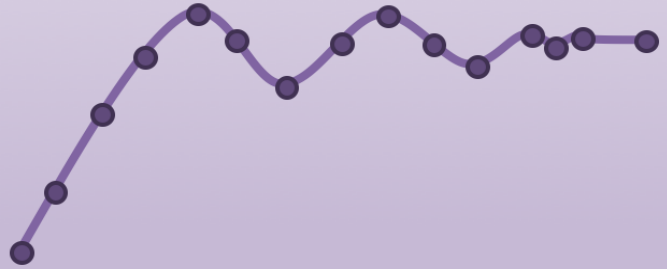
Transfer Functions with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Control Engineering

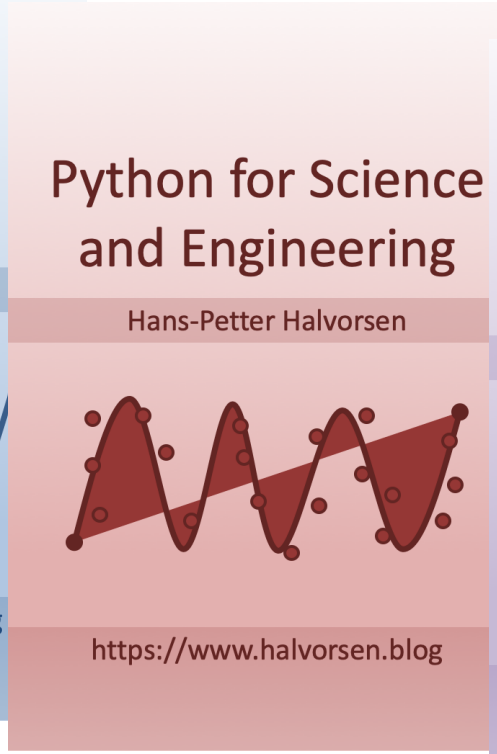
Hans-Petter Halvorsen

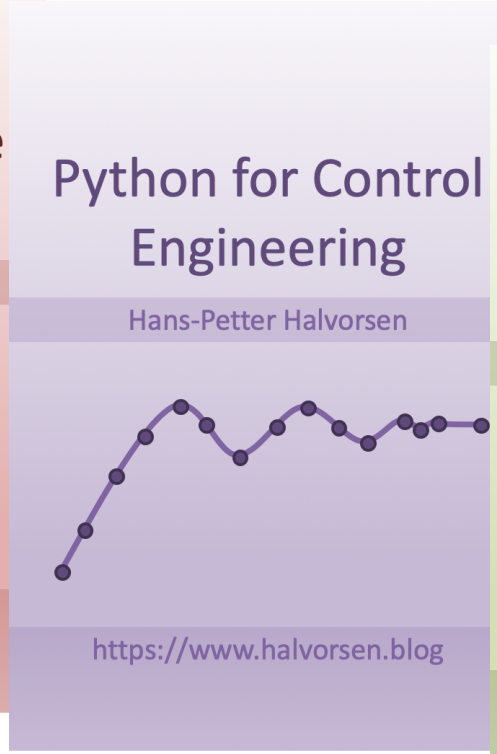


https://www.halvorsen.blog

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

Additional Python Resources



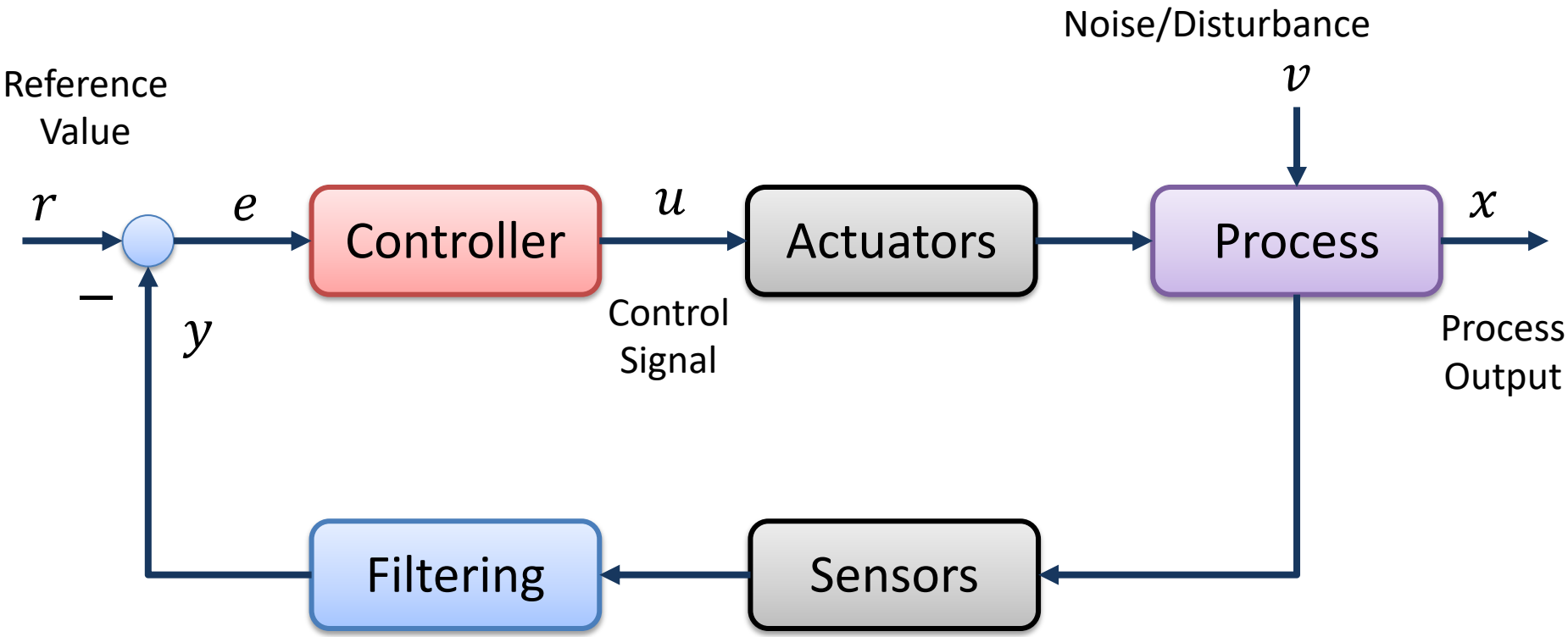


https://www.halvorsen.blog/documents/programming/python/

Contents

- Introduction to Control Systems
- Transfer Functions
- Step Response
- Poles and Zeros
- Python Examples
  - SciPy (SciPy.signal)
  - The Python Control Systems Library

Control System



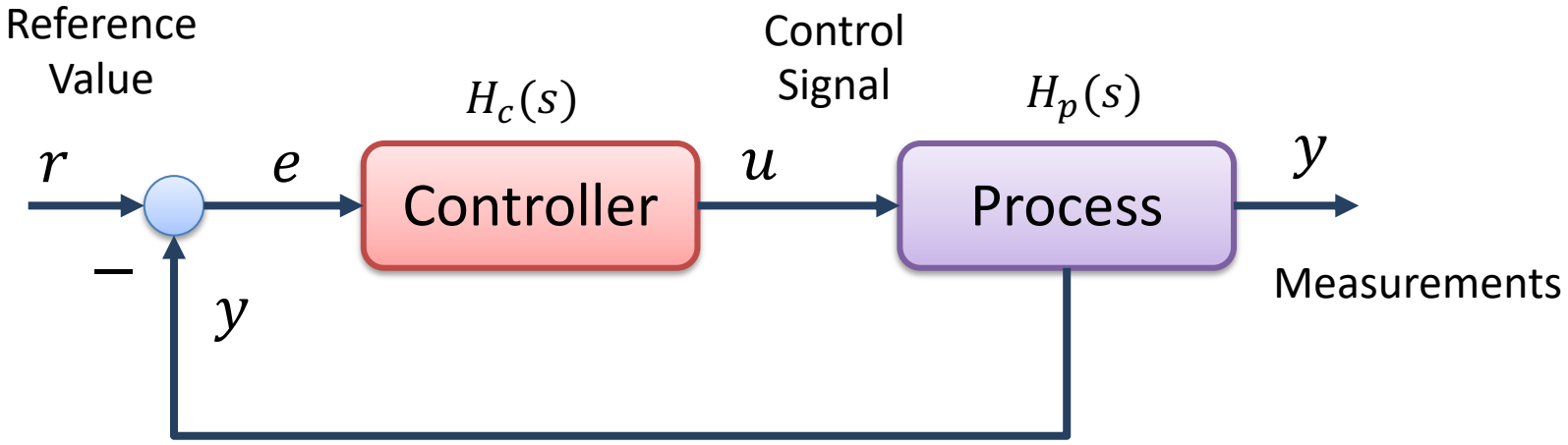
Control System

- r – Reference Value, SP (Set-point), SV (Set Value)
- y – Measurement Value (MV), Process Value (PV)
- e – Error between the reference value and the measurement value ($e = r - y$)
- v – Disturbance, makes it more complicated to control the process
- u - Control Signal from the Controller

Control System

Simplified Control System:

The purpose with the controller is to make sure the process stays on a desired level, e.g., it could be level in a liquid tank or the temperature at a specific value, e.g., 30°C



The Controller is typically a PID Controller that has K_p, T_i and T_d as Tuning Parameters

Transfer Functions

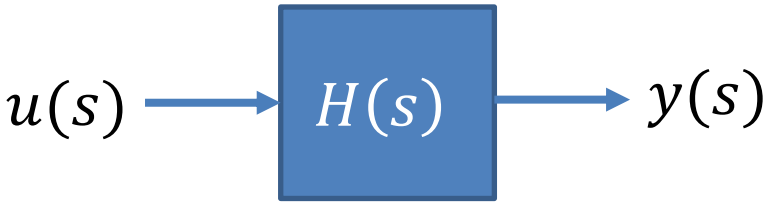
- Transfer functions are a model form based on the Laplace transform.
- Transfer functions are very useful in analysis and design of linear dynamic systems.

Transfer Functions

A general Transfer function is on the form:

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

Where y is the output and u is the input.
s is the Laplace operator



Transfer Functions

A general transfer function can be written on the following general form:

$$H(s) = \frac{numerator(s)}{denominator(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1\, s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1\, s + a_0}$$

The Numerators of transfer function models describe the locations of the zeros of the system, while the Denominators of transfer function models describe the locations of the poles of the system.

