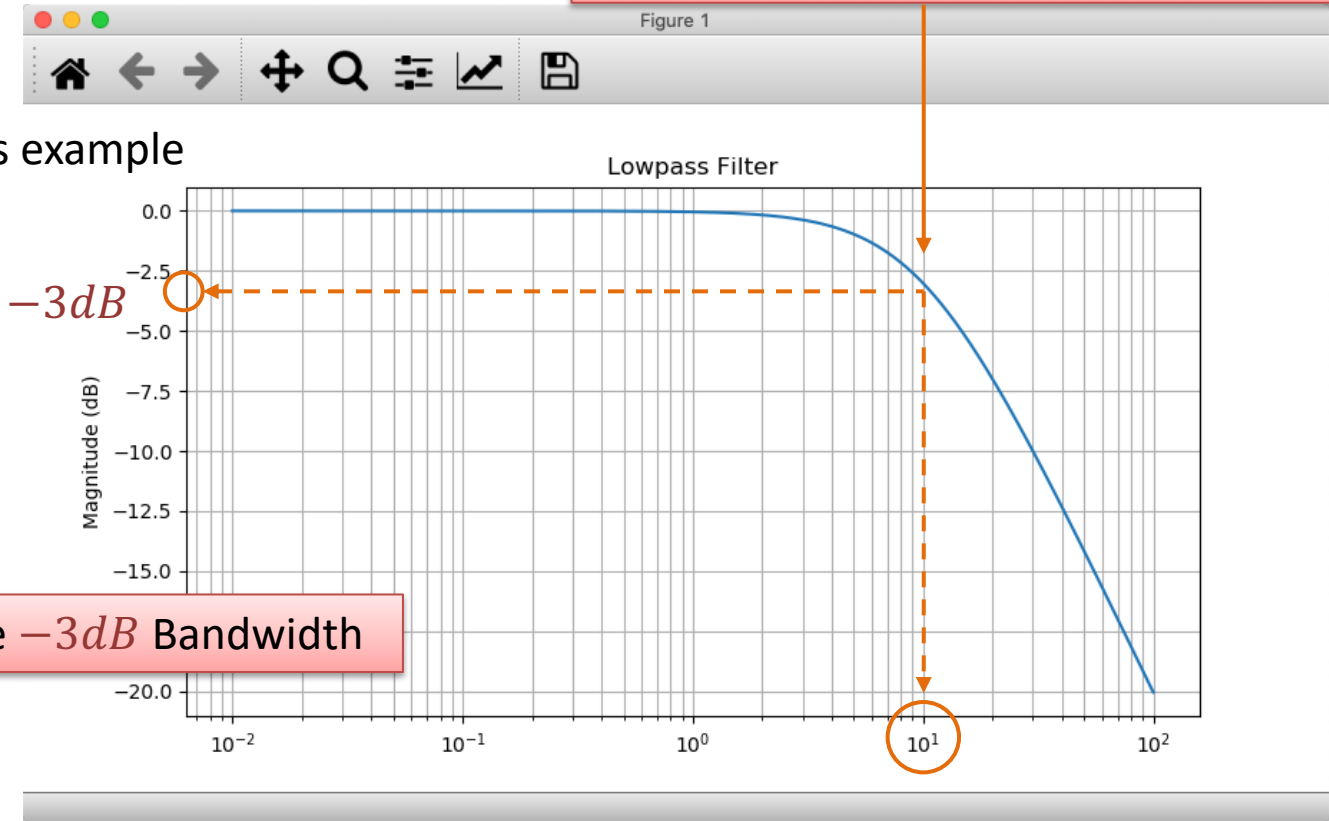

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define Transfer Function
wb = 10 #rad/s
Tf = 1/wb
num = np.array([1])
den = np.array([Tf , 1])

