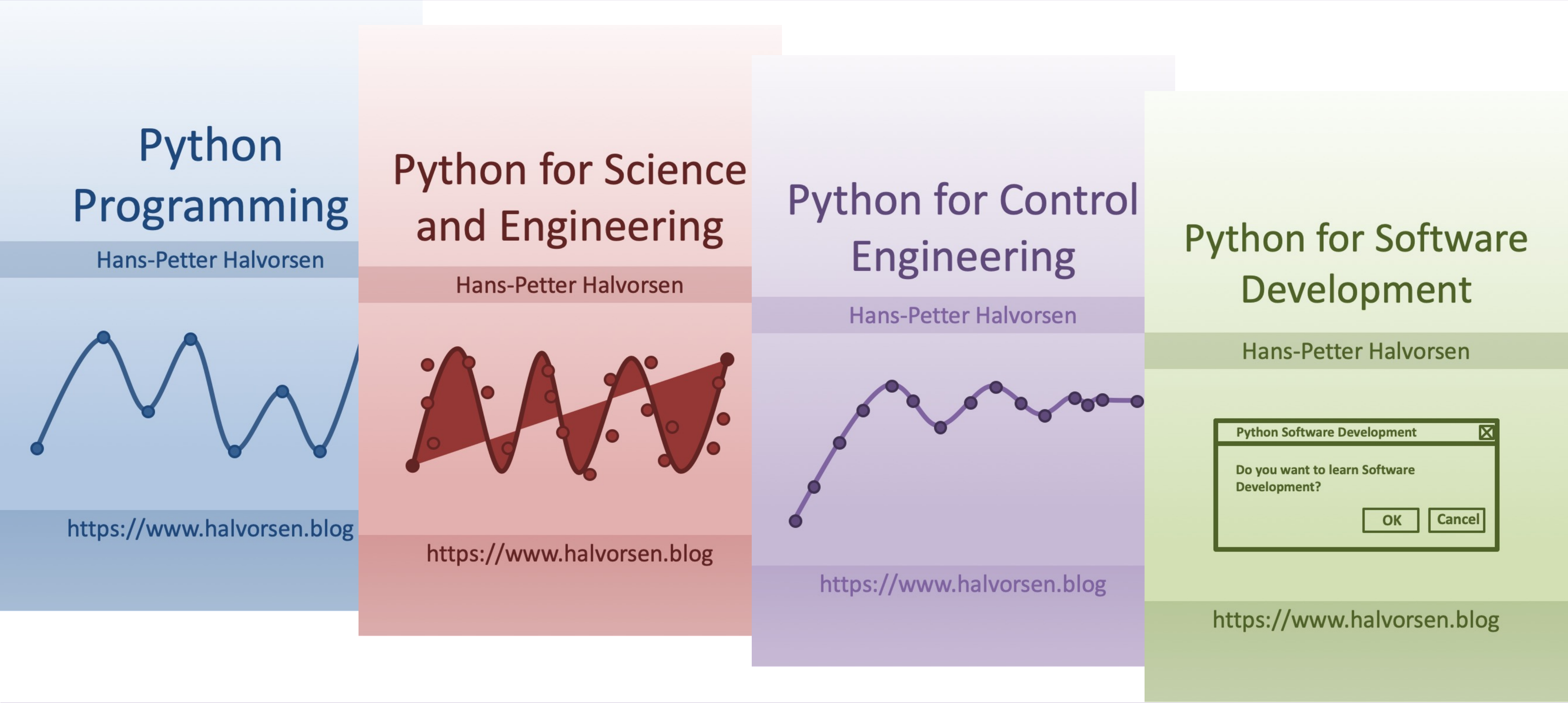
State Space Models with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

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

Additional Python Resources



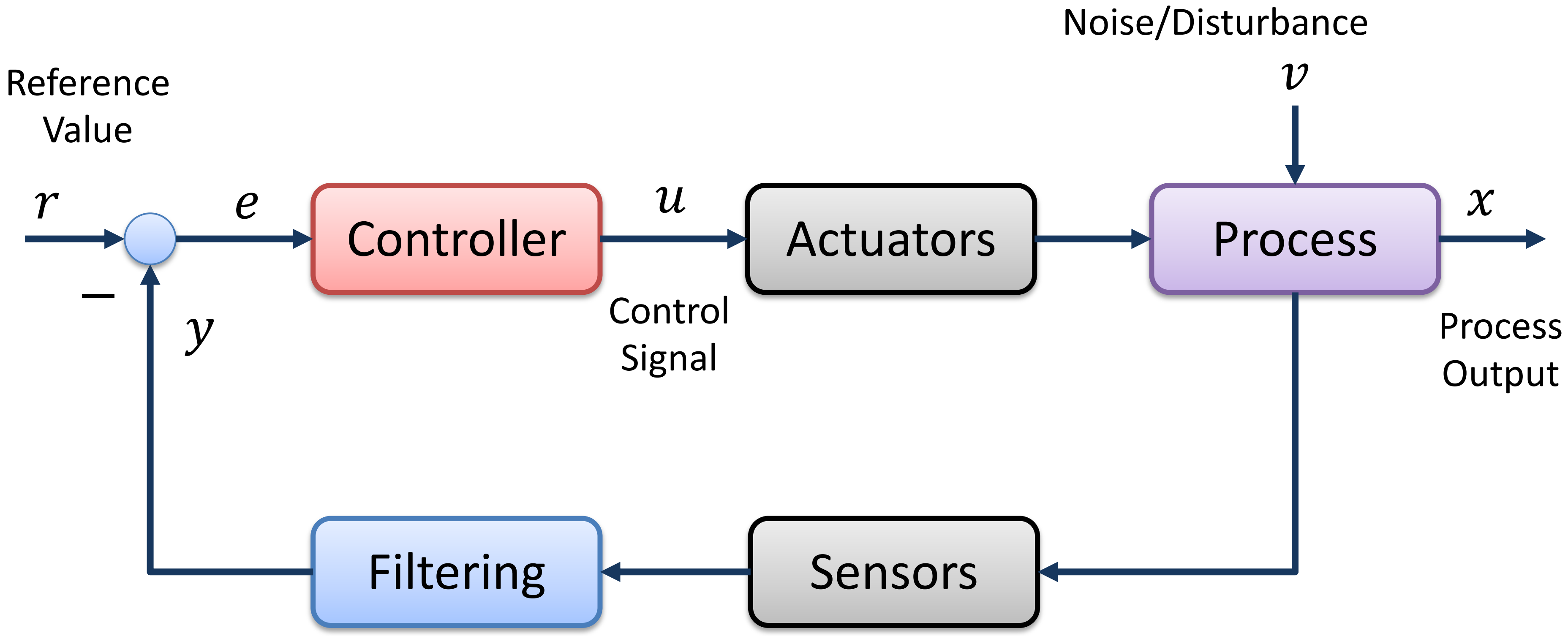
https://www.halvorsen.blog/documents/programming/python/

Contents

- Introduction to Control Systems
- State-space Models
  - State-space models are very useful in Control Theory and Design
- Python Examples
  - SciPy (SciPy.signal)
  - The Python Control Systems Library

It is recommended that you know about Vectors, Matrices and Linear Algebra. If not, take a closer look at my Tutorial “Linear Algebra with Python”. You should also know about differential equations, see “Differential Equations in Python“

Control System



The different blocks in the Control System can be, e.g., described as a Transfer Function or a State Space Model

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

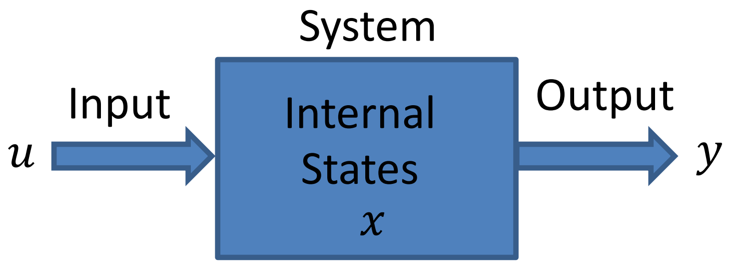
State Space Models

Hans-Petter Halvorsen

State-space Models

A general State-space Model is given by:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$



Note that $\dot{x}$ is the same as $\frac{dx}{dt}$

A, B, C and D are matrices

$x, \dot{x}, u, y$ are vectors

A state-space model is a structured form or representation of a set of differential equations. State-space models are very useful in Control theory and design. The differential equations are converted in matrices and vectors.

