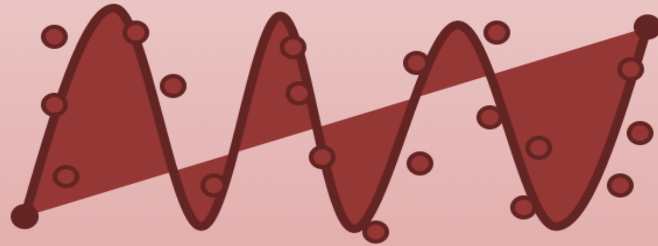
Discrete Systems with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Science and Engineering

Hans-Petter Halvorsen



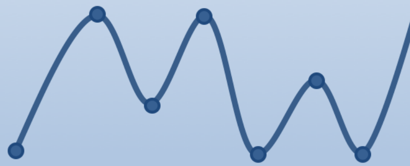
https://www.halvorsen.blog

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

Additional Python Resources

Python Programming

Hans-Petter Halvorsen



https://www.halvorsen.blog

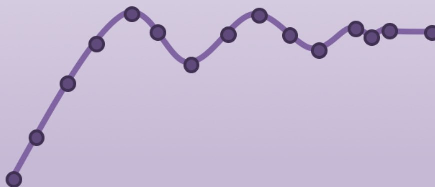
Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

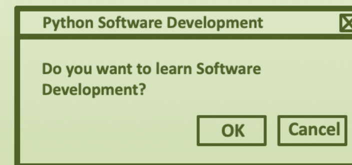
Hans-Petter Halvorsen



https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen



https://www.halvorsen.blog

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

Contents

- Differential Equations
- Simulation
- Discrete Systems
- Examples

Simulation of Discrete Systems

- Python has powerful features for simulation of continuous differential equations and dynamic systems.
- Sometimes we want to or need to discretize a continuous system and then simulate it in Python.
- This means we need to make a discrete version of our continuous differential equations.
- The built-in ODE solvers in Python use different discretization methods

Differential Equations

Hans-Petter Halvorsen

Differential Equations

Differential Equation on general form:

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0$$

Different notation is used:

$$\frac{dy}{dt} = y' = \dot{y}$$

Example:

$$\frac{dy}{dt} = 3y + 2, \qquad y(t_0) = 0$$

ODE – Ordinary Differential Equations

Differential Equations

Example:

$$\dot{x} = ax$$

Note that $\dot{x} = \frac{dx}{dt}$

Where $x_0 = x(0) = x(t_0)$ is the initial condition

Where $a = -\frac{1}{T}$, where T is denoted as the time constant of the system

The solution for the differential equation is found to be (learned in basic Math courses and will not be derived here):

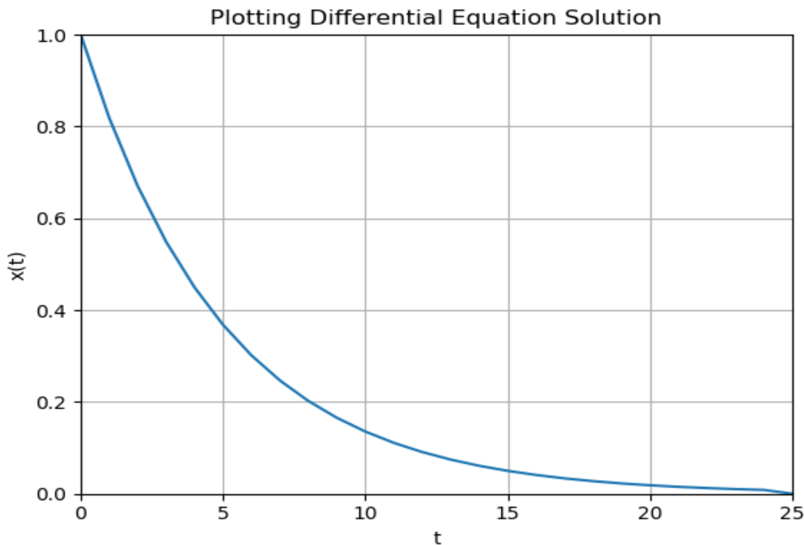
$$x(t) = e^{at}x_0$$

Where $x_0 = x(0) = x(t_0)$ is the initial condition

Python Code

$$x(t) = e^{at}x_0$$

In our system we can set $T = 5$ and the initial condition $x_0 = x(0) = 1$




```
import math as mt
import numpy as np
import matplotlib.pyplot as plt

# Parameters
T = 5
a = -1/T
x0 = 1
t = 0
tstart = 0
tstop = 25
increment = 1

x = []
x = np.zeros(tstop+1)
t = np.arange(tstart,tstop+1,increment)

# Define the Equation
for k in range(tstop):
    x[k] = mt.exp(a*t[k]) * x0

# Plot the Results
plt.plot(t,x)
plt.title('Plotting Differential Equation Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
```

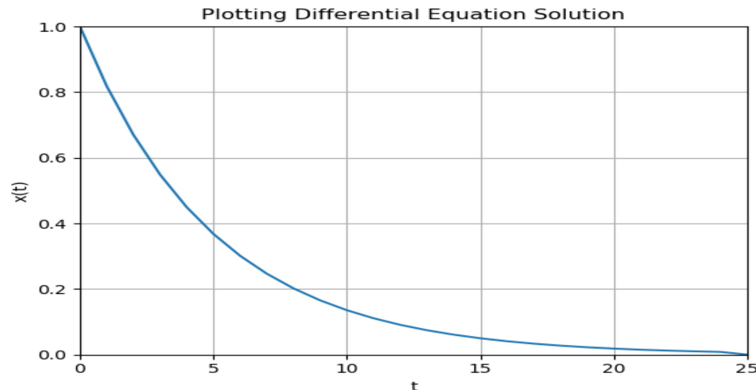
Alt. Solution

(without For Loop)

$$x(t) = e^{at}x_0$$

In our system we can set $T = 5$ and the initial condition $x_0 = x(0) = 1$

In this alternative solution no For Loop has been used




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

# Parameters
T = 5
a = -1/T
x0 = 1
t = 0

tstart = 0
tstop = 25
increment = 1
N = 25

#t = np.arange(tstart,tstop+1,increment)
#Alternative Approach
t = np.linspace(tstart, tstop, N)

