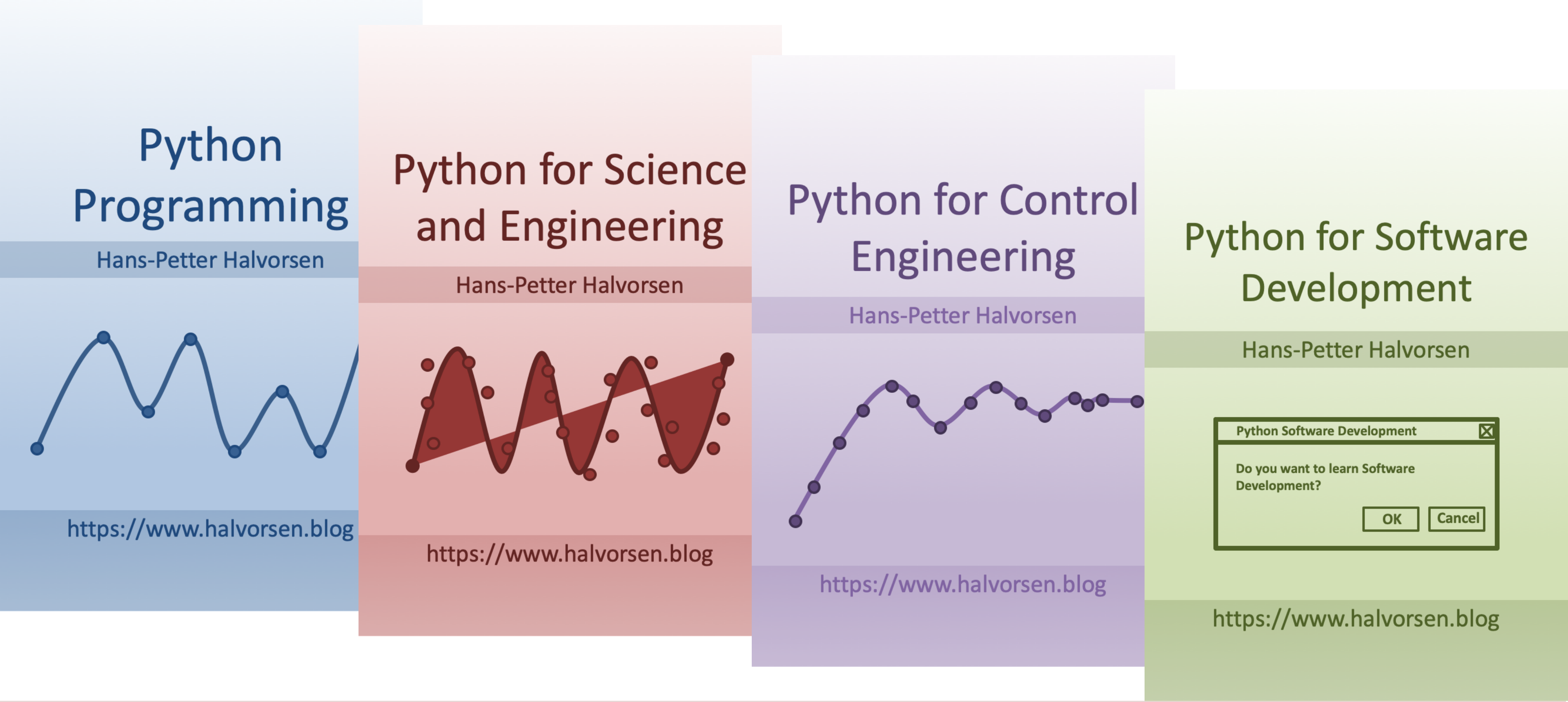
Mass-Spring-Damper System with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