Transfer Functions - Examples

1. Order System:

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

1. Order System with Integrator:

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

1. Order System with Time Delay:

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

2. Order System:

$$H(s) = \frac{4}{3s^2 - 2s + 1}$$

Transfer Functions - Python

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

```
import numpy as np
import control

num = np.array ([2])
den = np.array ([3 , 1])

H = control.tf(num , den)

print ('H(s) =', H)
```

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

```
import numpy as np
import control

num = np.array ([3])
den = np.array ([2 , 1, 0])

H = control.tf(num , den)

print ('H(s) =', H)
```

$$H(s) = \frac{4}{3s^2 - 2s + 1}$$

```
import numpy as np
import control

num = np.array ([4])
den = np.array ([3 , -2, 1])

H = control.tf(num , den)

print ('H(s) =', H)
```

```
import numpy as np
import control

num = np.array ([4, 1])
den = np.array ([2 , 5, -2])

H = control.tf(num , den)

print ('H(s) =', H)
```

$$H(s) = \frac{4s+1}{2s^2 + 5s - 2}$$

Transfer Functions

1.order Transfer Function:

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

Integrator:

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

1. order Transfer Function with Time Delay:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

Transfer Function for Time Delay:

$$H(s) = e^{-\tau s}$$

2. order Transfer Function:

$$H(s) = \frac{K}{(T_1 + 1)(T_2 s + 1)}$$

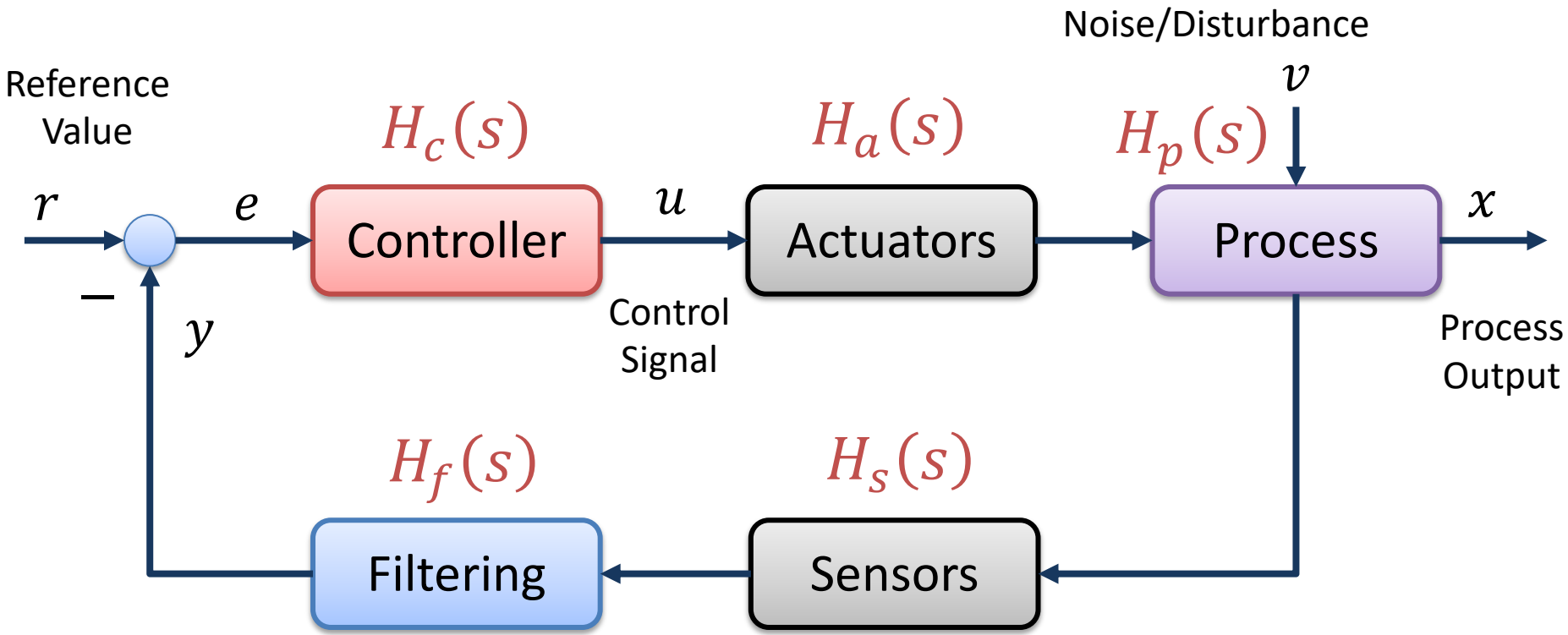
or:

$$H(s) = \frac{K}{as^2 + bs + c}$$

More about the different types of Transfer Functions later in this Tutorial

Control System

Back to the Control System: Each block can be described by a Transfer Function:



Python Examples

Hans-Petter Halvorsen

Python Examples

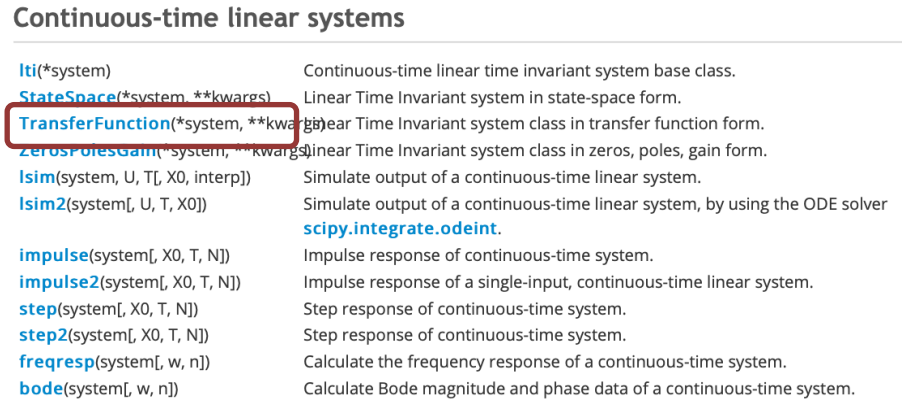
- SciPy (SciPy.signal)
  - Included with Anaconda Distribution
  - Limited Functions and Features for Control Systems
- Python Control Systems Library
  - I will refer to it as the “Control” Library
  - Very similar features as the MATLAB Control System Toolbox
  - You need to install it (“pip install control”)

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.

Python

Transfer Function:

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

```
H(s) =
TransferFunctionContinuous(
array([0.66666667]),
array([1., 0.33333333]),
dt: None
)
```

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

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

H = signal.TransferFunction(num , den)

print ('H(s) =', H)

# Step Response
t, y = signal.step(H)

# Plotting
plt.plot(t, y)
plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()
```

Comments to Results

Transfer Function:

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

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

$$H(s) = \frac{0.67}{s + 0.33}$$

