windUp

May 17, 2021

1 Integral saturation (Wind-up)

[1]:
```
import numpy as np
import control as ct
import matplotlib.pyplot as plt
```

1.1 Simple step response (Ignoring saturation)

[2]:
```
# Plant
num1 = [1]
den1 = [1,7,12,0]
G = ct.tf(num1, den1)

# Controller
Kp= 50.4
Ti = 0.9069
Td = 0.2267
Gc = ct.tf([Kp*Ti*Td, Kp*Ti, Kp],[Ti, 0])


# Closed-loop system
sys = ct.feedback(Gc*G)

print('Plant:', G)
```

```
Plant:
        1
------------------
s^3 + 7 s^2 + 12 s
```

1.2 Simulation parameters

[3]:
```
# Time parameters
tsim = 12
dt = 0.01

# Time and reference signal
```

```
t = np.arange(0, tsim, dt)
R = np.ones(len(t))

# Controlled system
t1, C1 = ct.forced_response(sys,t,R)
```

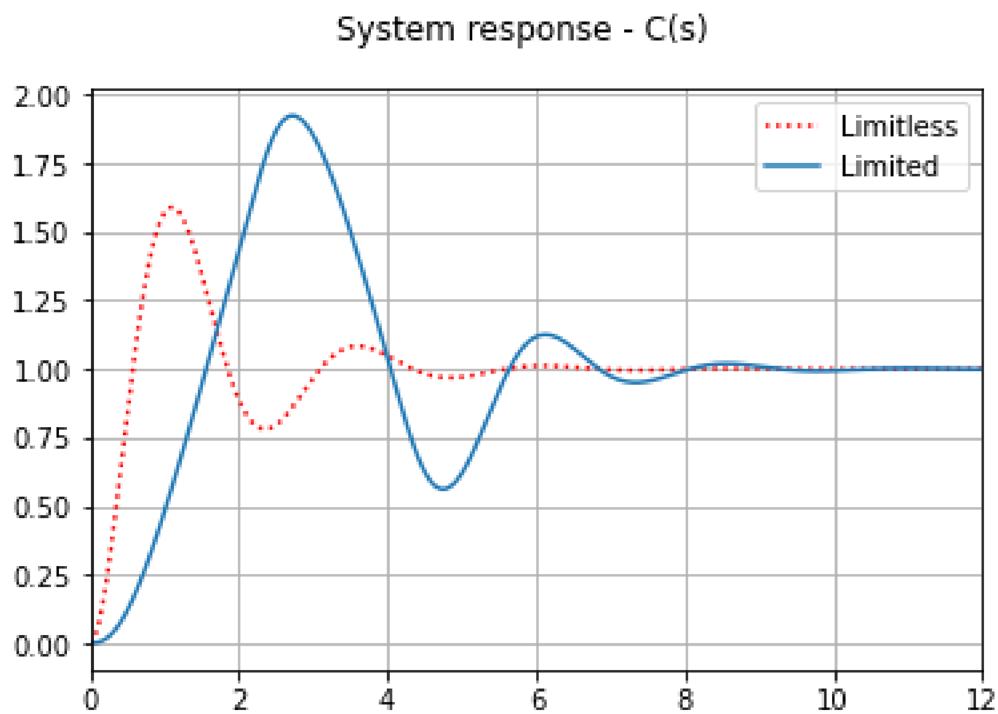
1.3 Integral Wind-up

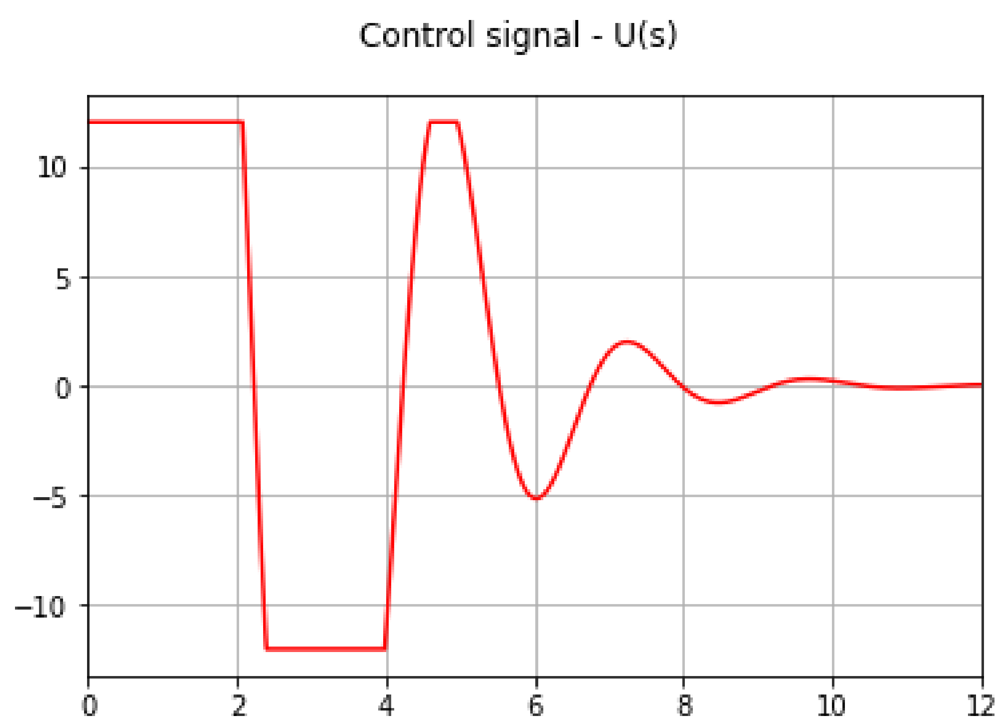
[4]:
```
# Conver to space states to allow initial conditions
Gss = ct.tf2ss(G)

