https://www.halvorsen.blog

Simulation and Control in Python

Hans-Petter Halvorsen

Contents

Simulation of Model

Model

Differential equation (1.order dynamic system):

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

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

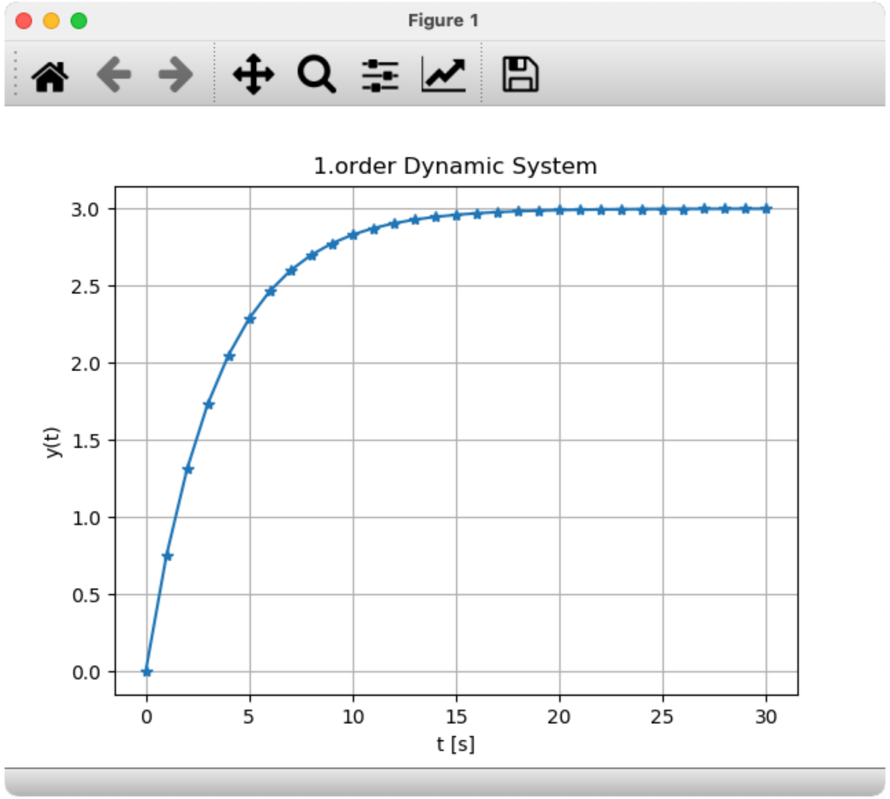
Discrete version that we can use in our Simulations:

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

In the Python code we can, e.g., use the following values in the Simulations:

$$K = 3$$
$$T = 4$$

Example 1



```
import numpy as np
import matplotlib.pyplot as plt
# Model Parameters
K = 3
T = 4
a = -1/T
b = K/T

#Simulation Parameters
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length

data = []
data.append(yk)

# Simulation
for k in range(N):
    #Model Implementation
    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()
```

Python Functions

The **Name** of the Function

Input **Arguments** (information that are passed into a function)

```
def add(x,y):
        z = x + y
        return z
```

The **Return** value

```
# Define Function
def add(x,y):
    return x + y

# Using the Function:
x = 2
y = 5

z = add(x,y)

print(z)
```

Variables and Functions

- Variables that are created outside of a function are known as **global variables**.
- Global variables can be used by everyone, both inside of functions and outside.
- Normally, when you create a variable inside a function, that variable is **local**, and can only be used inside that function.
- **If you create a variable with the same name inside a function, this variable will be local**, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Global vs Local Variables

```
x = 3

def myfunc():
  x = 4
  print(x)

myfunc()

print(x)
```

Result in Console window:

```
4
3
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example 2

In Example 2, the Model has been put into a **Function**

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

#Simulation Parameters
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length

data = []
data.append(yk)

def BasicModel(yk):
    # Model Parameters
    K = 3
    T = 4
    a = -1/T
    b = K/T

    #Model Implementation
    yk1 = (1 + a*Ts) * yk + Ts*b*uk

    return yk1

# Simulation
for k in range(N):
   yk1 = BasicModel(yk)
   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()
```