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

# Frequencies
w_start = 0.01
w_stop = 100
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

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

plt.figure()
plt.semilogx(w, mag)
plt.title("Lowpass Filter")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")
```

Bandwidth

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

We set $\omega_b = 10$ [rad/s] in this example

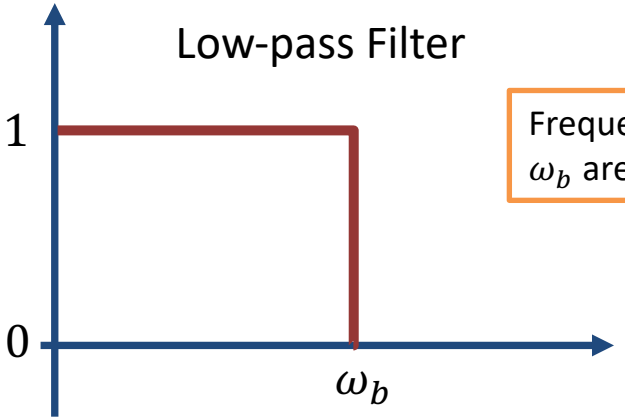
Bandwidth (or the cut-off frequency)



ω_b is also referred to as the $-3dB$ Bandwidth

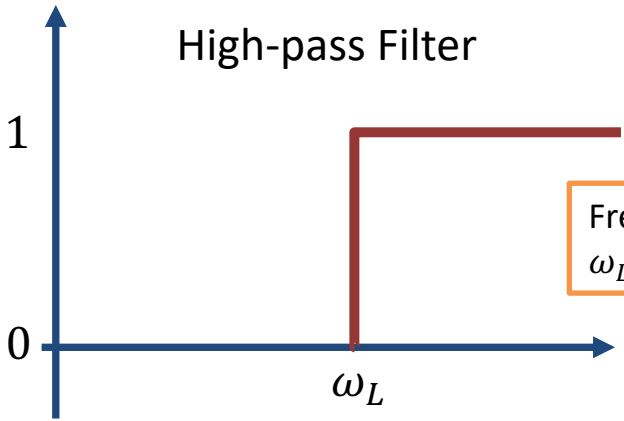
Filters

Low-pass Filter



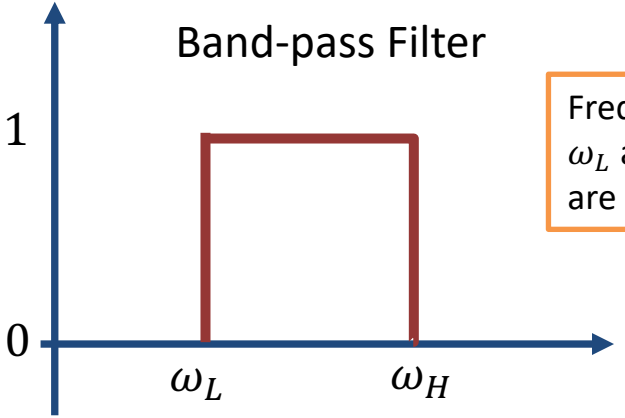
Frequencies above ω_b are reduced