State-space Models

Assume we have the following linear equations:

$$\dot{x}_1 = a_{11}x_1 + a_{21}x_2 + \cdots + a_{n1}x_n + \ b_{11}u_1 + b_{21}u_2 + \cdots + b_{n1}u_n$$

$$\vdots$$

$$\dot{x}_n = a_{1m}x_1 + a_{2m}x_2 + \cdots + a_{nm}x_n + \ b_{1m}u_1 + b_{2m}u_2 + \cdots + b_{n1}u_n$$

$$\vdots$$

We can set the system on matrix/vector form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1m} & \cdots & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{n1} \\ \vdots & \ddots & \vdots \\ c_{1m} & \cdots & c_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & \cdots & d_{n1} \\ \vdots & \ddots & \vdots \\ d_{1m} & \cdots & d_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

State-space Models

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1m} & \cdots & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{n1} \\ \vdots & \ddots & \vdots \\ c_{1m} & \cdots & c_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & \cdots & d_{n1} \\ \vdots & \ddots & \vdots \\ d_{1m} & \cdots & d_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

This gives the following compact form of a general linear State-space model:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Where A, B, C and D are matrices

Example

Given the following System:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -x_2 + u$$

$$y = x_1$$

This gives the following State-space Model:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Example

Given the following System:

$$\dot{x}_1 = x_2$$

$$2\dot{x}_2 = -2x_1 - 6x_2 + 4u_1 + 8u_2$$

$$y = 5x_1 + 6x_2 + 7u_1$$

We can reformulate:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -x_1 - 3x_2 + 2u_1 + 4u_2$$

$$y = 5x_1 + 6x_2 + 7u_1$$

This gives the following State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix}$$

$$C = \begin{bmatrix} 5 & 6 \end{bmatrix} \qquad D = \begin{bmatrix} 7 & 0 \end{bmatrix}$$

Example

Given the following System:

$$\dot{x}_1 = 2x_1 + 3x_3 + 7u_1$$

$$\dot{x}_2 = 4x_1 + 5u_2$$

$$\dot{x}_3 = 8x_3$$

$$y_1 = 6x_3$$

$$y_2 = 3x_1 + 3x_3 + 7u_1$$

This gives the following State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 4 & 0 & 0 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 7 & 0 \\ 0 & 5 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 6 \\ 3 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 0 & 3 \\ 4 & 0 & 0 \\ 0 & 0 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} 7 & 0 \\ 0 & 5 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 6 \\ 3 & 0 & 3 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \\ 7 & 0 \end{bmatrix}$$

https://www.halvorsen.blog

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”)

→ You can create, manipulate and simulate State Space Models with both these Python Libraries

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 Example

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Python Code:

```
import scipy.signal as signal

A = [[0, 1],
     [0, -1]]

B = [[0],
     [1]]

C = [[1, 0]]

D = 0

sys = signal.StateSpace(A, B, C, D)
```

Python Example

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 2 & 4\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}5 & 6\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}7 & 0\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

```
import scipy.signal as signal

A = [[0, 1], [-1, -3]]
B = [[0, 0], [2, 4]]
C = [[5, 6]]
D = [[7, 0]]

sys = signal.StateSpace(A, B, C, D)
```

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}2 & 0 & 3\\ 4 & 0 & 0\\ 0 & 0 & 8\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}7 & 0\\ 0 & 5\\ 0 & 0\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$\begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}0 & 0 & 6\\ 3 & 0 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0 & 0\\ 7 & 0\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

```
import scipy.signal as signal

A = ..
B = ..
C = ..
D = ..

sys = signal.StateSpace(A, B, C, D)
```

Step Response

We have the differential equations:

$$\dot{x}_1 = \frac{1}{T}(-x_1 + Ku)$$
$$\dot{x}_2 = 0$$

The State-space Model becomes:

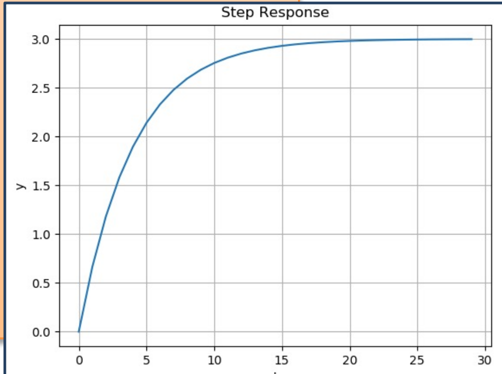
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{T} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{K}{T} \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Here we use the following function:

```
t, y = sig.step(sys, x0, t)
```

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

#Simulation Parameters
x0 = [0,0]

start = 0
stop = 30
step = 1
t = np.arange(start,stop,step)

K = 3
T = 4

# State-space Model
A = [[-1/T, 0],
     [0, 0]]
B = [[K/T],
     [0]]
C = [[1, 0]]
D = 0

sys = sig.StateSpace(A, B, C, D)

# Step Response
t, y = sig.step(sys, x0, t)

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



scipy.signal.lsim

We have the differential equations:

$$\dot{x}_1 = \frac{1}{T}(-x_1 + Ku)$$
$$\dot{x}_2 = 0$$

The State-space Model becomes:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{T} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{K}{T} \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

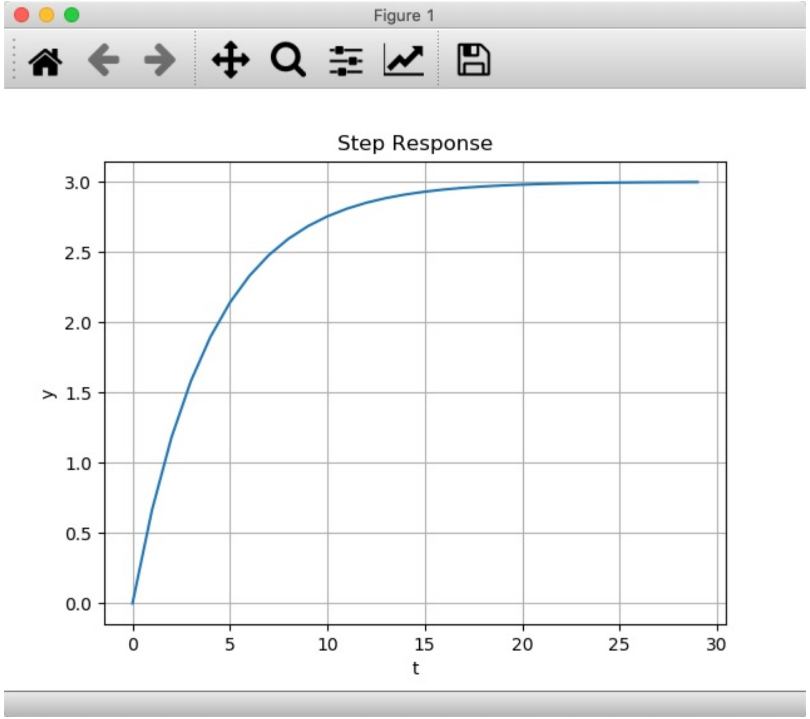
Here we use the following function:

`t, y, x = sig.lsim(sys, u, t, x0)`

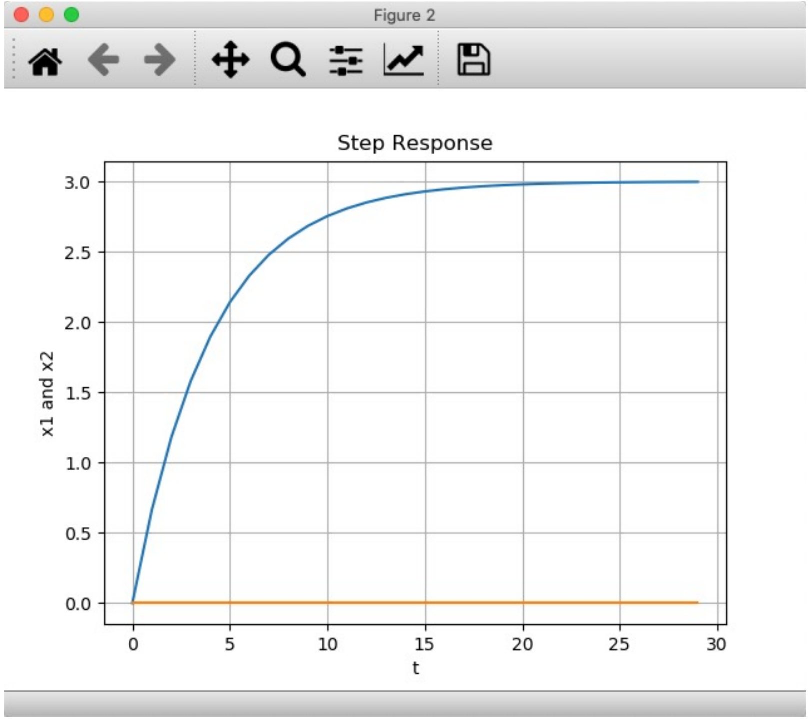
```
import scipy.signal as sig
import matplotlib.pyplot as plt
import numpy as np

#Simulation Parameters
x0 = [0,0]

start = 0
stop = 30
step = 1
t = np.arange(start,stop,step)

N = len(t)

u = np.ones(N)

K = 3
T = 4

# State-space Model
A = [[-1/T, 0],
     [0, 0]]

B = [[K/T],
     [0]]

C = [[1, 0]]

D = 0

sys = sig.StateSpace(A, B, C, D)

# Step Response
t, y, x = sig.lsim(sys, u, t)

# Plotting
plt.figure(1)
plt.plot(t, y)
plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()

# Alternatively you can plot one or more of the x variables
x1 = x[:, 0]
x2 = x[:, 1]

plt.figure(2)
plt.plot(t, x1, t, x2)
plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("x1 and x2")
plt.grid()
plt.show()
```

Results

Plotting y



Plotting x_1 and x_2



https://www.halvorsen.blog

Python Control Systems Library

Hans-Petter Halvorsen

Python Control Systems Library

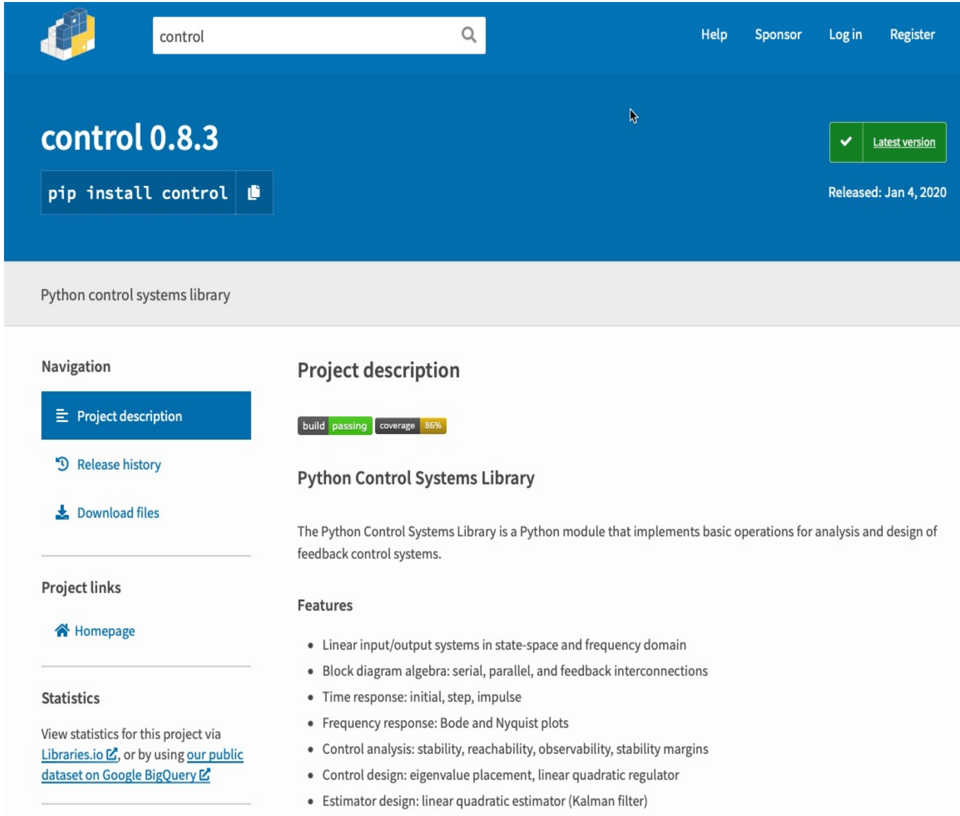
- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: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