```
H(s) =
TransferFunctionContinuous(
array([0.66666667]),
array([1., 0.33333333]),
dt: None
)
```

https://www.halvorsen.blog

Python Control Systems Library

Hans-Petter Halvorsen

Python Control Systems Library

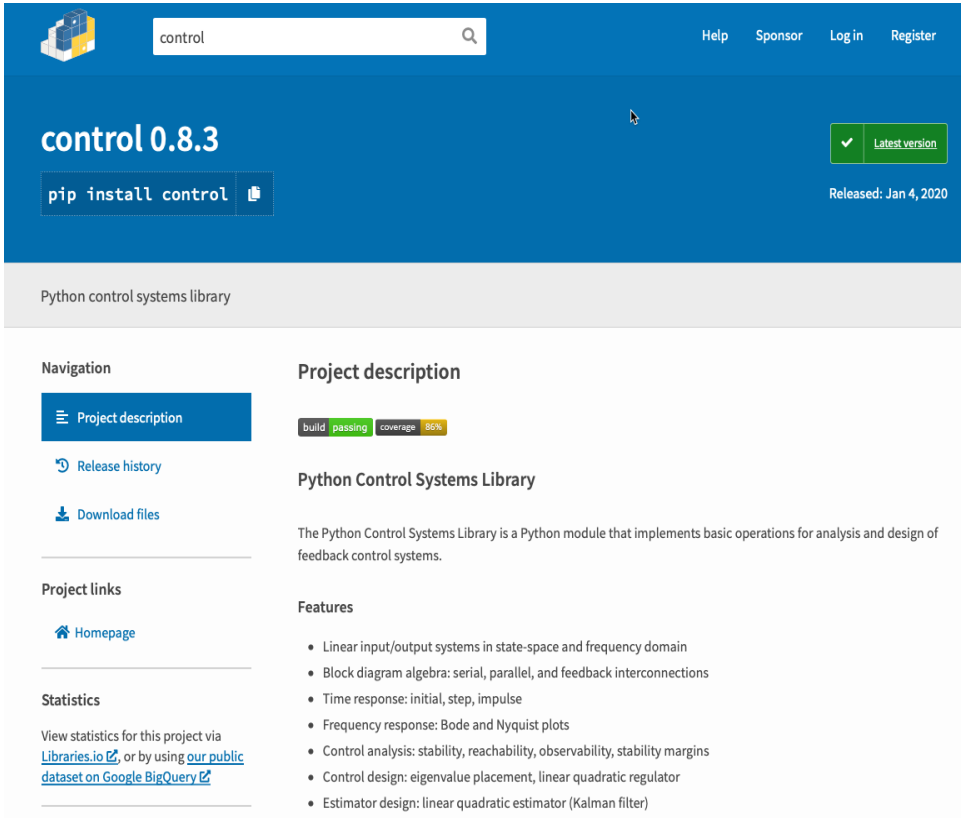
- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: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

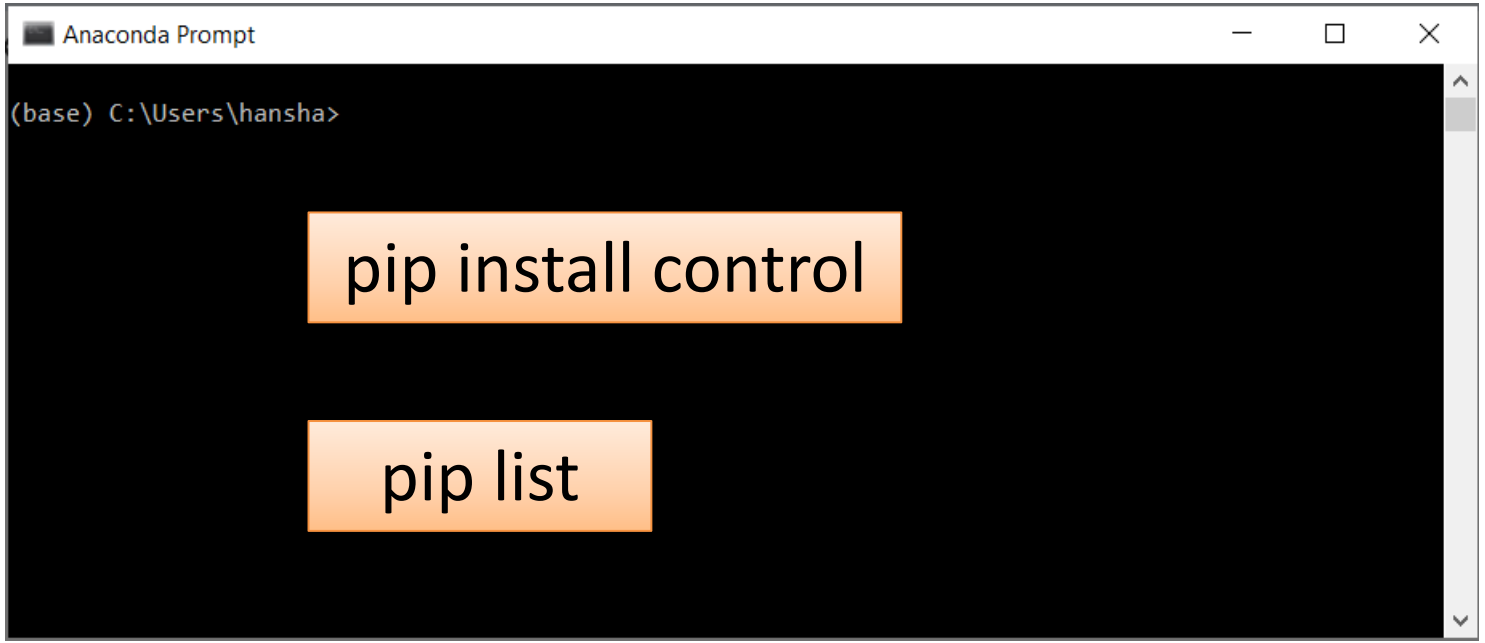
```
pip install control
```

- 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



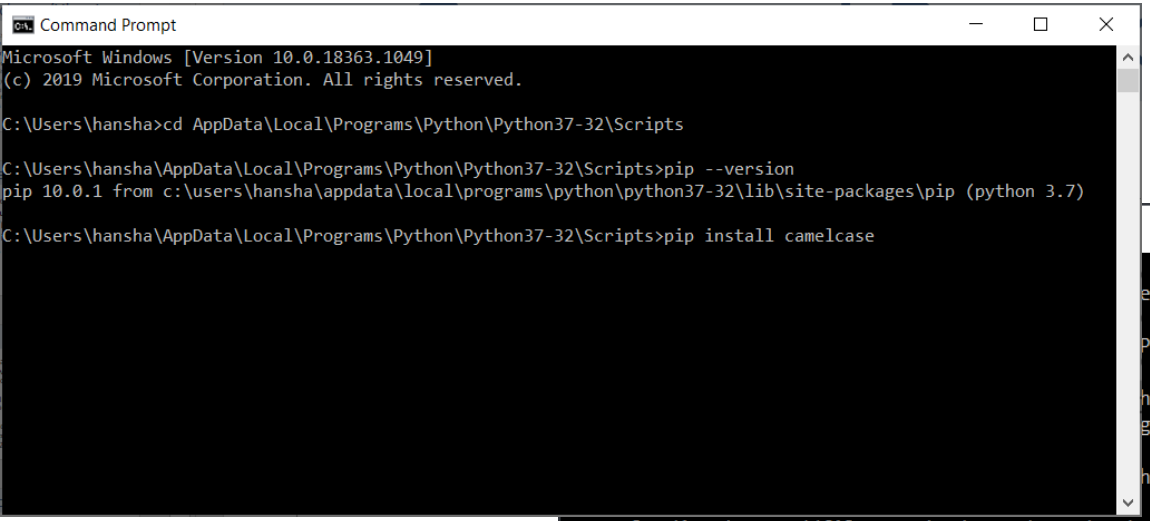
Anaconda Prompt

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).

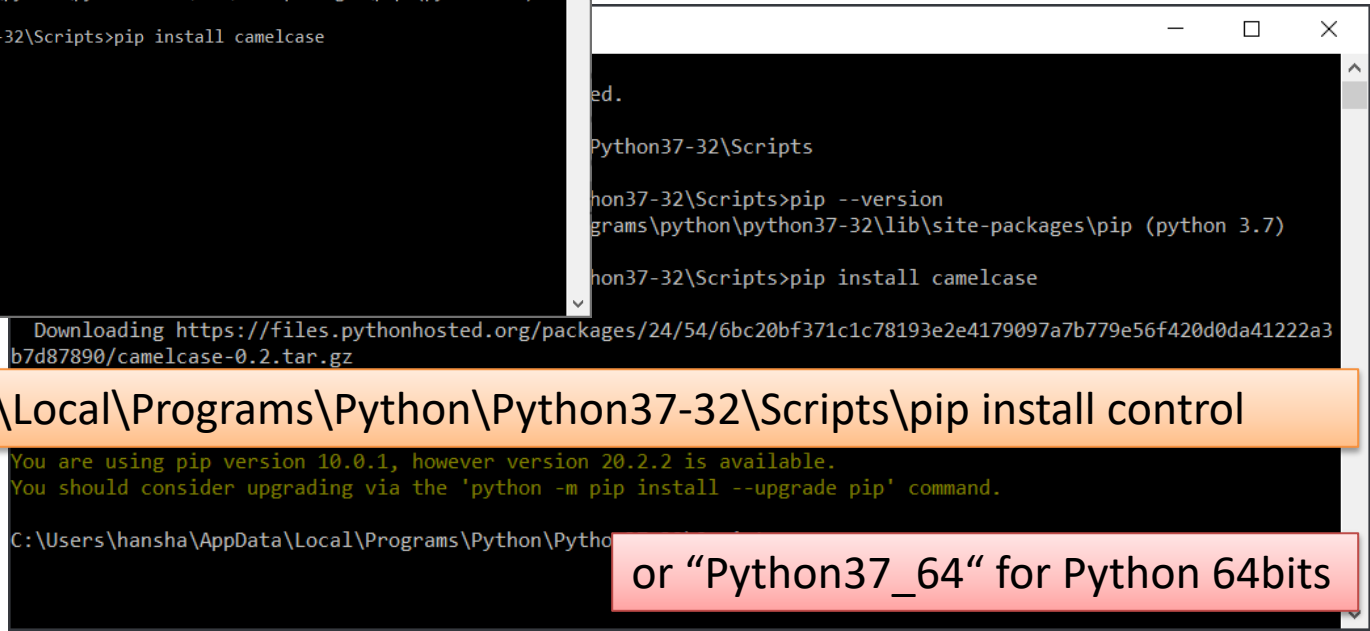


Command Prompt - PIP

Example: Install Python package “camelCase”:



pip install control



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

pip list

or “Python37_64“ for Python 64bits

Python Control Systems Library - Functions

Functions for Model Creation and Manipulation:

- `tf()` - Create a transfer function system
- `ss()` - Create a state space system
- `c2d()` - Return a discrete-time system
- `tf2ss()` - Transform a transfer function to a state space system
- `ss2tf()` - Transform a state space system to a transfer function.
- `series()` - Return the series of 2 or more subsystems
- `parallel()` - Return the parallel of 2 or more subsystems
- `feedback()` - Return the feedback of system
- `pade()` - Creates a Pade Aproxomation, which is a Transfer function representation of a Time Delay

Python Control Systems Library - Functions

Functions for Model Simulations:

- `step_response()` - Step response of a linear system
- `lsim()` - Simulate the output of a linear system

Functions for Stability Analysis:

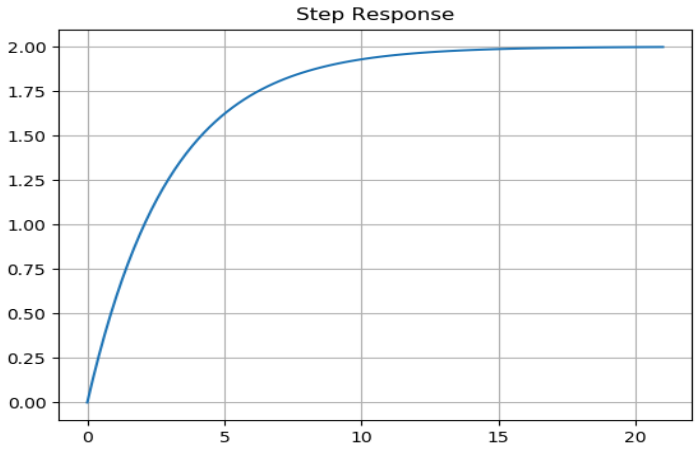
- `step_response()` - Step response of a linear system
- `lsim()` - Simulate the output of a linear system
- `pole()` - Compute system poles
- `zero()` - Compute system zeros
- `pzmap()` - Plot a pole/zero map for a linear system
- `margin()` - Calculate gain and phase margins and frequencies
- `stability_margins()` - Calculate stability margins and frequencies

+++ Many more Functions …

Python

Transfer Function:

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

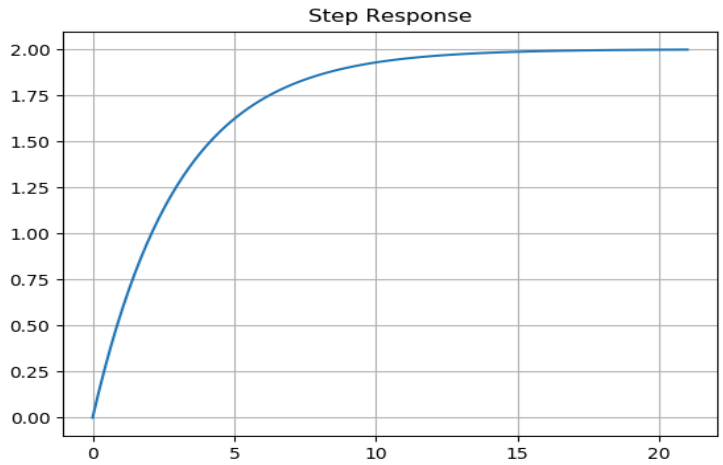



```
import control
import numpy as np
import matplotlib.pyplot as plt

num = np.array([2])
den = np.array([3 , 1])

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

t, y = control.step_response(H)

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

Alternative

Transfer Function:

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




```
import control
import matplotlib.pyplot as plt

s = control.TransferFunction.s

H  = (2)/(3*s + 1)

print ('H(s) =', H)

t, y = control.step_response(H)

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

Alternative

Transfer Function:

$$H(s) = \frac{6s + 1}{s^2 + 4s + 8}$$

Which approach you should use is a matter of taste and what kind of system you are implementing, etc.