High-pass Filter



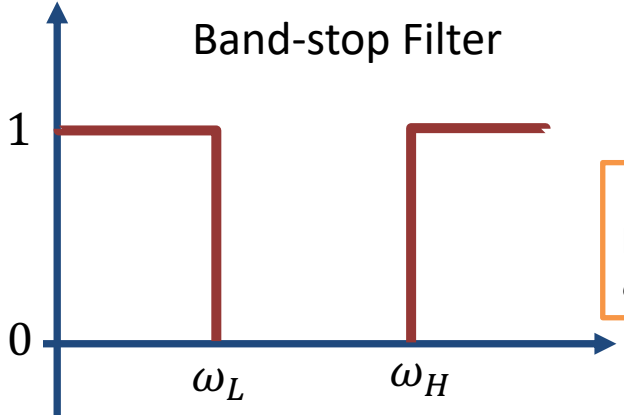
Frequencies below ω_L are reduced

Band-pass Filter



Frequencies below ω_L and above ω_H are reduced

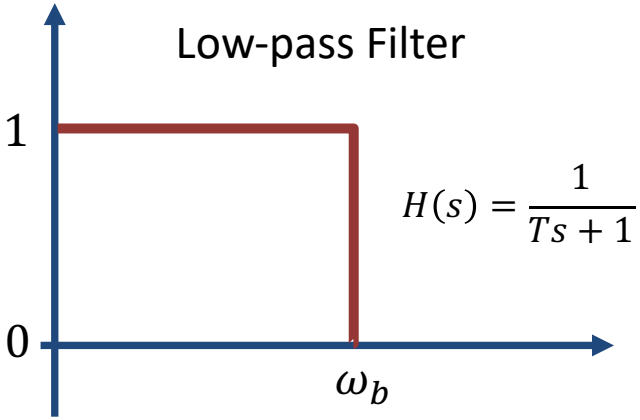
Band-stop Filter



Frequencies between ω_L and ω_H are reduced

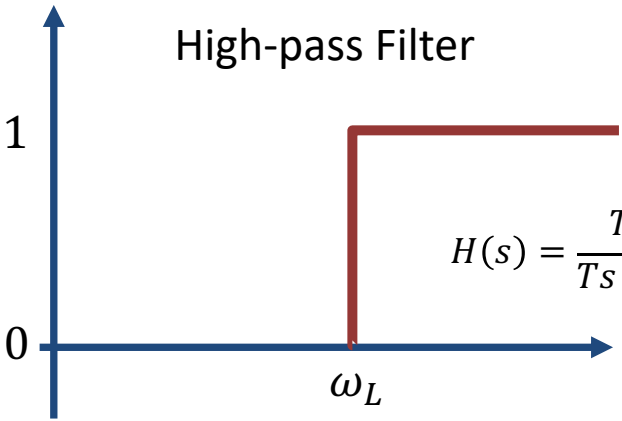
Filters

Low-pass Filter



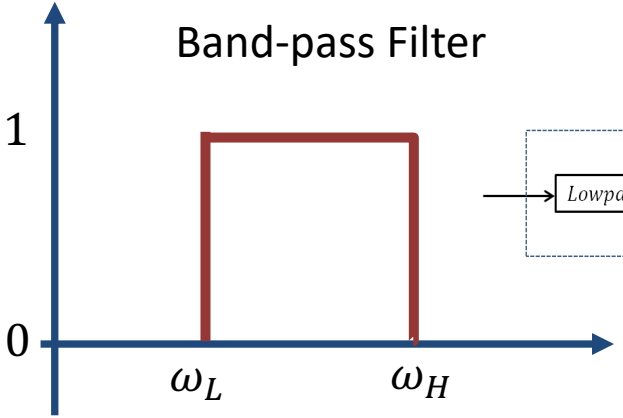
$$H(s) = \frac{1}{Ts+1} = \frac{1}{\frac{1}{\omega_b}s+1}$$

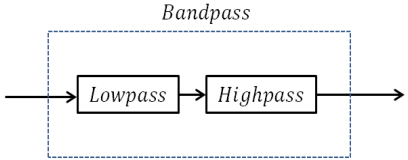
High-pass Filter



$$H(s) = \frac{Ts}{Ts+1} = \frac{\frac{1}{\omega_L}s}{\frac{1}{\omega_L}s+1}$$

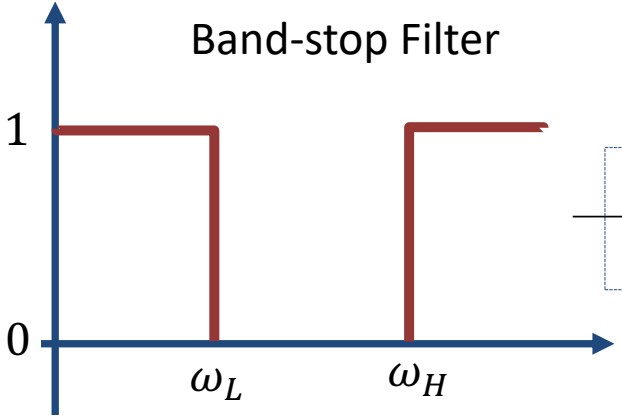
Band-pass Filter

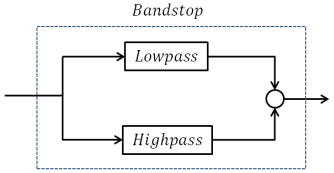




Series

Band-stop Filter