x = np.exp(a*t) * x0

# Plot the Results
plt.plot(t,x)
plt.title('Plotting Differential Equation Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.show()
```

Summary

- Solving differential equations like shown in these examples works fine
- But the problem is that we first have to manually (by "pen and paper") find the solution to the differential equation.
- An alternative is to use solvers for Ordinary Differential Equations (ODE) in Python, so-called ODE Solvers
- The next approach is to find the discrete version and then implement and simulate the discrete system

ODE Solvers in Python

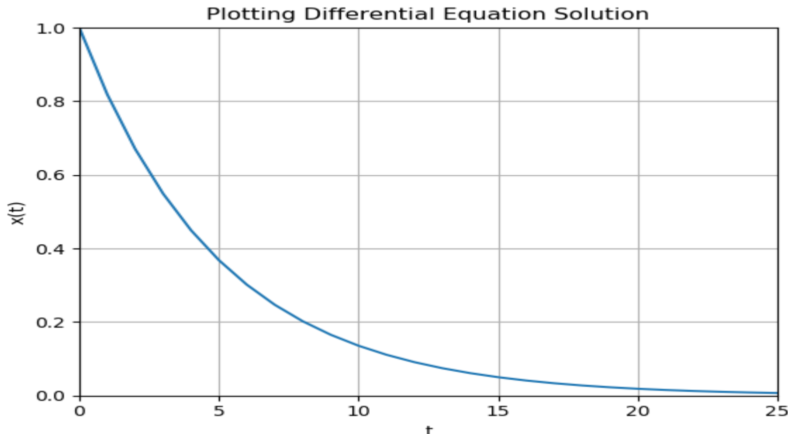
- The **scipy.integrate** library has two powerful powerful functions; ode() and odeint(), for numerically solving first order ordinary differential equations (ODEs).
- The ode() is more flexible, while odeint() (ODE integrator) has a simpler Python interface and works fine for most problems.
- For details, see the SciPy documentation:
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html
- https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.ode.html

Python Code

Here we use an ODE solver in SciPy

$$\dot{x} = ax$$

In our system we can set $T = 5$ $(a = -\frac{1}{T})$
and the initial condition $x_0 = x(0) = 1$

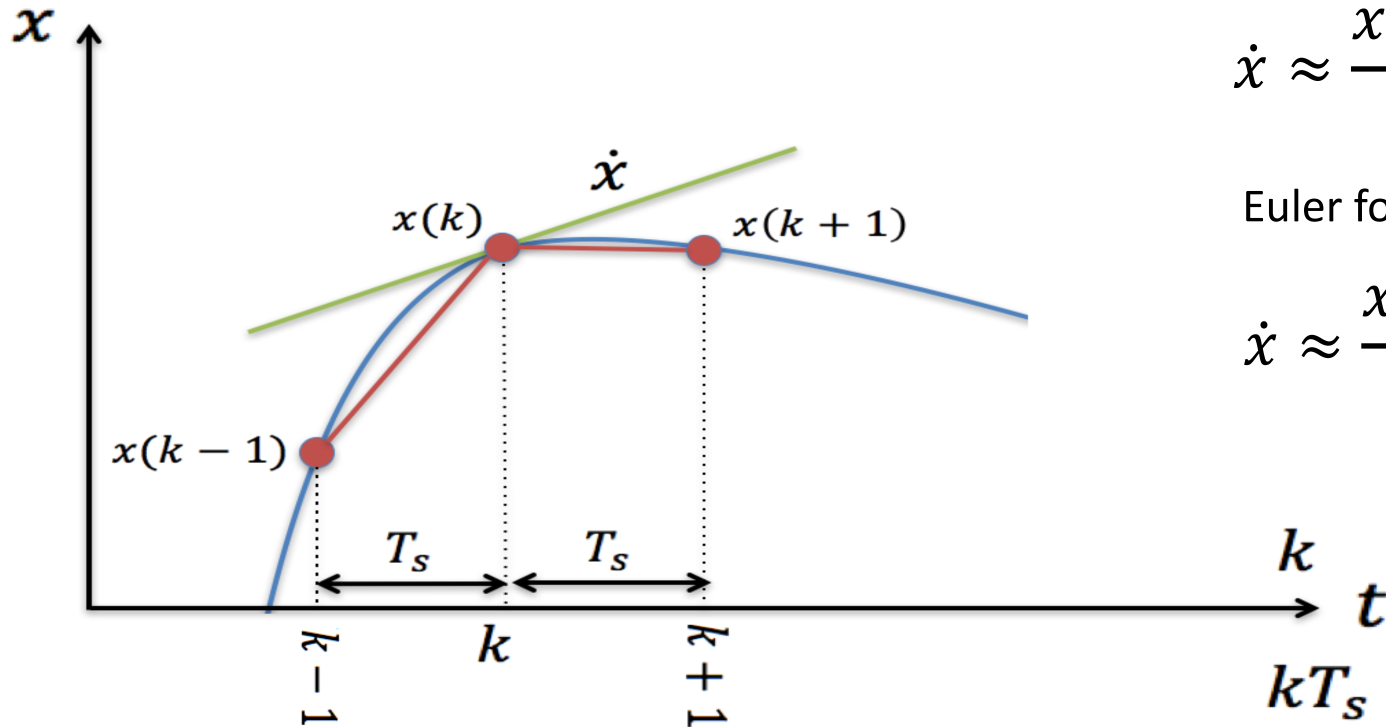



```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Initialization
tstart = 0
tstop = 25
increment = 1
x0 = 1
t = np.arange(tstart,tstop+1,increment)