Python Functions in a Separate File

- Although you can mix functions and code in one file, it is much better to create the functions in separate .py files
- In that way you can easily reuse the function in different Python scripts

1

We start by creating a separate Python File, e.g., "myfunctions.py" for the function:

myfunctions.py:

```
def average(x,y):

    return (x + y)/2
```

2 Next, we create a new Python File (e.g., testaverage.py) where we use the function we created:

```
from myfunctions import average

a = 2
b = 3

c = average(a,b)

print(c)
```

Example 3

In Example 3, the Model has been put into a **Function** that is put into a **separate Python File**

model.py

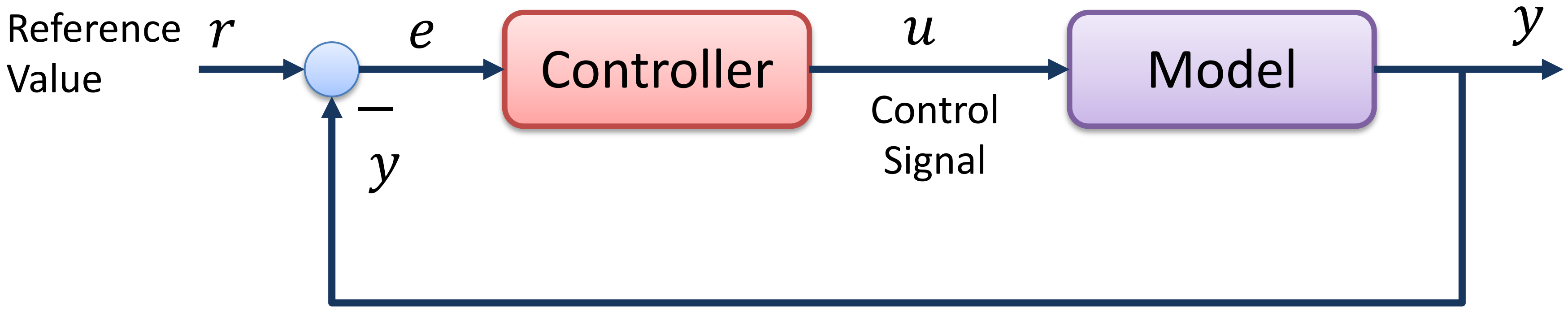
```python
def BasicModel(Ts, yk, uk):
    # Model Parameters
    K = 3
    T = 4
    a = -1/T
    b = K/T

    #Model Implementation
    yk1 = (1 + a*Ts) * yk + Ts*b*uk

    return yk1
```

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

#Simulation Parameters
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length

data = []
data.append(yk)


# Simulation
for k in range(N):
   yk1 = model.BasicModel(Ts, yk, 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()
```

Control System

Control System

The purpose with a Control System is to Control a Dynamic System, e.g., an industrial process, an airplane, a self-driven car, etc. (a Control System is “everywhere“).



PI Controller

$$u(t) = K_p e + \frac{K_p}{T_i}\int_0^t e d\tau$$

Discrete PI Controller that we can use in our Simulations:

$$e_k = r_k - y_k$$

$$u_k = u_{k-1} + K_p(e_k - e_{k-1}) + \frac{K_p}{T_i} T_s e_k$$

Example 4

controlsystem.py

```
def Model(Ts, yk, uk):
    # Model Parameters
    K = 3
    T = 4
    a = -1/T
    b = K/T