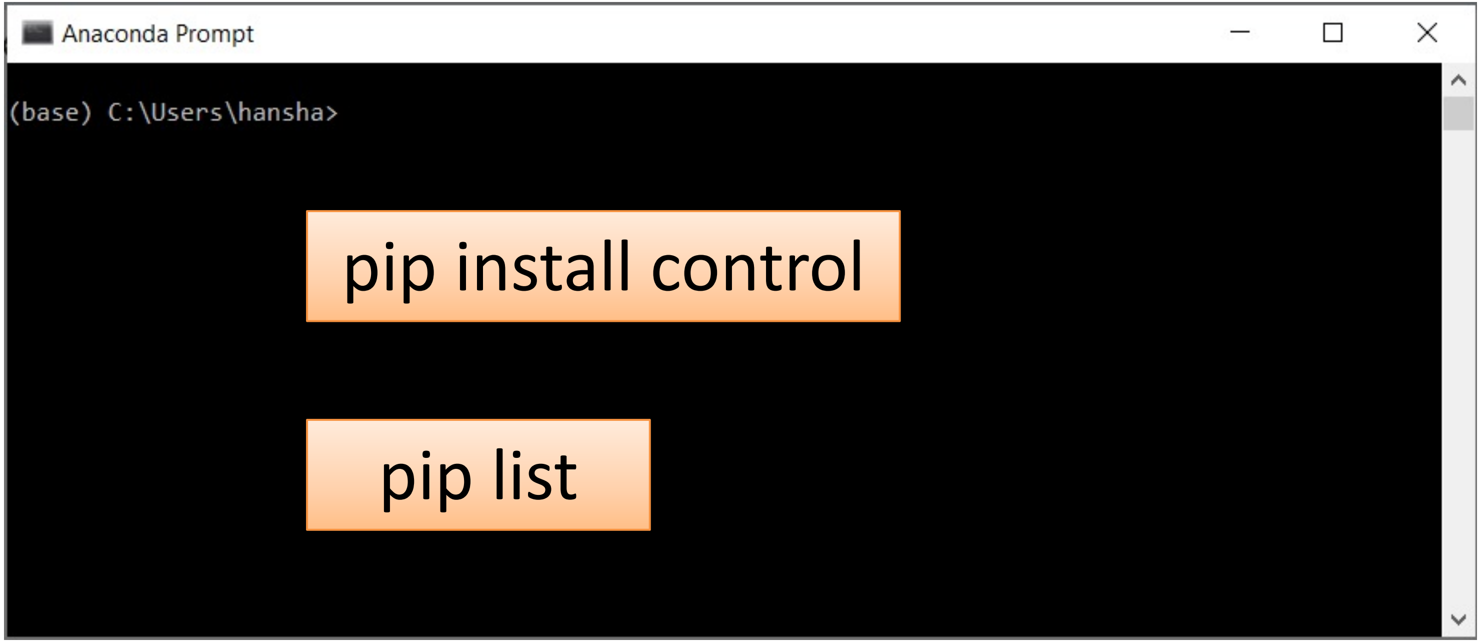
```
pip install control
```

- 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

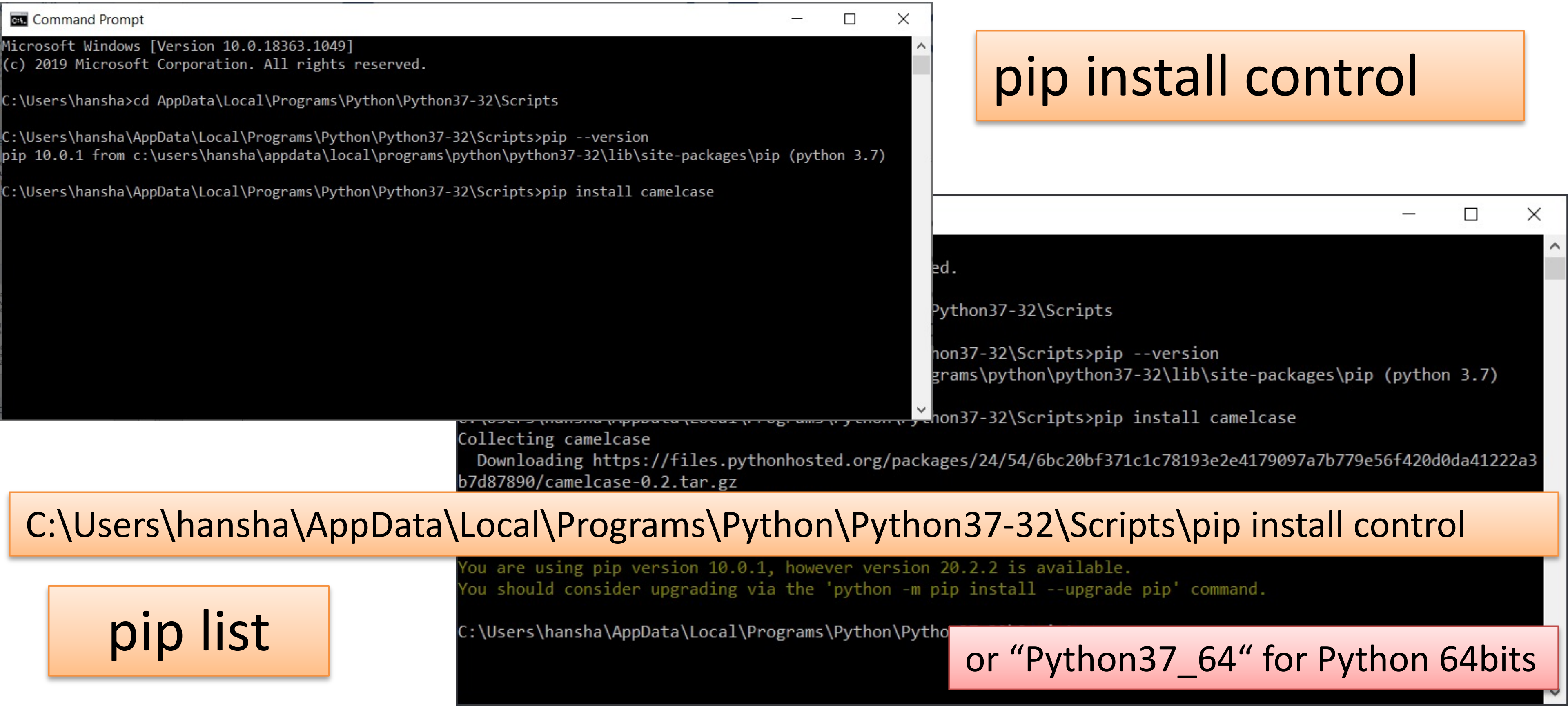


Anaconda Prompt

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

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
- …

Functions for Model Simulations:

- `step_response()` - Step response of a linear system (e.g., a State-space Model)
- `lsim()` - Simulate the output of a linear system (e.g., a State-space Model)

Python Example

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
import control

A = [[0, 1],
     [0, -1]]

B = [[0],
     [1]]

C = [[1, 0]]

D = 0

sys = control.ss(A, B, C, D)
```

Step Response

Here we use the following function:

`T,yout = step_response(sys, T, X0)`

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}u$$

$$y = \begin{bmatrix}5 & 6\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + [1]u$$

```
import control
import matplotlib.pyplot as plt

# Define State-space Model
A = [[0, 1], [-1, -3]]
B = [[1], [0]]
C = [[5, 6]]
D = [[1]]

ssmodel = control.ss(A, B, C, D)


# Step response for the system
t, y = control.step_response(ssmodel)

plt.plot(t, y)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()
```

This function uses the `forced_response()` function with the input set to a unit step

Step Response

Code Not Working!!!!!

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

```
import matplotlib.pyplot as plt
import control

# Define State-space Model
A = [[0, 1], [-1, -3]]
B = [[0, 0], [2, 4]]
C = [[5, 6]]
D = [[7, 0]]

ssmodel = control.ss(A, B, C, D)
print(ssmodel)

t, y = control.step_response(ssmodel)
plt.plot(t, y)
```

Note! This is a MISO system (Multiple Input/Single Output). So, the Solution is to split it into 2 systems, one for u_1 and one for u_2. Se next slides.

A similar code will work with MATLAB, and we will get 2 plots, but the Python Control package does not support that

Step Response1 (u_1)

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u_1$$

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [7] u_1$$

```
import matplotlib.pyplot as plt
import control

# Define State-space Model
A = [[0, 1], [-1, -3]]
B = [[0], [2]]
C = [[5, 6]]
D = [7]

ssmodel = control.ss(A, B, C, D)
print(ssmodel)

t, y = control.step_response(ssmodel)
plt.plot(t, y)
```

We can also plot both x_1 and x_2:

```
x1 = x[0 ,:]
x2 = x[1 ,:]

plt.plot(t, x1, t, x2)
```

Step Response2 (u_2)

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \end{bmatrix} u_2$$

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0] u_2$$

We can also plot both x_1 and x_2:

```
x1 = x[0 ,:]
x2 = x[1 ,:]

plt.plot(t, x1, t, x2)
```

```
import matplotlib.pyplot as plt
import control

# Define State-space Model
A = [[0, 1], [-1, -3]]
B = [[0], [4]]
C = [[5, 6]]
D = [0]

ssmodel = control.ss(A, B, C, D)
print(ssmodel)

t, y = control.step_response(ssmodel)
plt.plot(t, y)
```

State Space Models and Transfer Functions

Hans-Petter Halvorsen

SciPy.signal

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

LTI representations

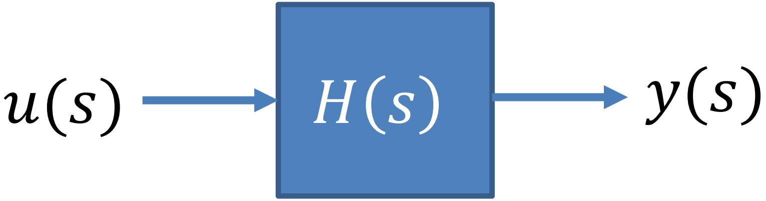
tf2zpk(b, a)	Return zero, pole, gain (z, p, k) representation from a numerator, denominator representation of a linear filter.
tf2sos(b, a[, pairing])	Return second-order sections from transfer function representation
tf2ss(num, den)	Transfer function to state-space representation.
zpk2tf(z, p, k)	Return polynomial transfer function representation from zeros and poles
zpk2sos(z, p, k[, pairing])	Return second-order sections from zeros, poles, and gain of a system
zpk2ss(z, p, k)	Zero-pole-gain representation to state-space representation
ss2tf(A, B, C, D[, input])	State-space to transfer function.
ss2zpk(A, B, C, D[, input])	State-space representation to zero-pole-gain representation.
sos2zpk(sos)	Return zeros, poles, and gain of a series of second-order sections
sos2tf(sos)	Return a single transfer function from a series of second-order sections
cont2discrete(system, dt[, method, alpha])	Transform a continuous to a discrete state-space system.
place_poles(A, B, poles[, method, rtol, maxiter])	Compute K such that eigenvalues (A - dot(B, K))=poles.

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 and s is the Laplace operator



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

SISO/MIMO Systems

We have 4 different Types of Systems:

- **SISO** – Single Input/Single Output
- **MISO** – Multiple Input/Single Output
- **SIMO** – Single Input/Multiple Output
- **MIMO** – Multiple Input/Multiple Output

SciPy Library

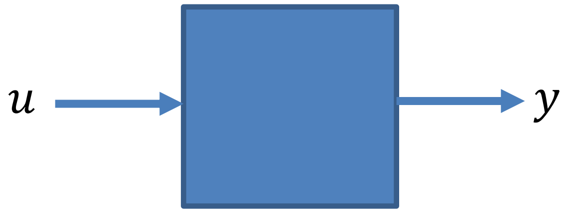
SISO

Single Input/Single Output

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [2]u$$



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

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

# SISO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[1],
    [0]]

C = [[1, 0]]

D = [[2]]

# Find Transfer Function from u to y
num, den = signal.ss2tf(A, B, C, D)
H = signal.TransferFunction(num, den)
print(H)

# Step response for the system
t, y = signal.step(H)

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

Control Library

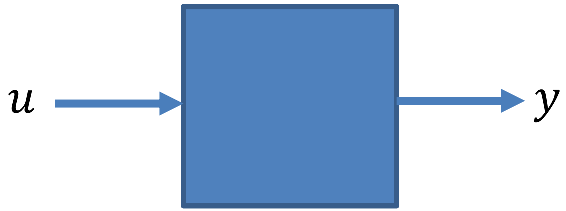
SISO

Single Input/Single Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}u$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + [2]u$$



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