# Initial conditions
xPre = np.zeros(len(G.pole()))

# Accumulated system response
C2 = np.zeros(len(t))

# Instantaneus control signal
Uacc = np.zeros(len(t))

# Accumulated integral signal
Iacc = np.zeros(len(t))

# Initialization of the integral signal and the error
I = 0
ePre = 0

# Limits of the control signal
lUp = 12
lDo = -12

for i, ti in enumerate(t):
    # Error
    e = R[i] - C2[i-1]

    # Controller
    P = Kp*e
    I = I + (Kp*e*dt)/Ti
    D = Kp*Td*(e - ePre)/dt
    U = P + I + D

    # Saturation of the control signal
    U = max(min(U, lUp), lDo)
    Uacc[i] = U

    # Plant response
```

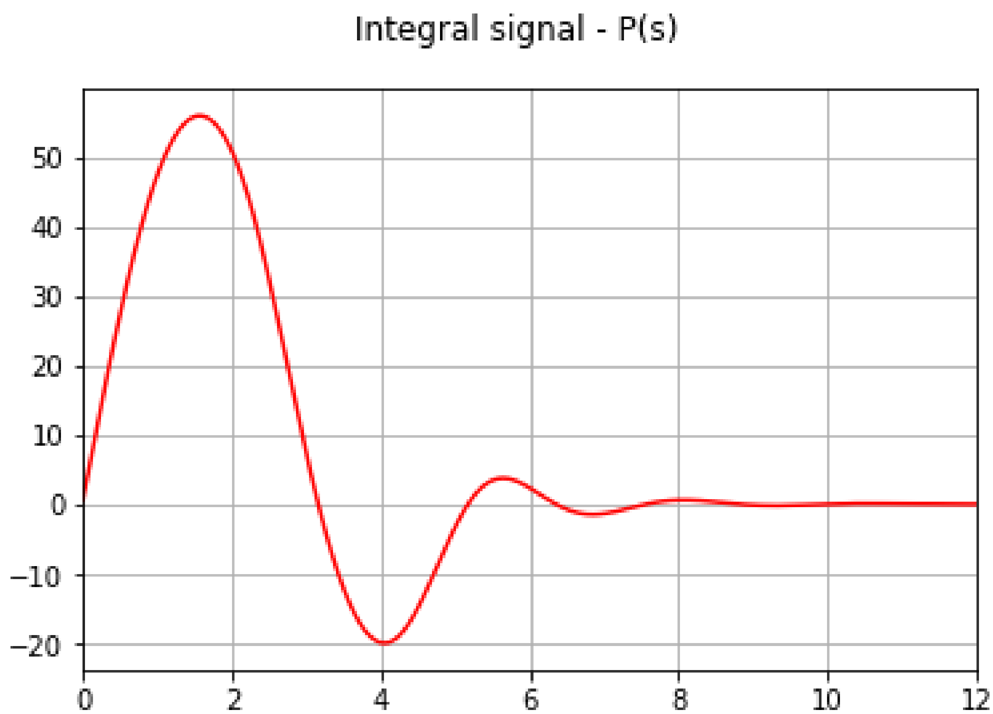
```
    _, Ci, Xi = ct.forced_response(Gss, [ti-dt,ti], [U,U], X0 = xPre, return_x␣
↪= True)

    # Save results
    C2[i] = np.squeeze(Ci[-1])
    xPre = np.squeeze(Xi[:,-1])
    ePre = e
    Iacc[i] = I
```