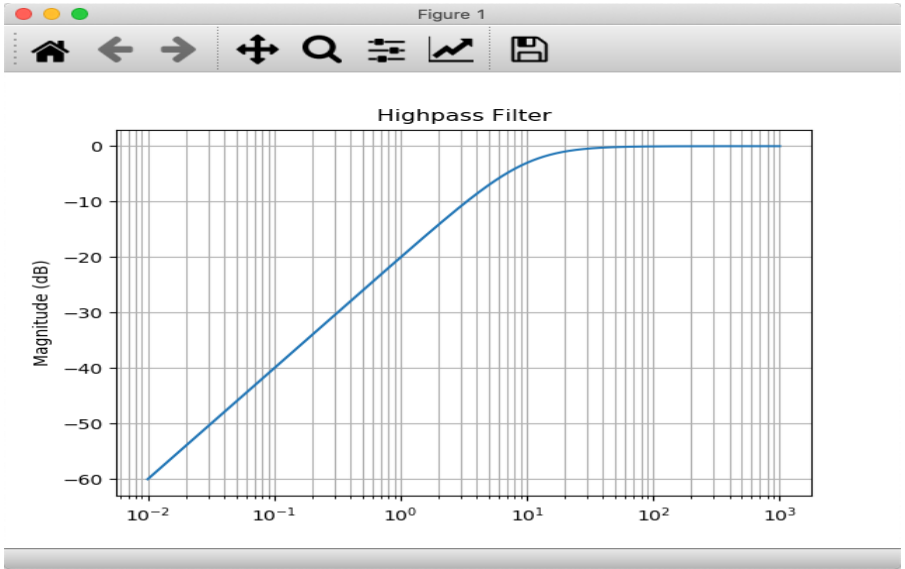




Parallel

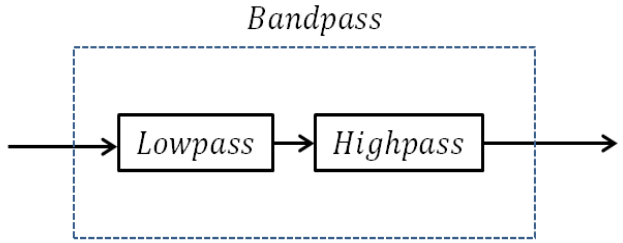
High-pass Filter

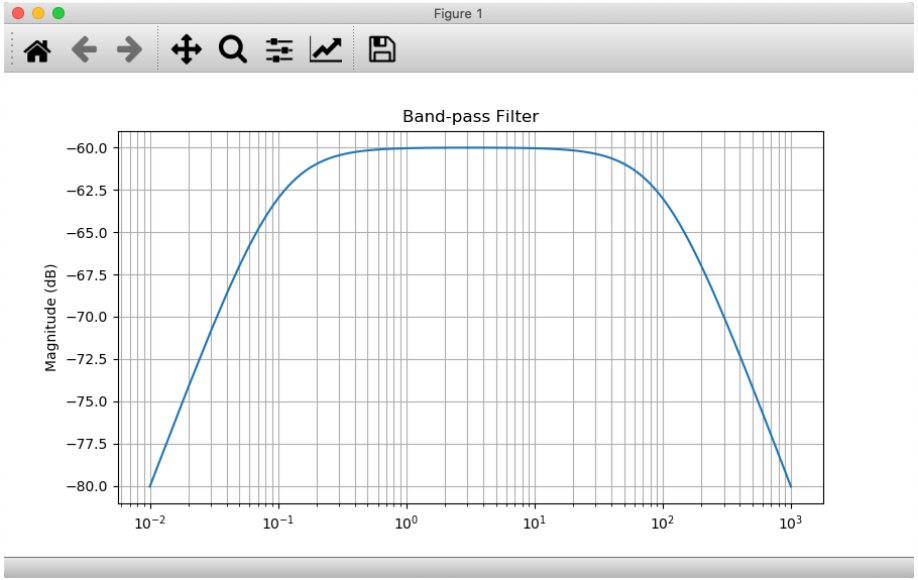
$$H(s) = \frac{Ts}{Ts+1} = \frac{\frac{1}{\omega_L}s}{\frac{1}{\omega_L}s+1}$$




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

# Define Transfer Function
wb = 10 #rad/s
Tf = 1/wb
num = np.array([Tf, 0])
den = np.array([Tf , 1])

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

# Frequencies
w_start = 0.01
w_stop = 1000
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

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

plt.figure()
plt.semilogx(w, mag)
plt.title("Highpass Filter")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")
```

