https://www.halvorsen.blog

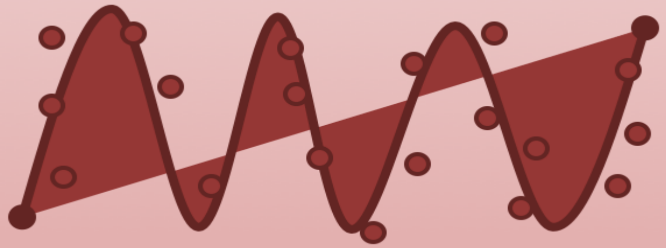
Differential Equations in Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Science and Engineering

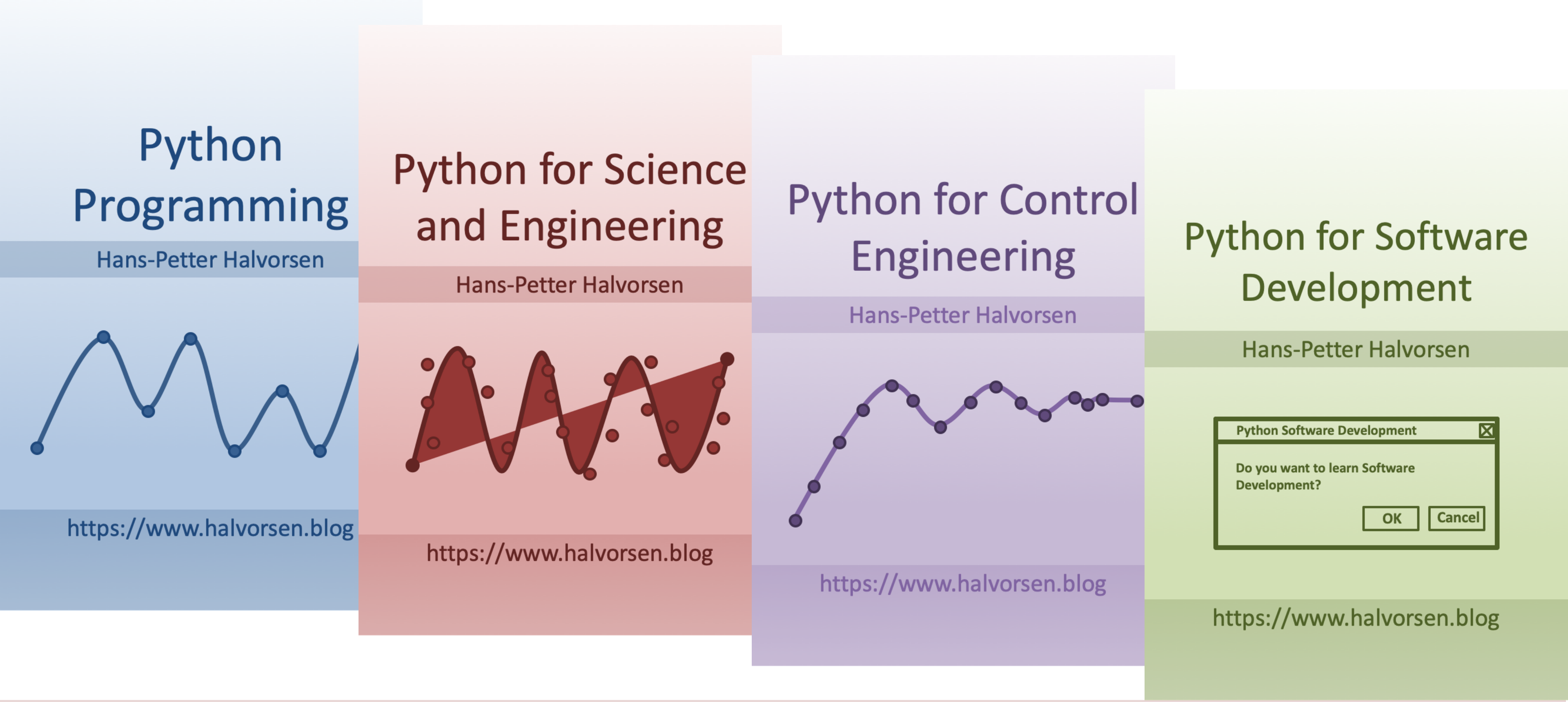
Hans-Petter Halvorsen



https://www.halvorsen.blog