```
import control
import matplotlib.pyplot as plt

s = control.TransferFunction.s

H  = (6*s + 1)/(s**2 + 4*s + 8)

print ('H(s) =', H)

t, y = control.step_response(H)

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

Types of Transfer Functions

Hans-Petter Halvorsen

Types of Transfer Functions

- 1. order Transfer Functions
- Integrator
- Time Delay
- 1. order Transfer Function with Time Delay
- 2. order Transfer Functions

1.order Transfer Function

Hans-Petter Halvorsen

1.order Transfer Functions

A 1.order transfer function is given by:

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

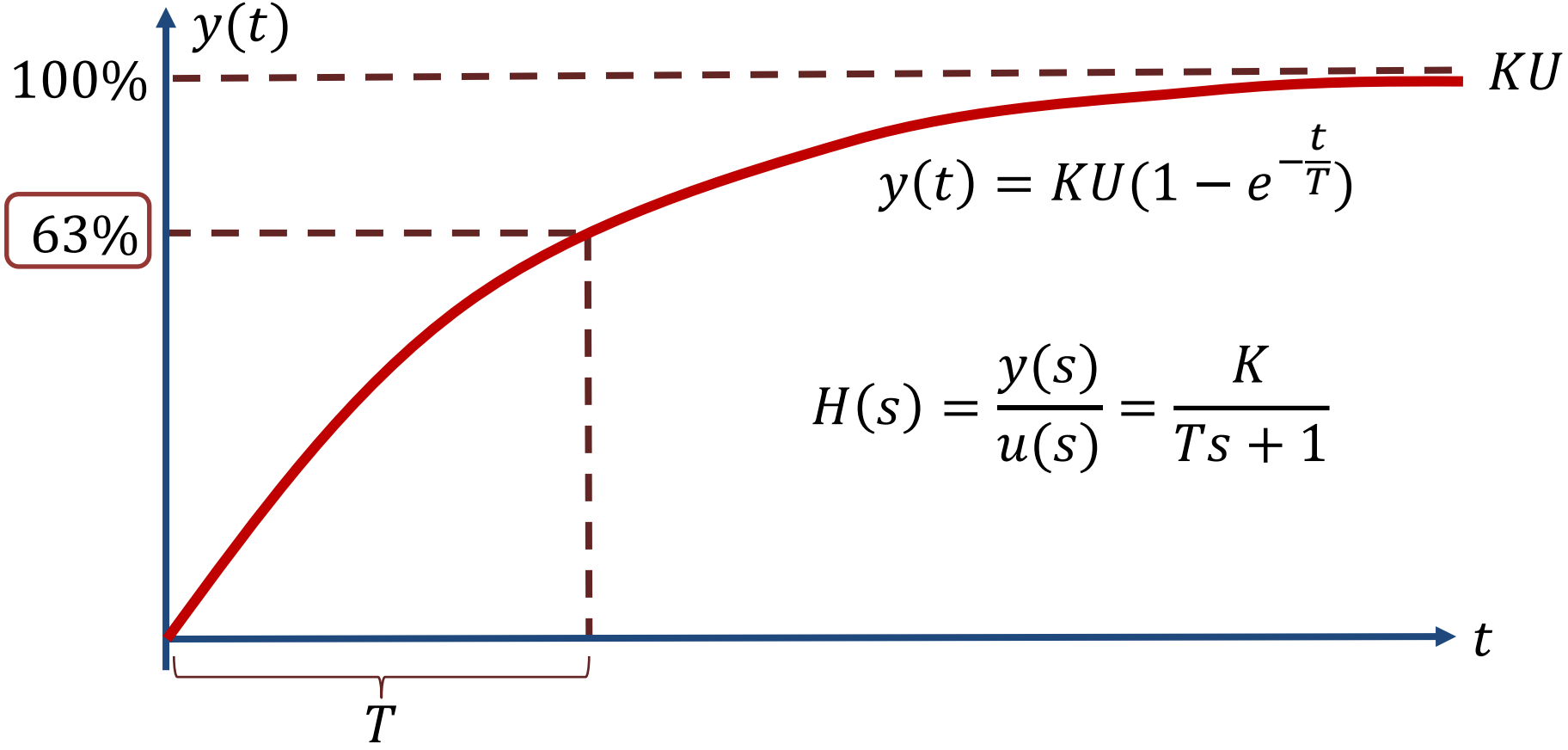
Where K is the Gain and T is the Time constant

In the time domain we get the following equation (using Inverse Laplace):

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

(After a Step U for the input signal $u(s)$)

1.order – Step Response

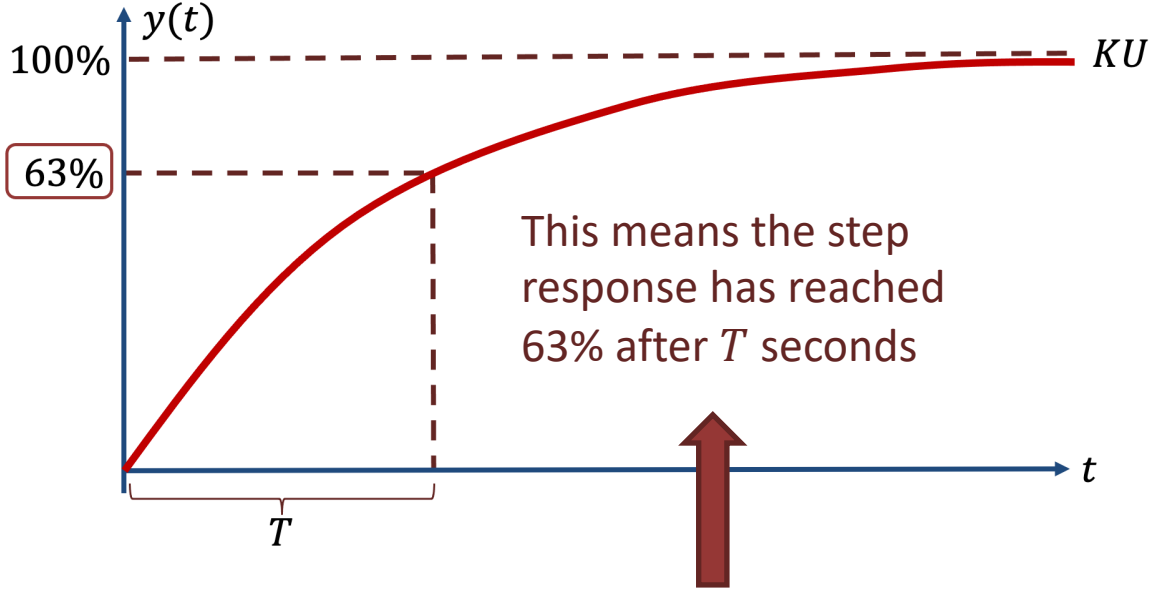


Why $T = 63\%$?

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

We have:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$



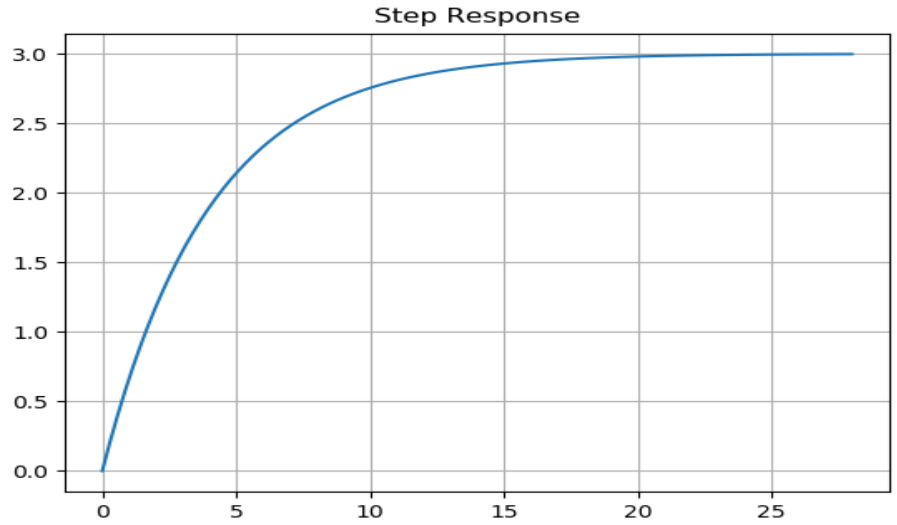
We set $t = T$:

$$y(t) = KU\left(1 - e^{-\frac{T}{T}}\right) = KU(1 - e^{-1}) = KU(1 - 0.37) = 0.63 \cdot KU$$

Python

Transfer Function:

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




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

K = 3
T = 4

num = np.array ([K])
den = np.array ([T , 1])

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

t, y = control.step_response(H)

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

1.order Transfer Functions

A 1.order transfer function is given by:

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

Where K is the Gain and T is the Time constant

In the time domain we get the following equation (using Inverse Laplace):

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

(After a Step U for the input signal $u(s)$)

Time Constant T

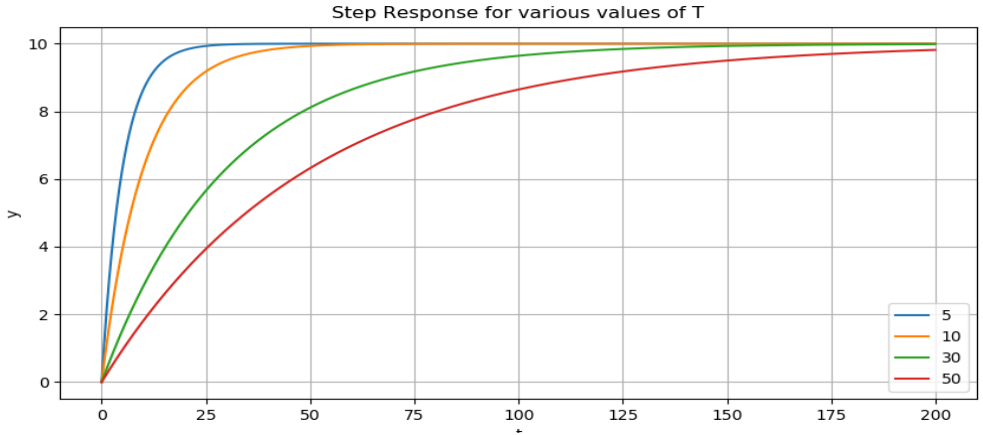
Transfer Function:

$$H(s) = \frac{y(s)}{u(s)} = \frac{10}{Ts + 1}$$

Step Response:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

We try with different values for T



Conclusion:

Larger T → Slower System

Smaller T → Faster System

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

K = 10
Tarray = [5, 10, 30, 50]

start = 0
stop = 200
step = 0.1
t = np.arange(start, stop, step)

for T in Tarray:
    #Create Transfer Function
    num = np.array ([K])
    den = np.array ([T , 1])

    H = control.tf(num , den)

    print ('H(s) =', H)
    # Step Response
    t, y = control.step_response(H, t)
    # Plot
    plt.plot(t, y)

plt.title("Step Response for different T")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(Tarray)
plt.grid()
plt.show()
```

Gain K

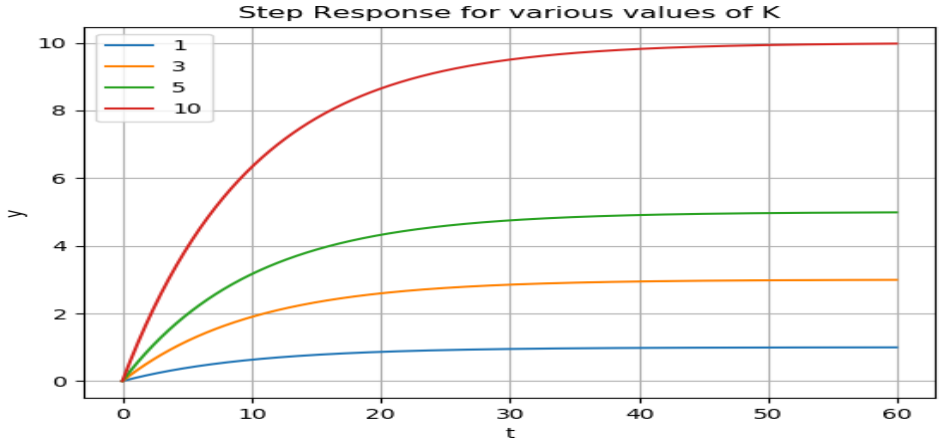
Transfer Function;

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{10s + 1}$$

Step Response:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

We try with different values for K



Steady State Response: $y_s = \lim_{t \to \infty} y(t) = KU$

(We used $U = 1$ in the simulations)

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

T = 10
Karray = [1, 3, 5, 10]

start = 0
stop = 60
step = 0.1
t = np.arange(start, stop, step)

for K in Karray:
    #Create Transfer Function
    num = np.array ([K])
    den = np.array ([T , 1])

    H = control.tf(num , den)

    print ('H(s) =', H)
    # Step Response
    t, y = control.step_response(H, t)
    # Plot
    plt.plot(t, y)

plt.title("Step Response for different K")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(Karray)
plt.grid()
plt.show()
```