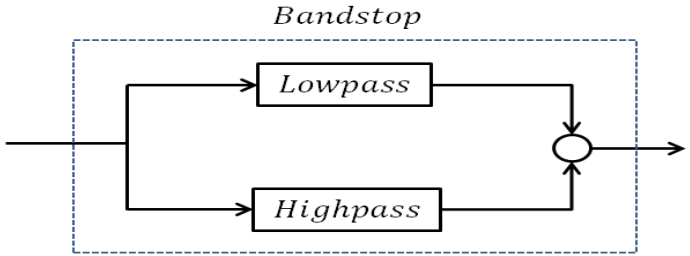
Band-pass Filter

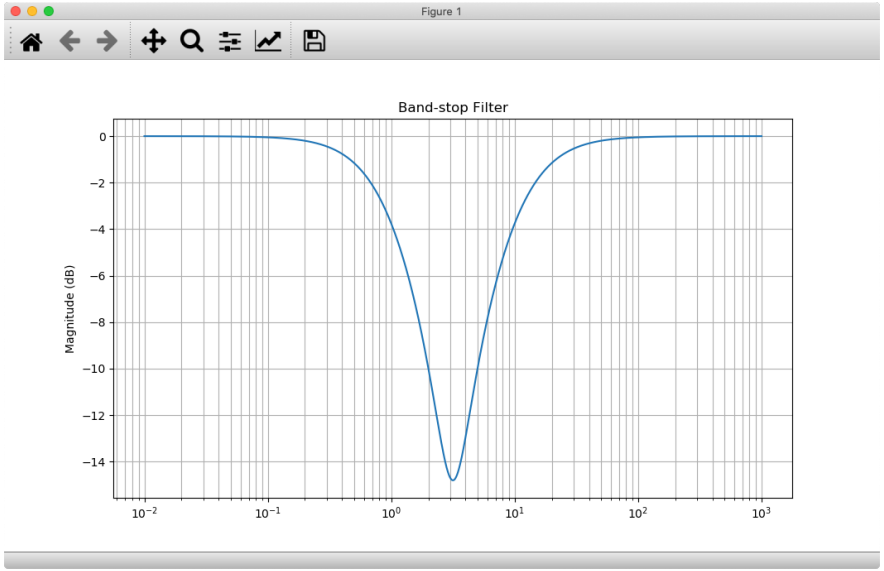





```
import numpy as np
import control
import matplotlib.pyplot as plt

# Low-pass Transfer Function
wl = 0.1 #rad/s
Tfl = 1/wl
num = np.array([1])
den = np.array([Tfl , 1])
HL = control.tf(num, den)

# High-pass Transfer Function
wh = 100 #rad/s
Tfh = 1/wh
num = np.array([Tfh, 0])
den = np.array([Tfh, 1])
HH = control.tf(num, den)

# Band-pass Transfer Function
HBP = control.series(HL, HH)

# Frequencies
w_start = 0.01
w_stop = 1000
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

# Frequency Response Plot
mag , phase_rad , w  = control.bode_plot(HBP, w, dB=True,
Plot=False)

# Convert to dB
mag_db = 20 * np.log10(mag)

plt.figure()
plt.semilogx(w, mag_db)
plt.title("Lowpass Filter")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")
```