[5]:

```
# Compare
plt.plot(t1, C1, 'r:', label = "Limitless" )
plt.plot(t, C2, label = "Limited ")
plt.xlim((0,tsim))
plt.suptitle("System response - C(s)")
plt.legend()
plt.grid()

#Controller
plt.figure()
plt.plot(t, Uacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Control signal - U(s)")
plt.grid()

# Integral part
plt.figure()
plt.plot(t, Iacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Integral signal - P(s)")
plt.grid()
```


System response - C(s)
Limitless
Limited
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
12


Control signal - U(s)
10
5
0
−5
−10
0
2
4
6
8
10
12



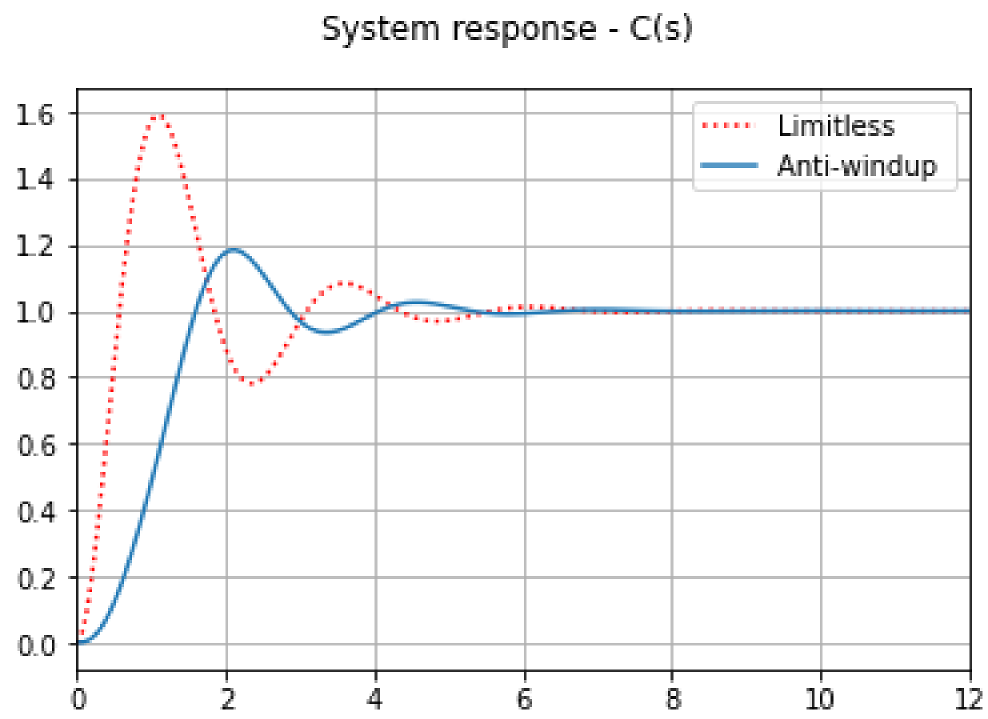
1.4 Conditional Anti-windup

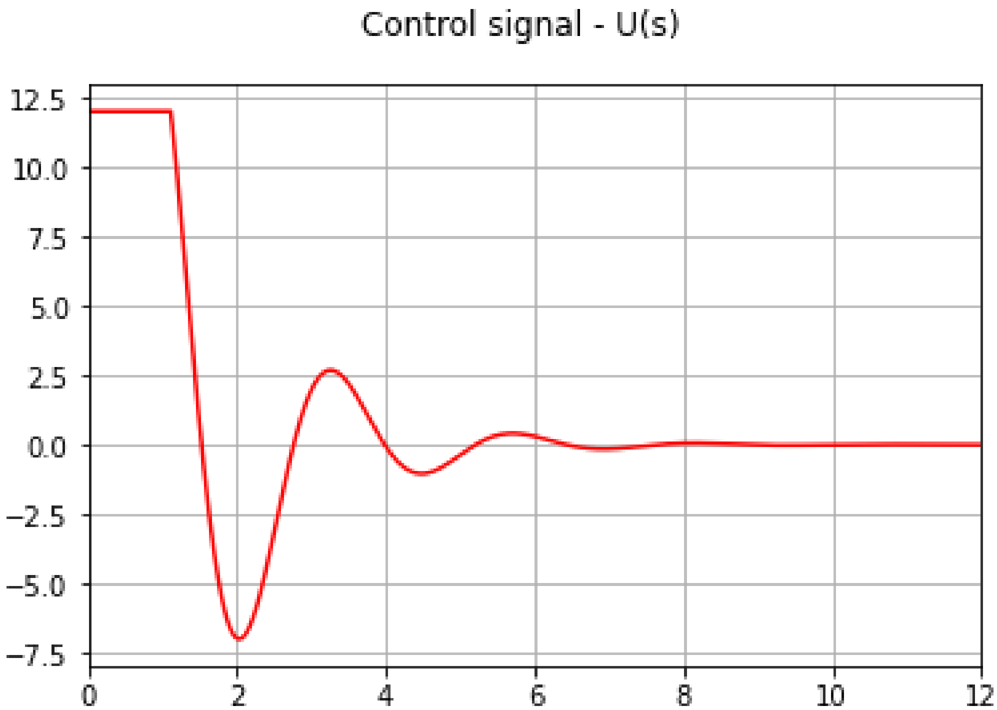
[6]:
```
# Conver to space states to allow initial conditions
Gss = ct.tf2ss(G)

# Initial conditions
xPre = np.zeros(len(G.pole()))

# Accumulated system response
C2 = np.zeros(len(t))

# Accumulated control signal
Uacc = np.zeros(len(t))

# Accumulated integral signal
Iacc = np.zeros(len(t))

# Initialization of the integral signal and the error
I = 0
ePre = 0

# Limits of the control signal
lUp = 12
```