# Function that returns dx/dt
def mydiff(x, t):
    T = 5
    a = -1/T
    dxdt = a * x
    return dxdt

# Solve ODE
x = odeint(mydiff, x0, t)
print(x)

# Plot the Results
plt.plot(t,x)
plt.title('Plotting Differential Equation Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.show()
```

Discrete Systems

Hans-Petter Halvorsen

Discretization



Euler backward method:

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

Euler forward method:

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

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

Discrete Systems

Different Discrete Symbols and meanings

Previous Value: $x(k-1) = x_{k-1} = x(t_{k-1})$

Present/Current Value: $x(k) = x_k = x(t_k)$

Next (Future) Value: $x(k+1) = x_{k+1} = x(t_{k+1})$

Note! Different Notation is used in different literature!

Euler

Euler backward method:

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

More Accurate!

Euler forward method:

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

Simpler to use!

(We will primary use this in the examples)

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

Simulation Example

Given the following differential equation:

$$\dot{x} = ax$$

where $a = -\frac{1}{T}$, where T is the time constant

Note! $\dot{x} = \frac{dx}{dt}$

Find the discrete differential equation and plot the solution for this system using Python.

Set $T = 5$ and the initial condition $x(0) = 1$.

We will create a script in Python where we plot the solution $x(t)$ in the time interval $0 \leq t \leq 25$

Discretization

The differential equation of the system is:

$$\dot{x} = ax$$

We need to find the discrete version:

We use the Euler forward method:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

This gives:

$$\frac{x_{k+1} - x_k}{T_s} = ax_k$$

Next:

$$x_{k+1} - x_k = T_s a x_k$$

Next:

$$x_{k+1} = x_k + T_s a x_k$$

This gives the following discrete version:

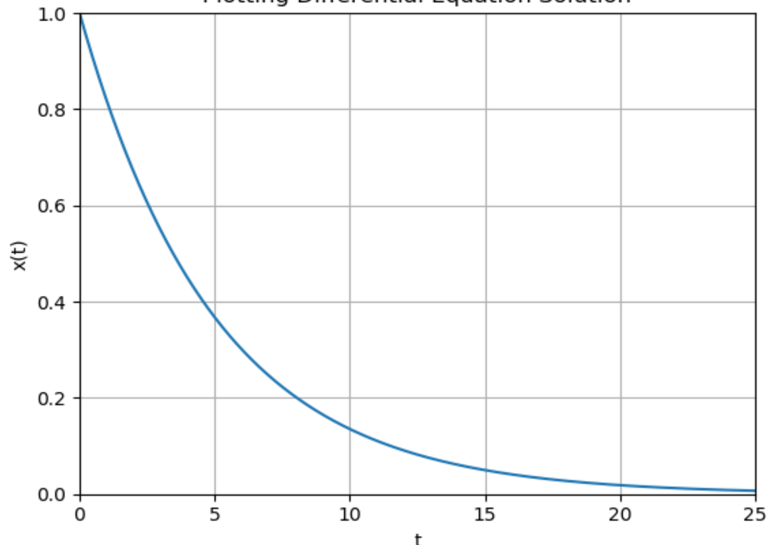
$$x_{k+1} = (1 + aT_s)\, x_k$$

Python Code

Differential Equation: $\dot{x} = ax$

Simulation of Discrete System:

$$x_{k+1} = (1 + aT_s)x_k$$




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

# Model Parameters
T = 5
a = -1/T
# Simulation Parameters
Ts = 0.01
Tstop = 25
xk = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)

# Simulation
for k in range(N):
    xk1 = xk + Ts* a * xk
    xk = xk1
    data.append(xk1)

# Plot the Simulation Results
t = np.arange(0,Tstop+Ts,Ts)

plt.plot(t,data)
plt.title('Plotting Differential Equation Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.show()
```

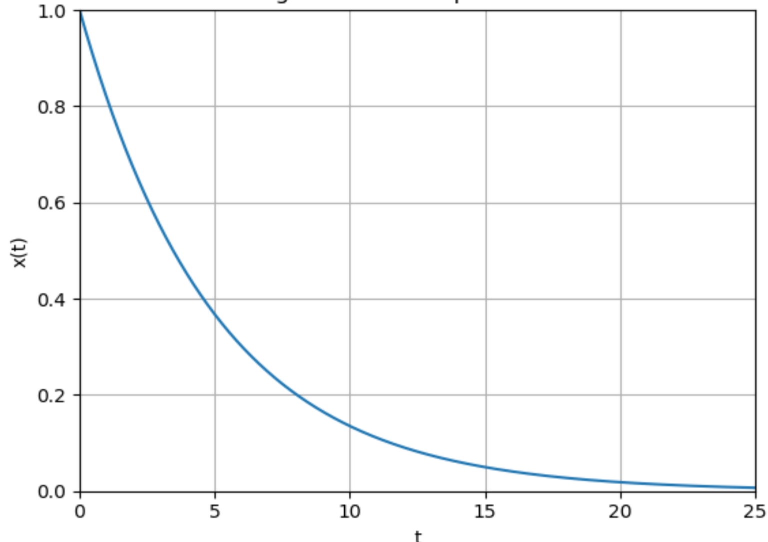
Alt. Code

(using Arrays)

Differential Equation: $\dot{x} = ax$

Simulation of Discrete System:

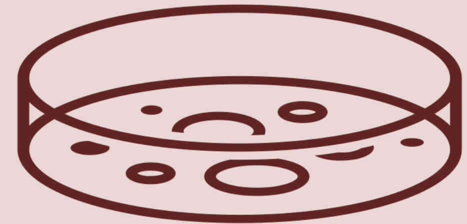
$$x_{k+1} = (1 + aT_s)x_k$$




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

# Model Parameters
T = 5
a = -1/T
# Simulation Parameters
Ts = 0.01
Tstop = 25


N = int(Tstop/Ts) # Simulation length
x = np.zeros(N+2) # Initialization the x vector
x[0] = 1 # Initial Condition