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

Additional Python Resources



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

Contents

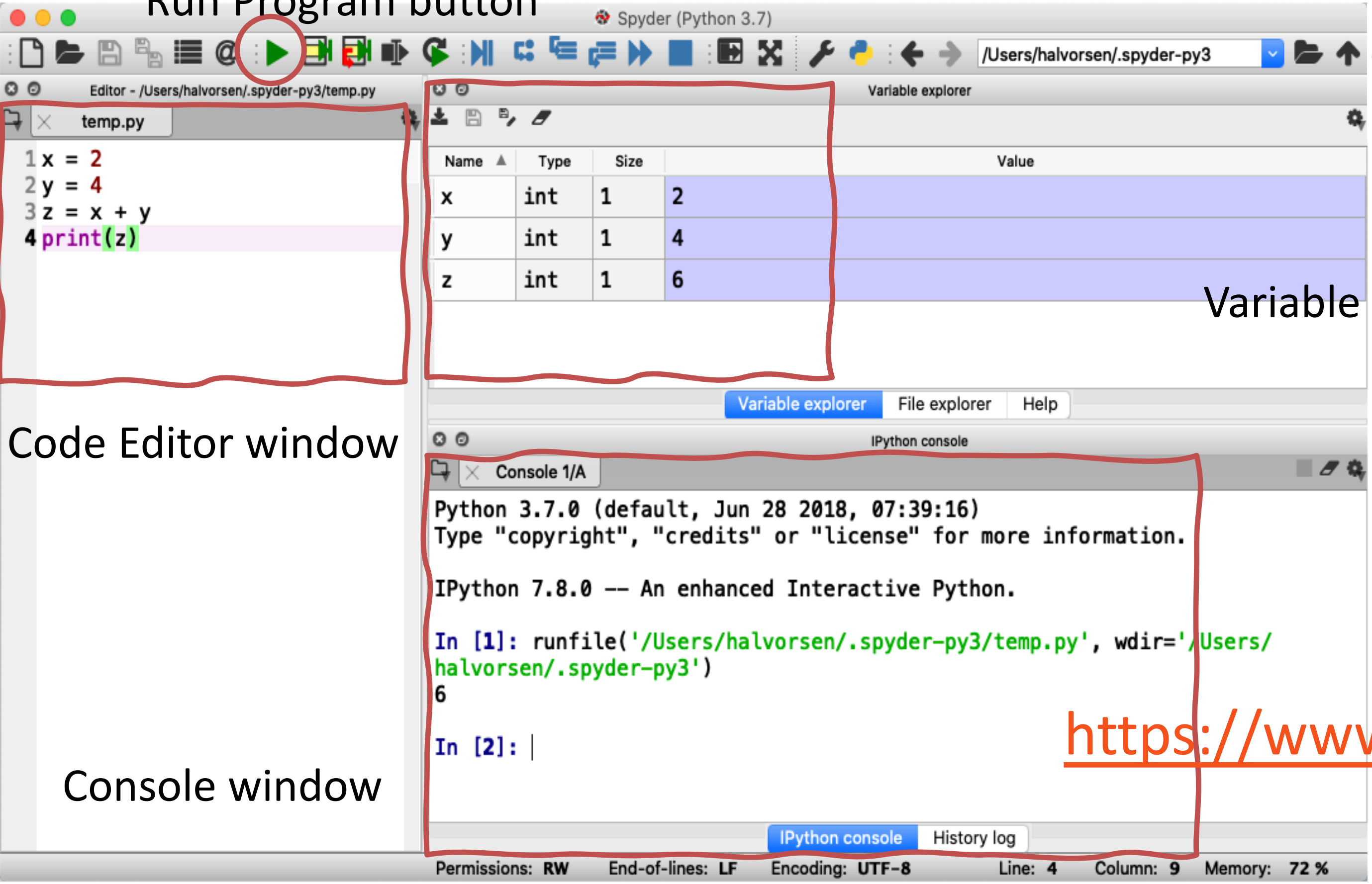
- Differential Equations
- Simulation
- ODE Solvers
- Discrete Systems
- Examples