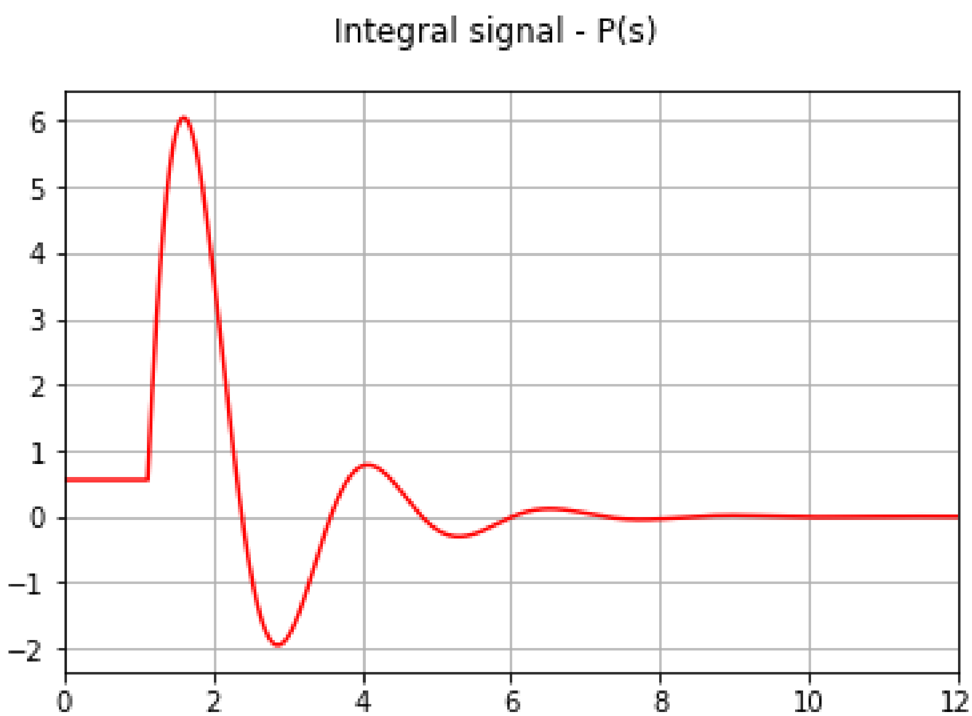
```
lDo = -12

for i, ti in enumerate(t):
    # Error
    e = R[i] - C2[i-1]

    # Controller
    P = Kp*e

    # Conditional integration
    if Uacc[i-1] > lDo and Uacc[i-1] < lUp:
        I = I + (Kp*e*dt)/Ti

    D = Kp*Td*(e - ePre)/dt
    U = P + I + D

    # Saturation of the control signal
    U = max(min(U, lUp), lDo)
    Uacc[i] = U

    # Plant response
    _, Ci, Xi = ct.forced_response(Gss, [ti-dt,ti], [U,U], X0 = xPre, return_x␣
 ↪= True)

    # Save results
    C2[i] = np.squeeze(Ci[-1])
    xPre = np.squeeze(Xi[:,-1])
    ePre = e
    Iacc[i] = I
```

[7]:
```
# Compare
plt.plot(t1, C1, 'r:', label = "Limitless" )