Integrator

Hans-Petter Halvorsen

Integrator

A Transfer Function for an Integrator is given by:

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

Where K is the Gain

Example of an Integrator:
A Water/Liquid Tank

In the time domain we get the following equation (using Inverse Laplace):

$$H(s) = \frac{K}{s} \quad \Rightarrow \quad y(t) = KUt$$

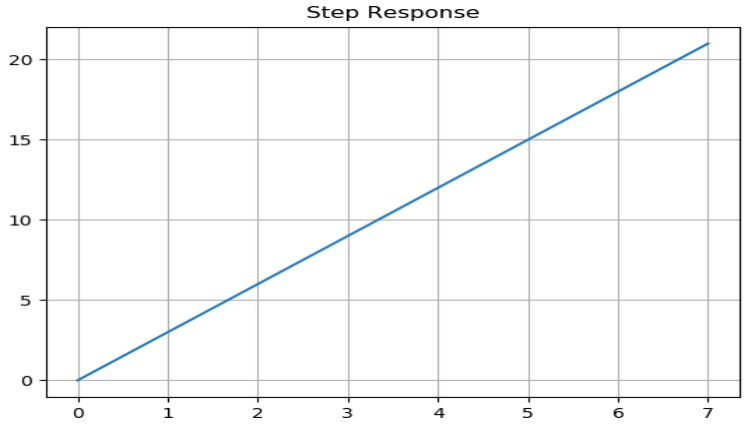
(After a Step U for the unput signal $u(s)$)

Python

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

We set $K = 3$ in this example:

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




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

K = 3

num = np.array ([K])
den = np.array ([1, 0])

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

t, y = control.step_response(H)

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

Time Delay

Hans-Petter Halvorsen

Time Delay

Transfer Function for Time Delay:

$$H(s) = \frac{y(s)}{u(s)} = e^{-\tau s}$$

1. order Transfer Function with Time Delay:

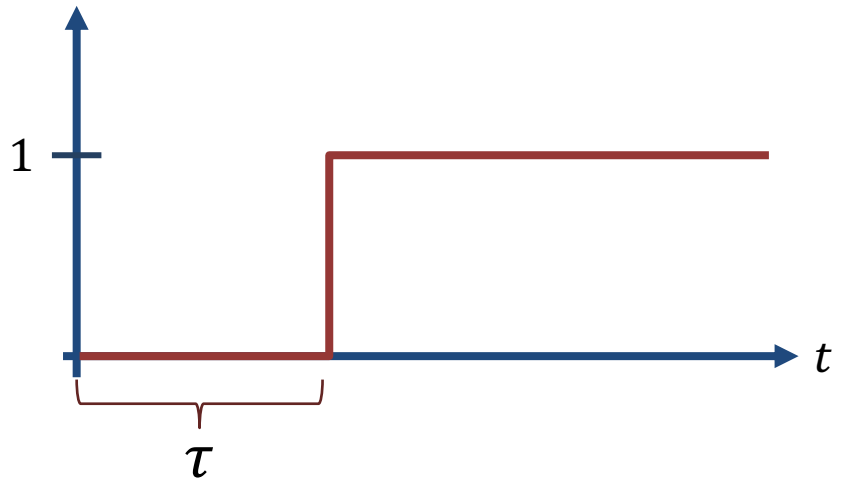
$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts + 1} e^{-\tau s}$$

Time Delay

Transfer Function for Time Delay:

$$H(s) = e^{-\tau s}$$

Step Response for Time Delay:



Padé Approximation

Transfer Function for Time Delay: $H(s) = e^{-\tau s}$

In some situations, it is necessary to substitute $e^{-\tau s}$ with an approximation, e.g., the Padé Approximation:

$$e^{-\tau s} \approx \frac{1 - k_1 s + k_2 s^2 + \cdots \pm k_n s^n}{1 + k_1 s + k_2 s^2 + \cdots + k_n s^n}$$

Where n is the order of the approximation and $k_1, k_2, \ldots$ are constants

1.order Padé Approximation:

$$e^{-\tau s} \approx \frac{1 - k_1 s}{1 + k_1 s} = \frac{-k_1 s + 1}{k_1 s + 1}$$

Where $k_1 = \frac{\tau}{2}$

etc.

2.order Padé Approximation:

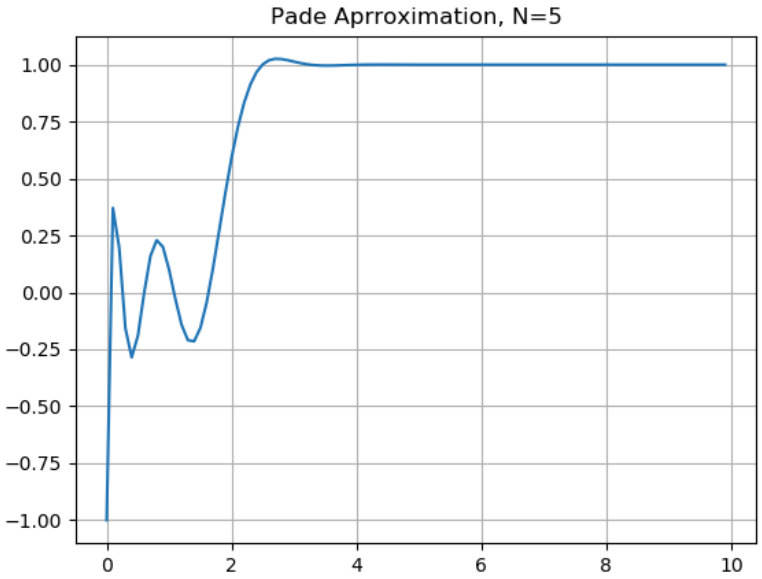
$$e^{-\tau s} \approx \frac{1 - k_1 s + k_2 s^2}{1 + k_1 s + k_2 s^2}$$

Where $k_1 = \frac{\tau}{2}$ and $k_2 = \frac{\tau^2}{12}$

Padé Approx.

$$H(s) = e^{-2s}$$

We can use the **pade()** Function:



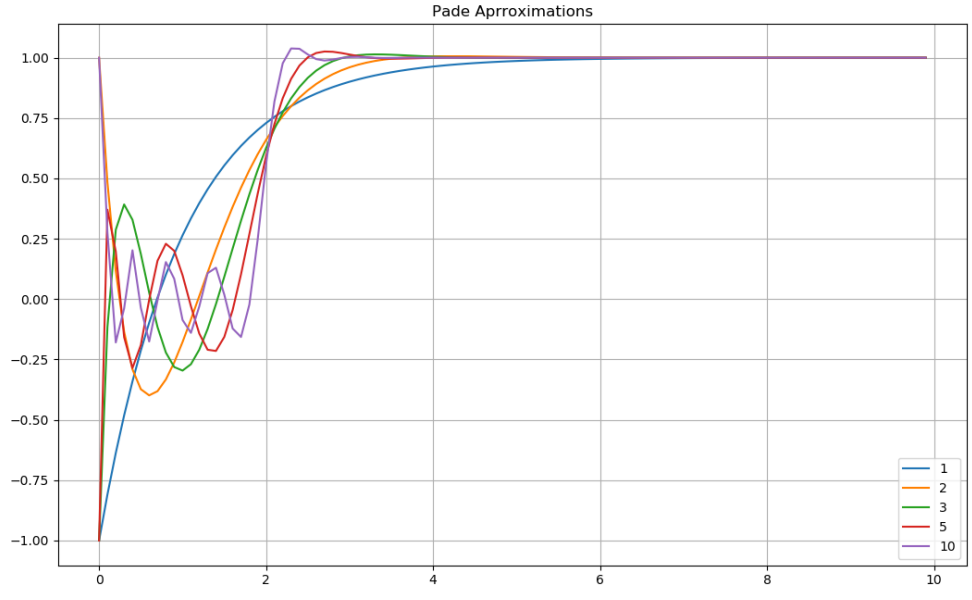

```
import numpy as np
import matplotlib.pyplot as plt
import control

# Time Delay
Tau = 2

# Approximation Order
N = 5

[num_pade,den_pade] = control.pade(Tau,N)
Hpade = control.tf(num_pade,den_pade)
print ('Hpade(s) =', Hpade)


start = 0
stop = 10
step = 0.1

t = np.arange(start, stop, step)

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

plt.plot(t,y)
title = "Pade Aprroximation, N=" + str(N)
plt.title(title)
plt.grid()
```

If you want a more accurate approximation, you can increase the order N. Padé approximations with order $N > 10$ should be avoided.

Padé Approx.

$$H(s) = e^{-2s}$$

We plot Padé Approximations with different orders in the same Plot:




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

# Time Delay
Tau = 2

N = 5

start = 0
stop = 10
step = 0.1
t = np.arange(start, stop, step)

N = [1, 2, 3, 5, 10]

for n in N:
    [num_pade,den_pade] = control.pade(Tau,n)
    Hpade = control.tf(num_pade,den_pade)
    print ('Hpade(s) =', Hpade)