Band-stop Filter





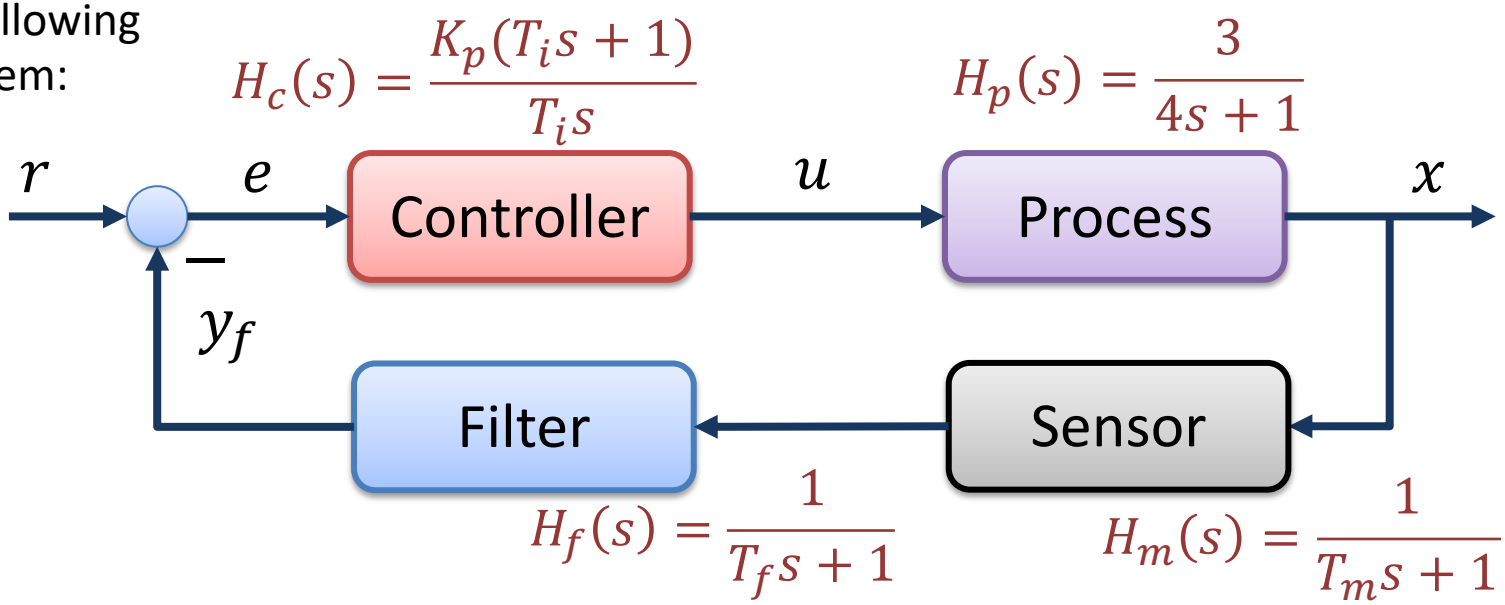

```
import numpy as np
import control
import matplotlib.pyplot as plt

# Low-pass Transfer Function
wl = 1 #rad/s
Tfl = 1/wl
num = np.array([1])
den = np.array([Tfl , 1])
HL = control.tf(num, den)

# High-pass Transfer Function
wh = 10 #rad/s
Tfh = 1/wh
num = np.array([Tfh, 0])
den = np.array([Tfh, 1])
HH = control.tf(num, den)

# Band-stop Transfer Function
HBS = control.parallel(HL, HH)

# Frequencies
w_start = 0.01
w_stop = 1000
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

# Frequency Response Plot
mag , phase_rad , w  = control.bode_plot(HBS, w, dB=True,
Plot=False)

# Convert to dB
mag_db = 20 * np.log10(mag)

plt.figure()
plt.semilogx(w, mag_db)
plt.title("Lowpass Filter")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")
```

Frequency Response Stability Analysis of Control Systems

Hans-Petter Halvorsen

Stability Analysis Example

Given the following Control System:



In Frequency Response Stability Analysis we use the Loop Transfer Function:

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

Frequency Response 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.