plt.plot(t, C2, label = "Anti-windup ")
plt.xlim((0,tsim))
plt.suptitle("System response - C(s)")
plt.legend()
plt.grid()

#Controller
plt.figure()
plt.plot(t, Uacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Control signal - U(s)")
plt.grid()

# Integral part
plt.figure()
```

```
plt.plot(t, Iacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Integral signal - P(s)")
plt.suptitle("Integral signal - P(s)")
plt.grid()
```




Control signal - U(s)
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0
2
4
6
8
10
12


Integral signal - P(s)
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10
12

1.5 Integral discharge Anti-windup

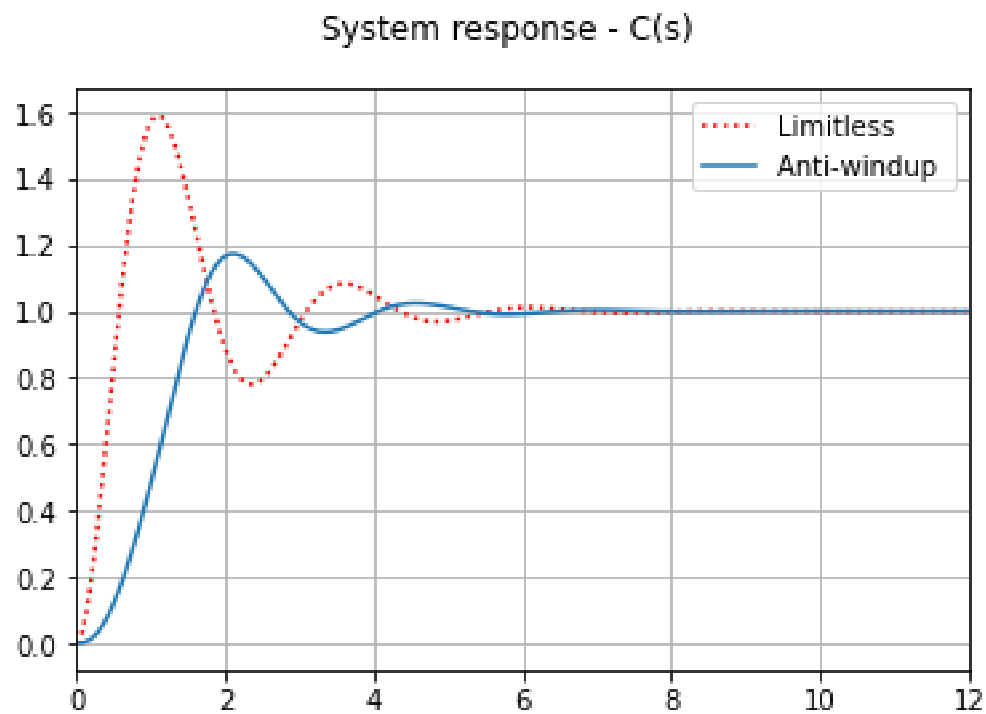
[8]:

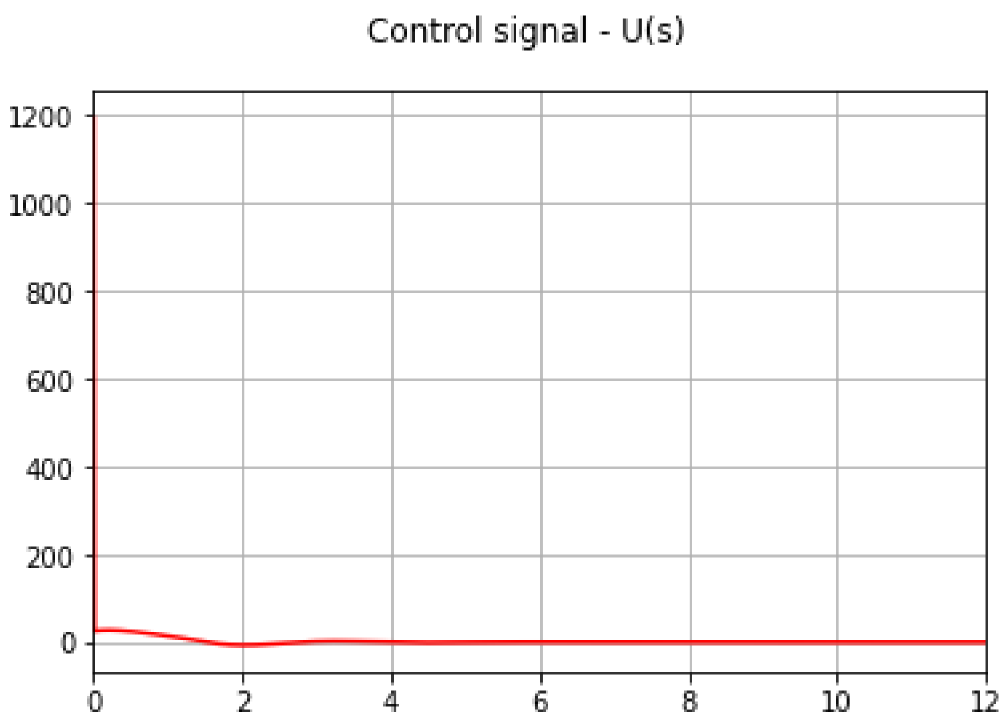
```
# Conver to space states to allow initial conditions
Gss = ct.tf2ss(G)

# Initial conditions
xPre = np.zeros(len(G.pole()))

# Accumulated system response
C2 = np.zeros(len(t))

# Accumulated control signal
Uacc = np.zeros(len(t))

# Accumulated integral signal
Iacc = np.zeros(len(t))

# Initialization of the integral signal and the error
I = 0
ePre = 0

# Limits of the control signal
lUp = 12
lDo = -12

# Discharge time constant
Tt = np.sqrt(Ti*Td)

for i, ti in enumerate(t):
    # Error
    e = R[i] - C2[i-1]

    # Controller
    P = Kp*e

    # Discharged integration
    es = U - Uacc[i-1]
    I = I + dt*(Kp*e/Ti + es/Tt)

    D = Kp*Td*(e - ePre)/dt
    U = P + I + D

    # Saturation of the control signal
    Uacc[i] = U
    U = max(min(U, lUp), lDo)
```

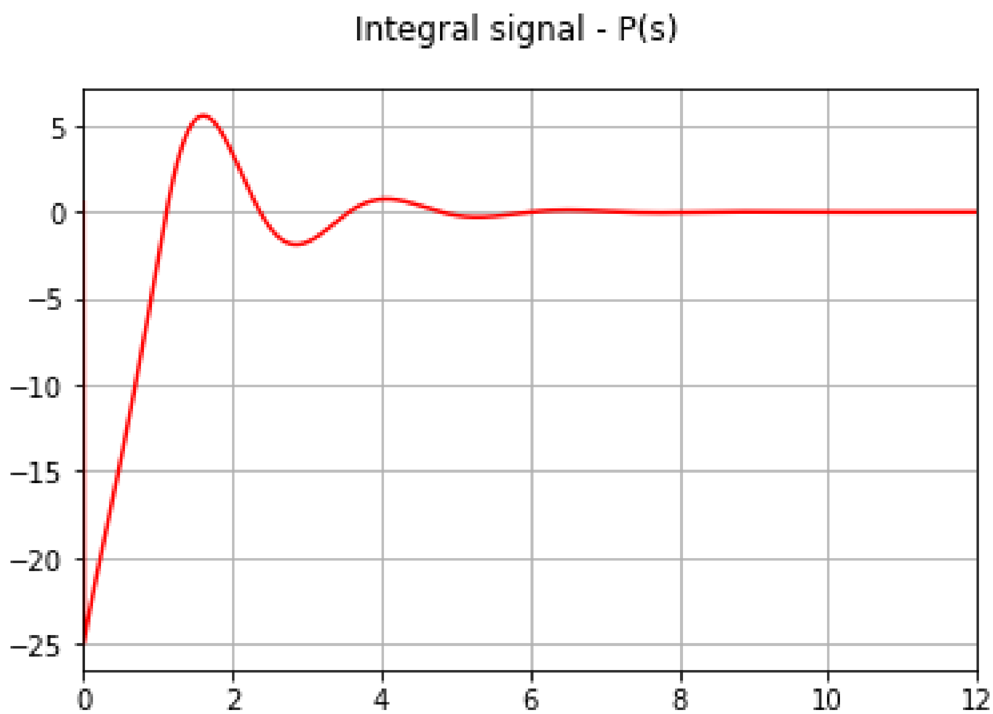
```
    # Plant response
    _, Ci, Xi = ct.forced_response(Gss, [ti-dt,ti], [U,U], X0 = xPre, return_x␣
↪= True)

    # Save results
    C2[i] = np.squeeze(Ci[-1])
    xPre = np.squeeze(Xi[:,-1])
    ePre = e
    Iacc[i] = I
```

[9]:

```
# Compare
plt.plot(t1, C1, 'r:', label = "Limitless" )
plt.plot(t, C2, label = "Anti-windup ")
plt.xlim((0,tsim))
plt.suptitle("System response - C(s)")
plt.legend()
plt.grid()

#Controller
plt.figure()
plt.plot(t, Uacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Control signal - U(s)")
plt.grid()

# Integral part
plt.figure()
plt.plot(t, Iacc, 'r');
plt.xlim((0, tsim))
plt.suptitle("Integral signal - P(s)")
plt.suptitle("Integral signal - P(s)")
plt.grid()
```


System response - C(s)
Limitless
Anti-windup
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12


Control signal - U(s)
1200
1000
800
600
400
200
0
0
2
4
6
8
10
12


Integral signal - P(s)
5
0
−5
−10
−15
−20
−25
0
2
4
6
8
10
12

```
[ ]:
```