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

    plt.plot(t,y)

plt.title("Pade Aprroximations")
plt.legend(N)
plt.grid()
```

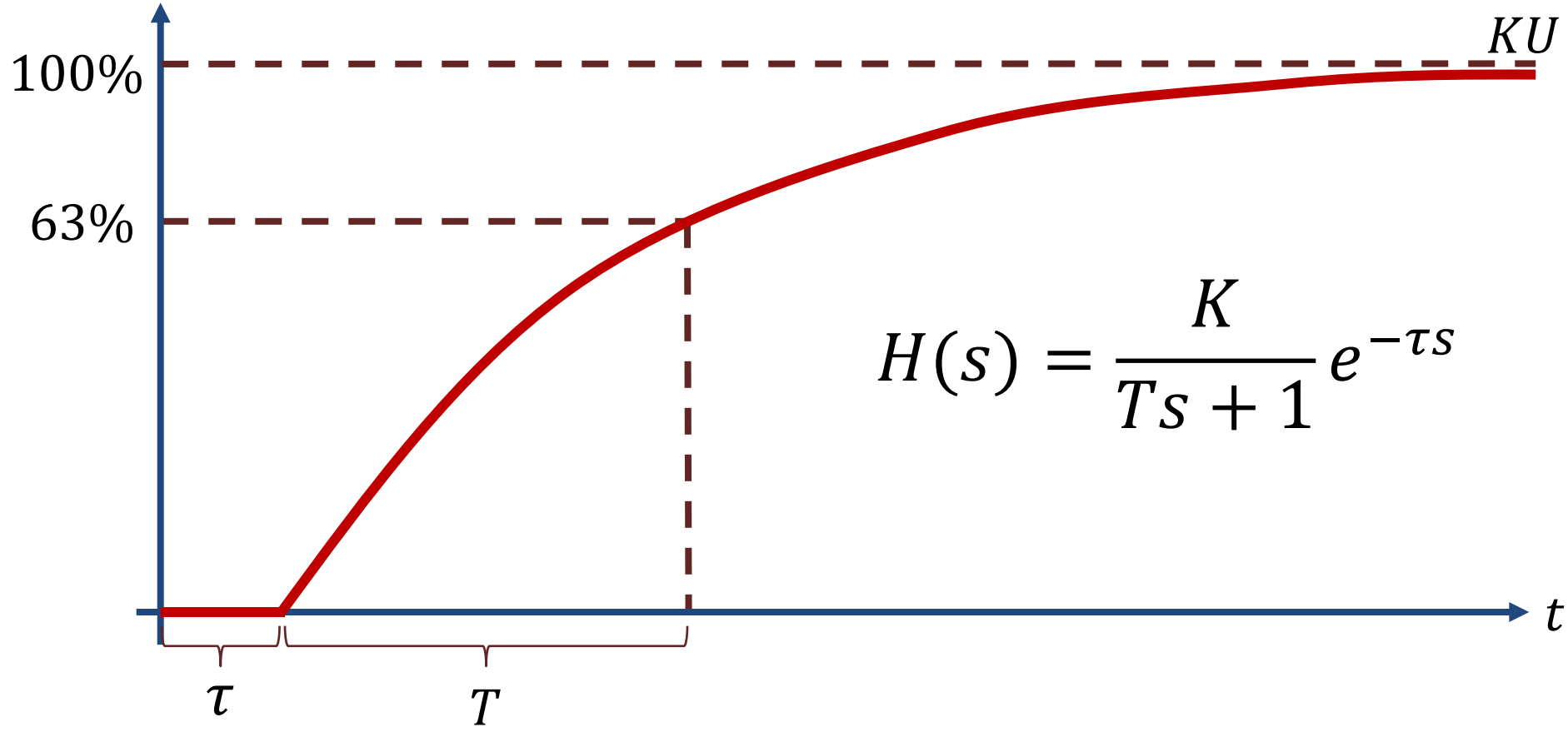
1. order with Time Delay

1. order Transfer Function with Time Delay:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

Where K is the Gain, T is the Time constant and τ is the Time Delay

1. order with Time Delay

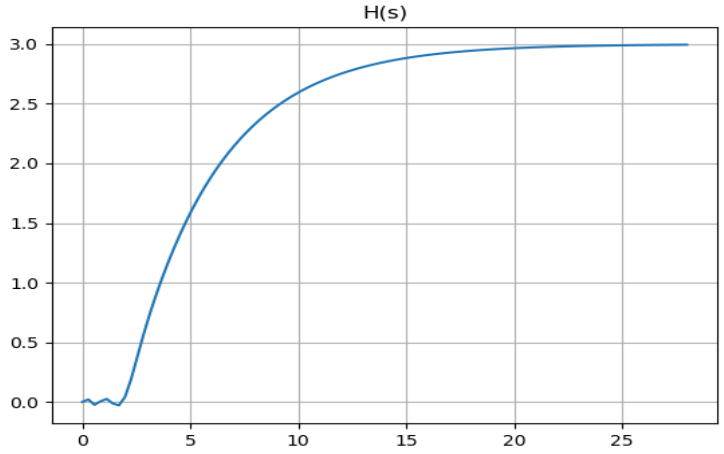


$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

Python

Transfer Function:

$$H(s) = \frac{3}{4s+1} e^{-2s}$$




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

K = 3
T = 4

num = np.array ([K])
den = np.array ([T , 1])

H1 = control.tf(num , den)
print ('H1(s) =', H1)

Tau = 2
N = 5 # Order of the Approximation
[num_pade,den_pade] = control.pade(Tau,N)
Hpade = control.tf(num_pade,den_pade)
print ('Hpade(s) =', Hpade)

H = control.series(H1, Hpade)
print ('H(s) =', H)

t, y = control.step_response(H)

plt.plot(t,y)
plt.title("H(s)")
plt.grid()
```

2.order Transfer Function

Hans-Petter Halvorsen

2.order Transfer Functions

2. order Transfer Function can be given on the following form:

$$H(s) = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$$

Or like this:

$$H(s) = \frac{K}{as^2 + bs + c}$$

Where T_1 and T_2 are Time Constants

Or like this:

$$H(s) = \frac{K{\omega_0}^2}{s^2 + 2\zeta\omega_0 s + {\omega_0}^2} = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta\frac{s}{\omega_0} + 1}$$

K is the gain
ζ zeta is the relative damping factor
ω_0[rad/s] is the undamped resonance frequency

2.order Transfer Functions

Special case: When $\zeta > 0$ and the poles are real and distinct we have:

$$H(s) = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$$

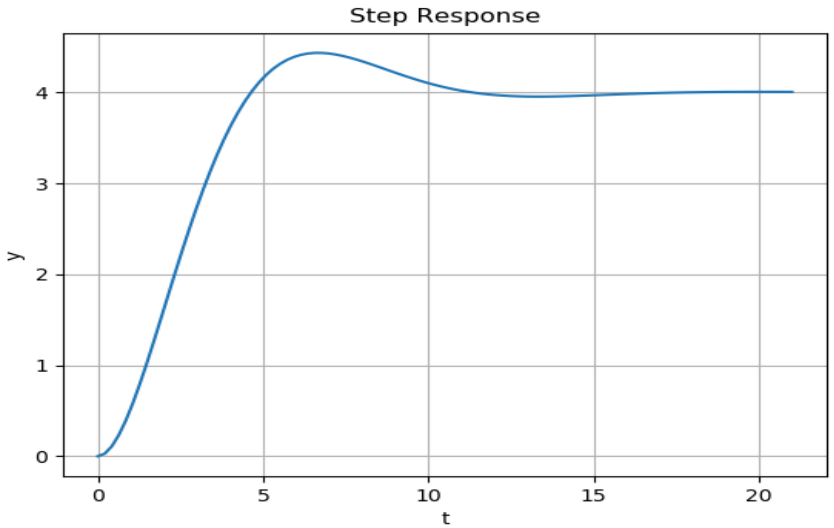
We see that this system can be considered as two 1.order systems in series:

$$H(s) = H_1(s)H_1(s) = \frac{K}{(T_1 s + 1)} \cdot \frac{1}{(T_2 s + 1)} = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$$

Python

SciPy.signal

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

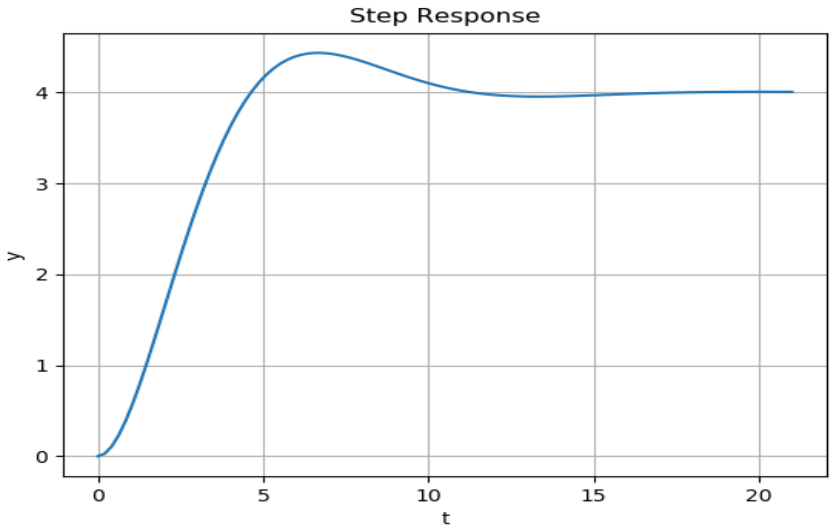



```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define Transfer Function
num = np.array([4])
den = np.array([3 , 2, 1])

H = signal.TransferFunction(num , den)

print ('H(s) =', H)

# Step Response
t, y = signal.step(H)

# Plotting
plt.plot(t, y)
plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()
```

Python

The Python Control Systems Library

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




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

# Define Transfer Function
num = np.array([4])
den = np.array([3 , 2, 1])

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

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

# Plotting
plt.plot(t, y)
plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()
```

Poles and Zeros

Hans-Petter Halvorsen

Transfer Functions

A general transfer function can be written on the following general form:

$$H(s) = \frac{numerator(s)}{denominator(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1\, s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1\, s + a_0}$$

The **Numerators** of transfer function models describe the locations of the **Zeros** of the system, while the **Denominators** of transfer function models describe the locations of the **Poles** of the system.

Transfer Functions

A general transfer function can be written on the following general form:

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

$$H(s) = \frac{Zeros}{Poles}$$

Example:

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

Zeros:

$s + 1 = 0$ $\quad z_1 = -1$

$2s + 1 = 0$ $\quad z_2 = -0.5$

Poles:

$3s + 1 = 0$ $\quad p_1 = -0.33$

$4s + 1 = 0$ $\quad p_2 = -0.25$

Python

Transfer Function:

$$H(s) = \frac{(s+1)(2s+1)}{(3s+1)(4s+1)} \Rightarrow H(s) = \frac{(s+1)}{(3s+1)} \cdot \frac{(2s+1)}{(4s+1)}$$

Alternatively we can multiply:

$$H(s) = \frac{2s^2 + s + 2s + 1}{12s^2 + 3s + 4s + 1} = \frac{2s^2 + 3s + 1}{12s^2 + 7s + 1}$$

```
num = np.array([2, 3, 1])
den = np.array([12 , 7, 1])
H = control.tf(num , den)
```

Or we can use the **`np.convolve()`** function:

```
num1 = np.array([1, 1])
num2 = np.array([2, 1])
num = np.convolve(num1, num2)

den1 = np.array([3 , 1])
den2 = np.array([4 , 1])
den = np.convolve(den1, den2)

H = control.tf(num , den)
```

```
import numpy as np
import control

num = np.array([1, 1])
den = np.array([3 , 1])
H1 = control.tf(num , den)

num = np.array([2, 1])
den = np.array([4 , 1])
H2 = control.tf(num , den)

H = control.series(H1,H2)
print ('H(s) =', H)

z = control.zero(H)
print ('z =', z)