To benefit from the tutorial, you should already know about differential equations

Python Editors

- Python IDLE
- **Spyder** (Anaconda distribution)
- PyCharm
- **Visual Studio Code**
- Visual Studio
- Jupyter Notebook
- …

Spyder (Anaconda distribution)



SPYDER

The Scientific Python Development Environment

Variable Explorer window

https://www.anaconda.com

Differential Equations

Hans-Petter Halvorsen

Differential Equations

Differential Equation on general form:

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

Initial condition

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):

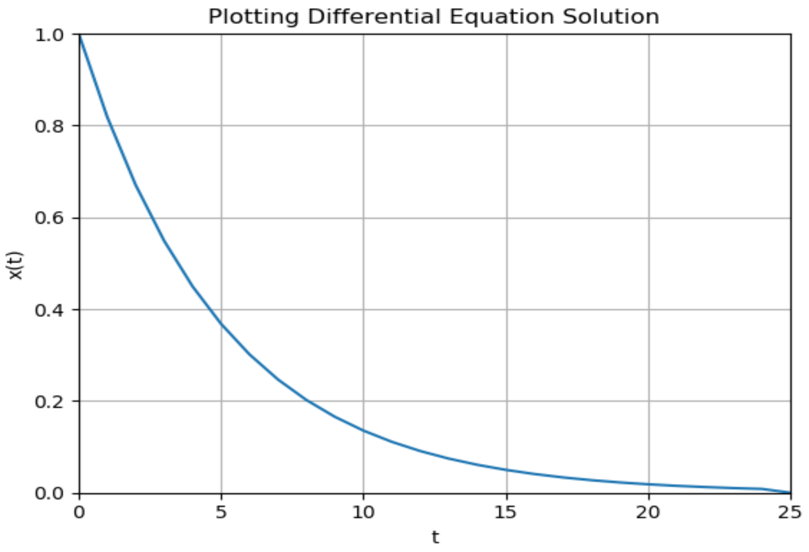
$x(t) = e^{at}x_0$

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

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

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()
```

Comments to Example

- 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

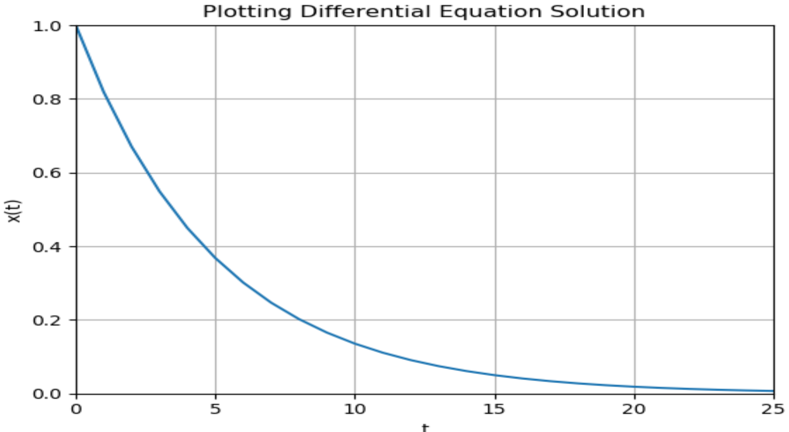
- The **scipy.integrate** library has two powerful powerful functions; ode() and odeint(), for numerically solving first order ordinary differential equations (ODEs).
- The ode() is more flexible, while odeint() (ODE integrator) has a simpler Python interface and works fine for most problems.
- For details, 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 $T = 5$ 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()
```

Passing Argument

$$\dot{x} = ax$$

We set $T = 5, a = -\frac{1}{T}$ and $x_0 = x(0) = 1$

In the modified example we have the parameters used in the differential equation (in this case a) as an input argument.