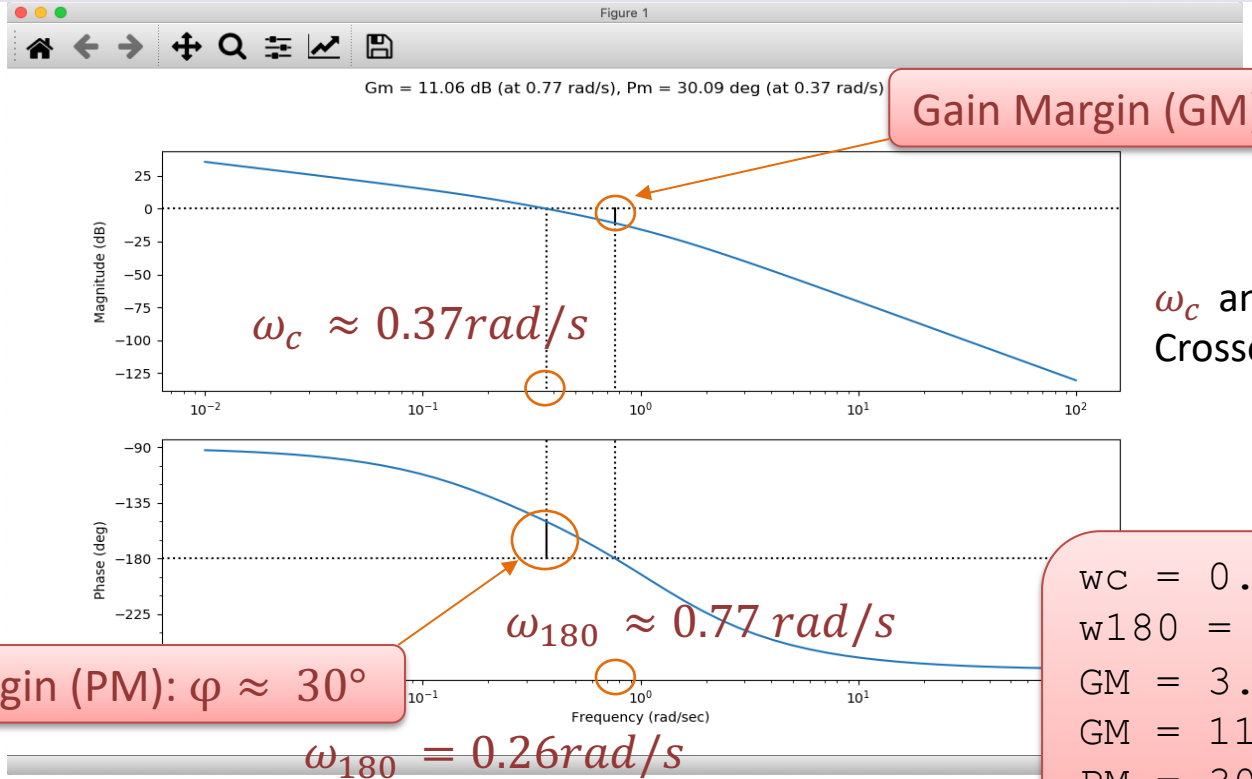
- **The Gain Margin – GM** (ΔK) is how much the loop gain can increase before the system become unstable.
- The **Phase Margin - PM** (ϕ) is how much the phase lag function of the loop can be reduced before the loop becomes unstable.

```
import numpy as np
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)

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

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

Frequency Response Analysis

$K_p = 0.4$
$T_i = 2s$



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

ω_c and ω_{180} are called Crossover Frequencies

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

$\omega_{180} = 0.26 rad/s$