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

control.pzmap(H)
```

Results

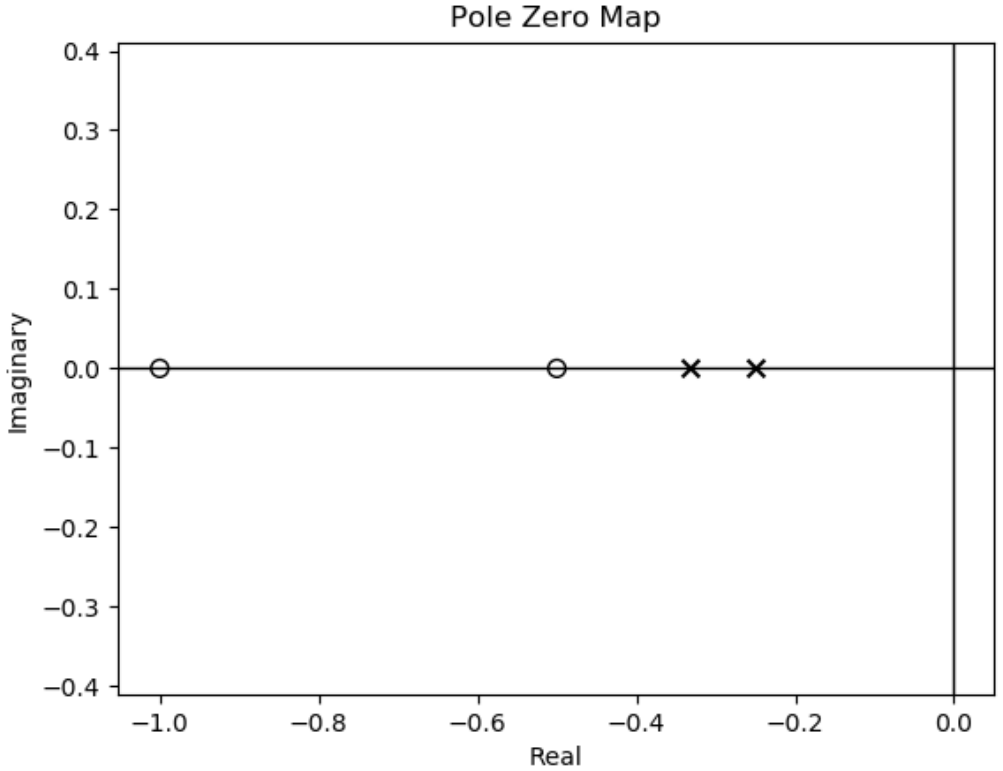
$$H(s) = \frac{(s+1)(2s+1)}{(3s+1)(4s+1)}$$

$z_1 = -1$ $\quad p_1 = -0.33$

$z_2 = -0.5$ $\quad p_2 = -0.25$

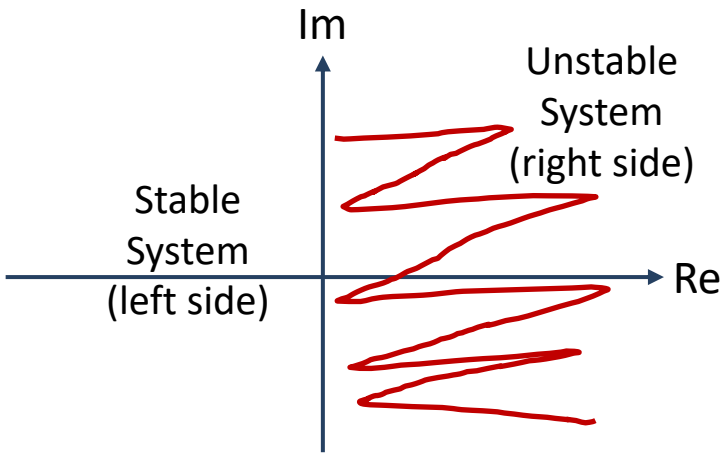
```
z = control.zero(H)
z = [-1.  -0.5]

p = control.pole(H)
p = [-0.33333333 -0.25]
```

```
control.pzmap(H)
```



Poles and Stability of the System

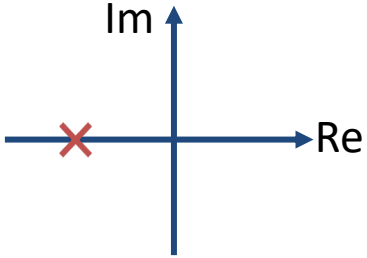
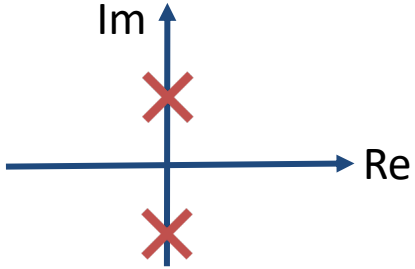
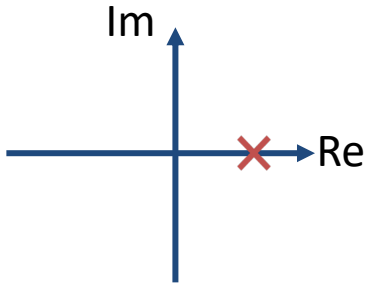
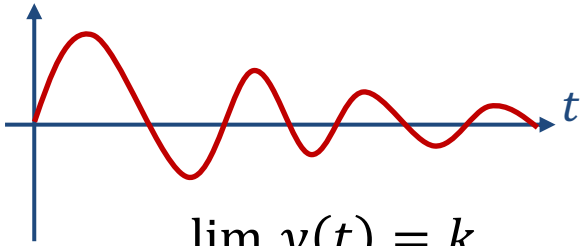
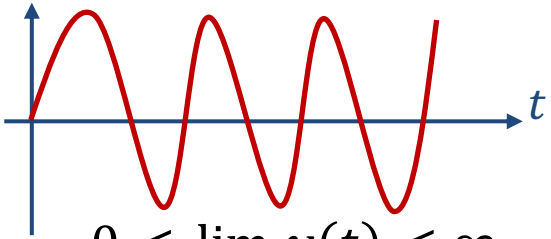
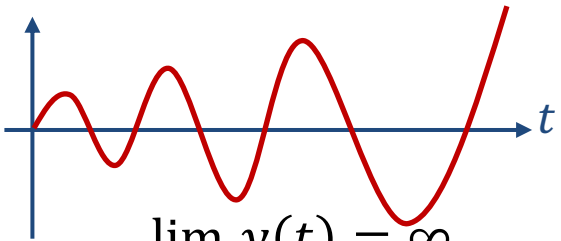
The poles are important when analyzing the stability of a system. The Figure below gives an overview of the poles impact on the stability of a system.



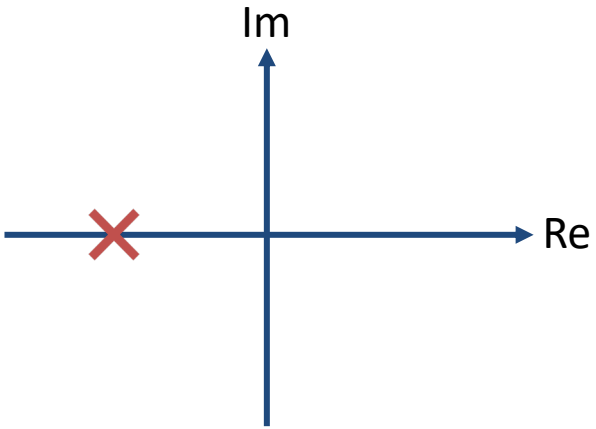
We have 3 different Alternatives:

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

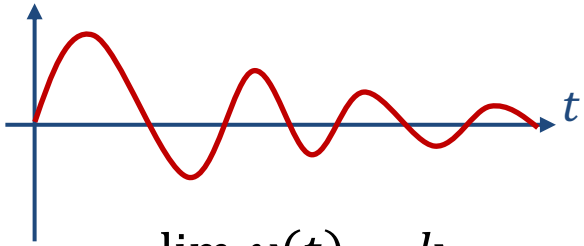
Poles and Stability

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

Asymptotically Stable System



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



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

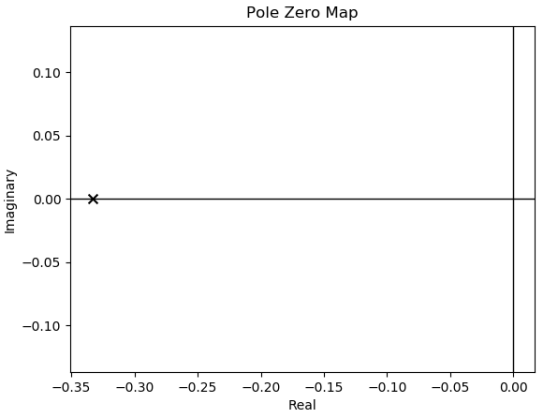
Python

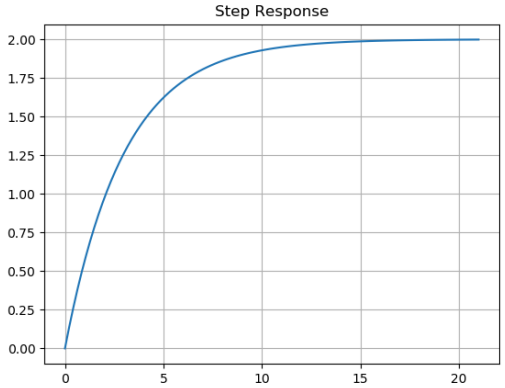
Asymptotically Stable System

Transfer Function:

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

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

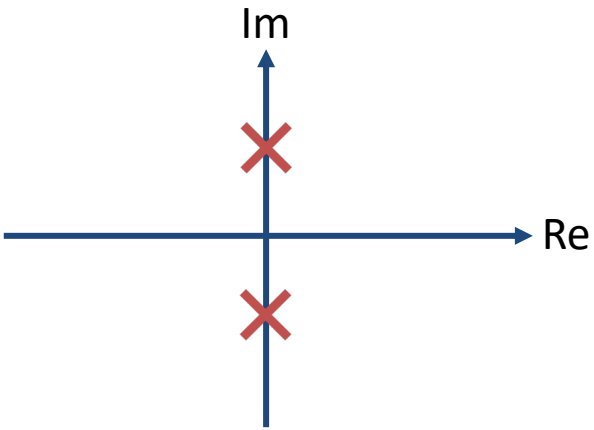




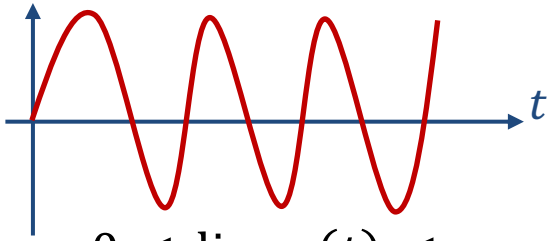
$$\lim_{t \to \infty} y(t) = k$$

```
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.