By doing this, it is very easy to change values for the parameters used in the differential equation without changing the code for the differential equation.

The differential equation can even be in a separate file.

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

# Initialization
T = 5
a = -1/T
tstart = 0
tstop = 25
increment = 1
x0 = 1
t = np.arange(tstart,tstop+1,increment)

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

# Solve ODE
x = odeint(mydiff, x0, t, args=(a,))
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()
```

Python Comments

Passing Arguments

You can also easily run multiple simulations like this:

```
..

a = -0.2
x = odeint(mydiff, x0, t, args=(a,))

a = -0.1
x = odeint(mydiff, x0, t, args=(a,))

..
```

Then you can run multiple simulations for different values of a.

To write a tuple containing a single value you have to include a comma, even though there is only one value

`args=(a,)`

Diff. Eq. in Separate .py File

"differential_equations.py":

```
def mydiff1(x, t, a):
    dxdt = a * x
    return dxdt
```

"test_mydiffq.py":

```
from differential_equations import mydiff1
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Initialization
T = 5
a = -1/T
tstart = 0
tstop = 25
increment = 1
x0 = 1
t = np.arange(tstart,tstop+1,increment)

# Solve ODE
x = odeint(mydiff1, x0, t, args=(a,))
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()
```

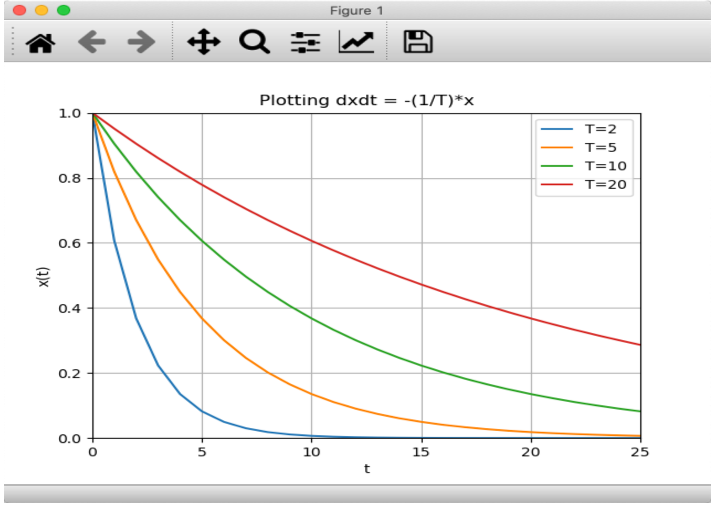
Diff. Eq. in For Loop

$\dot{x} = ax$

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

"differential_equations.py":

```
def mydiff1(x, t, a):
    dxdt = a * x
    return dxdt
```



"test_mydiffq.py":

```
from differential_equations import mydiff1
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Initialization
Tsimulations = [2, 5, 10, 20]

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


for T in Tsimulations:
    a = -1/T

    # Solve ODE
    x = odeint(mydiff1, x0, t, args=(a,))
    print(x)

    # Plot the Results
    plt.plot(t,x)


plt.title('Plotting dxdt = -(1/T)*x')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.legend(["T=2", "T=5", "T=10", "T=20"])
plt.show()
```

Discretization

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:

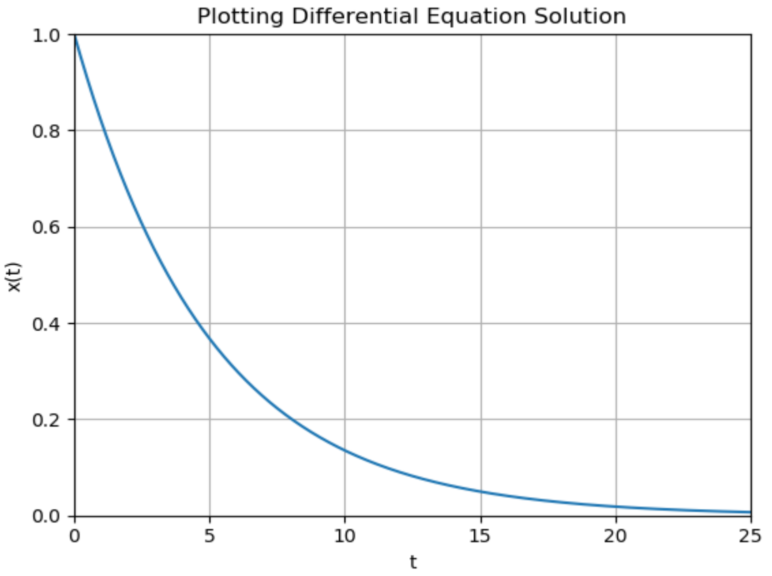
$$x_{k+1} = (1 + aT_s)\, x_k$$

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


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()
```