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

Additional Python Resources

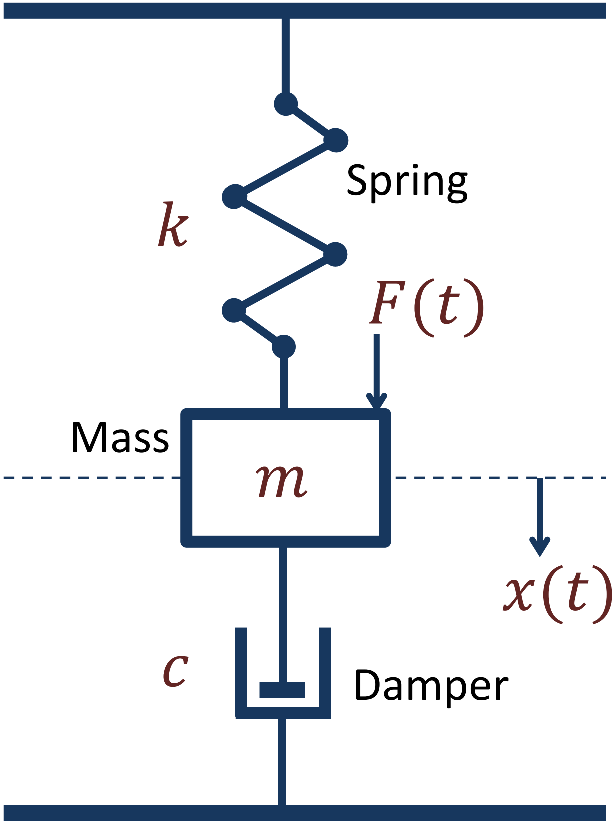


https://www.halvorsen.blog/documents/programming/python/

Contents

- Mass-Spring-Damper System
- Simulations:
  - SciPy ODE Solvers
  - State-space Model
  - Discrete System

Mass-Spring-Damper System

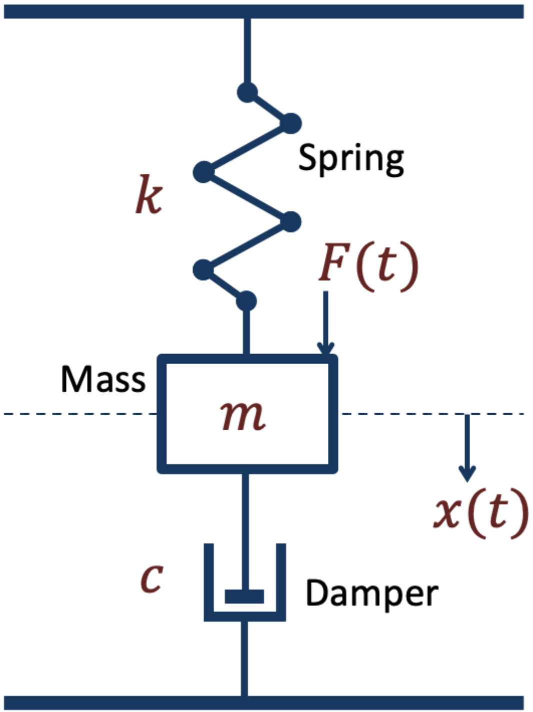


The ”Mass-Spring-Damper“ System is typical system used to demonstrate and illustrate Modelling and Simulation Applications

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:

$$\ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) \quad \Rightarrow \quad \begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{m}(F - cx_2 - kx_1) \end{aligned}$$

https://www.halvorsen.blog

SciPy ODE Solver

Hans-Petter Halvorsen

SciPy

- SciPy is a free and open-source Python library used for scientific computing and engineering
- SciPy contains modules for optimization, linear algebra, interpolation, image processing, ODE solvers, etc.
- SciPy is included in the Anaconda distribution