# Simulation
for k in range(N+1):
    x[k+1] = (1 + a*Ts) * x[k]

# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) # Create Time Series

plt.plot(t,x)
plt.title('Plotting Differential Equation Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.show()
```

More Examples

Bacteria Simulation



Hans-Petter Halvorsen

Bacteria Simulation

In this example we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

Birth rate: bx
Death rate: px^2

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2$$

Where x is the number of bacteria in the jar

Note that $\dot{x} = \frac{dx}{dt}$

We will simulate the number of bacteria in the jar after **1 hour**, assuming that initially there are **100 bacteria** present.

In the simulations we can set b=1/hour and p=0.5 bacteria-hour

Discretization

The differential equation of the system is:

$$\dot{x} = bx - px^2$$

We need to find the discrete version:

We use the Euler forward method:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

This gives:

$$\frac{x_{k+1} - x_k}{T_s} = bx_k - px_k^2$$

Next:

$$x_{k+1} - x_k = T_s(bx_k - px_k^2)$$

This gives the following discrete version:

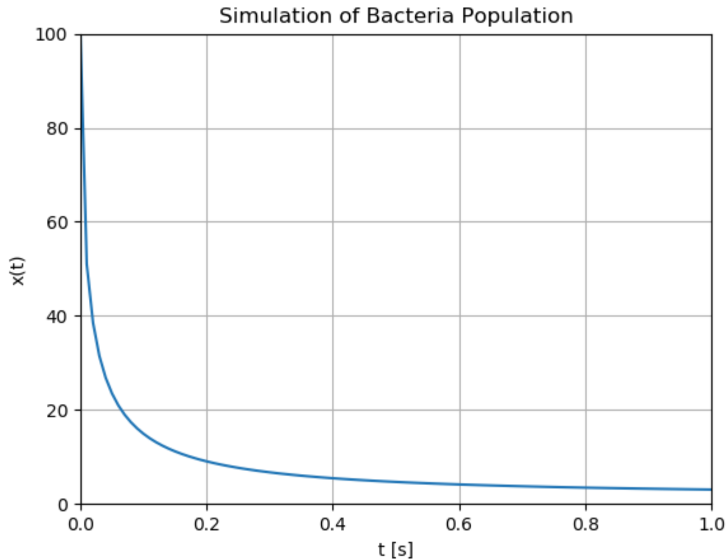
$$x_{k+1} = x_k + T_s(bx_k - px_k^2)$$

Python Code

Differential Equation: $\dot{x} = bx - px^2$

Simulation of Discrete System:

$$x_{k+1} = x_k + T_s(bx_k - px_k^2)$$




```
# Simulation of Bacteria Population
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.01
Tstop = 1
xk = 100
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)

# Simulation
for k in range(N):
    xk1 = xk + Ts* (b * xk - p * xk**2);
    xk = xk1
    data.append(xk1)

# Plot the Simulation Results
t = np.arange(0,Tstop+Ts,Ts)

plt.plot(t,data)
plt.title('Simulation of Bacteria Population')
plt.xlabel('t [s]')
plt.ylabel('x(t)')
plt.grid()
plt.xlim(0, 1)
plt.ylim(0, 100)
plt.show()
```

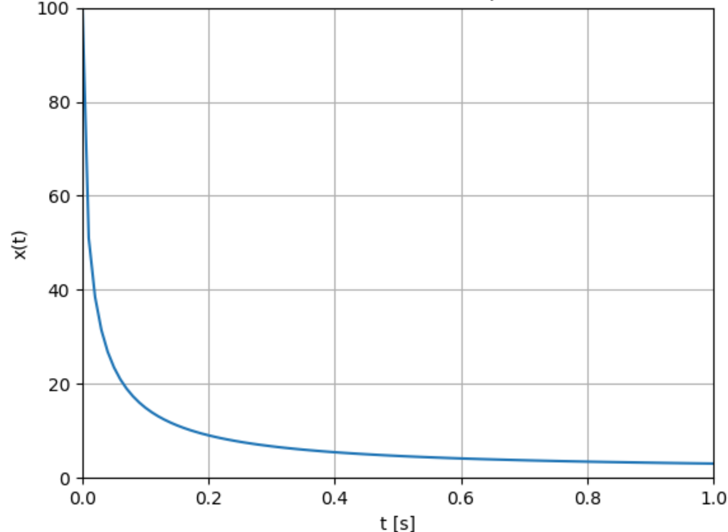
Alt. Code

(using Arrays)

Differential Equation: $\dot{x} = bx - px^2$

Simulation of Discrete System:

$$x_{k+1} = x_k + T_s(bx_k - px_k^2)$$




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

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.01
Tstop = 1

N = int(Tstop/Ts) # Simulation length
x = np.zeros(N+2) # Initialization the x vector
x[0] = 100 # Initial Condition

# Simulation
for k in range(N+1):
    x[k+1] = x[k] + Ts * (b * x[k] - p * x[k]**2)

# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) # Create Time Series

plt.plot(t,x)
plt.title('Simulation of Bacteria Population')
plt.xlabel('t [s]')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 1, 0, 100])
plt.show()
```

Simulations with 2 variables

Hans-Petter Halvorsen

Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2$$

$$\frac{dx_2}{dt} = x_1$$

Let's find the discrete system and simulate the discrete system in Python.

Discretization

Using Euler:

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

Then we get:

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

Which we can implement in Python

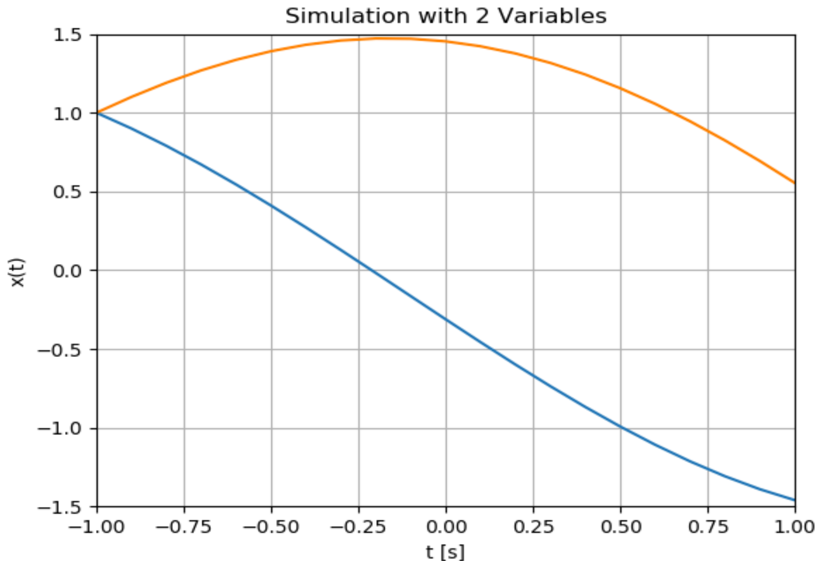
The equations will be solved in the time span [−1 1] with initial values [1, 1].