Discretization

- Discretization are covered more in detail in another Tutorial/Video
- You find also more about Discretization in my Textbook “Python for Science and Engineering“

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

More Examples

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

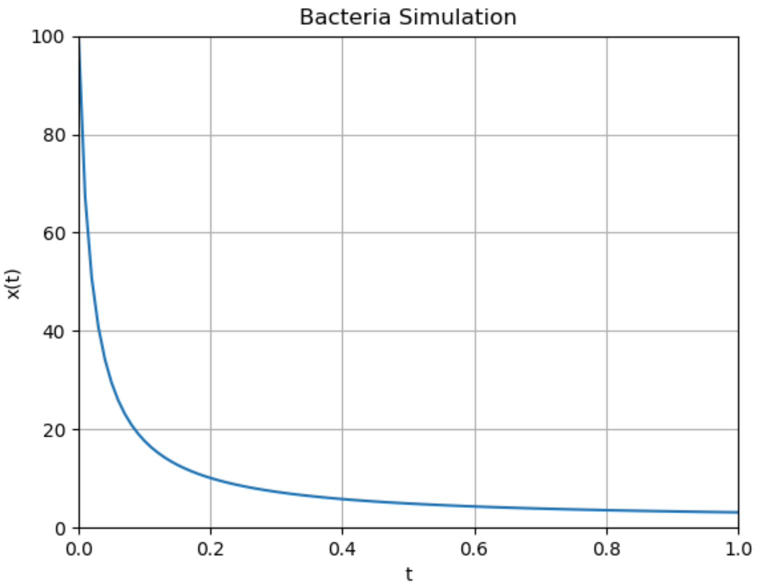
Note that $\dot{x} = \frac{dx}{dt}$

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

Python Code

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




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

# Initialization
tstart = 0
tstop = 1
increment = 0.01
x0 = 100
t = np.arange(tstart,tstop+increment,increment)

# Function that returns dx/dt
def bacteriadiff(x, t):
    b = 1
    p = 0.5
    dxdt = b * x - p * x**2
    return dxdt

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

# Plot the Results
plt.plot(t,x)
plt.title('Bacteria Simulation')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([0, 1, 0, 100])
plt.show()
```

Simulation with 2 variables

Given the following system:

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

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

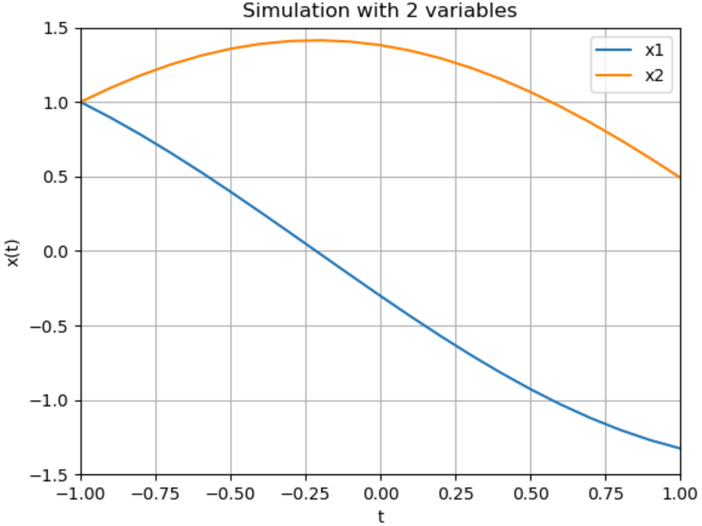
Let's simulate the system in Python. The equations will be solved in the time span [−1 1] with initial values [1, 1].