Python Code

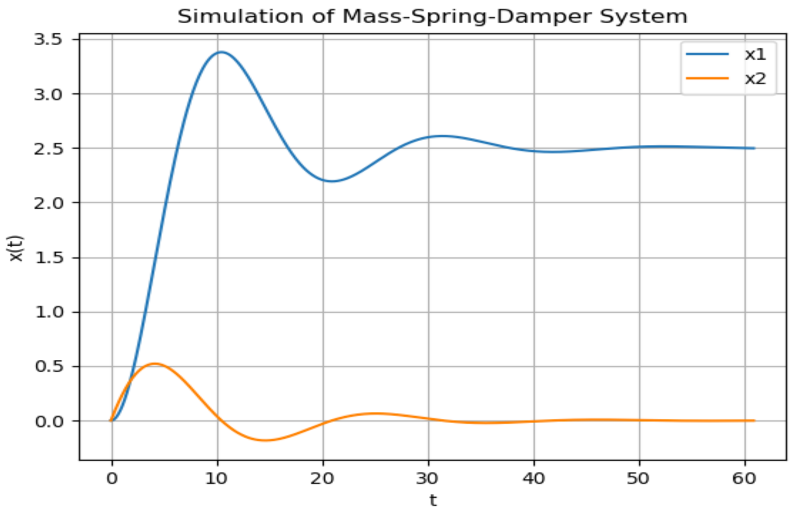
Using SciPy ODE Solver

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

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

x_1= Position

x_2= Velocity/Speed




```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Initialization
tstart = 0
tstop = 60
increment = 0.1

# Initial condition
x_init = [0,0]

t = np.arange(tstart,tstop+1,increment)

# Function that returns dx/dt
def mydiff(x, t):
    c = 4 # Damping constant
    k = 2 # Stiffness of the spring
    m = 20 # Mass
    F = 5

    dx1dt = x[1]
    dx2dt = (F - c*x[1] - k*x[0])/m

    dxdt = [dx1dt, dx2dt]
    return dxdt

# Solve ODE
x = odeint(mydiff, x_init, t)

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

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

State-space Model

State-space Model

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

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

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

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

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

Python Control Systems Library

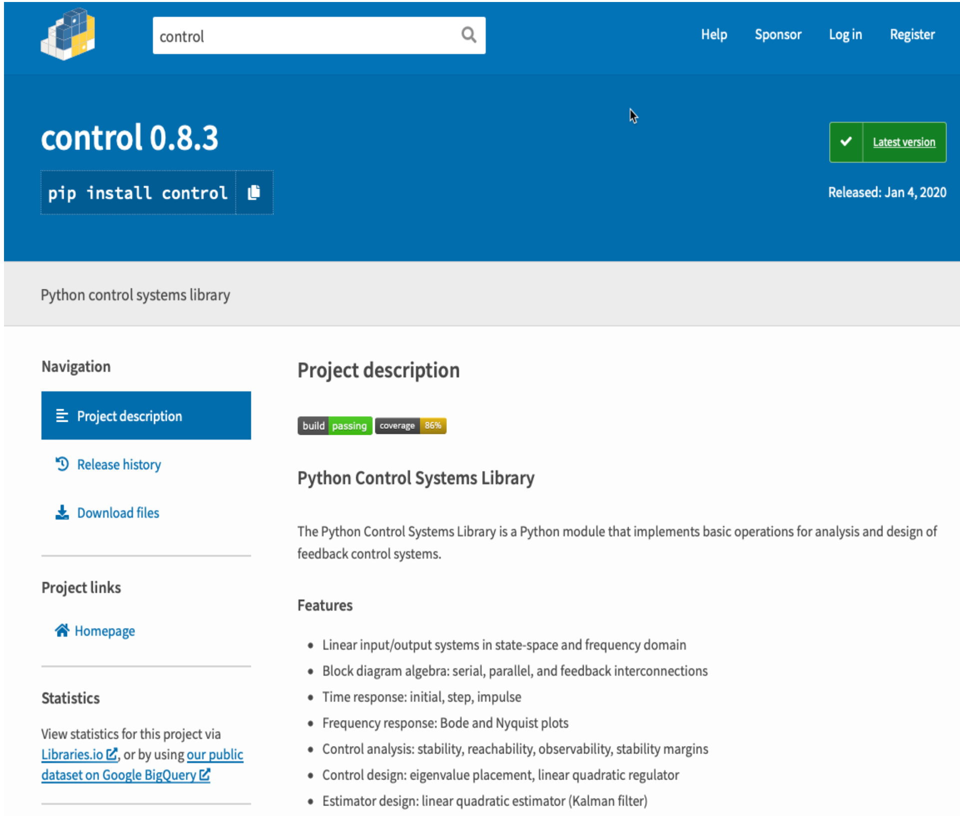
- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: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

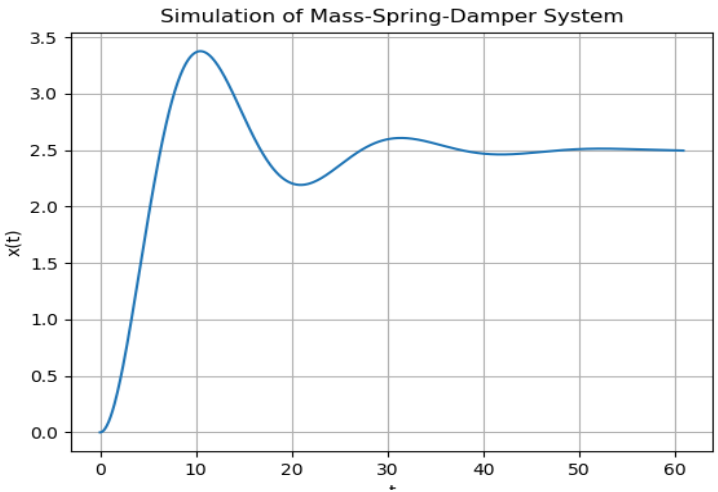


Python Code

State-space Model

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

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



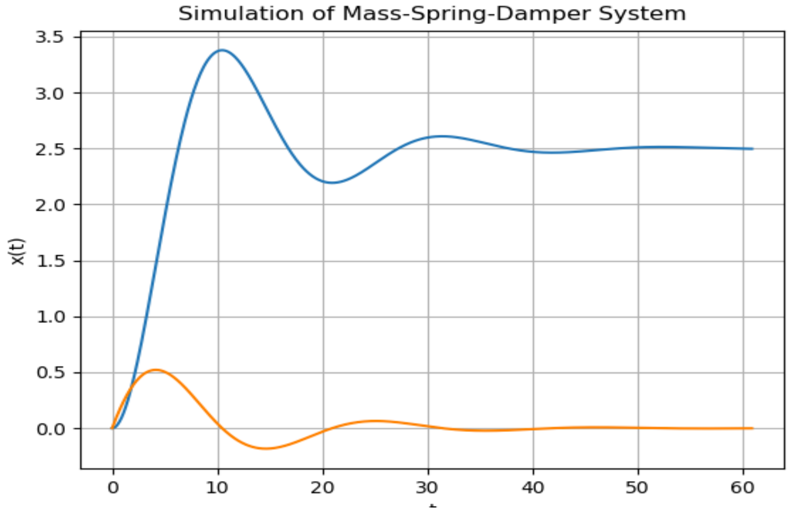

```
import numpy as np
import matplotlib.pyplot as plt
import control

# 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]]
sys = control.ss(A, B, C, 0)

# 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]]
sys = control.ss(A, B, C, 0)

# 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()
```

SciPy.signal

- An alternative to The Python Control Systems Library is SciPy.signal, i.e. the Signal Module in the SciPy Library
- https://docs.scipy.org/doc/scipy/reference/signal.html

SciPy is included with the Anaconda distribution

Continuous-time linear systems

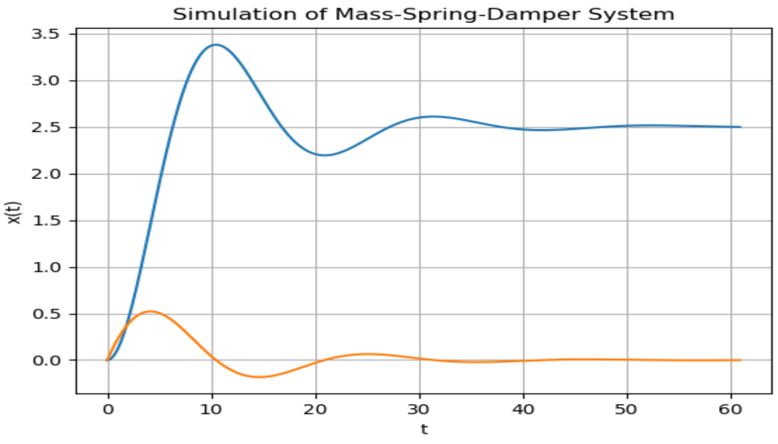
lti(*system)	Continuous-time linear time invariant system base class.
StateSpace(*system, **kwargs)	Linear Time Invariant system in state-space form.
TransferFunction(*system, **kwargs)	Linear Time Invariant system class in transfer function form.
ZerosPolesGain(*system, **kwargs)	Linear Time Invariant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

Discretization

Hans-Petter Halvorsen

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

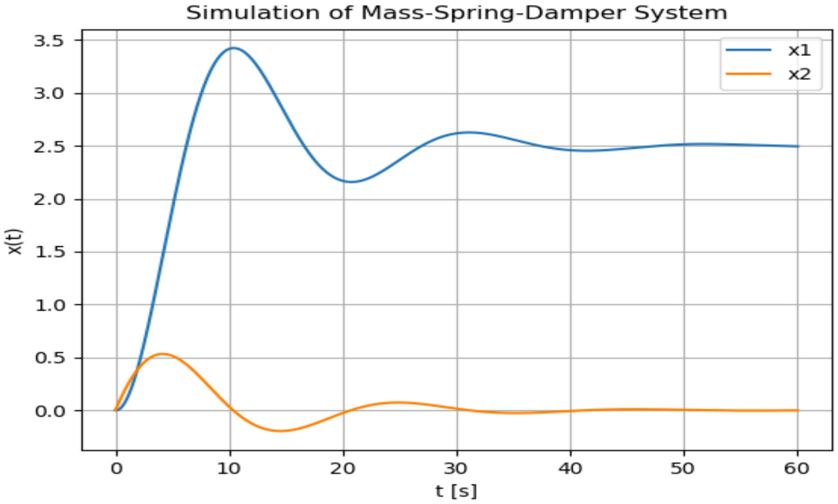
Discrete System

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

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

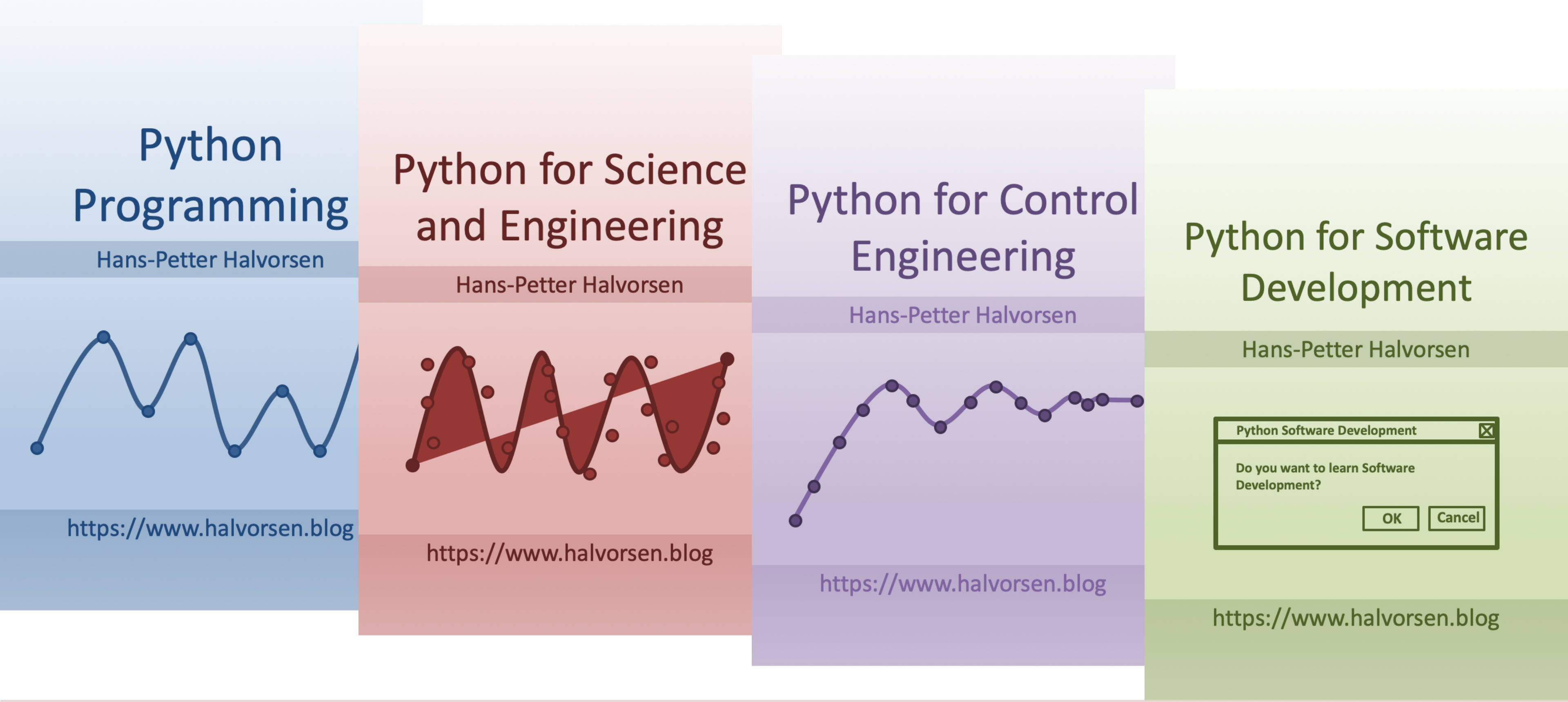
x_1= Position

x_2= Velocity/Speed




```python
# 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