Python Code

$$\frac{dx_1}{dt} = -x_2$$

$$\frac{dx_2}{dt} = x_1$$

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




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

# Model and Simulation Parameters
b = 1
p = 0.5
Ts = 0.1
Tstart = -1
Tstop = 1
x1k = 1
x2k = 1
N = int((Tstop-Tstart)/Ts) # Simulation length
datax1 = []
datax2 = []
datax1.append(x1k)
datax2.append(x2k)

# Simulation
for k in range(N):
   x1k1 = x1k - Ts * x2k
   x2k1 = x2k + Ts * x1k

   x1k = x1k1
   x2k = x2k1
   datax1.append(x1k1)
   datax2.append(x2k1)

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

plt.plot(t,datax1, t, datax2)
plt.title('Simulation with 2 Variables')
plt.xlabel('t [s]')
plt.ylabel('x(t)')
plt.grid()
plt.axis([-1, 1, -1.5, 1.5])
plt.show()
```

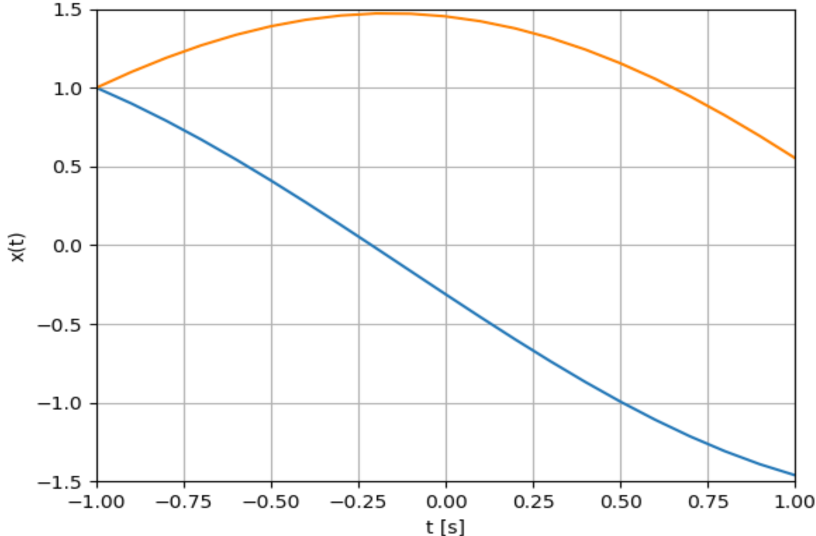
Alt. Code

(using Arrays)

$$\frac{dx_1}{dt} = -x_2$$

$$\frac{dx_2}{dt} = x_1$$

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



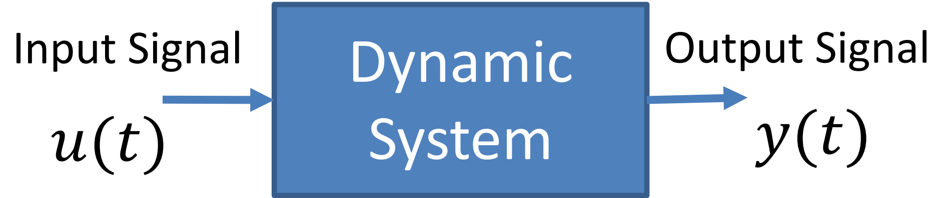

```
# Simulation with 2 Variables
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.1
Tstart = -1
Tstop = 1
N = int((Tstop-Tstart)/Ts) # Simulation length
x1 = np.zeros(N+2)
x2 = np.zeros(N+2)
x1[0] = 1
x2[0] = 1

# Simulation
for k in range(N+1):
    x1[k+1] = x1[k] - Ts * x2[k]
    x2[k+1] = x2[k] + Ts * x1[k]

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

plt.plot(t,x1, t, x2)
plt.title('Simulation with 2 Variables')
plt.xlabel('t [s]')
plt.ylabel('x(t)')
plt.grid()
plt.axis([-1, 1, -1.5, 1.5])
plt.show()
```

Simulation of 1.order System

Hans-Petter Halvorsen

1.order Dynamic System

Assume the following general Differential Equation:



$$\dot{y} = ay + bu$$

or

$$\dot{y} = \frac{1}{T}(-y + Ku)$$

Where $a = -\frac{1}{T}$ and $b = \frac{K}{T}$

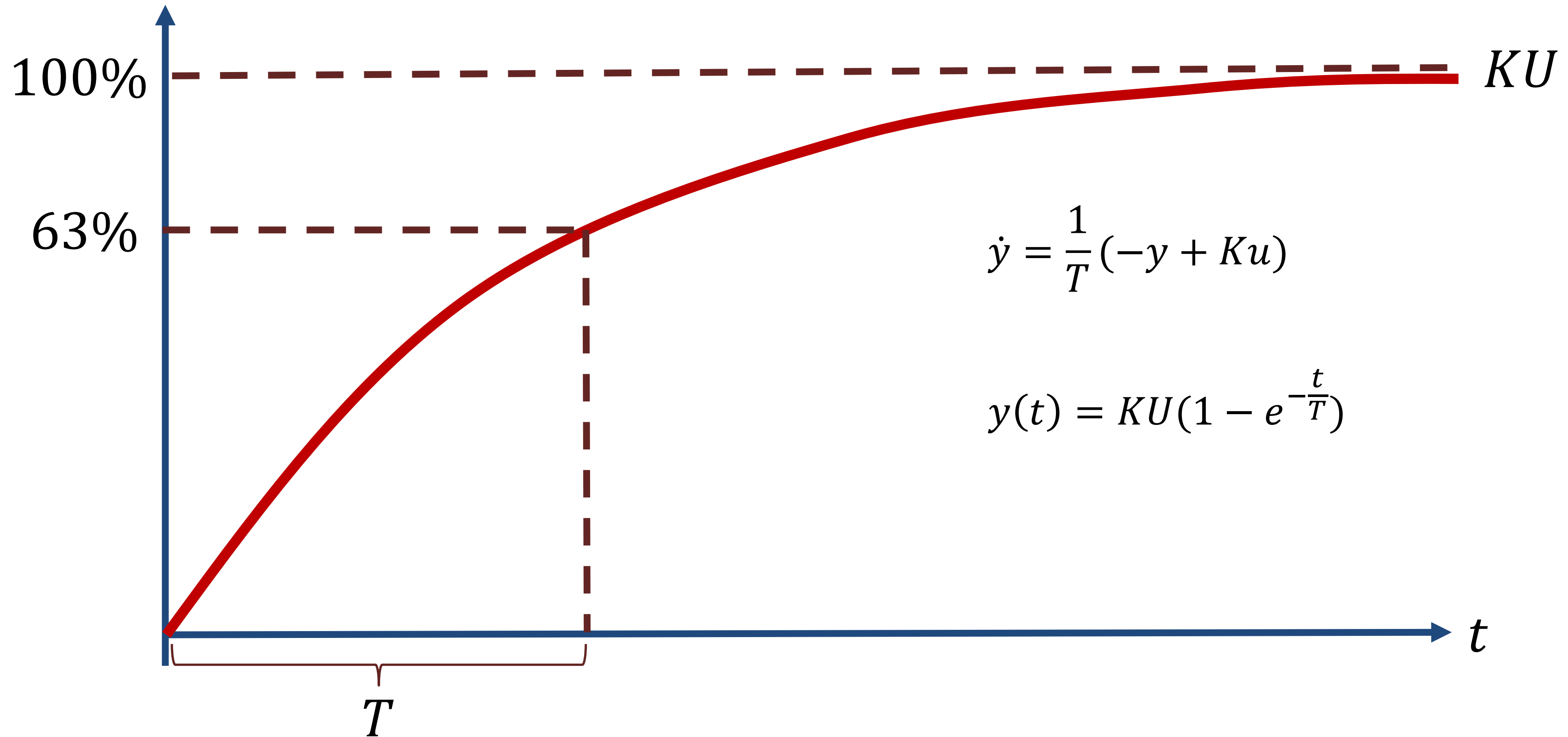
Where K is the Gain and T is the Time constant

This differential equation represents a 1. order dynamic system

Assume $u(t)$ is a step (U), then we can find that the solution to the differential equation is:

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

Step Response



1.order Dynamic System

Given the differential equation: $\dot{y} = \frac{1}{T}(-y + Ku)$

Let's find the mathematical expression for the step response

We use Laplace:

Note $\dot{y} \Leftrightarrow sy(s)$

$$sy(s) = \frac{1}{T}(-y(s) + Ku(s))$$

$$sy(s) + \frac{1}{T}y(s) = \frac{K}{T}u(s)$$

$$Tsy(s) + y(s) = Ku(s)$$

$$(Ts + 1)y(s) = Ku(s)$$

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

We apply a step in the input signal $u(s)$: $u(s) = \frac{U}{s}$

$$y(s) = \frac{K}{Ts + 1} \cdot \frac{U}{s}$$

Next, we use Inverse Laplace

We use the following Laplace Transformation pair in order to find $y(t)$: $\frac{k}{(Ts + 1)s} \Leftrightarrow k(1 - e^{-\frac{t}{T}})$

This gives the following step response:

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

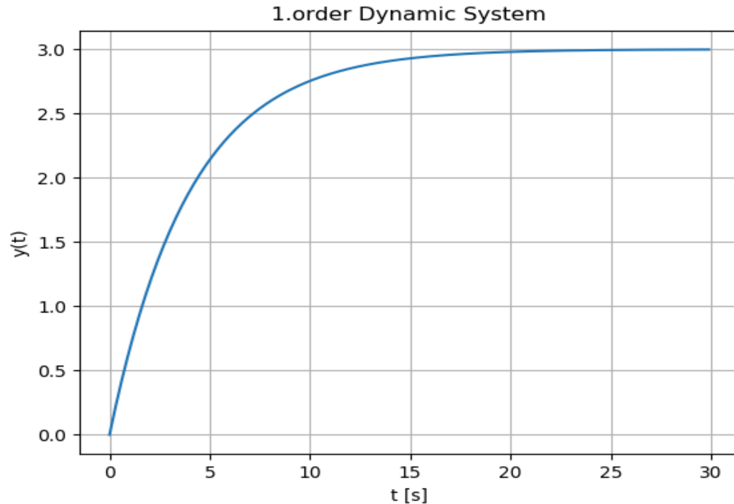
Python Code

We start by plotting the following:

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

In the Python code we can set:

$U = 1$
$K = 3$
$T = 4$




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

K = 3
T = 4
start = 0
stop = 30
increment = 0.1
t = np.arange(start,stop,increment)

y = K * (1-np.exp(-t/T))

plt.plot(t, y)
plt.title('1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)'
plt.grid()
```

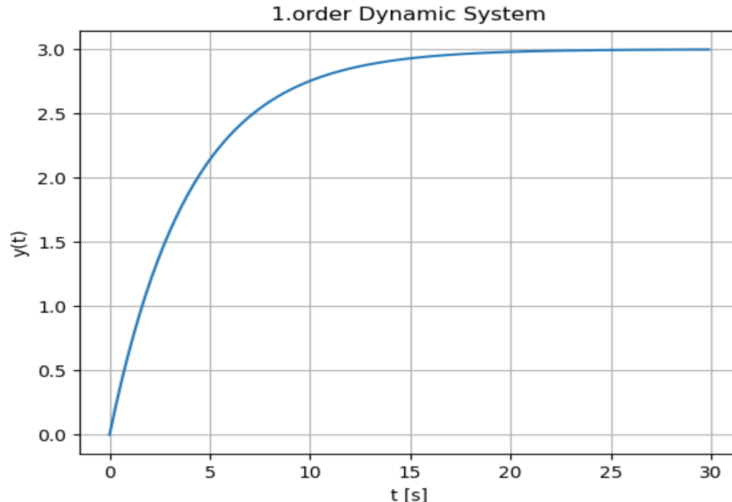
Python Code (Alternative Code)

We start by plotting the following:

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

In the Python code we can set:

$$U = 1$$
$$K = 3$$
$$T = 4$$




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

def model(t):
    K = 3
    T = 4
    y = K * (1-np.exp(-t/T))
    return y

start = 0
stop = 30
increment = 0.1
t = np.arange(start,stop,increment)

y = model(t)

plt.plot(t, y)
plt.title('1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)'
plt.grid()
```

Discretization

We start with the differential equation:

$$\dot{y} = ay + bu$$

We use the Euler forward method:

$$\dot{y} \approx \frac{y_{k+1} - y_k}{T_s}$$

This gives:

$$\frac{y_{k+1} - y_k}{T_s} = ay_k + bu_k$$

Further we get:

$$y_{k+1} = y_k + T_s(ay_k + bu_k)$$

$$y_{k+1} = y_k + T_s a y_k + T_s b u_k$$

This gives the following discrete differential equation:

$$y_{k+1} = (1 + T_s a) y_k + T_s b u_k$$

Python Code

Let's simulate the discrete system:

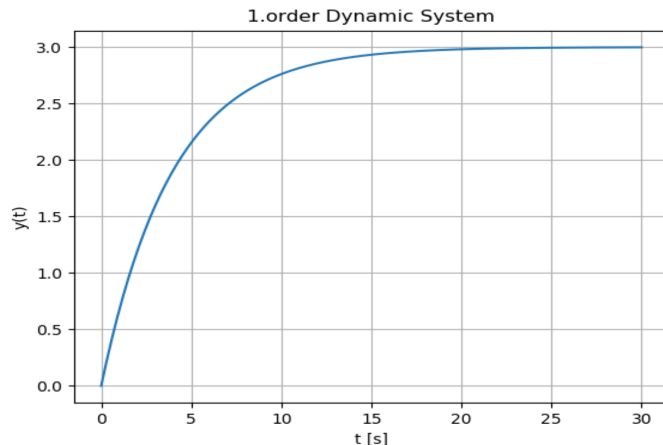
$$y_{k+1} = (1 + T_s a) y_k + T_s b u_k$$

Where $a = -\frac{1}{T}$ and $b = \frac{K}{T}$

In the Python code we can set:

$K = 3$

$T = 4$




```
# Simulation of discrete model
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
K = 3
T = 4

a = -1/T
b = K/T

# Simulation Parameters
Ts = 0.1
Tstop = 30
uk = 1 # Step Response
yk = 0 # Initial Value
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)


# Simulation
for k in range(N):
   yk1 = (1 + a*Ts) * yk  + Ts * b * uk
   yk = yk1
   data.append(yk1)


# Plot the Simulation Results
t = np.arange(0,Tstop+Ts,Ts)

plt.plot(t,data)
plt.title('1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

Alt. Solution

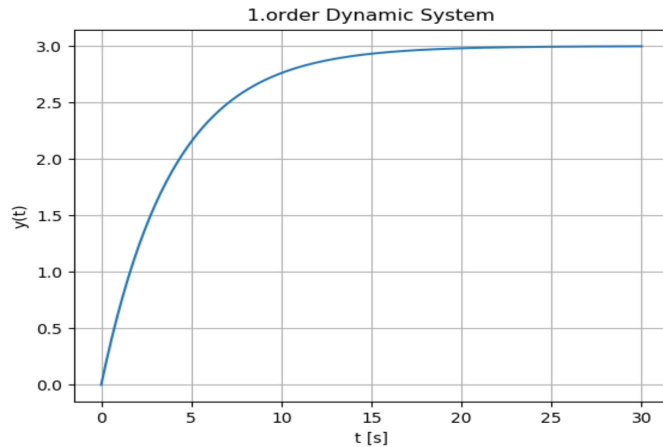
Let's simulate the discrete system:

$$y_{k+1} = (1 + T_s a)y_k + T_s b u_k$$

Where $a = -\frac{1}{T}$ and $b = \frac{K}{T}$

In the Python code we can set:

$K = 3$
$T = 4$




```
# Simulation of discrete model
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
K = 3
T = 4

a = -1/T
b = K/T

# Simulation Parameters
Ts = 0.1 # Sampling Time
Tstop = 30 # End of Simulation Time
uk = 1 # Step Response
N = int(Tstop/Ts) # Simulation length
y = np.zeros(N+2) # Initialization the x vector
y[0] = 0


# Simulation
for k in range(N+1):
    y[k+1] = (1 + a*Ts) * y[k]  + Ts * b * uk


# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) # Create Time Series
plt.plot(t,y)
# Formatting the appearance of the Plot
plt.title('1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

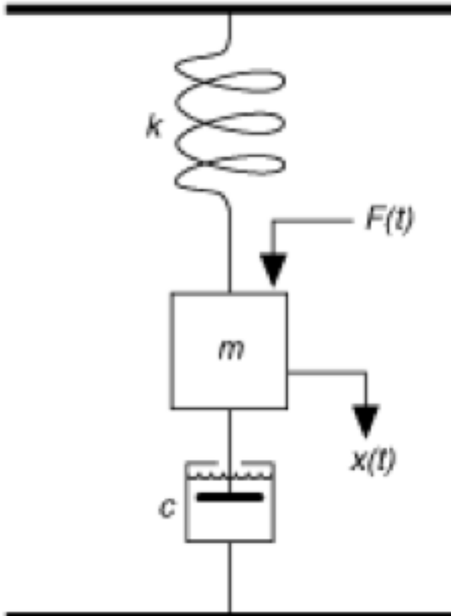
Higher Order Systems

Hans-Petter Halvorsen

Mass-Spring Damper

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

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

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

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

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

We set

$x = x_1$

$\dot{x} = x_2$

This gives:

$\dot{x}_1 = x_2$

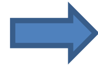
$\dot{x}_2 = \ddot{x} = \frac{1}{m}(F - c\dot{x} - kx) = \frac{1}{m}(F - cx_2 - kx_1)$

x_1 = Position

x_2 = Velocity/Speed

Finally:

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



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

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

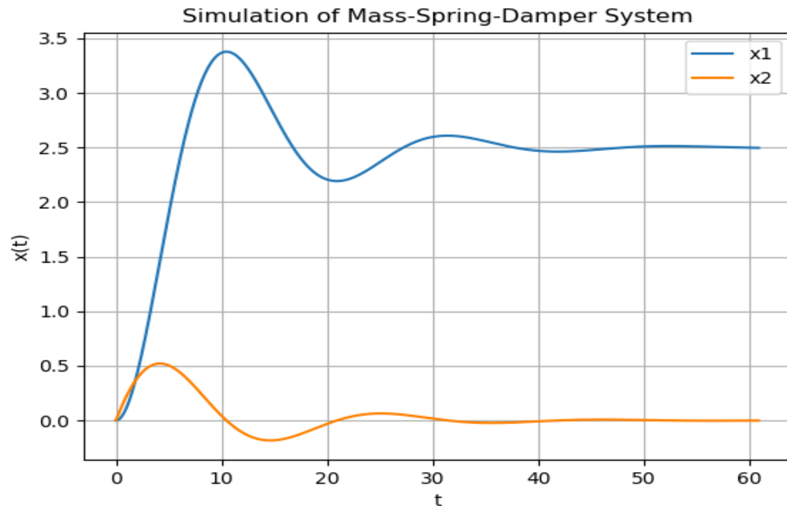
Python Code

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