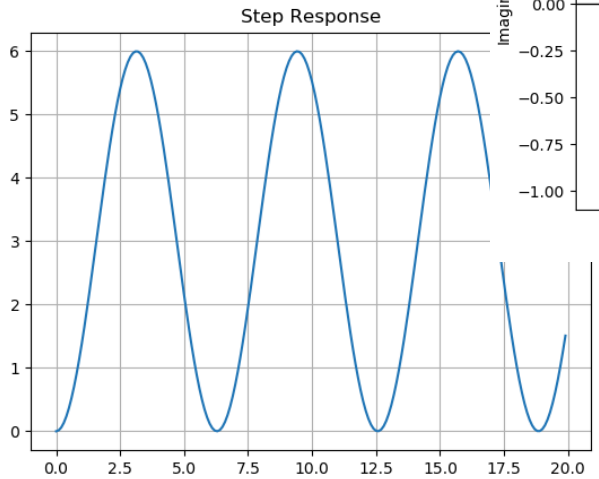
$$0 < \lim_{t \to \infty} y(t) < \infty$$

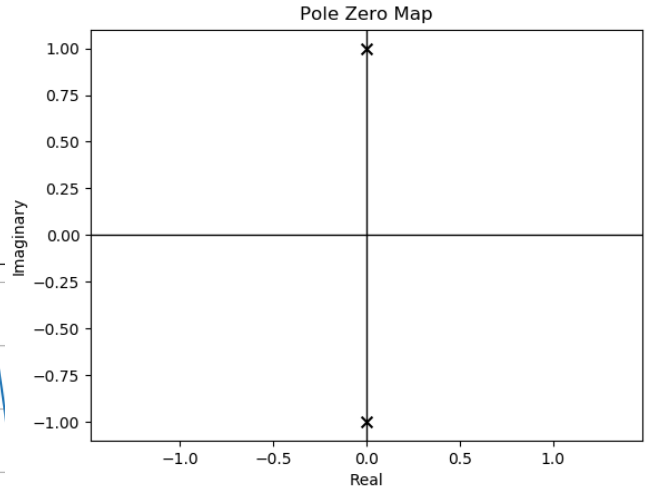
Python

Transfer Function:

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

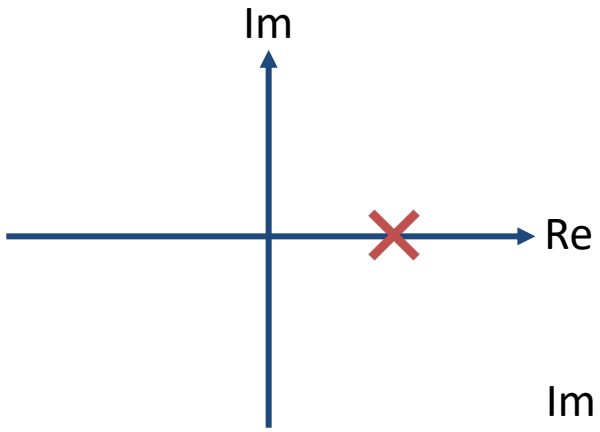
Marginally Stable System



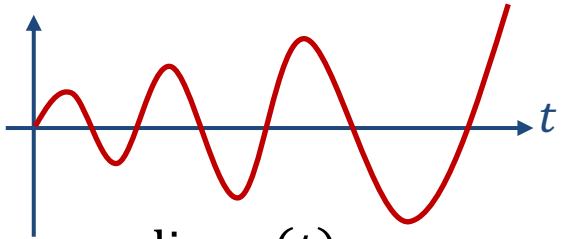



```
import control
import numpy as np
import matplotlib.pyplot as plt

# 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



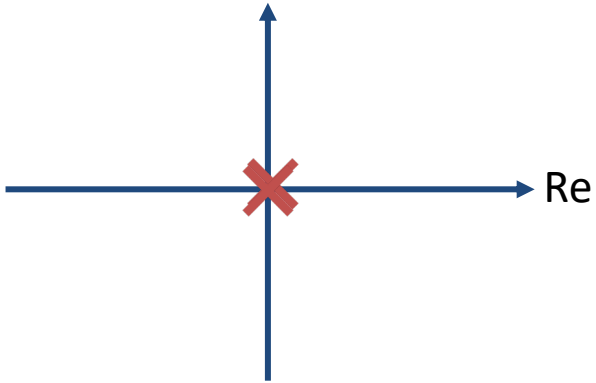
At least one pole lies in the right half plane (has real part greater than zero).



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

Im

or



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

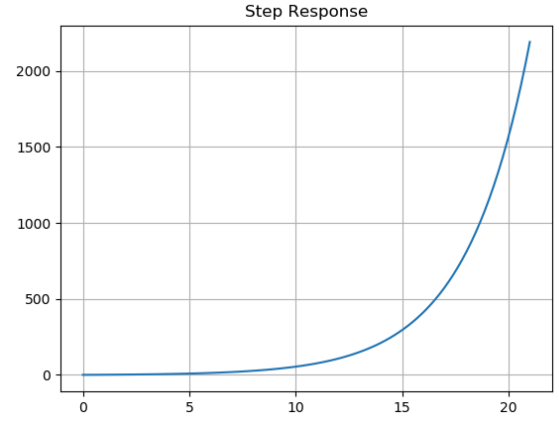
Example: double integrator $H(s) = \frac{1}{s^2}$

Python

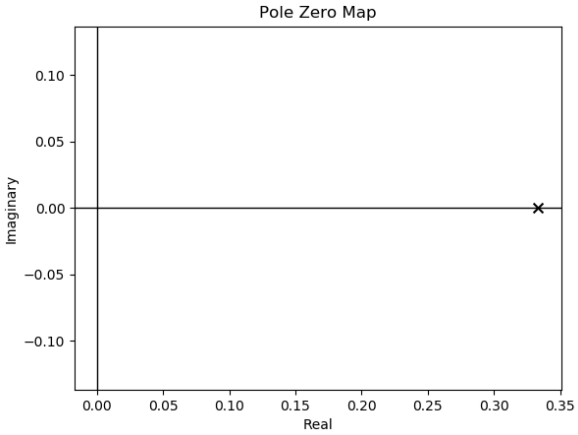
Unstable System

Transfer Function:

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



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



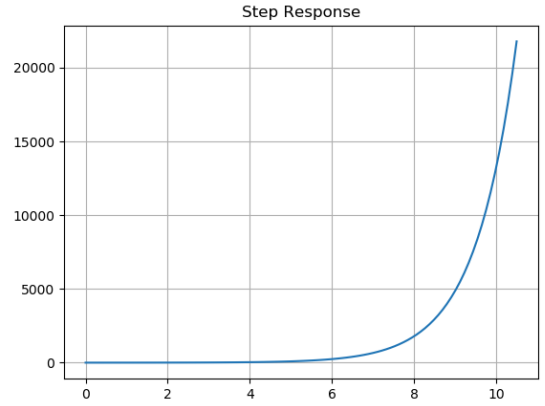

```
import control
import numpy as np
import matplotlib.pyplot as plt

# 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

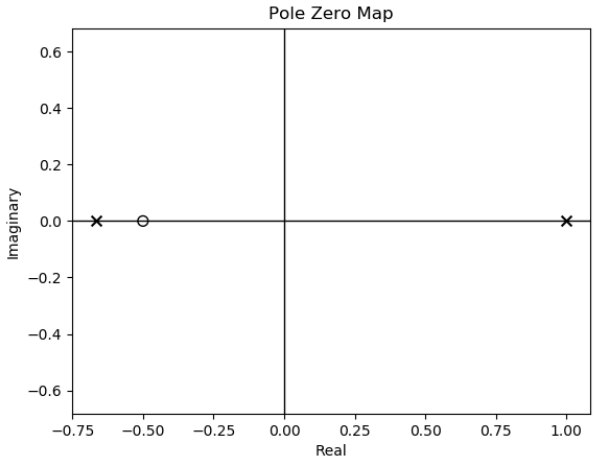
Unstable System

Transfer Function:

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



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



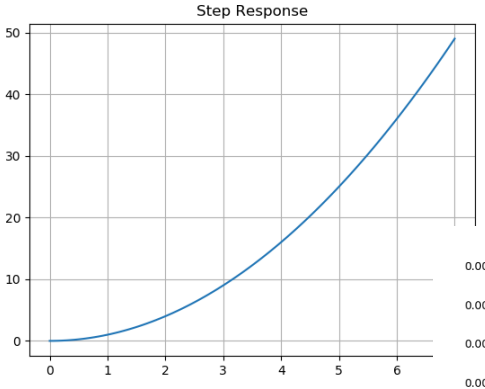

```
import control
import numpy as np
import matplotlib.pyplot as plt

# 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)
```

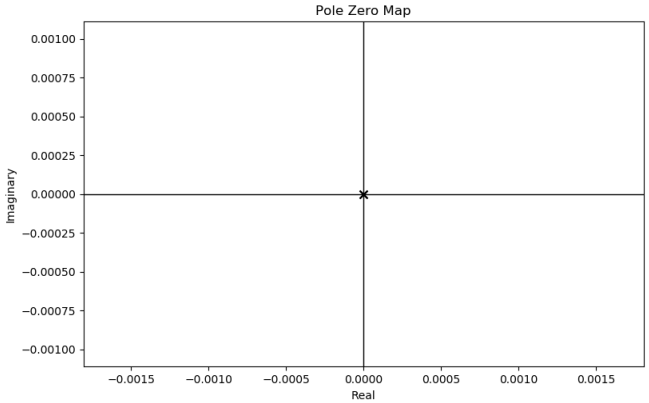
Python

Unstable System

Transfer Function:

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

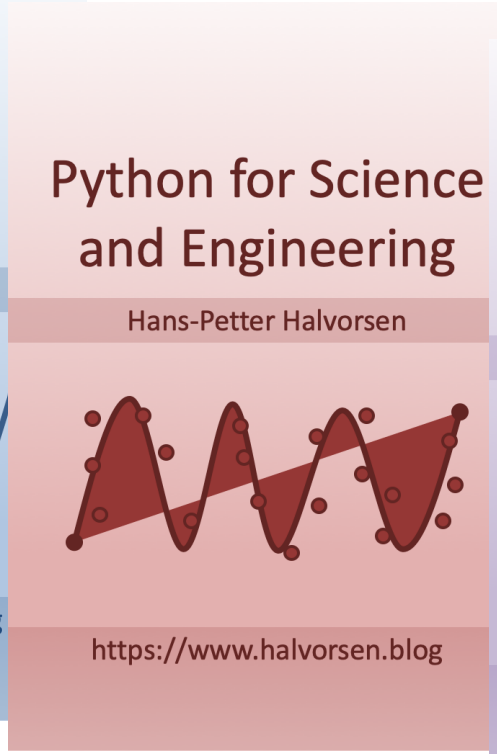


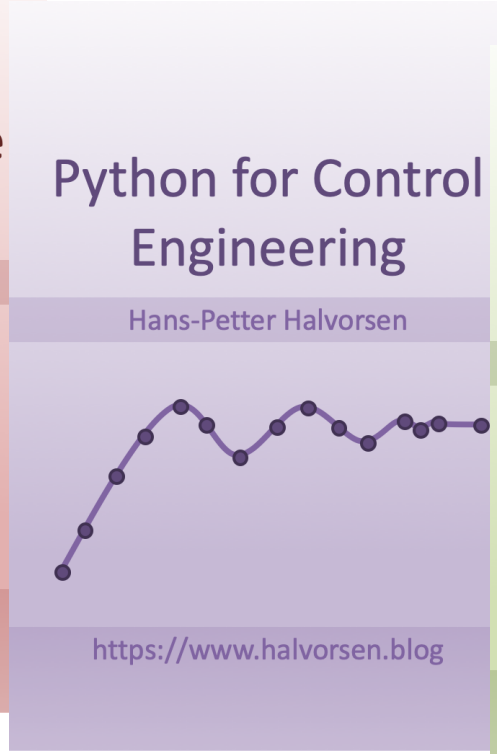


$$p_1 = 0, p_2 = 0$$

```
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)
```

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