    #Model Implementation
    yk1 = (1 + a*Ts) * yk + Ts*b*uk

    return yk1


def Controller(Ts, y, r, u_prev, e_prev):
    Kp = 0.5
    Ti = 5

    e = r - y
    u = u_prev + Kp*(e - e_prev) + (Kp/Ti)*Ts*e

    return u, e
```

In Example 4 we Control the Model using a PI Controller. The Model ad the Controller have been put into separate **Functions** that is put into a separate **Python File**

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

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

# Controller Initialization
r = 1
u_prev = 0
e_prev = 0

# Simulation
for k in range(N):
    u, e = cs.Controller(Ts, yk, r, u_prev, e_prev)
    u_prev = u
    e_prev = e

    yk1 = cs.Model(Ts, yk, u)
    yk = yk1
    data.append(yk1)


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

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

Python Classes

Class Example:

```
class Car:
    model = ""
    color = ""

car = Car()

car.model = "Volvo"
car.color = "Blue"

print(car.color + " " + car.model)

car.model = "Ford"
car.color = "Green"

print(car.color + " " + car.model)
```

Define the Class

We start using the Class by creating an Object of that Class

Set **Properties**

Use the Class

The __init__() Function

In Python all classes have a built-in function called __init__(), which is always executed when the class is being initiated.
In many other OOP languages we call this the **Constructor**.

We will create a simple example where we use the __init__() function to illustrate the principle:

The **self** parameter is a reference to the current instance of the class and is used to access variables that belongs to the class.

```
class Car:
  def __init__(self, model, color):
    self.model = model
    self.color = color

car1 = Car("Ford", "Green")
print(car1.color + " " + car1.model)

car2 = Car("Volvo", "Blue")
print(car2.color + " " + car2.model)
```

Python Classes

- Its normal to use the term "**Method**" for Functions that are defined within a Class.
- You declare class methods like normal functions with the exception that the first argument to each method is **self**.
- To **create instances of a class**, you call the class using class name and pass in whatever arguments its __init__() method accepts, e.g., car1 = Car("Tesla", "Red")

Example 5

Automation.py

```
class Automation:
    def __init__(self, Ts, Kp, Ti):
        self.Ts = Ts
        self.Kp = Kp
        self.Ti = Ti

    def Model(self, yk, uk):
        # Model Parameters
        K = 3
        T = 4
        a = -1/T
        b = K/T

        Ts = self.Ts

        #Model Implementation
        yk1 = (1 + a*Ts) * yk + Ts*b*uk

        return yk1

    def Controller(self, y, r, u_prev, e_prev):
        Kp = self.Kp
        Ti = self.Ti
        Ts = self.Ts

        e = r - y
        u = u_prev + Kp*(e - e_prev) + (Kp/Ti)*Ts*e

        return u, e
```

In Example 5 we have created a separate **Python Class** for the Model and the Controller

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

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

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Simulation
for k in range(N):
   u, e = control.Controller(yk, r, u_prev, e_prev)
   u_prev = u
   e_prev = e

   yk1 = control.Model(yk, u)
   yk = yk1
   data.append(yk1)

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

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

Next Steps

- We started to Simulate a basic **Model** in Python
- Then we improved the Python Code by using **Functions** and **Classes** in Python
- We Implemented also a basic **Control System**
- Next step would be to create a "**real-time control system**" where (not all will not be shown in this tutorial):
  - We update the Plot in every iteration
  - It is possible to change Reference/Setpoint value during execution
  - The Control System runs "forever" until we want to stop it by clicking a button, hitting a key or something

  => These things are more complicated in Python compared to, e.g., LabVIEW or WinForm App in Visual Studio/C#, but it is possible.

Real-Time Plotting

Basic Real-Time Plotting:

We will perform a very basic ”Real-Time” Plotting by putting the plot commands inside the For Loop:

```
for k in range(N):
       ..
       plt.plot(t, yk1, '-o', markersize=1, color='blue’)
       plt.show()
       plt.pause(Ts)
```

Advanced Real-Time Plotting:

- For more advanced features, we can use the **animation module** in the matplotlib library (**matplotlib.animation**)
- For more information and examples, see the textbook “**Python for Science**
- **and Engineering”, chapter 36: “Real-Time Simulations”**
- https://www.halvorsen.blog/documents/programming/python

Example 6

In Example 6 we have **placed the plotting inside the For Loop** for "Real-Time Plotting"