```python
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

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

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

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

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

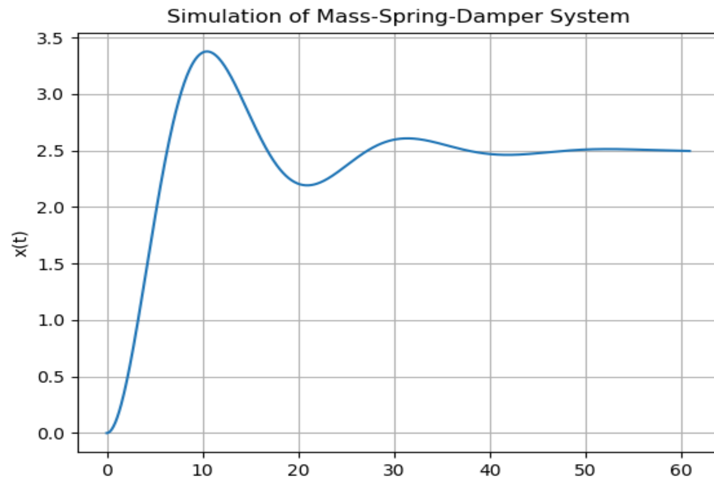
$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Python Code

State-space Model

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

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



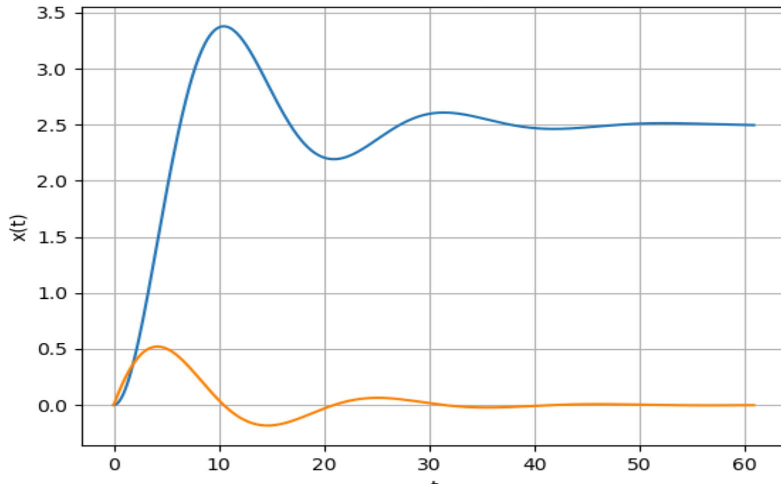

```python
import numpy as np
import matplotlib.pyplot as plt
import control

# 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()
```

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:

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

We can set it on Discrete state space form:

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

This gives:

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

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

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

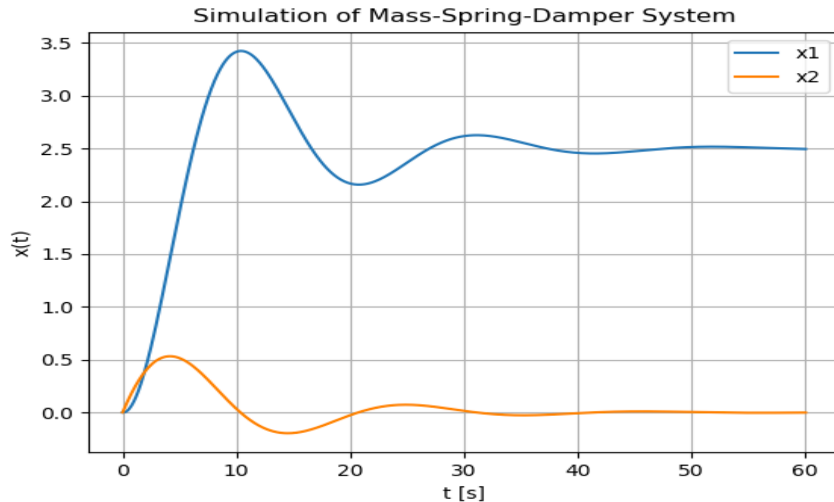
$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

Python Code

Discrete System

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

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



x_1 = Position

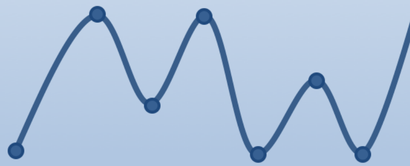
x_2 = Velocity/Speed

```
# 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

Python Programming

Hans-Petter Halvorsen



https://www.halvorsen.blog

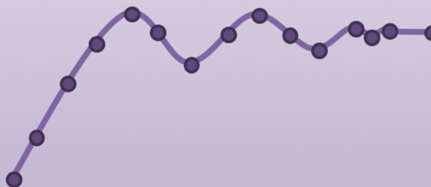
Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

Hans-Petter Halvorsen



https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen

https://www.halvorsen.blog

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