```
wc = 0.37 rad/s
w180 = 0.77 rad/s
GM = 3.57
GM = 11.06 dB
PM = 30.09 deg
Kc = 1.43
```

As you see we have an Asymptotically Stable System

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

Frequency Response Analysis

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

$$\omega_c < \omega_{180}$$

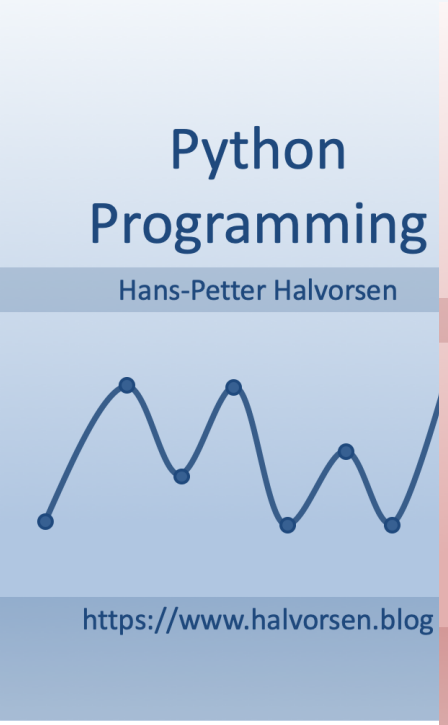
We have a Marginally Stable System when $K_p = K_c$

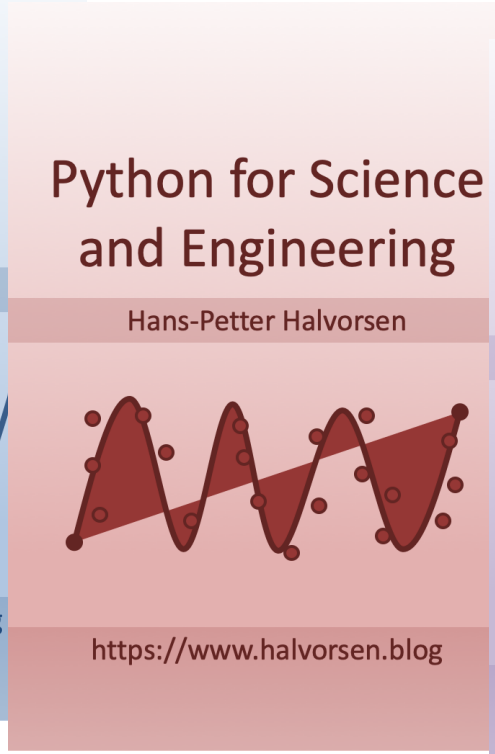
$$\omega_c = \omega_{180}$$

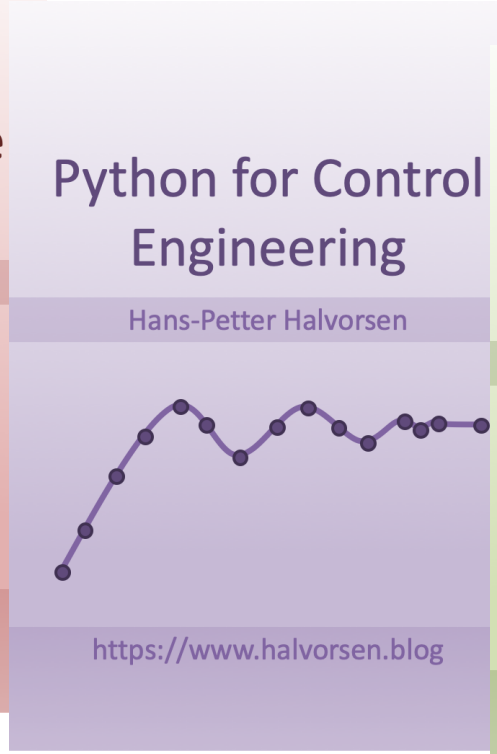
We have an Unstable System when $K_p > K_c$

$$\omega_c > \omega_{180}$$

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