Python Code

System:

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

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




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

# Initialization
tstart = -1
tstop = 1
increment = 0.1
x0 = [1,1]

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

# Function that returns dx/dt
def mydiff(x, t):
    dx1dt = -x[1]
    dx2dt = x[0]

    dxdt = [dx1dt,dx2dt]
    return dxdt

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

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

# Plot the Results
plt.plot(t,x1)
plt.plot(t,x2)
plt.title('Simulation with 2 variables')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.grid()
plt.axis([-1, 1, -1.5, 1.5])
plt.legend(["x1", "x2"])
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 $b = \frac{K}{T}$

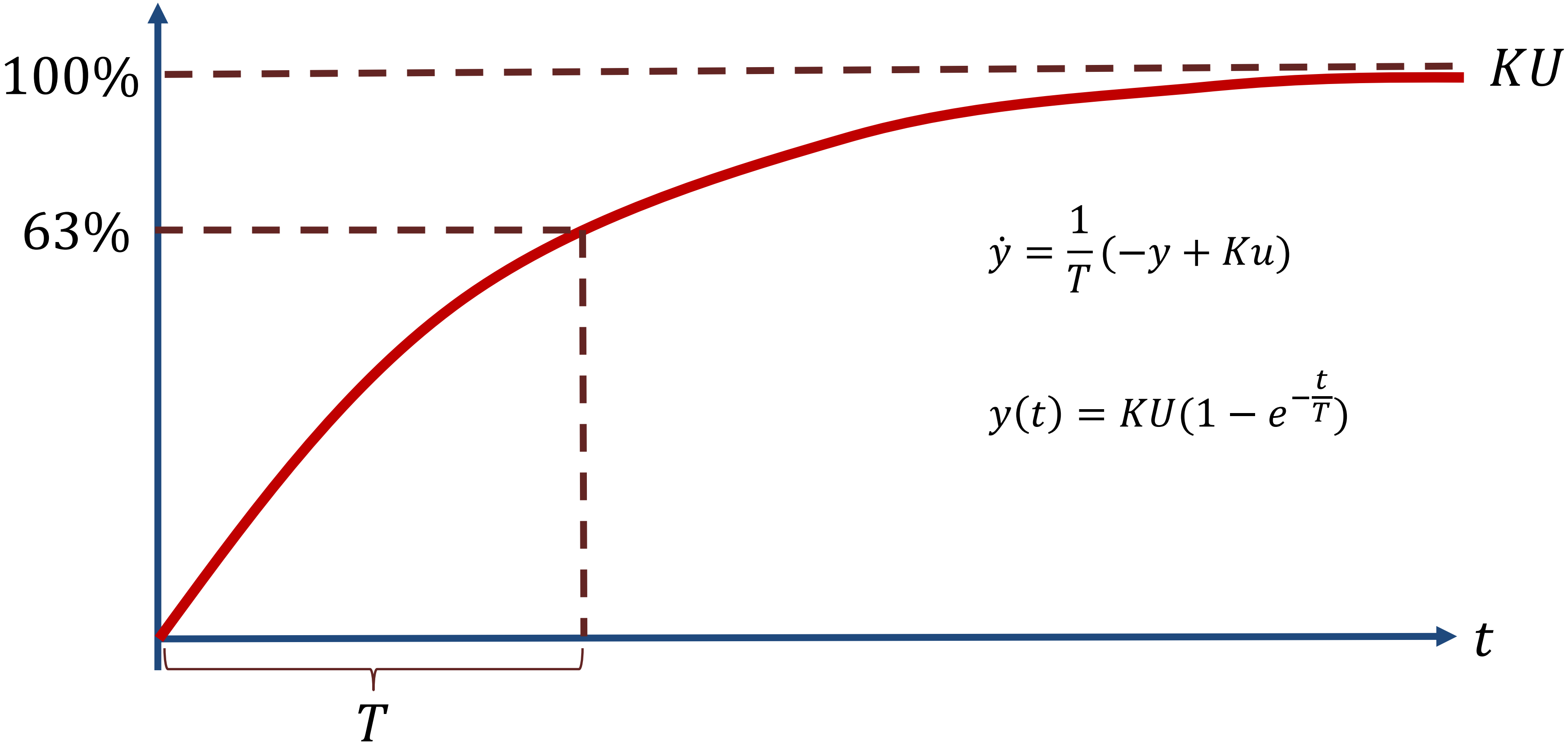
Where K is the Gain and T is the Time constant

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



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

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

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:

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

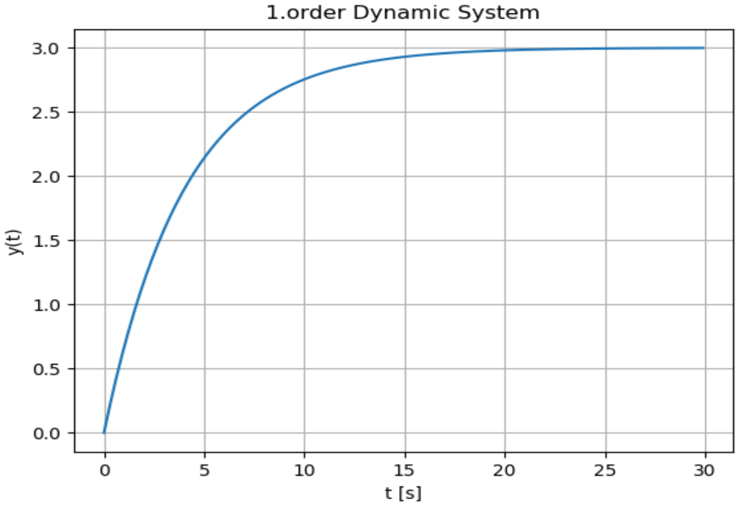
Python Code

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()
```

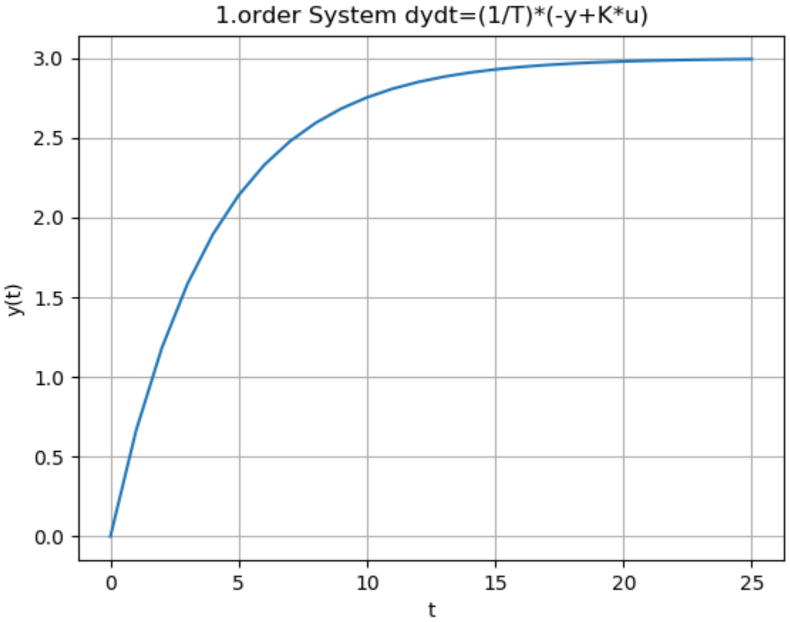
Python Code

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

In the Python code we can set:

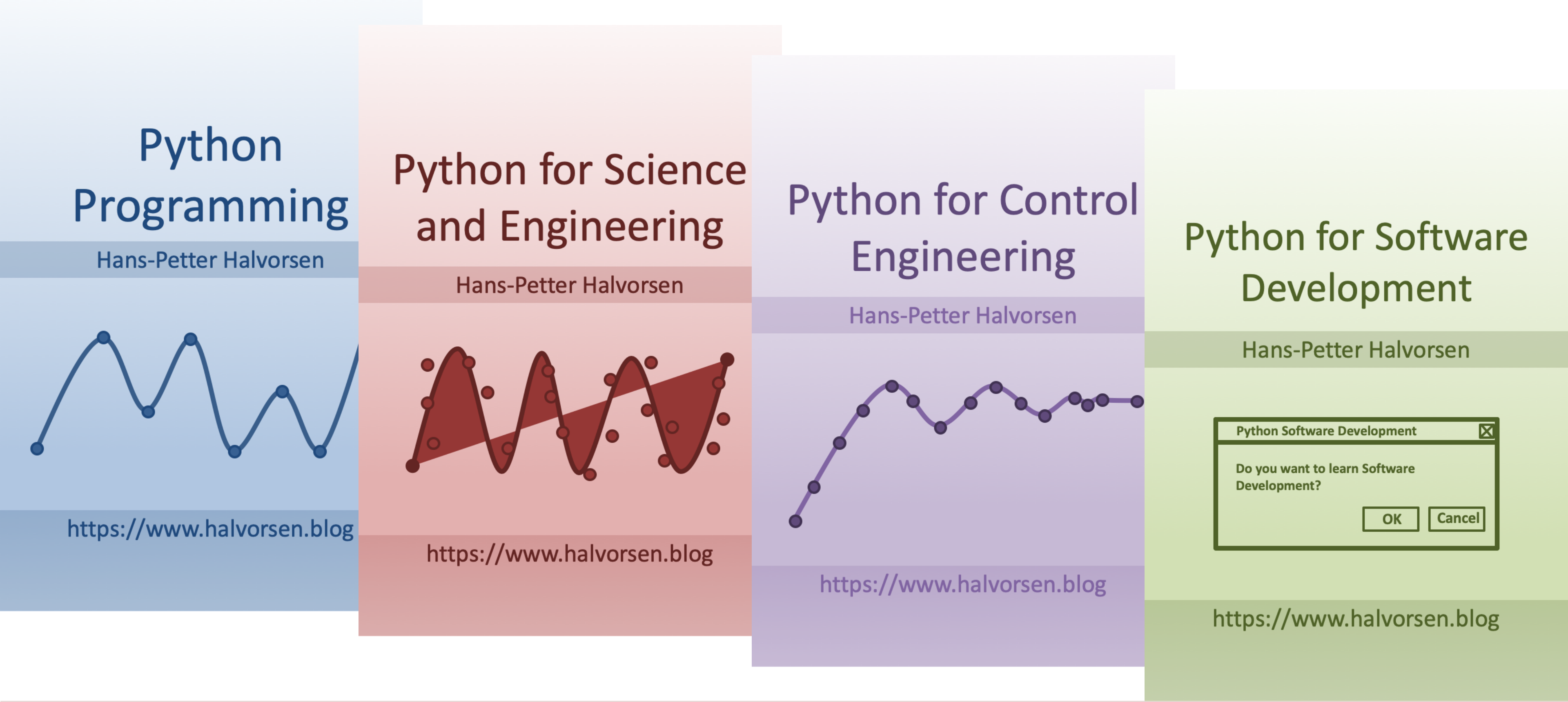
$K = 3$

$T = 4$




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

# Initialization
K = 3
T = 4
u = 1
tstart = 0
tstop = 25
increment = 1
y0 = 0
t = np.arange(tstart,tstop+1,increment)

# Function that returns dx/dt
def system1order(y, t, K, T, u):
    dydt = (1/T) * (-y + K*u)
    return dydt

# Solve ODE
x = odeint(system1order, y0, t, args=(K, T, u))
print(x)

# Plot the Results
plt.plot(t,x)
plt.title('1.order System dydt=(1/T)*(-y+K*u)')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.grid()
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