```
import control
import matplotlib.pyplot as plt

# SISO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[1],
     [0]]

C = [[1, 0]]

D = [[2]]

ssmodel = control.ss(A, B, C, D)


H = control.ss2tf(ssmodel)
print(H)

# Step response for the system
t, y = control.step_response(H)

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

SciPy Library

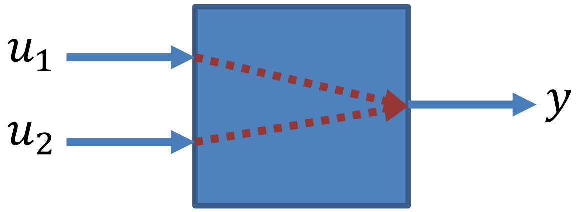
MISO

Multiple Input/Single Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 2 & 4\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



$$H_1(s) = \frac{y(s)}{u_1(s)} \qquad H_2(s) = \frac{y(s)}{u_2(s)}$$

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

# MISO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0, 0],
    [2, 4]]

C = [[1, 0]]


D = [[0, 0]]

# Find Transfer Function from u1 to y
num, den = signal.ss2tf(A, B, C, D, 0)
H1 = signal.TransferFunction(num, den)
print(H1)

# Find Transfer Function from u2 to y
num, den = signal.ss2tf(A, B, C, D, 1)
H2 = signal.TransferFunction(num, den)
print(H2)

# Step response for the system
t, y = signal.step(H1)
plt.plot(t, y)
t, y = signal.step(H2)
plt.plot(t, y)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2"])
plt.grid()
```

Control Library

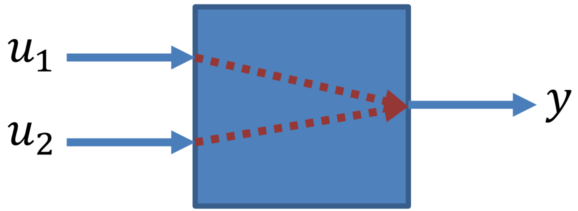
MISO

Multiple Input/Single Output

State-space Model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



$$H_1(s) = \frac{y(s)}{u_1(s)} \qquad H_2(s) = \frac{y(s)}{u_2(s)}$$

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

# MISO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0, 0],
    [2, 4]]

C = [[1, 0]]

D = 0

ssmodel = control.ss(A, B, C, D)

H = control.ss2tf(ssmodel)
print(H)
H1 = H[0,0]
print(H1)
H2 = H[0,1]
print(H2)

# Step response for the system
t, y = control.step_response(H1)
plt.plot(t, y)

t, y = control.step_response(H2)
plt.plot(t, y)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2"])
plt.grid()
plt.show()
```

SciPy Library

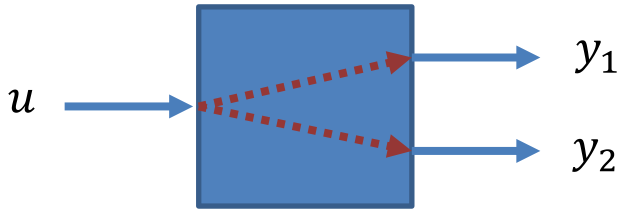
SIMO

Single Input/Multiple Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ 2\end{bmatrix}u$$

$$y = \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



$$H_1(s) = \frac{y_1(s)}{u(s)} \qquad H_2(s) = \frac{y_2(s)}{u(s)}$$

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

# SIMO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0],
    [2]]

C = [[1, 0],
     [0, 1]]

D = [[0]]

# Find Transfer Function from u to y1
C = [[1, 0]]
num, den = signal.ss2tf(A, B, C, D)
H1 = signal.TransferFunction(num, den)
print(H1)

# Find Transfer Function from u to y2
C = [[0, 1]]
num, den = signal.ss2tf(A, B, C, D)
H2 = signal.TransferFunction(num, den)
print(H2)

# Step response for the system
t, y = signal.step(H1)
plt.plot(t, y)

t, y = signal.step(H2)
plt.plot(t, y)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2"])
plt.grid()
plt.show()
```

Control Library

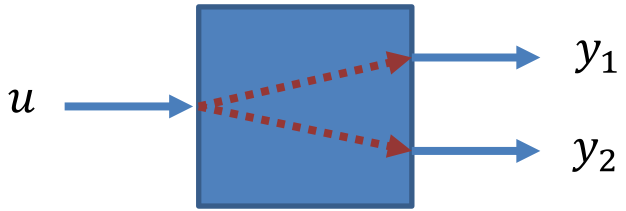
SIMO

Single Input/Multiple Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ 2\end{bmatrix}u$$

$$y = \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



$$H_1(s) = \frac{y_1(s)}{u(s)} \qquad H_2(s) = \frac{y_2(s)}{u(s)}$$

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

# SIMO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0],
    [2]]

C = [[1, 0],
     [0, 1]]

D = 0

ssmodel = control.ss(A, B, C, D)


H = control.ss2tf(ssmodel)
print(H)
H1 = H[0,0]
print(H1)
H2 = H[1,0]
print(H2)

# Step response for the system
t, y = control.step_response(H)

y1 = y[0, :]
y2 = y[1, :]

plt.plot(t, y1)
plt.plot(t, y2)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2"])
plt.grid()
plt.show()
```

SciPy Library

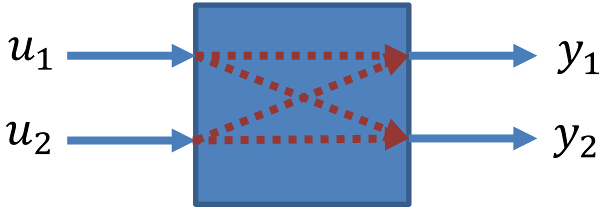
MIMO

Multiple Input/Multiple Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 2 & 4\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



$$H_1(s) = \frac{y_1(s)}{u_1(s)} \qquad H_2(s) = \frac{y_1(s)}{u_2(s)}$$

$$H_3(s) = \frac{y_2(s)}{u_1(s)} \qquad H_4(s) = \frac{y_2(s)}{u_2(s)}$$

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

# SIMO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0, 0],
    [2, 4]]

D = [[0, 0]]


C = [[1, 0]]
# Find Transfer Function from u1 to y1
num, den = signal.ss2tf(A, B, C, D, 0)
H1 = signal.TransferFunction(num, den)
print(H1)

# Find Transfer Function from u2 to y1
num, den = signal.ss2tf(A, B, C, D, 1)
H2 = signal.TransferFunction(num, den)
print(H2)

C = [[0, 1]]
# Find Transfer Function from u1 to y2
num, den = signal.ss2tf(A, B, C, D, 0)
H3 = signal.TransferFunction(num, den)
print(H3)

# Find Transfer Function from u1 to y2
num, den = signal.ss2tf(A, B, C, D, 1)
H4 = signal.TransferFunction(num, den)
print(H4)

# Step response for the system
t, y = signal.step(H1)
plt.plot(t, y)

t, y = signal.step(H2)
plt.plot(t, y)

t, y = signal.step(H3)
plt.plot(t, y)

t, y = signal.step(H4)
plt.plot(t, y)

plt.title("Step Response")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2", "H3", "H4"])
plt.grid()
plt.show()
```

Control Library

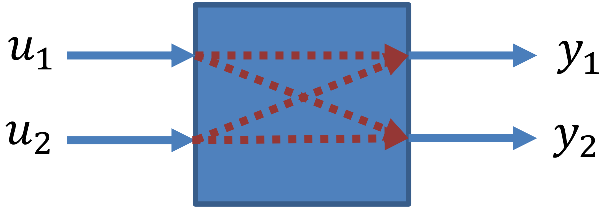
MIMO

Multiple Input/Multiple Output

State-space Model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -1 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 2 & 4\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



$$H_1(s) = \frac{y_1(s)}{u_1(s)} \qquad H_2(s) = \frac{y_1(s)}{u_2(s)}$$

$$H_3(s) = \frac{y_2(s)}{u_1(s)} \qquad H_4(s) = \frac{y_2(s)}{u_2(s)}$$