```
import matplotlib.pyplot as plt
from Automation import Automation

#Simulation Parameters
t = 0
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Initialize Plot
plt.figure(1)
plt.title('Control of 1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

```
# Simulation
for k in range(N):
   t = t + Ts
   u, e = control.Controller(yk, r, u_prev, e_prev)
   u_prev = u
   e_prev = e

   yk1 = control.Model(yk, u)
   yk = yk1
   data.append(yk1)

   # Plot the Simulation Results
   plt.figure(1)
   plt.plot(t, yk1, '-o', markersize=1, color='blue')
   plt.show()
   plt.pause(Ts)
```

How to Kill a While Loop with a Keystroke?

- Next, we will use “**while True**” instead of a For Loop to create a Program that goes “Forever”
- Some Examples will be provided

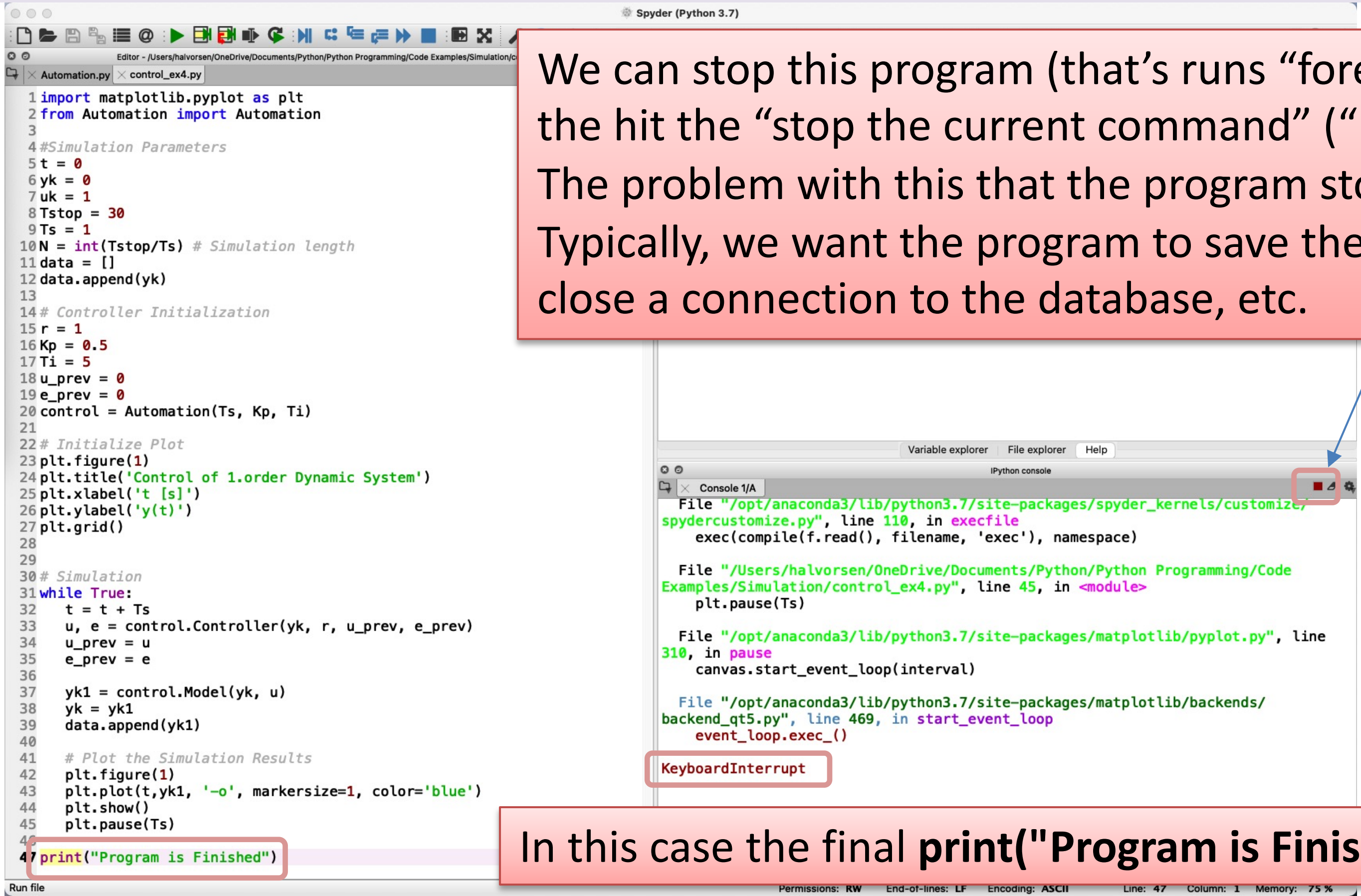
Example 7

In Example 7 we use “while True” to create a Program that goes “Forever”

```
import matplotlib.pyplot as plt
from Automation import Automation

#Simulation Parameters
t = 0
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Initialize Plot
plt.figure(1)
plt.title('Control of 1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

```
# Simulation
while True:
    t = t + Ts
    u, e = control.Controller(yk, r, u_prev, e_prev)
    u_prev = u
    e_prev = e

    yk1 = control.Model(yk, u)
    yk = yk1
    data.append(yk1)

    # Plot the Simulation Results
    plt.figure(1)
    plt.plot(t,yk1, '-o', markersize=1, color='blue')
    plt.show()
    plt.pause(Ts)

print("Program is Finished")
```

Problem with Example 7



```
1 import matplotlib.pyplot as plt
2 from Automation import Automation
3
4 #Simulation Parameters
5 t = 0
6 yk = 0
7 uk = 1
8 Tstop = 30
9 Ts = 1
10 N = int(Tstop/Ts) # Simulation length