```
import control
import matplotlib.pyplot as plt

# MIMO System
# Define State-space Model
A = [[0, 1],
     [-1, -3]]

B = [[0, 0],
    [2, 4]]

C = [[1, 0],
     [0, 1]]

D = 0

ssmodel = control.ss(A, B, C, D)

H = control.ss2tf(ssmodel)
print(H)
H1 = H[0,0]
print(H1)
H2 = H[0,1]
print(H2)
H3 = H[1,0]
print(H3)
H4 = H[1,1]
print(H4)

# Step response for the system
t, y = control.step_response(H1)
plt.plot(t, y)

t, y = control.step_response(H2)
plt.plot(t, y)

t, y = control.step_response(H3)
plt.plot(t, y)

t, y = control.step_response(H4)
plt.plot(t, y)

plt.title("Step Response H")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(["H1", "H2", "H3", "H4"])
plt.grid()
plt.show()
```

Discrete State Space Models

Hans-Petter Halvorsen

Discretization Methods

We have many different Discretization Methods

- Euler
  - Euler forward method
  - Euler backward method
- Zero Order Hold (ZOH)
- Tustin
- …

We will focus on this since it it easy to use and implement

It is recommended that you know about Discrete Systems. If not, take a closer look at my Tutorial "Discrete Systems with Python"

Discretization Methods

Euler forward method:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Where T_s is the sampling time, and $x(k+1)$, $x(k)$ and $x(k-1)$ are discrete values of $x(t)$

Discretization Example

Differential Equation (1.order system):

$$\dot{x} = \frac{1}{T}(-x + Ku) \qquad \text{or:} \qquad \dot{x} = -\frac{1}{T}x + \frac{K}{T}u$$

We use Euler forward method:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Then we get:

$$\frac{x(k+1) - x(k)}{T_s} = -\frac{1}{T}\,x(k) + \frac{K}{T}u(k)$$

Further:

$$x(k+1) = x(k) + T_s\left(-\frac{1}{T}\,x(k) + \frac{K}{T}u(k)\right)$$

And:

$$x(k+1) = x(k) - \frac{T_s}{T}\,x(k) + \frac{T_sK}{T}u(k)$$

Finally:

$$x(k+1) = \left(1 - \frac{T_s}{T}\right)x(k) + \frac{T_sK}{T}u(k)$$

Discrete State-space Models

Given a Continuous State-space Model:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

The Discrete State-space Model is then given by:

$$x_{k+1} = (I + T_s A)x_k + T_s B u_k$$
$$y_k = Cx_k + Du_k$$

T_s is the discrete **Sampling Time**

This equation is derived using the Euler forward method on a general state-space model.

$$x_{k+1} = A_d x_k + B_d u_k$$
$$y_k = C_d x_k + D_d u_k$$

SciPy.signal

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

Discrete-time linear systems

dlti(*system, **kwargs)	Discrete-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.
dlsim(system, u[, t, x0])	Simulate output of a discrete-time linear system.
dimpulse(system[, x0, t, n])	Impulse response of discrete-time system.
dstep(system[, x0, t, n])	Step response of discrete-time system.
dfreqresp(system[, w, n, whole])	Calculate the frequency response of a discrete-time system.
dbode(system[, w, n])	Calculate Bode magnitude and phase data of a discrete-time system.

Discrete State-space Models

Given the following:

We have the differential equations:

$$\dot{x}_1 = \frac{1}{T}(-x_1 + Ku)$$
$$\dot{x}_2 = 0$$

The State-space Model becomes:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{T} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{K}{T} \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

What is the discrete State-space Model?

Discrete State-space Models

We have the differential equations:

$$\dot{x}_1 = \frac{1}{T}(-x_1 + Ku)$$
$$\dot{x}_2 = 0$$

We use Euler forward method:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

$$\frac{x_2(k+1) - x_2(k)}{T_s} = 0$$

$$x_2(k+1) = x_2(k)$$

$$x_1(k+1) = \left(1 - \frac{T_s}{T}\right)x_1(k) + \frac{T_s K}{T}u(k)$$

$$x_2(k+1) = x_2(k)$$

This gives the following Discrete State-space model:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \left(1 - \frac{T_s}{T}\right) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} \frac{T_s K}{T} \\ 0 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

Discrete State-space Models

Discrete State-space model:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \left(1 - \frac{T_s}{T}\right) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} \frac{T_s K}{T} \\ 0 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

We set $K = 3, T = 4$

$$T_s = 0.1$$

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.975 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0.075 \\ 0 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

Python Example

```
import scipy.signal as sig

K = 3
T = 4

# State-space Model
A = [[-1/T, 0],
     [0, 0]]

B = [[K/T],
     [0]]

C = [[1, 0]]

D = 0

sys = sig.StateSpace(A, B, C, D)
print(sys)


sys_d = sys.to_discrete(dt=0.1, method='euler')
print(sys_d)
```

```
StateSpaceContinuous(
array([[-0.25,  0.  ],
       [ 0.  ,  0.  ]]),
array([[0.75],
       [0.  ]]),
array([[1, 0]]),
array([[0]]),
dt: None
)
```

```
StateSpaceDiscrete(
array([[0.975, 0.    ],
       [0.   , 1.    ]]),
array([[0.075],
       [0.   ]]),
array([[1., 0.]]),
array([[0.]]),
dt: 0.1
)
```

We see that we get the correct answer

Python Example

Implement Discretization from scratch

$$x_{k+1} = (I + T_s A)x_k + T_s B u_k$$
$$y_k = C x_k + D u_k$$

$$A_d = (I + T_s A)$$

$$B_d = T_s B$$

```
import numpy as np
import scipy.signal as sig

K = 3
T = 4

# State-space Model
A = np.array([[-1/T, 0], [0, 0]])
B = np.array([[K/T], [0]])
C = np.array([[1, 0]])
D = 0
sys = sig.StateSpace(A, B, C, D)

sys_d = sys.to_discrete(dt=0.1, method='euler')
print(sys_d)

I = np.eye(len(A[0]))

Ts = 0.1

Ad = I + Ts * A
print(Ad)

Bd = Ts * B
print(Bd)

sys_d2 = sig.StateSpace(Ad, Bd, C, D, dt=Ts)
print(sys_d)
```

Python Example

Implement **self-made** c2d() function

$$x_{k+1} = (I + T_s A)x_k + T_s B u_k$$
$$y_k = C x_k + D u_k$$

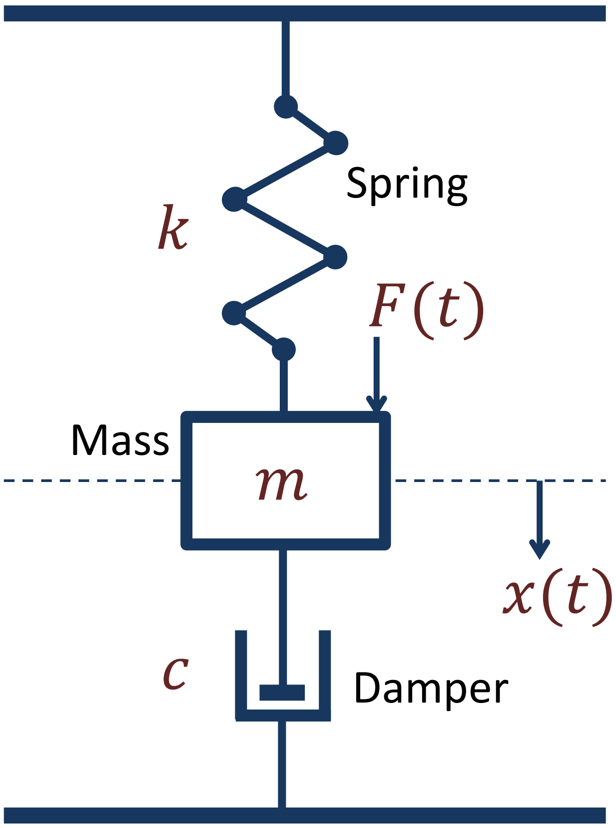
$$A_d = (I + T_s A)$$

$$B_d = T_s B$$

```
import numpy as np
import scipy.signal as sig


def c2d(A,B, Ts):

    I = np.eye(len(A[0]))

    Ts = 0.1
    Ad = I + Ts * A

    Bd = Ts * B

    return Ad, Bd


K = 3
T = 4

# State-space Model
A = np.array([[-1/T, 0], [0, 0]])

B = np.array([[K/T], [0]])

C = np.array([[1, 0]])

D = 0


Ts = 0.1

Ad, Bd = c2d(A, B, Ts)


sys_d = sig.StateSpace(Ad, Bd, C, D, dt=Ts)
print(sys_d)
```

Mass-Spring-Damper System with Python

Mass-Spring-Damper System

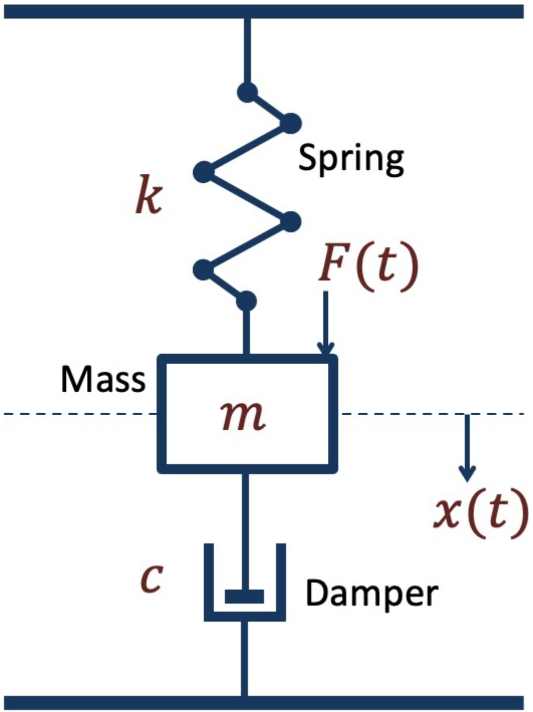


The ”Mass-Spring-Damper“ System is typical system used to demonstrate and illustrate Modelling and Simulation Applications

Mass-Spring-Damper System

Given a so-called "Mass-Spring-Damper" system

Newtons 2.law: $\sum F = ma$



The system can be described by the following equation:

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

Where t is the time, $F(t)$ is an external force applied to the system, c is the damping constant, k is the stiffness of the spring, m is a mass.

$x(t)$ is the position of the object (m)

$\dot{x}(t)$ is the first derivative of the position, which equals the velocity/speed of the object (m)

$\ddot{x}(t)$ is the second derivative of the position, which equals the acceleration of the object (m)

Mass-Spring-Damper System

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t)$$

Higher order differential equations can typically be reformulated into a system of first order differential equations

$$m\ddot{x} = F - c\dot{x} - kx$$

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx)$$

x_1= Position

x_2= Velocity/Speed

We set

$x = x_1$

$\dot{x} = x_2$

This gives:

$\dot{x}_1 = x_2$

$\dot{x}_2 = \ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) = \frac{1}{m}(F - cx_2 - kx_1)$

Finally:

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) \Rightarrow$$

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

State-space Model

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

$$\dot{x} = Ax + Bu$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F$$

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\frac{k}{m}x_1 - \frac{c}{m}x_2 + \frac{1}{m}F$$

$$\dot{x} = Ax + Bu$$

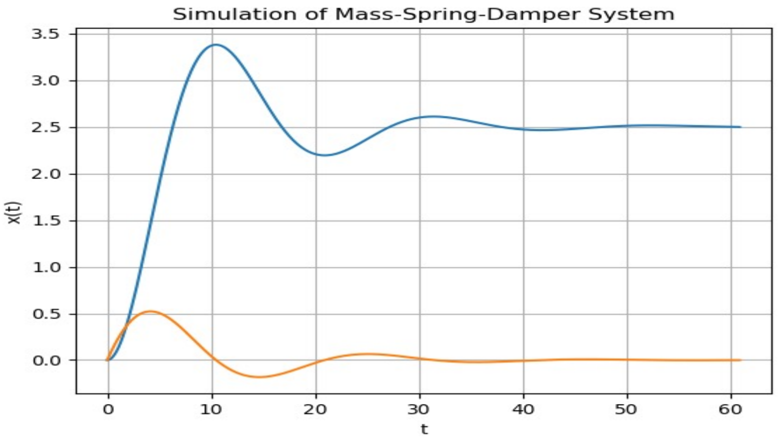
$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad F = u$$

Python Code

State-space Model

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{k}{m} & -\frac{c}{m}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix}F$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$




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

# Parameters defining the system
c = 4 # Damping constant
k = 2 # Stiffness of the spring
m = 20 # Mass
F = 5 # Force
Ft = np.ones(610)*F

# Simulation Parameters
tstart = 0
tstop = 60
increment = 0.1
t = np.arange(tstart,tstop+1,increment)

# System matrices
A = [[0, 1], [-k/m, -c/m]]
B = [[0], [1/m]]
C = [[1, 0]]
sys = sig.StateSpace(A, B, C, 0)

# Step response for the system
t, y, x = sig.lsim(sys, Ft, t)
x1 = x[:,0]
x2 = x[:,1]

plt.plot(t, x1, t, x2)
#plt.plot(t, y)
plt.title('Simulation of Mass-Spring-Damper System')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.show()
```

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
- …

Functions for Model Simulations:

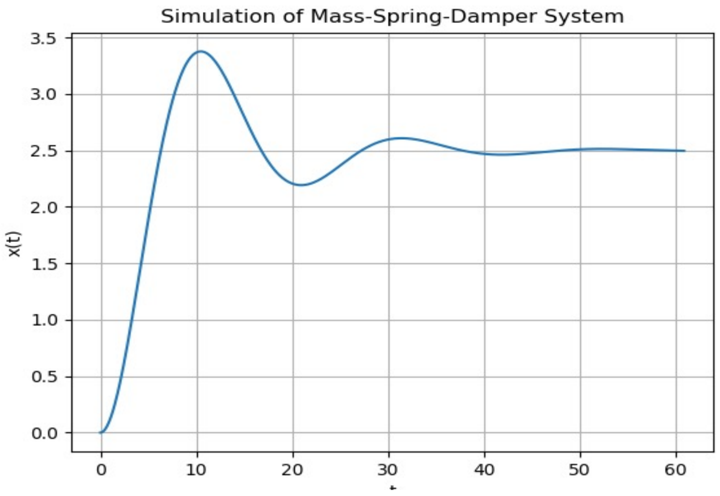
- `step_response()` - Step response of a linear system
- **`forced_response()`**
- `lsim()` - Simulate the output of a linear system
- …

Python Code

State-space Model

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{k}{m} & -\frac{c}{m}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix}F$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$



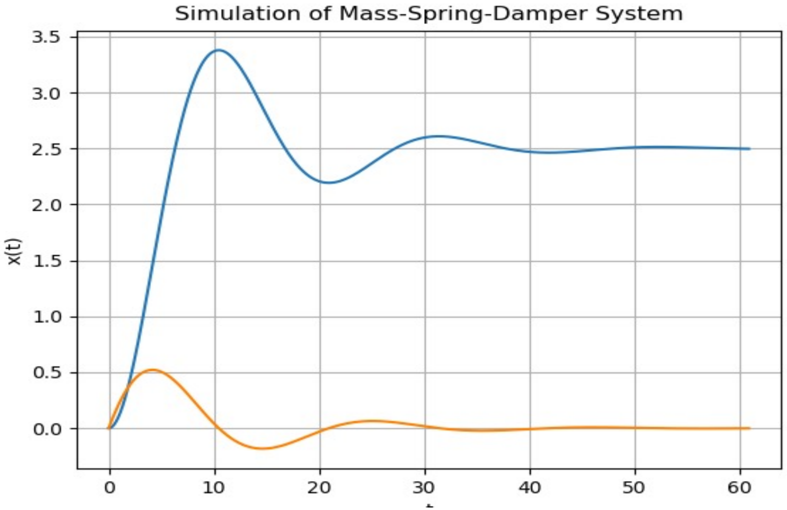

```
import numpy as np
import matplotlib.pyplot as plt
import control

# Parameters defining the system
c = 4 # Damping constant
k = 2 # Stiffness of the spring
m = 20 # Mass
F = 5 # Force

# Simulation Parameters
tstart = 0
tstop = 60
increment = 0.1
t = np.arange(tstart,tstop+1,increment)

# System matrices
A = [[0, 1], [-k/m, -c/m]]
B = [[0], [1/m]]
C = [[1, 0]]
D = 0
sys = control.ss(A, B, C, D)

# Step response for the system
t, y, x = control.forced_response(sys, t, F)
plt.plot(t, y)
plt.title('Simulation of Mass-Spring-Damper System')
plt.xlabel('t’); plt.ylabel('x(t)')
plt.grid()
plt.show()
```