11 data = []
12 data.append(yk)
13
14 # Controller Initialization
15 r = 1
16 Kp = 0.5
17 Ti = 5
18 u_prev = 0
19 e_prev = 0
20 control = Automation(Ts, Kp, Ti)
21
22 # Initialize Plot
23 plt.figure(1)
24 plt.title('Control of 1.order Dynamic System')
25 plt.xlabel('t [s]')
26 plt.ylabel('y(t)')
27 plt.grid()
28
29
30 # Simulation
31 while True:
32     t = t + Ts
33     u, e = control.Controller(yk, r, u_prev, e_prev)
34     u_prev = u
35     e_prev = e
36
37     yk1 = control.Model(yk, u)
38     yk = yk1
39     data.append(yk1)
40
41     # Plot the Simulation Results
42     plt.figure(1)
43     plt.plot(t,yk1, '-o', markersize=1, color='blue')
44     plt.show()
45     plt.pause(Ts)
46
47 print("Program is Finished")
```

```
  File "/opt/anaconda3/lib/python3.7/site-packages/spyder_kernels/customize/
spydercustomize.py", line 110, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "/Users/halvorsen/OneDrive/Documents/Python/Python Programming/Code
Examples/Simulation/control_ex4.py", line 45, in <module>
    plt.pause(Ts)

  File "/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py", line
310, in pause
    canvas.start_event_loop(interval)

  File "/opt/anaconda3/lib/python3.7/site-packages/matplotlib/backends/
backend_qt5.py", line 469, in start_event_loop
    event_loop.exec_()

KeyboardInterrupt
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 47 Column: 1 Memory: 75 %

We can stop this program (that's runs "forever") by use Ctrl-C or the hit the "stop the current command" ("red square") in Spyder. The problem with this that the program stops immediately. Typically, we want the program to save the lates data to a file, close a connection to the database, etc.

In this case the final **print("Program is Finished")** is not executed

Example 8

In Example 8 we use “while True” in combination with “**try .. except**”

```
import matplotlib.pyplot as plt
from Automation import Automation

print("Press Ctrl-C to Stop the Program")

#Simulation Parameters
t = 0
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Initialize Plot
plt.figure(1)
plt.title('Control of 1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

```
# Simulation
try:
    while True:
        t = t + Ts
        u, e = control.Controller(yk, r, u_prev, e_prev)
        u_prev = u
        e_prev = e

        yk1 = control.Model(yk, u)
        yk = yk1
        data.append(yk1)

        # Plot the Simulation Results
        plt.figure(1)
        plt.plot(t,yk1, '-o', markersize=1, color='blue')
        plt.show()
        plt.pause(Ts)

except KeyboardInterrupt:
    pass

print("Program is Finished")
```

Change Values during Execution

- Typically, we may want to change, e.g., Kp or Ti during execution
- At least it may be useful to change the Reference Value or the Setpoint during execution
- This can be done by reading updated values for these parameters from a database or similar (e.g., OPC, MQTT, etc.) inside the For/While Loop
  - This will not be part of this tutorial
- We will just change the Reference/Setpoint after a preset time, e.g., t=30s

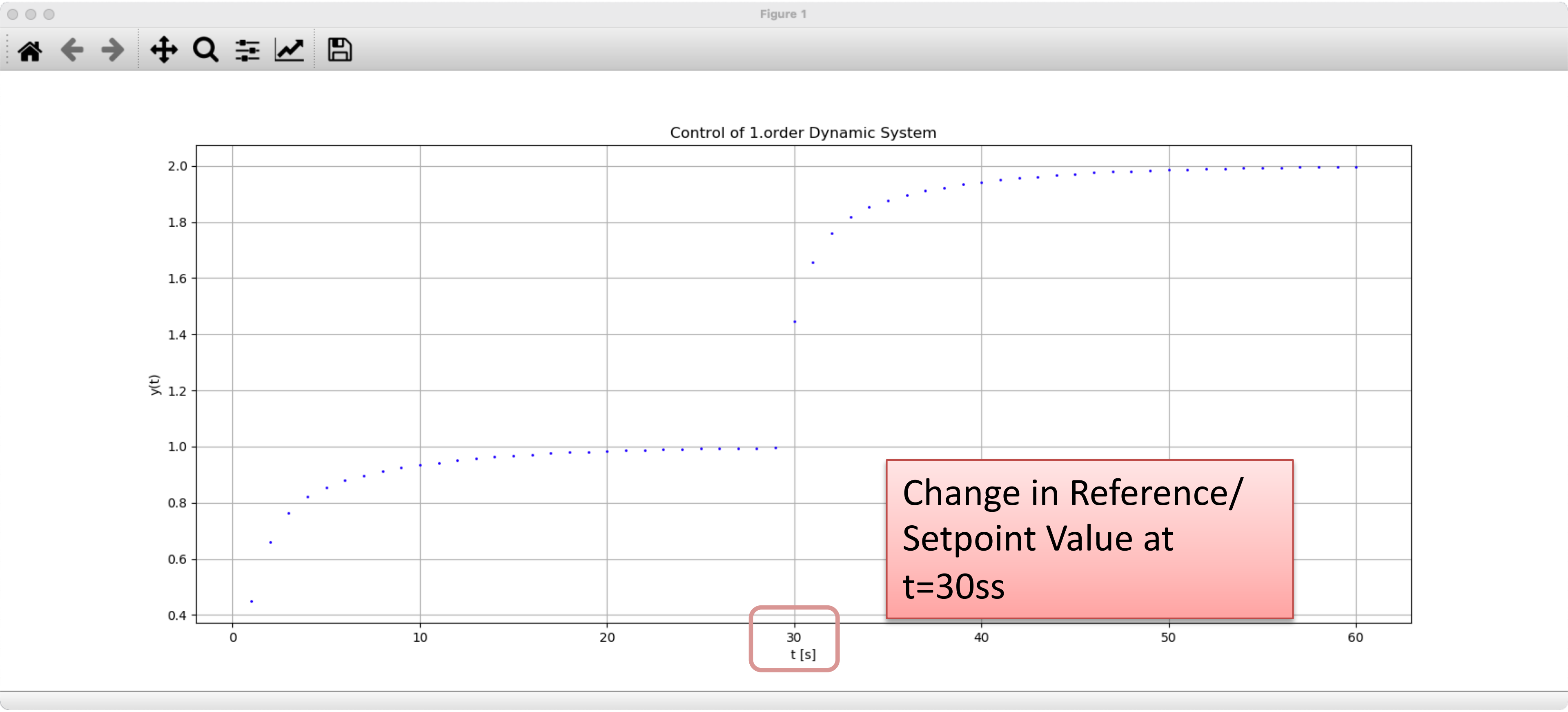
Example 9

In Example 9 we change the Reference value after a specific time