Python Code

State-space Model

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{k}{m} & -\frac{c}{m}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{m}\end{bmatrix}F$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$




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

# Parameters defining the system
c = 4 # Damping constant
k = 2 # Stiffness of the spring
m = 20 # Mass
F = 5 # Force

# Simulation Parameters
tstart = 0
tstop = 60
increment = 0.1
t = np.arange(tstart,tstop+1,increment)

# System matrices
A = [[0, 1], [-k/m, -c/m]]
B = [[0], [1/m]]
C = [[1, 0]]
D = 0
sys = control.ss(A, B, C, D)

# Step response for the system
t, y, x = control.forced_response(sys, t, F)
x1 = x[0 ,:]
x2 = x[1 ,:]
plt.plot(t, x1, t, x2)
plt.title('Simulation of Mass-Spring-Damper System')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.show()
```

Discretization

Given:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{1}{m}(F - cx_2 - kx_1)$$

Using Euler:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Then we get:

$$\frac{x_1(k+1) - x_1(k)}{T_s} = x_2(k)$$
$$\frac{x_2(k+1) - x_2(k)}{T_s} = \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

This gives:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = x_2(k) + T_s \frac{1}{m}[F(k) - cx_2(k) - kx_1(k)]$$

Then we get:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + x_2(k) - T_s \frac{c}{m} x_2(k) + T_s \frac{1}{m} F(k)$$

Finally:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k)$$

Discrete State-space Model

Discrete System:

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$
$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k)$$

We can set it on Discrete state space form:

$$x(k+1) = A_d x(k) + B_d u(k)$$

This gives:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ -T_s \frac{k}{m} & 1 - T_s \frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ T_s \frac{1}{m} \end{bmatrix} F(k)$$

$$A = \begin{bmatrix} 1 & T_s \\ -T_s \frac{k}{m} & 1 - T_s \frac{c}{m} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ T_s \frac{1}{m} \end{bmatrix}$$

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

We can also use `control.c2d()` function

Python Code

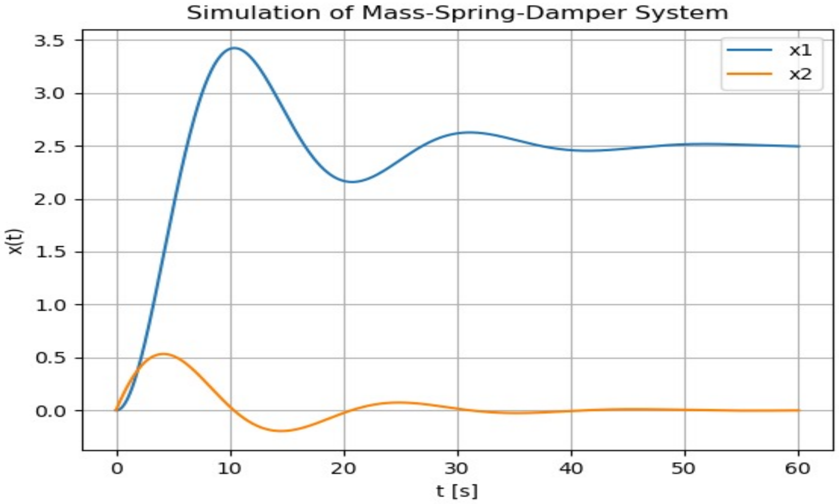
Discrete System

$$x_1(k+1) = x_1(k) + T_s x_2(k)$$

$$x_2(k+1) = -T_s \frac{k}{m} x_1(k) + (1 - T_s \frac{c}{m}) x_2(k) + T_s \frac{1}{m} F(k)$$

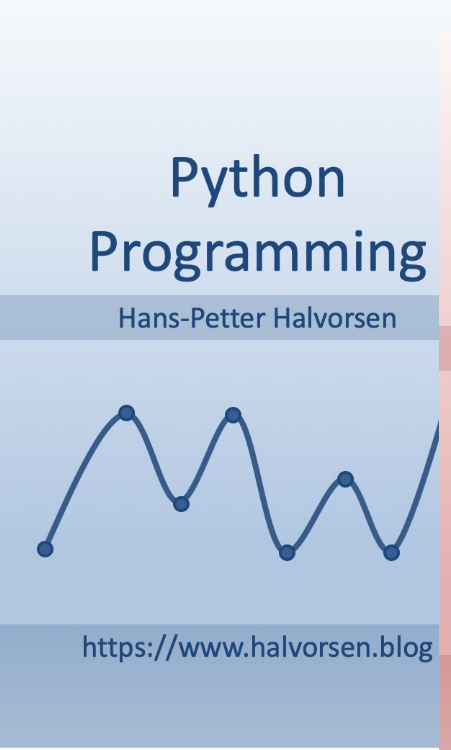
x_1= Position

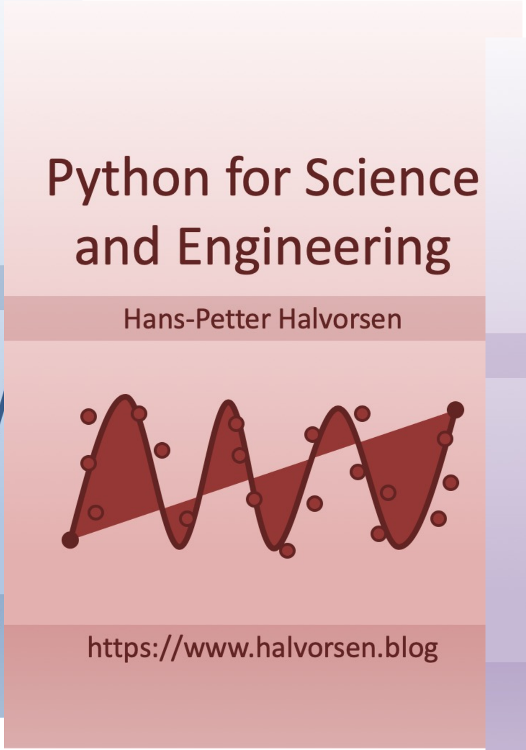
x_2= Velocity/Speed




```
# Simulation of Mass-Spring-Damper System
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
c = 4 # Damping constant
k = 2 # Stiffness of the spring
m = 20 # Mass
F = 5 # Force

# Simulation Parameters
Ts = 0.1
Tstart = 0
Tstop = 60
N = int((Tstop-Tstart)/Ts) # Simulation length
x1 = np.zeros(N+2)
x2 = np.zeros(N+2)
x1[0] = 0 # Initial Position
x2[0] = 0 # Initial Speed

a11 = 1
a12 = Ts
a21 = -(Ts*k)/m
a22 = 1 - (Ts*c)/m

b1 = 0
b2 = Ts/m

# Simulation
for k in range(N+1):
    x1[k+1] = a11 * x1[k] + a12 * x2[k] + b1 * F
    x2[k+1] = a21 * x1[k] + a22 * x2[k] + b2 * F

# Plot the Simulation Results
t = np.arange(Tstart,Tstop+2*Ts,Ts)

#plt.plot(t, x1, t, x2)
plt.plot(t,x1)
plt.plot(t,x2)
plt.title('Simulation of Mass-Spring-Damper System')
plt.xlabel('t [s]')
plt.ylabel('x(t)')
plt.grid()
plt.legend(["x1", "x2"])
plt.show()
```

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