```
import matplotlib.pyplot as plt
from Automation import Automation

print("Press Ctrl-C to Stop the Program")

#Simulation Parameters
t = 0
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Initialize Plot
plt.figure(1)
plt.title('Control of 1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

```
# Simulation
try:
    while True:
        t = t + Ts
        if (t==30):
            r = 2
        u, e = control.Controller(yk, r, u_prev, e_prev)
        u_prev = u
        e_prev = e

        yk1 = control.Model(yk, u)
        yk = yk1
        data.append(yk1)

        # Plot the Simulation Results
        plt.figure(1)
        plt.plot(t,yk1, '-o', markersize=1, color='blue')
        plt.show()
        plt.pause(Ts)

except KeyboardInterrupt:
    pass

print("Program is Finished")
```

Changing Reference Value



Change in Reference/ Setpoint Value at t=30ss

Summary

- We started to Simulate a basic **Model** in Python
- Then we improved the Python Code by using **Functions** and **Classes** in Python
- We Implemented also a basic **Control System** (just simulations, not connection to a real process)
- Then, we tried to implement a “**real-time control system**”, where:
  - We updated the Plot in every iteration
  - The Control System runs “forever” until we stop the program with “Ctrl-C”
  - We changed the Reference/Setpoint value during execution, i.e., r=1, then we changed to r=2 after 30s.

  => These things are more complicated in Python compared to, e.g., LabVIEW or WinForm App in Visual Studio/C#, but it is possible.

Final Solution

```python
import matplotlib.pyplot as plt
from Automation import Automation

print("Press Ctrl-C to Stop the Program")

#Simulation Parameters
t = 0
yk = 0
uk = 1
Tstop = 30
Ts = 1
N = int(Tstop/Ts) # Simulation length
data = []
data.append(yk)

# Controller Initialization
r = 1
Kp = 0.5
Ti = 5
u_prev = 0
e_prev = 0
control = Automation(Ts, Kp, Ti)

# Initialize Plot
plt.figure(1)
plt.title('Control of 1.order Dynamic System')
plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.grid()
```

```python
# Simulation
try:
    while True:
        t = t + Ts
        if (t==30):
            r = 2
        u, e = control.Controller(yk, r, u_prev, e_prev)
        u_prev = u
        e_prev = e

        yk1 = control.Model(yk, u)
        yk = yk1
        data.append(yk1)

        # Plot the Simulation Results
        plt.figure(1)
        plt.plot(t,yk1, '-o', markersize=1, color='blue')
        plt.show()
        plt.pause(Ts)

except KeyboardInterrupt:
    pass

print("Program is Finished")
```

Final Solution

```
class Automation:
    def __init__(self, Ts, Kp, Ti):
        self.Ts = Ts
        self.Kp = Kp
        self.Ti = Ti

    def Model(self, yk, uk):
        # Model Parameters
        K = 3
        T = 4
        a = -1/T
        b = K/T

        Ts = self.Ts

        #Model Implementation
        yk1 = (1 + a*Ts) * yk + Ts*b*uk

        return yk1

    def Controller(self, y, r, u_prev, e_prev):
        Kp = self.Kp
        Ti = self.Ti
        Ts = self.Ts

        e = r - y
        u = u_prev + Kp*(e - e_prev) + (Kp/Ti)*Ts*e

        return u, e
```

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog