SIMULAZIONI DI SISTEMI IN PYTHON (v 2.7)

(Prof. Fischetti Pietro)

Librerie in Python per il calcolo scientifico e la visualizzazione grafica:

Numpy – libreria per il calcolo numerico (https://numpy.org)
Scipy – libreria per il calcolo scientifico (http://www.scipy.org)
Matplotlib – libreria grafica (https://matplotlib.org)
Simpy – Libreria per il calcolo simbolico (https://simpy.readthedocs.io/en/latest)
Control – Libreria per simulazione sistemi (https://python-control.readthedocs.io/en/0.8.1/index.html)

VETTORI

#creazione di un vettore: import numpy as np x=np.array([1,2,3,4,5]) print x	[1 2 3 4 5]
#somma gli elementi di un vettore import numpy as np print np.sum([0.5, 1.5])	2.0
#esempio crea vettore di numeri pari da 0 a 10: print np.arange(start=0, stop=12, step=2) # print np.arange(0,12,2)	[0 2 4 6 8 10]
#stampa il numero di elementi di un vettore: x=np.array([1,2,0,2,8]) print np.size(x)	5
#stampa il min e il massimo di un vettore: x=np.array([1,-2,5,3,1]) print x,x.min(),x.max()	[1 -2 5 3 1] -2 5
#Trasposta di un vettore: x=np.array([1,2,3,4,5]) print np.array([x]).T	[[1] [2] [3] [4] [5]]
#somma di vettori: x=np.array([1,2,3,4,5]) print x+x	[2 4 6 8 10]
# moltiplicazione vettore * trasposta di un vettore: print np.dot(x,xT)	[55]
#eleva al quadrato gli elementi di un vettore x=np.arange(start=0, stop=12, step=2) y=x*2 print y	[0 4 8 12 16 20]

#calcola il prodotto di 2 vettori elemento per #elemento
A = np.matrix(x)
B = np.matrix(y)
print A,B
print np.multiply(A,B).sum(axis=0)
	[[0 2 4 6 8 10]] [[0 4 8 12 16 20]]
	[[0 8 32 72 128 200]]

#elevazione a potenza di un vettore A=np.array([1,2,3,4,5]) print np.power(A,2)	[1 4 9 16 25]

#Creazione di una matrice come seq. Di vettori B=[1,2,3] C=[4,5,6] D=[7,8,9] E=np.array([B,C,D]) print E	[[1 2 3] [4 5 6] [7 8 9]]
#Estrae un sottovettore da un vettore A=np.array([0,2,4,6,8,10]) print A[2:5]	[4 6 8]
#Crea un vettore dei quadrati dei numeri da 0 a 4: print np.array([i**2 for i in range(5)])	[0 1 4 9 16]
Creazione di sequenze numeriche np.linspace (start , stop , num = 50 , endpoint = True) Restituisce una sequenza di numeri equidistanti su un intervallo specificato [(start-end)/num che corrisponde all'intervallo fra campioni]	print np.linspace(0, 10, num=8, endpoint=True) [0. 1.42 2.85 4.28 5.71 7.14 8.57 10.] print np.linspace(0, 10, num=8, endpoint=False) [0. 1.25 2.5 3.75 5. 6.25 7.5 8.75]

MATRICI

$\begin{pmatrix} 3 & 5 & 1 \\ 0 & 4 & 7 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 4 & 7 \\ 1 & 0 & 0 \\ 2 & 6 & 9 \end{pmatrix} = \begin{pmatrix} 10 & 18 & 30 \\ 18 & 42 & 63 \\ 2 & 6 & 9 \end{pmatrix}$	A=np.array([[3,5,1],[0,4,7],[0,0,1]]) B=np.array([[1,4,7],[1,0,0],[2,6,9]]) print np.matmul(A,B)
$\begin{pmatrix} 3 & 5 & 1 \\ 0 & 4 & 7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 4 \\ 1 & 0 \\ 2 & 6 \end{pmatrix} = \begin{pmatrix} 10 & 18 \\ 18 & 42 \end{pmatrix}$	A=np.array([[3,5,1],[0,4,7]]) B=np.array([[1,4],[1,0],[2,6]]) print np.matmul(A,B)
$(3 \quad 5 \quad 1) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 5$	A=np.array([[3,5,1]]) B=np.array([[0,1,0]]) B=B.T print np.matmul(A,B)
$\begin{pmatrix} -1 \\ 4 \\ 1 \end{pmatrix} \cdot (2 \quad 4) = \begin{pmatrix} -2 & -4 \\ 8 & 16 \\ 2 & 4 \end{pmatrix}$	A=np.array([[-1,4,1]]) A=A.T B=np.array([[2,4]]) print np.matmul(A,B)
$(3 \quad 5 \quad 1) \cdot \begin{pmatrix} 0 & 4 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \quad (5 \quad 13)$	A=np.array([[3,5,1]]) B=np.array([[0,4],[1,0],[0,1]]) print np.matmul(A,B)
$\begin{pmatrix} 9 & 4 & 5 \\ 12 & 2 & -3 \\ 4 & 0 & 0 \\ 0 & 0 & 9 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \cdot = \begin{pmatrix} 13 \\ 14 \\ 4 \\ 0 \end{pmatrix}$	A=np.array([[9,4,5],[12,2,-3],[4,0,0],[0,0,9]]) B=np.array([[1,1,0]]) B=B.T print np.matmul(A,B)

#calcolo della matrice inversa:
import numpy as np
`A = np.array([[1,3,5],[2,5,1],[2,3,8]])`
`print np.linalg.inv(A)`
	$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix},$
	
	$\mathbf{A}^{-1} = \frac{1}{-25} \begin{bmatrix} 37 & -9 & -22 \\ -14 & -2 & 9 \\ -4 & 3 & -1 \end{bmatrix} = \begin{bmatrix} -1.48 & 0.36 & 0.88 \\ 0.56 & 0.08 & -0.36 \\ 0.16 & -0.12 & 0.04 \end{bmatrix}$

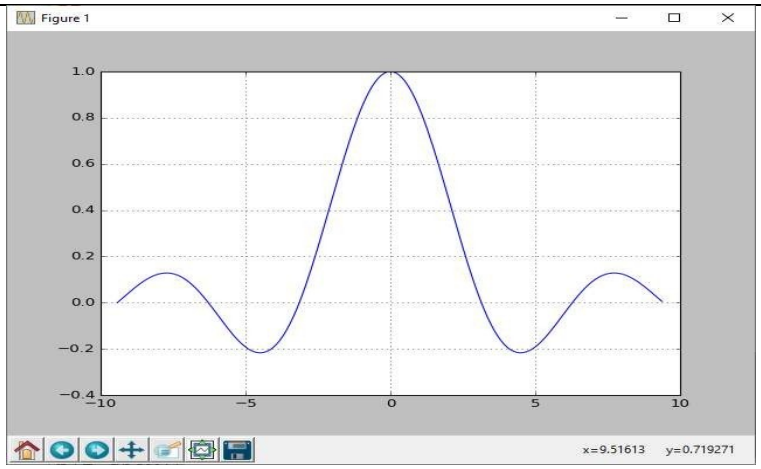
#calcolo della trasposta:
import numpy as np
`A = np.array([[1,3,5],[2,5,1],[2,3,8]])`
`print A.transpose()`
`#Oppure:`
`print A.T`
	[[1 2 2]
	[3 5 3]
	[5 1 8]]

GRAFICI

Per visualizzare i grafici sia in 2D che in 3D si utilizzera' la libreria *matplotlib*.

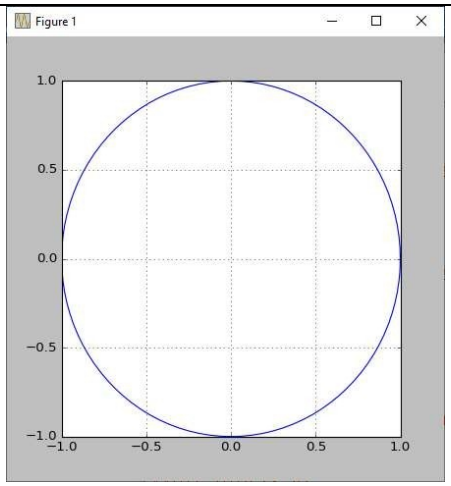
#Grafico di funzione

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

x = np.arange(-3*np.pi,3*np.pi,0.1)
y = np.sin(x)/x
plt.grid(True)
plt.plot(x,y)
plt.show()
```



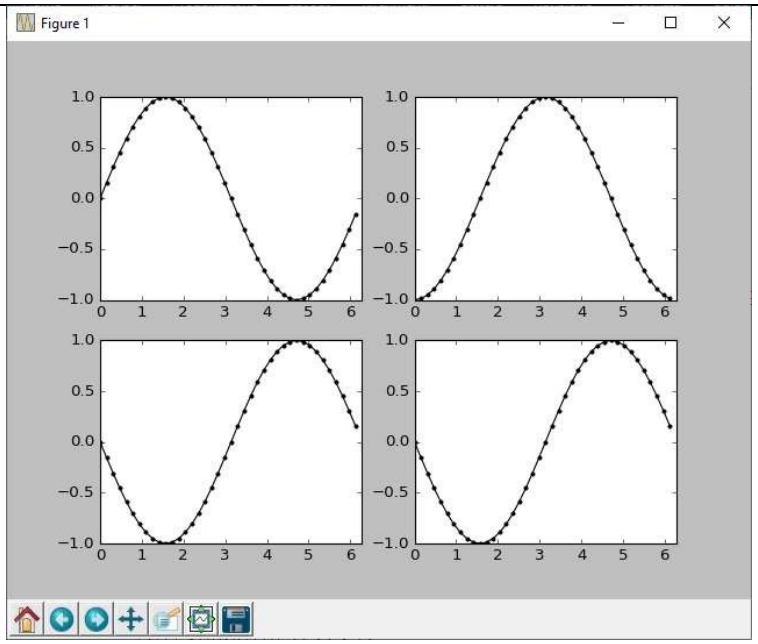
#Visualizzazione di un cerchio

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

a = np.arange(0,2*np.pi,2*np.pi/100)
x = np.sin(a)
y = np.cos(a)
plt.grid(True)
plt.plot(x,y)
plt.show()
```

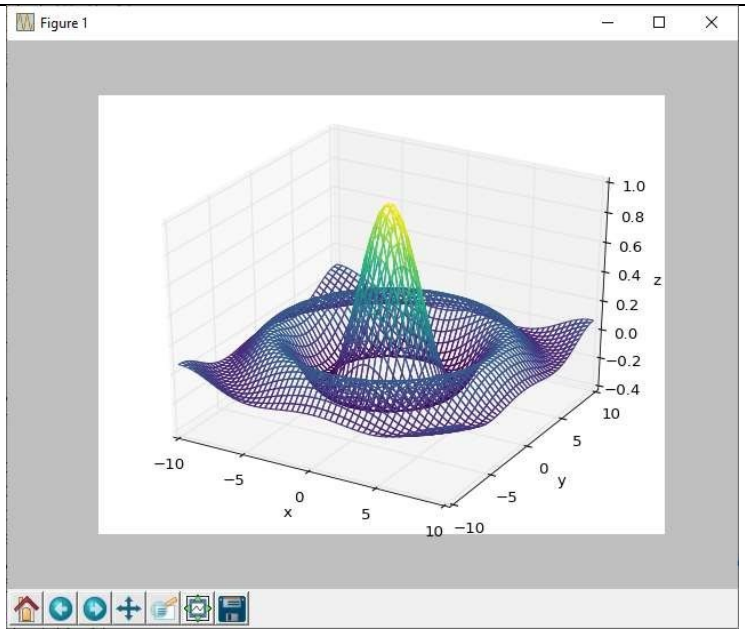


#Grafico di funzioni 2D

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

X = np.arange(0,2*np.pi,np.pi/20)
Y1=np.sin(X)
plt.subplot(2,2,1)
plt.plot(X,Y1,'k.-')
plt.axis([0,2*np.pi,-1,1])
Y2=np.sin(X-np.pi/2)
plt.subplot(2,2,2)
plt.plot(X,Y2,'k.-')
plt.axis([0,2*np.pi,-1,1])
Y3=np.sin(X-np.pi)
plt.subplot(2,2,3)
plt.plot(X,Y3,'k.-')
plt.axis([0,2*np.pi,-1,1])
Y4=np.sin(X+np.pi)
plt.subplot(2,2,4)
plt.plot(X,Y4,'k.-')
plt.axis([0,2*np.pi,-1,1])
```

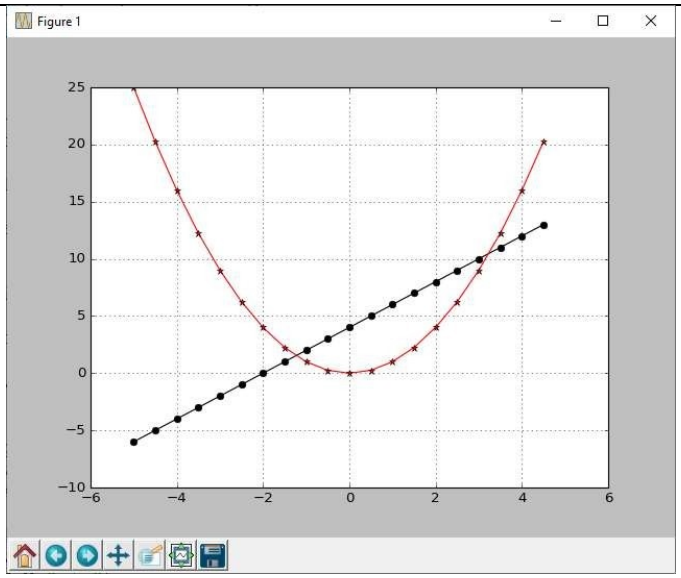


plt.show()	

#Grafico di funzioni 3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import proj3d
 	
def my_function(x, y):
  r=np.sqrt(x**2+y**2)
  return np.sin(r)/r
 	
x = np.linspace(-10,10,50)
y = x
X, Y = np.meshgrid(x, y)
Z = my_function(X, Y)
 	
norm = plt.Normalize(Z.min(), Z.max())
colors = cm.viridis(norm(Z))
ax = plt.axes(projection="3d")
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,facecolors=colors, shade=False)
surf.set_facecolor((0,0,0,0))
 	
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
	

APPLICAZIONI MATEMATICHE

#calcolo cubo di un polinomio import numpy as np C=np.convolve([1,1],[1,1]) C=np.convolve(C,[1,1]) print C	[1 3 3 1]
#calcolo radici di un polinomio import numpy as np C=[1, 3, 3, 1] R=np.roots(C) np.set_printoptions(precision=3) np.set_printoptions(suppress=True) print R	[-1.+0.j -1.-0.j -1.+0.j]
#dalle radici al polinomio import numpy as np R=[-1,-1,-1] print np.poly(R)	[-1.+0.j -1.-0.j -1.+0.j]
#calcolo rapporto import numpy as np B=[1,3,3,1] A=[1,2,1]	[[1.0, 1.0], [0.0]]

#deconvolve C= np.polydiv(B,A) C=[t.tolist() for t in np.polydiv(B,A)] print C	
#calcolo dei residui import numpy as np import scipy.signal as sps N=[1,1] D=np.convolve([1,2],[1,3]) print D #stampa residui,poli,coeff print sps.residue(N, D)	[1 5 6] (array([2., -1.]), array([-3., -2.]), array([0.]))
#Grafico di funzioni matematiche import matplotlib.pyplot as plt import numpy as np P1=[2,4] X=np.arange(-5,5,0.5) Y=np.polyval(P1,X) plt.plot(X,Y,"ko-") P2=[1,0,0] Y=np.polyval(P2,X) plt.plot(X,Y,"r*-") plt.grid(True) plt.show()	
#Risoluzione sistema di equazioni import numpy as np A=np.array([1,-1,1,1]) A=np.reshape(A, (2, 2)) B=np.array([2,6]) invA=np.linalg.inv(A) X=np.matmul(invA,B.T) print X	[4. 2.]

CALCOLO TRASFORMATA DI LAPLACE

import sympy t, s = sympy.symbols('t, s') f = sympy.exp(-2*t) F = sympy.laplace_transform(f, t, s) print F	1/(s + 2)

CALCOLO ANTITRASFORMATE

#Calcolo residui import numpy as np import scipy.signal as sps N=np.convolve([1,1],[1,1]) D=np.convolve([1,2],[1,4]) D=np.convolve(D,[1,0]) (r,p,k)=sps.residue(N,D) print r,p	[1.125 -0.25 0.125] [-4. -2. 0.]

#Antitrasformata di Laplace

```
import numpy as np
import sympy as sym
s, t = sym.symbols('s, t')
F=(s**2+2*s+1)/(s**3+6*s**2+8*s)
print F
print sym.inverse_laplace_transform(F, s, t)
```

```
(exp(4*t) - 2*exp(2*t) + 9)*exp(-4*t)*1/8
```

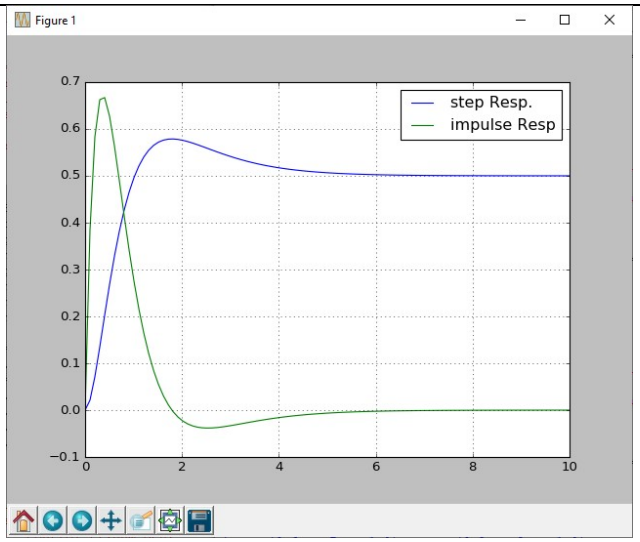
#Calcolo Antitrasformata:
G(s)=(s+5)/((s+2)*(s+8))

```
import sympy
s, t = sympy.symbols('s, t')
F=(s+5)/((s+2)*(s+8))
A=F.apart(s)
for i in A.args:
   print sympy.inverse_laplace_transform(i, s, t)
```

```
exp(-2*t)/ 2 exp(-8*t)/2
```

#Calcolo Residui, Antitrasformata,
#risposta al gradino e all'impulso:
#G(s)=(5*s+3)/((s+1)*(s+2)*(s+3))

```
s, t = sympy.symbols('s, t')
G=(5*s+3)/((s+1)*(s+2)*(s+3))
print "Laplace Antitrasform:"
A=G.apart(s)
for i in A.args:
   print sympy.inverse_laplace_transform(i, s, t),
print '\n\n'

s = control.tf('s')
G=(5*s+3)/((s+1)*(s+2)*(s+3))

ss=control.tfdata(G)
N=np.array(ss[0])[0][0]
D=np.array(ss[1])[0][0]
print "Residui:\n",signal.residue(N, D)


t = np.linspace(0, 10, 100)
_,y=control.step_response(G,T=t)
plt.plot(t,y,label="step Resp.")
_,y=control.impulse_response(G,T=t)
plt.plot(t,y,label="impulse Resp")
plt.legend(loc='best')

plt.grid()
plt.show()

>>>
(array([-6.,  7., -1.]),
array([-3., -2., -1.]), array([0.]))
>>>
Laplace Antitrasform:
-exp(-t)*Heaviside(t) -6*exp(-3*t)*Heaviside(t)
7*exp(-2*t)*Heaviside(t)


Residui:
```



(array([-6., 7., -1.]), array([-3., -2., -1.]), array([0.]))	

SCOMPOSIZIONE DI UN POLINOMIO IN FATTORI

G(s)=(s+5)/(S^3+3s^2+3s+1) from sympy import Symbol,factor s=Symbol('s') print factor(s**3 + 3*s**2 + 3*s + 1)	(s + 1)**3
G(s)=(10s^3+30s^2+20s)/(s^3+5s^2-2s-24) print factor(s**3 + 5*s**2 - 2*s - 24)	(s - 2)*(s + 3)*(s + 4)

La Libreria control

La funzione control.tf() viene utilizzata per creare funzioni di trasferimento con la seguente sintassi:
H = control.tf(num, den)
dove H è la funzione di trasferimento risultante num (che rappresenta il numeratore) e
den (che rappresentano il denominatore) sono matrici in cui gli elementi sono i coefficienti del
s-polinomi in ordine decrescente da sinistra a destra.
Naturalmente, puoi usare qualsiasi nome diverso da H, num e den nei tuoi programmi.
Per illustrare la sintassi, supponiamo che la funzione di trasferimento sia

$$H(s) = \frac{b_1 \cdot s + b_0}{a_1 \cdot s + a_0}$$

In questo caso,
num = np.array([b1, b0])
e
den = np.array([a1, a0])
Esempio:

$$H(s) = \frac{2}{5 \cdot s + 1}$$

Viene tradotto in Python come:

```
import numpy as np
import control

# %% Creating the transfer function:

num = np.array([2])
den = np.array([5, 1])
H = control.tf(num, den)

# %% Displaying the transfer function:

print('H(s) =', H)
```

Esempio:

$$H(s) = \frac{3 \cdot (2 \cdot s + 1)}{(3 \cdot s + 1) \cdot (5 \cdot s + 1)}$$

In Python:

```
num1 = np.array([3])
num2 = np.array([2, 1])
num = np.convolve(num1, num2)
den1 = np.array([3, 1])
den2 = np.array([5, 1])
den = np.convolve(den1, den2)

#oppure:
num = np.polymul( np.array([3]),np.array([2, 1]))
den = np.polymul( np.array([3, 1]),np.array([5, 1]))
```

H = control.tf(num, den)
print('H(s) =', H)

La Funzione di Trasferimento

Un sistema lineare generico sara' descritto da un'equazione differenziale del tipo:

$$a_0 \frac{d^n}{dt^n} y(t) + a_1 \frac{d^{n-1}}{dt^{n-1}} y(t) + \cdots + a_{n-1} \frac{d}{dt} y(t) + a_n y = b_0 \frac{d^m}{dt^m} x(t) + b_1 \frac{d^{m-1}}{dt^{m-1}} x(t) + \cdots + a_{m-1} \frac{d}{dt} x(t) + b_m x$$

$$(n \geq m)$$

Dove y e' l'uscita e x l'ingresso. La funzione di trasferimento di questo sistema e' il rapporto della trasformata di Laplace dell'uscita e della trasformata di Laplace dell'ingresso. Assumendo che tutte le condizioni iniziali nulle, cioe':

Funzione di Trasferimento=

G(s)=$\frac{\Gamma[\text{Uscita}]}{\Gamma[\text{Ingresso}]}\{Condizioni\ iniziali\ Nulle\} = \frac{Y(s)}{X(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1} s + b_m}{a_0 s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n}$

Che si scrive anche come: $G(s) = \frac{N(s)}{D(s)}$ ccioe' come rapporto di polinomi in s. Le soluzioni di N(s)=0 si chiamano zeri e si indicano con z1,…zm mentre le soluzioni del polinomio D(s) (il cui grado indica anche l'ordine della funzione di trasferimento) vengono chiamati i poli della fdt. I poli e gli zeri possono ovviamente essere omplessi e quindi coniugati. Analizzando zeri e poli si possono valutare le principali caratteristiche di un sistema, come ad esempio la stabilità.

Forma Canonica della funzione di trasferimento

Si scompongono in fattori il numeratore e il denominatore in una forma che evidenzia gli zeri e i poli_

$$G(s) = \frac{b_m (s - z_1)(s - z_2) \dots (s - z_m)}{a_n (s - p_1)(s - p_2) \dots (s - p_n)}$$

Dove zi e pi sono gli zeri e i poli.

Si raccoglie:

$$G(s) = \frac{b_m (-z_1)(-z_2) \dots (-z_m)}{a_n (-p_1)(-p_2) \dots (-p_n)} \cdot \frac{\left(1 - \frac{s}{z_1}\right)\left(1 - \frac{s}{z_2}\right) \dots \left(1 - \frac{s}{z_m}\right)}{\left(1 - \frac{s}{p_1}\right)\left(1 - \frac{s}{p_2}\right) \dots \left(1 - \frac{s}{p_n}\right)}$$

Il fattore costante, che può essere positivo che negativo, prende il nome di guadagno statico (costante di Bode o fattore di trasferimento).

$$K = \frac{b_m (-z_1)(-z_2) \dots (-z_m)}{a_n (-p_1)(-p_2) \dots (-p_n)}$$

Cosi la fdt diventa:

$$G(s) = K \cdot \frac{\left(1 - \frac{s}{z_1}\right)\left(1 - \frac{s}{z_2}\right) \dots \left(1 - \frac{s}{z_m}\right)}{\left(1 - \frac{s}{p_1}\right)\left(1 - \frac{s}{p_2}\right) \dots \left(1 - \frac{s}{p_n}\right)}$$

In termini delle costanti di tempo del sistema cioe':

$$\tau_1 = -\frac{1}{z_1};\quad \tau_2 = -\frac{1}{z_2};\ \dots\ \tau_m = -\frac{1}{z_m}$$

$$T_1 = -\frac{1}{p_1};\quad T_2 = -\frac{1}{p_2};\quad \dots\ T_n = -\frac{1}{p_n}$$

Dove $\tau_1 \dots \tau_m$; $T_1 \dots T_n$ costanti di tempo.

Si ottiene:

$$G(s) = K \cdot \frac{(1 + s\tau_1)(1 + s\tau_2) \dots (1 + s\tau_m)}{(1 + sT_1)(1 + sT_2) \dots (1 + sT_n)}$$

Il Guadagno Statico K rappresenta il valore assunto dalla G(s) per s = 0 , ovvero per segnali d'ingresso costanti.

Se G(s) ha poli nell'origine si ha K = ∞ .

Se invece, per G(s) con zeri nell'origine si ha K = 0.

~~Se assumiamo che l'ingresso equindi per linearita' l'uscita siano costanti x_0 e y_0 avremo $a_n y_0 = b_m x_0$ cioe'~~

$$\sout{G(0) = G_0 = \frac{y_0}{x_0} = \frac{b_m}{a_n}}$$

~~Che viene chiamatoguadagno statico o guadagno a frequenza zero.~~

Dato che la fdt e' un rapporto di polinomi di numeri complessi, si ha nel caso di risposta in frequemza (cioe' s=jw)

Modulo in Db:

$$GdB(\omega) = 20Log|K| + 20Log|1 + j\omega\tau_1| + \cdots + 20Log|1 + j\omega\tau_m| - 20Log|1 + j\omega T_1| - \cdots - 20Log|1 + j\omega T_n|$$

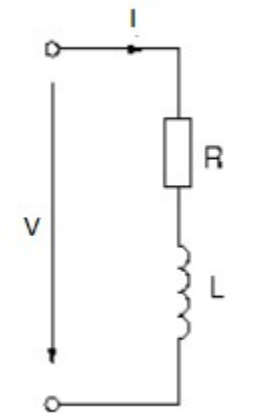
e fase:

$$\Phi = \angle G(j\omega) = arctg\frac{0}{K} + arctg(\omega\tau_1) + \cdots + arctg(\omega\tau_m) - arctg(\omega T_1) - \cdots - arctg(\omega T_n)$$

Dove:

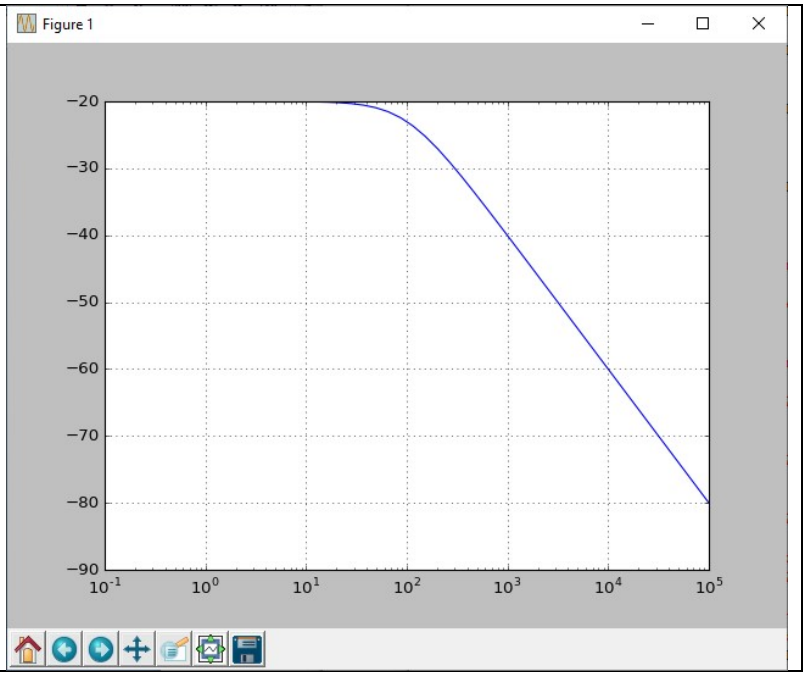
$$arctg\frac{0}{K} = \begin{cases} 0, & se\ K > 0 \\ 180^0, & se\ K < 0 \end{cases}$$

Esempio



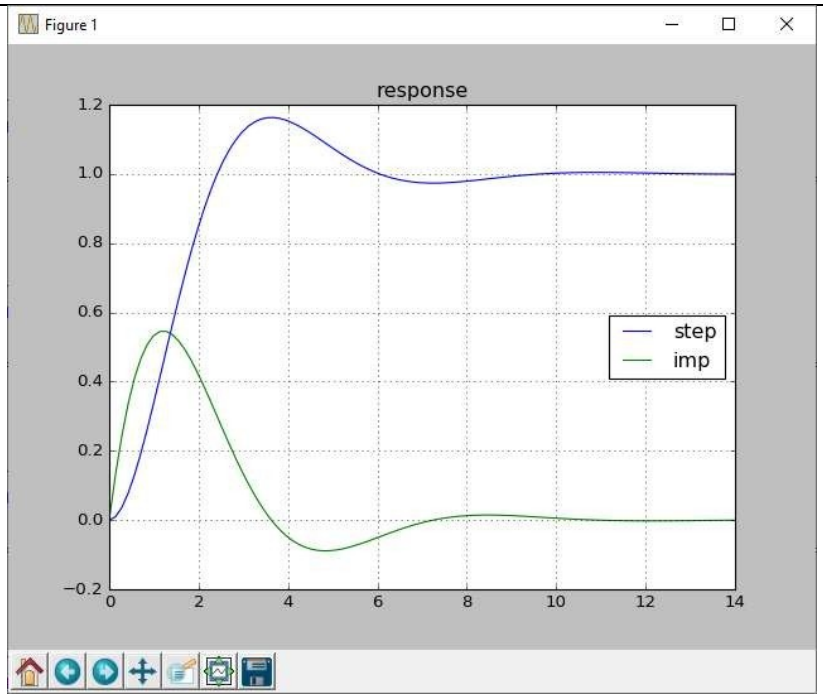
$$G(s) = \frac{1}{R}\frac{1}{(1+s\tau)} \qquad \tau = \frac{L}{R} \qquad G_0 = \frac{1}{R}$$

```
def G(w,tau,G0):
   G = G0/(1+tau*1j*w)
   return G
w = np.logspace(-1,5)
R=10.
L=0.1
tau=L/R
G0=1./R
GDb=20*np.log10(abs(G(w,tau,G0)))
plt.plot(w, GDb)
plt.xscale('log')
plt.grid()
plt.show()
```



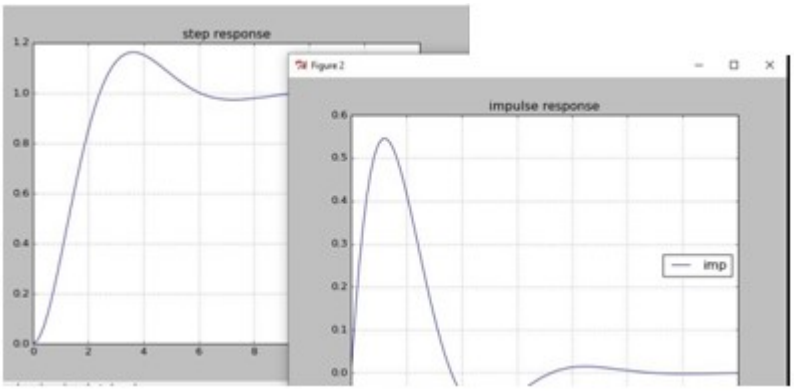
#Visualizza la risposta del #sistema:1/(s^2 + s + 1) #al gradino e all'impulso

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

sys = tf(1,[1,1,1])
print sys
t,y=step_response(sys)
plt.title('response')
plt.plot(t,y,label='step')

t,y=impulse_response(sys)
plt.plot(t,y,label='imp')
plt.legend(loc="center right")
plt.grid()
plt.show()
```



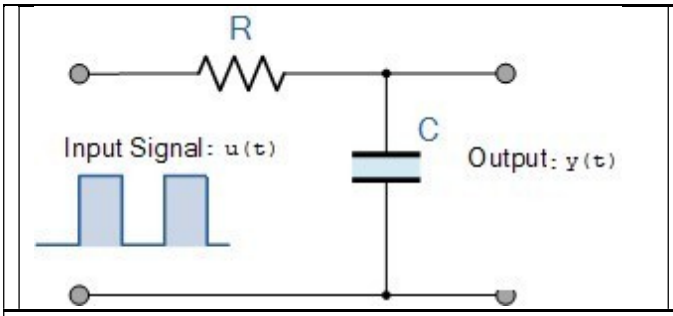
#Visualizza la risposta del #sistema:1/(s^2 + s + 1) #al gradino e all'impulso #visualizzate in due #finestre

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

sys = tf(1,[1,1,1])
print sys
t,y=step_response(sys)
plt.figure()
plt.title('step response')
plt.plot(t,y,label='step'
plt.grid()

t,y=impulse_response(sys)
plt.figure()
plt.title('impulse response')
plt.plot(t,y,label='imp')
plt.legend(loc="center right")
plt.grid()
plt.show()
```



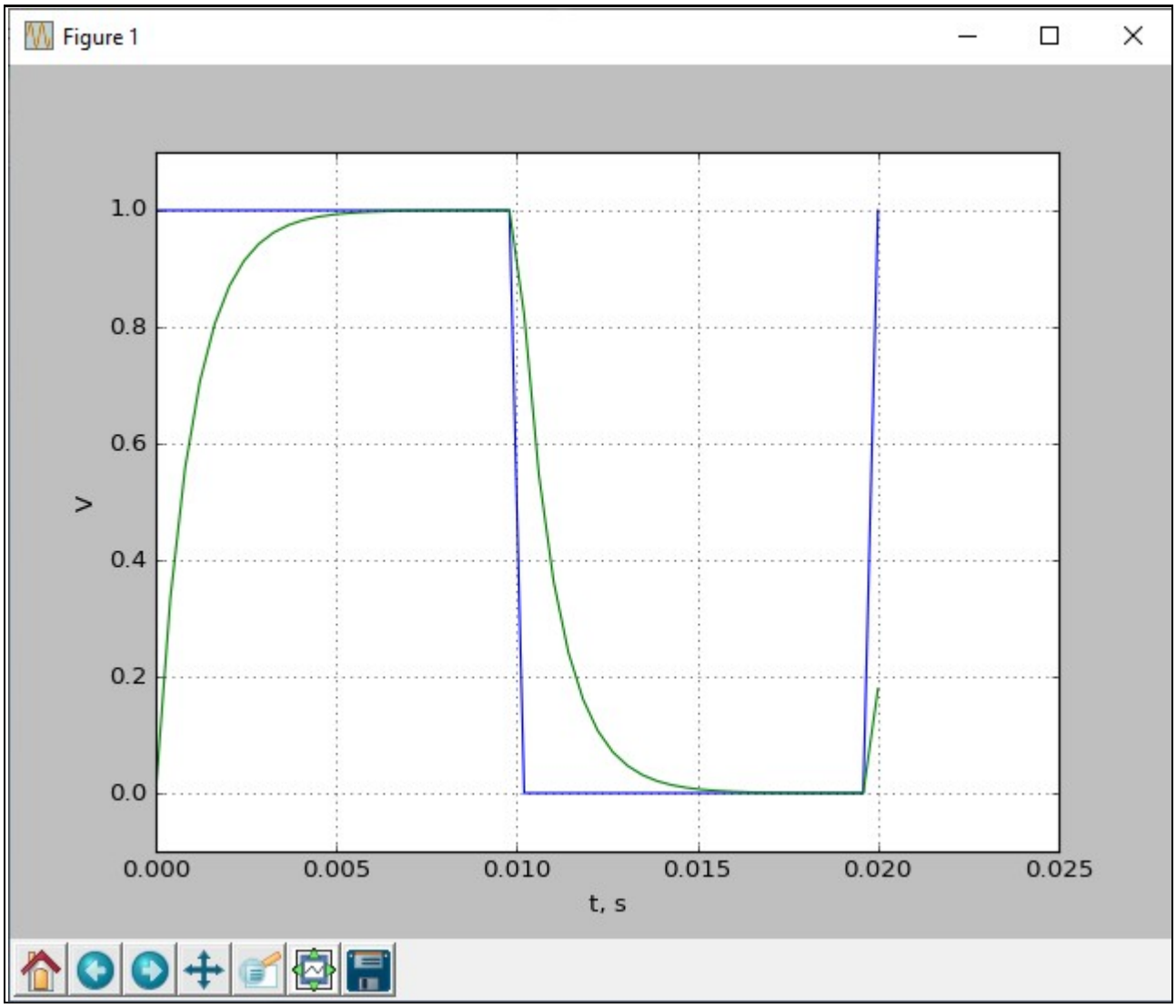
SISTEMI DEL PRIMO ORDINE

$$G(s) = \frac{1}{1 + \tau \cdot s}$$



R=1K
C=1uF
u(t): 0-1V, f=20ms

```
R = 1000
C = 1E-6
num=[1]
den=[R*C,1]
G_n = control.tf
(num, den)
print G_n

f=1./(20E-3)
Tf=1./f
t = np.linspace(0, Tf, 1./Tf)
pwm = 1*(signal.square(2*np.pi*f*t)+1)/2
t, y, _ = control.forced_response(G_n, T=t, U=pwm)
plt.plot(t, pwm)
plt.plot(t, y)
plt.xlabel('t, s')
plt.ylabel('V')
plt.ylim(-0.1, 1.1)
plt.grid()
plt.show()
```



Esempio: G(s)=1/(1+ts)
tau=1. s = control.tf('s') G=1./(1+tau*s) print G t = np.linspace(0, 5, 100) _,y=control.step_response(G,T=t) plt.plot(t,y) plt.grid() plt.show()

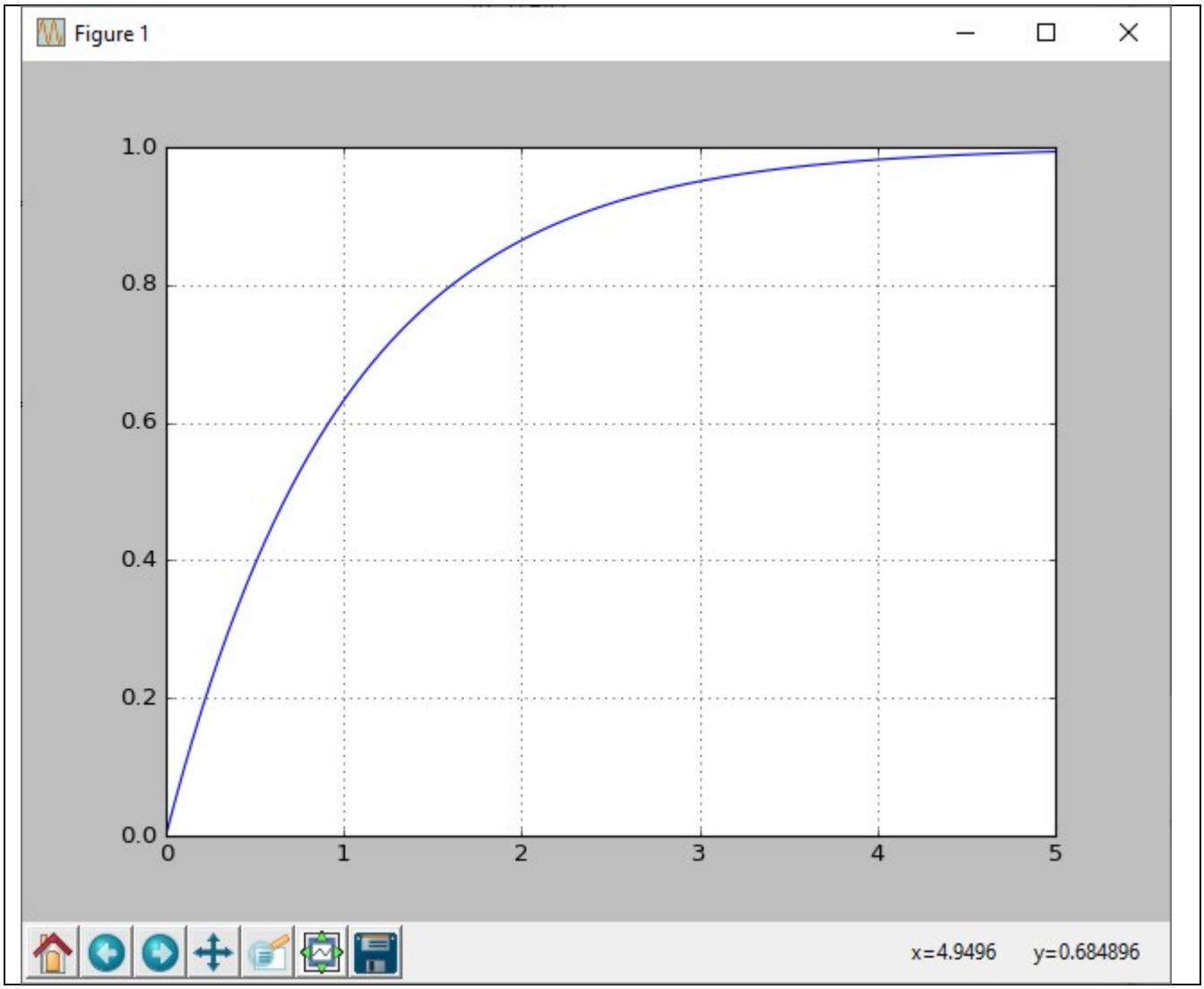


Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
x=4.9496
y=0.684896

SISTEMI DEL SECONDO ORDINE

$$G(s) = \frac{K \cdot \omega_n^2}{s^2 + 2 \cdot \zeta \cdot \omega_n \cdot s + \omega_n^2}$$

Esercizio: determinazione di+ ζ e ω_n

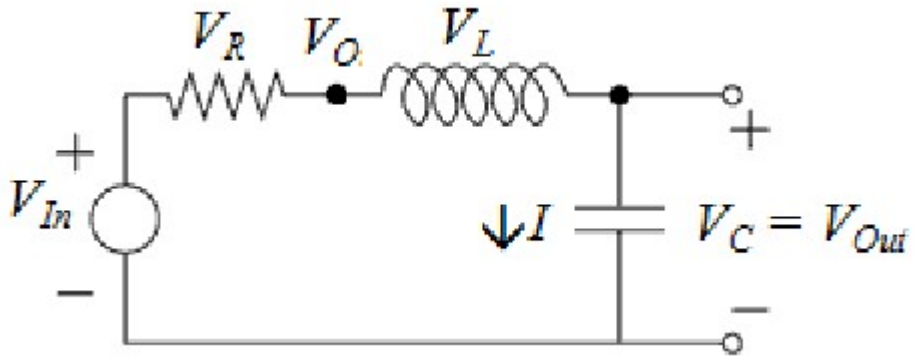
$$G(s) = \frac{200}{s^2 + 50 \cdot s + 100}$$

$$\omega_n^2 = 100 \Rightarrow \omega_n = 10$$

$$2 \cdot \zeta \cdot \omega_n = 50 \Rightarrow \zeta = \frac{50}{2 \cdot 10} = 2.5$$

$$K \cdot \omega_n^2 = 200 \Rightarrow K = \frac{200}{100} = 2$$

Determinazione della fdt di un sistema elettrico



$$G(s) = \frac{K \cdot \omega_n^2}{s^2 + 2 \cdot \zeta \cdot \omega_n \cdot s + \omega_n^2} = \frac{V_s(s)}{V_i(s)} \frac{\frac{1}{LC}}{s^2 + \frac{R}{L} \cdot s + \frac{1}{LC}}$$

$$\omega_n^2 = \frac{1}{LC} \Rightarrow \omega_n = \frac{1}{\sqrt{LC}}$$

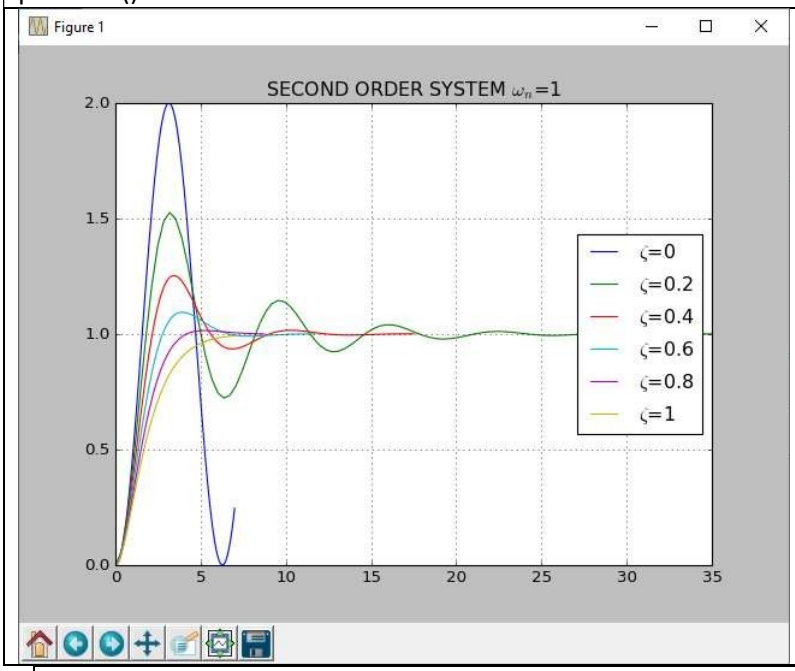
$$2\xi\omega_n = 2\xi \frac{1}{\sqrt{LC}} = \frac{R}{L} \Rightarrow \xi = \frac{R}{L} \cdot \frac{\sqrt{LC}}{2} = \frac{R}{2} \cdot \sqrt{\frac{C}{L}}$$

K=1

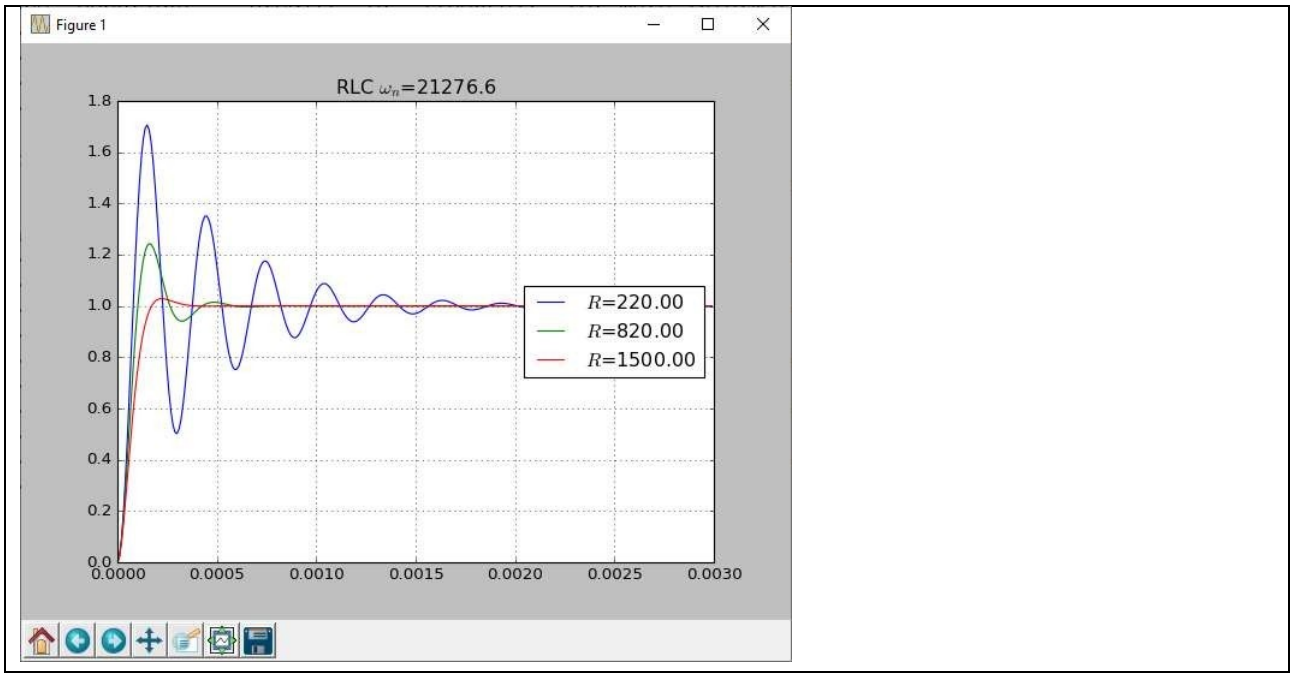
#Risposta al gradino al variare del parametro zita

```
from control import *
import matplotlib.pyplot as plt
import numpy as np
import math
wn=1
zita=[0,0.2,0.4,0.6,0.8,1]
num = [1]

for n in range(len(zita)):
    den = [1,2*zita[n],1]
```

```
    sys = tf(num,den)
    t,y=step_response(sys)
    plt.plot(t,y,label=r'$\zeta$='+str(zita[n]))
    plt.title('SECOND ORDER SYSTEM ' +r'$\omega_n$='+str(wn))
    plt.legend(loc="center right")
  plt.grid()
plt.show()
```

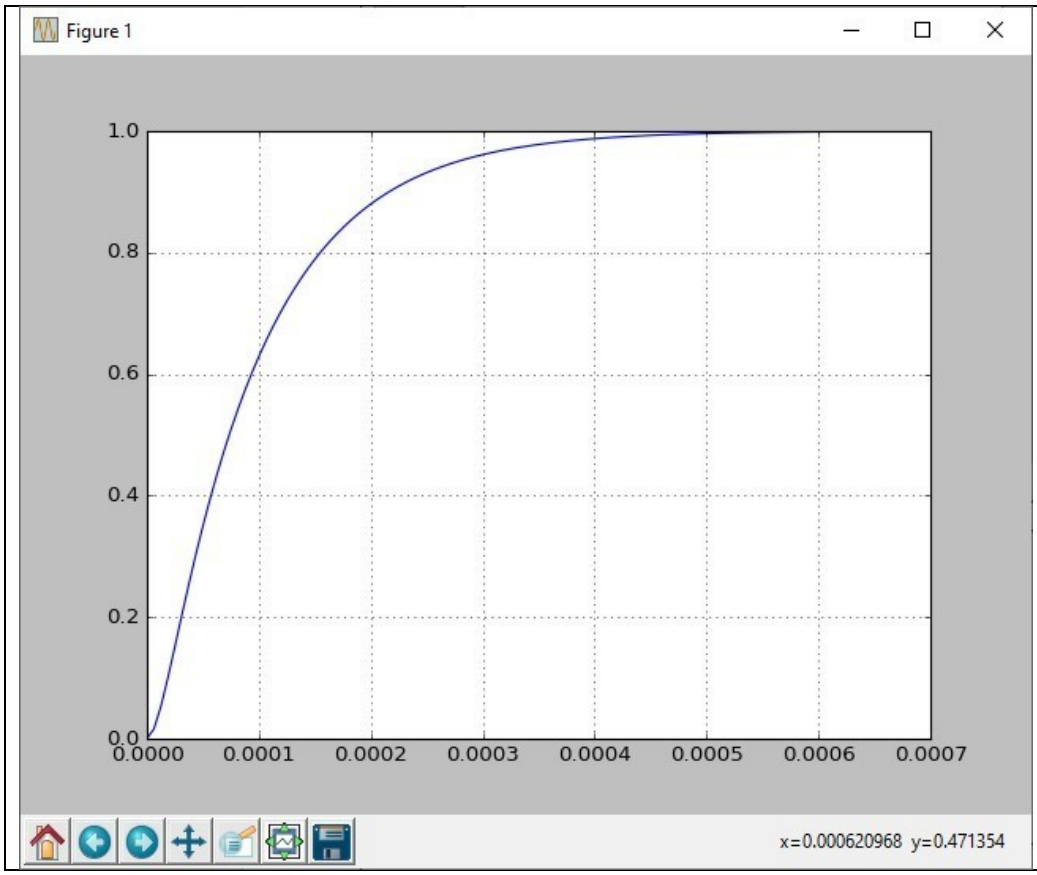



```
#Risposta al gradino del sistema RLC Serie., in un intervallo temporale 0, 0.003 sec
#ad intervalli 10E-6 sec L=47mH, C=47nF, R=220,820,1.5K
 L=47E-3
C=47E-9
RR=[220,820,1.5E3]
K=1
wn=1./math.sqrt(L*C)
num = [K*wn**2]
ta=np.arange(0,0.003,0.00001)
for R in RR:
  z=(R/2)*math.sqrt(C/L)
  den = [1,2*z*wn,wn**2]
  sys = tf(num,den)
  t,y=step_response(sys,ta)
  plt.plot(t,y,label=r'$R$='+str(format(R, '.2f')))

plt.title('RLC ' +r'$\omega_n$='+str(round(wn,2)))
plt.legend(loc="center right")
plt.grid()
plt.show()
```

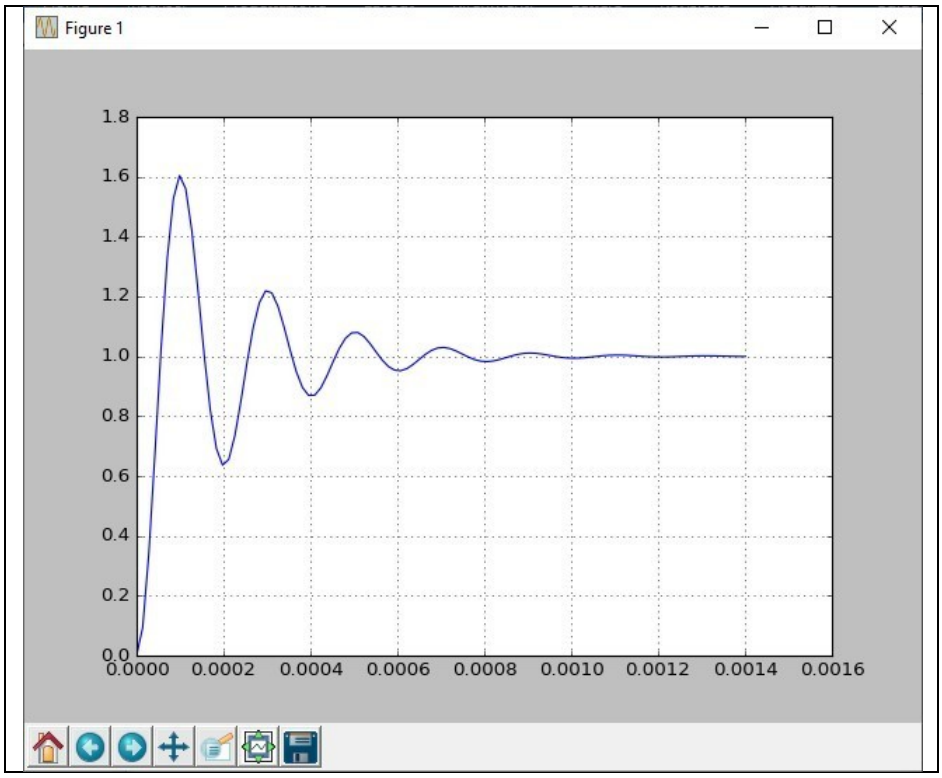


Risposta al gradino di un sistema del secondo ordine

```
#
#              1e+09
# -------------------------------------
# s^2 + 1e+05 s + 1e+09

s = control.tf('s')
L=0.001
C=0.000001
R=100.
wn=1./((L*C)**(1./2))
print wn
csi=(R/2)*((C/L)**(1./2))
print csi
G=wn**2/(s**2+2*csi*wn*s+wn**2)

print G----------
t = np.linspace(0, 0.001, 100)
_,y=control.step_response(G,T=t)
plt.plot(t,y)
plt.grid()
plt.show()
```

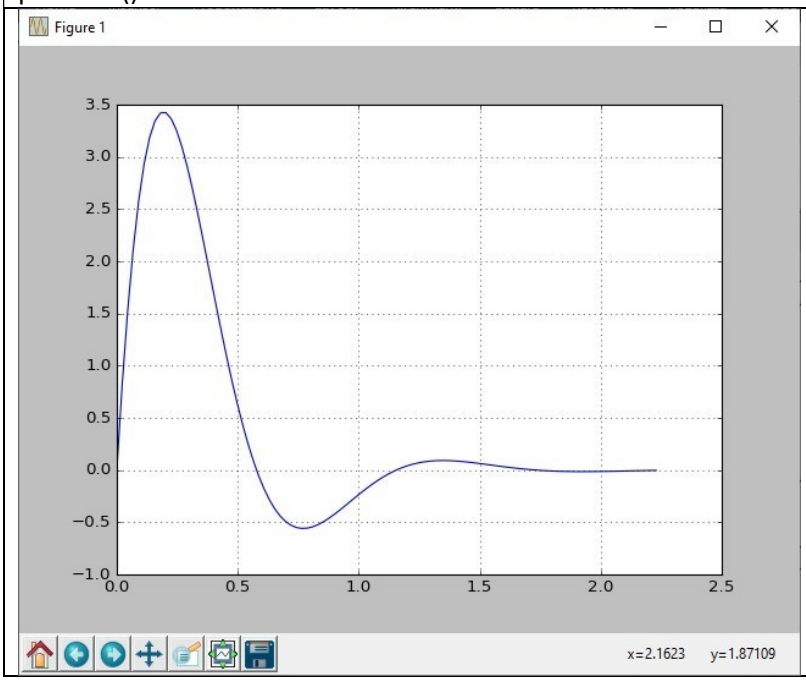



```
#Risposta al gradino di un sistema del secondo ordine (con R=10)
#          1e+09
#-----------------------------
# s^2 + 1e+04 s + 1e+09

L=1E-3
C=1E-6
R=10
wn=1./((L*C)**(1./2))
csi=(R/2)*(C/L)**(1./2)
s=control.tf([wn**2],[1,2*csi*wn,wn**2])
print s
t,y=control.step_response(s)
plt.plot(t,y)
plt.grid()
plt.show()
```



Risposta all' impulso di un sistema del secondo ordine

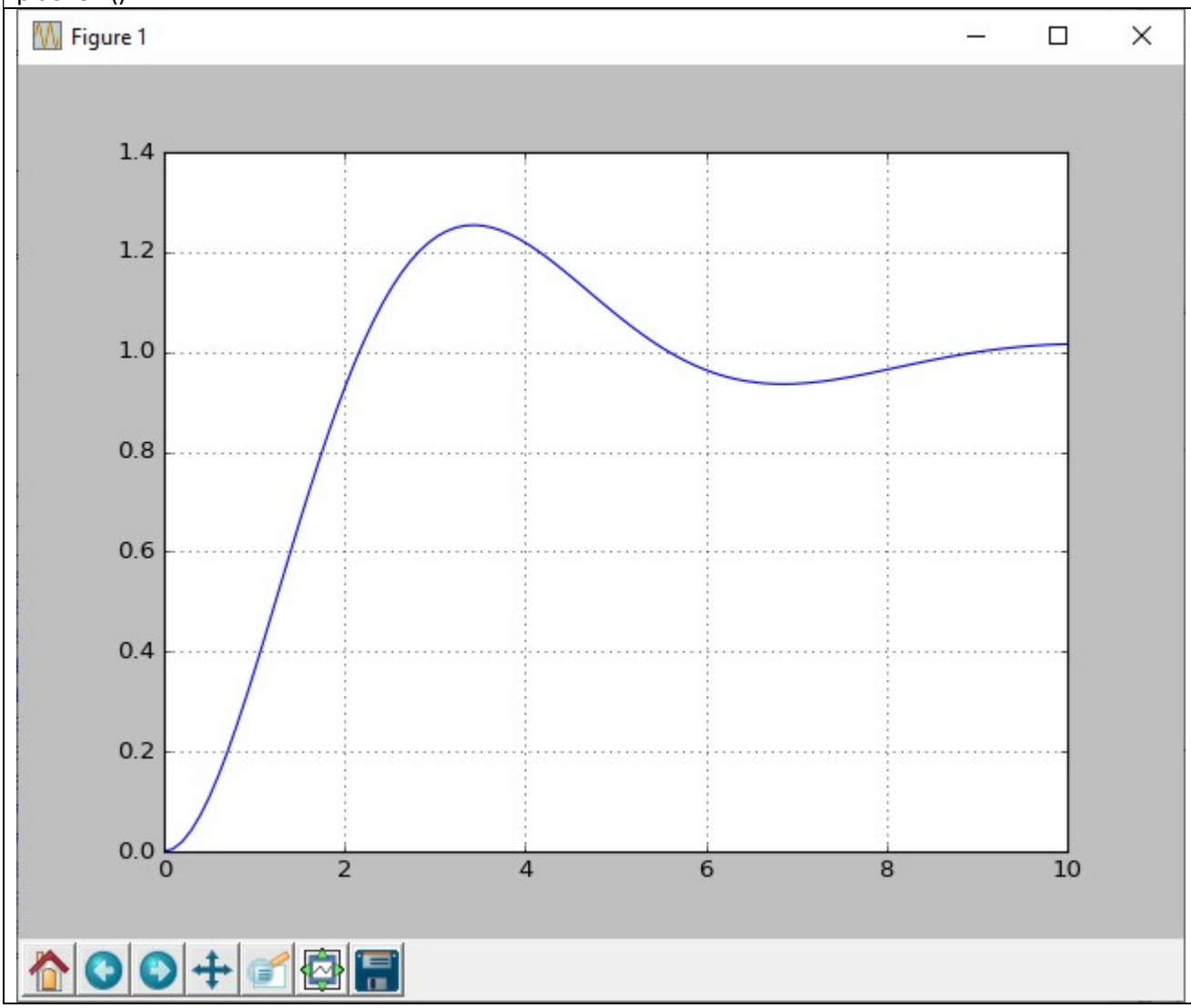
```
#         39.44
# ------------------------------
# s^2 + 6.28 s + 39.44

wn=6.28
csi=0.5
sys = control.tf([wn**2],[1, 2*csi*wn, wn**2])
print sys
t,y=impulse_response(sys)
plt.plot(t,y)
plt.grid()
plt.show()
```



Esercizio: Risposta al gradino del sistema del secondo ordine con $\zeta = 0.4$ e $\omega_n = 1$

$$G(s) = \frac{K \cdot \omega_n^2}{s^2 + 2 \cdot \zeta \cdot \omega_n \cdot s + \omega_n^2} = \frac{1}{s^2 + 2 \cdot \zeta \cdot s + 1}$$

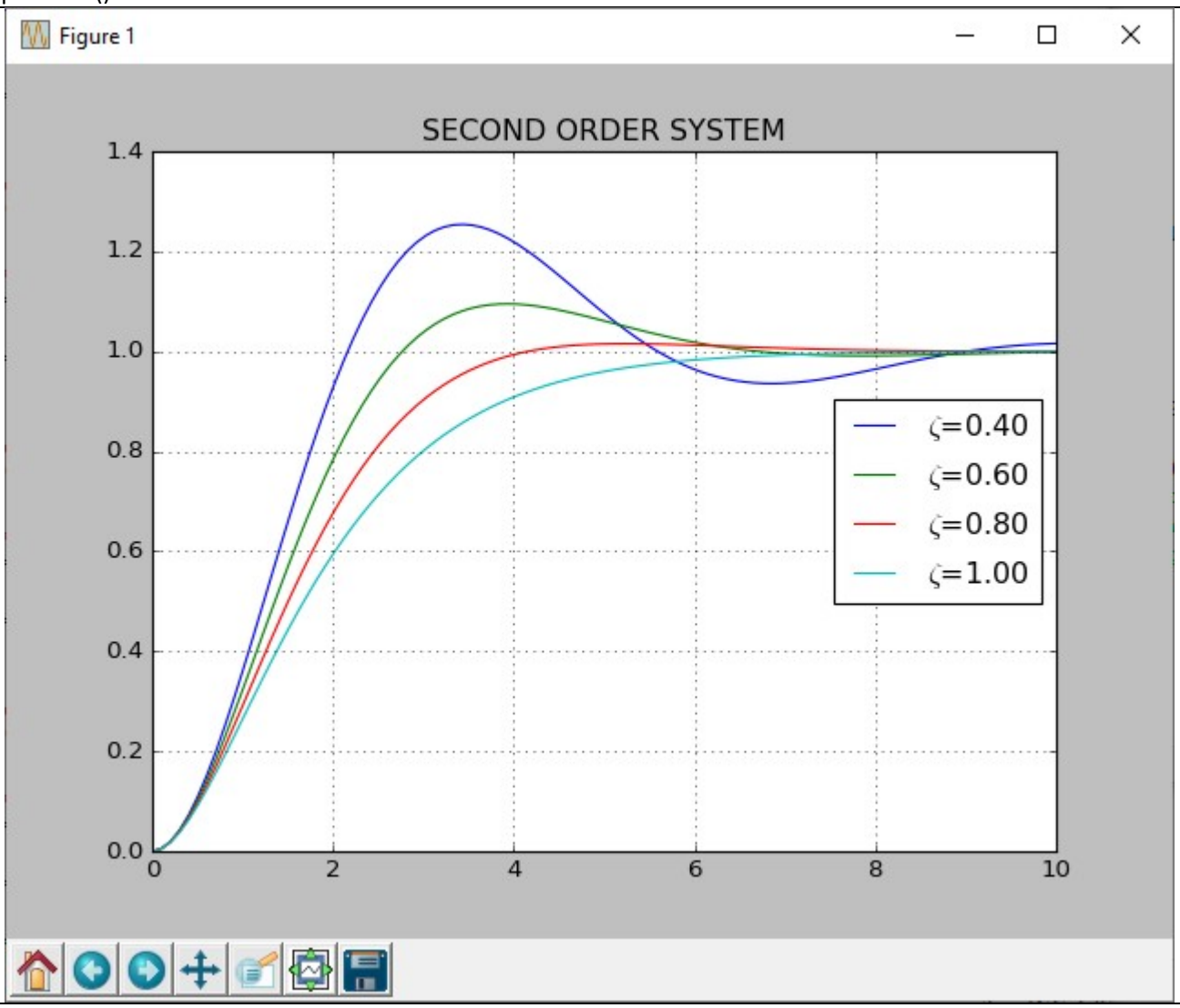
```
#Risposta al gradino di un sistema del secondo ordine
#          1
#-----------------------------
# s^2 + 0.8 s + 1

s=control.tf([1],[1,0.8,1])
print s
t = np.linspace(0, 10, 100)
_,y=control.step_response(s,T=t)
plt.plot(t,y)
plt.grid()
plt.show()
```



ξ che varia 0.4 a 1 con passo 0.2

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

s = control.tf('s')
t = np.linspace(0,10, 100)
for csi in np.arange(0.4,1+0.2,0.2):
```

```
    G=1./(s**2+2*csi*s+1)
    _,y=control.step_response(G,T=t)
    plt.plot(t,y,label=r'$\zeta$='+"{:.2f}".format(csi))#str(csi))
    plt.title('SECOND ORDER SYSTEM')
    plt.legend(loc="center right")
plt.grid()
plt.show()
```



Esempio con valori di ξ presi da un vettore

```
s = control.tf('s')
t = np.linspace(0,10, 100)
coefficienti=[2,1,0.707,0.5,0.3,0.1]
for I in range(len(coefficienti)):
    G=1./(s**2+2*coefficienti[I]*s+1)
    _,y=control.step_response(G,T=t)
    plt.plot(t,y,label=r'$\zeta$='+"{:.2f}".format(coefficienti[I]))
    plt.title('SECOND ORDER SYSTEM')
    plt.legend(loc="lower right",)
plt.grid()
plt.show()
```

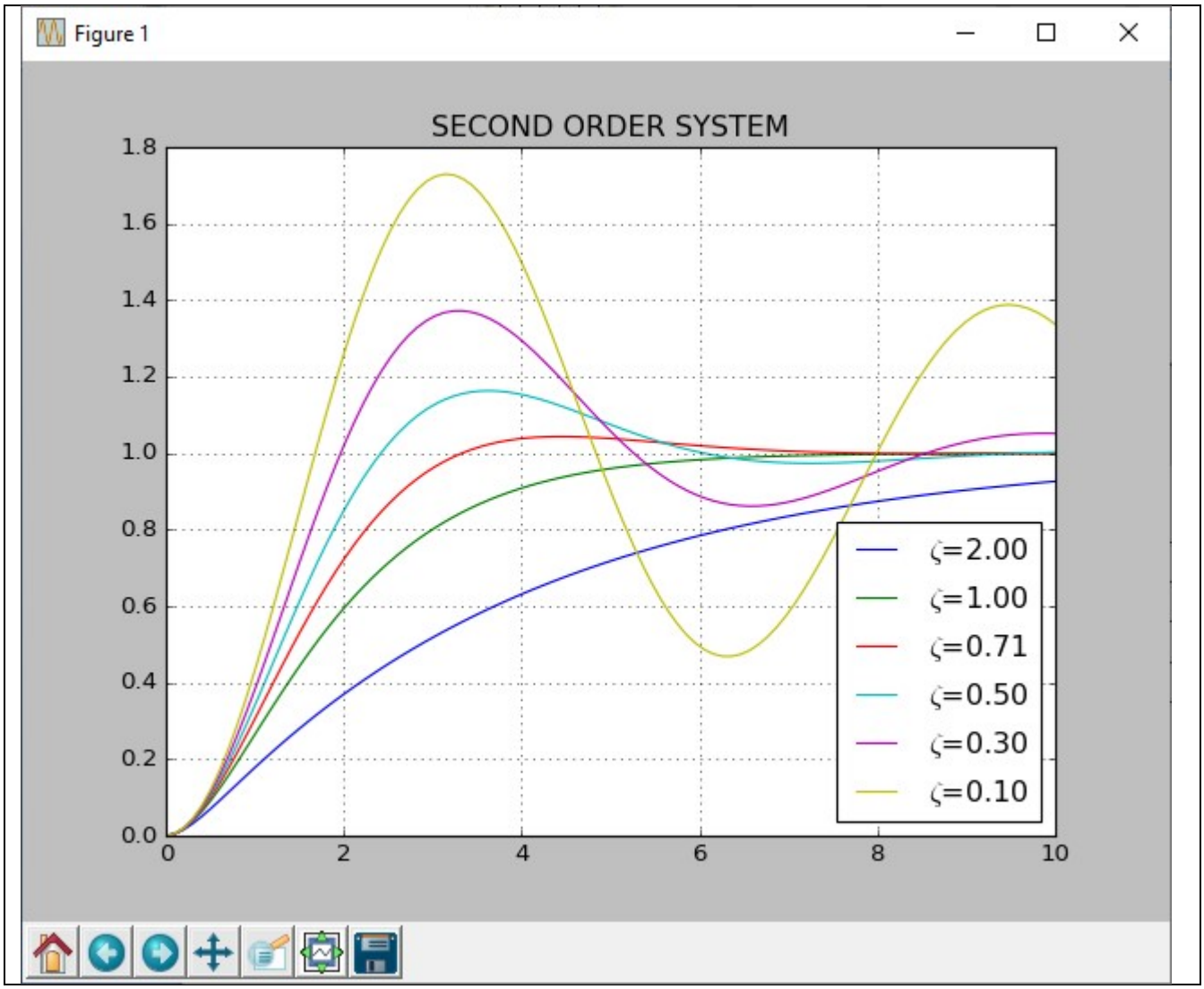
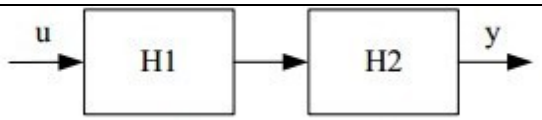


Figure 1
SECOND ORDER SYSTEM
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
ζ=2.00
ζ=1.00
ζ=0.71
ζ=0.50
ζ=0.30
ζ=0.10

ALGEBRA DEI BLOCCHI

SERIE



$$\frac{y(s)}{u(s)} = H(s) = H_2(s)H_1(s)$$

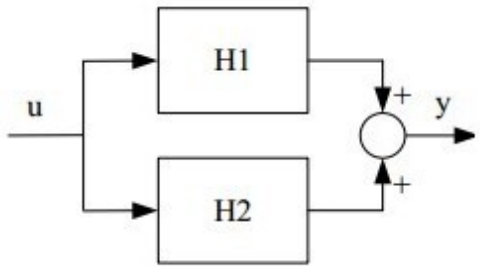
$$H_1(s) = \frac{2}{s} \qquad H_2(s) = \frac{3}{4s+1}$$

$$H(s) = \frac{2}{s} \cdot \frac{3}{4s+1} = \frac{2 \cdot 3}{Ts^2 + s} = \frac{6}{4s^2 + s}$$

```
import numpy as np
import control

K1 = 2
K2 = 3
T = 4
num1 = np.array ([ K1 ])
den1 = np.array ([1 , 0])
num2 = np.array ([ K2 ])
den2 = np.array ([T, 1])
H1 = control.tf(num1 , den1 )
H2 = control.tf(num2 , den2 )
 H = control.series (H1 , H2)print 'H =', H
```

```
        6
H=  ----------
    4 s^2 + s
```

PARALLELO



$$\frac{y(s)}{u(s)} = H(s) = H_2(s) + H_1(s)$$

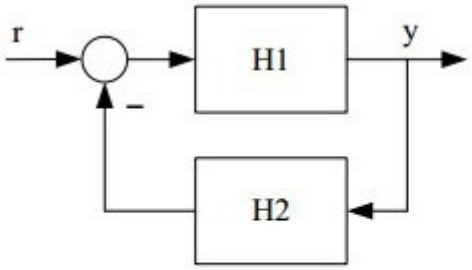
$$H_1(s) = \frac{2}{s} \qquad H_2(s) = \frac{3}{4s+1}$$

$$H(s) = \frac{2}{s} + \frac{3}{4s+1} = \frac{2(4s+1) + 3s}{s(4s+1)} = \frac{11s+2}{4s^2+s}$$

```
import numpy as np
import control
K1 = 2
K2 = 3
T = 4
num1 = np.array ([ K1 ])
den1 = np.array ([1 , 0])
num2 = np.array ([ K2 ])
den2 = np.array ([T, 1])
H1 = control.tf(num1 , den1 )
H2 = control.tf(num2 , den2 )
H = control.parallel (H1 , H2)
print 'H =', H
```

```
        11 s + 2
H=  ----------
    4 s^2 + s
```

FEEDBACK



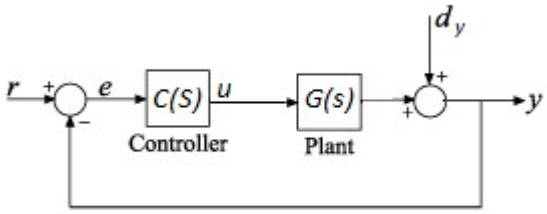
$$\frac{y(s)}{r(s)} = H(s) = \frac{H_1(s)}{1 + H_1(s)H_2(s)}$$

$$H_1(s) = \frac{2}{s} \qquad H_2(s) = 1 \qquad H(s) = \frac{\frac{2}{s}}{1 + \frac{2}{s}} = \frac{2}{s+2}$$

```
import numpy as np
import control
num = np.array ([2])
den = np.array ([1 , 0])
 L = control.tf(num , den )
H = control.feedback (L, 1)
print 'H =', H
```

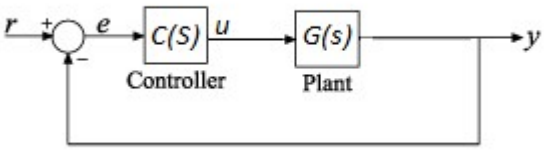
```
        2
H=  -------
        S+2
```

Blocchi con disturbo additivo



Applichiamo il principio di sovrapposizione degli effetti.

1) dy=0



Funzione di trasferimento da r a y:

$$F(s) = \frac{Y(s)}{R(s)} = \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s)}$$

Funzione di trasferimento da r a e:

$$E(s) = R(s) - Y(s) = R(s) - \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s)} \cdot R(s) = \frac{1}{1 + C(s) \cdot G(s)} \cdot R(s)$$

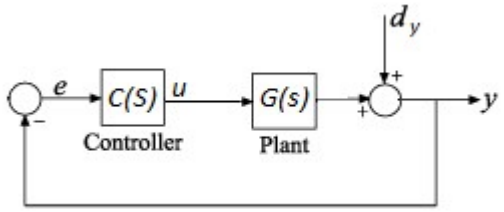
$$\Rightarrow T(s) = \frac{E(s)}{R(s)} = \frac{1}{1 + C(s) \cdot G(s)}$$

Funzione di trasferimento da r a u:

$$U(s) = C(S) \cdot E(s) = \frac{C(s)}{1 + C(S) \cdot G(s)} \cdot R(s)$$

$$\Rightarrow S(s) = \frac{U(s)}{R(s)} = \frac{C(s)}{1 + C(S) \cdot G(s)}$$

2)r=0



Funzione di trasferimento da dy a y:

$$Y(s) = D_y(s) + \frac{C(s)\cdot G(s)}{1 + C(s)\cdot G(s)} \cdot E(s) = D_y(s) + \frac{C(s)\cdot G(s)}{1 + C(s)\cdot G(s)} \cdot \big(-Y(s)\big)$$

$$Y(s) = \frac{1}{1 + C(s)\cdot G(s)} \cdot D_y(s)$$

Funzione di trasferimento da dy a e:
Dato che E(s)=-Y(s)
T(s)=-T(s) precedentemente calcolata

Funzione di trasferimento da dy a u:

$$U(s) = C(s)\cdot E(s) = \frac{C(s)}{1 + C(s)\cdot G(s)} \cdot D_y(s)$$

Funzione di trasferimento da dy a y:

$$W(s) = \frac{G(s)}{1 + C(s)\cdot G(s)}$$

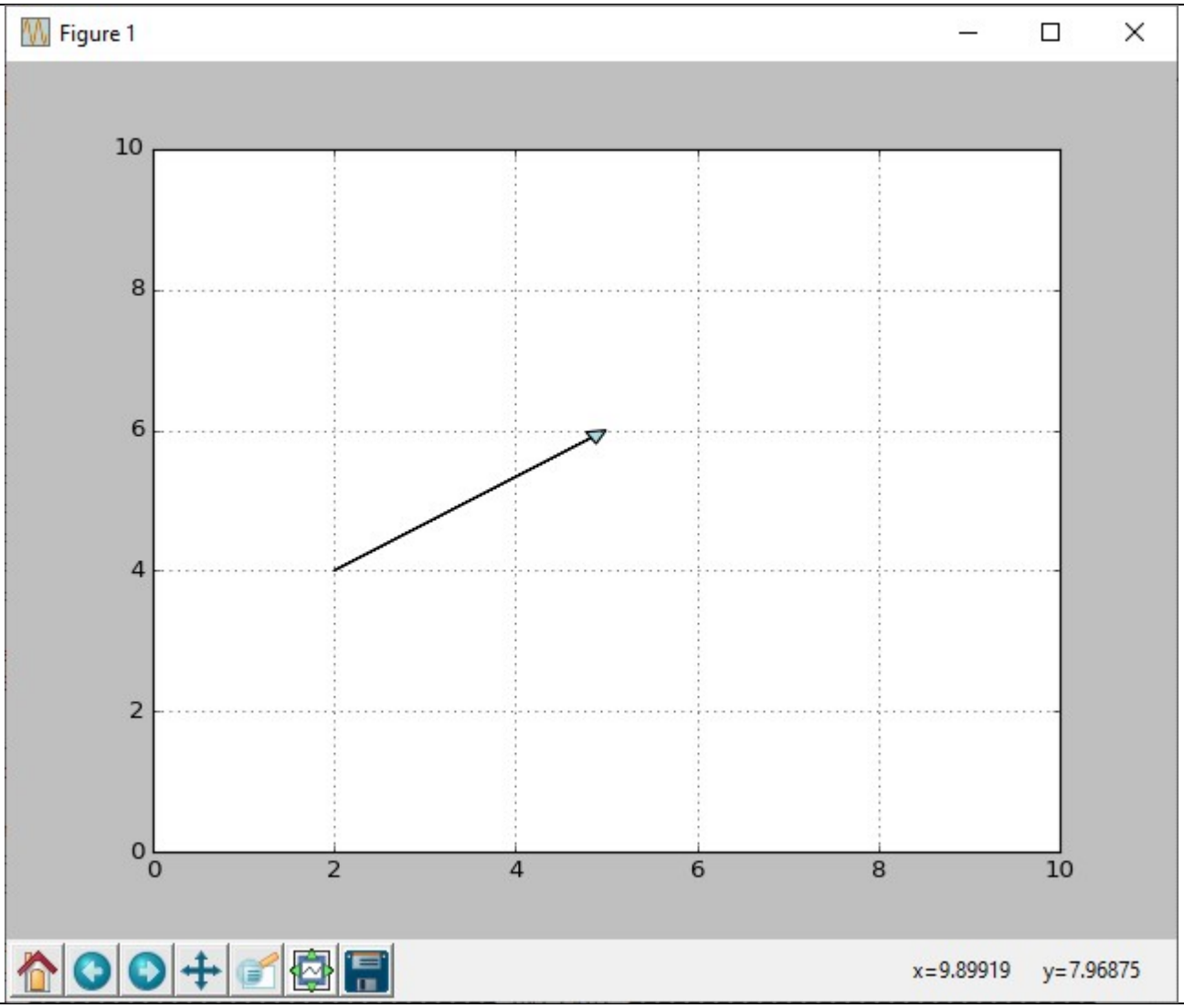
Uscita complessiva:

$$Y(s) = \frac{C(s)G(s)}{1 + C(s)\cdot G(s)} \cdot R(s) + \frac{1}{1 + C(s)\cdot G(s)} \cdot D_y(s)$$

IL DOMINIO DELLA FREQUENZA

VETTORI

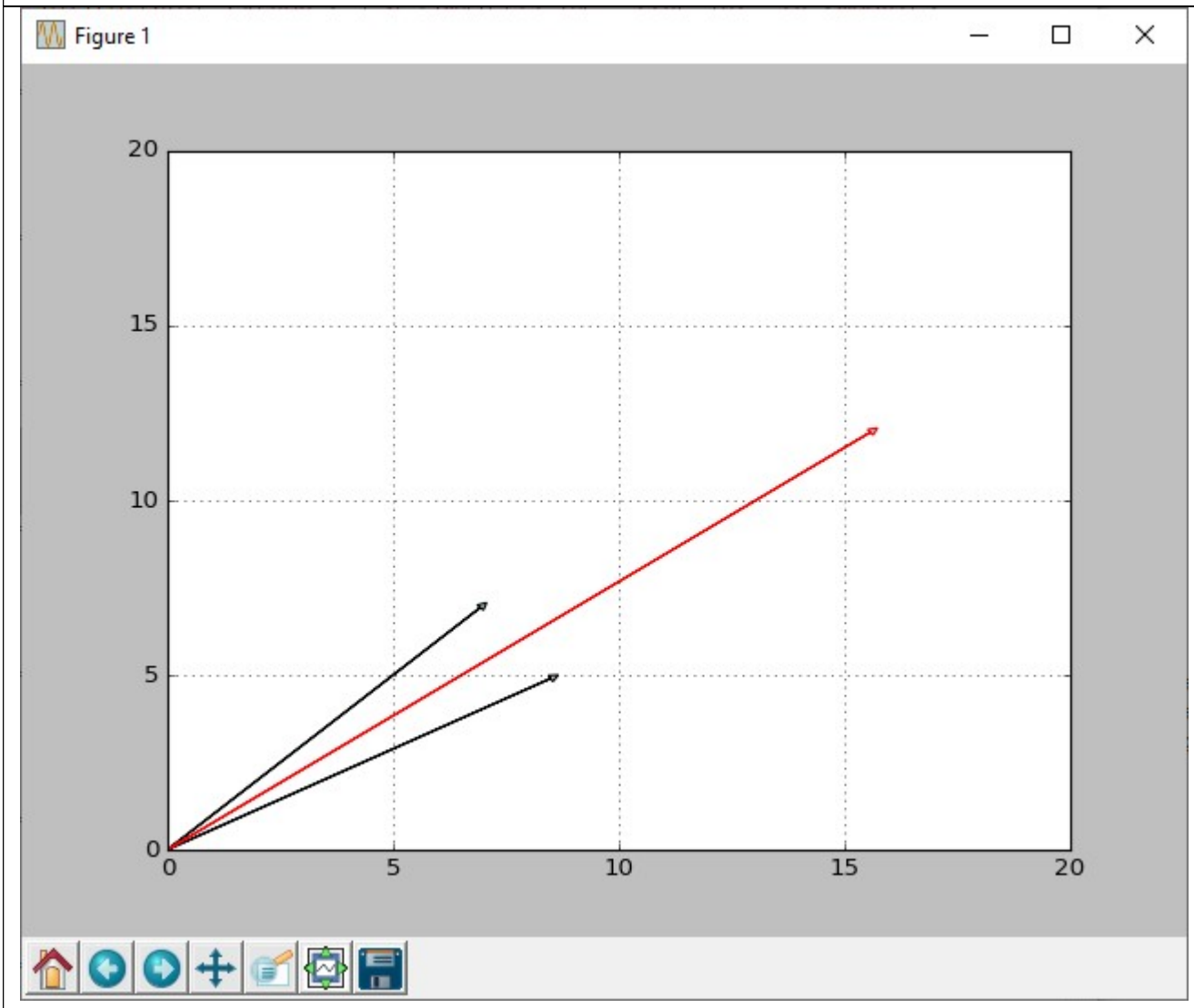
```
import matplotlib.pyplot as plt

P1=[2,4]
D=[3,2] #N.B. !!! Lunghezza del vettore lungo x e y
plt.arrow(P1[0],P1[1],D[0],D[1], head_width=0.2, head_length=0.2, fc='lightblue', ec='black',
length_includes_head=True)

plt.grid()

plt.xlim(0,10)
plt.ylim(0,10)

plt.show()
plt.close()
```



Somma di due vettori

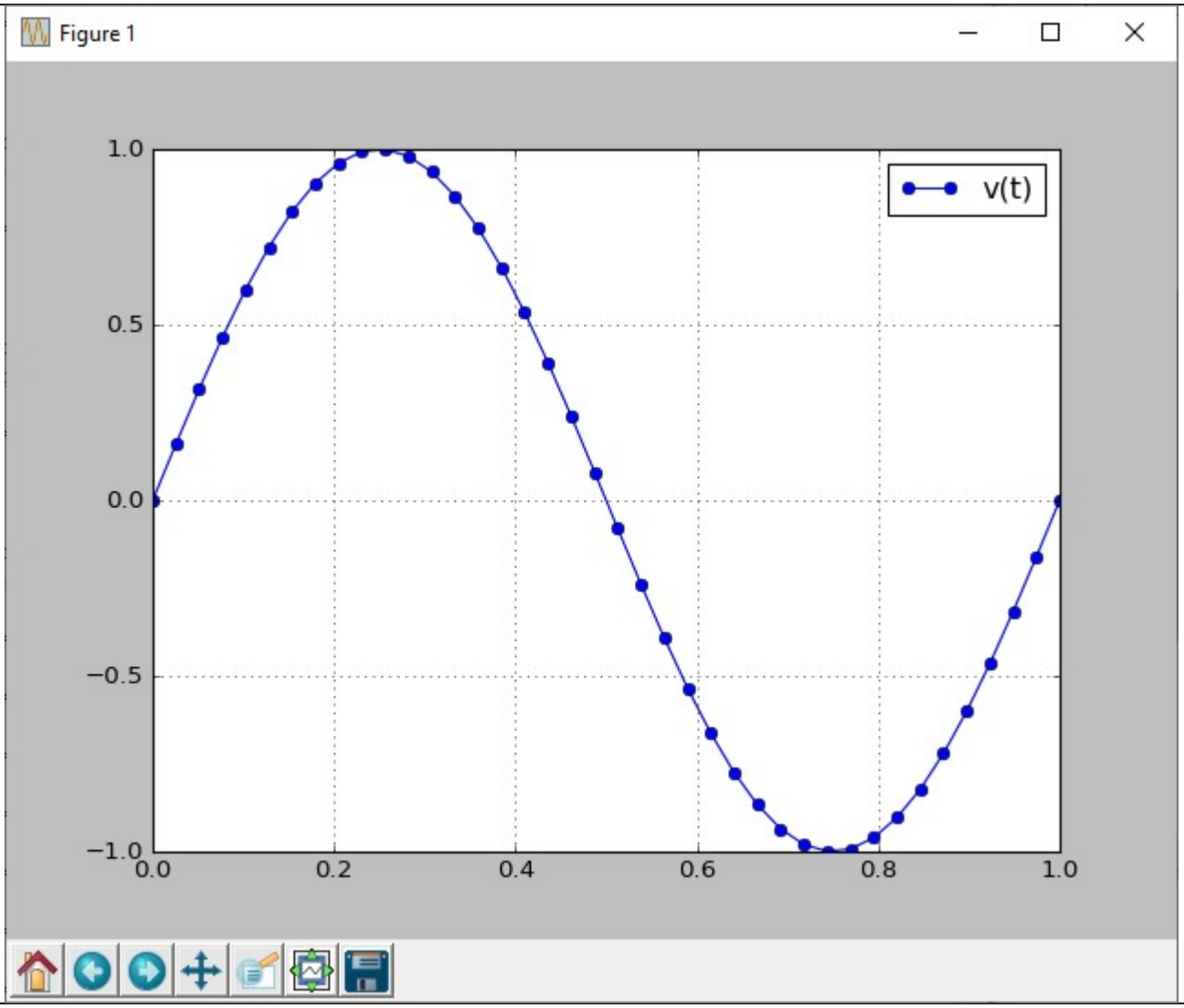
```
phi1=np.pi/4.#math.radians()
V1M=10
V1=V1M*math.sin(2*np.pi+phi1)
MV1=abs(V1M)
ReV1=MV1*math.cos(phi1)
ImV1=MV1*math.sin(phi1)
print ReV1,ImV1
c1=complex(ReV1,ImV1)

phi2=np.pi/6.
V2M=10
V2=V2M*math.sin(2*np.pi+phi2)
MV2=abs(V2M)
ReV2=MV2*math.cos(phi2)
ImV2=MV2*math.sin(phi2)
print ReV2,ImV2
c2=complex(ReV2,ImV2)

c3=c1+c2
import cmath
print c3.real,c3.imag,abs(c3),cmath.phase(c3)

plt.arrow(0,0,c1.real,c1.imag, head_width=0.2, head_length=0.2, fc='lightblue', ec='black', length_includes_head=True)
plt.arrow(0,0,c2.real,c2.imag, head_width=0.2, head_length=0.2, fc='lightblue', ec='black', length_includes_head=True)
plt.arrow(0,0,c3.real,c3.imag, head_width=0.2, head_length=0.2, fc='lightblue', ec='red', length_includes_head=True)
plt.grid()
```

```
plt.xlim(0,20)
plt.ylim(0,20)

plt.show()
plt.close()

>>>
7.07106781187 7.07106781187
8.66025403784 5.0
15.7313218497 12.0710678119 19.8288972275 0.654498469498
```



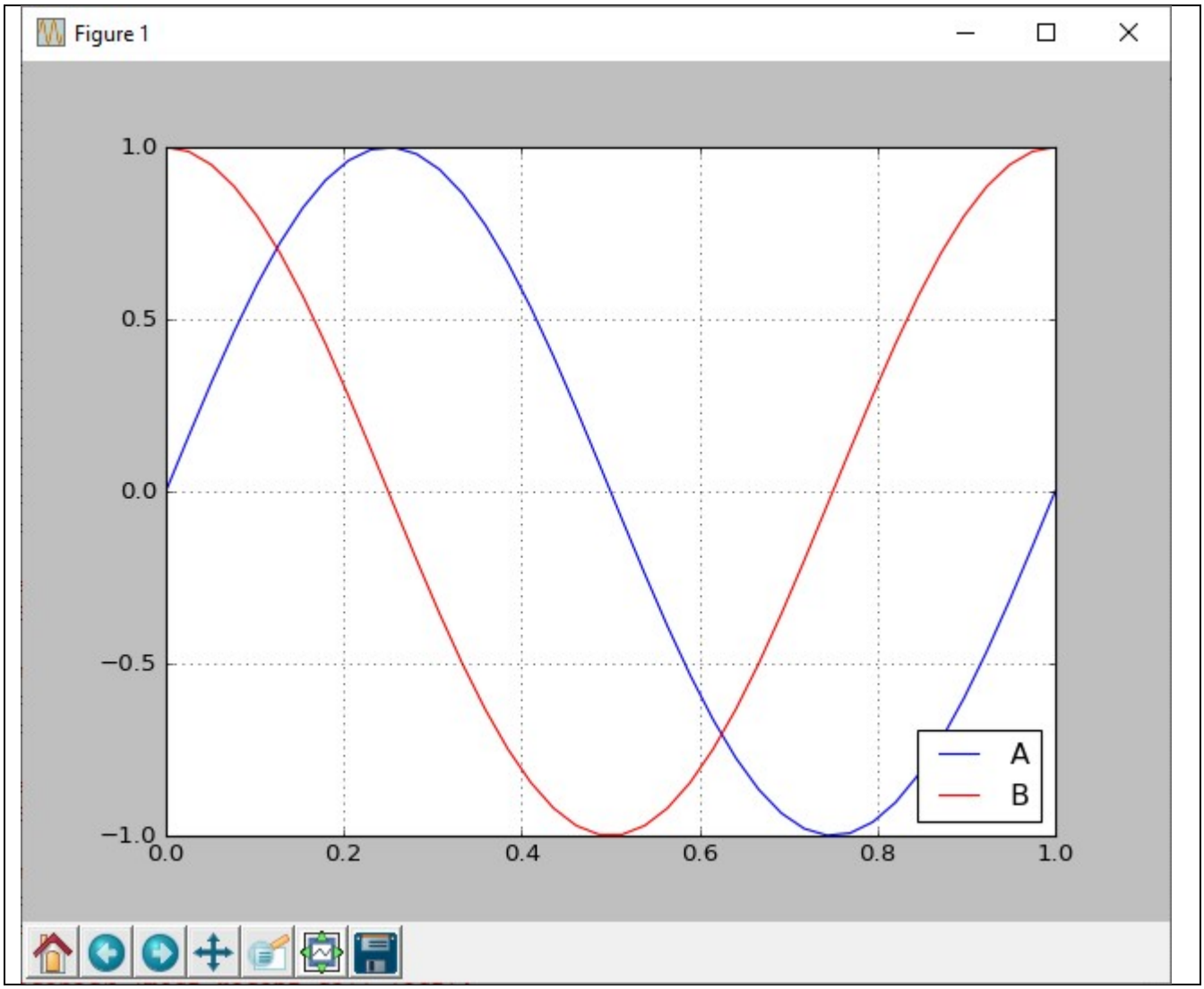
Una sinusoide: f=1Hz, fase=0, ampiezza=1

```
V=1
f1=1

phi=math.radians(0)
dt=1./40
t = np.linspace(0.0, 1, 1./dt)
v=V*np.sin(2*np.pi*f1*t + phi)
plt.plot(t,v, "-bo", label="v(t)")
plt.legend(loc="upper right")
plt.grid()
plt.show()
```



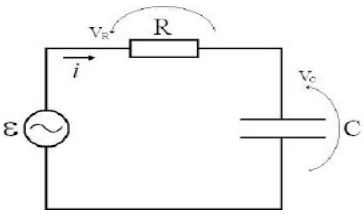
Due sinusoidi

```
A1=1
B1=1
f1=1
f2=1
phi1=0
phi2=math.radians(90)
dt=1./40
t = np.linspace(0.0, 1, 1./dt)
A=A1*np.sin(2*np.pi*f1*t + phi1)
B=B1*np.sin(2*np.pi*f2*t + phi2)

plt.plot(t,A, "-b", label="A")
plt.plot(t,B, "-r", label="B")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



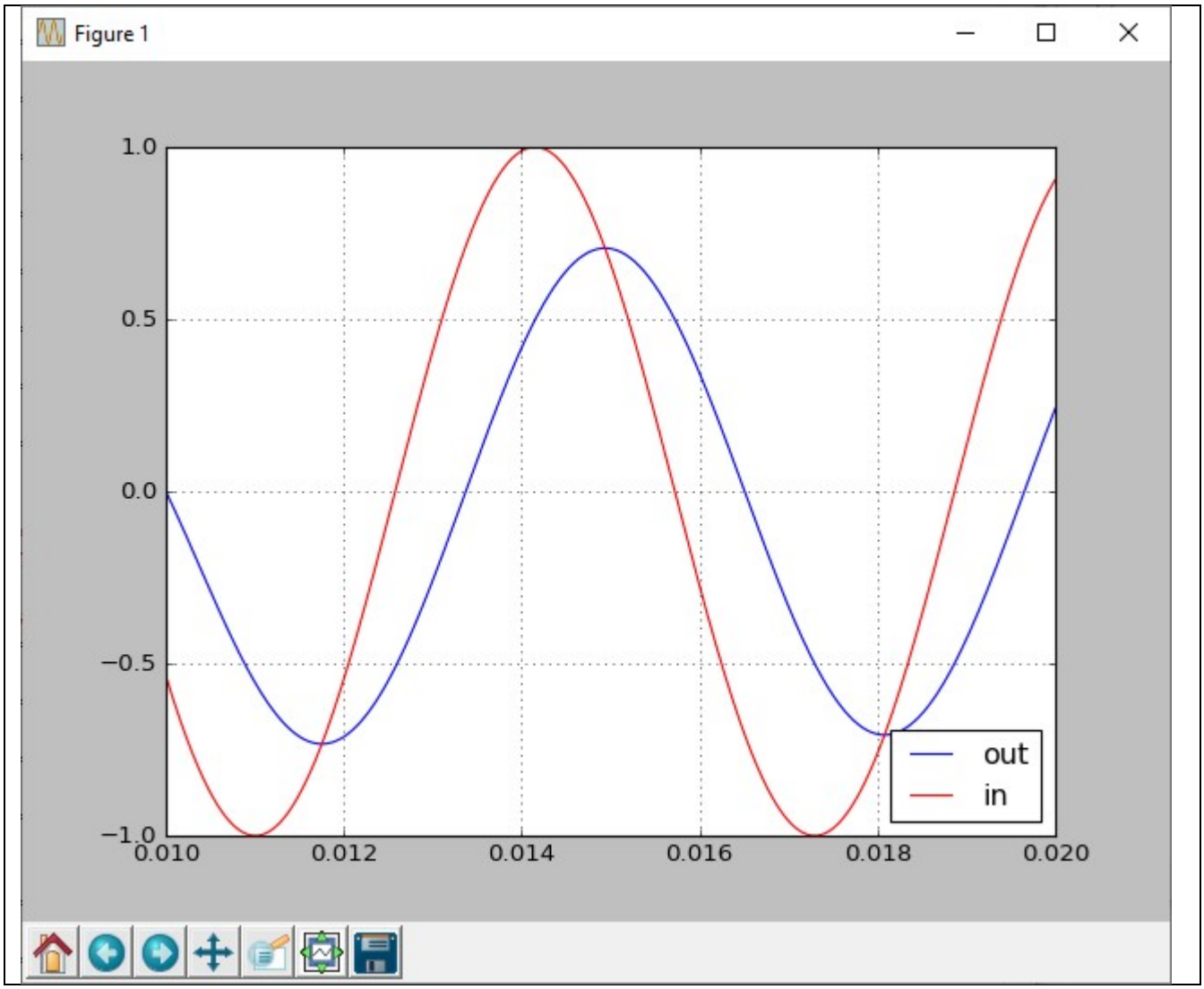
RISPOSTA IN FREQUENZA

Simulazione circuito RC



R=1K C=1uF f=159Hz

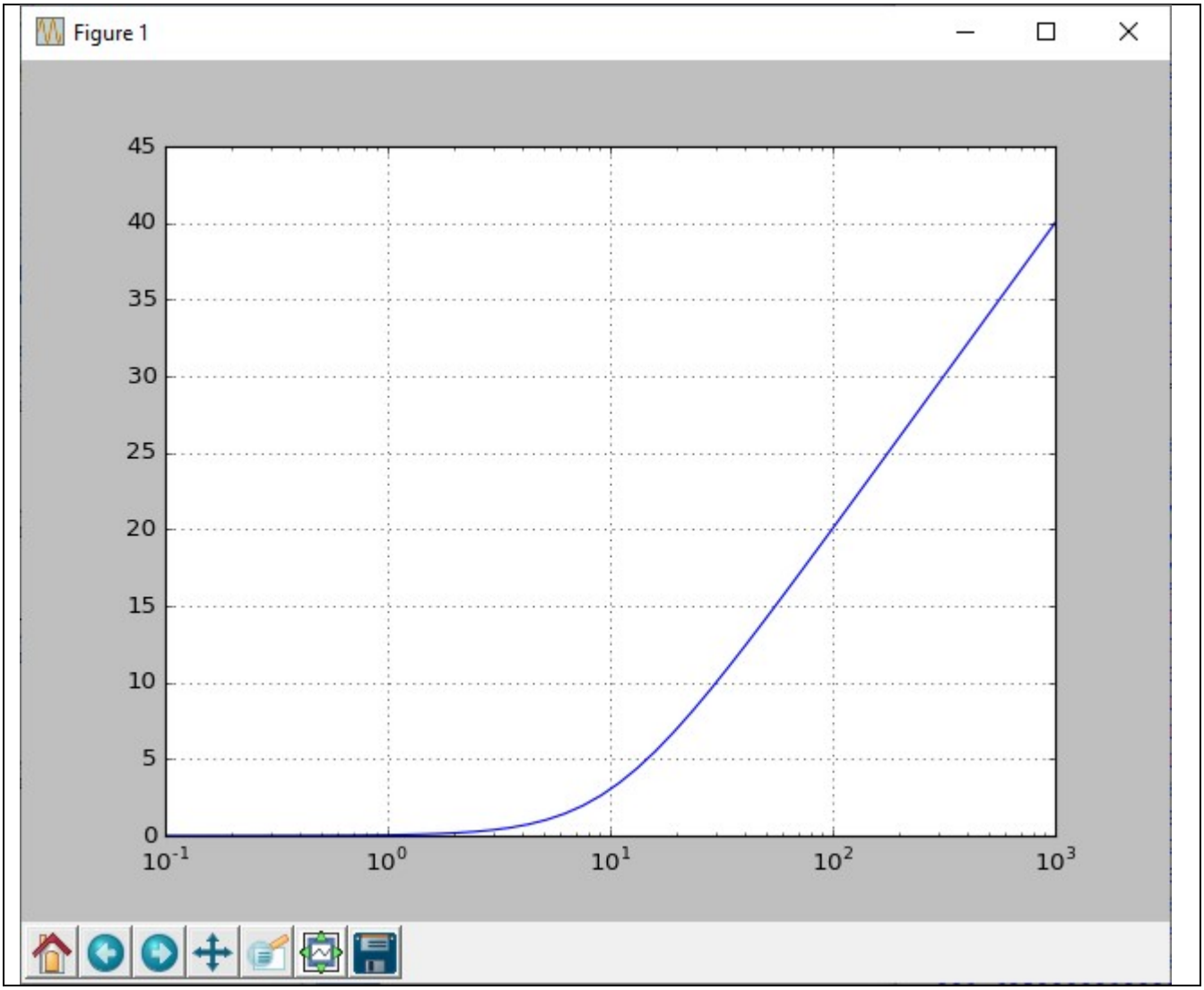
```
import numpy as np
import control
import matplotlib.pyplot as plt

t = np.linspace(0.01, 0.02, 1000)
f = 159
phi = 0
u = np.sin(2*np.pi*f*t + phi)
C=1.E-6
R=1.E3
s = control.tf('s')
G=(1.)/(1.+s*R*C)
y,_,_=mlab.lsim(G, u, t)
plt.plot(t,y, "-b", label="out")
plt.plot(t,u, "-r", label="in")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



***** BODE *****

Per il disegno dei diagrammi di Bode del modulo in dB e della fase si puo' usare i metodi descritti negli esempi seguenti:

Esempio di grafico del modulo della fdt:G[db]=20*Log(|1+0.1*jw|)
def G(w):   return (1+0.1*1j * w) w = np.logspace(-1,3) Gdb=20*np.log10(abs(G(w))) plt.plot(w, Gdb) plt.xscale('log') plt.grid() plt.show()

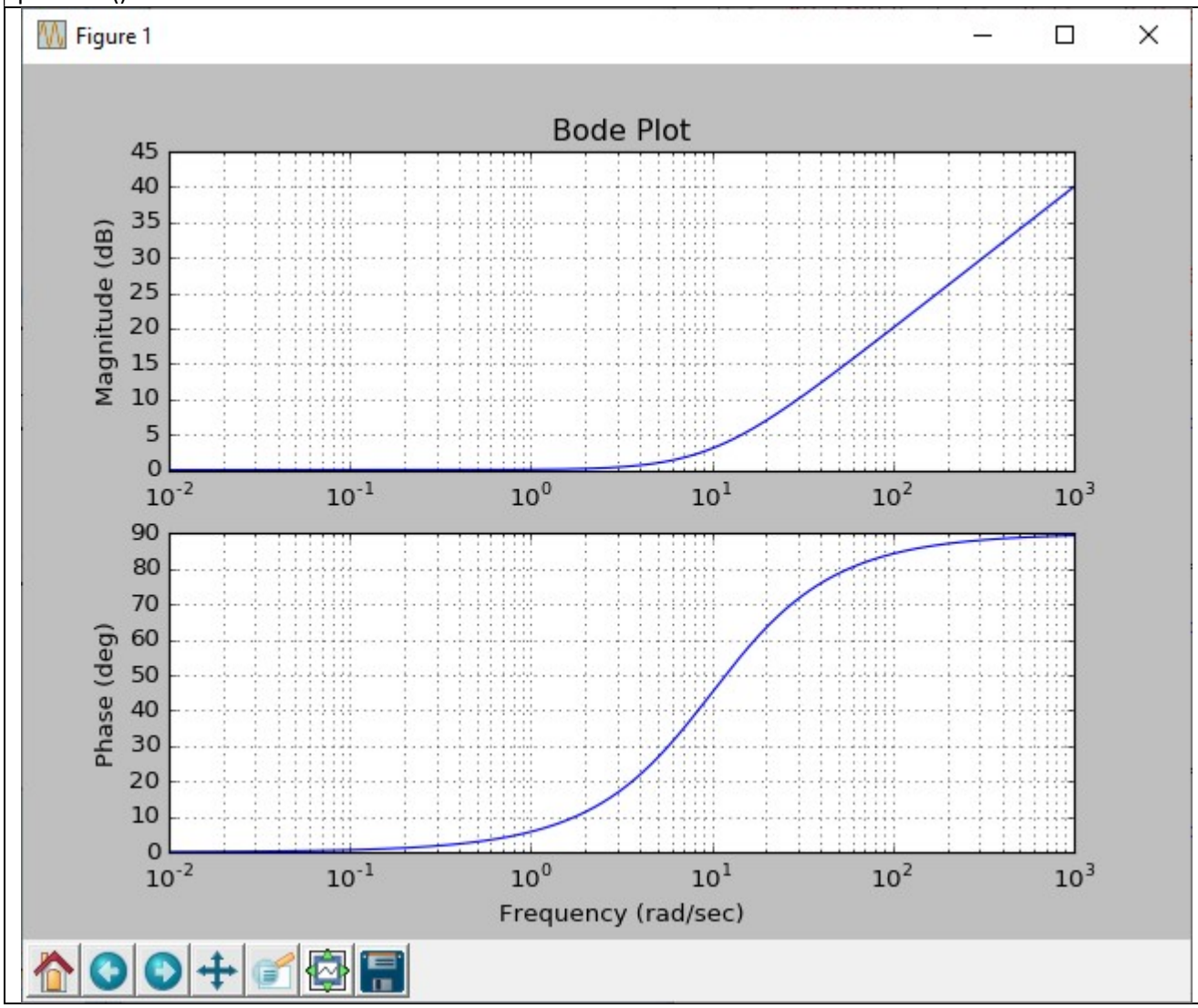


Oppure si puo' utilizzare l'esempio seguente sia per il modulo che per la fase:

Esempio di grafico del modulo e della fase della fdt:G[db]=20*Log(|1+0.1*jw|)

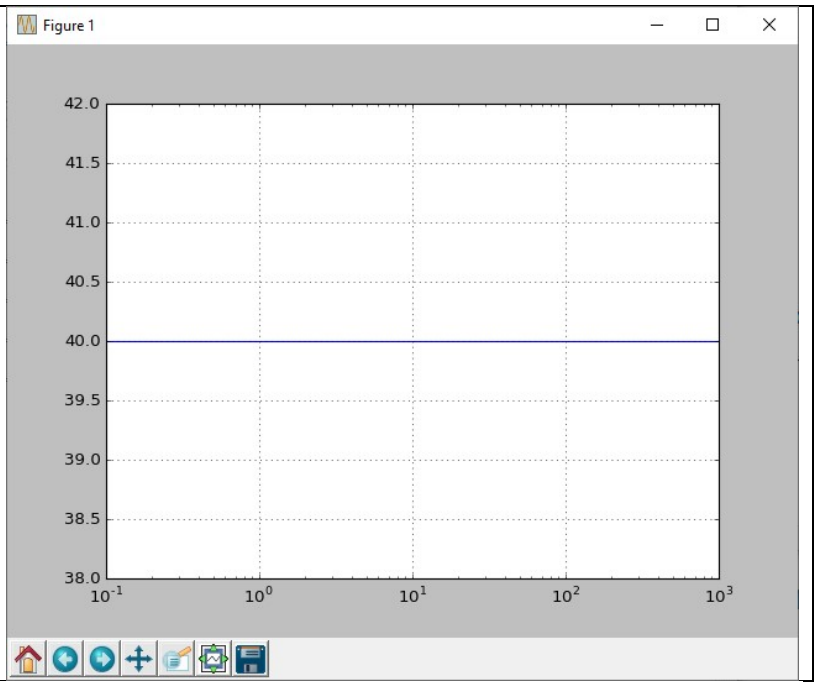
```
def calcWMP(G,w_start,w_stop,step=1./100):
    w_start=10**(w_start)
    w_stop=10**(w_stop)
    N = int ((w_stop-w_start )/step) + 1
    w = np.linspace (w_start , w_stop , N)
    w, mag, phase = signal.bode(G, w)
    return w,mag,phase
def plotModule(w,mag):
    plt.subplot (2, 1, 1)
    plt.semilogx(w, mag) # Bode Magnitude Plot
    plt.title("Module")
    plt.grid(b=None, which='major', axis='both')
    plt.grid(b=None, which='minor', axis='both')
    plt.ylabel("Magnitude (dB)")
def plotPhase(w,phase):
    plt.subplot (2, 1, 2)
    plt.semilogx(w, phase) # Bode Phase plot
    plt.title("Phase")
    plt.grid(b=None, which='major', axis='both')
    plt.grid(b=None, which='minor', axis='both')
    plt.ylabel("Phase (deg)")
    plt.xlabel("Frequency (rad/sec)")
num = np.array([0.1,1])
```

```
den = np.array([ 1])
G = signal.TransferFunction(num, den)
print ('G(s) =', G)
w,mag,phase=calcWMP(G,-2,3)
plt.figure('BODE Plot')
plotModule(w,mag)
plotPhase(w,phase)
plt.show()
```



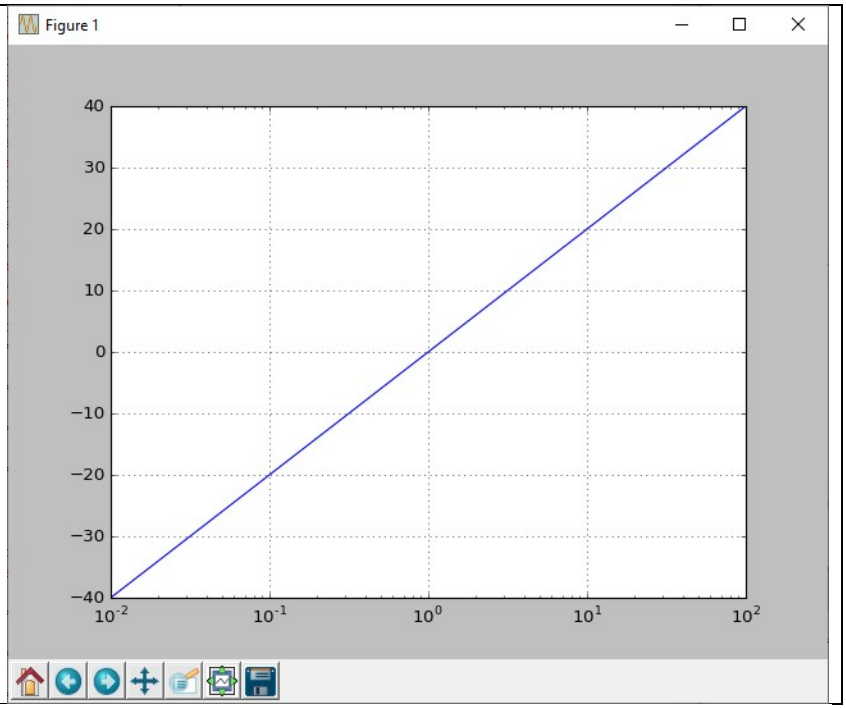
Esempio:
20Log|K|, K=100 =>20*Log(100)=40

```
def G(w):
    K=100
    G=K+0j * w
    return G
w = np.logspace(-1,3)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



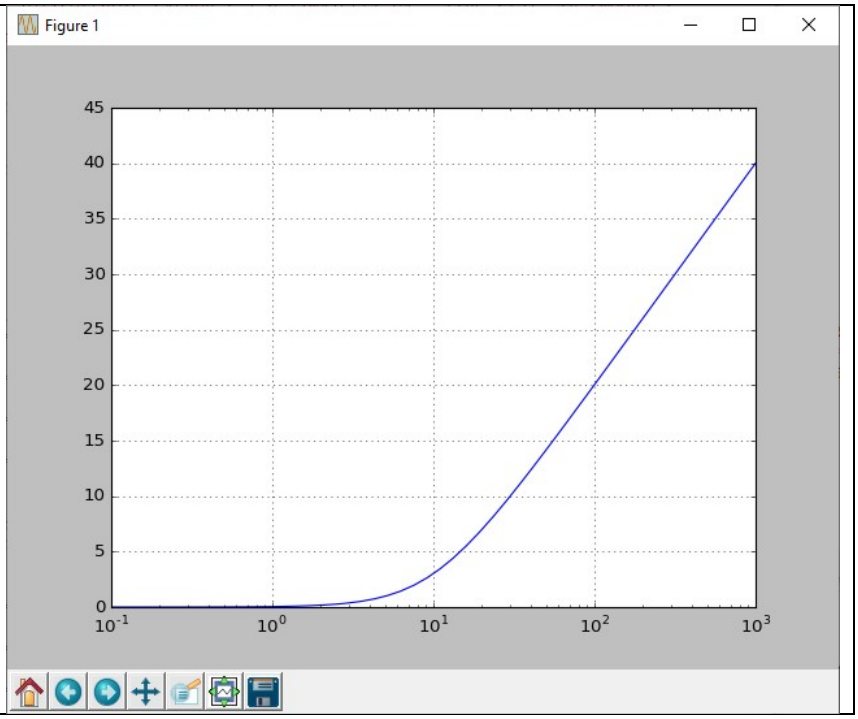
Esempio:20Log|jω|

```
def G(w):
    G=1j * w
    return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



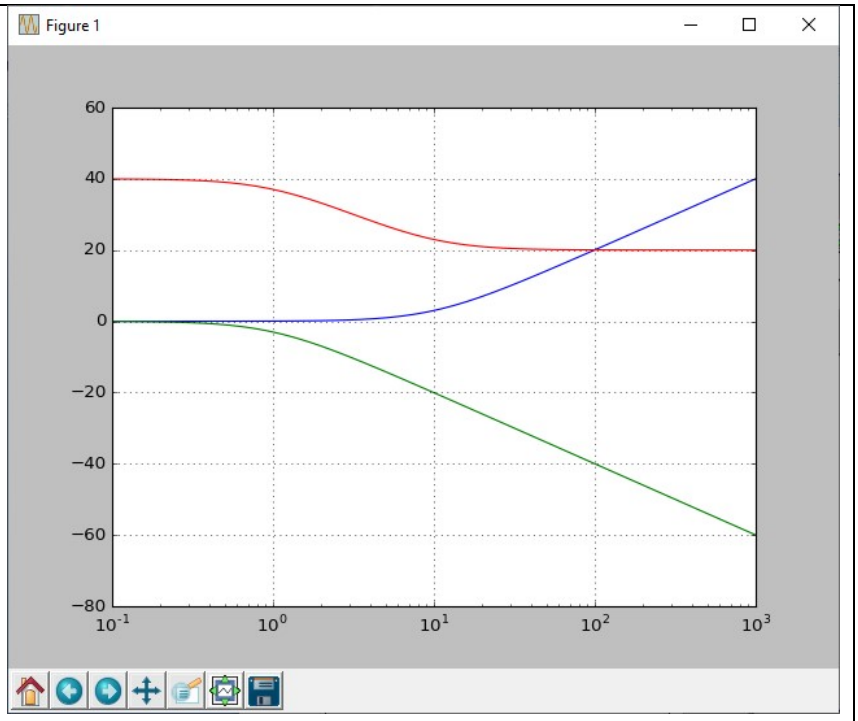
Esempio:20Log|1+jωT|

```
def G(w):
    T=0.1
    G=(1+1j*w*T)
    return G
w = np.logspace(-1,3)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



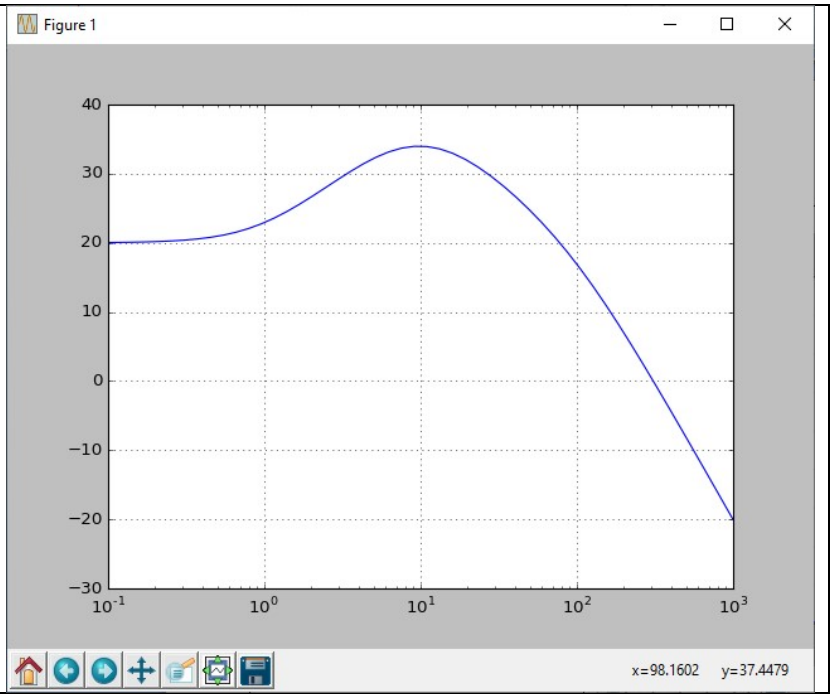
Esempio:
20Log|100*(1+jωT)/(1+jω)|

```
def G1(w):
    T=0.1
    G=(1+T*1j * w)
    return G
def G2(w):
    G=1./(1+1j * w)
    return G
def G(w):
    return 100*G1(w)*G2(w)
w = np.logspace(-1,3)
Gdb=20*np.log10(abs(G1(w)))
plt.plot(w, Gdb)
Gdb=20*np.log10(abs(G2(w)))
plt.plot(w, Gdb)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



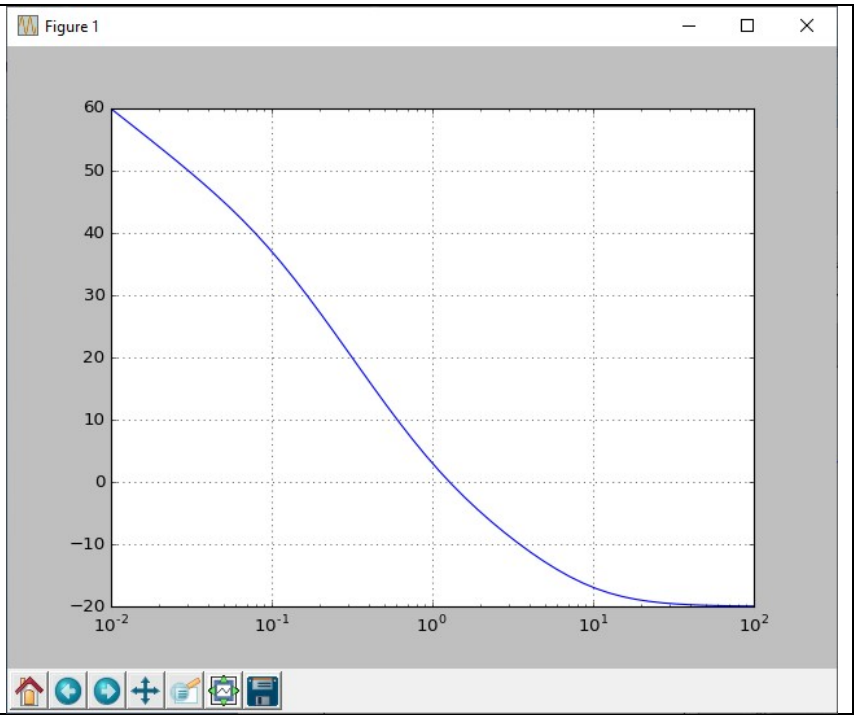
Esempio:
$10(1+j\omega)/(1+j\omega 10^{-1})^2(1+j\omega 10^{-2})$

```
def G (w):
   G=10.*((1+1j * w)/((1+0.1*1j *
w)**2*(1+1j * w*0.01)))
   return G
w = np.logspace(-1,3)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



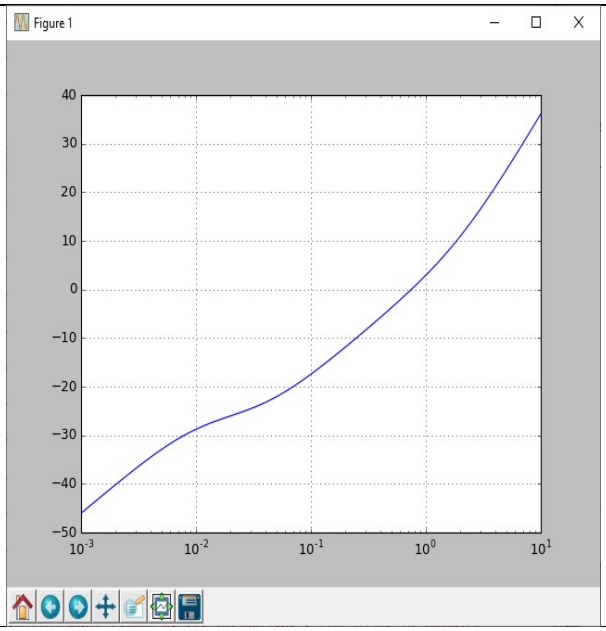
Esempio:
$10(1+j\omega)(1+j\omega 10^{-1})/(j\omega(1+j\omega 10))$

```
def G (w):
   G=(10.*(1+1j*w)*(1+1j*w*1E-
1))/((1j*w)*(1+1j*w*10))
   return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



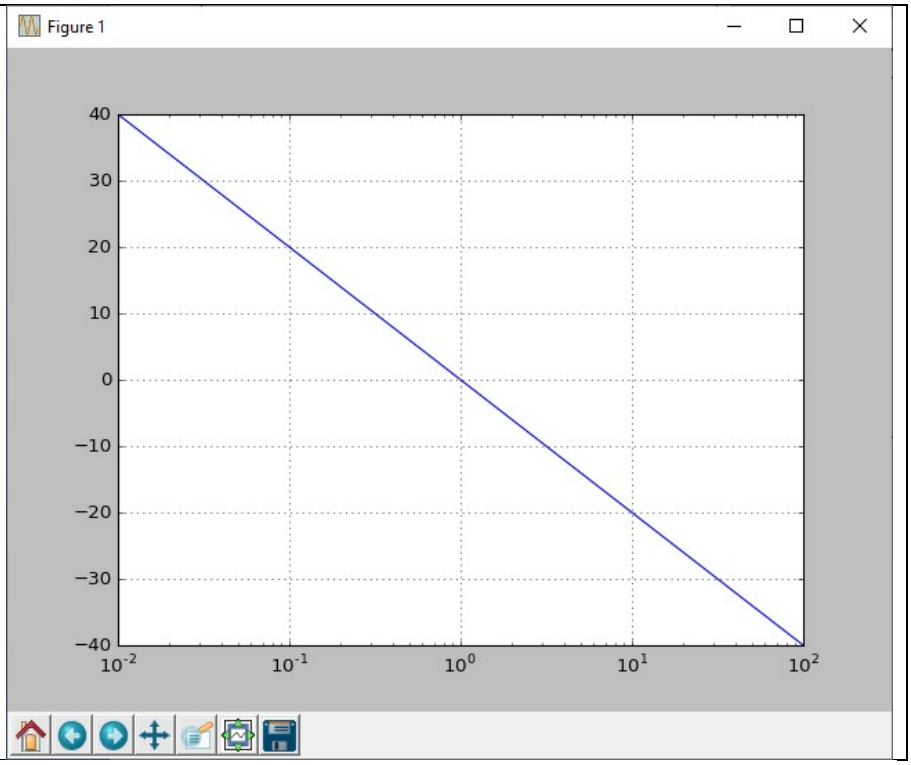
Esempio:
$5\ j\omega\ (1+0.25*10^2 j\omega)(1+0.5j\omega)/(1+j\omega 10^2)$

```
def G (w):        c
   G=(5*(1j*w)*(1+0.25*100*1j*w)*(1+0.5*1j*w))/(1
+1j*w*100)
   return G
w = np.logspace(-3,1)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



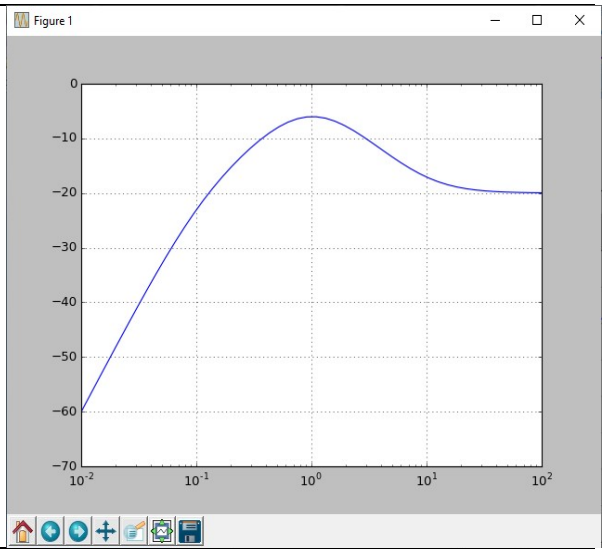
Esempio:
$1/\ j\omega$

```
def G (w):
   G=1/(1j * w)
   return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



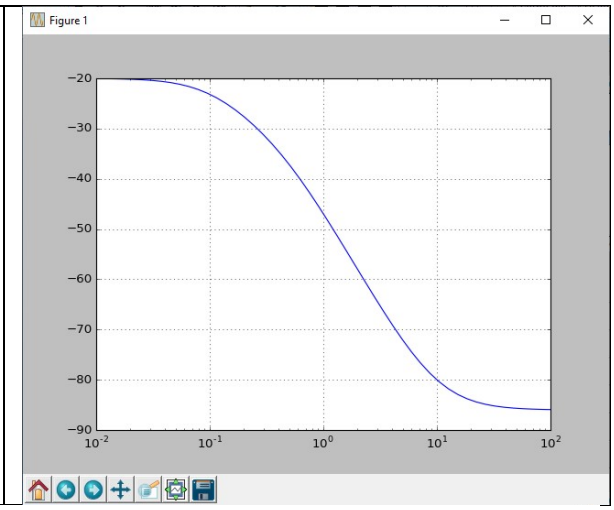
Esempio:
$10(j\omega)\ 2(1+j\omega 10^{-1})/((1+j\omega 10)(1+j\omega)^2)$

```
def G (w):
   G=10*(1j*w)**2*(1+1j*w*0.1)/((1+1j*w*10)*(1+1j*w)*
**2)
   return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



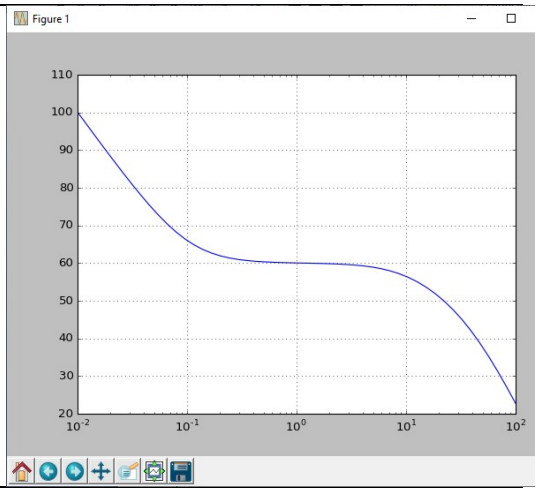
Esempio:
$0.1(1+j\omega 10^{-1})^2/((1+j\omega 2)(1+j\omega 10))$

```
def G (w):
   G=(0.1*(1+1j*w*0.1)**2)/((1+1j*w*2)*(1+1j*w*10))
   return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



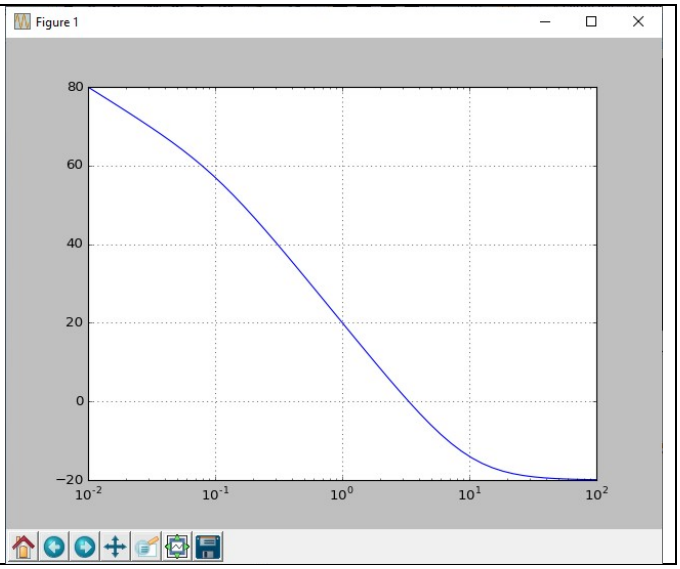
Esempio:
$10(1+j\omega 10)^2/((j\omega)^2 (1+j\omega)^2(1+j\omega 0.1)(1+j\omega 0.025)^2)$

```
def G (w):
   G=(10*(1+1j*w*10)**2)/((1j*w)**2*(1+1j*w*0.1)*(1+1j*w*0.
025)**2)
     return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



Esempio:
$(10+j\omega)^2/((j\omega) (1+j\omega))$

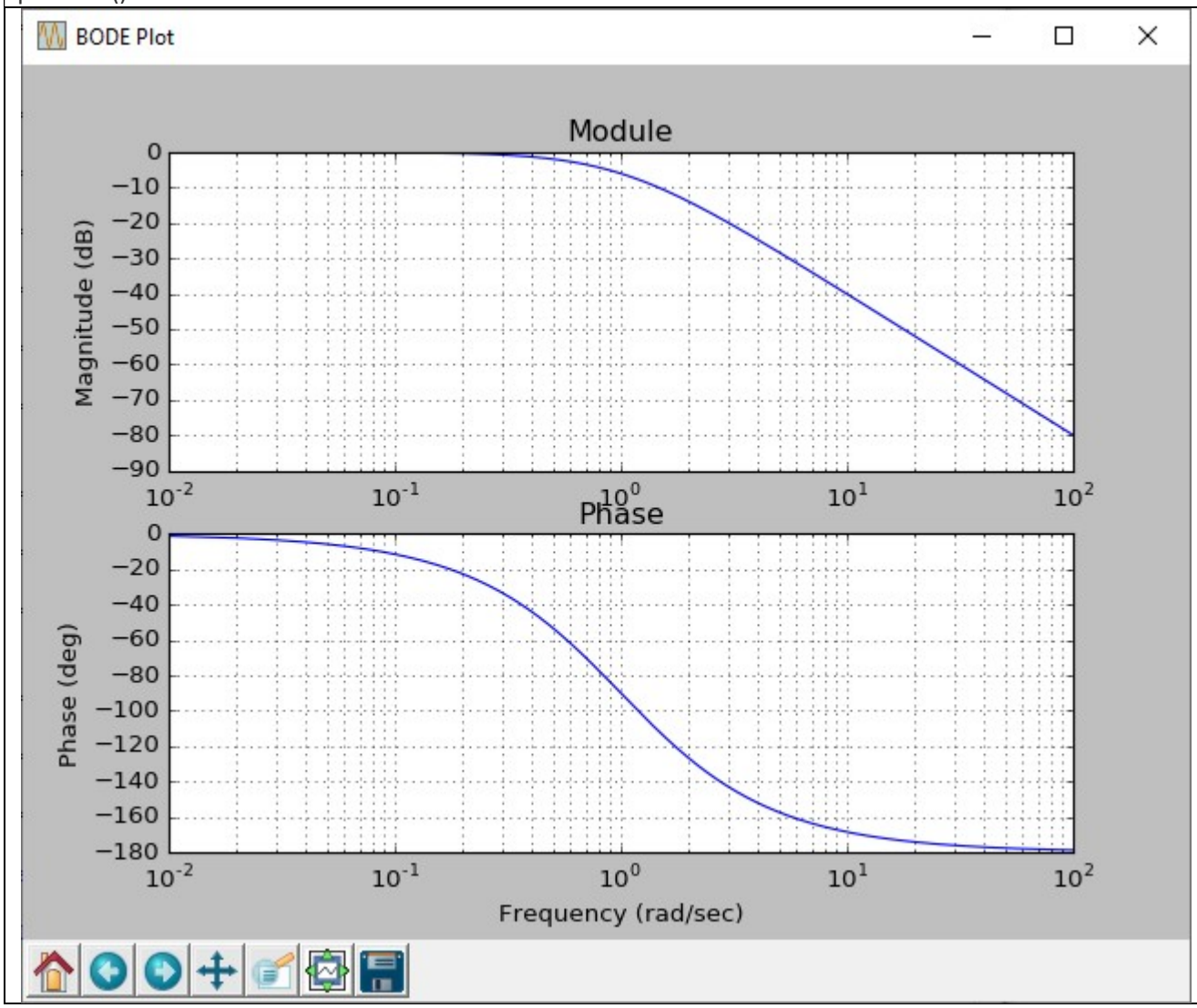
```
def G (w):
     G=(100*(1+1j*w*0.1)**2)/((1j*w)*(1+1j*w*10))
     return G
w = np.logspace(-2,2)
Gdb=20*np.log10(abs(G(w)))
plt.plot(w, Gdb)
plt.xscale('log')
plt.grid()
plt.show()
```



Esempio: modulo e fase di: $1/(1+2*j\omega + j\omega^2)$

```
def calcWMP(G,w_start,w_stop,step=1./100):
   w_start=10**(w_start)
   w_stop=10**(w_stop)
   N = int ((w_stop-w_start )/step) + 1
   w = np.linspace (w_start , w_stop , N)
   w, mag, phase = signal.bode(G, w)
   return w,mag,phase
def plotModule(w,mag):
   plt.subplot (2, 1, 1)
```

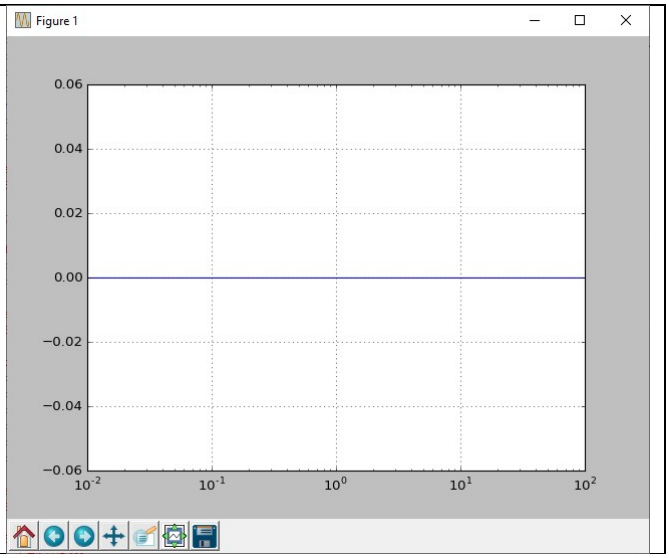
```
    plt.semilogx(w, mag) # Bode Magnitude Plot
    plt.title("Module")
    plt.grid(b=None, which='major', axis='both')
    plt.grid(b=None, which='minor', axis='both')
    plt.ylabel("Magnitude (dB)")
def plotPhase(w,phase):
    plt.subplot (2, 1, 2)
    plt.semilogx(w, phase) # Bode Phase plot
    plt.title("Phase")
    plt.grid(b=None, which='major', axis='both')
    plt.grid(b=None, which='minor', axis='both')
    plt.ylabel("Phase (deg)")
    plt.xlabel("Frequency (rad/sec)")
num = np.array([1])
den = np.array([1,2,1])
G = signal.TransferFunction(num, den)
print ('G(s) =', G)
w,mag,phase=calcWMP(G,-2,2)
plt.figure('BODE Plot')
plotModule(w,mag)
plotPhase(w,phase)
plt.show()
```



BODE PHASE

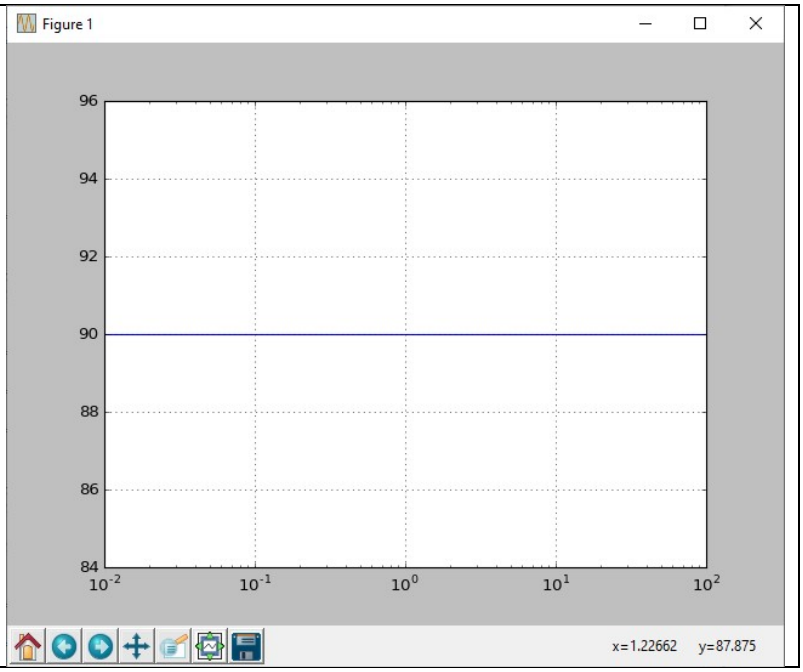
Esempio:
G(jw)=K, K=10

```
def G(w):
    K=10
    G=K+0j*w
    return G
w = np.logspace(-2,2)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



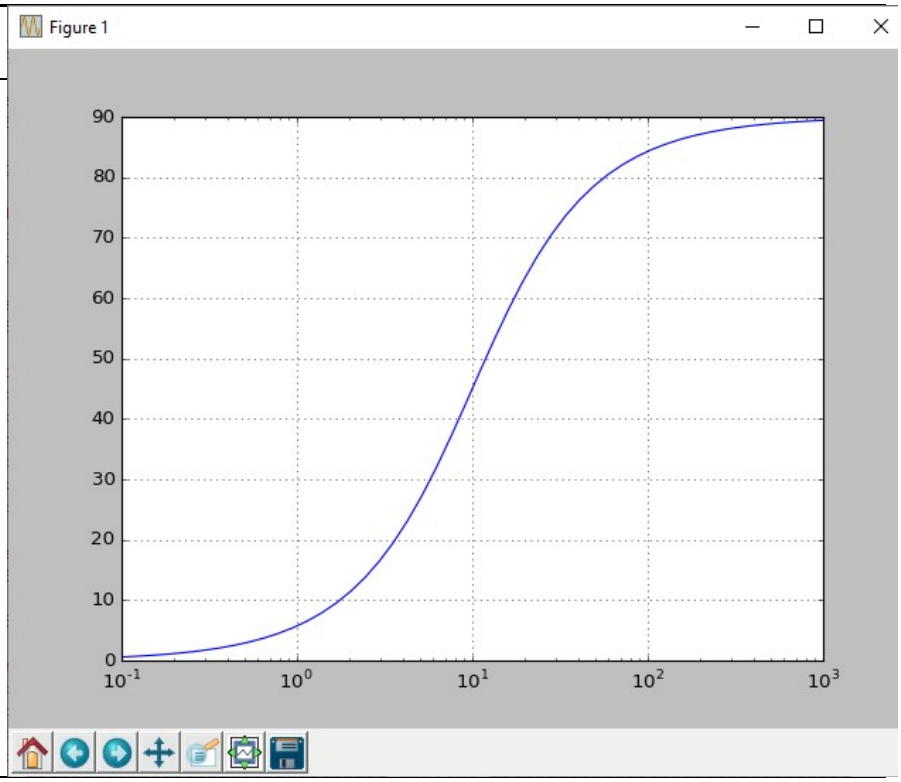
Esempio:
G(jw)=jw

```
def G(w):
    G=1j*w
    return G
w = np.logspace(-2,2)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



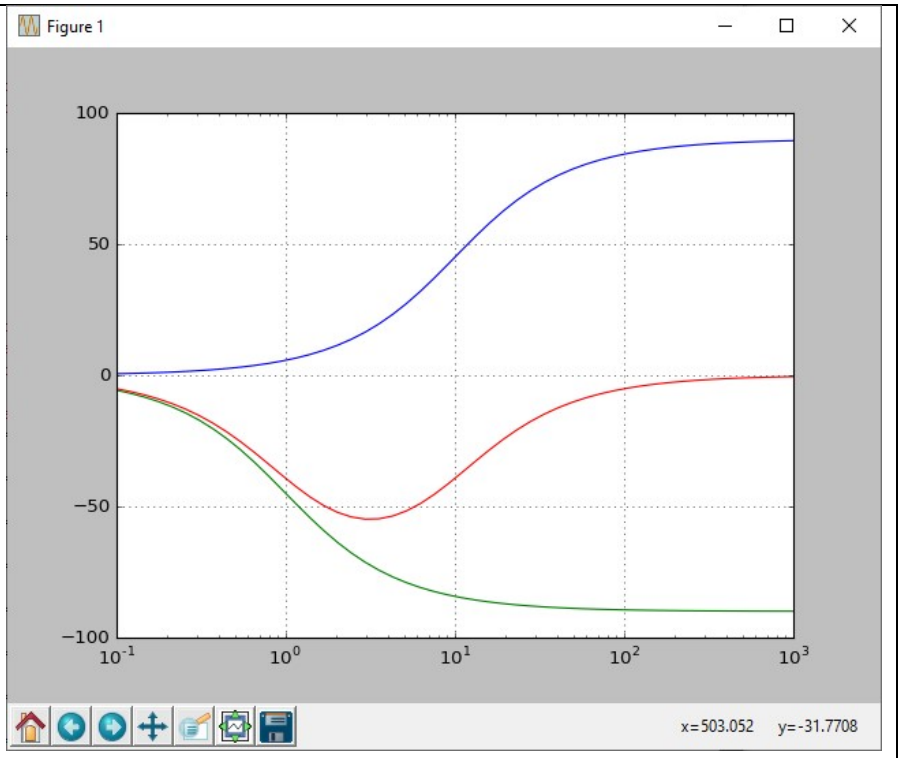
Esempio:
G(jw)=1+jwT

```
def G(w):
    T=0.1
    G=1+1j*w*T
    return G
w = np.logspace(-1,3)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



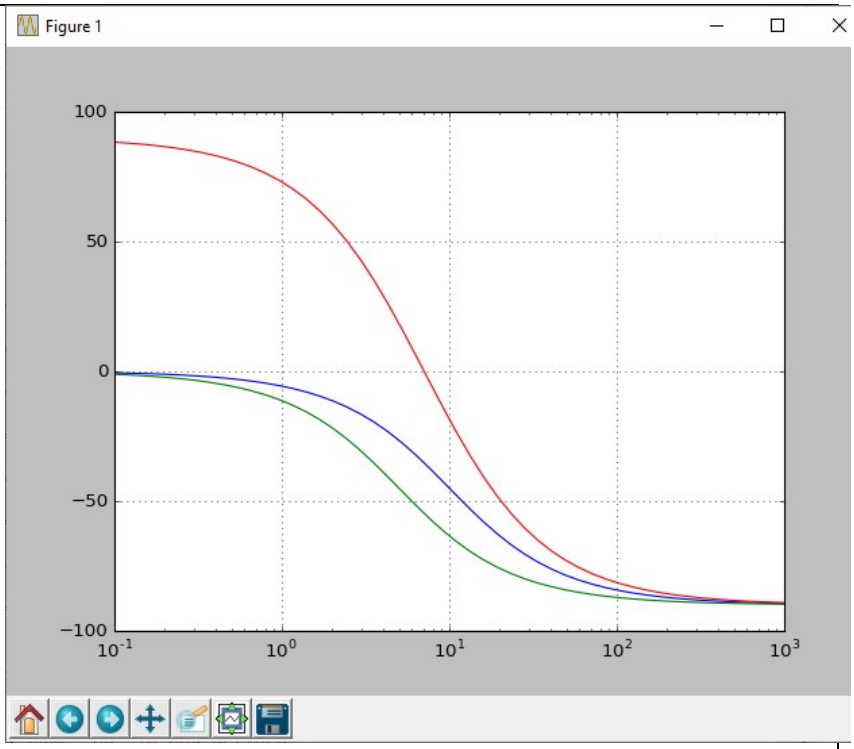
Esempio:
G(jw)=1+jwT

```
def G1(w):
    T=0.1
    G=(1+T*1j * w)
    return G
def G2(w):
    G=1./(1+1j * w)
    return G
def G(w):
    G=100*G1(w)*G2(w)
    return G
w = np.logspace(-1,3)
phase =
np.angle(G1(w),deg=True)
plt.plot(w, phase)
phase =
np.angle(G2(w),deg=True)
plt.plot(w, phase)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



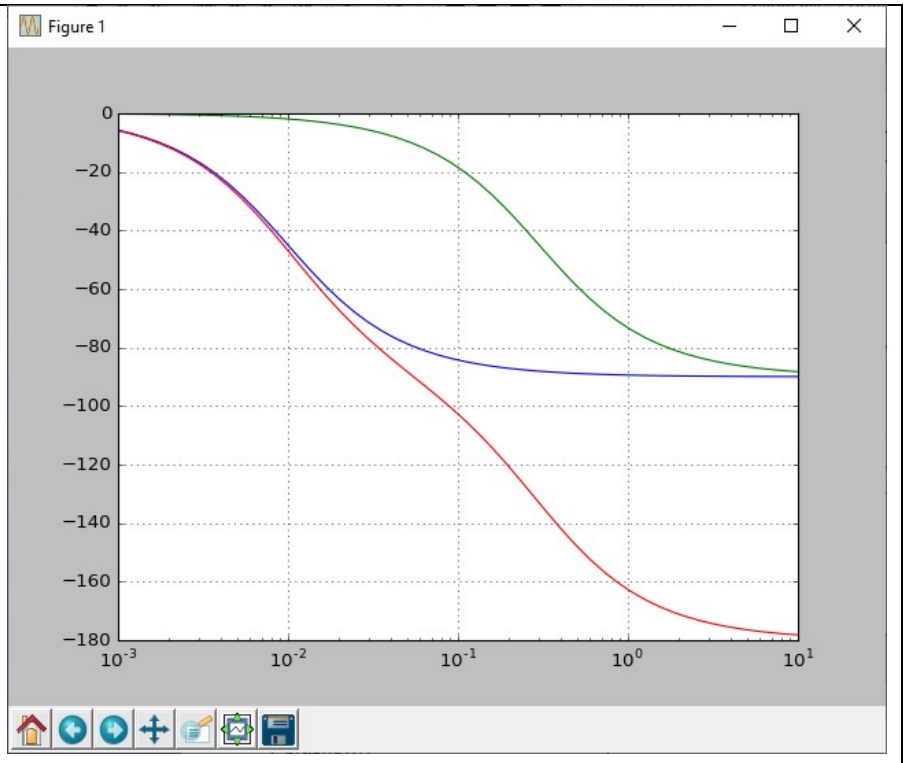
Esempio:
G(jw)= (1jw)/((1+1j w0.1)(1+1j w0.2))

```
def G1(w):
   G=1./(1+1j * w*0.1)
   return G
def G2(w):
   G=1./(1+1j * w*0.2)
   return G
def G(w):
   G=(1j * w)*G1(w)*G2(w)
   return G
w = np.logspace(-1,3)
phase = np.angle(G1(w),deg=True)
plt.plot(w, phase)
phase = np.angle(G2(w),deg=True)
plt.plot(w, phase)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



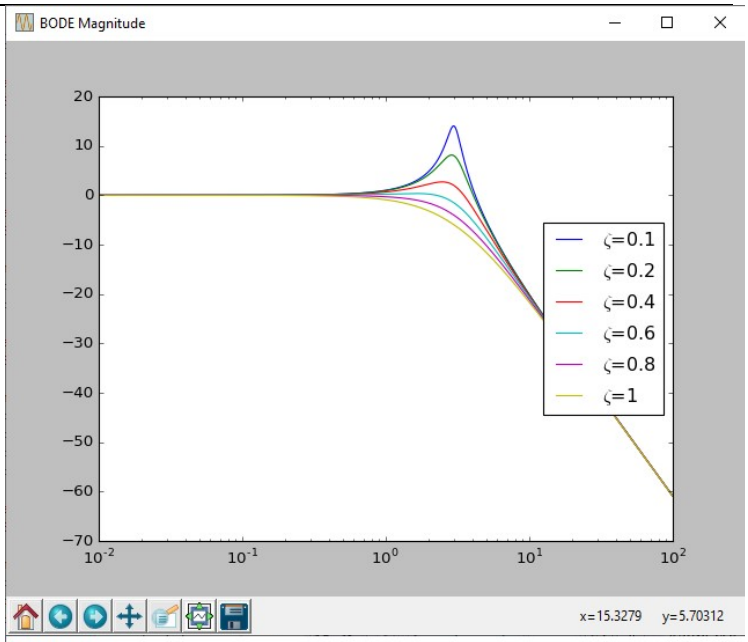
Esempio:
G(jw)= 1./((1+1j w10^{2}) (1+1j w3.33))

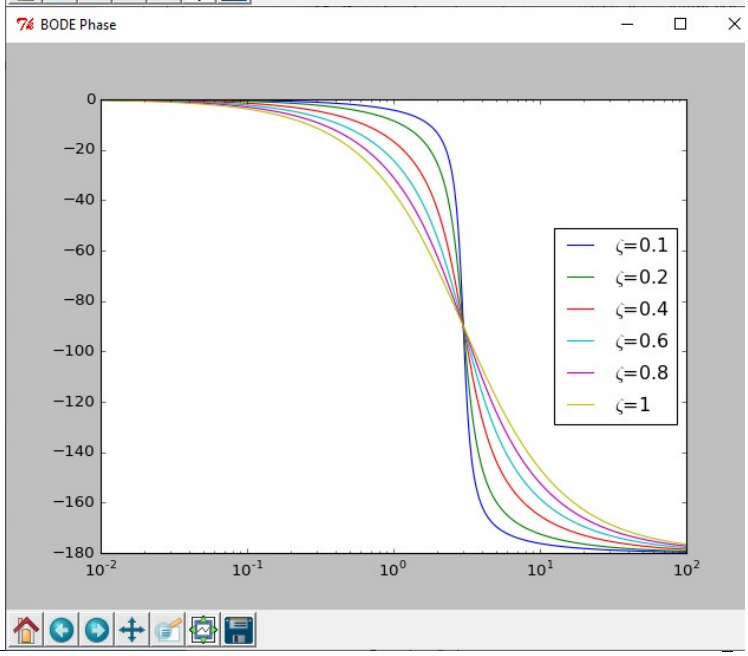
```
def G1(w):
   G=1./(1+1j * w*10**2)
   return G
def G2(w):
   G=1./(1+1j * w*3.33)
   return G
def G(w):
   G=G1(w)*G2(w)
   return G
w = np.logspace(-3,1)
phase =
np.angle(G1(w),deg=True)
plt.plot(w, phase)
phase =
np.angle(G2(w),deg=True)
plt.plot(w, phase)
phase = np.angle(G(w),deg=True)
plt.plot(w, phase)
plt.xscale('log')
plt.grid()
plt.show()
```



Esempio:
G(jw)= $w_n^2/(jw^2+2\xi w_n jw+wn^2)$
w_n=3 ξ=[0.1, 0.2, 0.4, 0.6, 0.8, 1]

```
def G(w,csi):
    wn=3
    G=(wn**2)/((1j*w)**2+2*(1j*w)*csi*wn+(w
n**2))
    return G
csi=[0.1,0.2,0.4,0.6,0.8,1]
w = np.logspace(-2,2,1000)
plt.gcf().canvas.set_window_title('BODE
Magnitude')
for c in range(len(csi)):
    Gdb=20*np.log10(abs(G(w,csi[c])))
    plt.grid()
    plt.xscale('log')
    plt.plot(w, Gdb,label=r'$\zeta$='+str(csi[c]))
    plt.legend(loc="center right")
plt.figure("BODE Phase")
for c in range(len(csi)):
    phase = np.angle(G(w,csi[c]),deg=True)
    plt.plot(w, phase,label=r'$\zeta$='+str(csi[c]))
    plt.legend(loc="center right")
    plt.xscale('log')
    plt.grid()

plt.show()
```





CAMPIONAMENTO

Esempio Sample & Hold

```
import numpy as np

def interp0(x, xp, yp):
    """Zeroth order hold interpolation w/ same
    (base)   signature  as numpy.interp."""

    def func(x0):
        if x0 <= xp[0]:
            return yp[0]
        if x0 >= xp[-1]:
            return yp[-1]
        k = 0
        while x0 > xp[k]:
            k += 1
        return yp[k-1]

    if isinstance(x,float):
```

```
        return func(x)
    elif isinstance(x, list):
        return [func(x) for x in x]
    elif isinstance(x, np.ndarray):
        return np.asarray([func(x) for x in x])
    else:
        raise TypeError('argument must be float, list, or ndarray')

import matplotlib.pyplot as plt
import numpy as np

# choose a function
f = np.sin

# sampled signal
xp = np.linspace(0,10,20)
yp = f(xp)

# interpolation grid with 'true' function
x = np.linspace(0,12,1000)
plt.plot(x,f(x),'--')

# plot
plt.hold(True)
plt.scatter(xp,yp)
plt.plot(x,interp0(x,xp,yp),'r')
plt.xlim([x.min(),x.max()])
plt.title('Zero Order Hold/Interpolation')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend(['Signal','ZOH'])
plt.show()
```

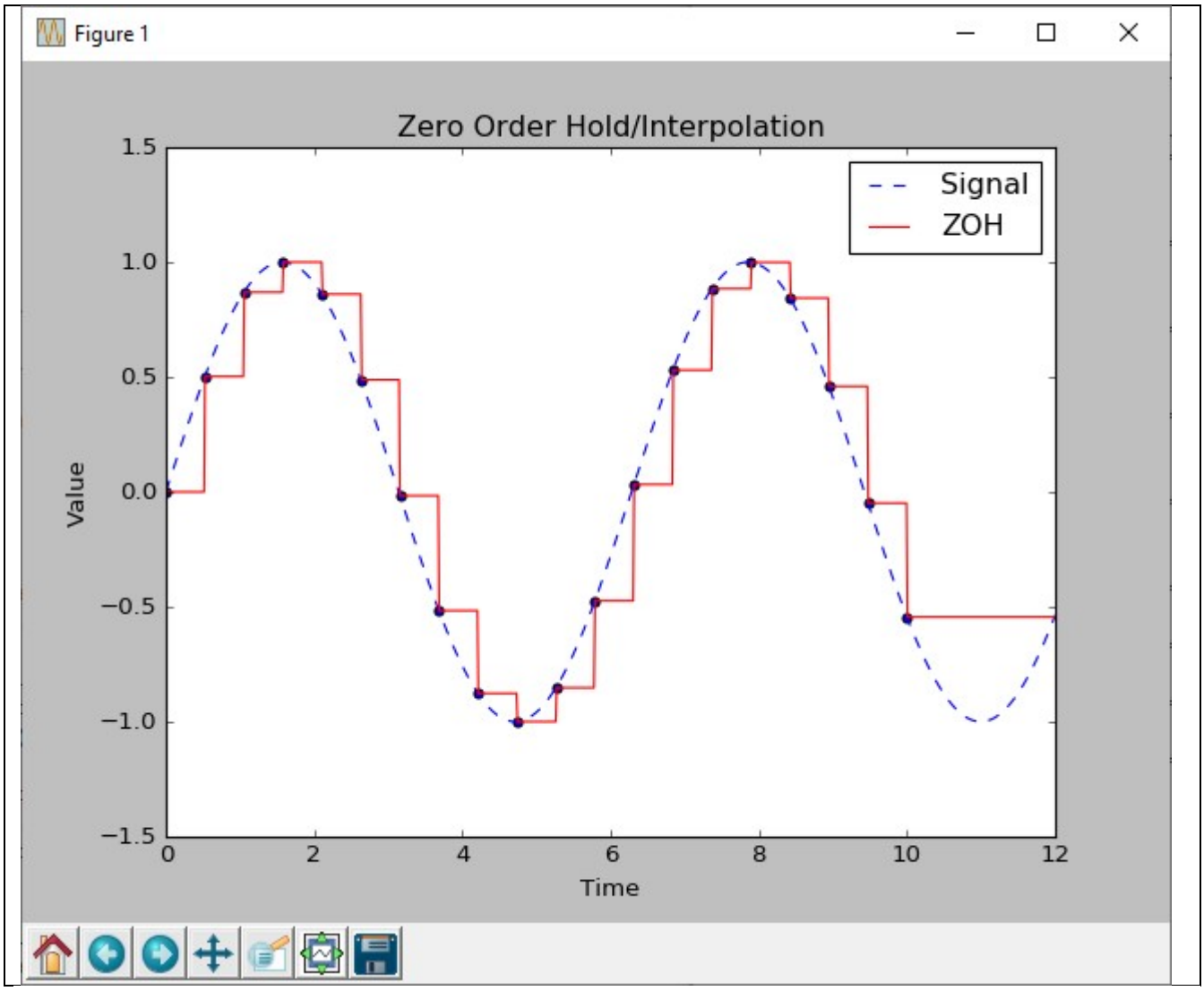


DIAGRAMMA DI NYQUIST

```
#G(Jw)=_____10____
        (1+jw)*(5+jw)

s = control.tf('s')
G = (10)/((1+s)*(5+s))
print G
real, imag, freq =control.nyquist_plot(G)
print real
print imag
print freq
plt.show()
```

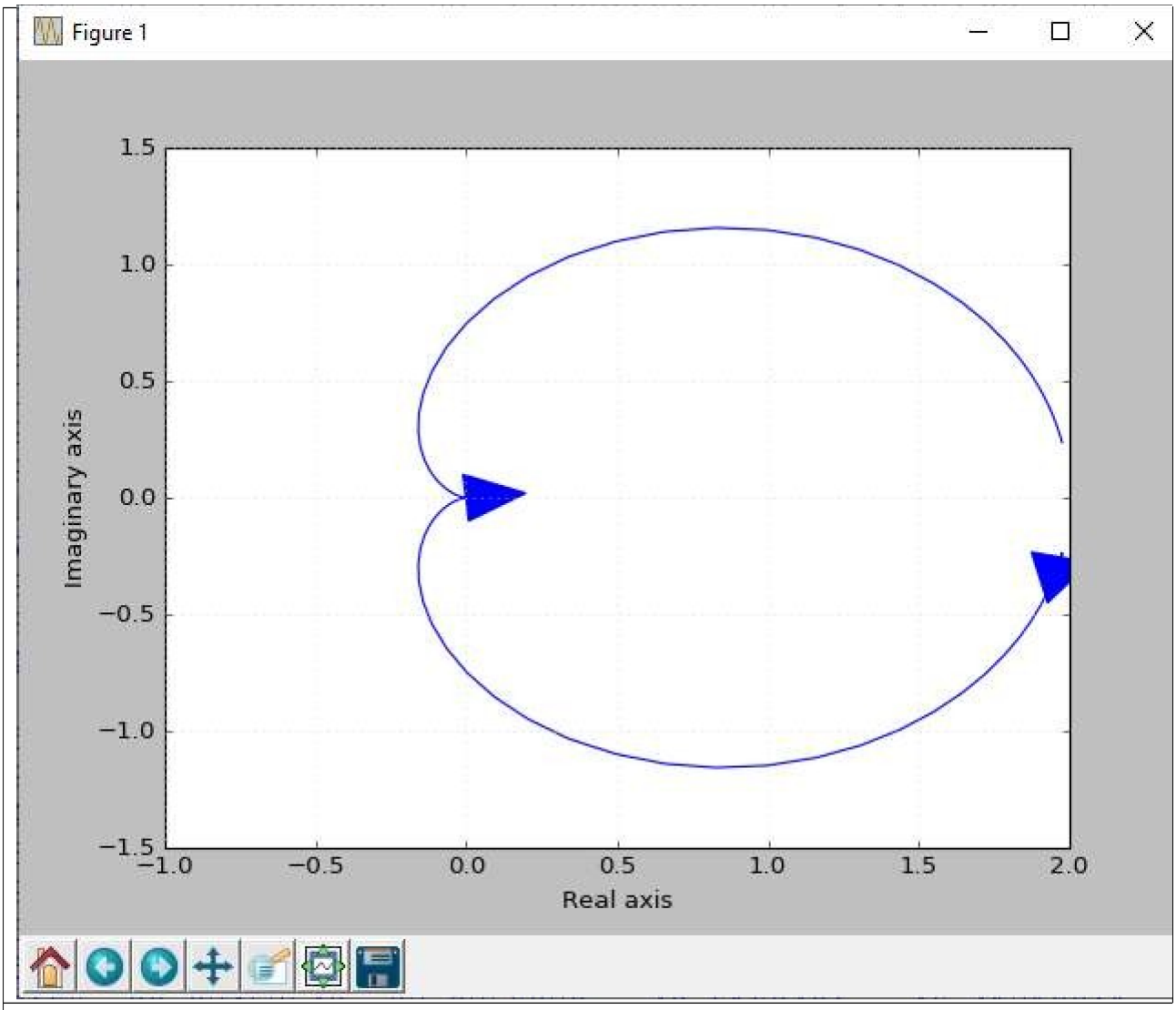


Figure 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Real axis

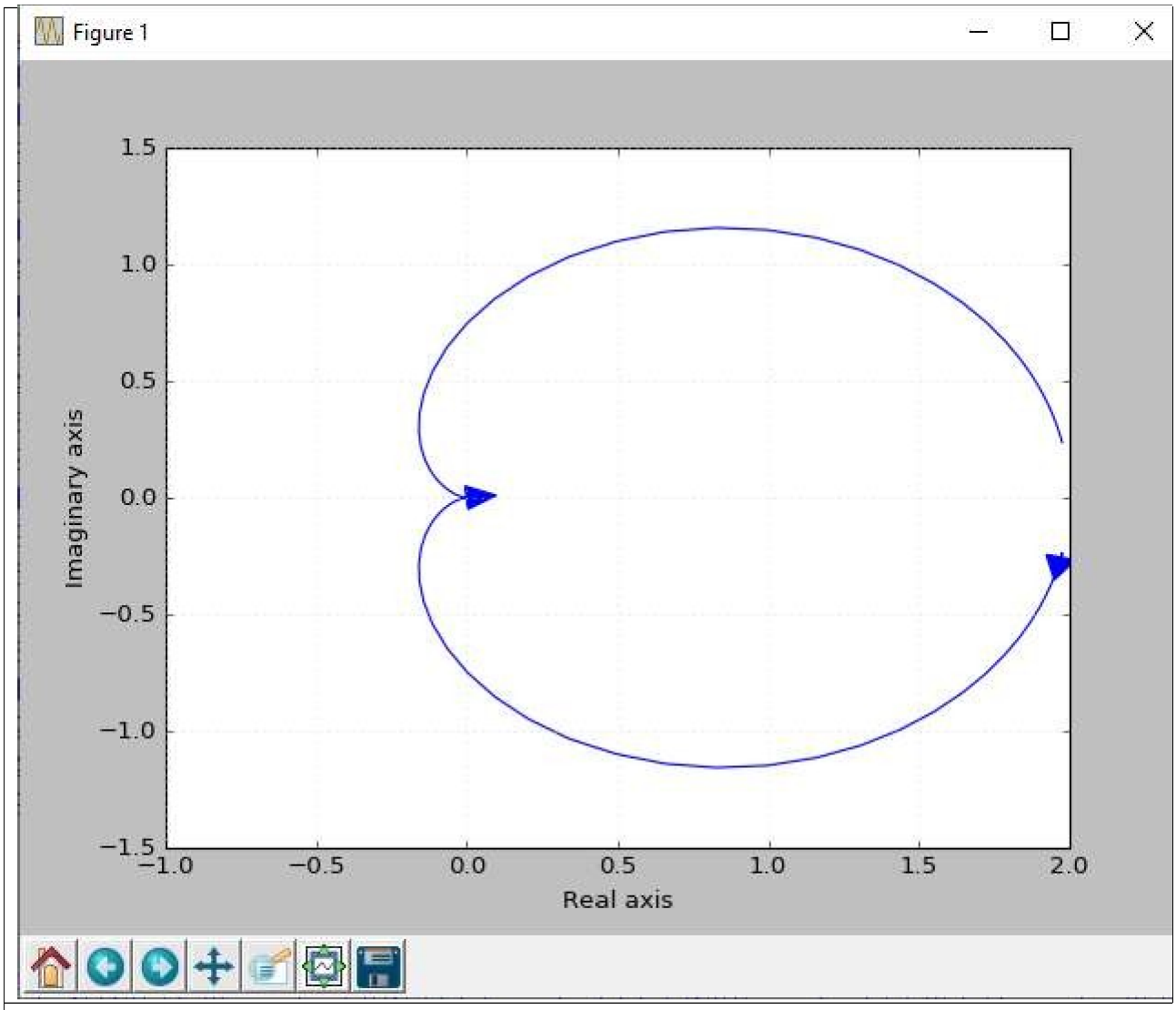


Figure 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Real axis

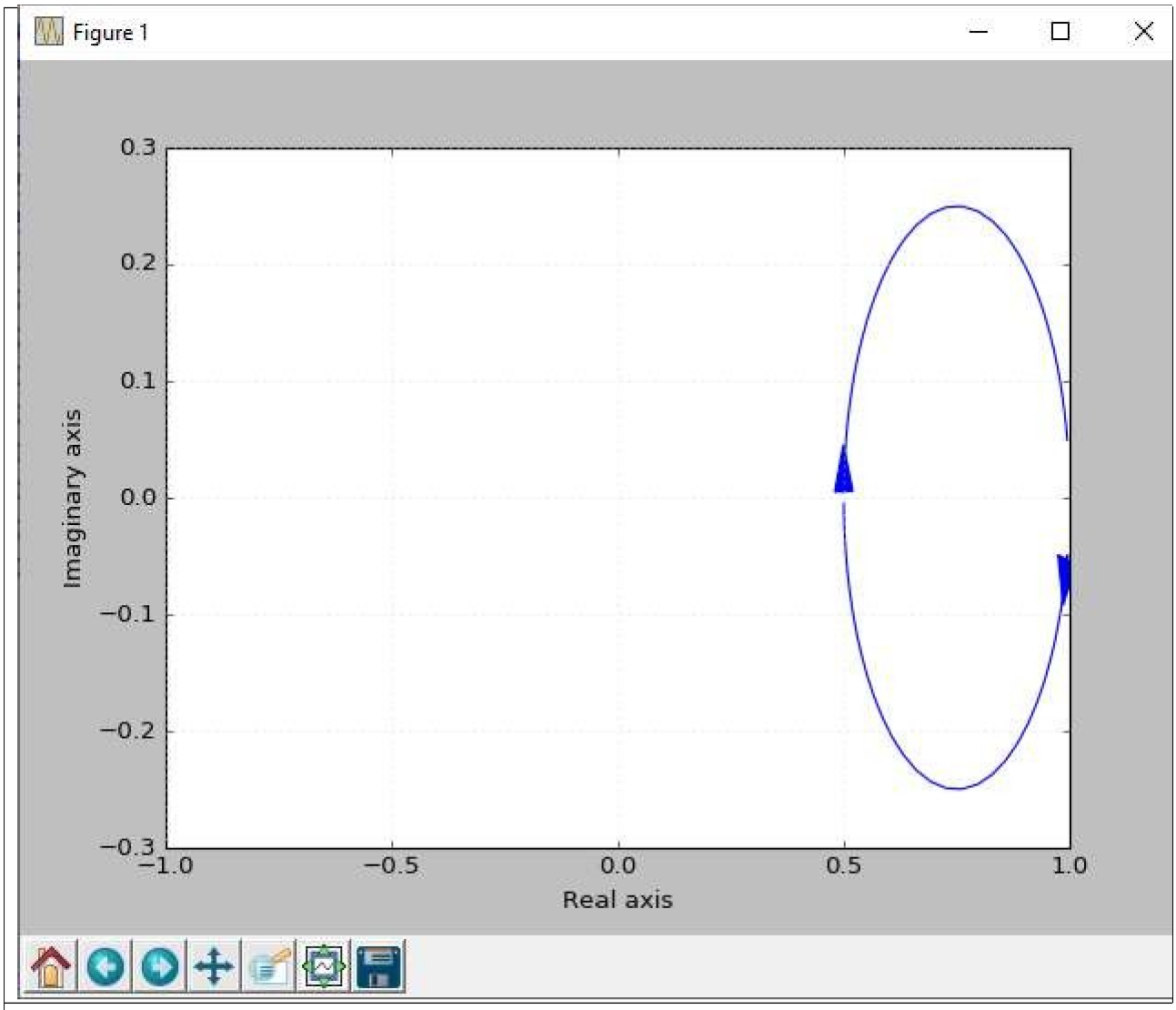


Figure 1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
Real axis

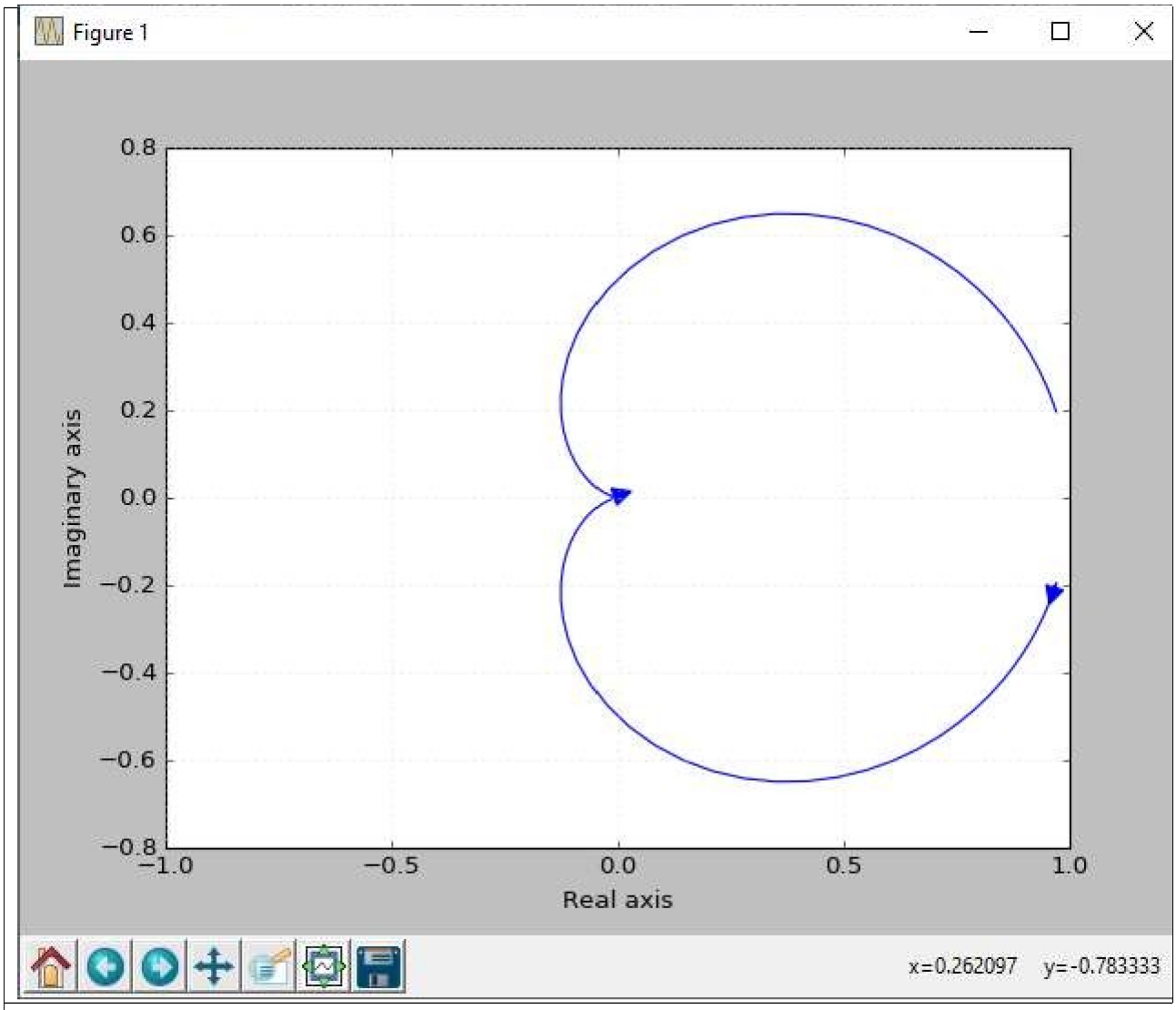


Figure 1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
Real axis
x=0.262097
y=-0.783333

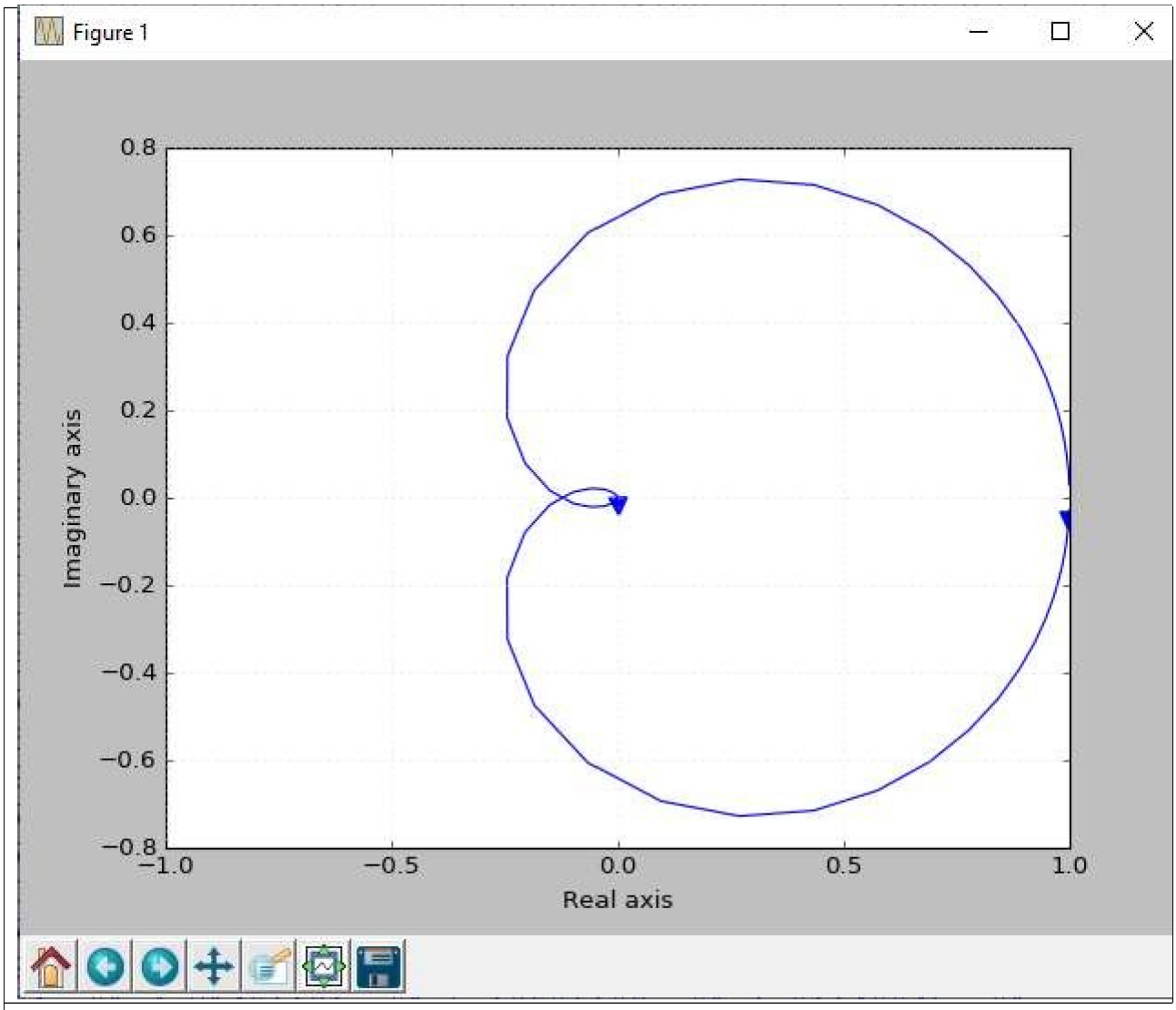


Figure 1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
Real axis

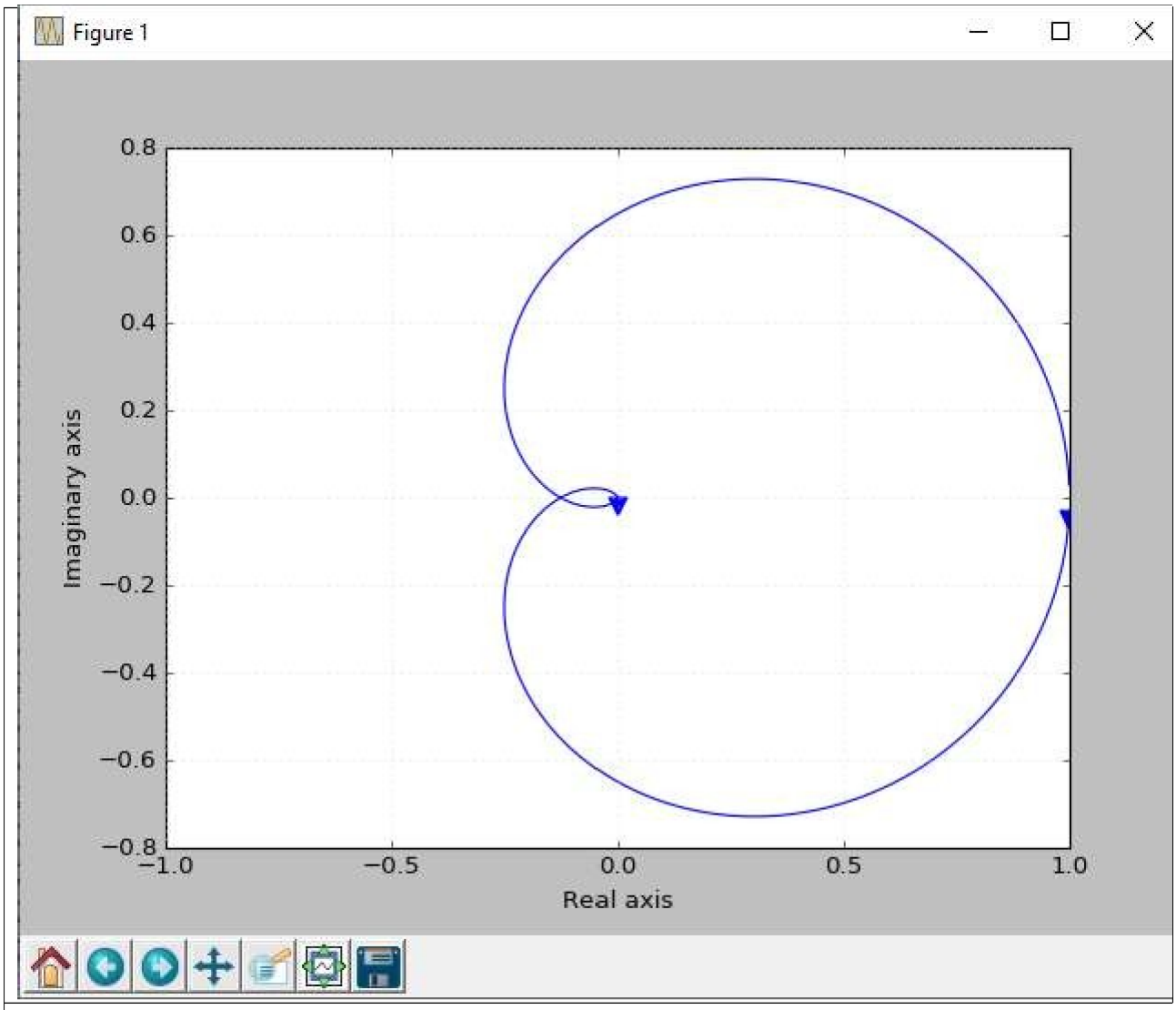


Figure 1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
Real axis

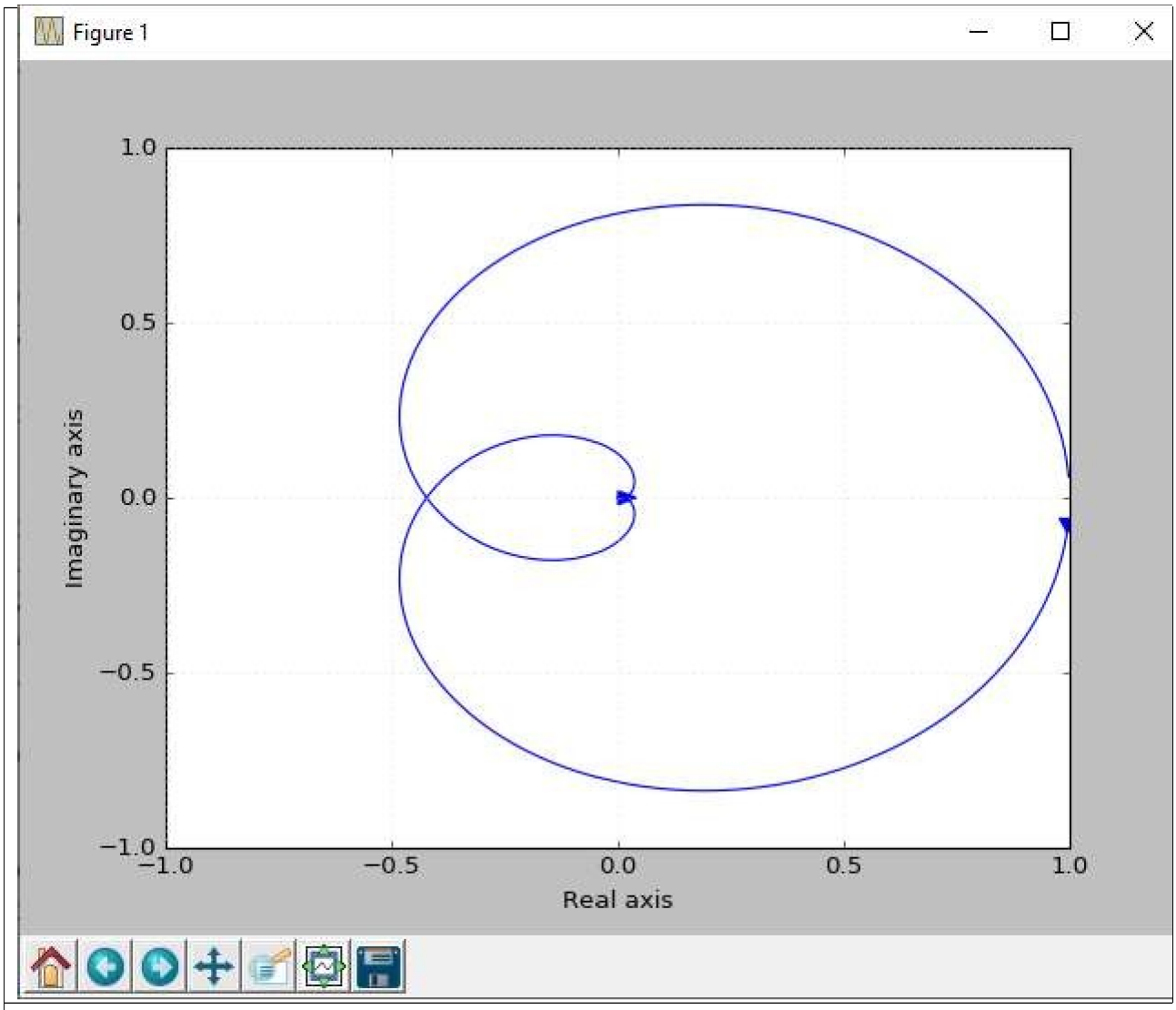
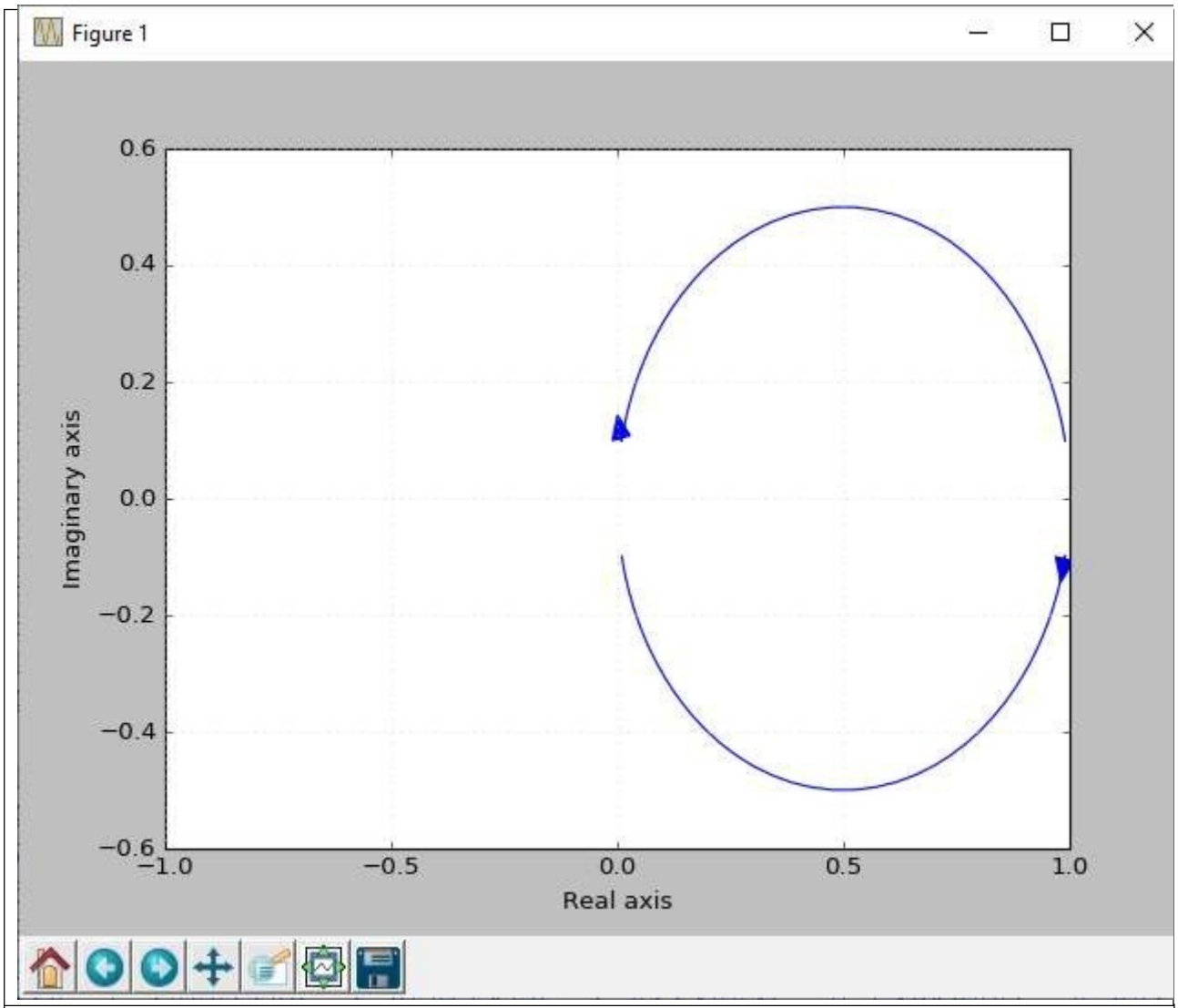
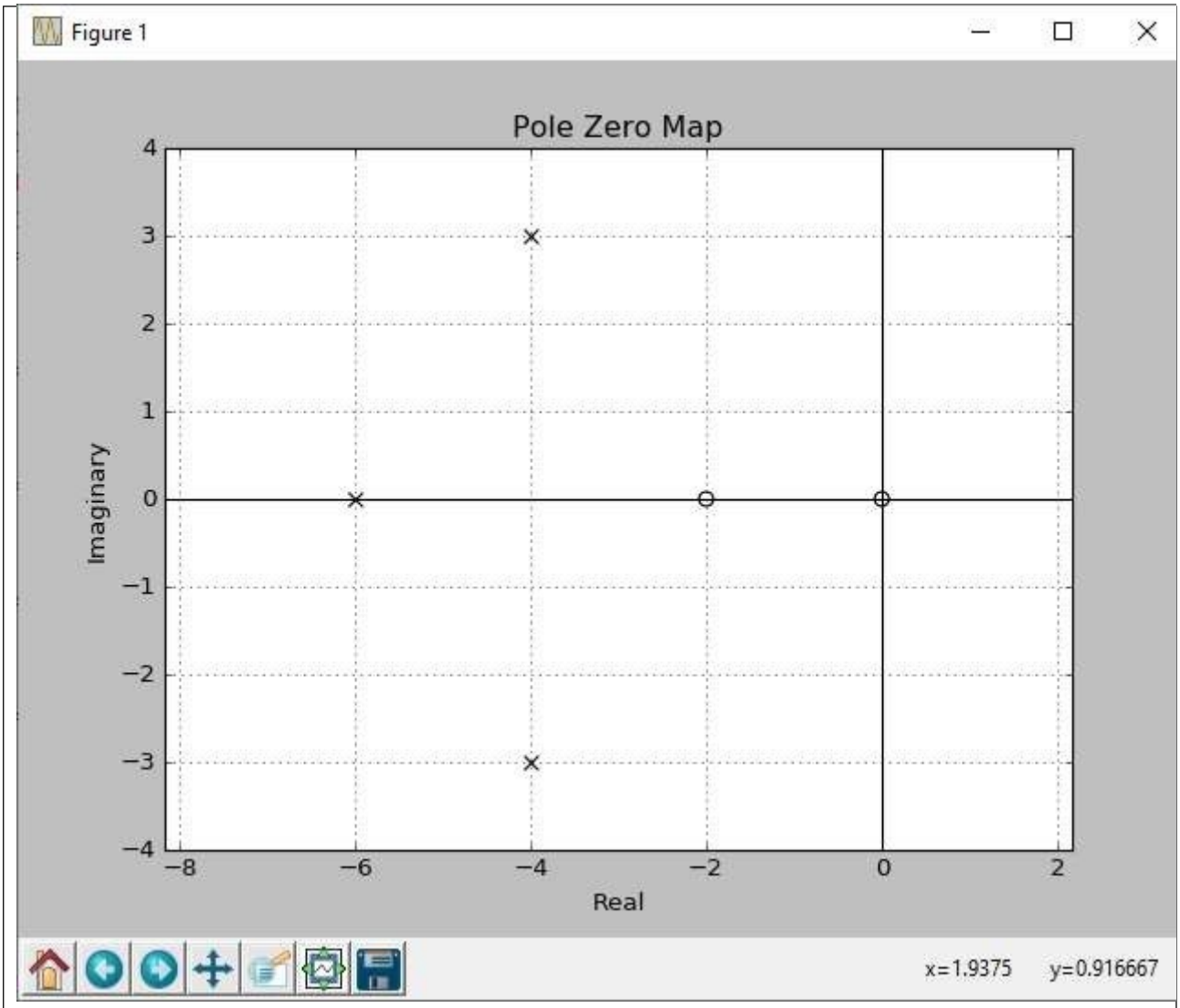


Figure 1
1.0
0.5
0.0
−0.5
−1.0
Imaginary axis
−1.0
−0.5
0.0
0.5
1.0
Real axis

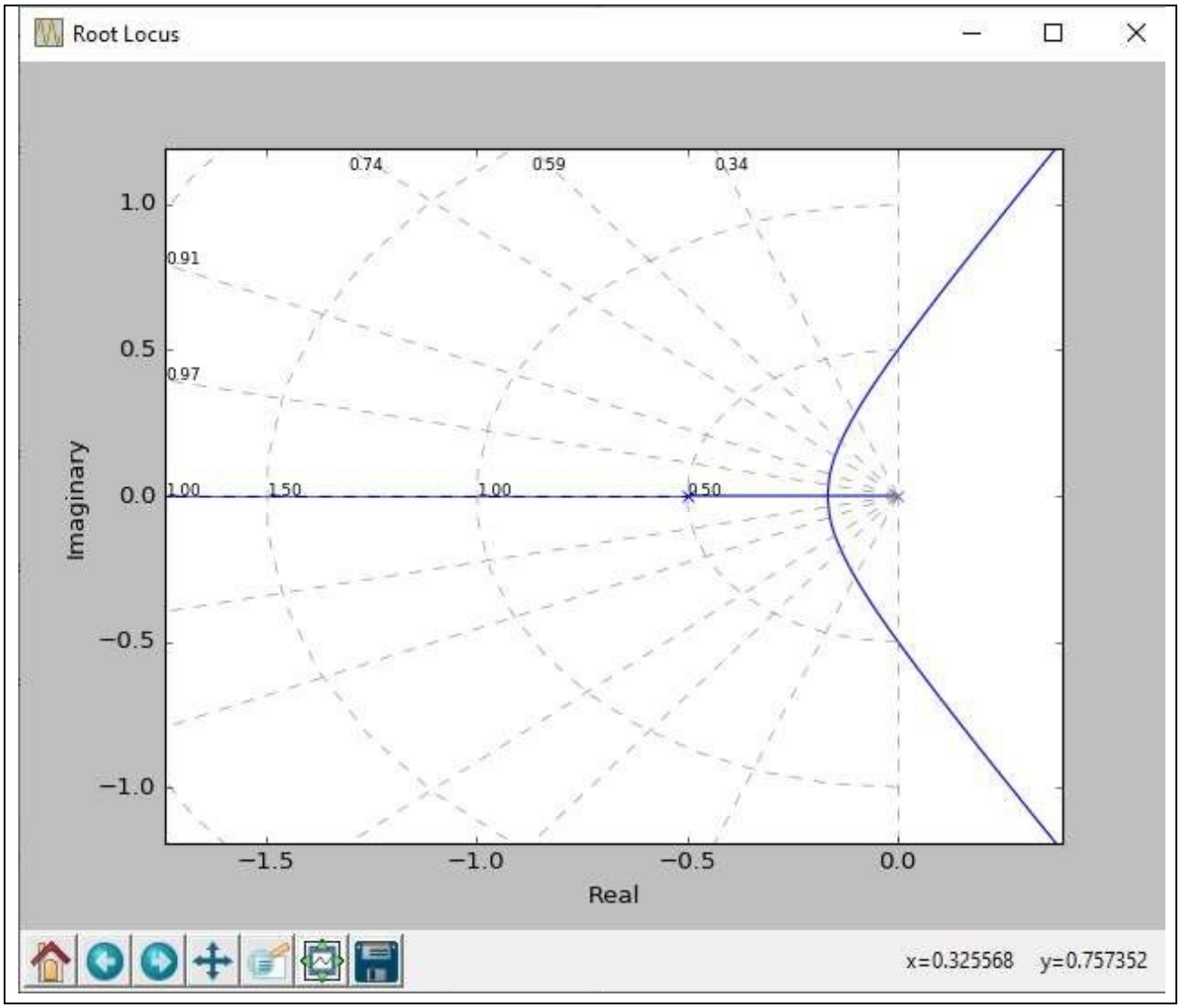



```
#                    s(s+2)
#G(s)=-----------------------------------
#             (s+6)(s^2+8s+25)

s = control.tf('s')
G = (s*(s+2))/((s+6)*(s**2+8*s+25))
pz = control.pzmap(G)
plt.grid()
```



ROOT LOCUS

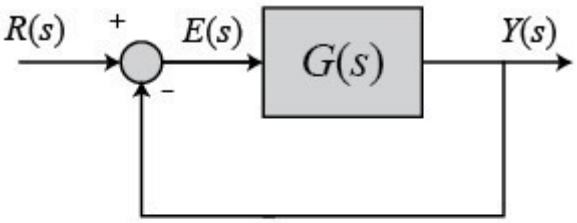
```
#              K
#G(s)=----------------------
#         4s^3+4s^2+s

from control import *
from matplotlib import pyplot as plt
sy1 = tf([1],[4,4,1,0])
rlocus(sy1)
plt.show()
```



STEADY-STATE

https://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_ess3

Vediamo la risposta ad anello chiuso per questo sistema quando utilizziamo input diversi.

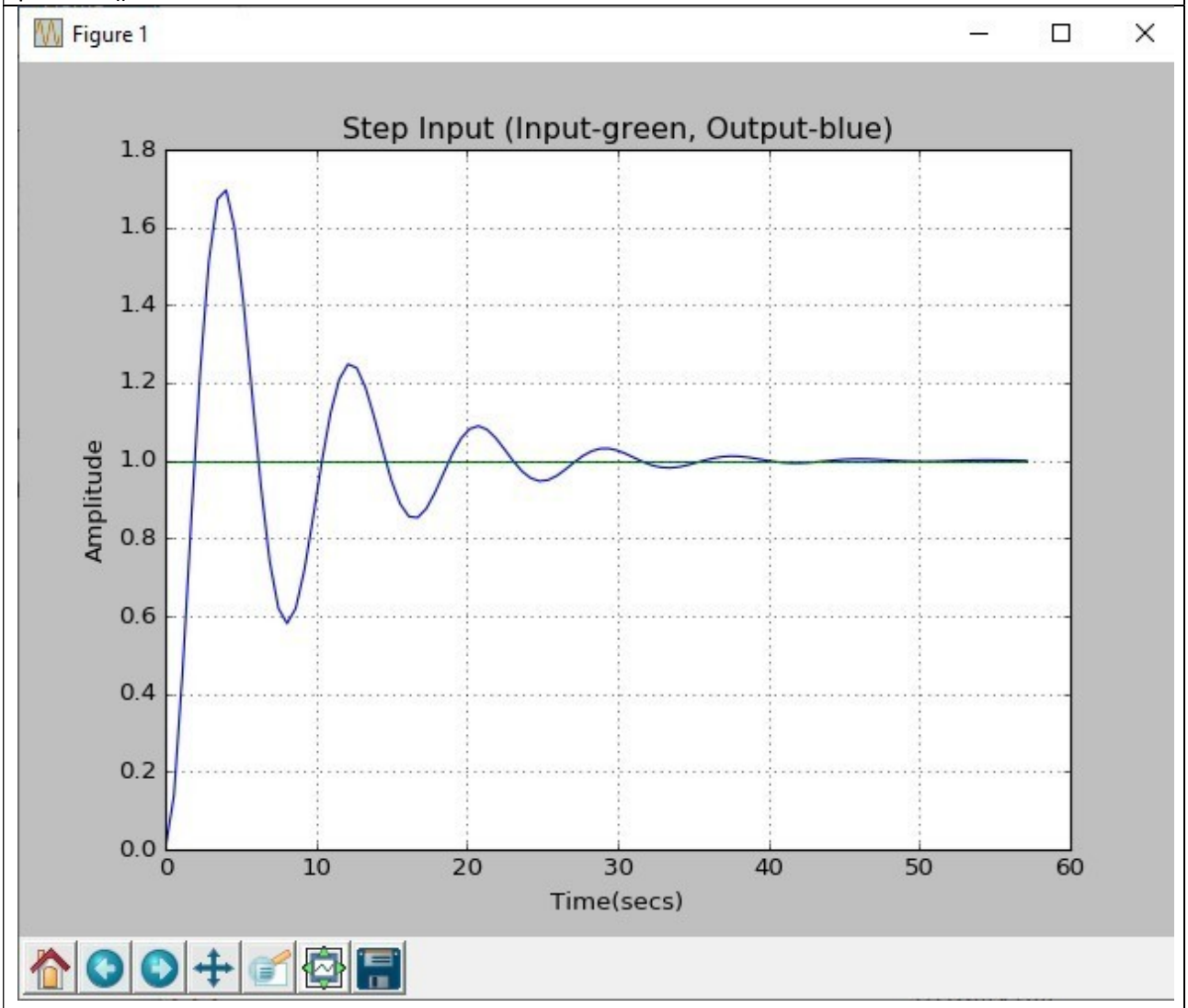


$$G(s) = \frac{(s+1)(s+3)}{s^2(s+2)(s+3)}$$

Step Input

```
s = control.tf('s')
```

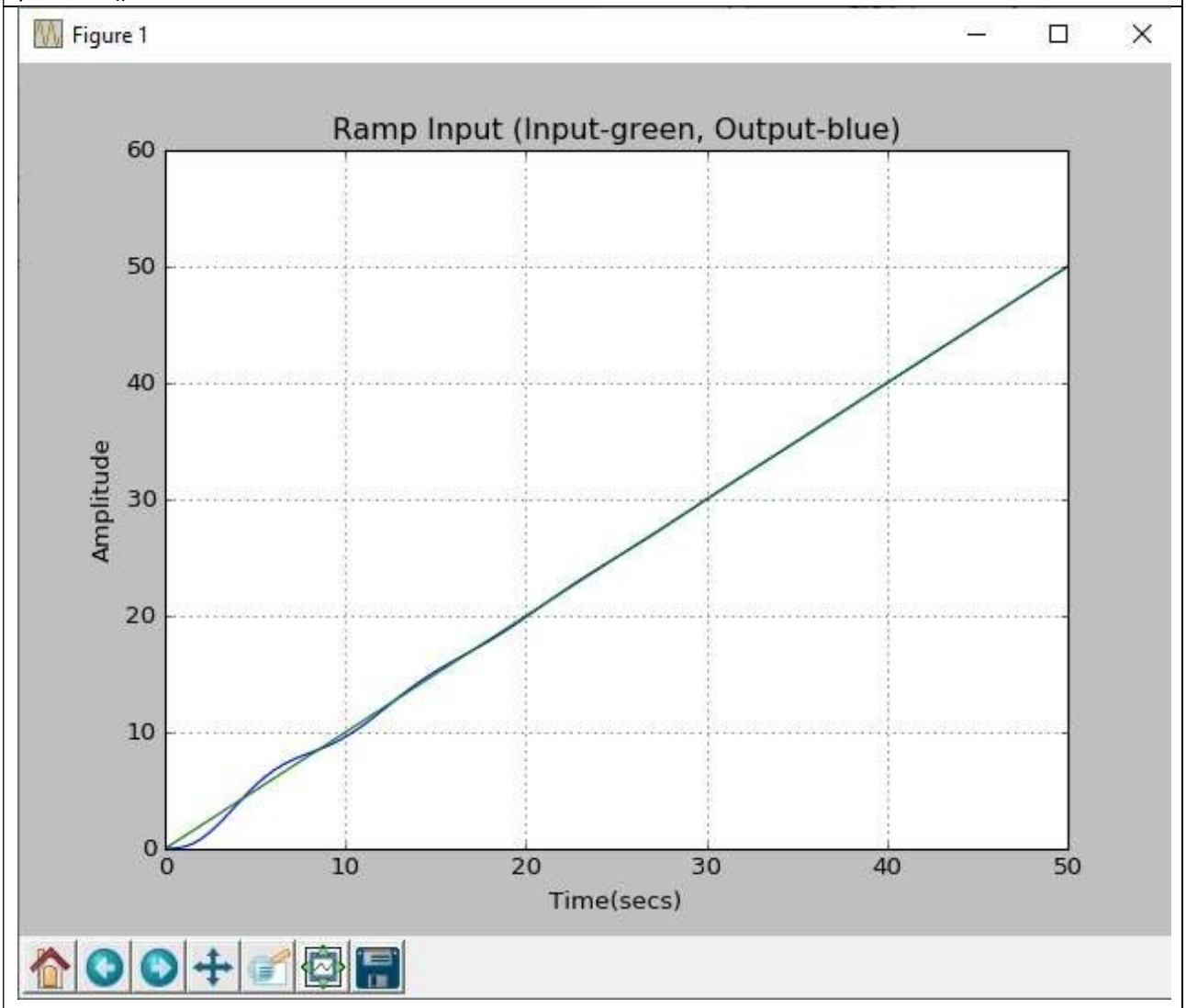
```
G = ((s+1)*(s+3))/(s**2*(s+2)*(s+3))
sys_cl = control.feedback(G,1)
t,y=control.step_response(sys_cl)
u = np.ones(t.size)
plt.plot(t,y,'b',t,u,'g')
plt.xlabel('Time(secs)')
plt.ylabel('Amplitude')
plt.title('Step Input (Input-green, Output-blue)')
plt.grid()
plt.show()
```



Ramp Input

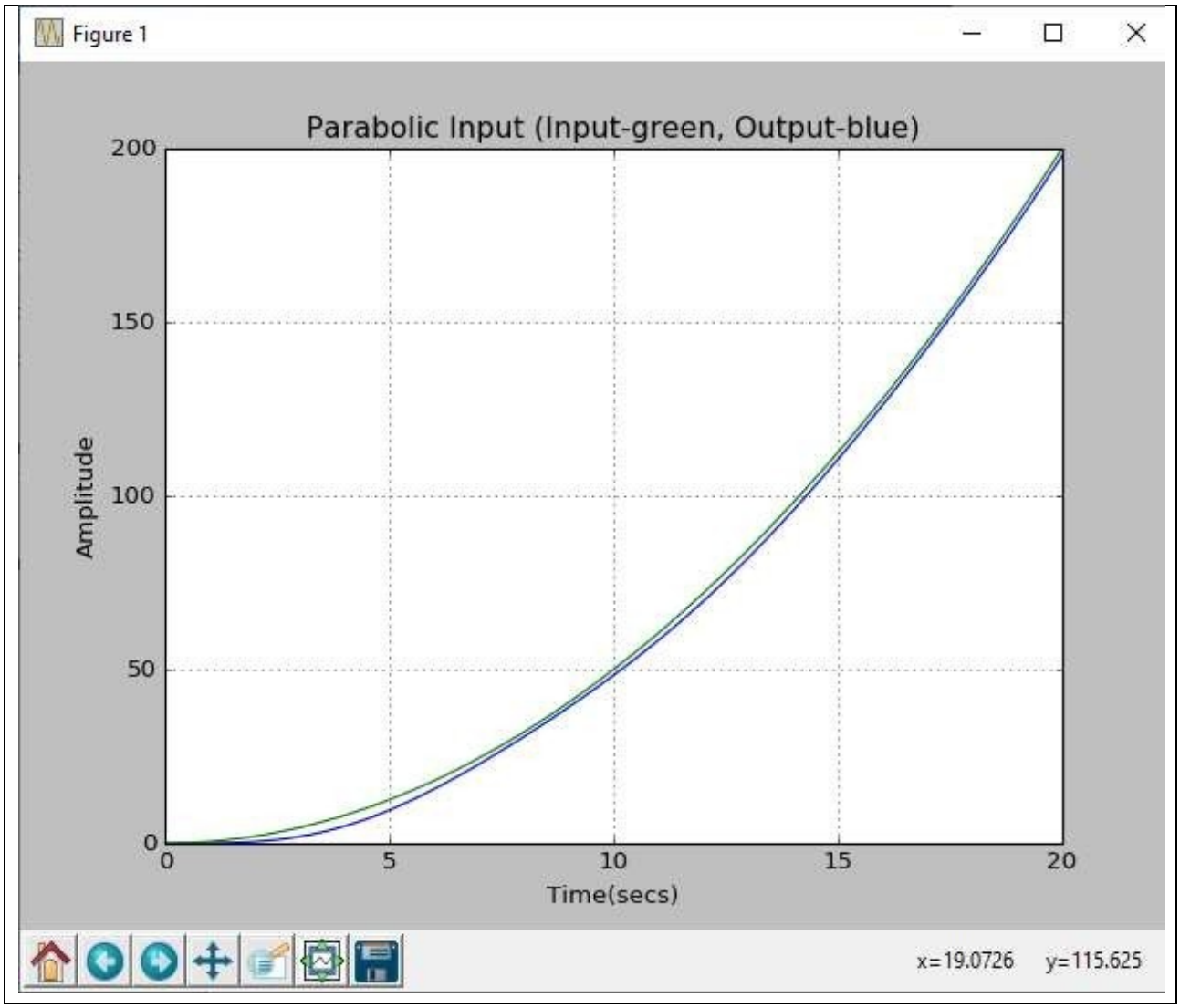
```
s = control.tf('s')
G = ((s+1)*(s+3))/(s**2*(s+2)*(s+3))
sys_cl = control.feedback(G,1)
t = np.linspace(0, 50, 100)
u=t
y,_,x = mlab.lsim(sys_cl,u,t)

plt.plot(t,y,'b',t,u,'g')
plt.xlabel('Time(secs)')
plt.ylabel('Amplitude')
```

```
plt.title('Ramp Input (Input-green, Output-blue)')
plt.grid()
plt.show()
```

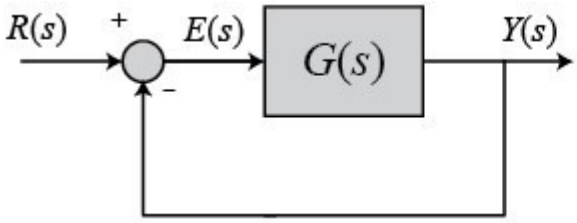


Parabolic Input

```
s = control.tf('s')
G = ((s+1)*(s+3))/(s**2*(s+2)*(s+3))
sys_cl = control.feedback(G,1)
t = np.linspace(0, 20, 100)
u = 0.5*t*t
y,_,x = mlab.lsim(sys_cl,u,t)
plt.plot(t,y,'b',t,u,'g')
plt.xlabel('Time(secs)')
plt.ylabel('Amplitude')
plt.title('Parabolic Input (Input-green, Output-blue)')
plt.grid()
plt.show()
```



https://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_Ess

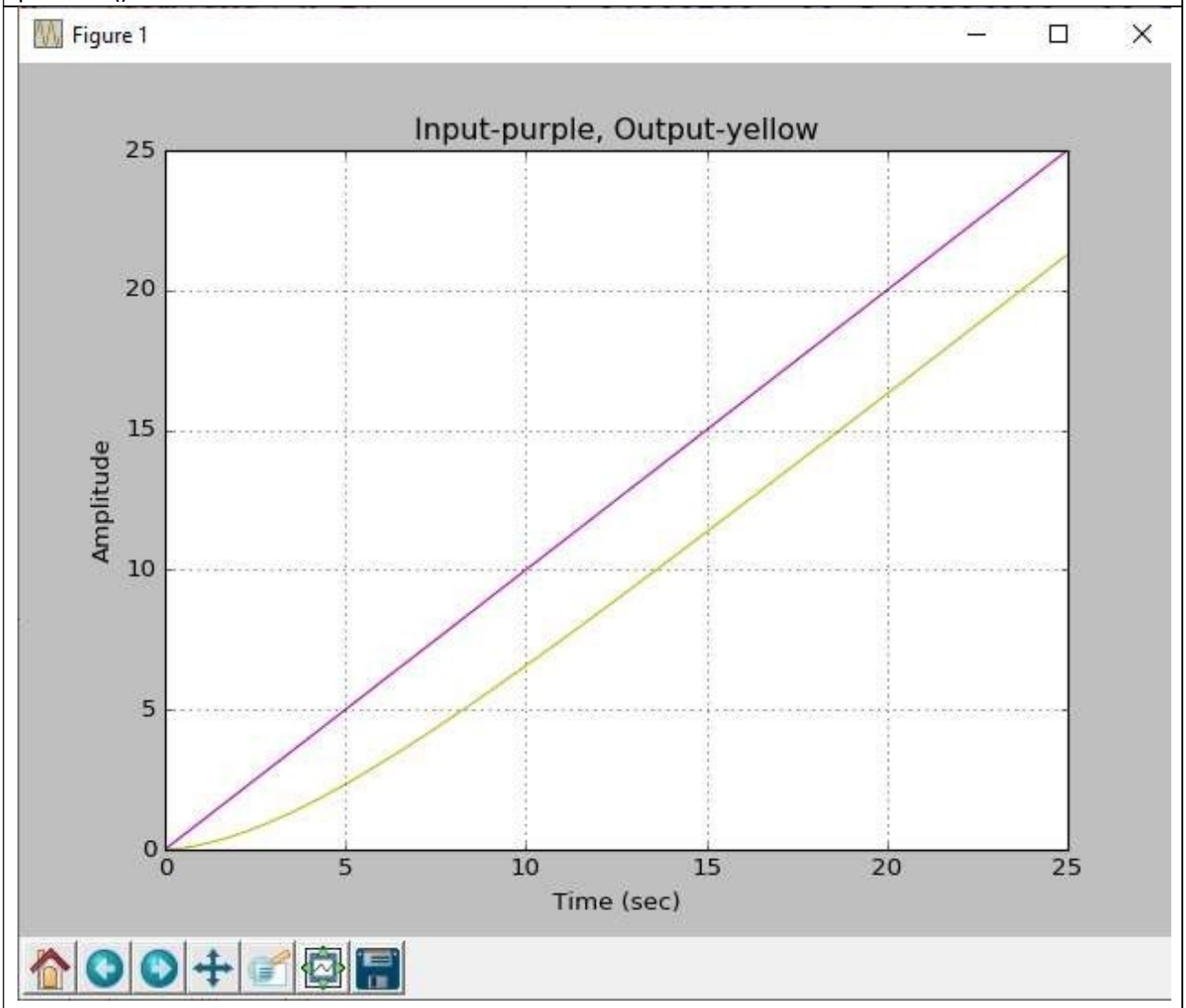


$$G(s) = \frac{K(s+3)(s+5)}{s(s+7)(s+8)}$$

Sistema di Tipo 1. Scegliere K affinche' l'errore a regime sia 0.1, in risposta ad un ingresso a rampa. Vediamo prima la risposta in caso di K=1.

```
K=1
s = control.tf('s')
G = (K*(s+3)*(s+5))/(s*(s+7)*(s+8))
T = control.feedback(G,1)
n,d= control.tfdata(T)
H= control.tf(n,d)
```

```
t=np.linspace(0, 25, 100)
u=t
import control.matlab as mlab
y, t, x = mlab.lsim(H, u, t)

plt.plot(t,y,'y',t,u,'m')
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.title('Input-purple, Output-yellow')
plt.grid()
plt.show()
```



Dalla figura si vede che l'uscita a 25 sec e' circa 16 l'errore e' di 4 circa. Dai dati dell'esercizio per un errore a regime di 0.1:

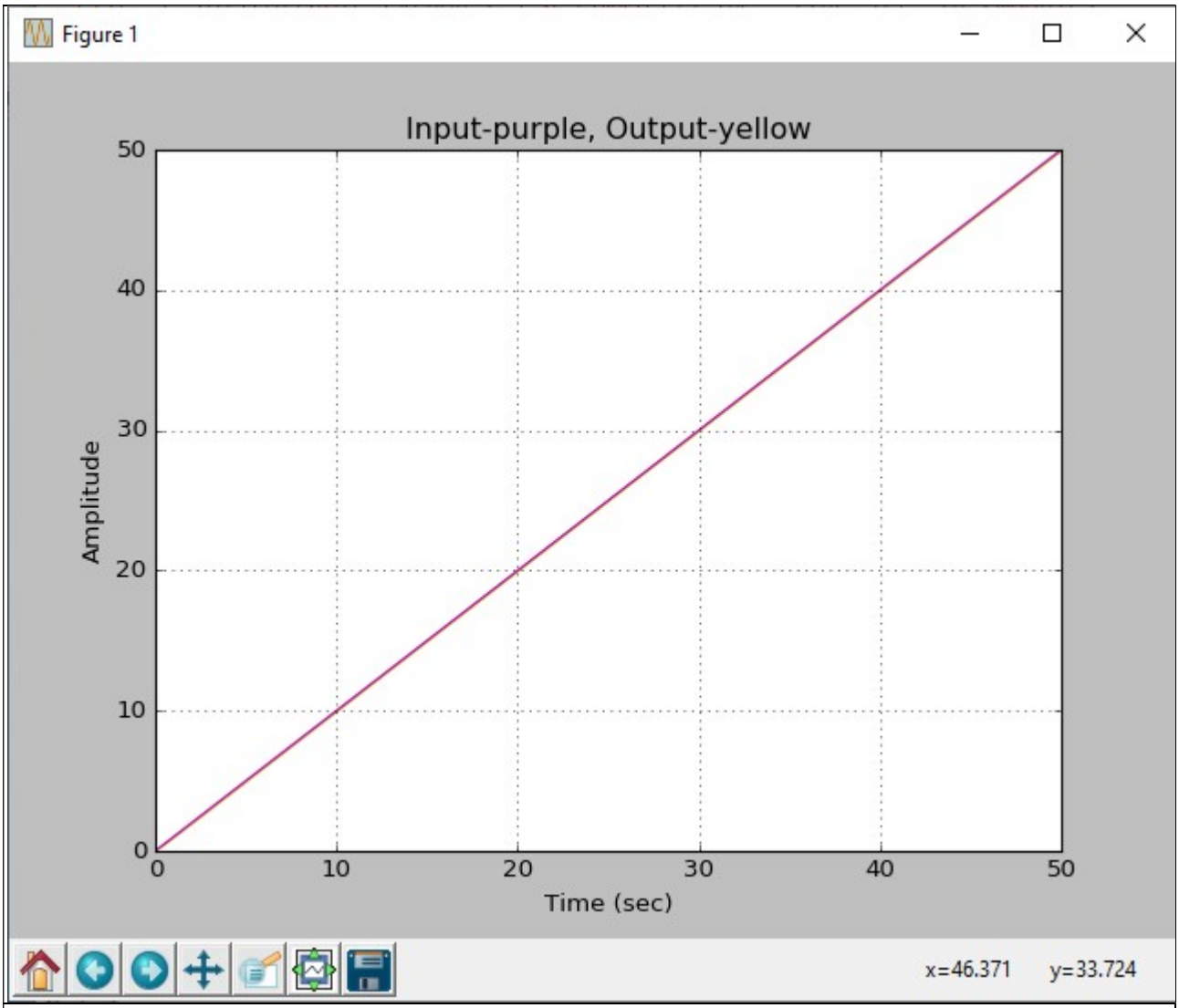
$$e(\infty) = \frac{1}{K_v} = 0.1$$

$$K_v = 10 = \lim_{s \to 0} sG(s) = \frac{15K}{56}$$

$$\Rightarrow K = 37.33$$

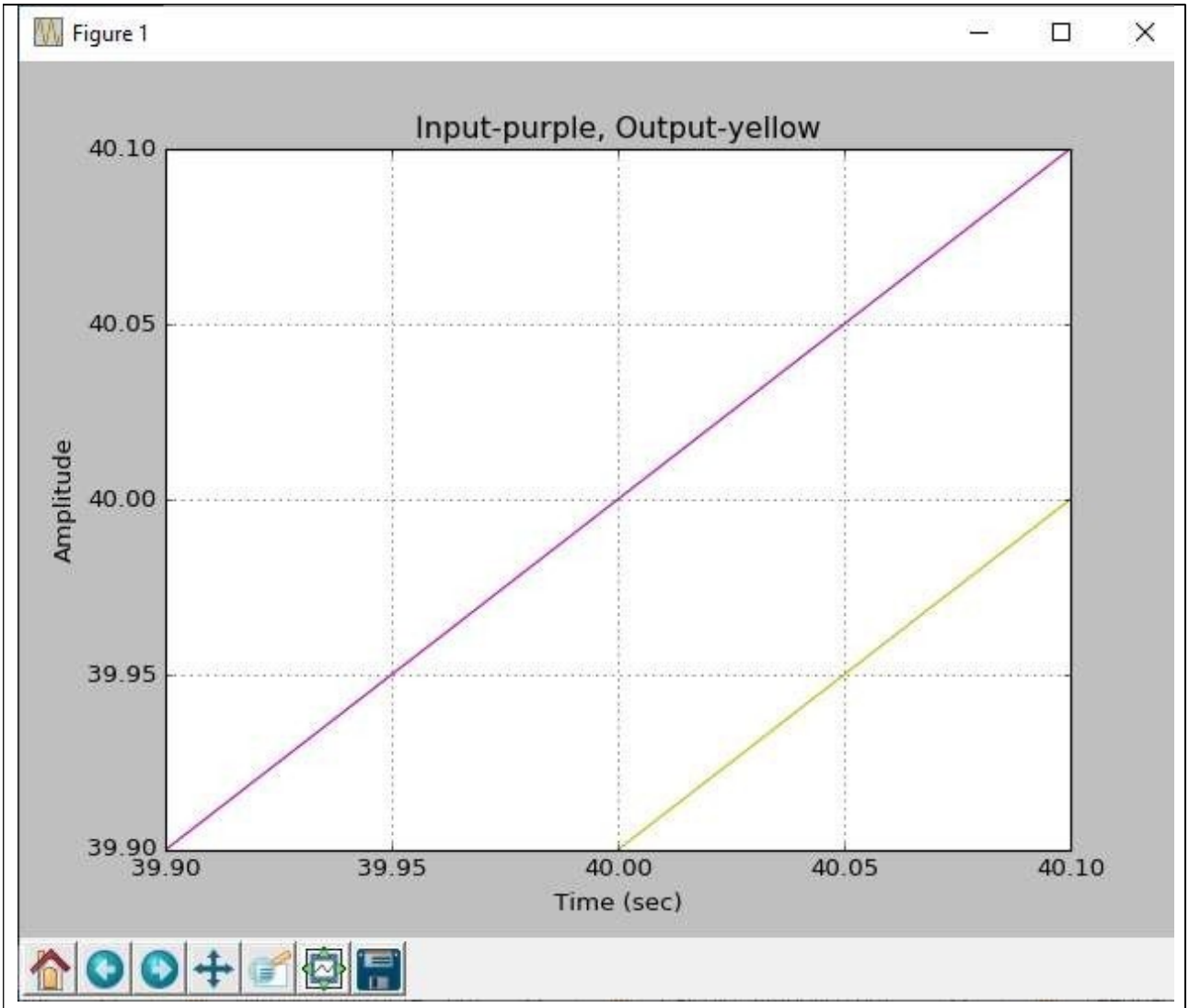
Quindi il codice Python per K=37.33

```
K = 37.33
s = control.tf('s')
G = (K*(s+3)*(s+5))/(s*(s+7)*(s+8))
T = control.feedback(G,1)
n,d= control.tfdata(T)
H= control.tf(n,d)

t=np.linspace(0, 50, 100)
u=t
import control.matlab as mlab
y, t, x = mlab.lsim(H, u, t)

plt.plot(t,y,'y',t,u,'m')
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.title('Input-purple, Output-yellow')
plt.grid()
plt.show()
```

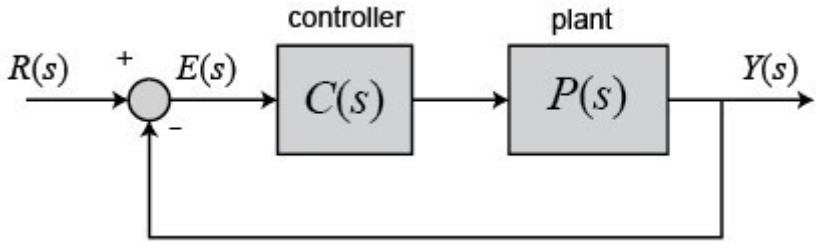


Zoomimamo per verificare l'errore a 0.1.

```
K = 37.33
s = control.tf('s')
G = (K*(s+3)*(s+5))/(s*(s+7)*(s+8))
T = control.feedback(G,1)
n,d=  control.tfdata(T)
H= control.tf(n,d)
t=np.linspace(0, 50, 100)
u=t
import control.matlab as mlab
y, t, x = mlab.lsim(H, u, t)
plt.axis([39.9,40.1,39.9,40.1])
plt.plot(t,y,'y',t,u,'m')
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.title('Input-purple, Output-yellow')
plt.grid()
plt.show()
```



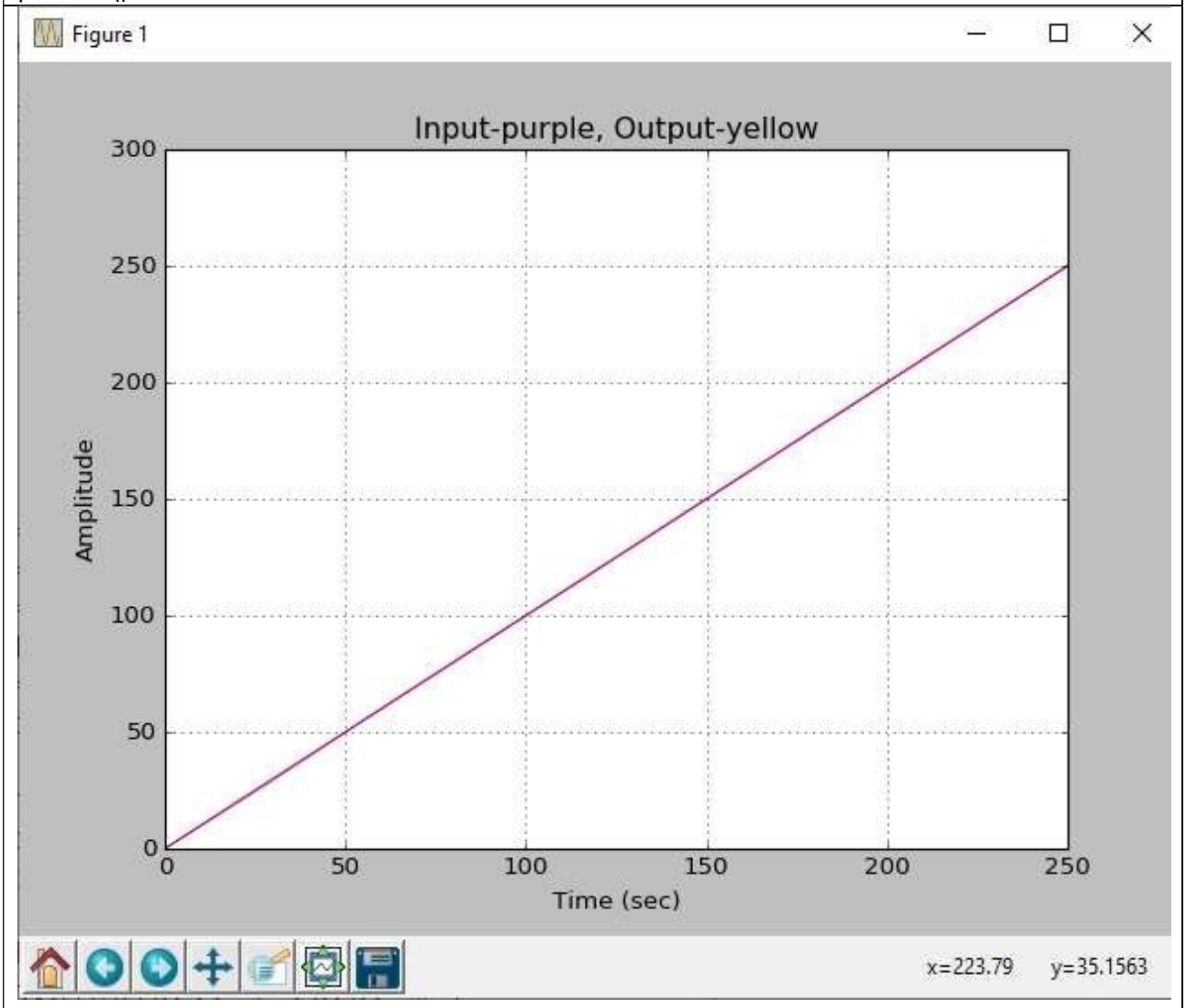
Sempre nello stesso esercizio vogliamo ora imporre un errore a regime uguale a 0:



Si sa che un sistema di tipo 2 presenta errore a regime nullo in caso di ingresso a rampa. Percio' aggiungiamo un controllore al nostro sistema di tipo 1 e utilizziamo K=1.

```
K = 1
s = control.tf('s')
P = (K*(s+3)*(s+5))/(s*(s+7)*(s+8))
C=1/s
T = control.feedback(C*P,1)
n,d= control.tfdata(T)
H= control.tf(n,d)
t=np.linspace(0, 250, 100)
u=t
import control.matlab as mlab
y, t, x = mlab.lsim(H, u, t)
```

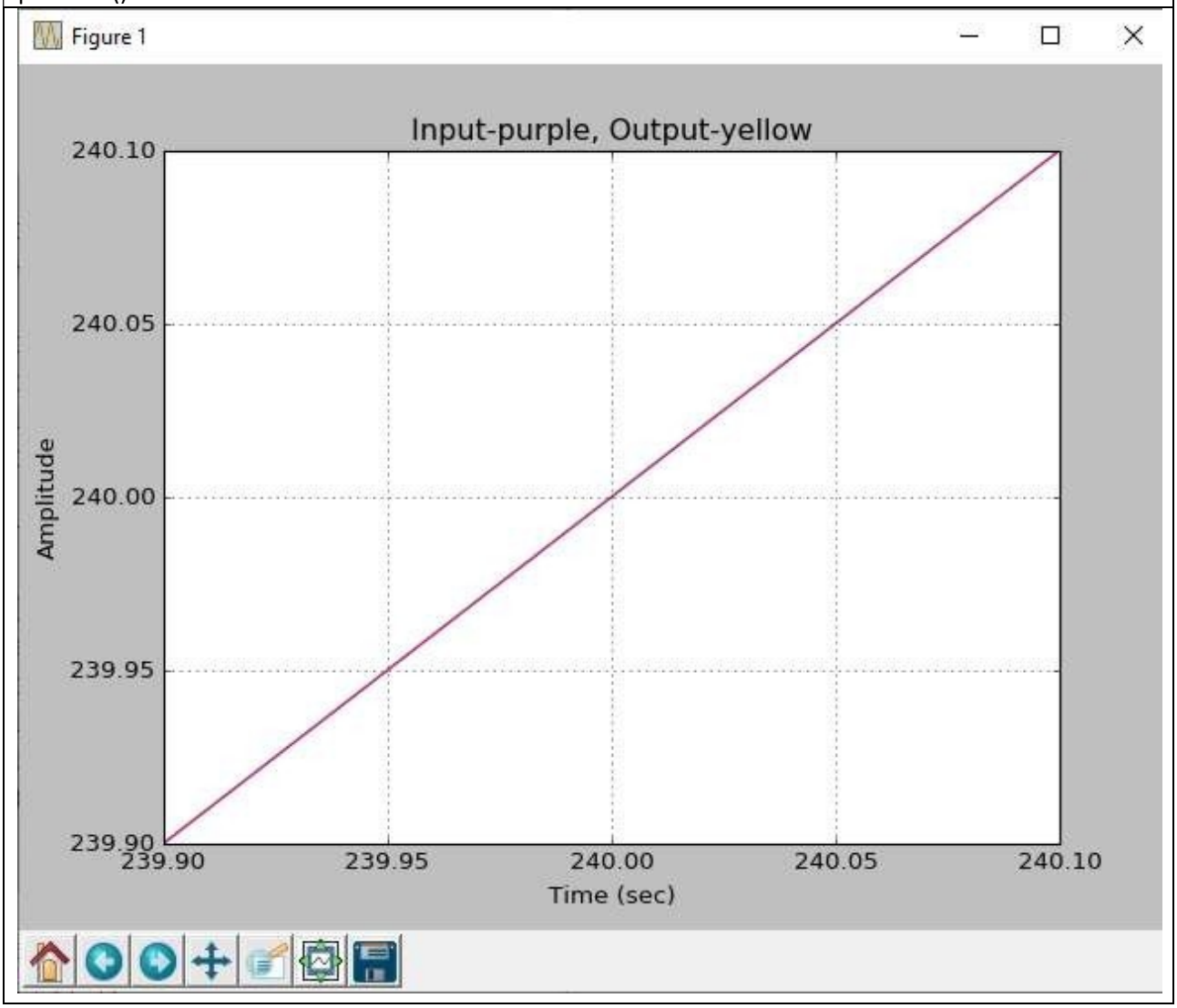
```
plt.plot(t,y,’y’,t,u,’m’)
plt.xlabel(‘Time (sec)’)
plt.ylabel(‘Amplitude’)
plt.title(‘Input-purple, Output-yellow’)
plt.grid()
plt.show()
```



Zoomando:

```
K = 1
s = control.tf('s')
P = (K*(s+3)*(s+5))/(s*(s+7)*(s+8))
C=1/s
T = control.feedback(C*P,1)
n,d= control.tfdata(T)
H= control.tf(n,d)
t=np.linspace(0, 250, 100)
u=t
import control.matlab as mlab
y, t, x = mlab.lsim(H, u, t)
plt.axis([239.9,240.1,239.9,240.1])
plt.ticklabel_format(useOffset=False, style='plain')
plt.plot(t,y,'y',t,u,'m')
plt.xlabel('Time (sec)')
```

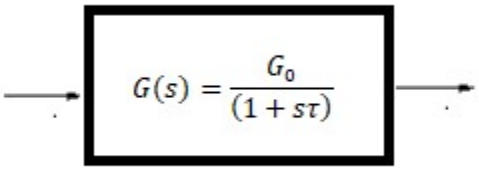
```
plt.ylabel('Amplitude')
plt.title('Input-purple, Output-yellow')
plt.grid()
plt.show()
```



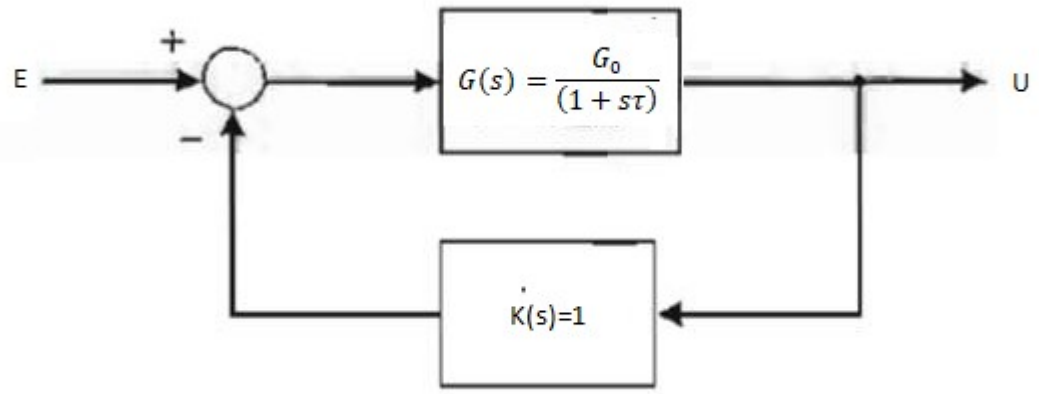
IL CONTROLLO AUTOMATICO

Retroazionare un sistema. Consideriamo il sistema del 1' ordine

$$G(s) = \frac{G_0}{(1 + s\tau)}$$



Retrozioniamo:



$$GR(s) = \frac{G(s)}{1 + G(s)K(s)} = \frac{\frac{G_0}{1+s\tau}}{1 + K(s)\frac{G_0}{1+s\tau}} = \frac{G_0}{1 + s\tau + K(s)G_0}$$

Ponendo K(s)=1 per semplificare:

$$GR(s) = \frac{G_0}{1 + s\tau + G_0} = \frac{\frac{G_0}{1+G_0}}{\frac{1+G_0}{1+G_0} + \frac{s\tau}{1+G_0}} = \frac{G_0'}{1 + s\tau'}$$

Cioe' la funzione di trasferimento del sistema retroazionato e' dello stesso tipo di quello non retroazionato, ma con guadagno statico e costante di tempo ridotto di (1+sτ) . Iinoltre se G0>>1 avro':

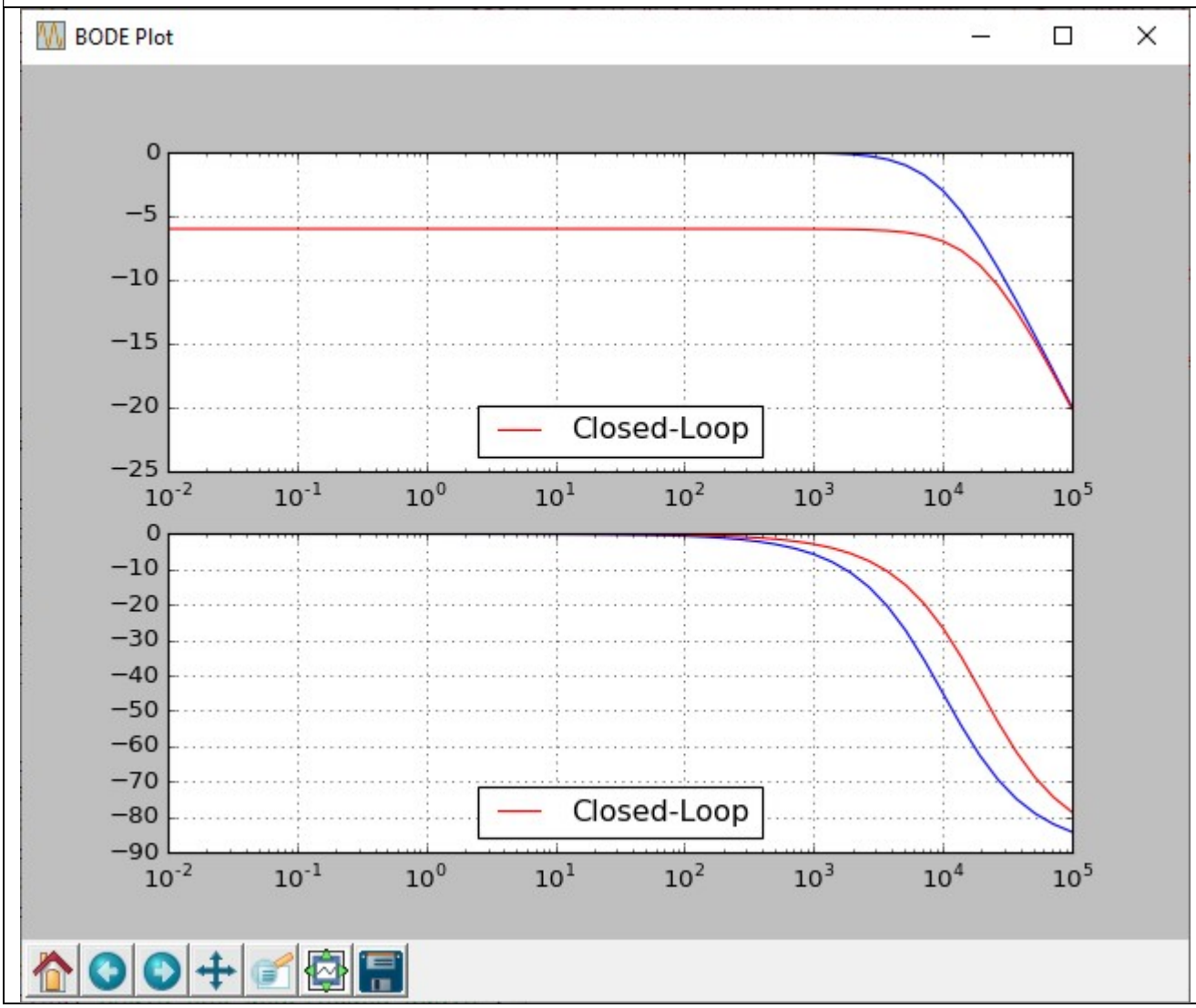
$$GR(s) \simeq \frac{1}{1 + s\frac{\tau}{G_0}}$$

In questo caso quindi il guadagno statico uguale a 1, mentro la costante di tempo risulta ridotta di G0.

Si puo vedere che la distanza delle due pulsazioni di taglio (anello aperto anello chiuso) sia 20log(1+G_0)

```
def G(w,tau,G0):
   G = G0/(1+tau*1j*w)
   return G
def GR(w,tau,G0):
   GR=G(w,tau,G0)/(1+G(w,tau,G0))
   return GR
w = np.logspace(-2,5)
G0=1
tau=1E-4
print 'G0','tau','w0','G0R','tauR','w0R'
print G0,tau,1/tau,G0/(1+G0),tau/(1+G0),(1+G0)/tau
Gdb=20*np.log10(abs(G(w,tau,G0)))
phase = np.angle(G(w,tau,G0),deg=True)
plt.figure('BODE Plot')
plt.subplot (2, 1, 1)
plt.plot(w, Gdb,'b')
GRDb=20*np.log10(abs(GR(w,tau,G0)))
phaseR = np.angle(GR(w,tau,G0),deg=True)
plt.plot(w, GRDb,'r',label='Closed-Loop')
plt.legend(loc="lower center")
plt.grid()
plt.xscale('log')
plt.subplot (2, 1, 2)
plt.plot(w, phase)
plt.plot(w, phaseR,'r',label='Closed-Loop')
plt.legend(loc="lower center")
plt.xscale('log')
plt.grid()
```

```
plt.show()
>>>
G0 tau w0 G0R tauR w0R
1 0.0001 10000.0 0 5e-05 20000.0
```



SISTEMA TERMICO – ANELLO APERTO

Equazione differenziale del sistema forno:

$$\tau \cdot \frac{\Delta T(t)}{\Delta t} + T(t) = T_e(t)$$

Approssimazione numerica:

$$\tau \cdot \frac{T(n\Delta t + \Delta t) - T(n\Delta t)}{\Delta t} + T(n\Delta t) = T_e(n\Delta t)$$

Cioe':

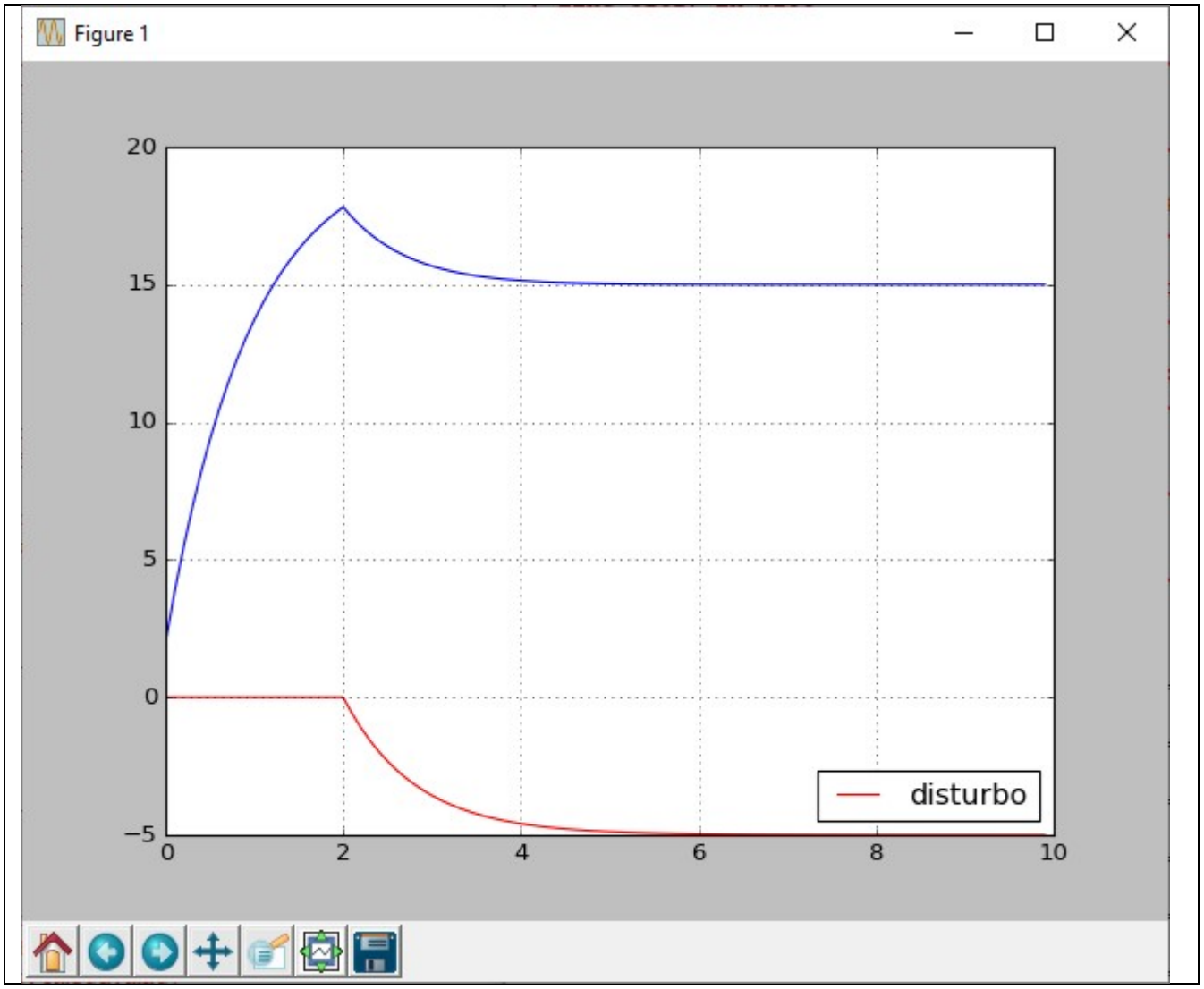
$$T(n\Delta t + \Delta t) = \left(T_e(n\Delta t) - T(n\Delta t)\right) \cdot \frac{\Delta t}{\tau} + T(n\Delta t)$$

Ponendo per semplificare $\tau = 1$:

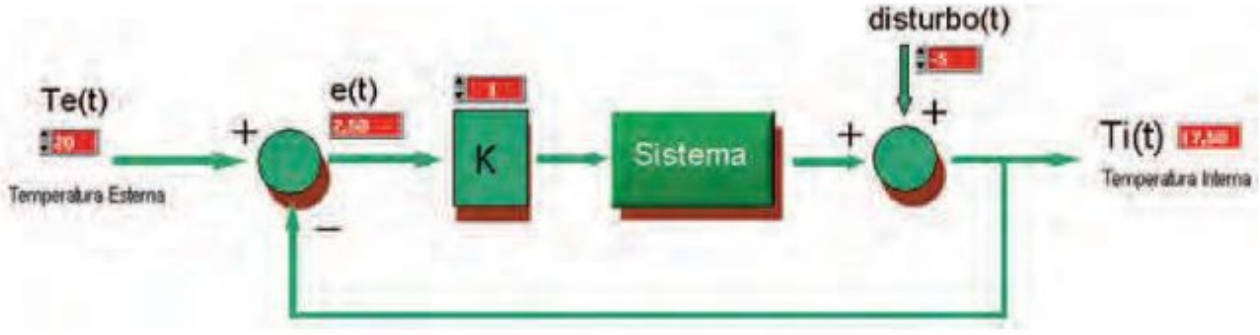
$$T=(Te-T)dt + T$$

Disturbo additivo ad Anello aperto

```
N=100
Tf=0
dt=5./50
Te=20
T0=0
T=T0
t=0
tl=[0]*N
T=[0]*N
To=[0]*N
Td=[0]*N
k=int(2/dt)
Td[k:] = [-5 for c in Td[k:]]
for k in range(20,100):
   Td[k]=-5*(1.-math.exp(-t/(8.E-1)))
   t += dt
t=0
for i in range (0,N):
   if i>0:
      T[i]=Te*dt-T[i-1]*dt+T[i-1]
   else:
      T[i]=Te*dt-T0*dt+T0
   To[i]=T[i]+Td[i]
   tl[i]=i*dt
plt.plot(tl,To)
plt.plot(tl,Td,'r',label='disturbo')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

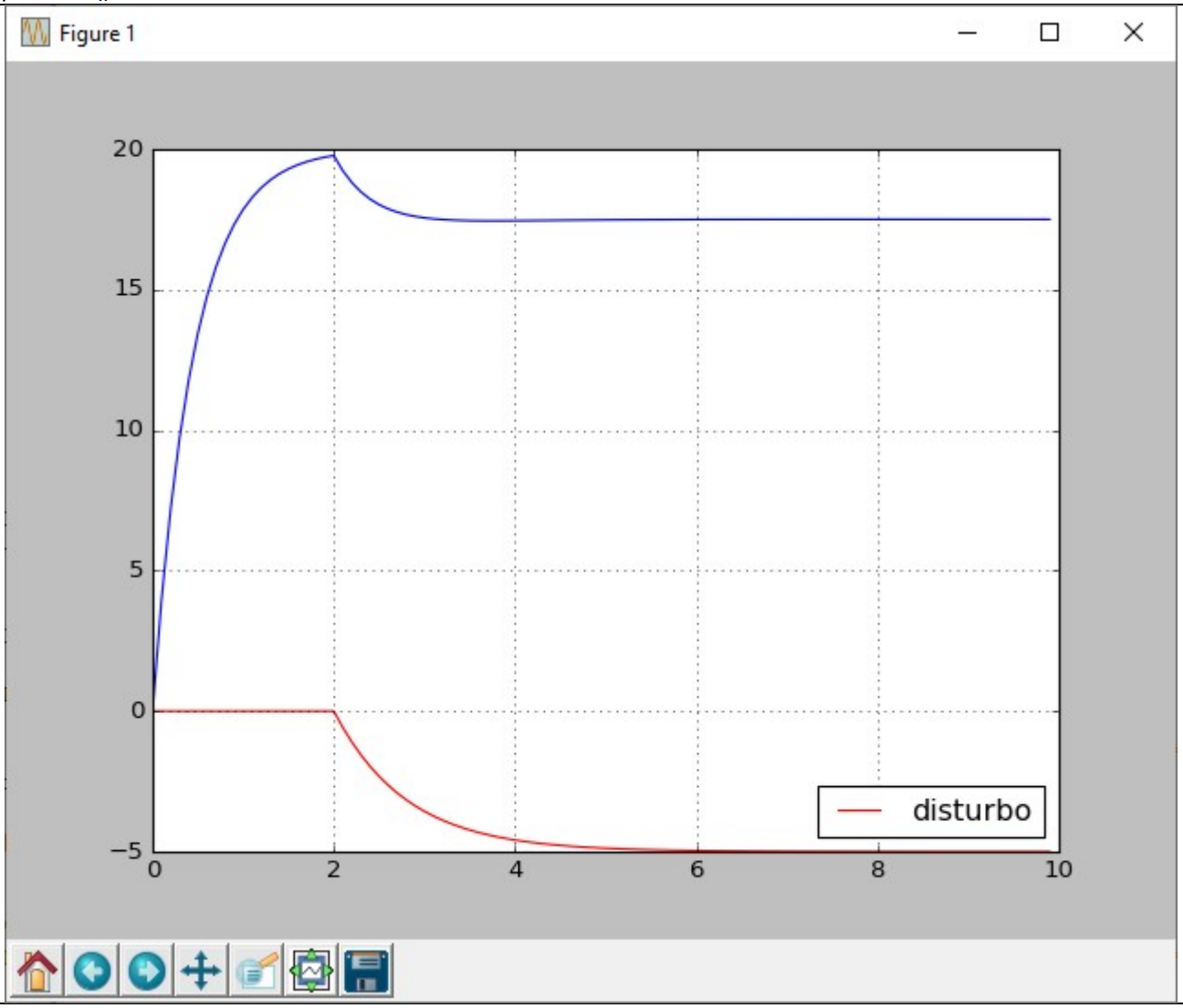


Disturbo additivo ad Anello Chiuso

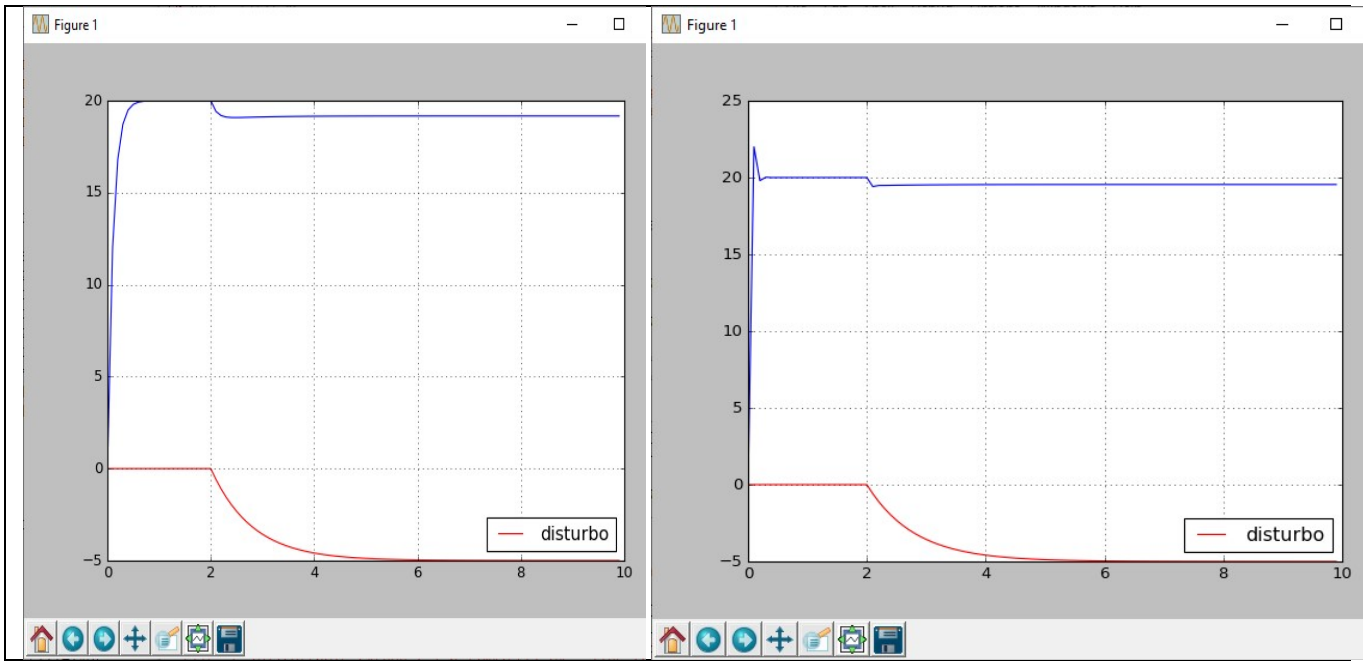



```
N=100
K=1     #Blocco K
dt=5./50
Te=20
T0=0
t=[0]*N
td=20   #tempo di attivazione disturbo
T=[0]*N
TiR=[0]*N
To=[0]*N
Td=[0]*N
t=0
```

```
for i in range(td,100):#disturbo
   Td[i]=-5*(1.-math.exp(-t/(8.E-1)))
   t += dt

t=[0]*N
for i in range (0,N):
   if i>0:
      T[i]=(((Te-To[i-1])*K+Te)-T[i-1])*dt+T[i-1]
   else:
      T[i]=T0
   To[i]=T[i]+Td[i]
   t[i]=i*dt
   #print Te-To[i],To[i],T[i]

plt.plot(t,To)
plt.plot(t,Td,'r',label='disturbo')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

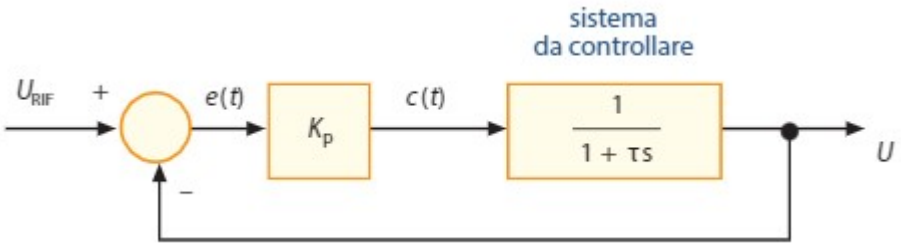


…… K=5 ……	…… K=10 ……



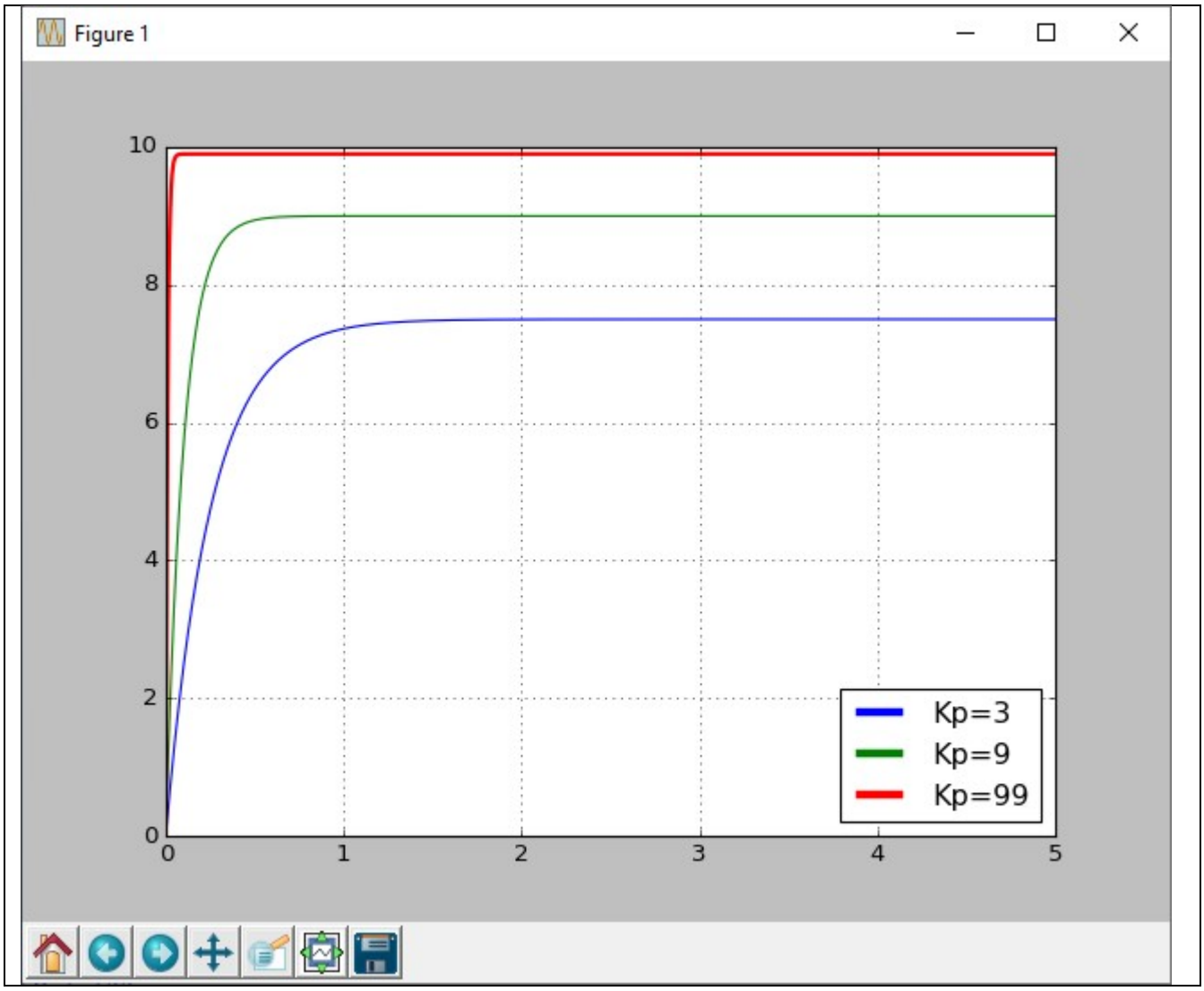
CONTROLLO PROPORZIONALE

Sistema- calcolare la risposta al variare di K_p




```
#NB:usiamo la funzione control.TransferFunction per avere le fdt semplificate
##Usando il codice seguente si ottiene lo stesso grafico
##s = control.tf('s')
##t = np.linspace(0, 5, 1000)
##Kp=3
##G=1/(1+s)
##GR=(Kp*G)/(1+Kp*G)
##print GR
##_,y=control.step_response(GR*10,T=t)
##plt.plot(t,y,label="Kp="+str(Kp))
##Kp=9
##GR=(Kp*G)/(1+Kp*G)
##print GR
##_,y=control.step_response(GR*10,T=t)
##plt.plot(t,y,label="Kp="+str(Kp))
##Kp=99
##GR=(Kp*G)/(1+Kp*G)
##print GR
##_,y=control.step_response(GR*10,T=t)
##plt.plot(t,y,label="Kp="+str(Kp),linewidth=2.0)
##leg=plt.legend(loc="lower right")
##for line in leg.get_lines():
##    line.set_linewidth(4.0)
##plt.grid()
##plt.show()
```

```
t = np.linspace(0, 5, 1000)
Kp=3
G=control.TransferFunction(1, [1,1])
GR=control.feedback(Kp*G,1)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp))
Kp=9
GR=control.feedback(Kp*G,1)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp))
Kp=99
GR=control.feedback(Kp*G,1)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp),linewidth=2.0)
leg=plt.legend(loc="lower right")
for line in leg.get_lines():
    line.set_linewidth(4.0)
plt.grid()
plt.show()
>>>
  3
-----
s + 4


   9
------
s + 10


   99
-------
s + 100
```



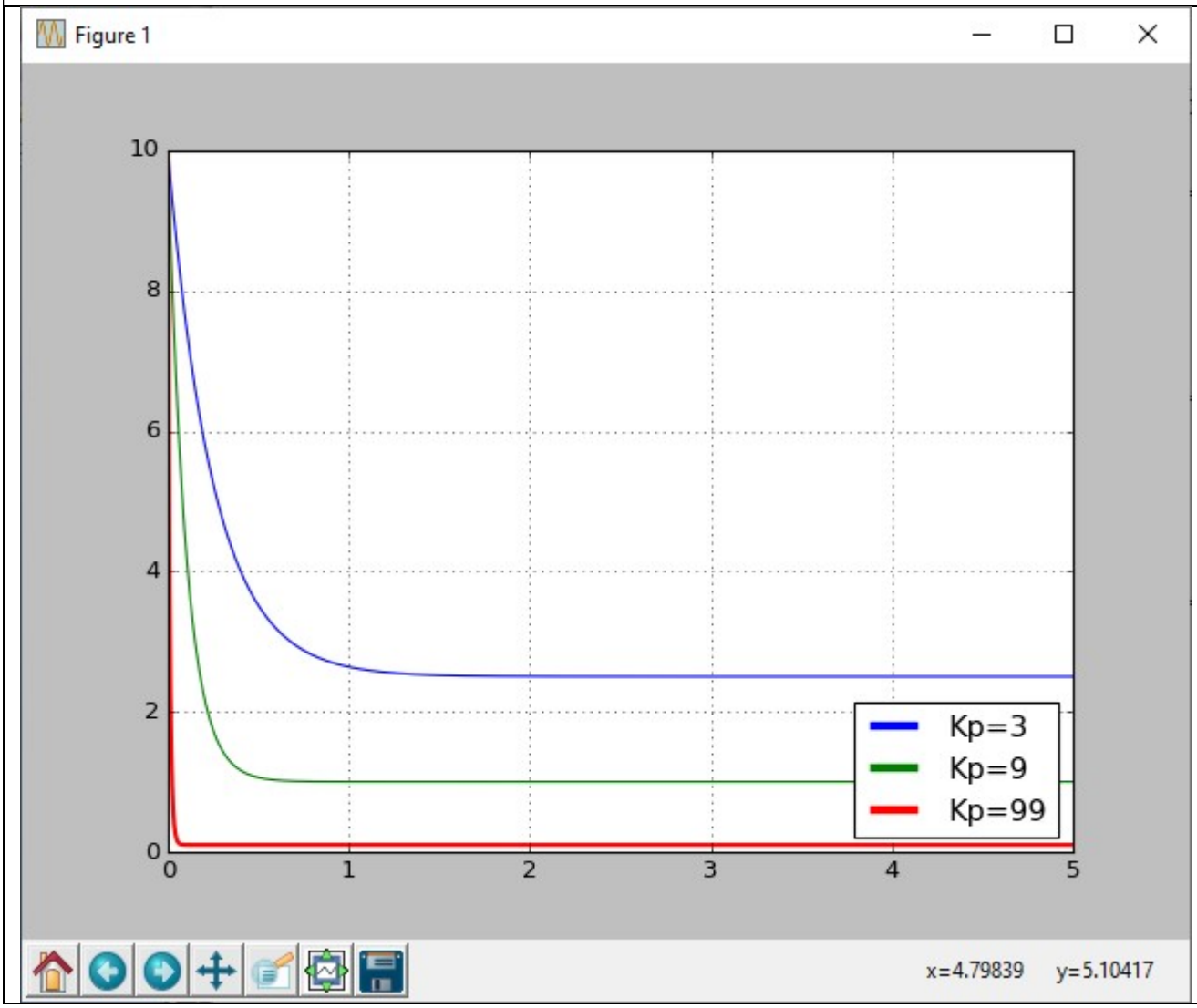
Sistema- calcolare l'errore al variare di K_p

```
t = np.linspace(0, 5, 1000)
Kp=3
G=control.TransferFunction(1, [1,1])
GR=control.feedback(1,Kp*G)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp))
Kp=9
GR=control.feedback(1,Kp*G)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp))
Kp=99
GR=control.feedback(1,Kp*G)
print GR
_,y=control.step_response(GR*10,T=t)
plt.plot(t,y,label="Kp="+str(Kp),linewidth=2.0)
leg=plt.legend(loc="lower right")
for line in leg.get_lines():
    line.set_linewidth(4.0)
plt.grid()
plt.show()
>>>
```

s + 1
-----
s + 4

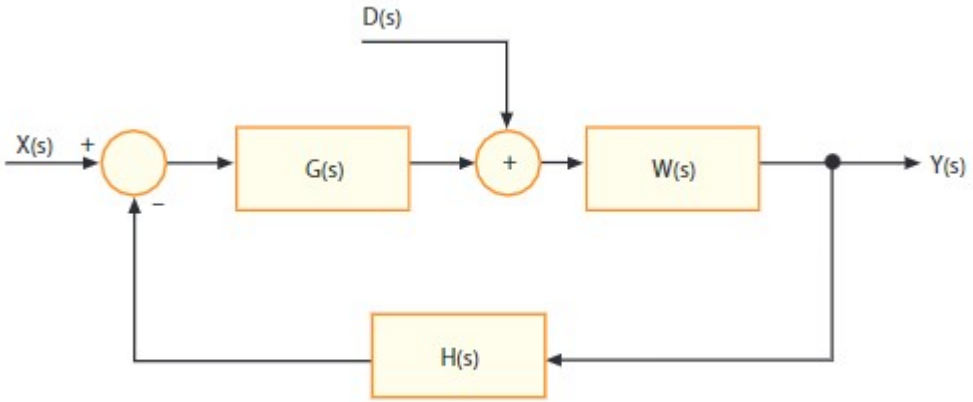
s + 1
------
s + 10

s + 1
----
s + 100



CONTROLLO STATICO

Dato il seguente sistema:



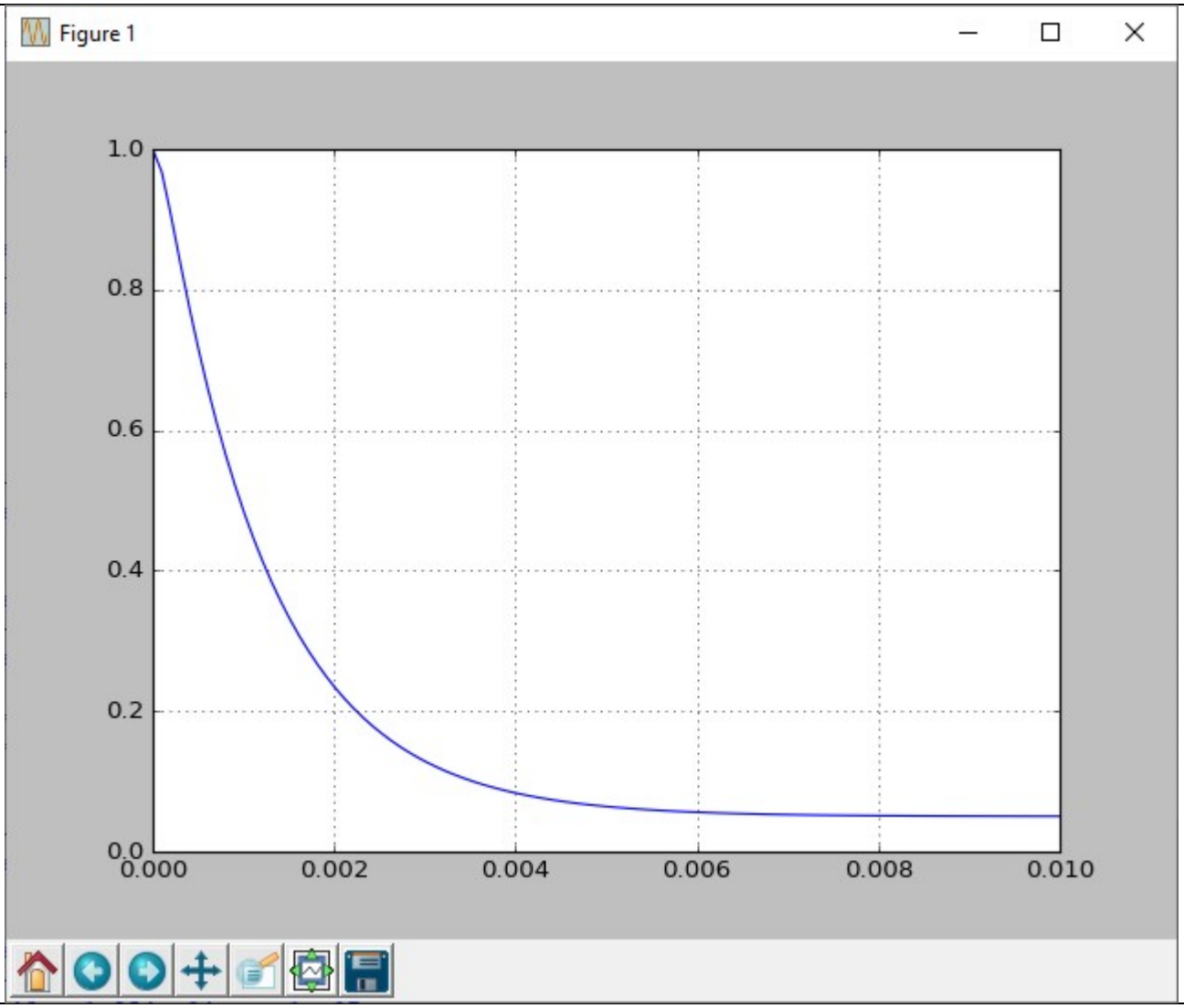
$$G(s) = \frac{400}{s+40} \qquad W(s) = \frac{25000}{s+12500} \qquad H(s) = \frac{1}{10}$$

Con x(t) ingresso unitario, progettare un blocco C(s) a monte di G(s) che determini un errore statico in uscita ineriore o uguale a 0.05.

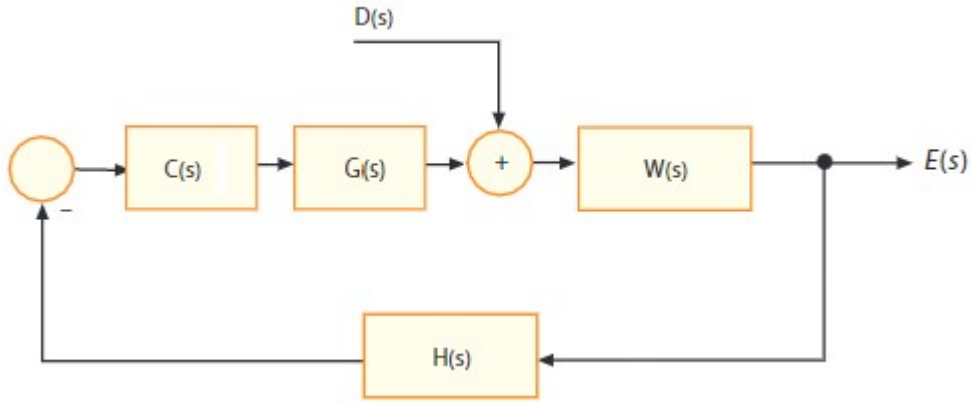
Per il principio di sovrapposizione degli effetti analizziamo la risposta all'ingresso con disturbo nullo:

$$E(s) = X(s)\frac{1}{1 + H(s)W(s)G(s)C(s)}$$

```
s = control.tf('s')
t = np.linspace(0, 0.01, 100)
G=400./(s+40)
H=1./10
W=25000./(s+12500.)
C=9.5
GTOT=W*G*H*C
print GTOT
GTOT=1./(1+GTOT)
print GTOT
_,y=control.step_response(GTOT,T=t)

plt.plot(t,y)
plt.grid()
plt.show()

>>>
      9.5e+06
-------------------------
s^2 + 1.254e+04 s + 5e+05


s^2 + 1.254e+04 s + 5e+05
-------------------------
s^2 + 1.254e+04 s + 1e+07
```



Per quanto riguarda la risposta al disturbo con ingresso nullo:

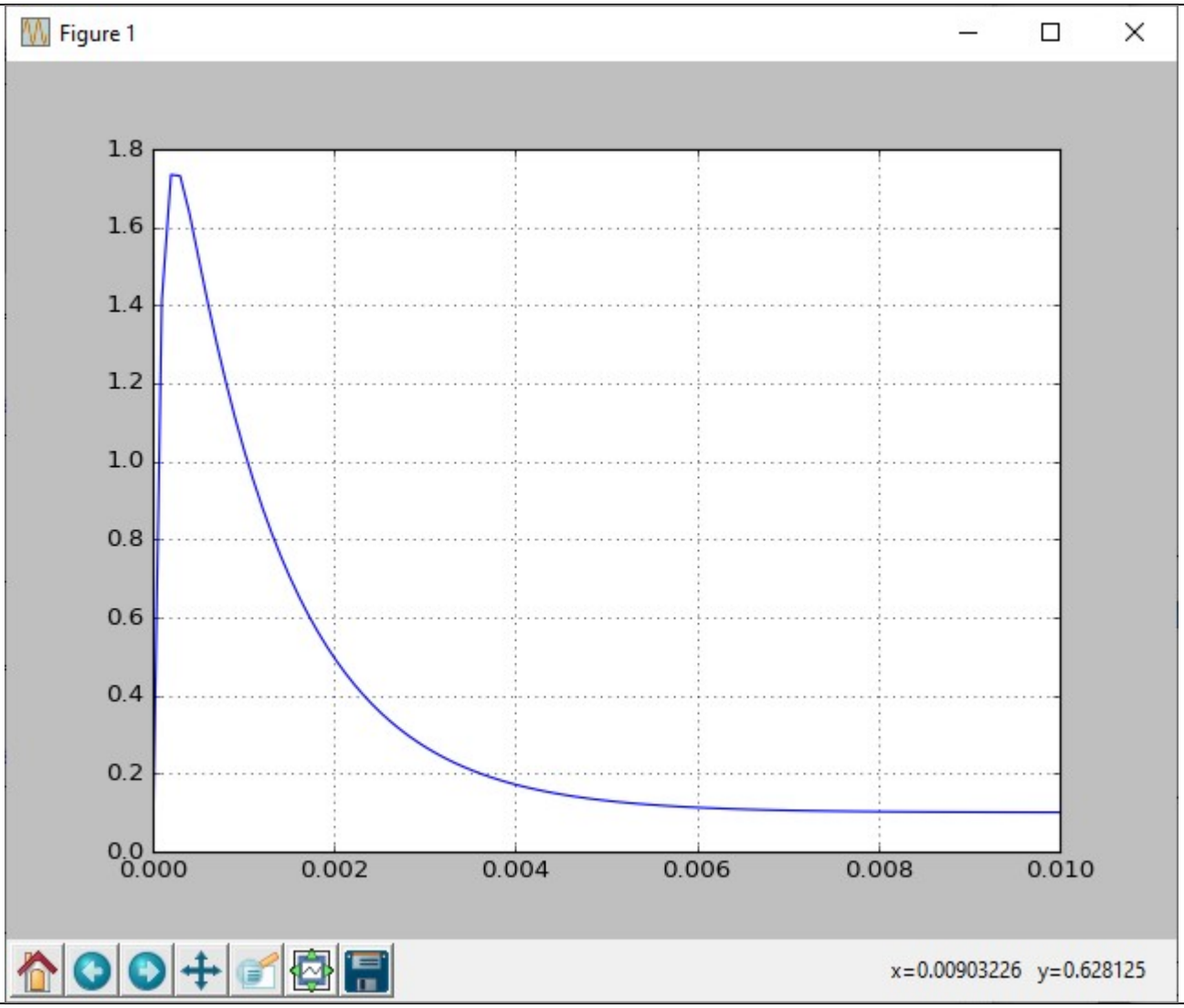


.

$$E(s) = D(s)\frac{W(s)}{1 + H(s)W(s)G(s)C(s)}$$

```
s = control.tf('s')
t = np.linspace(0, 0.01, 100)
G=400./(s+40)
H=1./10
W=25000./(s+12500.)
C=9.5

GTOT=W*G*H*C
print GTOT
GTOT=W/(1+GTOT)
```

```
print GTOT
_,y=control.step_response(GTOT,T=t)
plt.plot(t,y)
plt.grid()
plt.show()
>>>
        9.5e+06
-------------------------
s^2 + 1.254e+04 s + 5e+05


   2.5e+04 s^2 + 3.135e+08 s + 1.25e+10
-----------------------------------------------
s^3 + 2.504e+04 s^2 + 1.668e+08 s + 1.25e+11
```



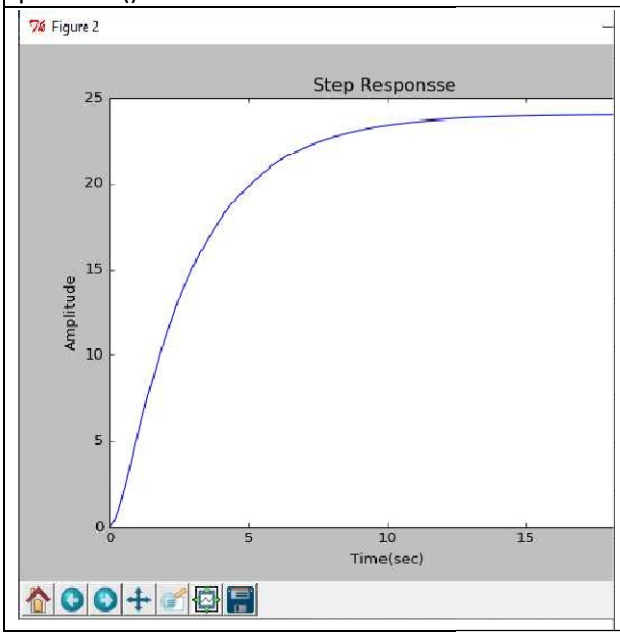
CONTROLLO DINAMICO

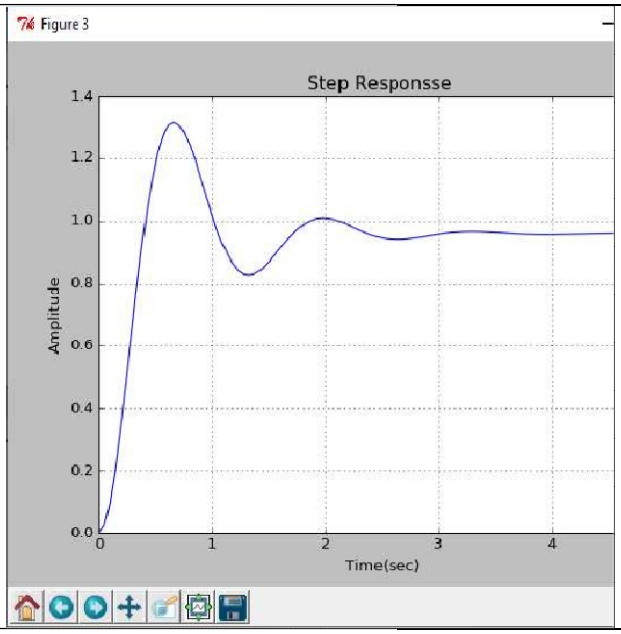
Esercizio – Comparsa di oscillazioni retroazionando un sistema

$$G(s) = \frac{24}{s^2 + 3s + 1}$$

```
import control
import matplotlib.pyplot as plt
import math
import numpy as np
from control.matlab import *
s=control.tf('s')
G=24/(s**2+3*s+1)
```

```
print G
GR=control.feedback (G, 1)
print GR
(p, z) = control.pzmap (G)
print p,z
t,y=control.step_response(G)#N.B.!!!: y,t=step(G)
plt.figure()
plt.plot(t,y,"b")
plt.title('Step Responsse ')
plt.ylabel('Amplitude')
plt.xlabel('Time(sec)')
t,y=control.step_response(GR)
plt.figure()
plt.plot(t,y,"b")
plt.title('Step Responsse ')
plt.ylabel('Amplitude')
plt.xlabel('Time(sec)')
plt.grid(True)
plt.show()
```





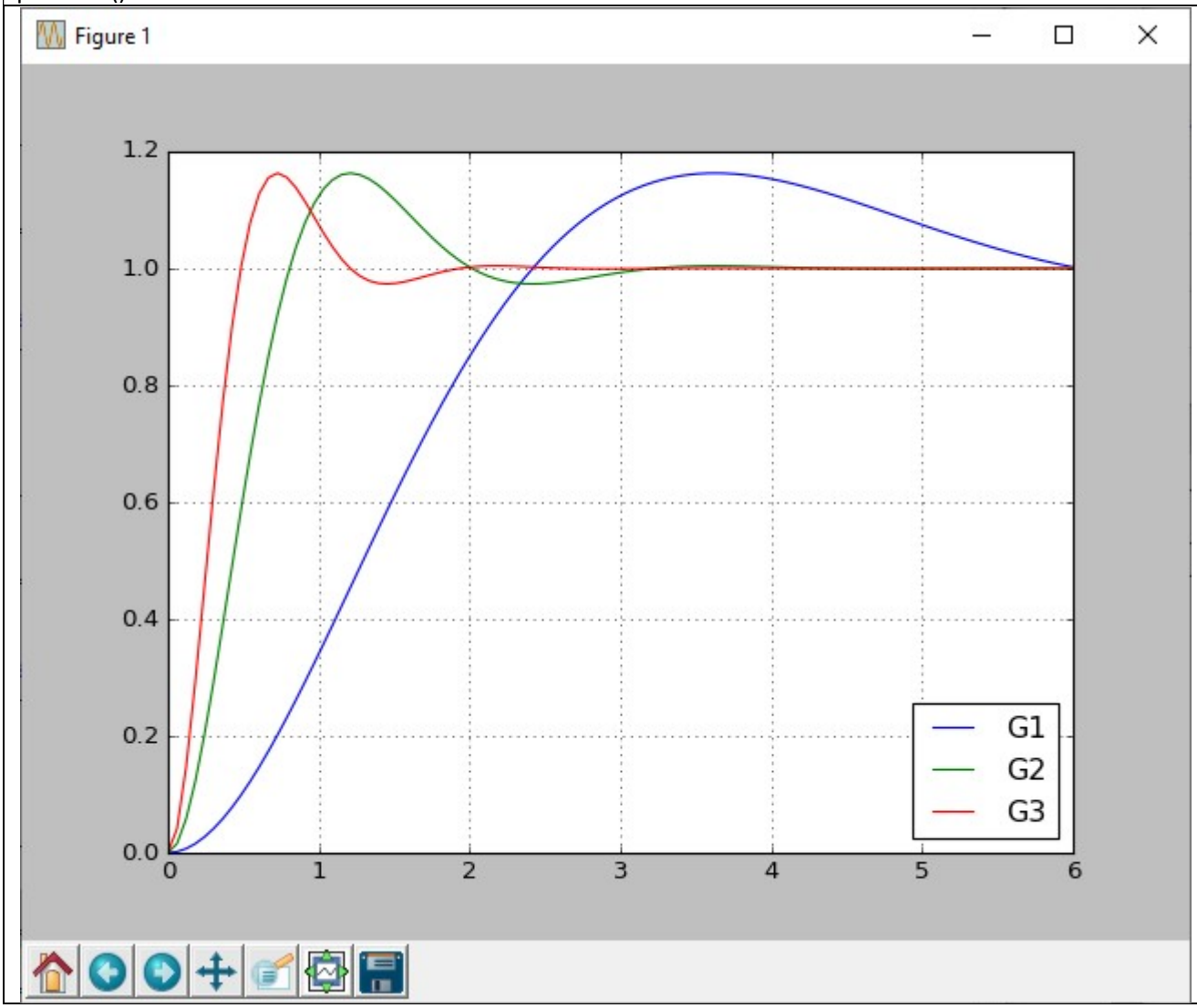
Esempio sistemi del secondo ordine – stesso ζ:

$$G1(s) = \frac{1}{s^2 + s + 1} \qquad G2(s) = \frac{9}{s^2 + 3s + 9} \qquad G3(s) = \frac{25}{s^2 + 5s + 25}$$

Visualizzare le risposte al gradino

```
import control
import matplotlib.pyplot as plt
import math
import numpy as np
from control.matlab import *

s = control.tf('s')
t = np.linspace(0, 6, 100)
G1=1/(s**2+s+1)
G2=9/(s**2+3*s+9)
G3=25/(s**2+5*s+25)
_,y1=control.step_response(G1,T=t)
_,y2=control.step_response(G2,T=t)
_,y3=control.step_response(G3,T=t)
plt.plot(t,y1,label="G1")
```

```
  plt.plot(t,y2,label="G2")
  plt.plot(t,y3,label="G3")
  plt.legend(loc="lower right")
  plt.grid()
plt.show()
```

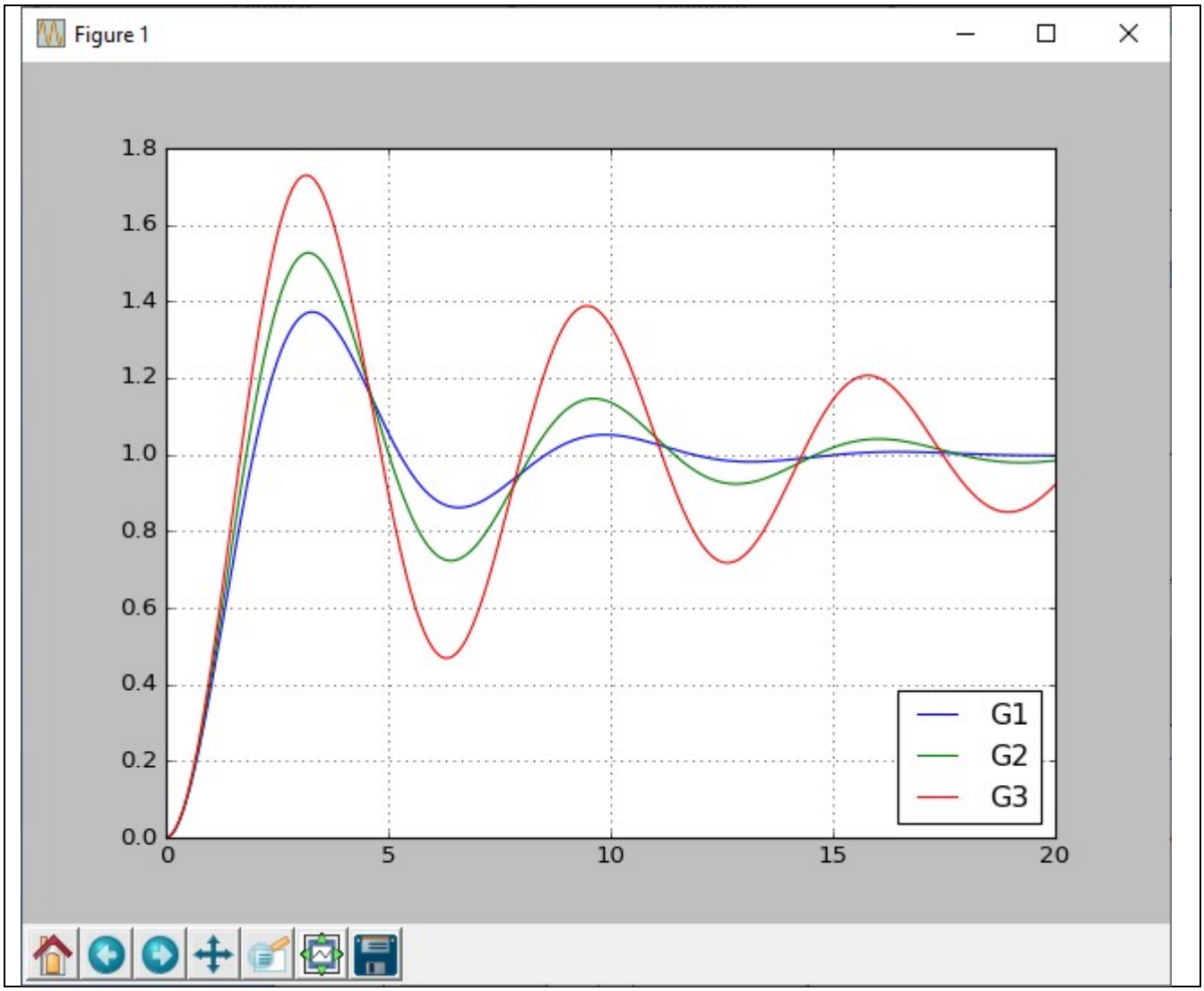


Esempio sistemi del secondo ordine - stesso ω_n:

$$G1(s) = \frac{1}{s^2 + 0.6s + 1} \qquad G2(s) = \frac{1}{s^2 + 0.4s + 1} \qquad G3(s) = \frac{1}{s^2 + 0.2s + 1}$$

Visualizzare le risposte al gradino

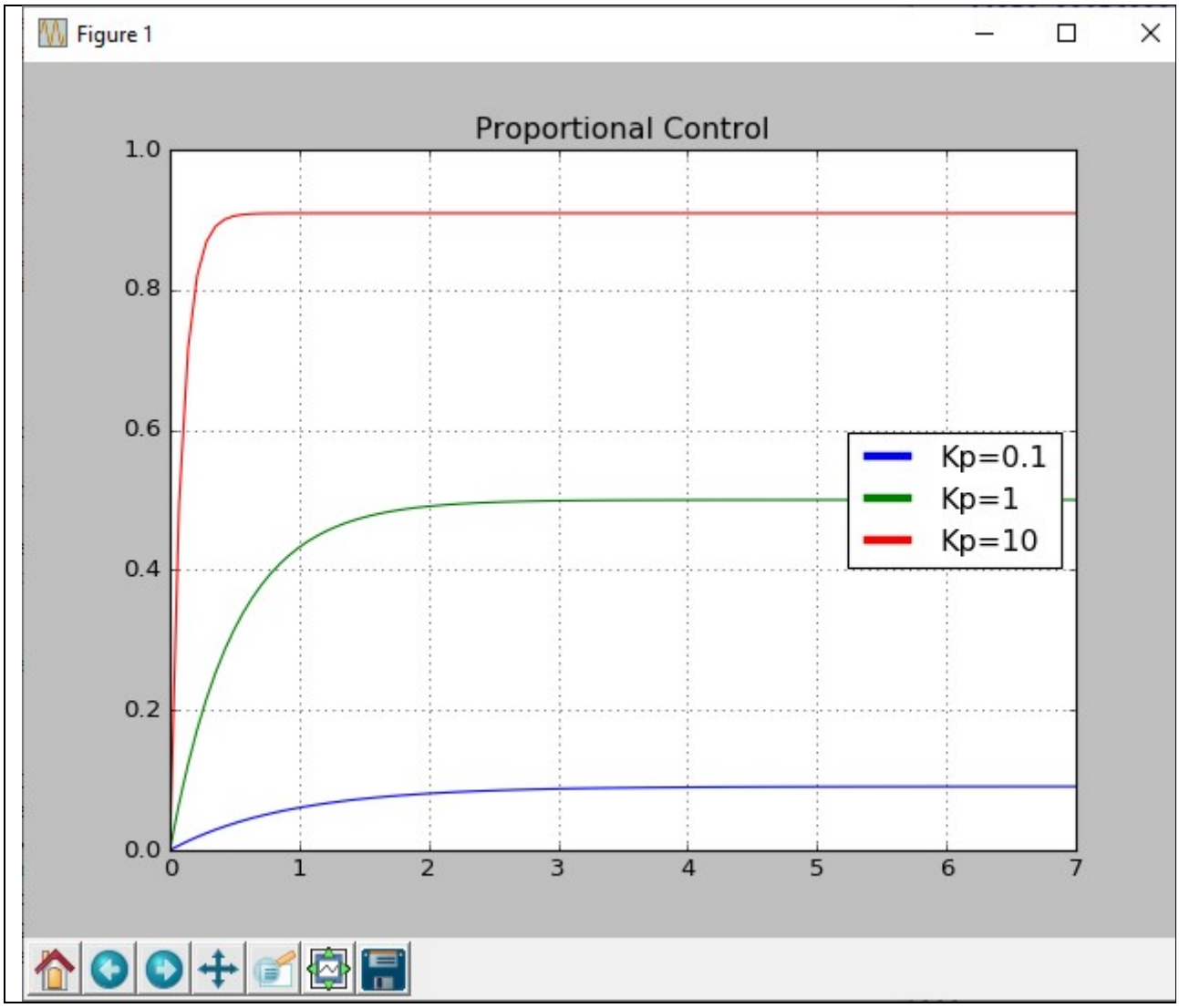
```
s = control.tf('s')
t = np.linspace(0, 20, 1000)
G1=1/(s**2+0.6*s+1)
G2=1/(s**2+0.4*s+1)
G3=1/(s**2+0.2*s+1)
_,y1=control.step_response(G1,T=t)
_,y2=control.step_response(G2,T=t)
_,y3=control.step_response(G3,T=t)
plt.plot(t,y1,label="G1")
plt.plot(t,y2,label="G2")
plt.plot(t,y3,label="G3")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Controllori PID

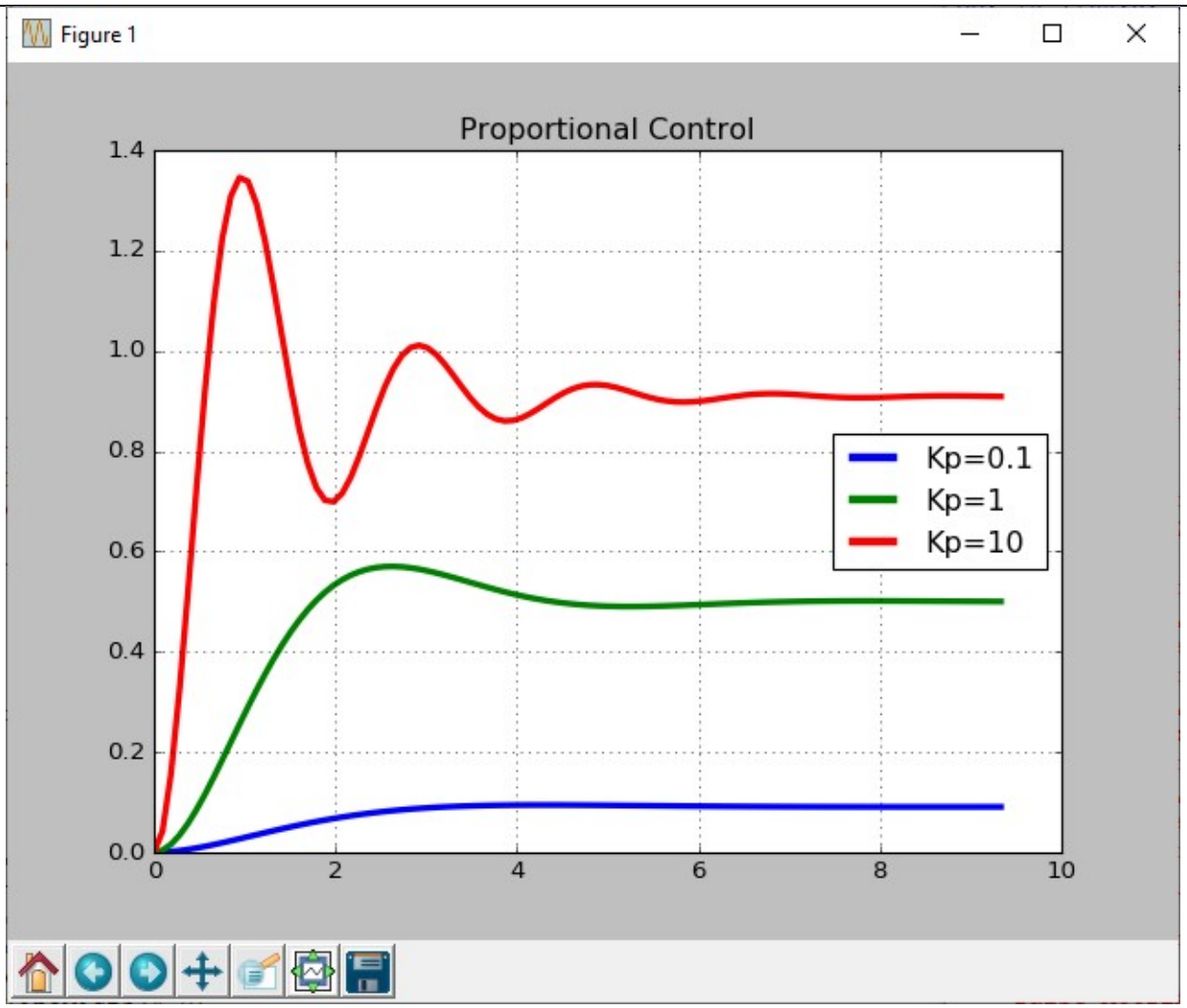
Controllo Peoporzionale con poli complessi e coniugati
Simulare l'azione proporzionale nel caso di poli reali, con ingresso scalino unitario e Kp=0.1; Kp=1; Kp=!0

import control
import matplotlib.pyplot as plt
import math
import numpy as np
 	
import control.matlab as mlab
from scipy import signal
 	
Kp=[0.1,1,10]
s = control.tf('s')
G = 1/(1+s)
 	
for k in Kp:
  GR=k*G/(1+k*G)
  t,y=control.step_response(GR)
  plt.plot(t,y,label='Kp='+str(k))
plt.title('Proportional Control')
plt.legend(loc="center right")
plt.grid()
plt.show()



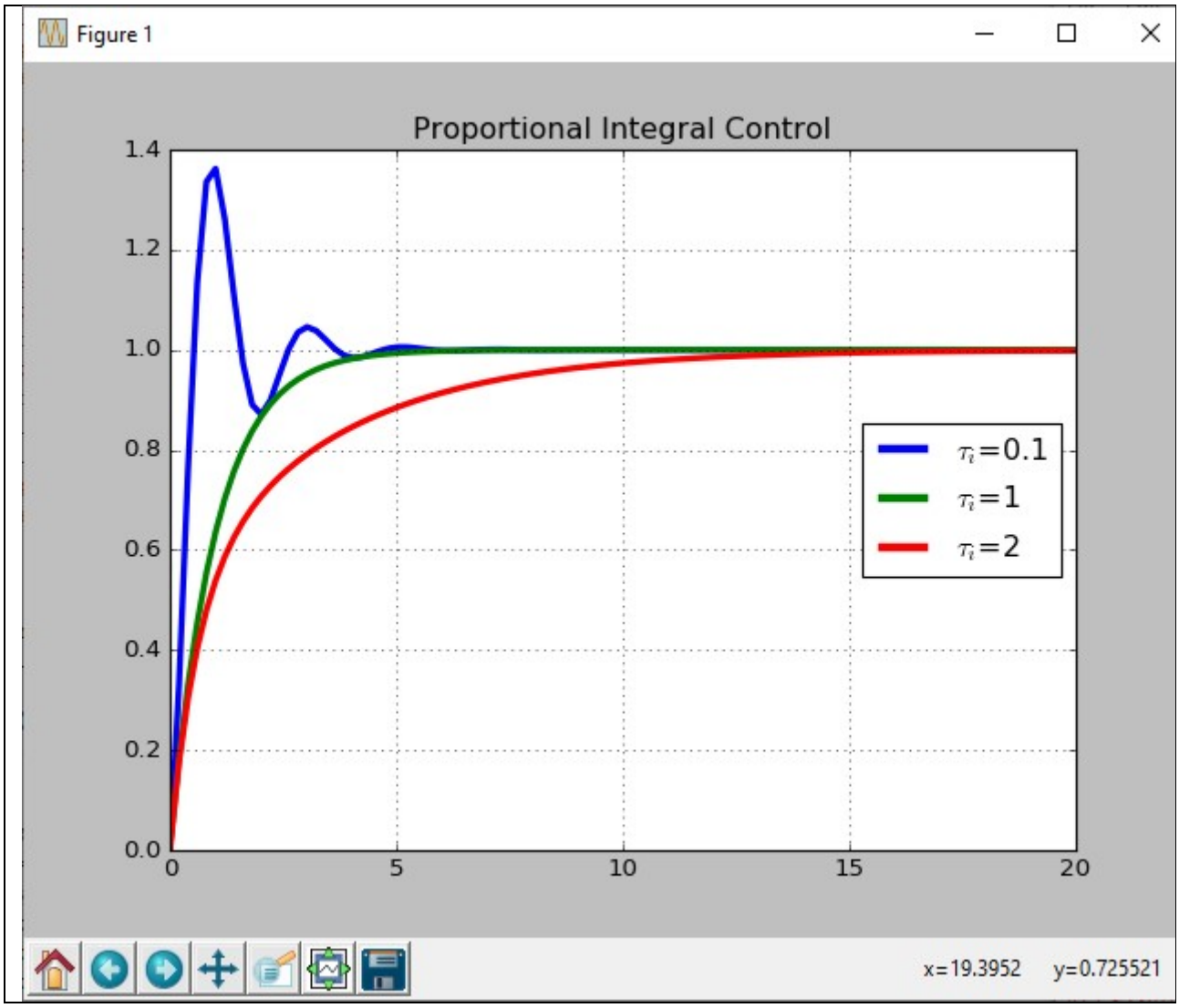
Simulare l'azione proporzionale nel caso di poli complessi coniugati, con ingresso scalino unitario e Kp=0.1;Kp=1;Kp=10

```
Kp=1
s = control.tf('s')
Ti=[0.1,1,2]

G = 1/(1+s)
t = np.linspace(0, 20, 100)

for ti in Ti:
   C=Kp*(1+1/(ti*s))
   GR=C*G/(1+C*G)
   _,y=control.step_response(GR,T=t)
   plt.plot(t,y,label=r'$\tau_{i}$='+str(ti),linewidth=3)
plt.title('Proportional Integral Control')
leg=plt.legend(loc="center right")
for line in leg.get_lines():
   line.set_linewidth(4.0)
plt.grid()
plt.show()
```



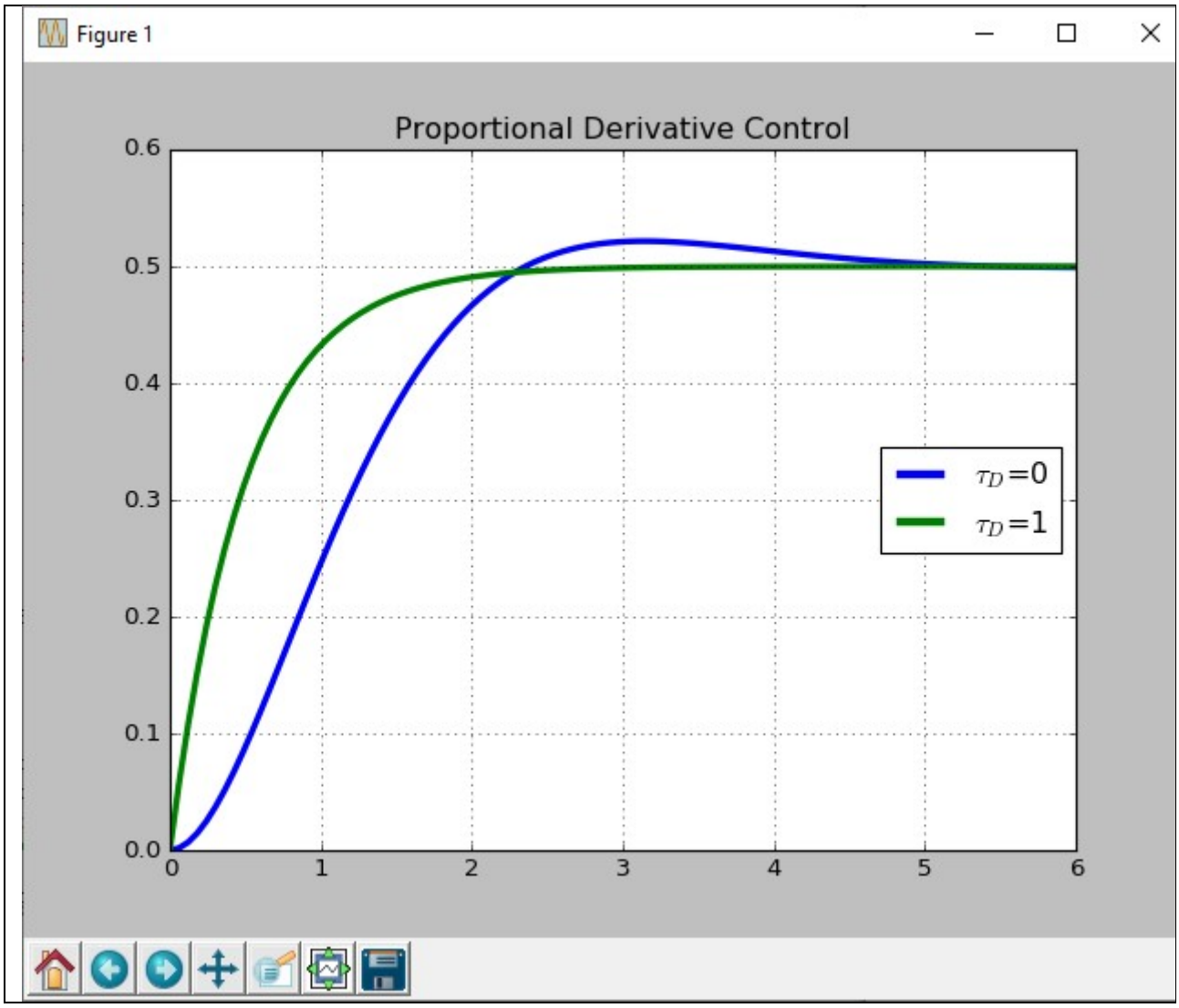
Azione proporzionale integrale e proporzionale derivativa
Simulare l'azione proporzionale-integrale nel caso di poli reali, con ingresso scalino unitario e Kp=1;τ_1=0.1; τ_1=1; τ_1=2

Kp=1
s = control.tf('s')
Ti=[0.1,1,2]
 	
G = 1/(1+s)
t = np.linspace(0, 20, 100)
 	
for ti in Ti:
  C=Kp*(1+1/(ti*s))
  GR=C*G/(1+C*G)
  _,y=control.step_response(GR,T=t)
  plt.plot(t,y,label=r'τ_{i}='+str(ti))
plt.title('Proportional Integral Control')
plt.legend(loc="center right")
plt.grid()
plt.show()

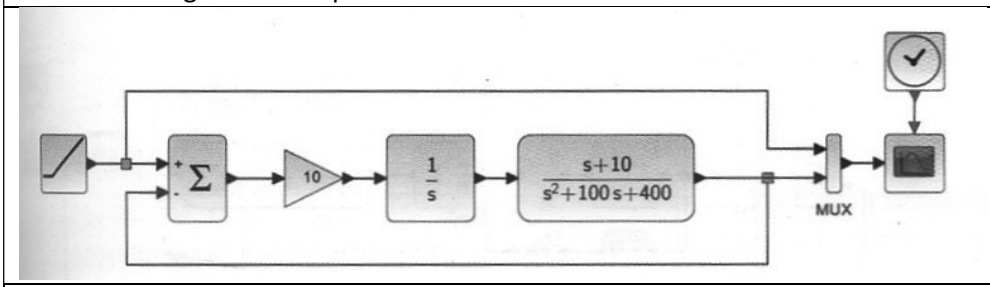


Simulare l'azione proporzionale-derivativa nel caso di poli reali, con ingresso scalino unitario e Kp=1;τ_D=0; τ_D=1

```
Kp=1
s = control.tf('s')
TD=[0,1]
G = 1/(s**2+2*s+1)
t = np.linspace(0, 6, 100)

for tD in TD:
    C=Kp*(1+tD*s)
    GR=C*G/(1+C*G)
    _,y=control.step_response(GR,T=t)
    plt.plot(t,y,label=r'$\tau_{D}$='+str(tD),linewidth=3)
plt.title('Proportional Derivative Control')
leg=plt.legend(loc="center right")
for line in leg.get_lines():
    line.set_linewidth(4.0)
plt.grid()
plt.show()
```

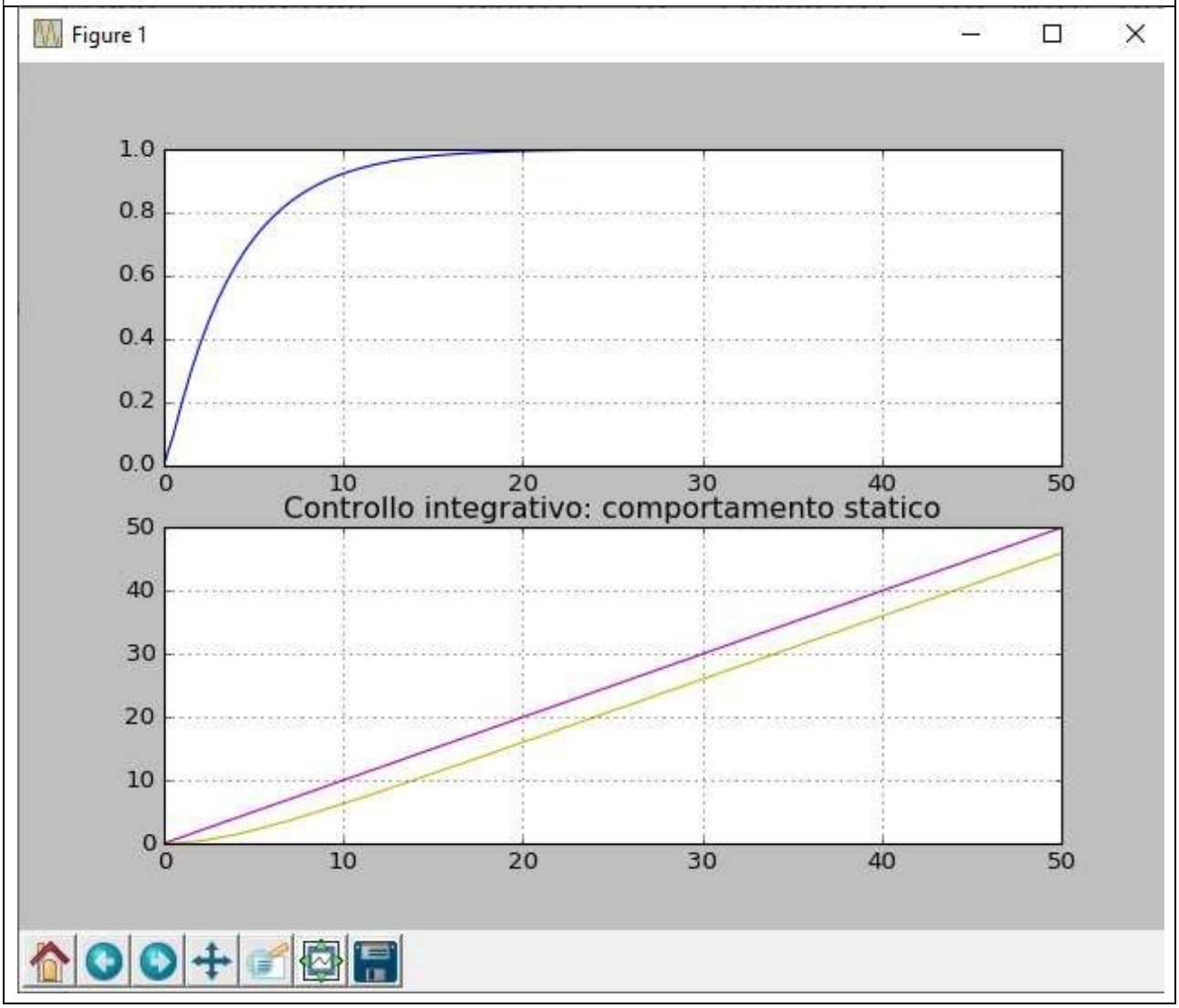


Controllo integrativo: comportamento statico

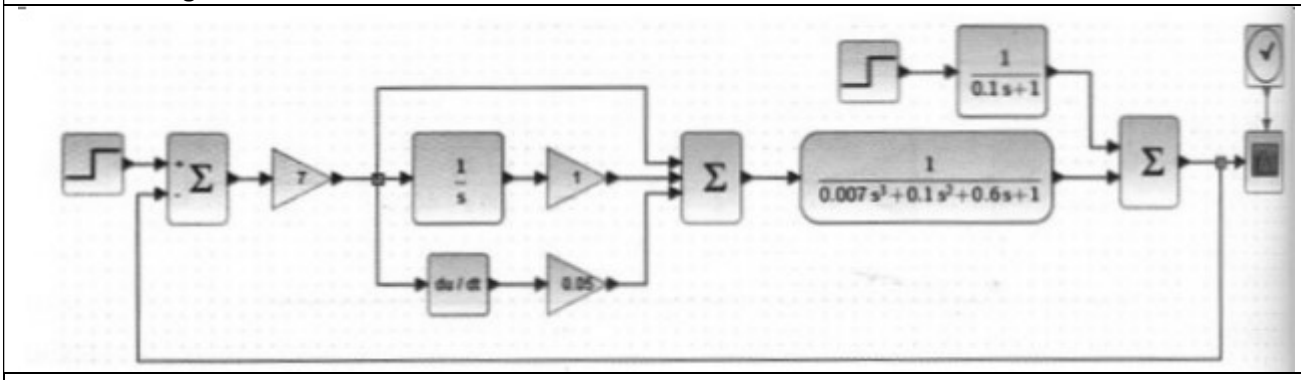



```
Kp=10
s = control.tf('s')
TD=0
G = (s+10)/(s**2+100*s+400)
t = np.linspace(0, 50, 100)
u=t

C=Kp*(1/s)
GR=C*G/(1+C*G)
n,d= control.tfdata(GR)
GR= control.tf(n,d)

plt.subplot(2, 1, 1)
_,y=control.step_response(GR,T=t)
plt.grid()
plt.plot(t,y)
```

```
plt.subplot(2, 1, 2)
y,_,x=mlab.lsim(GR, u, t)
plt.plot(t,y,'y',t,u,'m')

plt.title('Controllo integrativo: comportamento statico')
plt.grid()
plt.show()
```

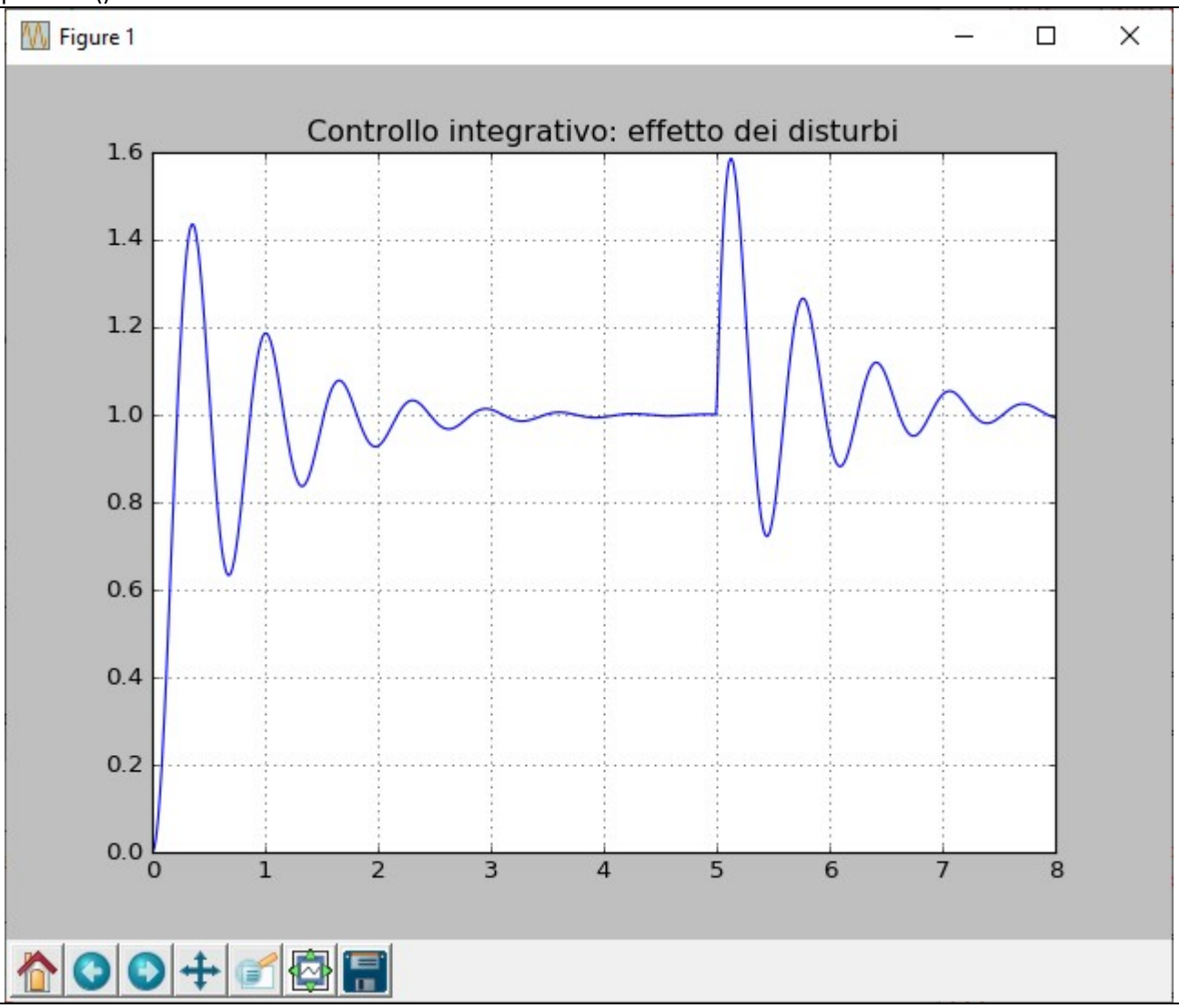


Controllo integrativo- effetto dei disturbi



```
def unit_step(t,n0,a,b,nP):
    t0=n0*((nP+0)/(b-a))
    u=[0]*t0;u.extend([1]*(t.size-t0))
    return u
```

```
Kp=7
s = control.tf('s')

C = Kp*(1+1*(1/s)+0.05*s)
G = 1/(0.007*s**3+0.1*s**2+0.6*s+1)
G=G*C
D=1/(0.1*s+1)
t = np.linspace(0, 8, 1000)
H=1
R=1
d=1
GR=(G*R)/(1+G*H)
GD=D*d/(1+G*H)

_,y1=control.step_response(GR,T=t)
d=unit_step(t,5,int(t[0]),int(t[t.size-1]),t.size)
yd,_,_= mlab.lsim(GD, d, t)
y=y1+yd

plt.plot(t,y)
plt.title('Controllo integrativo: effetto dei disturbi')
plt.grid()
plt.show()
```

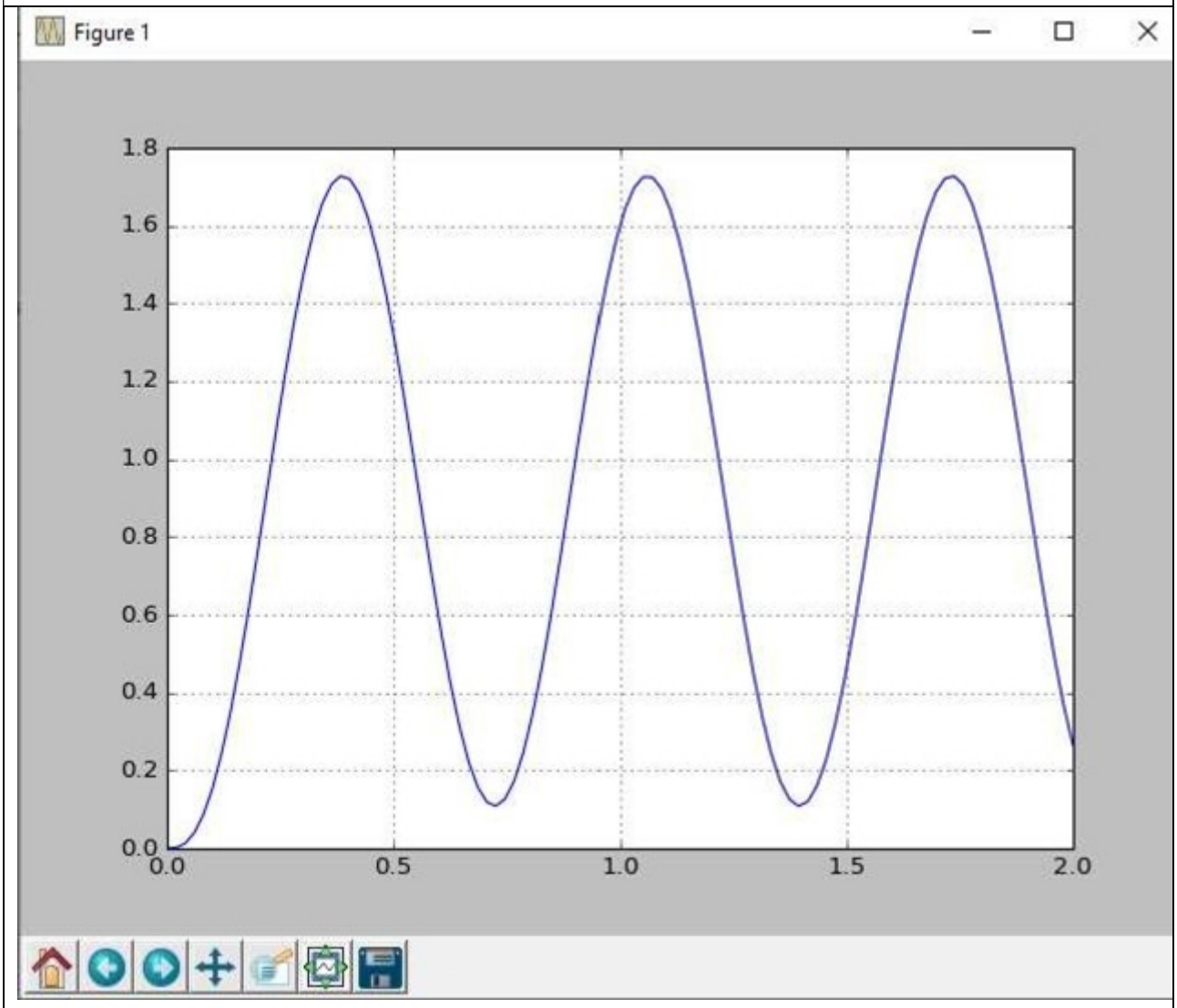


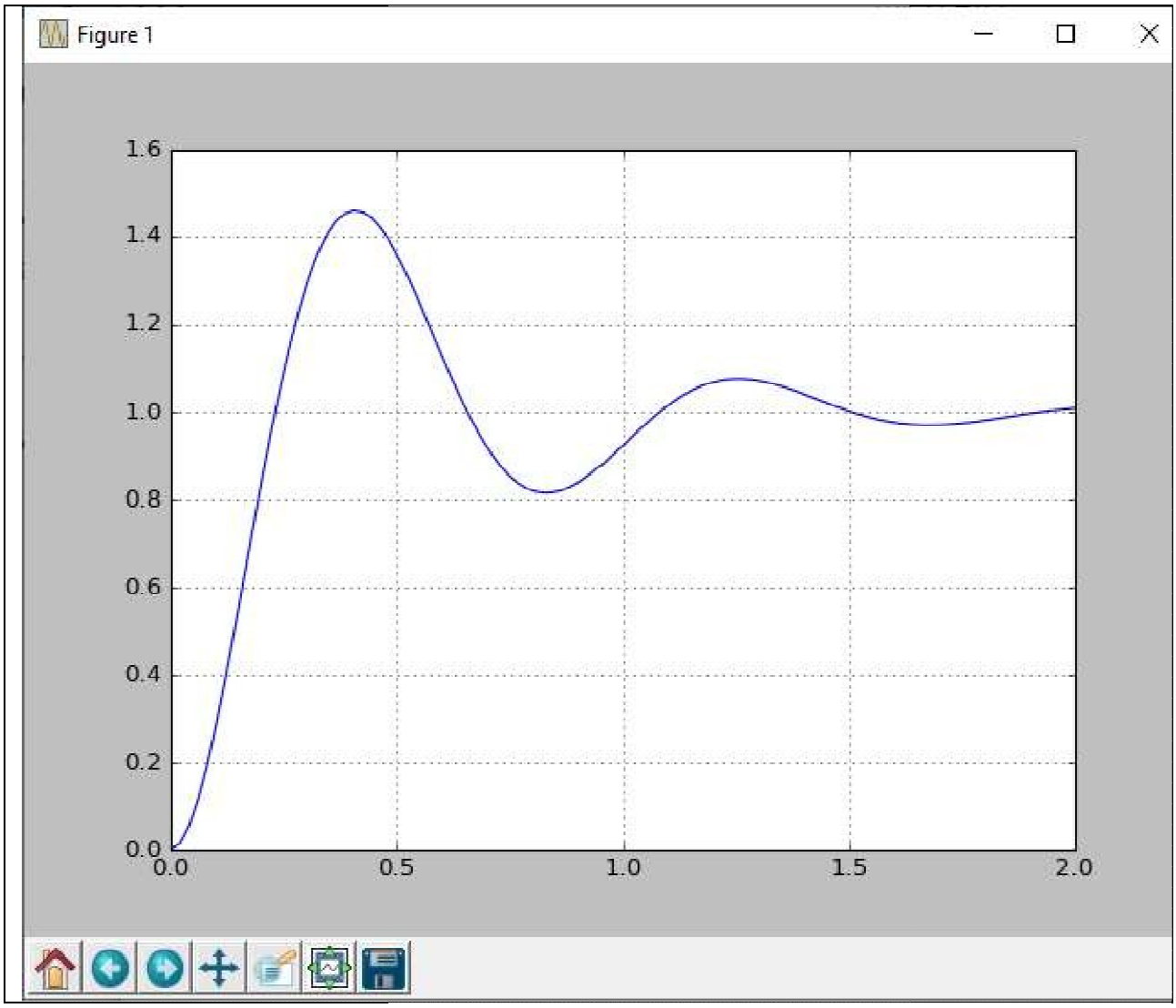
Controllo PID metodo di Ziegler-Nichols

Stabilizzare in retroazione unitaria la seguente fdt con il metodo di Ziegler-Nichols:

$$G(s) = \frac{1}{(1 + 0.1 \cdot s) \cdot (1 + 0.2 \cdot s) \cdot (1 + 0.4 \cdot s)}$$

```
s = control.tf('s')
Kp=11.25
C=Kp
G=1/((1+0.1*s)*(1+0.2*s)*(1+0.4*s))
GR=(C*G)/(1+C*G)
t = np.linspace(0, 2, 100)
_,y=control.step_response(GR,T=t)
plt.plot(t,y)
plt.grid()
plt.show()
```





PID - Verificare l'effetto della stabilizzazione

```
s = control.tf('s')
Kp=6.75
G=1/((1+0.1*s)*(1+0.2*s)*(1+0.4*s))
print 'G:',G

T=0.67
TI=0.335
TD=0.084

C=Kp*(1+1/(TI*s)+TD*s)
print 'C:',C
CG=control.series(C,G)
GR=control.feedback(CG,1)
print "GR:",GR

t = np.linspace(0, 2, 100)
_,y=control.step_response(GR,T=t)
plt.plot(t,y)
plt.grid()
plt.show()

>>>
```

G:

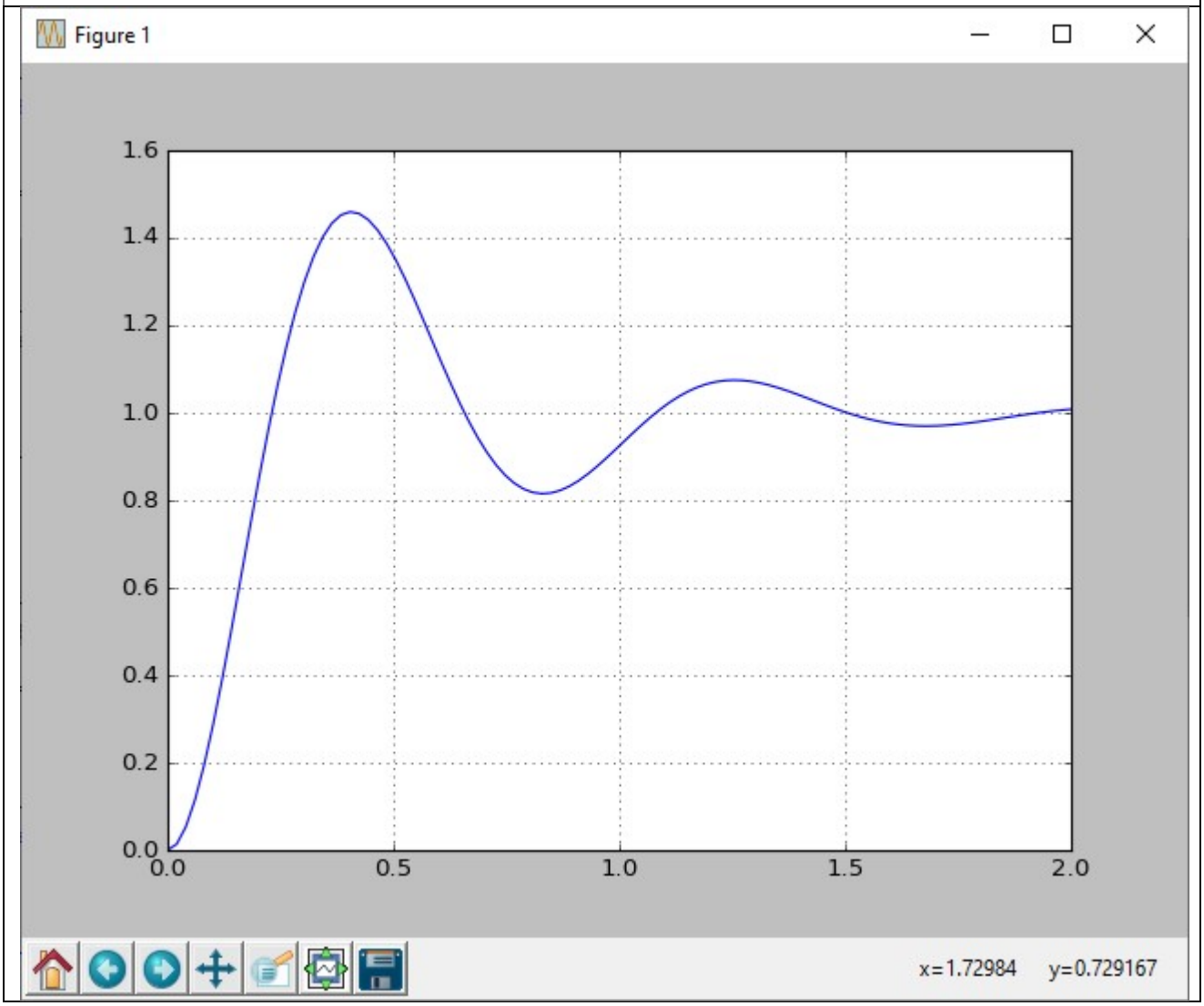
$$\frac{1}{0.008\,s^3 + 0.14\,s^2 + 0.7\,s + 1}$$

C:

$$\frac{0.1899\,s^2 + 2.261\,s + 6.75}{0.335\,s}$$

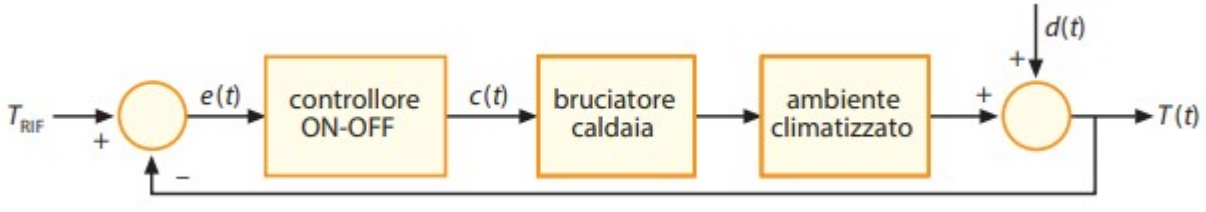
GR:

$$\frac{0.1899\,s^2 + 2.261\,s + 6.75}{0.00268\,s^4 + 0.0469\,s^3 + 0.4244\,s^2 + 2.596\,s + 6.75}$$

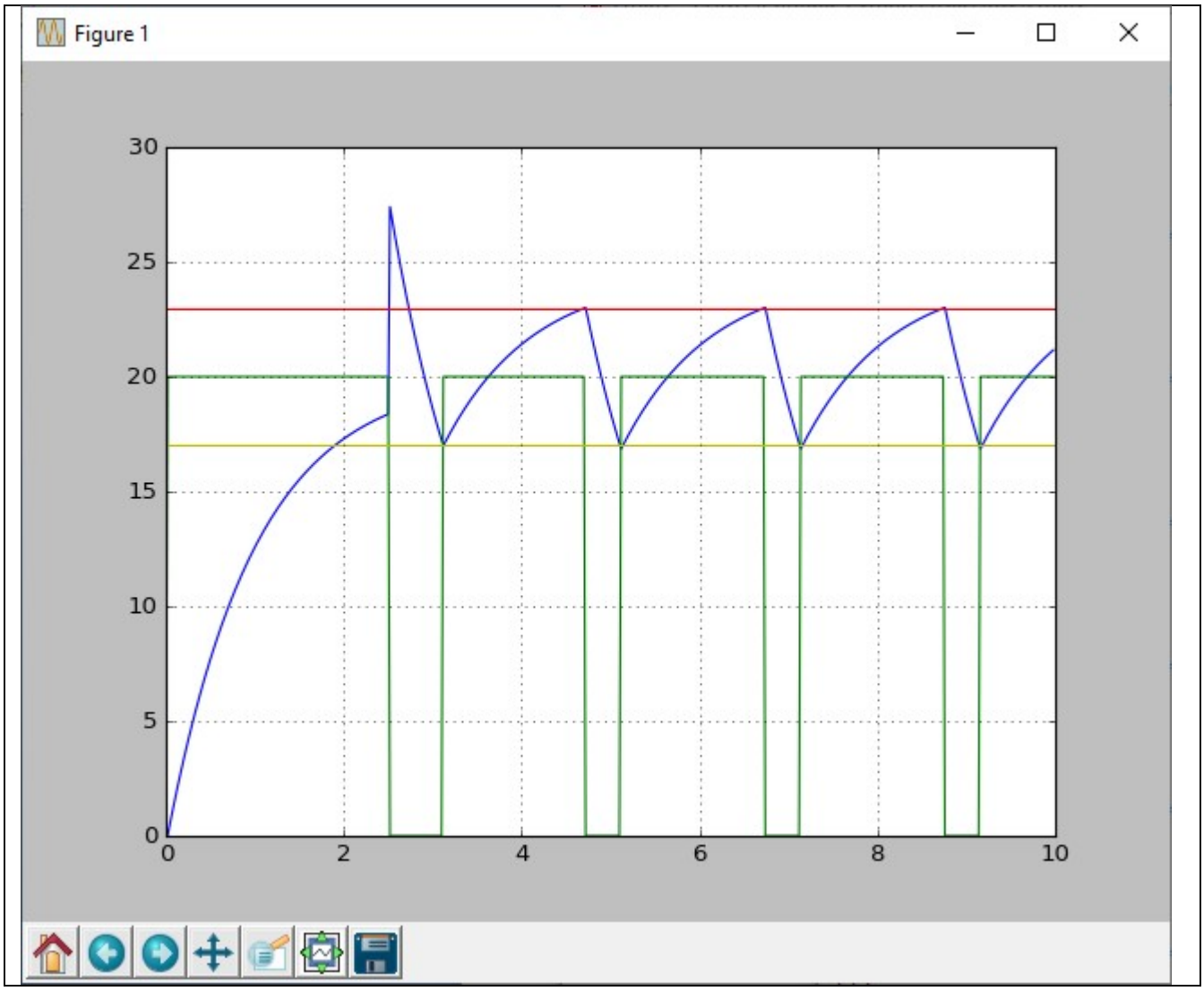


CONTROLLO ON-OFF

Esempio di controllo ON-OFF di temperatura per la climatizzazione di un ambiente. Scopo del controllo è di mantenere il livello di temperatura T(t) di unastanza entro i margini prestabiliti, rappresentati da una soglia inferiore TINF e una soglia superiore TSUP.



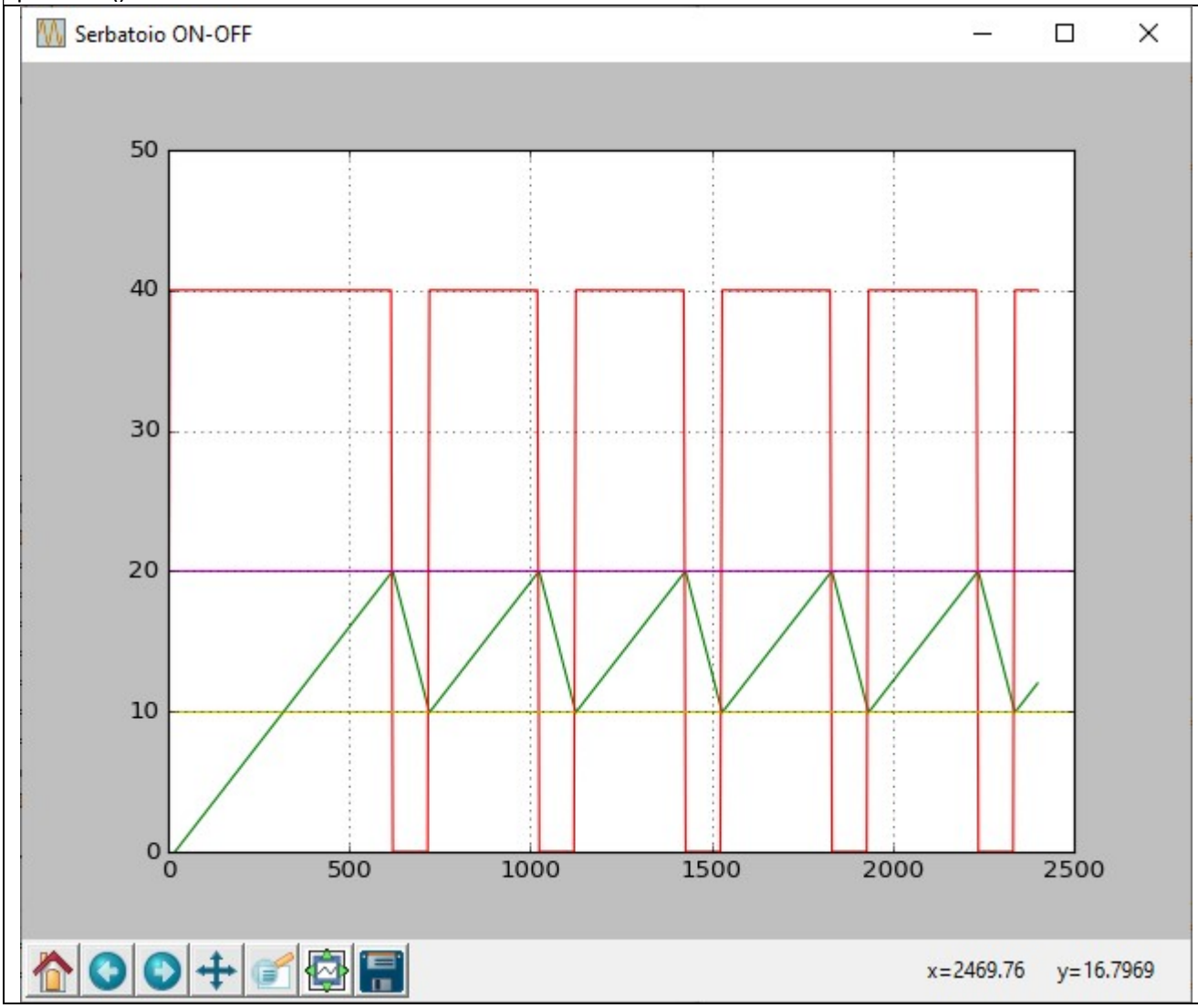
```
N=500
t1=0.
t2=10.
K=1
dt=(t2-t1)/N
TRIF=20
T0=0
TSUP=TRIF+3
TINF=TRIF-3
TRIF = (TSUP + TINF)/2.
T=[0]*N
To=[0]*N
Td=[0]*N
NOISE=9
x=[0]*N
t=0
pos = int(math.log10(abs(dt)))
n=abs(pos)+1
tau=dt*10**(n)
t=[0]*N
ON=TRIF
OFF=0
c=ON
for i in range (0,N):
   if(i>N/4):
      Td[i]=NOISE
   if i>0:
      T[i]=(((c-To[i-1])*K+c)-T[i-1])*(dt/tau)+T[i-1]
   else:
      T[i]=T0
   To[i]=T[i]+Td[i]
   t[i]=i*dt
   if(i*dt>6):
      x[i]=0
   if(i>0):
      if(To[i]>=TINF and To[i]<TSUP and x[i-1]==ON):
         x[i]=ON
      if(To[i]>TINF and To[i]<TSUP and x[i-1]==OFF):
         x[i]=OFF
      if(To[i]<=TINF):
         x[i]=ON
      if(To[i]>=TSUP):
         x[i]=OFF
      if(To[i]==TRIF):
         x[i]=OFF
   c=x[i]

plt.plot(t,To)
plt.plot(t,x)
plt.axhline(y=TSUP, color='r', linestyle='-')
plt.axhline(y=TINF, color='y', linestyle='-')
plt.grid()
plt.show()
```



Esempio – Controllo ON-OFF serbatoio

```
qi_def=40
qi=qi_def
qu=30.
h0=0
A=20.
h=h0
hsup=20.
hinf=10.
hrif=(hinf+hsup)/2.
v=0
t1=0.
t2=(60.*60.)/1.5
N=500
dt=(t2-t1)/N
t = np.linspace(t1, t2, N)
V=[]
H=[]
ON=1
OFF=0
tau=15.
for i in range(t.size):
    if i>0:
        h=h+((qi*v-qu)*(dt/tau))/A
```

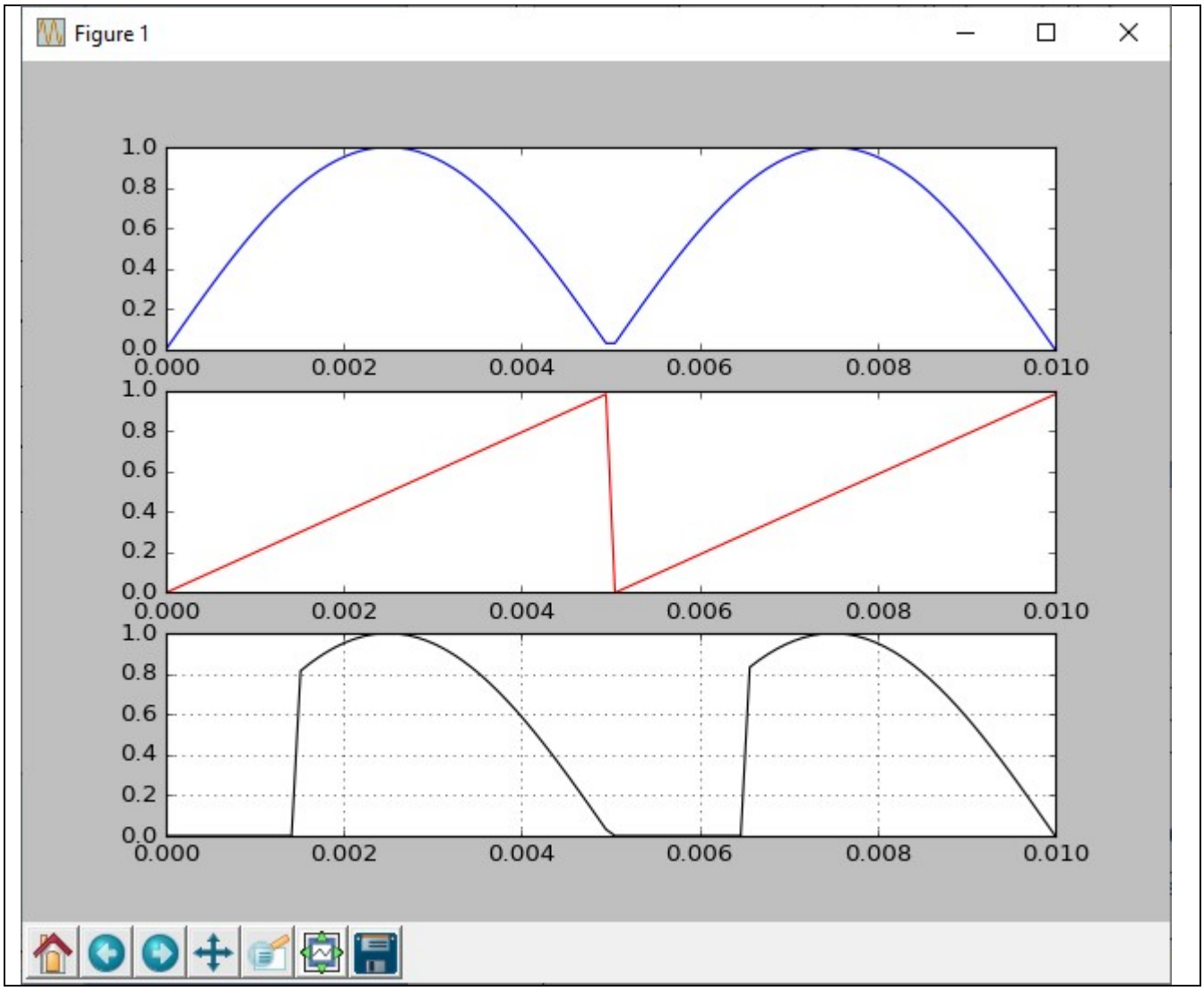
```
    else:
        h=h0

    if(i>0):
        if h>=hinf and h<hsup and v==ON:
            v=ON
        if h>hinf and h<hsup and v==OFF:
            v=OFF
        if(h<=hinf):
            v=ON
        if(h>=hsup):
            v=OFF
        if(h==hrif):
            v=OFF
    V.append(qi*v)
    H.append(h)
plt.gcf().canvas.set_window_title('Serbatoio ON-OFF')
plt.plot(t,H,'g',t,V,'r')
plt.ylim(0,50)
plt.axhline(y=hsup, color='m', linestyle='-')
plt.axhline(y=hinf, color='y', linestyle='-')
plt.grid()
plt.show()
```



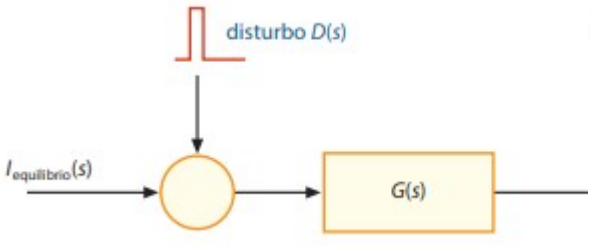
CONTROLLO DI POTENZA

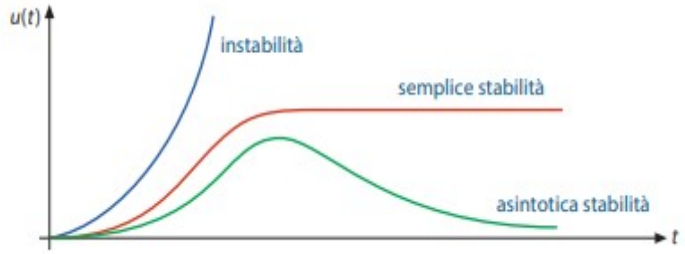
Parzializzazione di semionda

```
N=100
f=100.
t1=0.
t2=1./f
t = np.linspace(t1, t2, N)
X=[]
T=[]
dt=(t2-t1)/N
thresh=0.3
x=np.abs(np.sin(2*np.pi*f*t))
r=[(2./N)*(i%(N/2)) for i in range(N)]
#y = [x if i >= thresh  else i for i in r]
#y =[j if i >= thresh  else 0 for i in r for j in x]
y=[0]*len(x)
for i in range(len(r)):
   if r[i]>=thresh:
      y[i]=x[i]
   else:
      y[i]=0
plt.subplot(3, 1, 1)
plt.plot(t,x,'b')
plt.subplot(3, 1, 2)
plt.plot(t,r,'r')
plt.subplot(3, 1, 3)
plt.plot(t,y,'k')
plt.grid()
plt.show()
```



STABILITA'





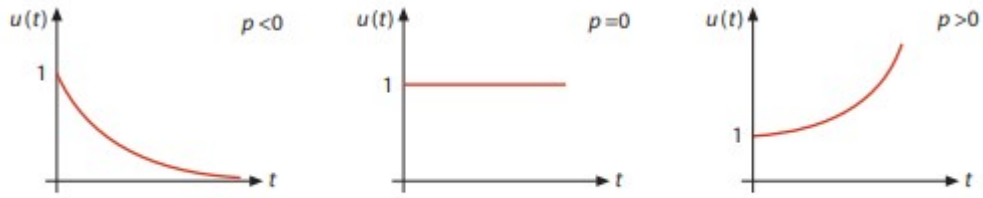
- **curva asintotica stabilità**: $u(t)$ tende a 0 per $t \to \infty$;
- **curva semplice stabilità**: $u(t)$ tende a un valore diverso d
- **curva instabilità**: $u(t)$ tende a ∞ per $t \to \infty$.

Assumiamo che il disturbo sia un impulso matematico ideale δ(t). Ciò non toglie nulla alla generalità della dimostrazione in quanto, per la sovrapposizione degli effetti, un generico disturbo può essere ricavato come combinazione di impulsi ideali.

$$G(s) = \frac{U(s)}{D(s)} = \frac{1}{(s-p_1)\cdot(s-p_2)} \qquad U(s) = D(s)\cdot G(s) = L\{\delta(t)\}\cdot G(s) = \frac{1}{(s-p_1)\cdot(s-p_2)}$$

Antitrasformando:

$$U(s) = \frac{A}{(s-p_1)} + \frac{B}{(s-p_2)} \overset{L^{-1}}{\Longrightarrow} u(t) = A\cdot e^{p_1 \cdot t} + B\cdot e^{p_2 \cdot t}$$



esponenziale decrescente tendente a 0 per $t \to \infty$ | funzione unitaria costante | esponenziale crescente tendente a ∞ per $t \to \infty$

In questo calcolo non è stato previsto un caso particolare: quello di due o più poli nulli. Con sideriamo per esempio la seguente f.d.t., avente solo due poli nulli:

$$G(s) = \frac{1}{s^2} = \frac{1}{(s - p_1) \cdot (s - p_2)}$$

Con risposta all'impulso:

$$U(s) = D(s) \cdot G(s) = L\{\delta(t)\} \cdot \frac{1}{s^2} \overset{L^{-1}}{\Longrightarrow} u(t) = t$$

Considerando ora che la risposta all'impulso è data dalla somma del contributo di ciascun polo, possiamo osservare che:

• se il sistema presenta poli tutti negativi, la ri sposta all'impulso è la somma di diversi esponenziali decrescenti, quindi u(t) tende a 0 per t → ∞, come nella curva di asintotica stabilità;

• se il sistema presenta almeno un polo positivo, il corrispondente esponenziale è crescente, quindi, anche se tutti gli altri esponenziali sono decrescenti, u(t) tende a ∞ per t → ∞, come nella curva di instabilità;

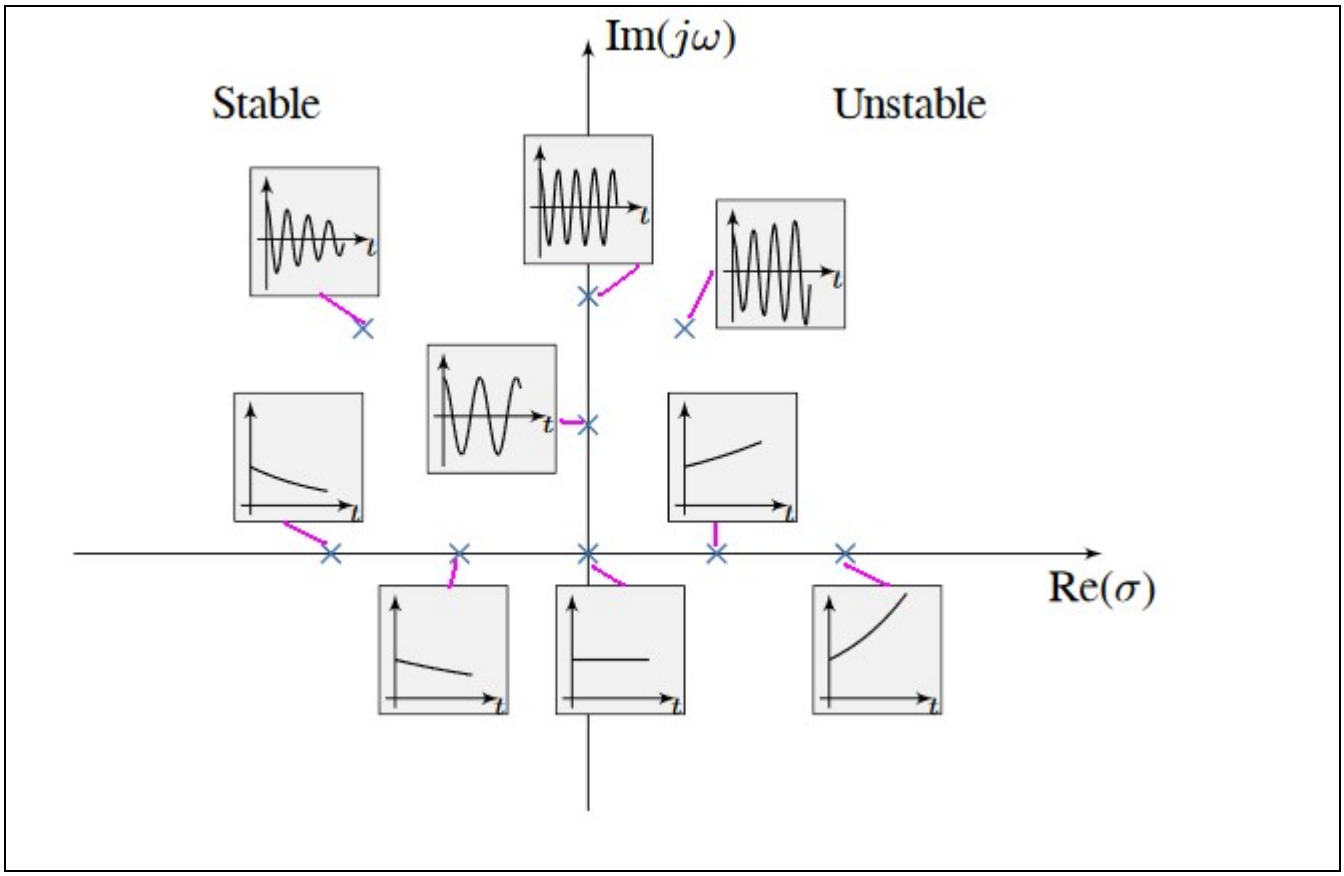
• se il sistema presenta poli tutti negativi, tranne uno nullo, u(t) tende a un valore diverso da 0 per t → ∞ (quello dovuto appunto al polo nullo), come nella curva di semplice sta bilità;

• se il sistema presenta due o più poli nulli, questi danno luogo a funzioni u(t) tendenti a ∞ per t → ∞, come nella curva di instabilità.

In conclusione possiamo enunciare le seguenti condizioni necessarie e sufficienti:

1. condizione necessaria e sufficiente per l'asintotica stabilità: tutti i poli negativi;
2. condizione necessaria e sufficiente per la semplice stabilità: tutti i poli negativi tranne uno solo nullo;
3. condizione necessaria e sufficiente per l'instabilità: almeno un polo positivo e/o più di un polo nullo

Risposta all'impulso – posizione poli



Esempio - SEGNO DEI POLI E STABILITÀ

```
def CheckStability(G):
    poles=control.pole(G)
    print poles,
    np=0
    pp=0
    zp=0
    for  p in poles:
      if p<0:
         np+=1
      if p==0:
         zp+=1
      if p>0:
         pp+=1
    if np==poles.size:
       print "asintotica stabilita'"
    if pp>0:
       print "instabile"
    if(zp>1):
       print "instabile"
    if zp==1:
       print "semplice stabilita"

s = control.tf('s')
G = 1/s**2
CheckStability(G)
G=1/(s**2+6*s+8)
CheckStability(G)
G=1/(s**2-1)
CheckStability(G)
>>>
```

```
[0. 0.] instabile
[-4. -2.] asintotica stabilita'
[-1.  1.] instabile
```

Esempi

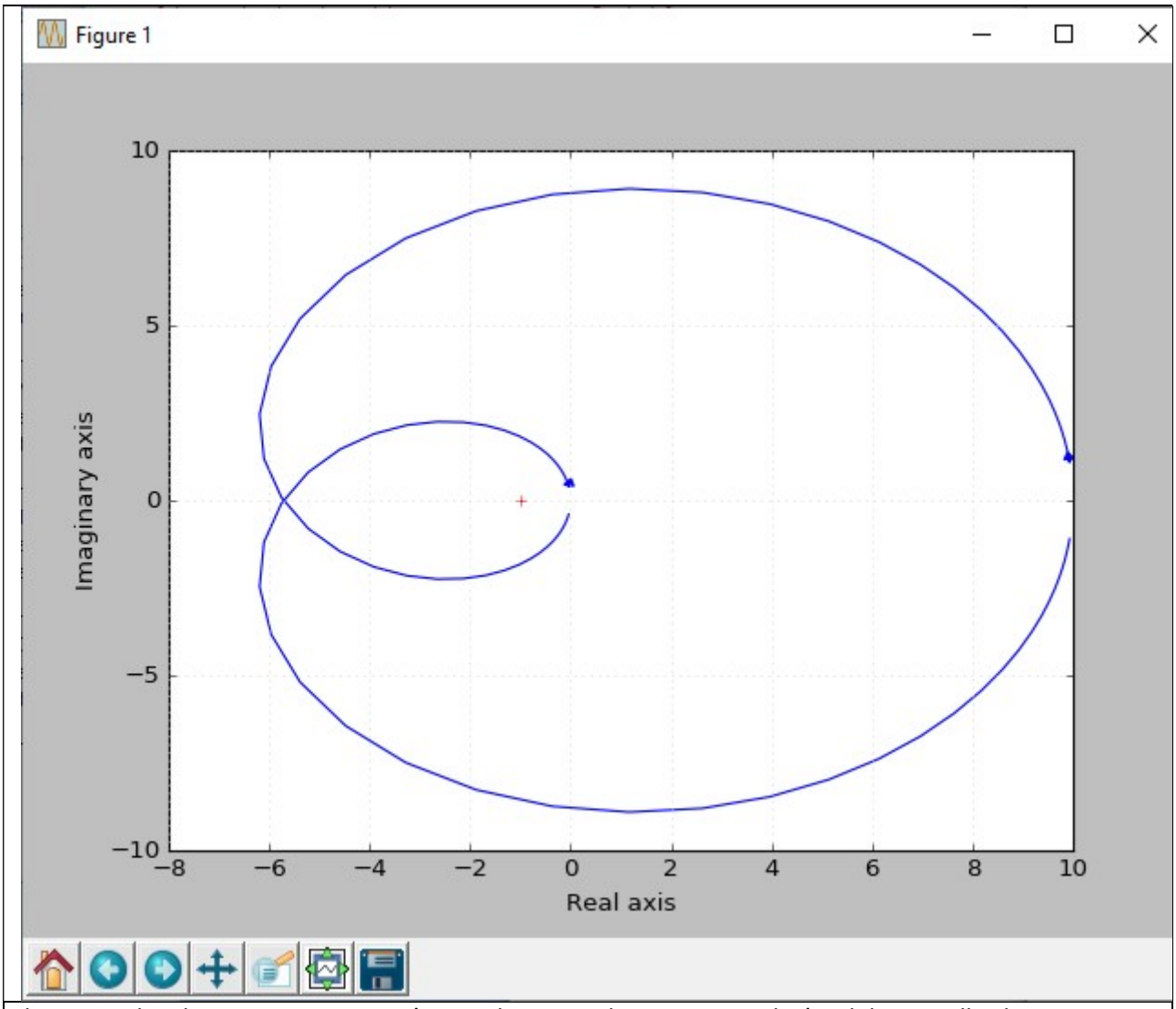
```
s = control.tf('s')
G = (s+2)/((s+1)*(s+3))
CheckStability(G)
G=(s+2)/(s*(2*s+1)*(2*s+3))
CheckStability(G)
G=(s+2)/(s**3+3*s**2+3*s+1)
CheckStability(G)
G=(s+2)/(s**3+5*s**2-2*s-24)
CheckStability(G)
>>>
[-3. -1.] asintotica stabilita'
[-1.5 -0.5  0. ] semplice stabilita
[-0.99999588+7.12759326e-06j -0.99999588-7.12759326e-06j
 -1.00000823+0.00000000e+00j] asintotica stabilita'
[-4. -3.  2.] instabile
```

STABILITA' – CRITERIO DI NYQUIST

Es.- Verificare se la f.d.t. seguente e' stabile in anello chiuso:

$$G(s) = \frac{10 \cdot (1 + 4s)}{(1 - 2s) \cdot (1 - 5s)}$$

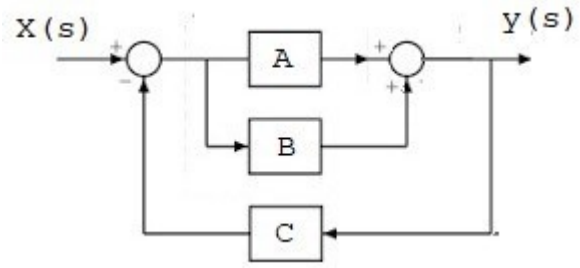
```
s = control.tf('s')
G = (10*(1+4*s))/((1-2*s)*(1-5*s))
p=control.pole(G)
print "zero(s):",control.zero(G),"\npole(s):",p
control.nyquist_plot(G)
plt.show()

jw = control.tf('s')
G = (10*(1+4*jw))/((1-2*jw)*(1-5*jw))
p=control.pole(G)
print "zero(s):",control.zero(G),"\npole(s):",p
control.nyquist_plot(G)
plt.show()
```



Il numero di poli con parte positiva e' P=2 e il numero di giri N=2 quindi e' stabile in anello chiuso.

STABILITA DI UNO SCHEMA A BLOCCHI

Verificare la stabilita' del seguente sistema, e calcolare la risposta al gradino unitario:



$A(s) = \frac{10}{s}$ $A(s) = \frac{5}{s+3}$ $A(s) = \frac{1}{10}$

```
s = control.tf('s')
A=10/s
B=5/(s+3)
C=1/(10+s*0)
H = control.parallel (A , B)
G=control.feedback(H,C)
print G,"\nz:",control.zero(G),"\np:",control.pole(G)
```

```
    150 s + 300
 ----------------------
 10 s^2 + 45 s + 30
```

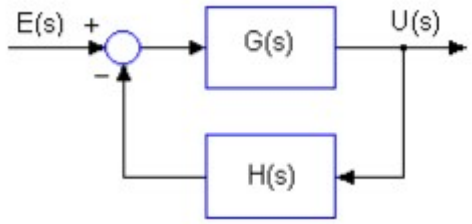
z: [-2.]
p: [-3.68614066 -0.81385934]

STABILIZZAZIONE – CRITERIO DI BODE

CRITERIO GENERALE DI STABILITÀ DI BODE

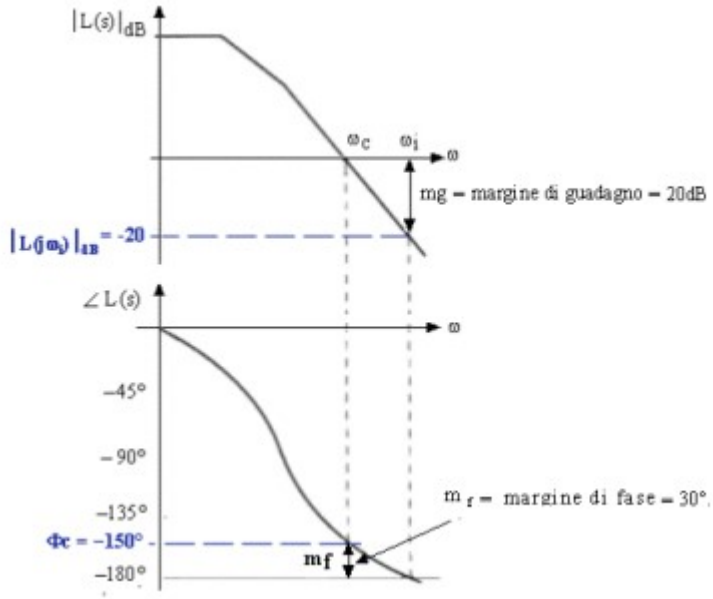
Il criterio di stabilità di Bode può essere applicato solo se la f.d.t. del sistema ad anello aperto è stabile e a sfasamento minimo, cioè la L(s) non deve avere poli e zeri a parte reale positiva.

Consideriamo un sistema ad anello chiuso



Il criterio di stabilità di Bode1 permette di determinare la stabilità quando è nota la funzione f.d.t. ad anello aperto cioè la L(s) = G(s)·H(s). Un sistema a catena chiusa è stabile: se lo sfasamento della f.d.t. ad anello aperto calcolato in corrispondenza della pulsazione critica2 è inferiore in valore assoluto a 180°

Esempio: consideriamo un sistema ad anello chiuso i cui diagrammi di Bode della f.d.t. ad anello aperto siano i seguenti



Il sistema è stabile in quanto in corrispondenza della pulsazione critica ωC, lo sfasamento in valore assoluto della f.d.t ad anello aperto è ⍰Φc⍰ =150°< 180°. Il margine di fase è m f =180°-|Φc|= 180° - |-150°| = 180°-150° = +30°. Il margine di guadagno è mg = 0 - |L(jωi)|d B = - |-20|= +20Db

CRITERIO SEMPLIFICATO DI STABILITÀ DI BODE

Un sistema ad anello chiuso è stabile de il diagramma del modulo del guadagno ad anello aperto interseca l'asse delle ascisse con una pendenza di – 40dB/dec. Se la pendenza è maggiore o uguale a –60dB/dec il sistema è sicuramente instabile; con una pendenza di –40dB/dec il sistema potrebbe essere instabile e si deve ricorrere al criterio generale di stabilità

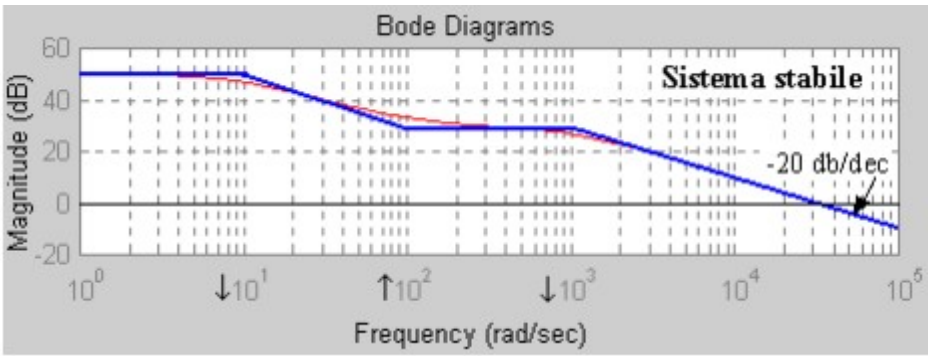
Esercizio 1 Verificare la stabilità del sistema ad anello chiuso

$$G(s)\cdot H(s)=\frac{316\cdot(1+0.01s)}{(1+0,1s)\cdot(1+0,001s)}$$

Zeri: z1= -100 ⇒ ↑ωZ=100 rad/sec

Poli: p1= -10 ⇒ ↓ωP 1 =10 rad/s ; p2= -1000 ⇒ ↓ωP 2 =1000 rad/s

Costante: K = 316 ⇒ Kd B = 20log 316 = 50 dB



L'attraversamento dell'asse delle ascisse avviene con una pendenza di – 20dB/dec, pertanto il sistema è stabile

Esercizio 2 Verificare la stabilità del sistema ad anello chiuso

$$G(s)\cdot H(s)=\frac{3162}{(1+0{,}1s)\cdot(1+0{,}01s)\cdot(1+0{,}001s)}$$

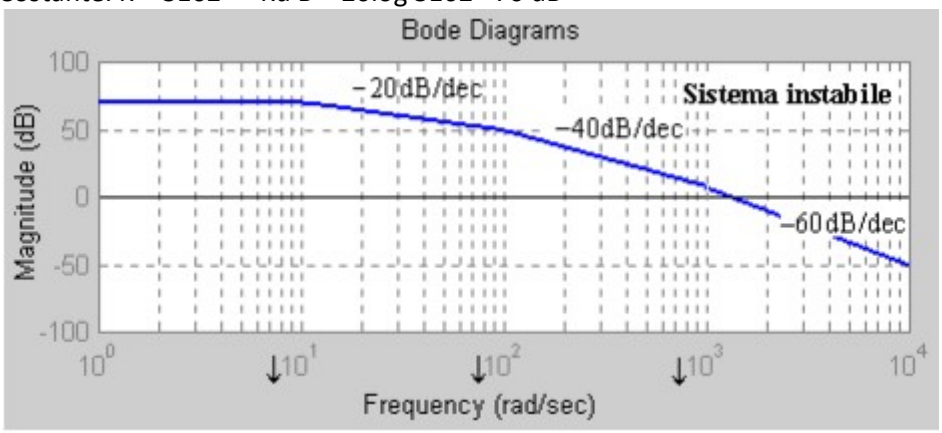
Poli:

p1= -10 ⇒ ↓ωP 1 =10 rad/s

p2= -100 ⇒ ↓ωP 2 =100 rad/s

p3= -1000 ⇒ ↓ωP 3 =1000 rad/s

Costante: K = 3162 ⇒ Kd B = 20log 3162= 70 dB



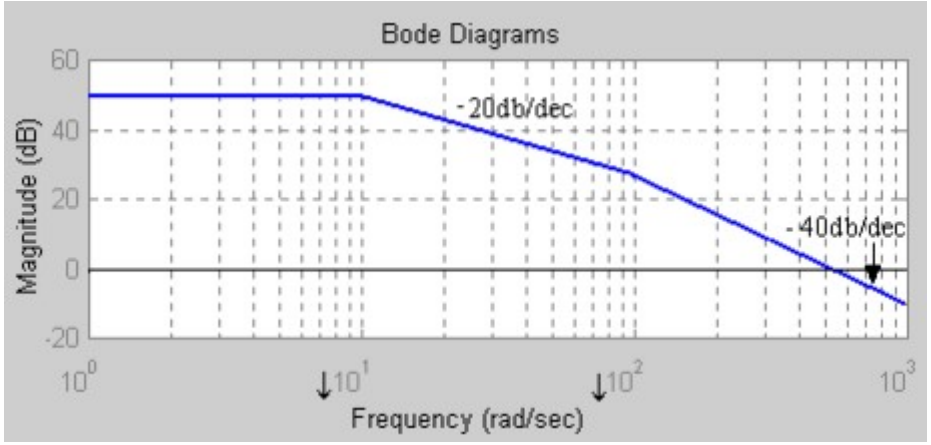
L'attraversamento dell'asse delle ascisse avviene con una pendenza di – 60dB/dec,pertanto il sistema è instabile

Esercizio 3

Verificare la stabilità del sistema ad anello chiuso

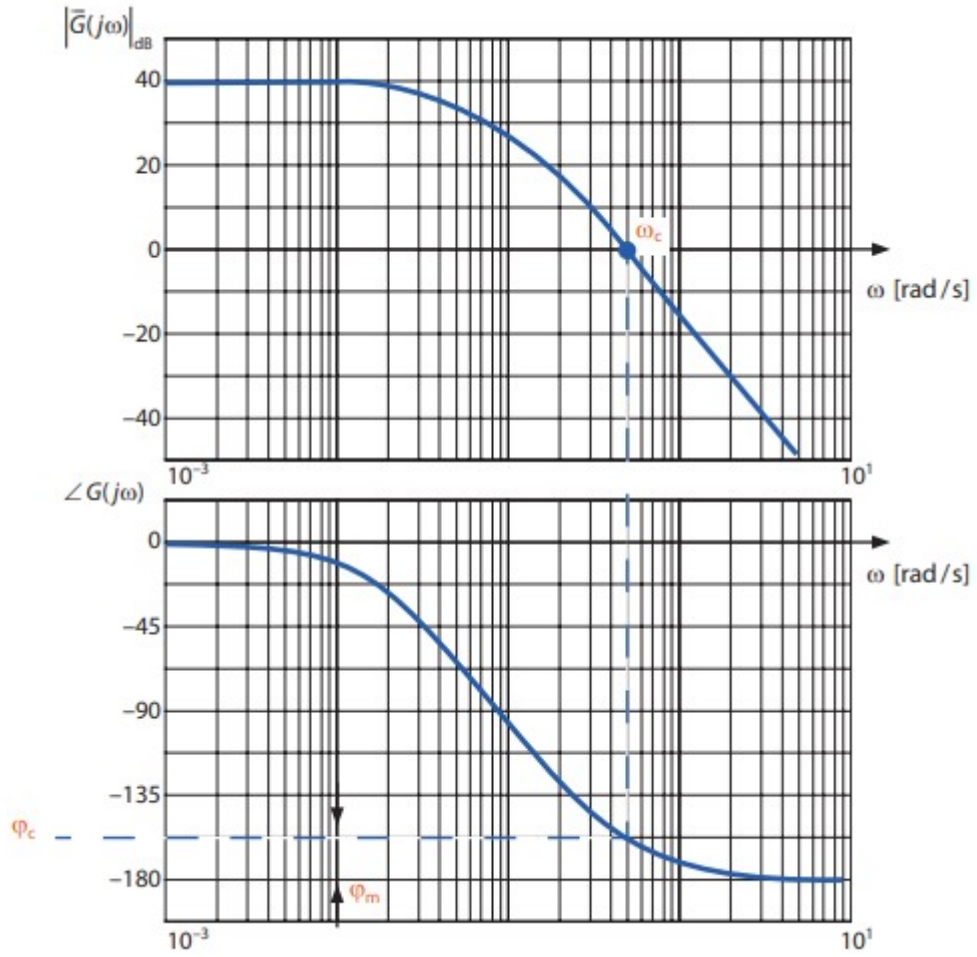
$$G(s)\cdot H(s)=\frac{316}{(1+0{,}1s)\cdot(1+0{,}01s)}$$

Poli: p1= -10 ⇒ ↓ωP 1 =10 rad/s ; p2= -100 ⇒ ↓ωP 2 =100 rad/s Costante: K = 316 ⇒ Kd B = 20log 316= 50 dB



L'attraversamento dell'asse delle ascisse avviene con una pendenza di – 40dB/dec, il criterio semplificato di Bode non ci permette di valutare la stabilità.

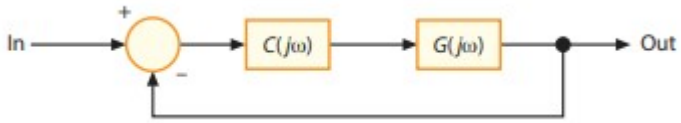
Per mezzo del diagramma di Bode è possibile:
• sapere se il sistema è stabile;
• indagare in termini quantitativi il grado di stabilità;
• valutare gli effetti sulla stabilità di modifiche apportate al sistema.
A tal fine si introducono dei parametri che misurano il grado di stabilità del sistema.
Per definirli consideriamo un tipico diagramma di Bode come quello della figura,
ed evidenziamo su esso i seguenti parametri.
• Pulsazione critica ωc
È il punto di incrocio tra il diagramma dei moduli e l'asse delle ω.
In questo punto il modulo vale 0 dB, ovvero 1.
Matematicamente: $|G(j\omega c)| = 1$.
• Sfasamento critico φ_c
È la fase del sistema in corrispondenza di ωc
Matematicamente: $\varphi_c = \angle G(j\omega c)$
• Margine di fase φ_m.
È l'intervallo di fase tra φ_c e 180°.
Matematicamente: $\varphi_m = 180 - |\varphi_c|$



Il criterio di Bode fornisce un metodo per quantificare la stabilità del sistema ad anello chiuso, in base ai valori dei parametri riscontrati sul diagramma di Bode del sistema ad anello aperto.
In particolare possiamo osservare che:
• se $\varphi_m = 0$ il sistema è al limite della stabilità.
Il sistema introduce infatti fase φ_c=180 e guadagno unitario per $\omega = \omega_c$. Di conseguenza si ingenerano autoscillazioni;
• se $\varphi_m < 0$ il sistema è instabile.
Il sistema introduce infatti fase φ_c> 180 e guadagno unitario per $\omega = \omega_c$. Percorrendo il diagramma verso le ω decrescenti si giunge a un punto in cui la fase vale 180° e il modulo è maggiore di 1. In queste condizioni le oscillazioni si esaltano progressivamente in ampiezza, determinando l'instabilità;

• se $\varphi_m > 0$ il sistema è stabile.
Il sistema introduce infatti fase φ_c <180 e guadagno unitario per ω = ωc. Percorrendo il diagramma verso le ω crescenti si giunge a un punto in cui la fase vale 180° e il modulo è minore di 1. In queste condizioni le oscillazioni si attenuano progressivamente in ampiezza fino a sfumare, pertanto il sistema è stabile.

Il margine di fase φm è, per quanto detto, il principale indicatore della stabilità.Il grado di stabilità è tanto più accentuato quanto maggiore è φ_m. Un buon margine di sicurezza è dato da valori di φ_m tra 45° e 60°.
Per modificare φm si deve manipolare la funzione di trasferimento del sistema,
mediante interposizione in cascata di opportune reti, dette reti correttrici o stabilizzatrici o anche compensatrici.
Le modifiche possono interessare il guadagno statico K della f.d.t. oppure le costanti di tempo al numeratore e denominatore, ovvero gli zeri e i poli.
La figura seguente descrive uno schema di principio nel quale la rete correttrice
C(jω) è inserita a monte della G(jω) del sistema da stabilizzare. Il tutto è calato
all'interno della schema in retroazione del controllo automatico.



Consideriamo nel seguito tre possibili tecniche di stabilizzazione:
• stabilizzazione mediante riduzione del guadagno di anello;
• stabilizzazione mediante spostamento a destra di un polo;
• stabilizzazione mediante spostamento a sinistra di un polo.
Dato che che a 0db corrisponde un'ampiezza unitaria il Margine di guadagno e uguale al numero di db in cui |GH(jw)| cioe' il guadagno ad anello aperto e' inferiore a 0db alla frequenza di attraversamento di fase w_π(arg $GH(Jw_\pi)$=180°). Il margine di fase e' uguale al numero di gradi in cui argGH(jw) e' superiore a -180° alla pulsazione critica w1(|GH(jw1)|=1).

Stability Criteria - (Gain Margin and Phase Margin)

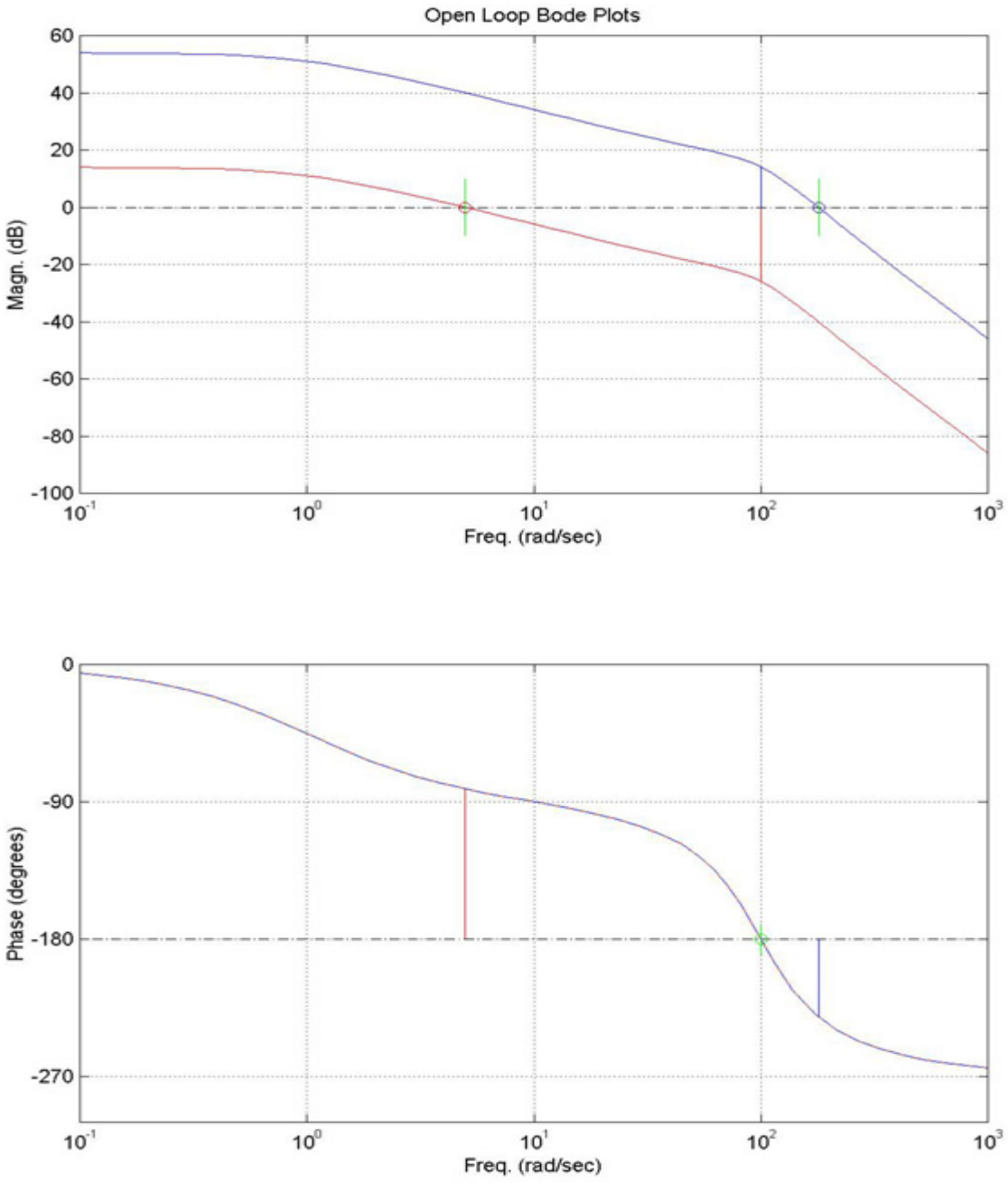
http://www.mit.edu/afs.new/athena/course/2/2.010/www_f00/psets/hw3_dir/tuto3_dir/tut3_g.html

Definizione pulsazione crtitica ω_c: pulsazione alla quale il guadagno in modulo e' uguale a 1 G(j ω_c)|=1
Pulsazione di attraversamento ω_t: pulsazione alla quale la fase e' uguale a -180°: arg[G(ω_t)]=-180°

Vengono definiti i margini di guadagno e di fase, in modo che intituivamente i margini positivi indichino che c'è ancora un margine di sicurezza (prima dell'instabilità). Al contrario, i margini negativi in un sistema ad anello aperto indicano problemi di instabilità se si tenta di chiudere l'anello.
Definiamo ciascuno, usando la figura a destra come ausilio:
MARGINE DI GUADAGNO
Si cerca la frequenza di attraversamento cioe' la freq. in cui la FASE diventa -180 gradi.
--- Nella nostra immagine, questo è a 100 (rad/sec) (contrassegnato da una "o" verde nella parte inferiore della trama).
- Troviamo il Guadagno, G (in dB), a questa STESSA FREQUENZA (dal grafico in alto).
- Quindi, definiamo il MARGINE DI GUADAGNO come:
Margine di Guadagno = 0 - G dB
Se non voglio calcolare in DB si puo' scrivere:
Margine di Guadagno = 0 - G dB=20Log(1)-20Log(M)=20Log(1/M)=>Margine di guadagno=1/M

MARGINE DI FASE
- Trovare la frequenza in cui il GAIN è 0 dB. (Ciò significa che le ampiezze di uscita e di ingresso sono identiche a questa particolare frequenza; sul grafico di Bode, è dove la funzione di trasferimento incrocia 0 dB sul grafico] superiore. Cioe' la frequenza o pulsazione critica.
--- Per il grafico di Bode rosso, questo avviene a circa 5 (rad/sec) [contrassegnato da una 'o' rossa nel grafico superiore].

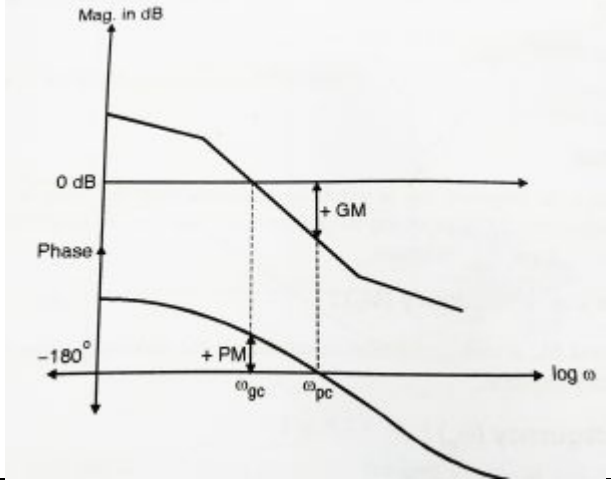
--- Per il grafico di Bode blu, il crossover 0 dB si verifica a una frequenza di circa 181 (rad/sec) ed è mostrato con una "o" blu.
- Trovare la FASE, P (in gradi), a questa STESSA FREQUENZA (vedi il grafico inferiore).
(Questa particolare fase è contrassegnata nel grafico in basso a destra per entrambe le funzioni di trasferimento rossa e blu con linee di colore corrispondente...)
- Definiamo quindi il MARGINE DI FASE come:
Margine di fase = +P + 180°

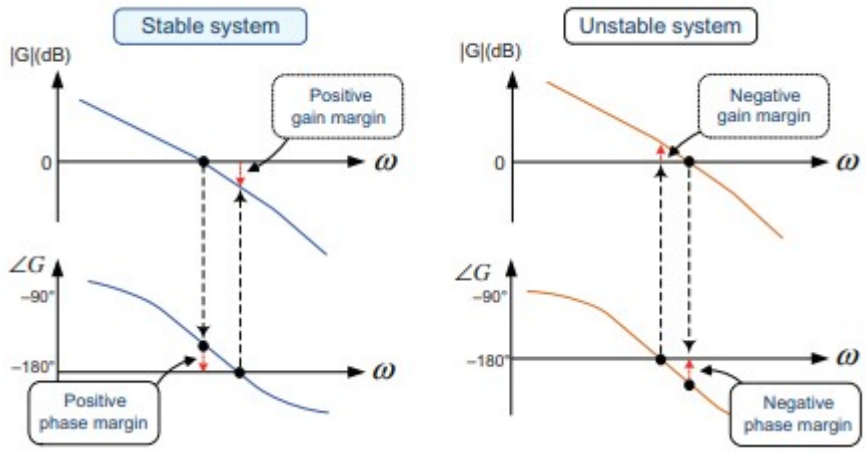


	points picked from Bode plots.	Phase Margin=+P + 180	Gain Margin=0 - G
red tf	P = -81.3 degrees (at crossover freq. of 5 (rad/sec)). G = -25.9 dB (at freq of 100 (rad/sec)).	+98.7 degrees	+25.9 dB
blue tf	P = -231.0 degrees (at crossover freq. of 181 (rad/sec)). G = +14.1 dB (at freq of 100 (rad/sec)).	-50.0 degrees	-14.1 dB

Ricapitolando:

GM e PM sono positivi =>stabile
GM e PM negativi =>instabile
GM e PM zero => limite della stabilita'





++++++++++++++++++++++++

Esempio calcolare il Margine di Guadagno del sistema:

$$G(s) = \frac{5}{s \cdot (s+5) \cdot (s+15)}$$

con feedback unitario (cioe' H(s)=1).

Determiniamo il guadagno alla crossover frequency:

$$GM = \frac{1}{|G(j\omega_t) \cdot H(j\omega_t)|}$$

Dove ω_t e' la frequenza di attraversamento della fase cioe' $\angle|G(j\omega_t) \cdot H(j\omega_t)| = -180°$

La ω_c e' la frequenza di attraversamento del guadagno o pulsazione critica:

$$|G(j\omega_c) \cdot H(j\omega_c)| = 1$$

Calcoliamo la ω_t cioe' la frequenza di attraversamento della fase:

$$\angle|G(j\omega_t) \cdot H(j\omega_t)| = -180° \Rightarrow -\frac{\pi}{2} - tan^{-1}\left(\frac{\omega_t}{5}\right) - tan^{-1}\left(\frac{\omega_t}{15}\right) = -\pi$$

$$tan^{-1}\left(\frac{\frac{\omega_t}{5}+\frac{\omega_t}{15}}{1-\frac{\omega_t}{5}\cdot\frac{\omega_t}{15}}\right) = \frac{\pi}{2} \Rightarrow den = 0 \Rightarrow 1 - \frac{\omega_t}{5}\cdot\frac{\omega_t}{15} \Rightarrow \omega_t = \sqrt{75} rad/sec \cong 8.7 rad/sec \Rightarrow f \cong 1.38 Hz$$

$$|G(j\omega_t) \cdot H(j\omega_t)| = \frac{5}{\omega_t \cdot \sqrt{5^2+{\omega_t}^2} \cdot \sqrt{15^2+{\omega_t}^2}} = \frac{5}{75 \cdot \sqrt{100} \cdot \sqrt{300}} = \frac{1}{300} \Rightarrow$$

Margine di Guadagno, $GM = \frac{1}{|G(j\omega_t)\cdot H(j\omega_t)|} = 20 \cdot Log_{10} 300 dB \cong 49 dB$

Esempio cslcolo Margine di fase (|G(jwc)|=1) del seguente sistema:

$$G(s) = \frac{10}{s \cdot (s+10)} \Rightarrow G(j\omega) = \frac{10}{j\omega \cdot (j\omega+10)} \Rightarrow |G(j\omega)| = \frac{10}{\omega \cdot \sqrt{\omega^2+100}} = 1 \Rightarrow \omega^4 + 100 \cdot \omega^2 - 100 = 0$$

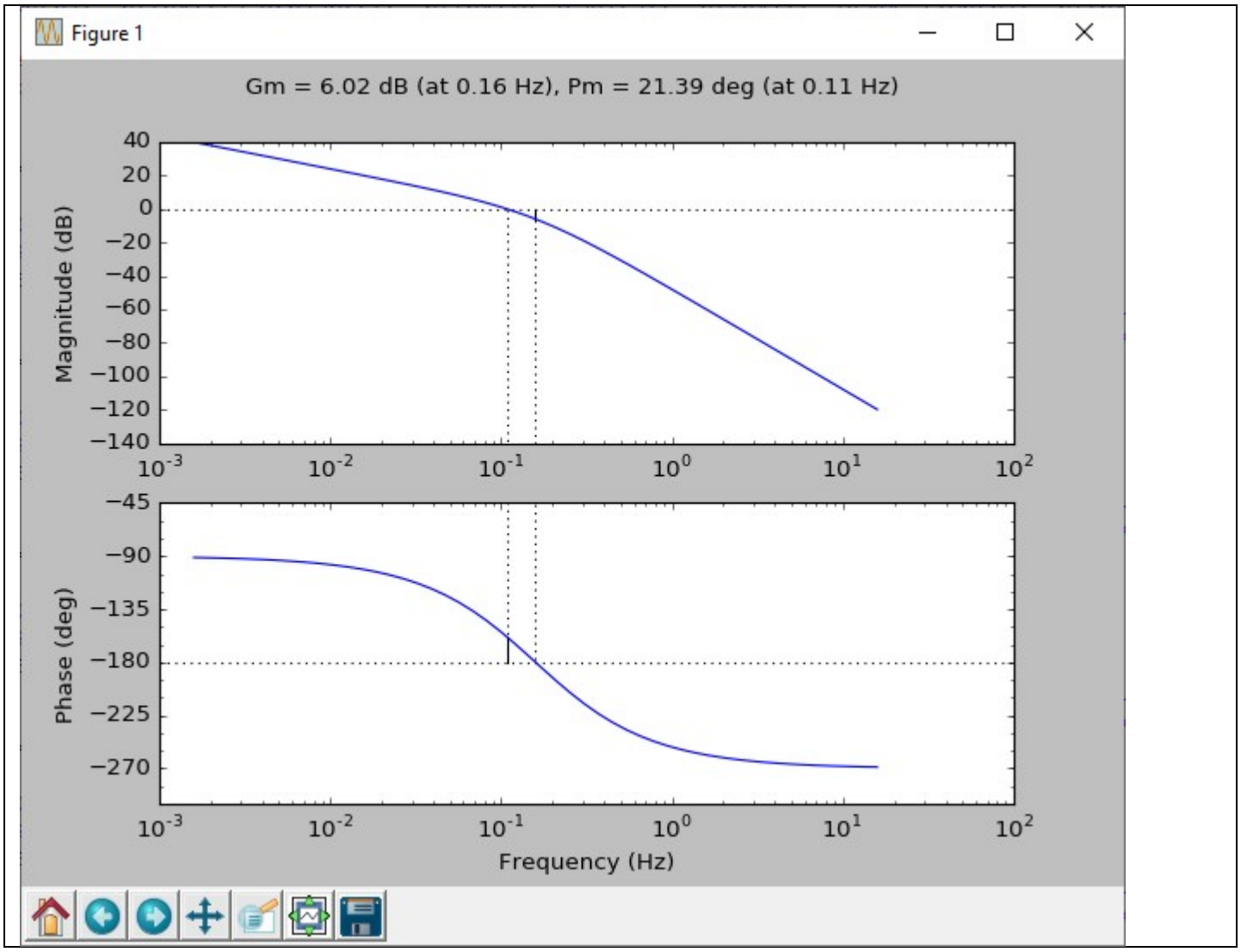
Ponendo: $x = \omega^2 \Rightarrow x^2 + 100x - 100 = 0 \Rightarrow x_{1,2} = \frac{-100 \pm 101.98}{2} \Rightarrow \omega = 1, -1 \Rightarrow \omega = 1 rad/sec$

$$\phi = \angle G(j\omega)|_{\omega=1} = -\operatorname{atan}\left(\frac{\omega}{0}\right) - \operatorname{atan}\left(\frac{\omega}{10}\right)$$
$$= -90° - \operatorname{atan}\left(\frac{1}{10}\right) = -95.71° \Rightarrow P.M. = 180° + \phi = 180 - 95.71 = 84.29°$$

Esempio - Gain margin, phase margin, and crossover frequencies

```
#N.B.!!!:nell'attuale libreria control (0.8.3) e' presente un bug talvolta nel calcolo dei margini, ad es.:
s = control.tf('s')
G=100./((1+s*0.1)*(1+0.001*s))
print control.margin(G)
num=[100.]
den=np.polymul( np.array([0.1,1.]),np.array([0.01,1.]))
G = control.tf(num,den)
print control.margin(G)
>>>
(inf, 52.55773904488177, nan, 786.1053548759002)
(inf, 19.826436218434765, nan, 308.33818697209335)

Soluzione: E' Possibile usare la seguente funzione per disegnare i margini nei diagrammi di Bode:

def isNumDef(n):
   return not n is None and (not (math.isinf(n) or math.isnan(n)))
def myBode_plot(G,pz=False,margins=False,omega=None):
   try:
      if pz==True:
         zeros=[abs(x) for x in control.zero(G)]
         poles=[abs(x) for x in control.pole(G)]
         print "zeros:",zeros,"poles:",poles
      mag, phase, omega = control.bode(G,dB=True,Plot=False,omega=omega)
      magdB = 20*np.log10(mag)
      phase_deg = phase*180.0/np.pi
      Gm=None;Pm=None;Wcg=None;Wcp=None;GmdB=None
      if margins:
         Gm,Pm,Wcg,Wcp = control.margin(G)
         GmdB = 20*np.log10(Gm)
      ##Plot Gain and Phase
      f,(ax1,ax2) = plt.subplots(2,1)
      ax1.semilogx(omega,magdB)
      ax1.grid(which="both")
      ax1.set_xlabel('Frequency (rad/s)')
      ax1.set_ylabel('Magnitude (dB)')
      ax2.semilogx(omega,phase_deg)
      ax2.grid(which="both")
      ax2.set_xlabel('Frequency (rad/s)')
      ax2.set_ylabel('Phase (deg)')
      sTitle=""
      if isNumDef(GmdB):
         sTitle += 'Gm = '+str(np.round(GmdB,2))
      if isNumDef(Wcg):
         sTitle += ' dB (at '+str(np.round(Wcg,2))+' rad/s)'
         ##Plot the vertical line from -180 to 0 at Wcg
         ax2.plot([Wcg,Wcg],[-180,0],'r--',linewidth=2)
         ##Plot the vertical line from min(magdB) to 0-GmdB at Wcg
         ax1.plot([Wcg,Wcg],[np.min(magdB),0-GmdB],'r--',linewidth=2)
      if isNumDef(Pm):
         sTitle += ' , Pm = '+str(np.round(Pm,2))
      if isNumDef(Wcp):
```

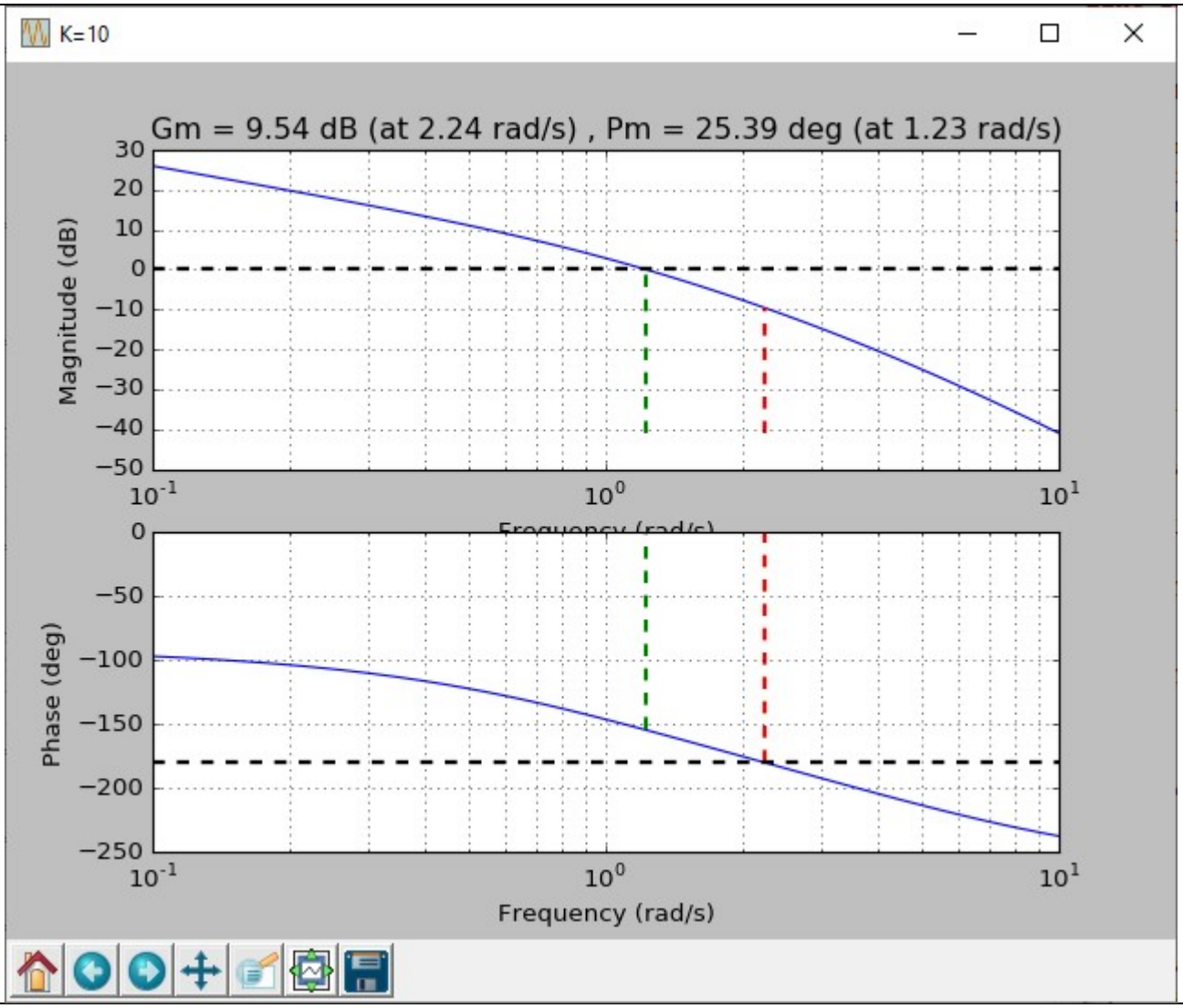
```
        sTitle += ' deg (at '+str(np.round(Wcp,2))+' rad/s)'
        ##Plot the vertical line from -180+Pm to 0 at Wcp
        ax2.plot([Wcp,Wcp],[-180+Pm,0],'g--',linewidth=2)
        ##Plot the vertical line from min(magdB) to 0db at Wcp
        ax1.plot([Wcp,Wcp],[np.min(magdB),0],'g--',linewidth=2)
      ax1.set_title(sTitle)

      if margins:
        ###Plot the zero dB line
        ax1.plot(omega,0*omega,'k--',linewidth=2)
        ###Plot the -180 deg lin
        ax2.plot(omega,-180+0*omega,'k--',linewidth=2)
      if pz==True:
        xmin, xmax, ymin, ymax = plt.axis()
        for z in zeros:
          ax1.plot([z,z], [ymin,ymin], marker="o", markersize=10, markeredgecolor="red", markerfacecolor="green")

        for p in poles:
          ax1.plot([p,p], [ymin,ymin],color='brown', linewidth=4,marker='x', markerfacecolor='lightgreen',
markeredgewidth=2,markersize=10, markevery=3)
      return Gm,Pm,Wcg,Wcp

    except Exception as e:
      print ("EXCEPTION: " + e.message)

G = control.tf(1,[1, 2, 1, 0])
Gm,Pm,Wcg,Wcp=control.margin(G)
print Gm,Pm,Wcg,Wcp
print "Gain margin:",20*np.log10(Gm),"dB"
print "Phase margin:",Pm
print "Crossover frequency associated with gain margin (phase crossover frequency), where phase crosses below -
180 degrees:",Wcg
print "Crossover frequency associated with phase margin (gain crossover frequency), where gain crosses below 1:",
Wcp
control.bode(G,margins=True,omega=np.logspace(-2, 2))
plt.show()
>>>
2.0 21.386389751875043 1.0 0.6823278038280193
Gain margin: 6.020599913279624 dB
Phase margin: 21.386389751875043
Crossover frequency associated with gain margin (phase crossover frequency), where phase crosses below -180
degrees: 1.0
Crossover frequency associated with phase margin (gain crossover frequency), where gain crosses below 1:
0.6823278038280193
```



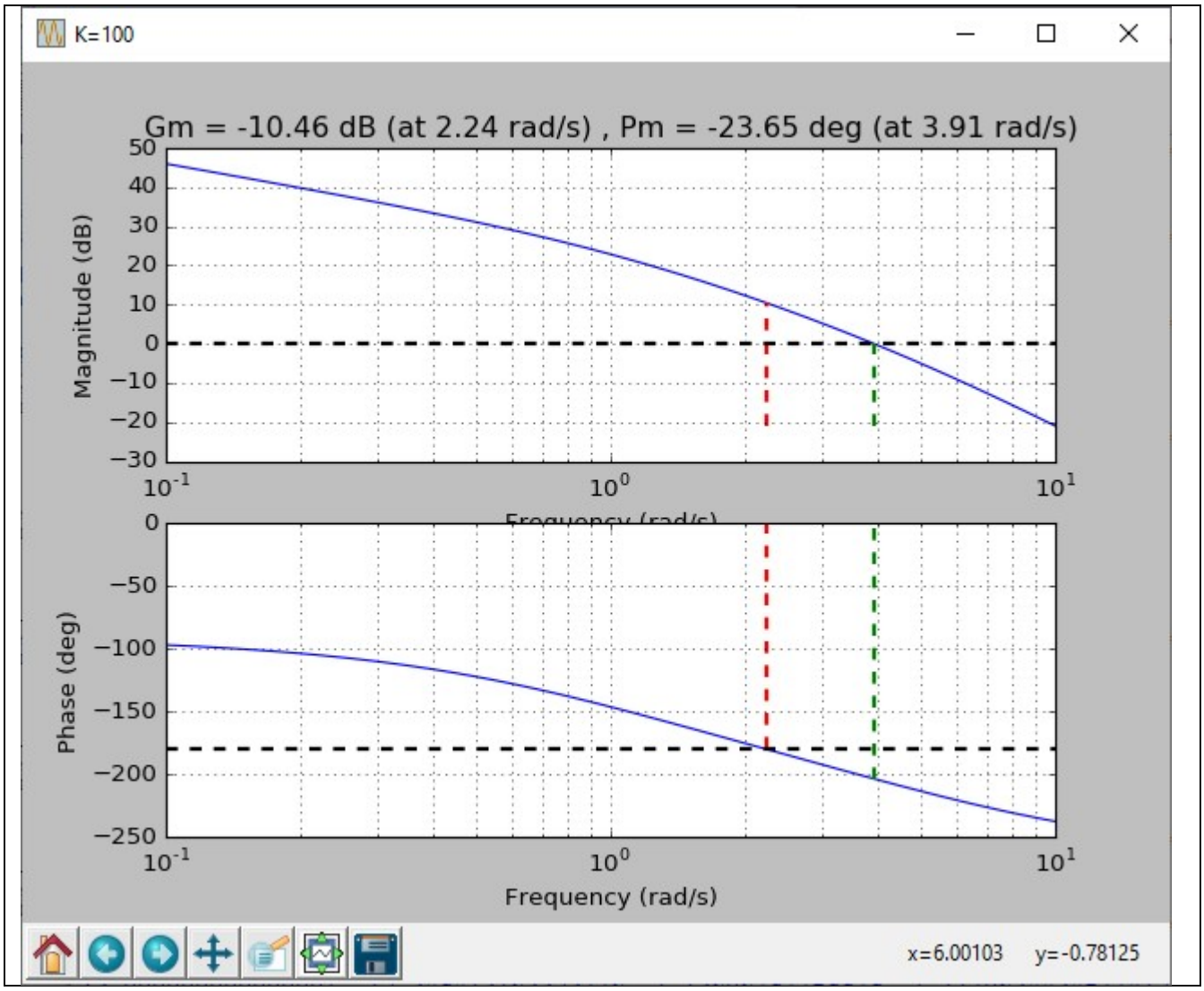
Trovare i margini di guadagno e fase nel caso K=10 e K=100 della f.d.t.:

$$G(s) = \frac{K}{s \cdot (s+1) \cdot (s+5)}$$

```
s = control.tf('s')
K=10
G=K/(s*(s+1)*(s+5))
print myBode_plot(G, margins=True,omega=np.logspace(-1, 1, 100))
plt.gcf().canvas.set_window_title('K=%s' % K)
plt.show()
```




```
s = control.tf('s')
K=100
G=K/(s*(s+1)*(s+5))
print myBode_plot(G, margins=True,omega=np.logspace(-1, 1, 100))
plt.gcf().canvas.set_window_title('K=%s' % K)
plt.show()
```



STABILIZZAZIONE – CRITERIO DI BODE

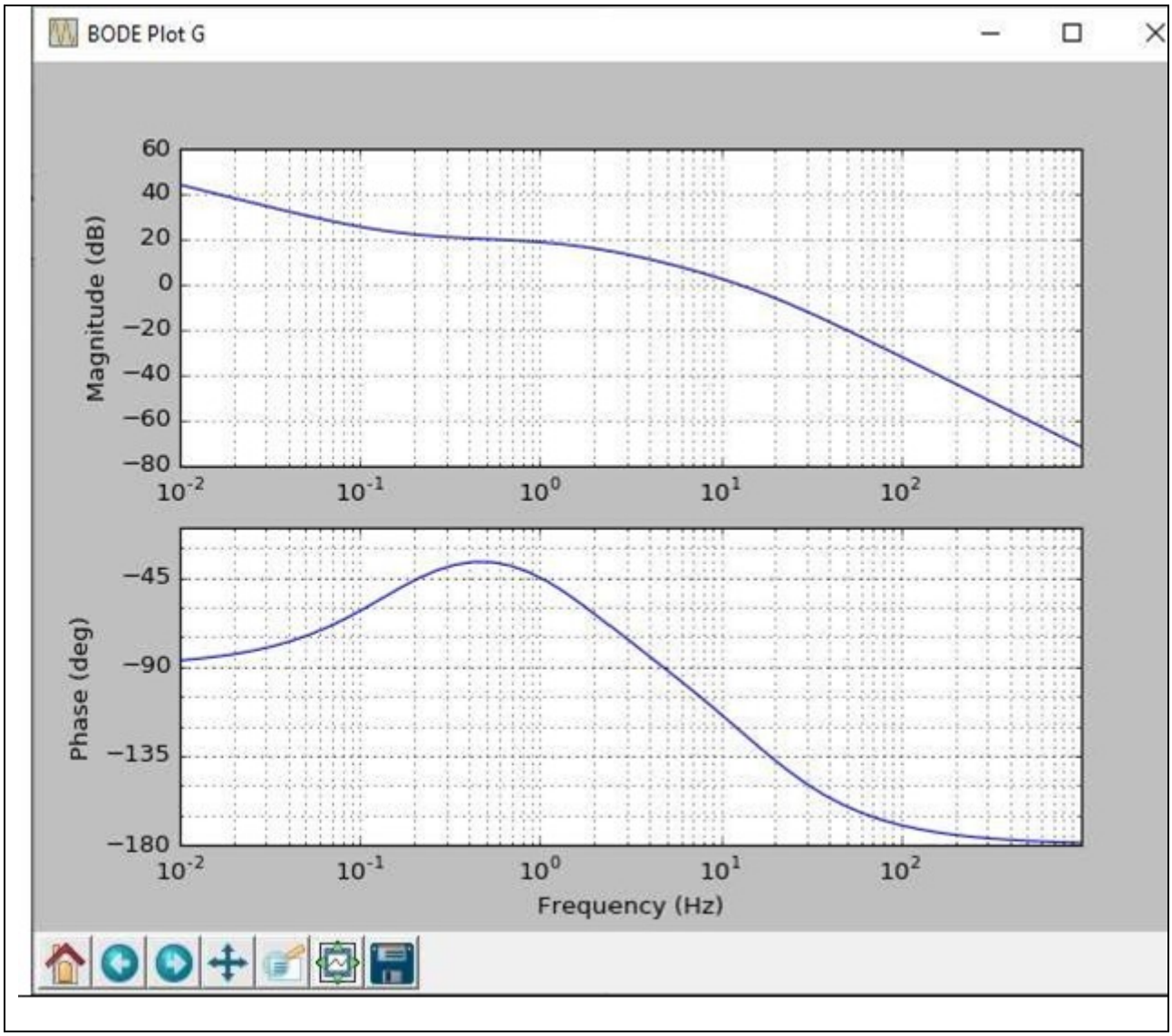
Simulazione del sistema:

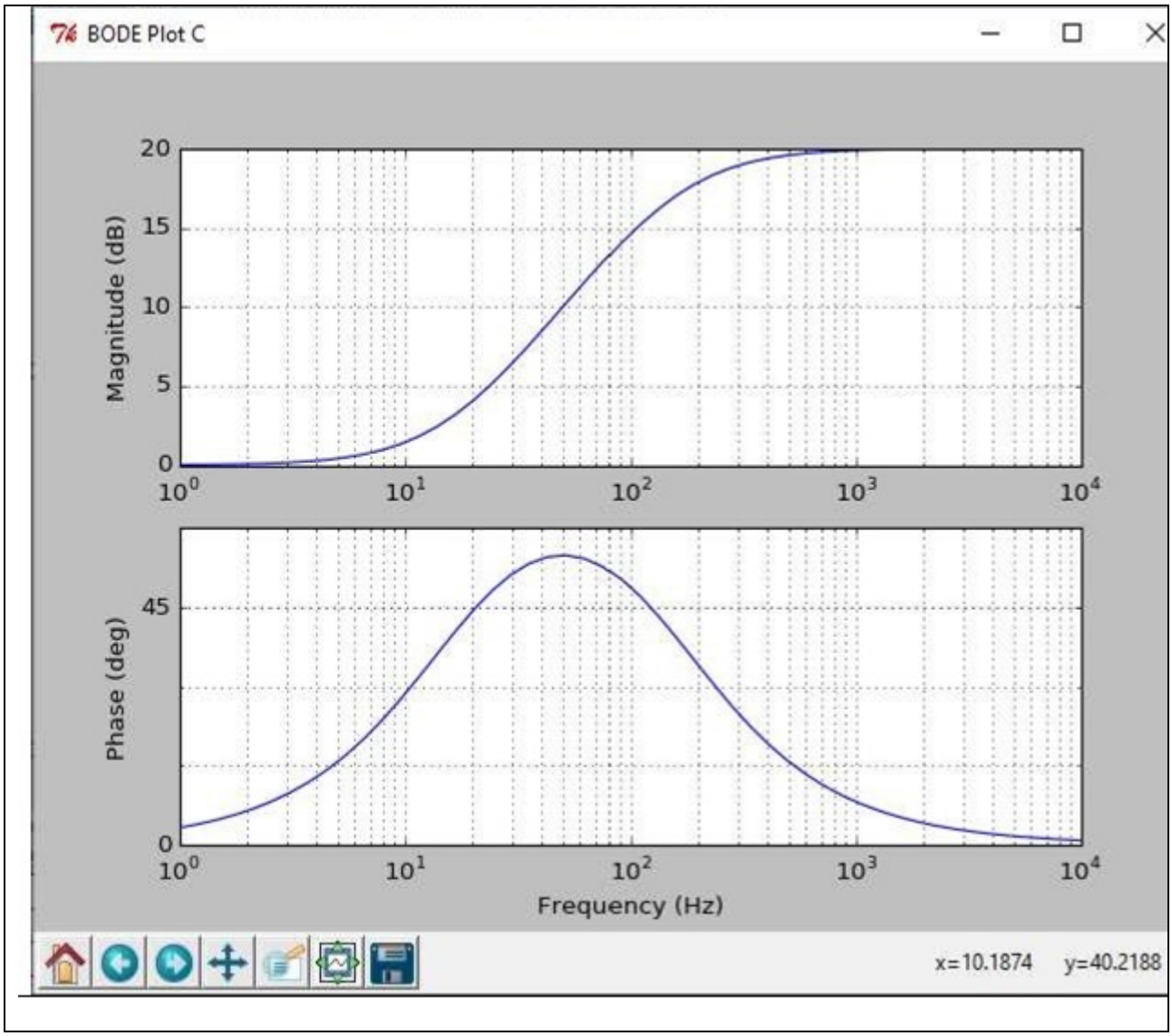
$$G(s) = \frac{10 \cdot (1+s)}{s \cdot (1+0.1 \cdot s) \cdot (1+0.01 \cdot s)}$$

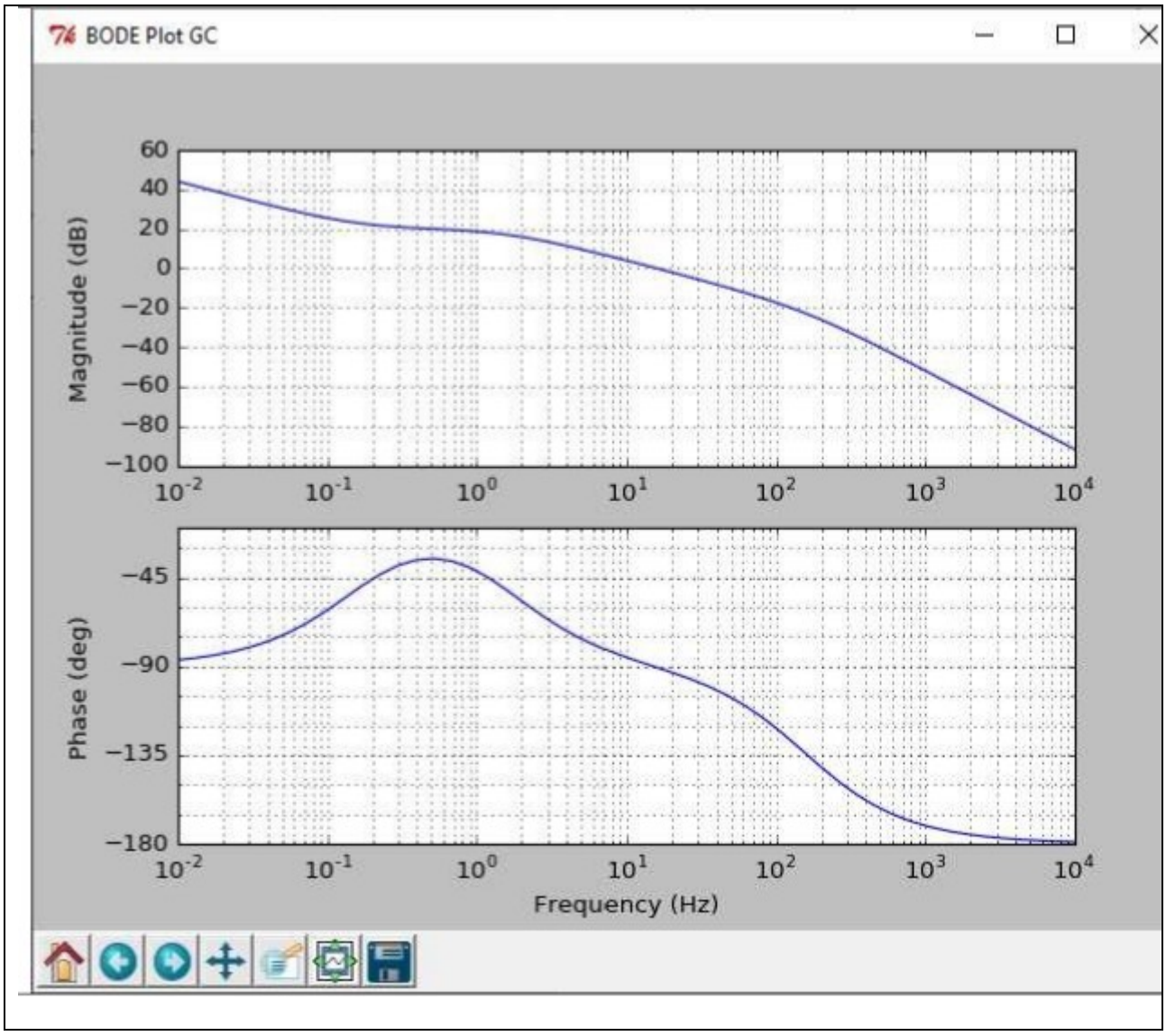
Con rete correttrice:

$$C(s) = \frac{(1+0.01 \cdot s)}{(1+0.001 \cdot s)}$$

```
#pag 182(v3)
s = control.tf('s')
G=(10*(1+s))/(s*(1+0.1*s)*(1+0.01*s))
fig, axs = plt.subplots()
fig.canvas.set_window_title('BODE Plot G')
control.bode_plot(G)
C=(1+0.01*s)/(1+0.001*s)
fig, axs = plt.subplots()
fig.canvas.set_window_title('BODE Plot C')
control.bode_plot(C)
GC=G*C
fig, axs = plt.subplots()
fig.canvas.set_window_title('BODE Plot GC')
control.bode_plot(GC)
plt.show()
```

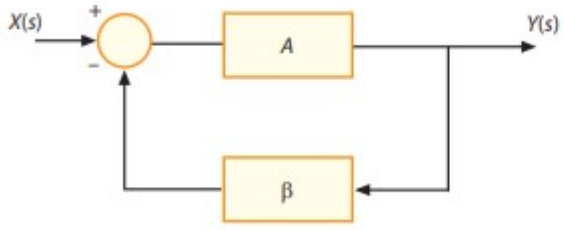

BODE Plot G
Magnitude (dB)
60
40
20
0
−20
−40
−60
−80
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Phase (deg)
−45
−90
−135
−180
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (Hz)


BODE Plot C
Magnitude (dB)
20
15
10
5
0
10^0
10^1
10^2
10^3
10^4
Phase (deg)
45
0
Frequency (Hz)
x=10.1874
y=40.2188



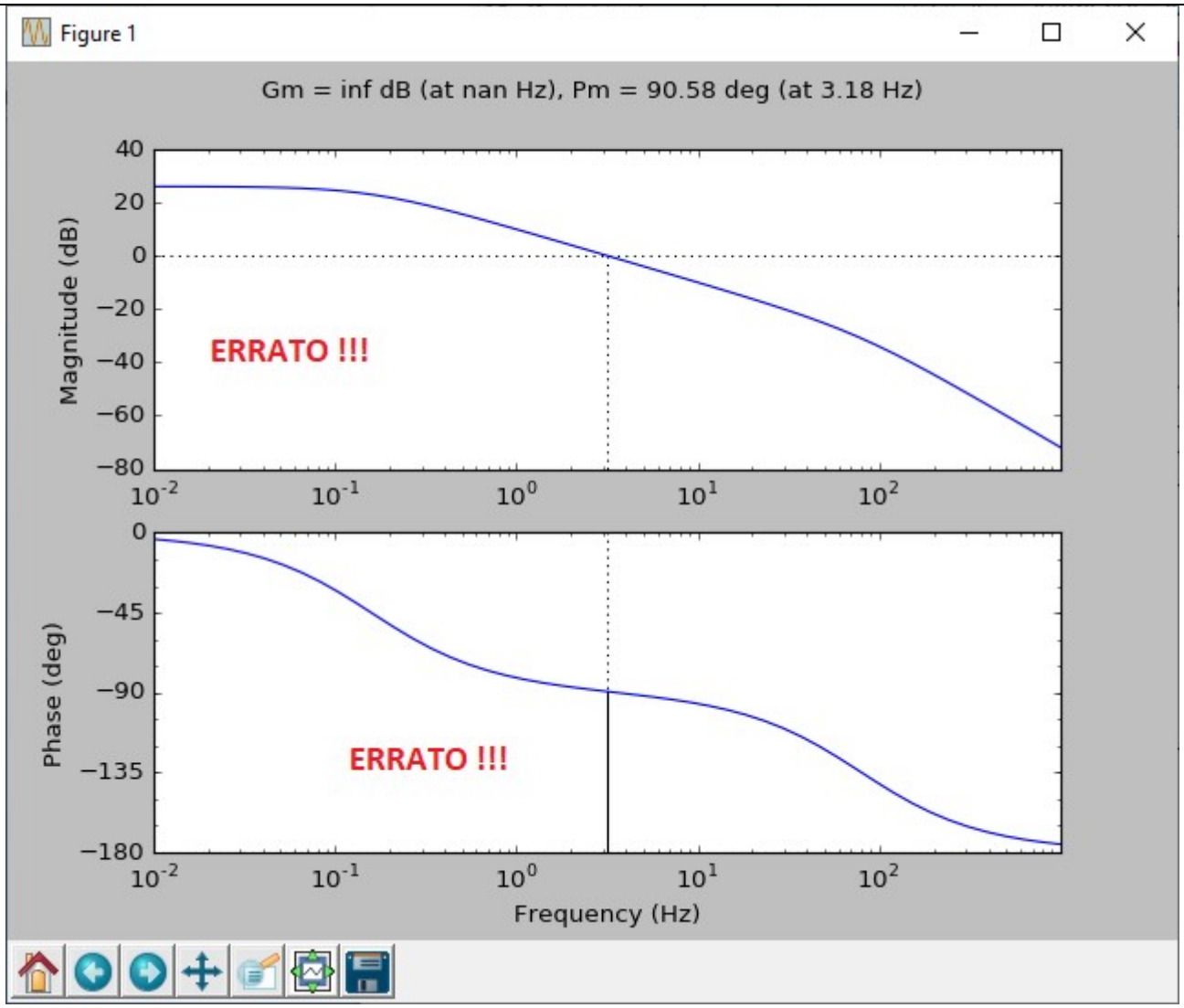
DIMENSIONAMENTO SISTEMA CON DATO MARGINE DI FASE

Disegnare il diagramma di Bode della funzione di trasferimento d'anello dello schema di Figura seguente, considerando il blocco β di tipo proporzionale puro e unitario.
Calcolare quindi il margine di fase φm ottenuto per un fattore di retroazione unitario e il valore da assegnare al blocco β per ottenere φm = 45°.

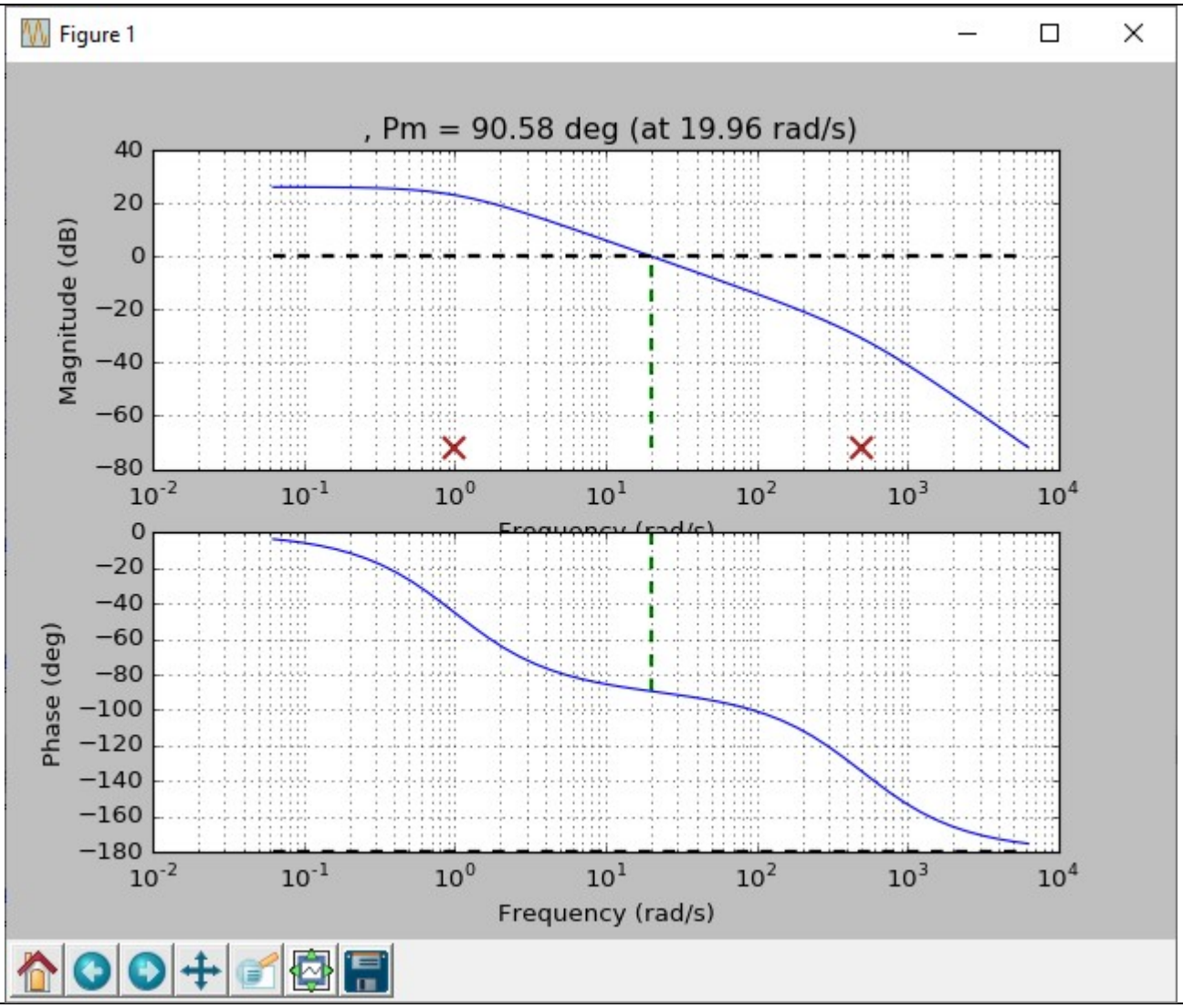


$$A(s)=\frac{10^4}{(s+1)(s+500)}$$

```
##ERRATO BUG Nel Calcolo Margini
s = control.tf('s')
G=20./((1+s)*(1.0+0.002*s))
print G
control.bode_plot(G)
gm, pm, wcg, wcp = control.margin(G)
print "Phase Margin","{:.2f}".format(pm)
```

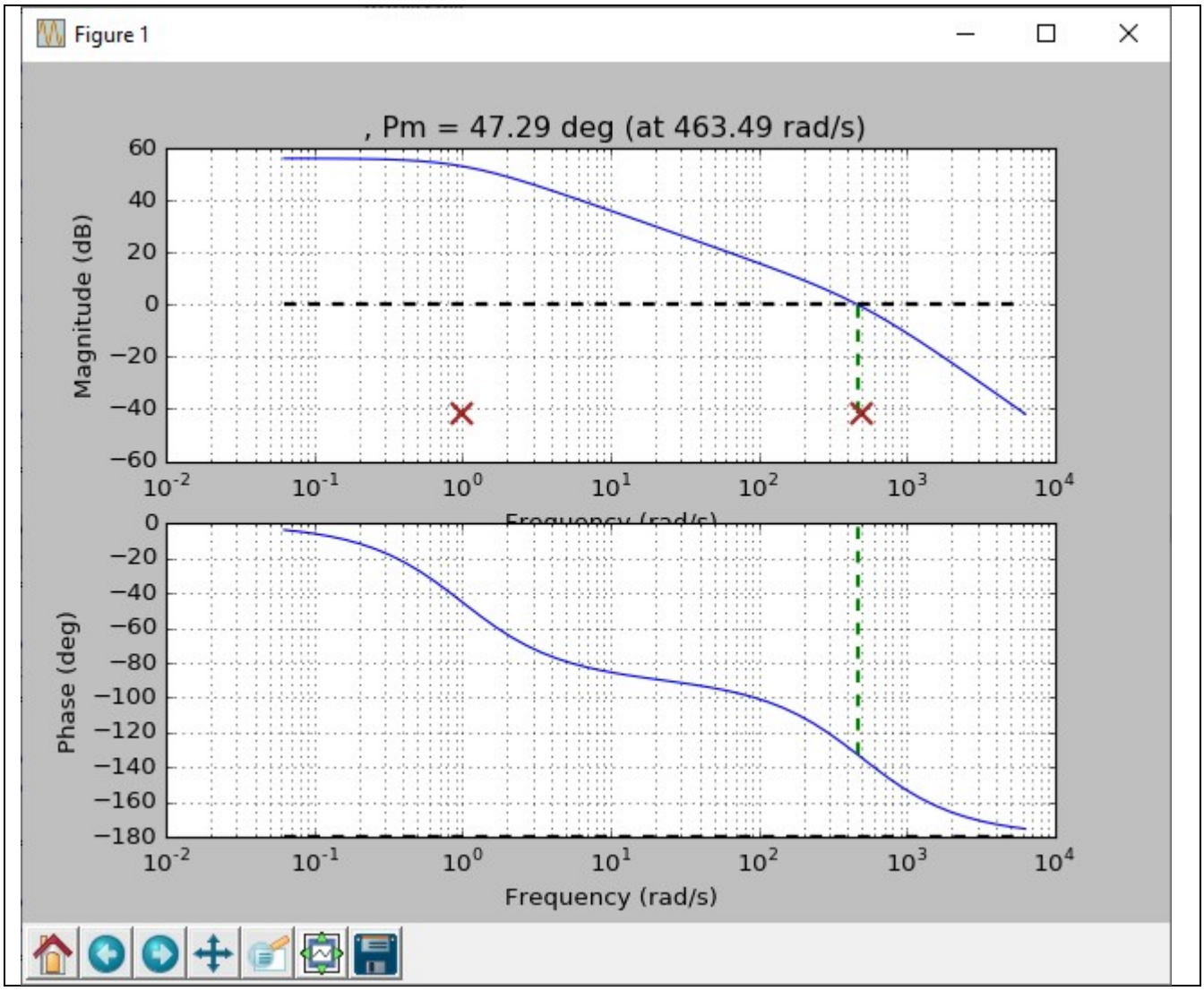



```
s = control.tf('s')
G=20./((1+s)*(1.0+0.002*s))
print G
print myBode_plot (G,pz=True,margins=True)
plt.show()
>>>
        20
----------------------
0.002 s^2 + 1.002 s + 1

zeros: [] poles: [500.0, 1.0]
(inf, 90.58234351819186, nan, 19.959048880435052)
```



MODIFICA m.f. CON β=31.6

```
s = control.tf('s')
G=(20*31.6)/((1+s)*(1.0+0.002*s))
print G
print myBode_plot (G,pz=True,margins=True)
plt.show()
>>>
        632
-----------------------
0.002 s^2 + 1.002 s + 1

zeros: [] poles: [500.0, 1.0]
(inf, 47.29363059494872, nan, 463.4913926084713)
```

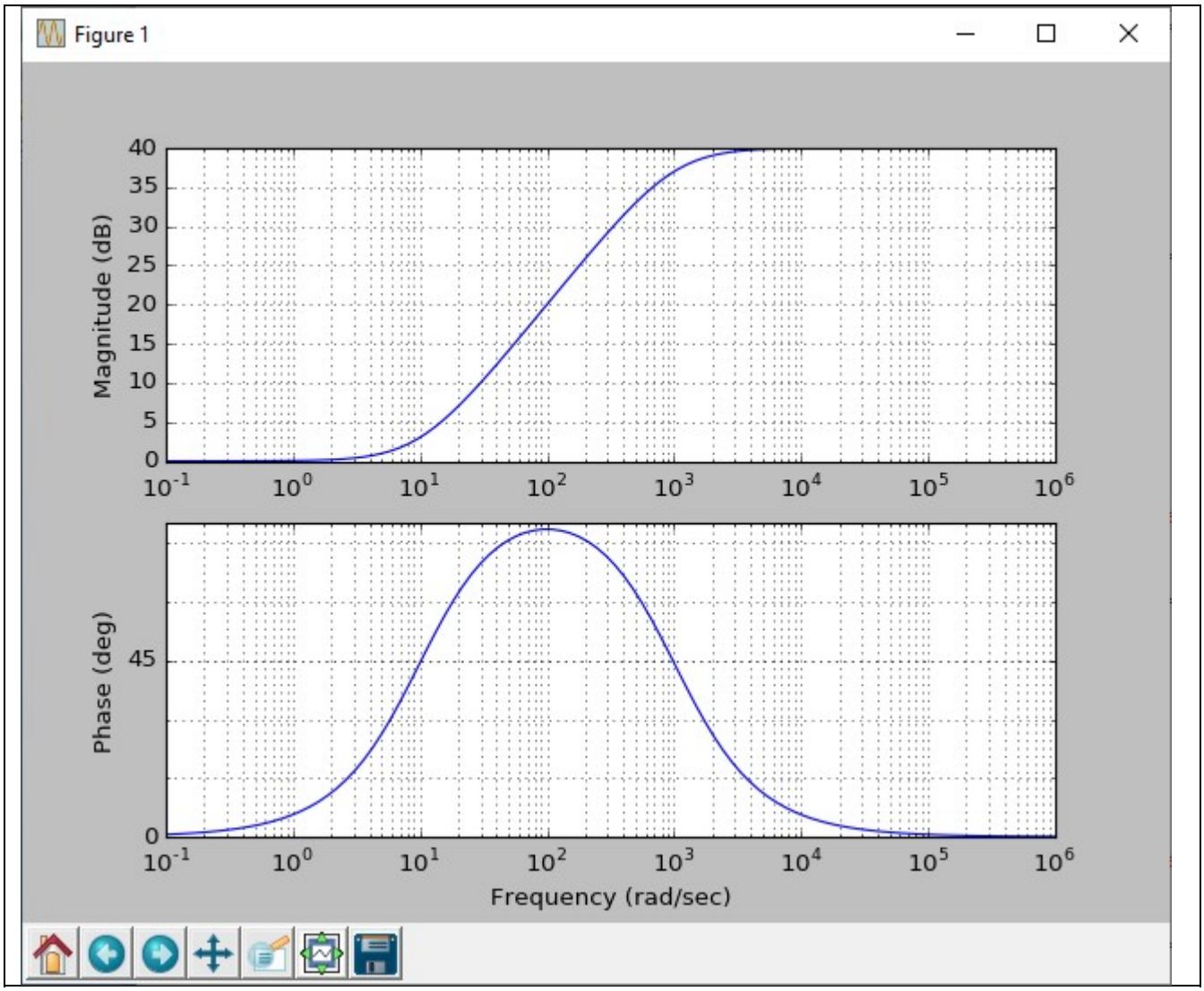


RETI CORRETTRICI

RETE ANTICIPATRICE

$$G(j\omega) = \frac{1 + j\omega \cdot 0.1}{1 + j\omega \cdot 0.001}$$

```
s = control.tf('s')
C=(1+0.1*s)/(1.0+0.001*s)
print C
gm, pm, wcg, wcp = control.margin(G)
print "Phase Margin","{:.2f}".format(pm)
control.bode_plot(G, omega=np.logspace(-1, 6, 100),dB=True)
plt.show()
```

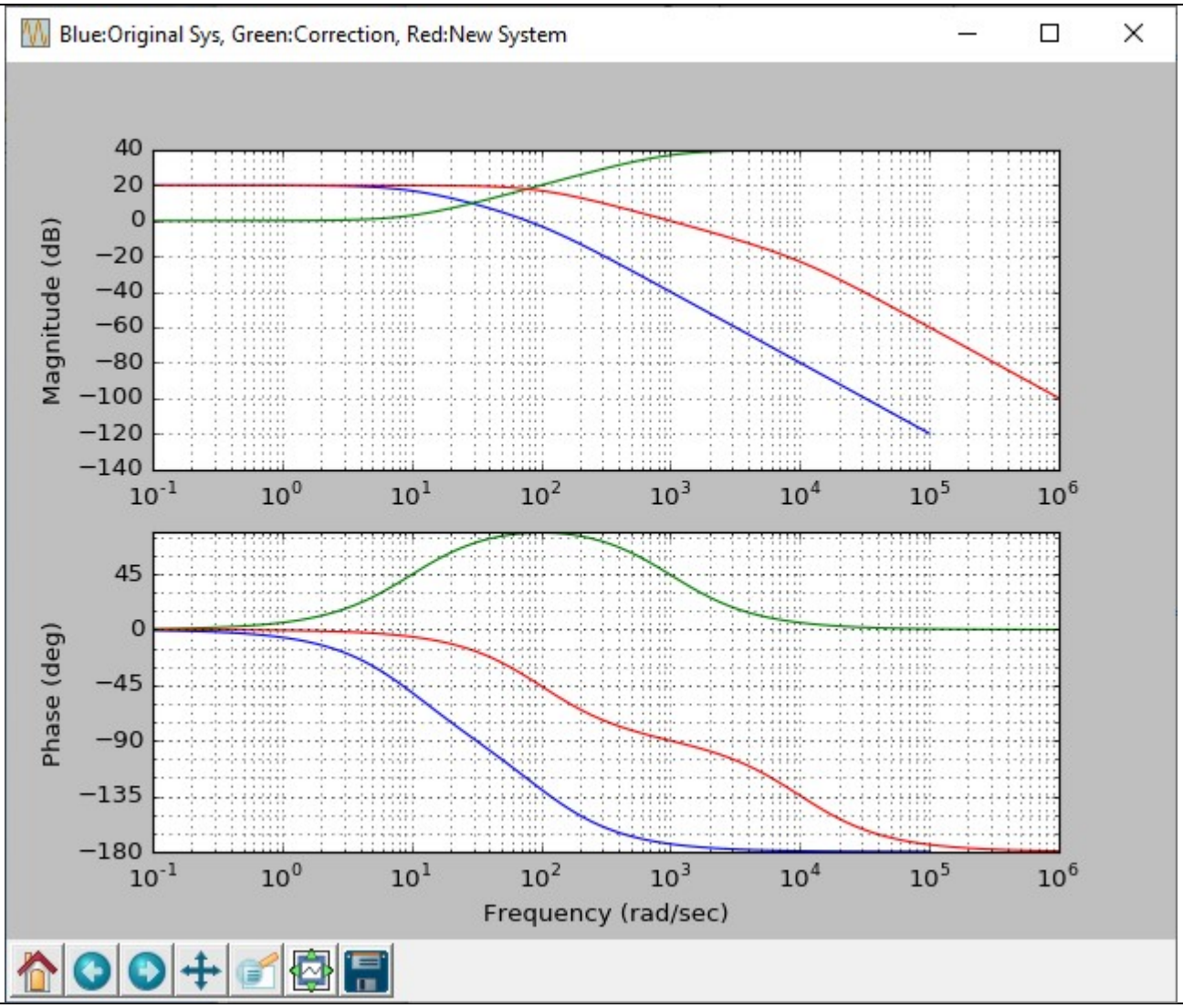


Sistema - Correzione con rete anticipatrice

$$G(j\omega) = \frac{10}{(1 + j\omega \cdot 0.1) \cdot (1 + j\omega \cdot 0.01)}$$

```
s = control.tf('s')
G=10./((1+0.1*s)*(1+0.01*s))
print G
gm, pm, wcg, wcp = control.margin(G)
print "Phase Margin","{:.2f}".format(pm)
control.bode_plot(G, omega=np.logspace(-1, 5, 100),dB=True)
C=(1+0.1*s)/(1.0+0.001*s)
print C
control.bode_plot(C, omega=np.logspace(-1, 6, 100),dB=True)
GC=10./((1.+0.01*s)*(1+0.0001*s))
print GC
gm, pm, wcg, wcp = control.margin(GC)
print "Phase Margin","{:.2f}".format(pm)
control.bode_plot(GC, omega=np.logspace(-1, 6, 100),dB=True)
fig=plt.gcf()
fig.canvas.set_window_title('Blue:Original Sys, Green:Correction, Red:New System ')
plt.show()

>>>

       10
----------------------
```

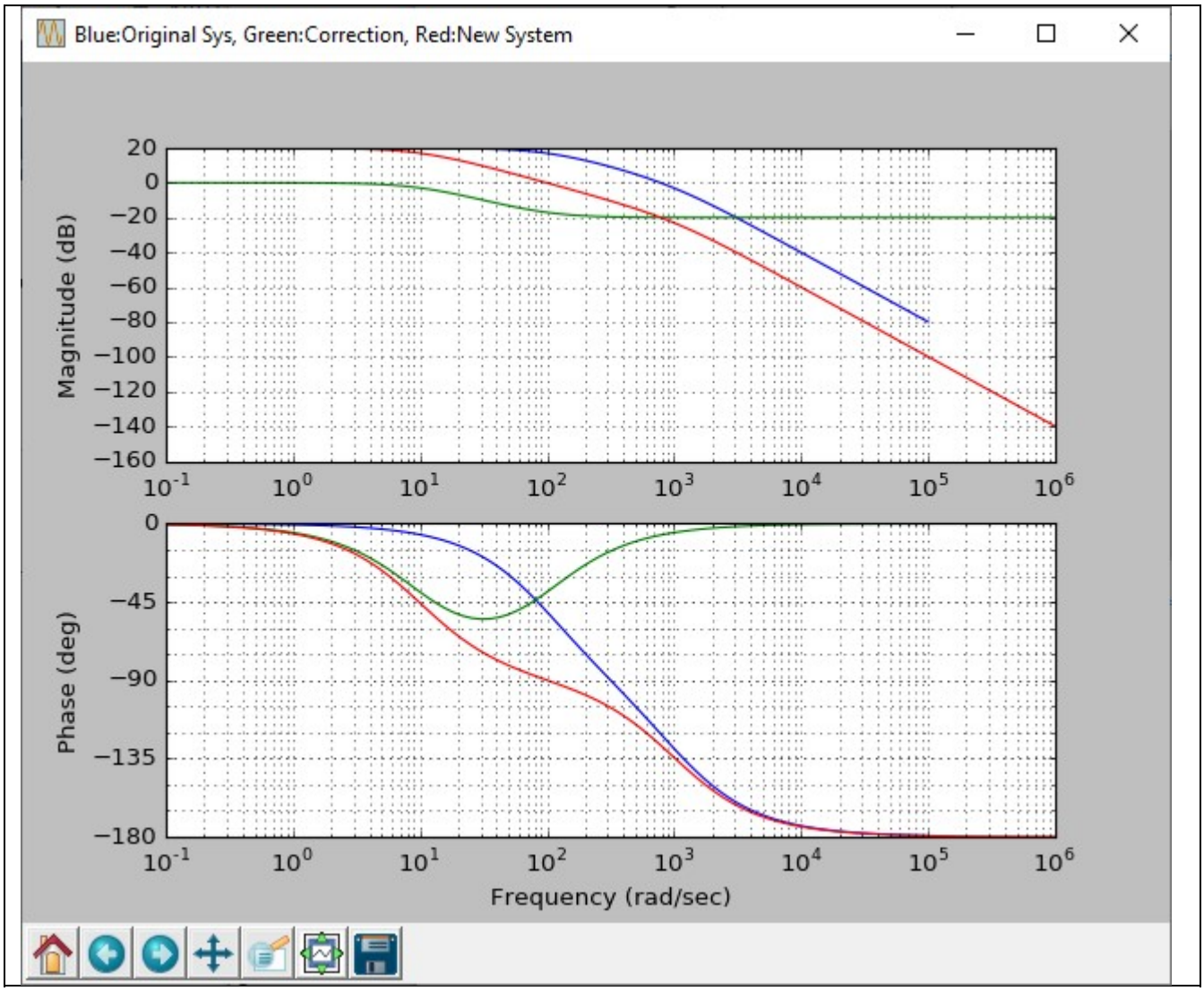
```
0.001 s^2 + 0.11 s + 1

Phase Margin 59.28

 0.1 s + 1
-----------
0.001 s + 1


         10
------------------------
1e-06 s^2 + 0.0101 s + 1

Phase Margin 90.11
>>>
```



Sistema - Correzione con rete ritardatrice

```
s = control.tf('s')
G=10./((1+0.01*s)*(1.0+0.001*s))
print G
gm, pm, wcg, wcp = control.margin(G)
print "Phase Margin","{:.2f}".format(pm)
control.bode_plot(G, omega=np.logspace(-1, 5, 100),dB=True)
C=(1+0.01*s)/(1.0+0.1*s)
print C
```

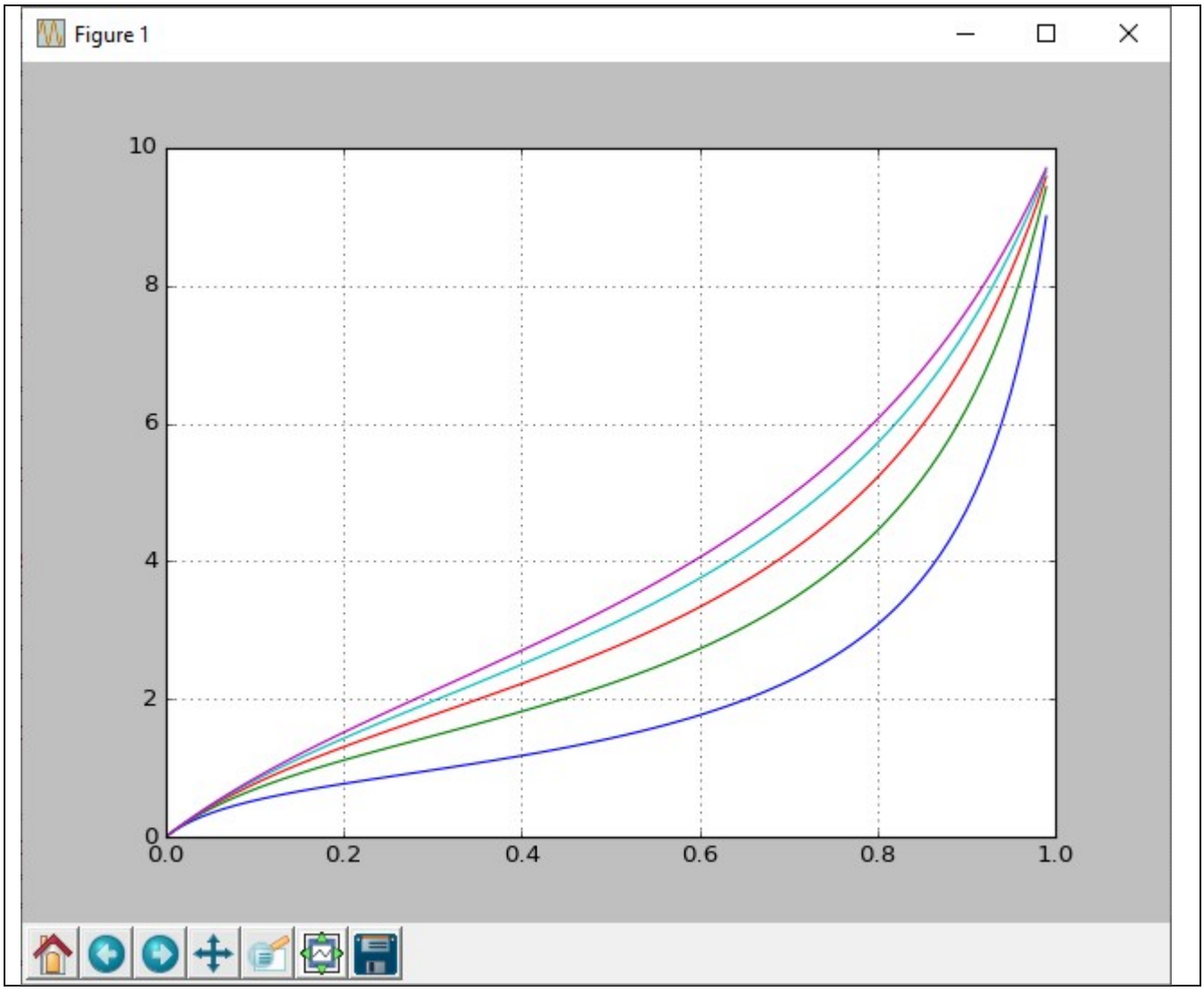
```
control.bode_plot(C, omega=np.logspace(-1, 6, 100),dB=True)
GC=10./((1.+0.001*s)*(1+0.1*s))
print GC
gm, pm, wcg, wcp = control.margin(GC)
print "Phase Margin","{:.2f}".format(pm)
control.bode_plot(GC, omega=np.logspace(-1, 6, 100),dB=True)
fig=plt.gcf()
fig.canvas.set_window_title('Blue:Original Sys, Green:Correction, Red:New System ')
plt.show()
>>>
        10
----------------------
1e-05 s^2 + 0.011 s + 1

Phase Margin 59.28

0.01 s + 1
----------
 0.1 s + 1


        10
------------------------
0.0001 s^2 + 0.101 s + 1

Phase Margin 90.11
```



TRASDUTTORI

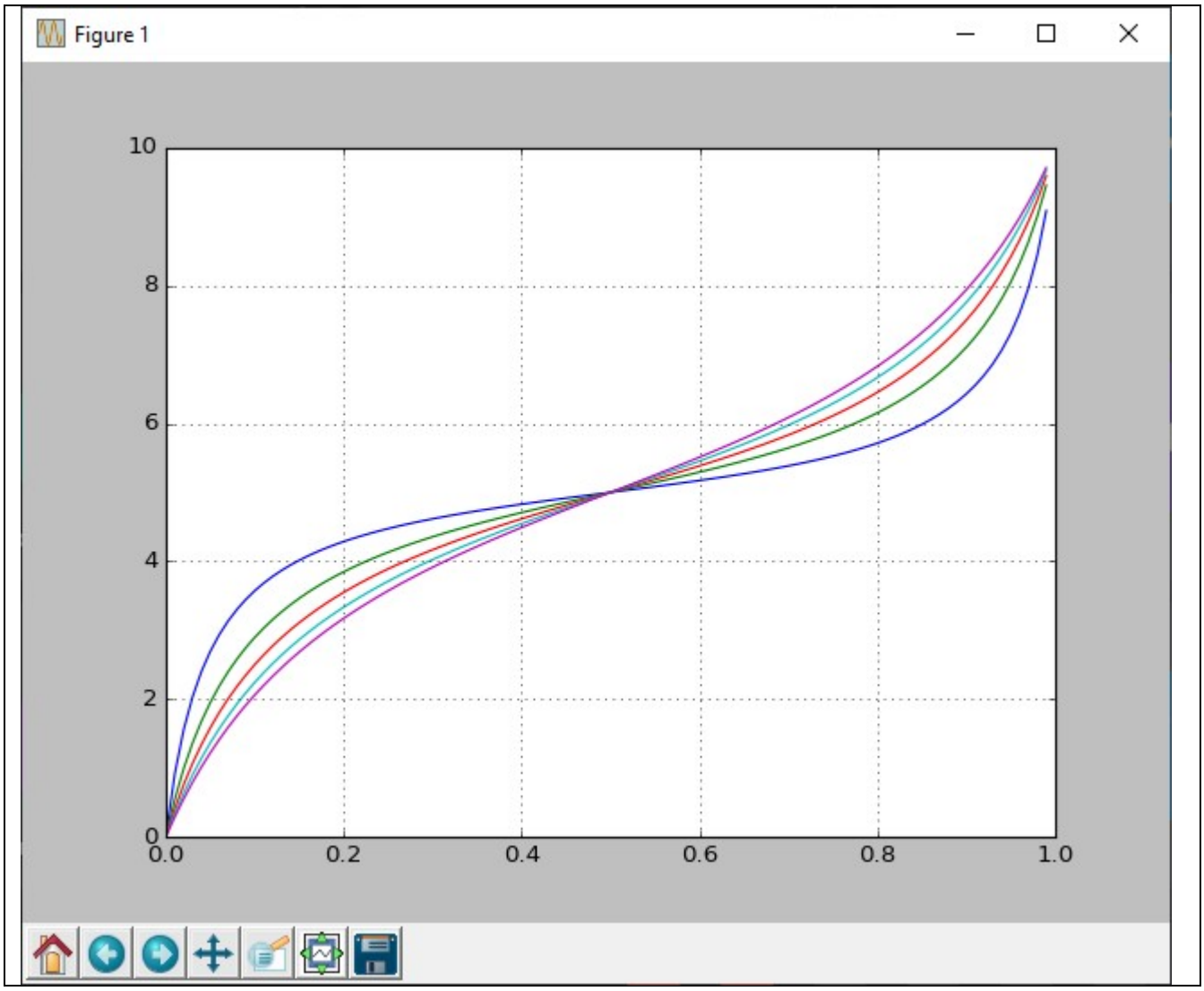
Trasduttore a potenziometro lineare con resistenza di carico

```
R=1000
x = list(np.arange(0, 1, 0.01))

VU=list(np.arange(0, 1, 0.01))#Inizializzazione vettore tensione di uscita VU
for j in range(1,5+1): #Ciclo for esterno per variazione RU
   RU=100*j
   for i in range(0,100):#Ciclo for interno per valutazione della formula
      VU[i]=10*x[i]*R*RU/(R*RU+x[i]*R*R-x[i]*x[i]*R*R)

   plt.plot(x,VU)

plt.grid()
plt.show()
```

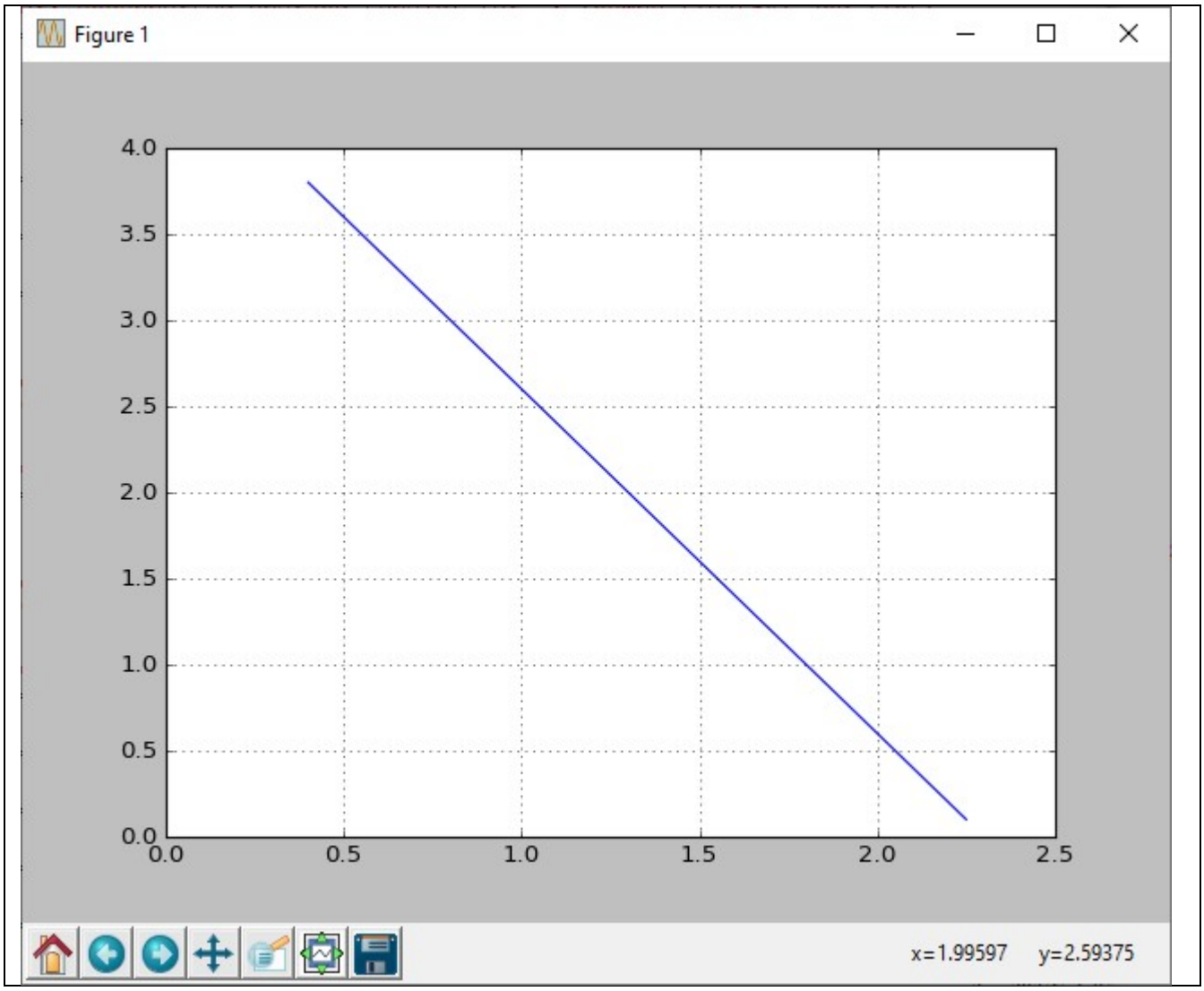


Trasduttore a potenziometro lineare con resistenza di linearizzazione

```
R=1000
x = list(np.arange(0, 1, 0.01))

VU=list(np.arange(0, 1, 0.01))
for j in range(1,5+1):
   RU=100*j
   for i in range(0,100):
      VU[i]=10*(x[i]*R**2*RU-x[i]**2*R**2*RU+x[i]*R*RU**2)/(2*x[i]*R**2*RU-2*x[i]**2*R**2*RU+R*RU**2)
   plt.plot(x,VU)

plt.grid()
plt.show()
```

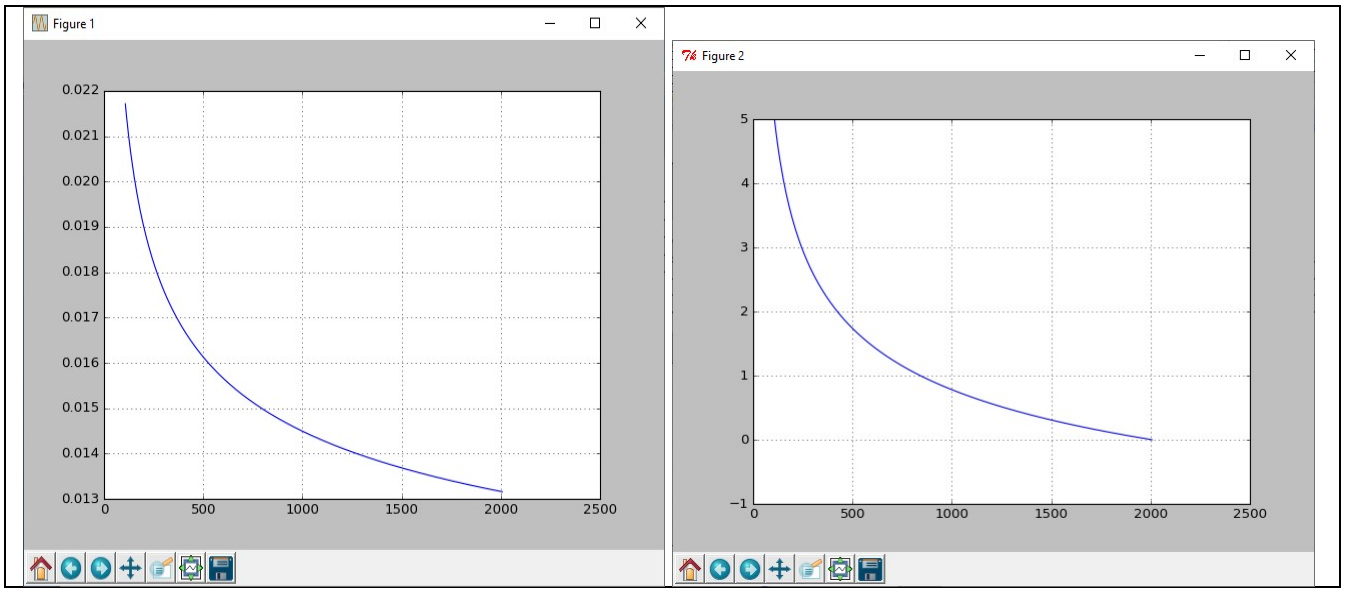


Broccio Robot - Condizionamento
R1=100 R2=263 V1=list(np.arange(0.4, 2.3, 0.05)) V2=2.3 Vo=[(R2/R1)*(V2-v1) for v1 in V1] plt.plot(V1,Vo) plt.grid() plt.show()



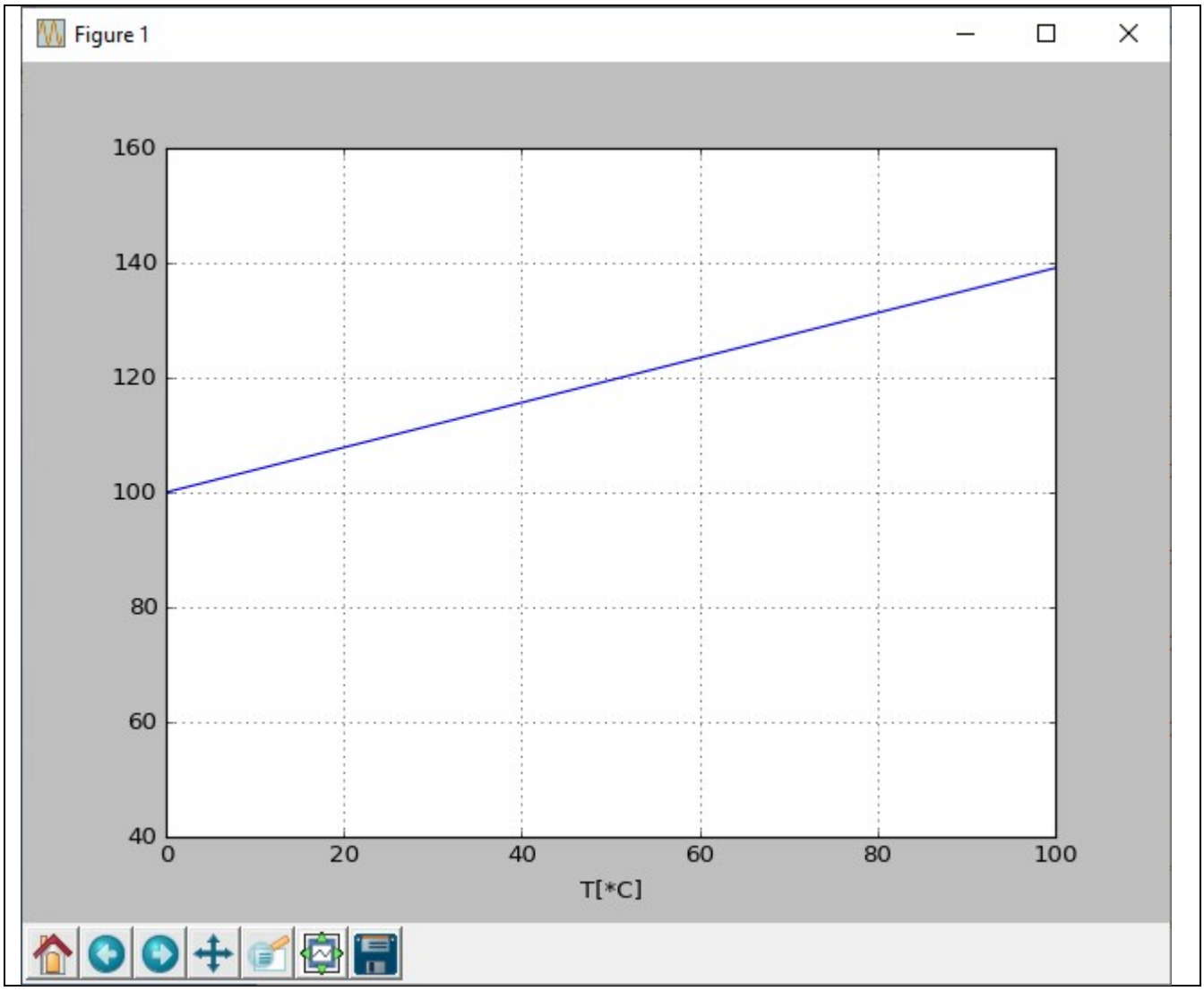
Condizionamento del trasduttore di Forza

```
N=256
C2 = [0] * N
D1 = [0] * N
E1 = [0] * N
C1=100
RT=584.2219
VMIN=7.686218

for i in range(0 , N):
   C2[i]=C1+((2000-100)/255.)
   D1[i]=1./(10*math.log(C1))
   E1[i]=D1[i]*RT-VMIN
   C1=C2[i]
plt.plot(C2,D1)
plt.grid()
plt.figure()
plt.plot(C2,E1)
plt.grid()
plt.show()
```



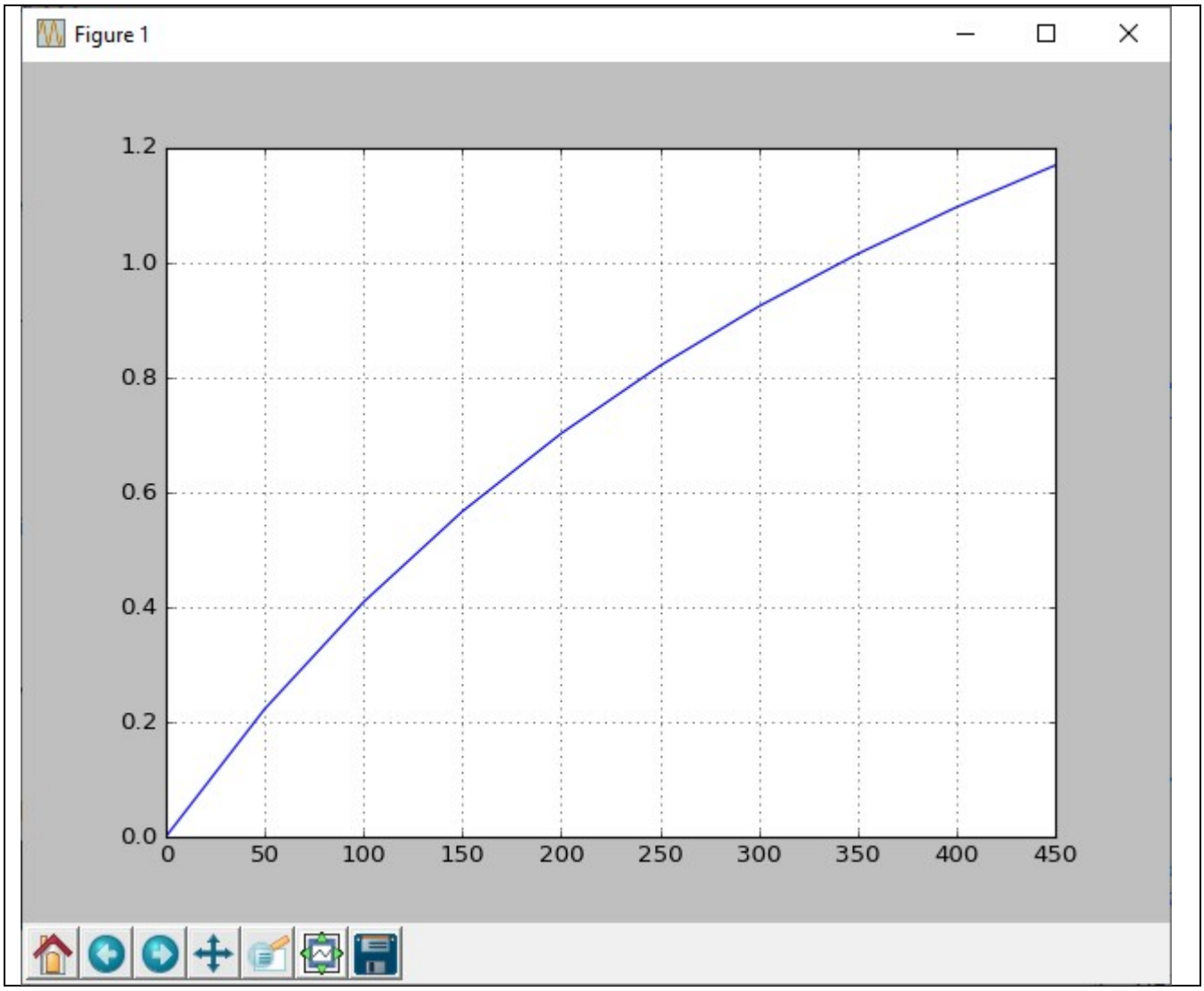
TERMORESISTENZA RTD

```
T = list(range(0,100+10,10))
a=3.9083E-03
R=[0]*len(T)
i=0
for t in T:
    R[i]=100*(1+a*t)
    i+=1
plt.plot(T,R)
plt.ylim(40,160)
plt.xlabel('T[*C]')
plt.grid()
plt.show()
```



MISURA DELLA TEMPERATURA CON CONFIGURAZIONE A PONTE

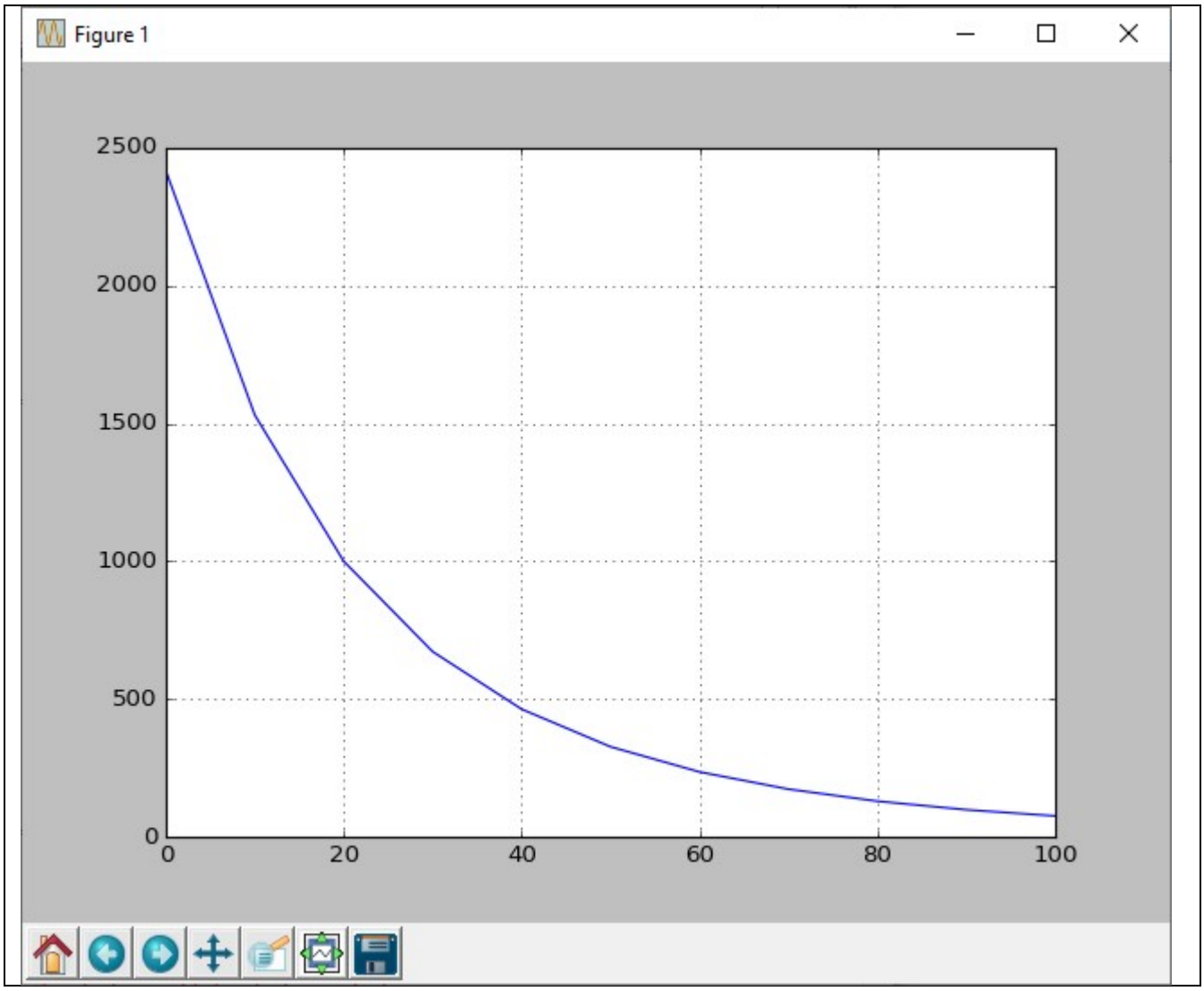
```
T = list(range(0,500,50))
a=3.9083E-03
R=[0]*len(T)
C=[0]*len(T)
i=0
for t in T:
   R[i]=100*(1+a*t)
   C[i]=5.*(R[i]-100)/(2.*(100.+R[i]))
   i+=1
ax=plt.plot(T,C)
print T,'\n',R
print
print C
plt.xlabel('T[*C]')
plt.grid()
plt.show()
```



TERMISTORI

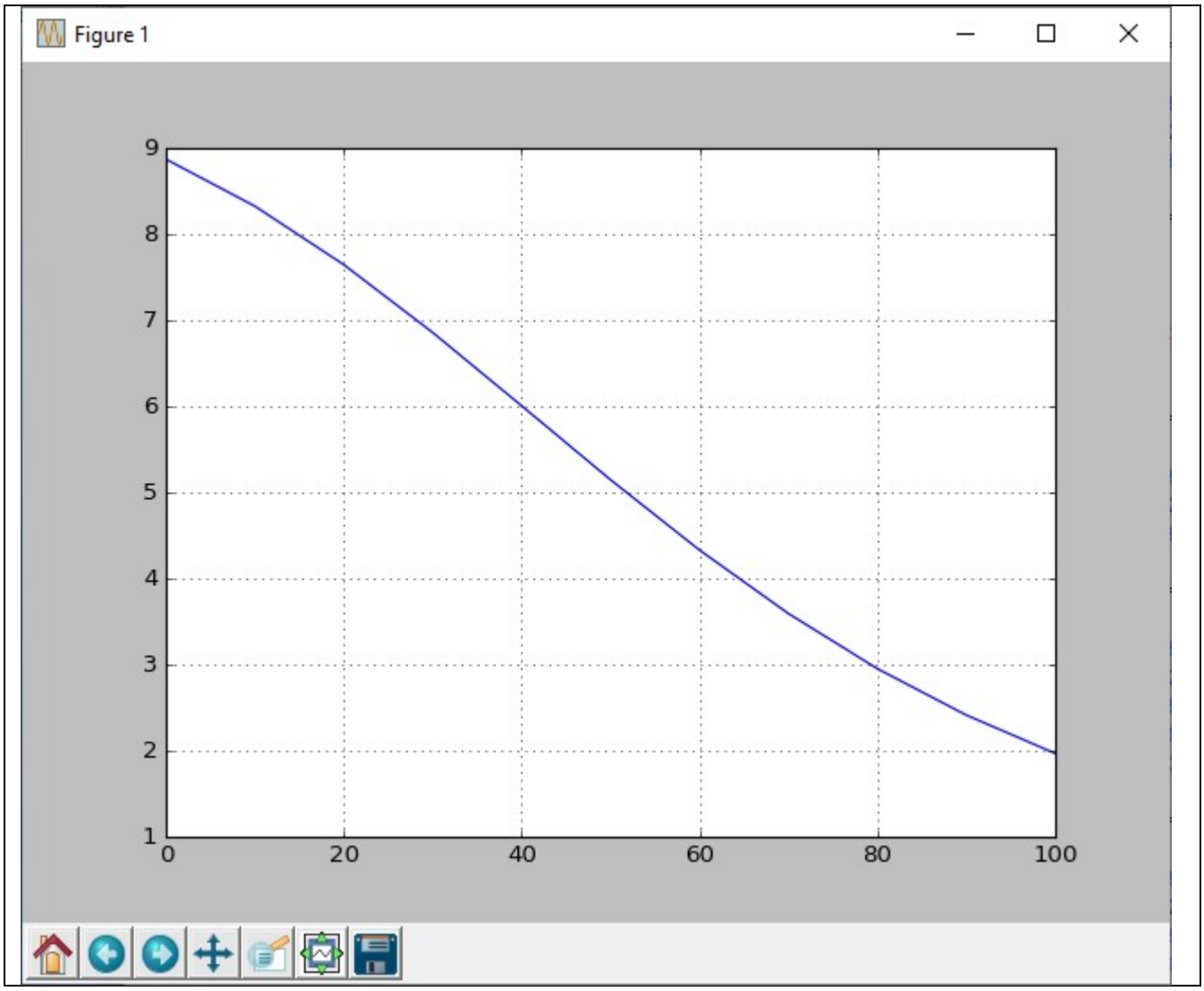
Analizzare il termistore NTC k25-ak. E diagrammare $R_T(T)$ per 0°C<T<100°C

```
T = list(range(0,100+10,10))
TK=[0]*len(T)
R=[0]*len(T)
T0=20
TK0=T0+273
R0=1000.
B=3530.
i=0
for t in T:
   TK[i]=T[i]+273
   #print -B*(TK[i]-TK0)/(TK[i]*TK0)
   R[i]=R0*math.exp(-B*(TK[i]-TK0)/(TK[i]*TK0) )
   i+=1

print TK,R,math.exp(1)

#print math.exp(0.88262136)
##print R0*math.exp(-B*(0-20)/(0*TK0))
plt.plot(T,R)
plt.grid()
plt.show()
```



…disegnare il grafico $V_u(T)$ con il circuito di linearizzazione

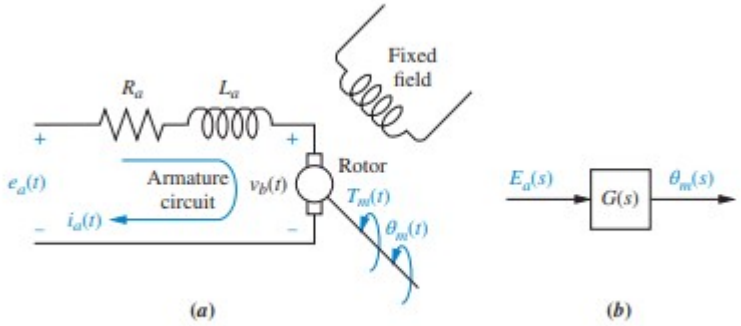
```
T = list(range(0,100+10,10))
TK=[0]*len(T)
R=[0]*len(T)
Vu=[0]*len(T)
T0=20
TK0=T0+273
R0=1000.
B=3530.
i=0
for t in T:
   TK[i]=T[i]+273
   R[i]=R0*math.exp(-B*(TK[i]-TK0)/(TK[i]*TK0) )
   Vu[i]=10*R[i]/(R[i]+308.)
   i+=1
plt.plot(T,Vu)
plt.xlabel(r'T[$\degree$C]')
plt.grid()
plt.show()
```



Attuatori

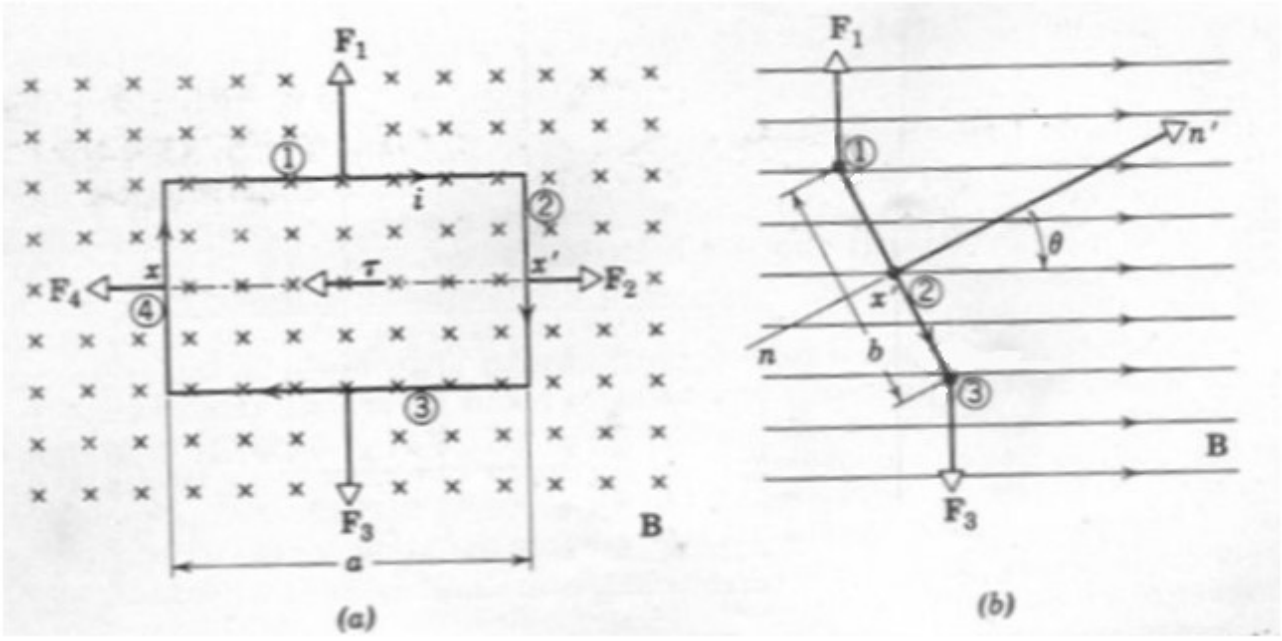
Il motore DC

Nella figura seguente un campo magnetico e' generato o da magneti permanenti o da un elettromagnete chiamato campo fisso. Un circuito rotante quale una spira, chiamato di armatura, nel quale circola una corrente $i_a(t)$, immerso in un campo magnetico B,



Il motore DC - (a) Schema (b) Schema a Blocchi

produce una coppia di forze su un conduttore di lunghezza ℓ pari a: $F=i_a * \ell * B$ che fa ruotare la spira.



Inoltre nel circuito d'armatura viene generata una fcem. $v_b(t) = BA\omega$ dove B e' il campo magnetico, A e' l'area della spira e ω la velocita' angolare della spira, cioe' anche: $v_b(t) = BA\frac{d\theta(t)}{dt}$, in Laplace:

$$V_b(s) = BAs\theta(s)$$

Quindi scrivendo l'equazione della maglia di armatura:

$$e_a(t) = R_a \cdot i_a(t) + L_a \frac{di_a(t)}{dt} + v_b(t)$$

Applicando Laplace:

$$E_a(s) = R_a \cdot I_a(s) + L_a \cdot s \cdot I_a(s) + V_b(s)$$
$$E_a(s) = (R_a + L_a \cdot s) \cdot I_a(s) + V_b(s)$$

Jntroducendo alcune costanti:

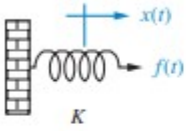
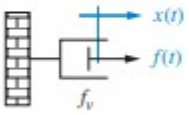
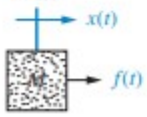
$T_m(s) = K_t \cdot I_a(s)$ $\quad Kt\ e'la\ costante\ di\ coppia\ del\ motore$

$V_b(s) = K_b s\theta(s)$ $\quad Kb\ e'detta\ costante\ di\ induzione\ elettromagnetica$

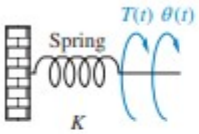
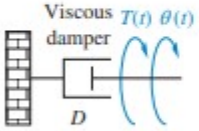
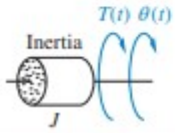
Si ottiene:

$$E_a(s) = (R_a + L_a \cdot s) \cdot \frac{T_m(s)}{K_t} + K_b s\theta(s)$$

Sistemi traslazionali meccanici, equazioni caratteristiche

Component	Force-velocity	Force-displacement	Impedence $Z_M(s) = F(s)/X(s)$
Spring 	$f(t) = K\int_0^t v(\tau)d\tau$	$f(t) = Kx(t)$	K
Viscous damper 	$f(t) = f_v v(t)$	$f(t) = f_v \frac{dx(t)}{dt}$	$f_v s$
Mass 	$f(t) = M\frac{dv(t)}{dt}$	$f(t) = M\frac{d^2x(t)}{dt^2}$	Ms^2

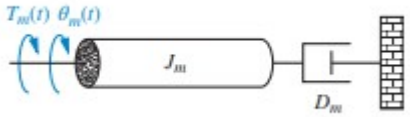
Sistemi rotazinali meccanici, equazioni caratteristiche

Component	Torque-angular velocity	Torque-angular displacement	Impedence $Z_M(s) = T(s)/\theta(s)$
	$T(t) = K\int_0^t \omega(\tau)d\tau$	$T(t) = K\theta(t)$	K
	$T(t) = D\omega(t)$	$T(t) = D\frac{d\theta(t)}{dt}$	Ds
	$T(t) = J\frac{d\omega(t)}{dt}$	$T(t) = J\frac{d^2\theta(t)}{dt^2}$	Js^2

ANALOGIA FRA GRANDEZZE LINEARI E ANGOLARI

LINEARI		ANGOLARI	
Spostamento	s	Spostamento	θ
Velocita'	v	Velocita'	ω
Accelerazione	a	Accelerazione	α
Massa	m	Momento d'Inerzia	J
Forza	F	Momento o coppia	τ
Quantita' di moto	Q=mv	Momento	J ω

Rappresentando il carico sul motore come un corpo con momento di inerzia J_m e viscosita' D_m



Avremo che la forza applicata sull'albero motore o meglio la coppia sara' data dall'equazione:

$$T_m(t) = J_m\frac{d^2\theta}{dt^2} + D_m\frac{d\theta}{dt}$$

$$T_m(s) = J_m s^2\theta(s) + D_m s\theta(s)$$

Sostiruendo in:

$$E_a(s) = (R_a + L_a\cdot s)\cdot\frac{T_m(s)}{K_t} + K_b s\theta(s)$$

Si ottiene:

$$E_a(s) = (R_a + L_a\cdot s)\cdot\frac{[J_m s^2\theta(s) + D_m s\theta(s)]}{K_t} + K_b s\theta(s)$$

Se $L_a << R_a$:

$$E_a(s) = \frac{R_a}{K_t}[(J_m s + D_m) + K_b]\cdot s\theta(s)$$

Dopo semplificazioni si ottiene:

$$\frac{\theta(s)}{E_a(s)} = \frac{K_t/(R_a\cdot J_m)}{s\left[s + \frac{1}{J_m}\cdot\left(D_m + \frac{K_t\cdot K_b}{R_a}\right)\right]}$$

Sostituendo:

$$V_b(s) = K_b s\theta(s)$$

e:

$$I_a(s) = \frac{T_m(s)}{K_t}$$

in

$$E_a(s) = (R_a + L_a \cdot s) \cdot I_a(s) + V_b(s)$$

si ottiene ponendo L_a=0:

$$E_a(s) = (R_a) \cdot \frac{T_m(s)}{K_t} + K_b s\theta(s)$$

antitrasformando:

$$e_a(t) = \frac{R_a}{K_t} \cdot \tau_m(t) + K_b \omega_m(t)$$

ricavando:

$$\tau_m(t) = -\frac{K_b \cdot K_t}{R_a}\omega_m(t) + \frac{K_t}{R_a}e_a(t)$$

Se si applica al motore una tensione continua Ea questo si mettera' a girare con coppia Tm e wm costanti, quindi a regime a seguito di una tensione di ingresso costante si puo' togilere la variabile tempo, ovvero:

$$\tau_m = -\frac{K_b \cdot K_t}{R_a}\omega_m + \frac{K_t}{R_a}e_a$$

Questa relazione definisce una retta, detta curva coppia-velocita'. La coppia per una velocita' angolare nulla, viene dettacoppia di spunto o di stallo Tstall che varra':

$$\tau_{stall} = \frac{K_t}{R_a}e_a$$

Il valore della velocita' angolare per una coppia nulla e' detto velocita' a vuoto, wno-load:

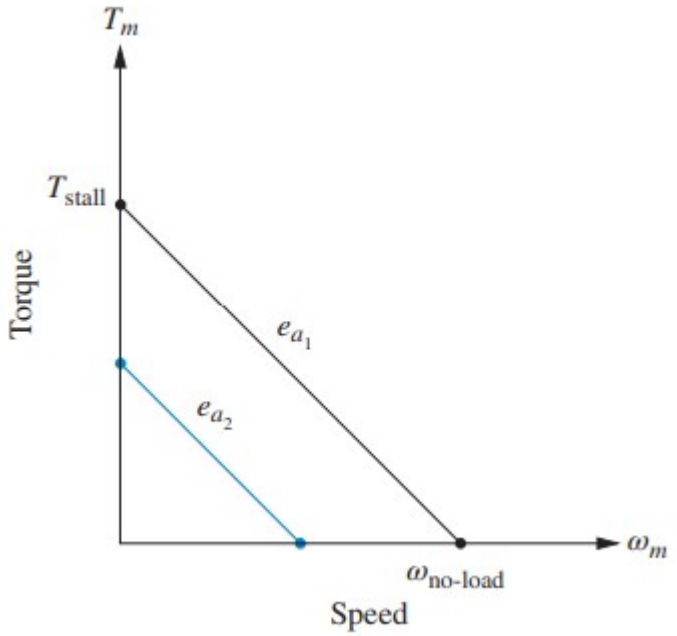
$$\omega_{no-lo} \quad = \frac{e_a}{K_b}$$

A questo punto le costanti elettriche della f.d.t. possono essere ricavate:

$$\frac{K_t}{R_a} = \frac{\tau_{stall}}{e_a}$$

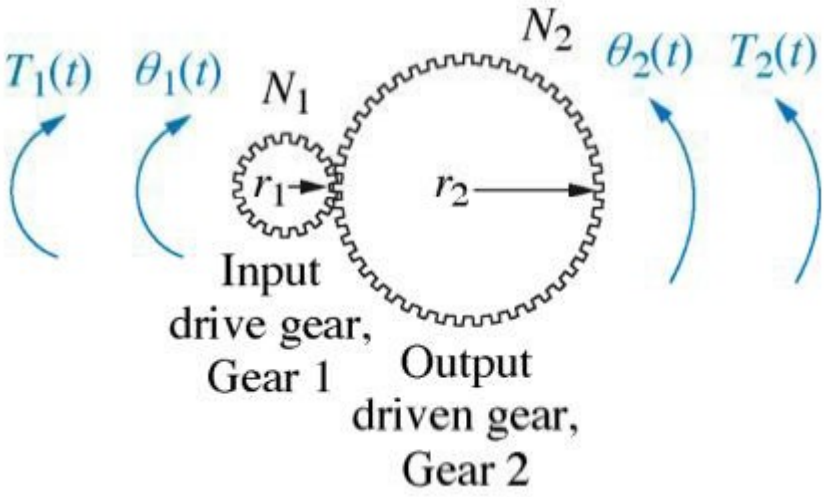
e

$$K_b = \frac{e_a}{\omega_{no-load}}$$



INGRANAGGI

Nella figura seguente un ingranaggio di ingresso con raggio r1 e N1 i denti vengono ruotati di un angolo θ1(t) a causa di una coppia, τ_1(
t). In uscita l'ingranaggio con raggio r2 e denti N2 risponde ruotando di un angolo θ2(t) e producendo una coppia, τ_2(t). Troviamo ora la relazione tra la rotazione di θ1(t) e θ2(t).



La relazione tra spostamento angolare e spazio percorso nel moto circolare e':

$$\theta(t) \cdot r = s(t)$$

Dato che se non ci sono scivolamenti lo spazio (o anche la velocita') e' lo stesso:

$$\theta_1(t) \cdot r_1 = \theta_2(t) \cdot r_2$$

$$\frac{\theta_1}{\theta_2} = \frac{r_2}{r_1}$$

Oppure con le velocita' angolari:

$$\frac{\omega_1}{\omega_2} = \frac{r_2}{r_1}$$

Esprimendo invece in numero di giri/minuto ed uguagliando la velocita' peri0efriche:

$$v = \frac{2\pi \cdot n_1 \cdot r_1}{60} = \frac{2\pi \cdot n_2 \cdot r_2}{60} \Rightarrow \frac{n_2}{n_1} = \frac{r_1}{r_2}$$

Nel caso di ruota dentata di raggio r con N denti e relativo passo tra i denti p avremo $p*N=2\pi r$
Per due ruote dentate con lo stesso passo avremo quindi:
$p*N_1=2\pi r_1$ $p*N_2=2\pi r_2$
quindi:

$$\frac{\theta_1}{\theta_2} = \frac{r_2}{r_1} = \frac{N_2}{N_1}$$

Quindi il rapporto fra gli spostamenti angolari e' inversamente proporsionale al rapporto tra numeri di denti.
Se si ipotizza che non ci siano perdite per attrito usura avremo l'uguaglianza tra potenza appliata sulla prima ruota motrice e potenza sulla ruota in uscita. Cioe' nel caso di rotazione pura avremo la potenza P:

$$P = \tau \cdot \omega$$

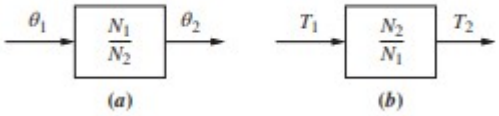
Uguale al prodotto tra coppia applicata e velocita' angolare (analogo a F*v nel caso di moto traslazionale)
Uguagliando le due potenze (quindi rendimento=1):

$$\tau_1 \cdot \omega_1 = \tau_2 \cdot \omega_2$$

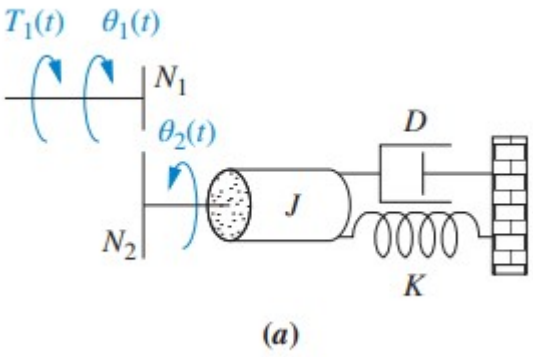
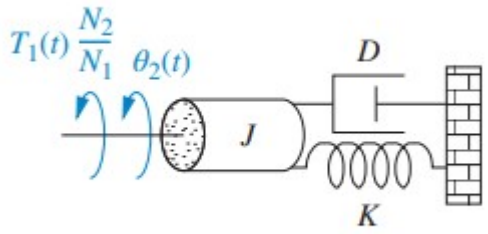
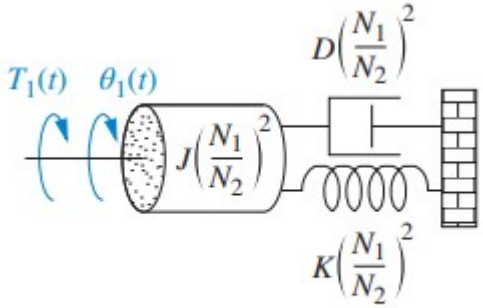
Cioe'

$$\frac{\tau_2}{\tau_1} = \frac{\theta_1}{\theta_2} = \frac{r_2}{r_1} = \frac{N_2}{N_1}$$

Quindi le coppie sono direttamente proporzionali al numero dei denti.
Rappresentando tramite sistemi a blocchi:

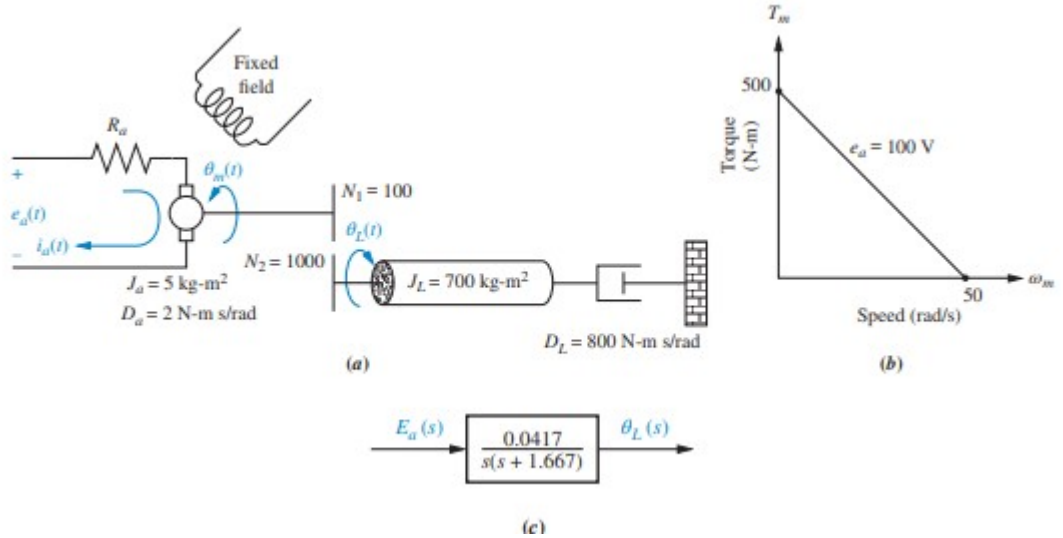


Vediamo cosa succede alle impedenze meccaniche nel caso in cui ci siano ingranaggi.

Sistema rotante con ingranaggi	 (a)
Sistema equivalente con la coppia in ingresso riportata in uscita all'ingranaggio: $T_2(s) = (Js^2 + Ds + K)\theta_2(s) \Rightarrow$ $T_1(s)\frac{N_2}{N_1} = (Js^2 + Ds + K)\theta_1(s)\frac{N_1}{N_2} \Rightarrow$ $T_1(s) = (Js^2 + Ds + K)\theta_1(s)\left(\frac{N_1}{N_2}\right)^2 \Rightarrow$ $T_1(s) = \left(J\left(\frac{N_1}{N_2}\right)^2 s^2 + D\left(\frac{N_1}{N_2}\right)^2 s + K\left(\frac{N_1}{N_2}\right)^2\right)\theta_1(s)$	 (b)
Sistema equivalente con le impedenze in uscita riportata in ingresso all'ingranaggio $T_1(s) = (Js^2 + Ds + K)\theta_1(s) \Rightarrow$ $T_2(s)\frac{N_1}{N_2} = (Js^2 + Ds + K)\theta_2(s)\frac{N_2}{N_1} \Rightarrow$ $T_2(s) = (Js^2 + Ds + K)\theta_2(s)\left(\frac{N_2}{N_1}\right)^2 \Rightarrow$ $T_2(s) = \left(J\left(\frac{N_2}{N_1}\right)^2 s^2 + D\left(\frac{N_2}{N_1}\right)^2 s + K\left(\frac{N_2}{N_1}\right)^2\right)\theta_2(s)$	 (c)

Esempio:

Dato il sistema e la curva coppia-velocita' riportate nell figura seguente, determinare la f.d.t. $\theta_L(s)/E_a(s)$



Dato che l'inerzia e lo smorzamento cdel carico puo' essere riporetato all'ingreasso dell'ingranaggio si ha:

$$J_m = J_a + J_L\left(\frac{N_1}{N_2}\right)^2 \qquad D_m = D_a + D_L\left(\frac{N_1}{N_2}\right)^2$$

Dove il pedice a indica l'armatura e L il carico. Quindi se si sostituiscono i valori:

$$J_m = J_a + J_L\left(\frac{N_1}{N_2}\right)^2 = 5 + 700\left(\frac{1}{10}\right)^2 = 12$$

$$D_m = D_a + D_L\left(\frac{N_1}{N_2}\right)^2 = 2 + 800\left(\frac{1}{10}\right)^2 = 10$$

Dato che:

$$T_{stall} = 500 \qquad \omega_{no-load} = 50 \qquad e_a = 100$$

Si possono ottenere le costanti:

$$\frac{K_t}{R_a} = \frac{T_{stall}}{e_a} = \frac{500}{100} = 5$$

$$K_b = \frac{e_a}{\omega_{no-load}} = \frac{100}{50} = 2$$

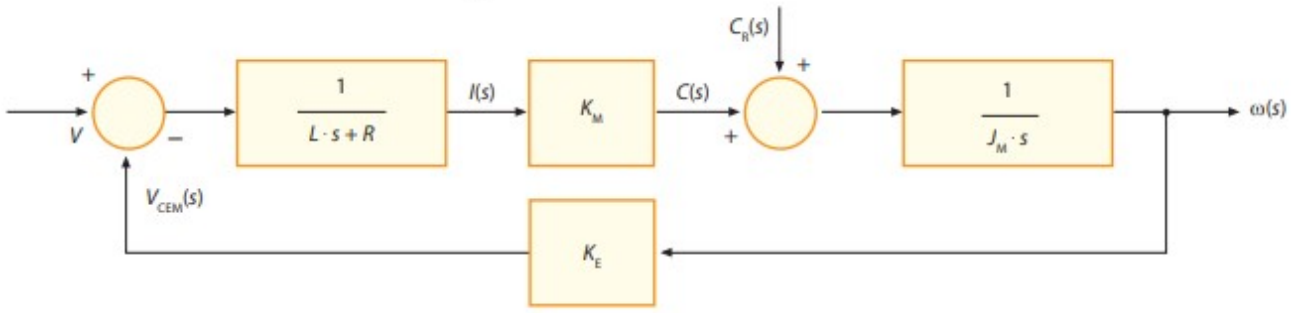
Dalla:

$$\frac{\theta_m(s)}{E_a(s)} = \frac{K_t/(R_a \cdot J_m)}{s\left[s + \frac{1}{J_m}\cdot\left(D_m + \frac{K_t \cdot K_b}{R_a}\right)\right]} = \frac{5/12}{s\left\{s + \frac{1}{12}[10 + (5)(2)]\right\}} = \frac{0.417}{s(s + 1.667)}$$

Si ricava dato che: $\theta_L(s) = \left(\frac{N_1}{N_2}\right)\theta_m(s)$, con N1/N2=100/1000=1/10

$$\frac{\theta_L(s)}{E_a(s)} = \frac{1}{10}\frac{0.417}{s(s + 1.667)} = \frac{0.0417}{s(s + 1.667)}$$

Risposta allo scalino di un motore CC:



La funzione di trasferimento a vuoto, ovvero con $C_R(s) = 0$, vale, assimilando le costanti a un'unica K:

$$G(s) = \frac{\frac{K_M}{Ls+R}\cdot\frac{1}{J_M s}}{1+\frac{K_M K_E}{Ls+R}\cdot\frac{1}{J_M s}} = \frac{K}{K^2 + (Ls+R)\cdot J_M s} = \frac{\frac{1}{K}}{1+(1+s\tau_E)s\tau_M} = \frac{\frac{1}{K}}{s^2\tau_e\tau_m + s\tau_m + 1}$$

nella quale:

$$\tau_E = \frac{L}{R} \qquad \tau_M = \frac{J_M R}{K^2}$$

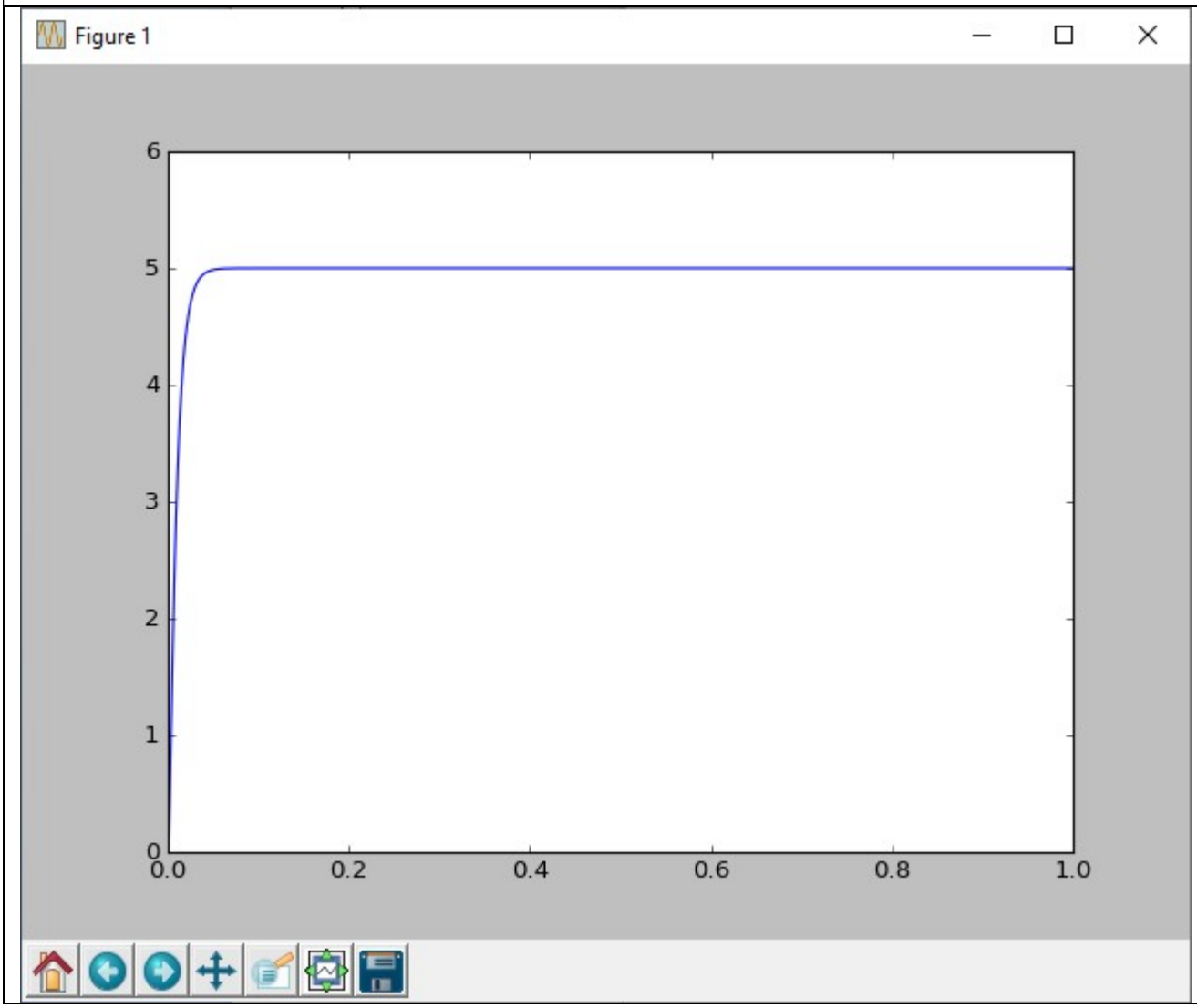
Quindi dato il sistema (Figura precedente con Cr(s)=0)

$$G(s) = \frac{K}{Jm \cdot La \cdot s^2 + Ra \cdot Jm \cdot s}$$

retroazionato con il blocco costante K, calcolare la risposta al gradino.

```
La=12E-3
Ra=8.
Jm=0.05E-3
K=0.2
num=[K]
den=[La*Jm,Jm*Ra,0]
G=control.tf(num,den)
print G
```

```
GR=control.feedback(G,K)
print GR   #dividere num e den per il coeff di s**2 per ottenere la formula presente nel libro
t = np.linspace(0, 1, 1000)
_,y=control.step_response(GR,T=t)

plt.plot(t,y)
plt.show()
>>>

        0.2
--------------------
6e-07 s^2 + 0.0004 s


          0.2
----------------------------
6e-07 s^2 + 0.0004 s + 0.04

>>>
```



Regole di tracciamento del diagramma di Bode:
Si sommano i diagrammi dei singoli blocchi elementari oppure,
si segnano sull'asse delle ascisse i valori di ω corrispondenti ai moduli dei poli (crocette) e degli zeri

(cerchietti). Se il sistema non contiene nella sua f.d.t nessun termine nell'origine cioe' $j\omega$ o $1/j\omega$, il diagramma del guadagno parte con pendenza zero e con ordinata uguale a $20*\log G_{st}$. Quindi la pendenza aumenta d uno in corrispondenza di uno zer, e dimunuisce in corrispondenza di ogni polo. Analogamente lo sfasamento iniziera' da zero, e passera' a 90° in corrispondenza della pandenza +1, 180° in corrispondenza della pendenza +2, -90° alla pendenza -1 e cosi via; il passaggio da uno all'altro sfasamento avviene nella banda di pulsazioni fra 1/10 e 10 volte quella di ciascun polo o zero; se questi sono vicini fra loro, si ha un'accentuaszioneo una compensazione delle relative curve degli sfasamenti. Se invece la f.d.t. contiene una derivazione o una integrazione (cioe' una radice nell'origine) si puo' procedere come segue:

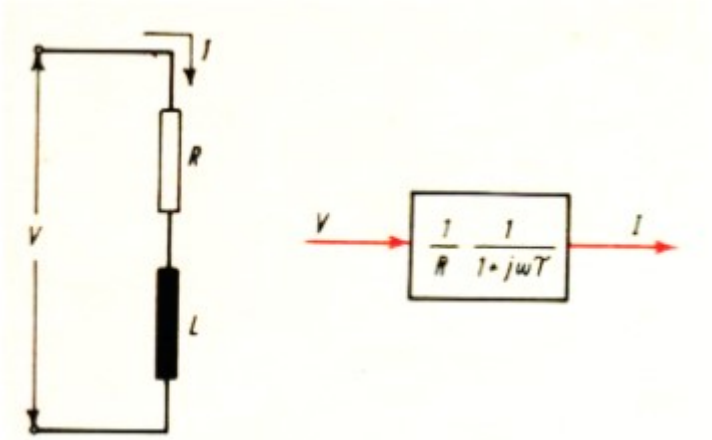
1)si traccia il diagramma dei guadagni senza considerare il termine $j\omega$.

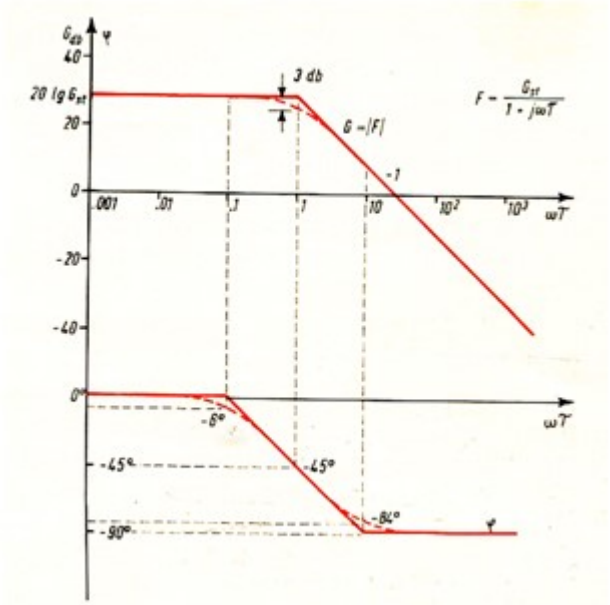
2)partendo dal punto di diagramma corrispondente alla pulsazione $\omega=1$, si varia la pendenza di -1 se vi e' un'integrazione, o di +1 se si ha una derivazione (il diagramma quindi partira' con pendenza -1 o +1).

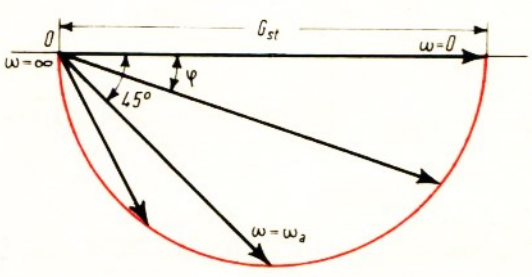
Se vi fosse una doppia derivazione o una doppia integrazione,la pendenza dovrebbe essere variata di +2 o di -2. In questo caso il diagramma partirebbe con pendenza +2 o -2. Per il diagramma degli sfasamenti si procede come nel caso precedente, tenendo presente che lo sfasamento per w=0 sara' di +90° nel caso di una derivazione , -90° nel caso di una integrazione, + o -180° nel caso di una doppia derivazione o integrazione.

ESEMPI DI FUNZIONI DI TRASFERIMENTO

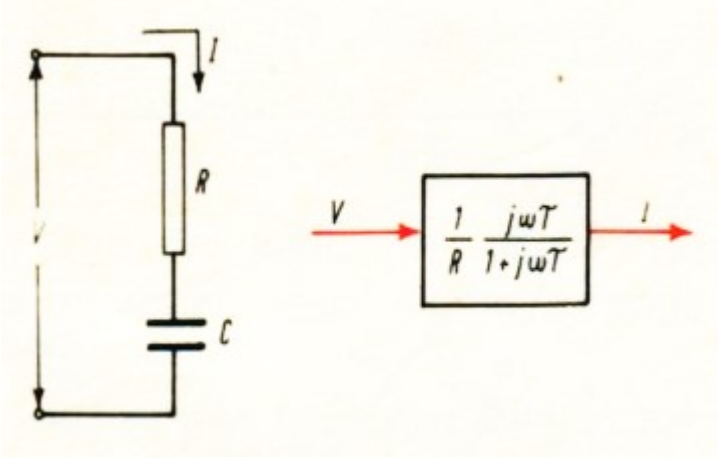
Blocco con una costante di tempo in ritardo

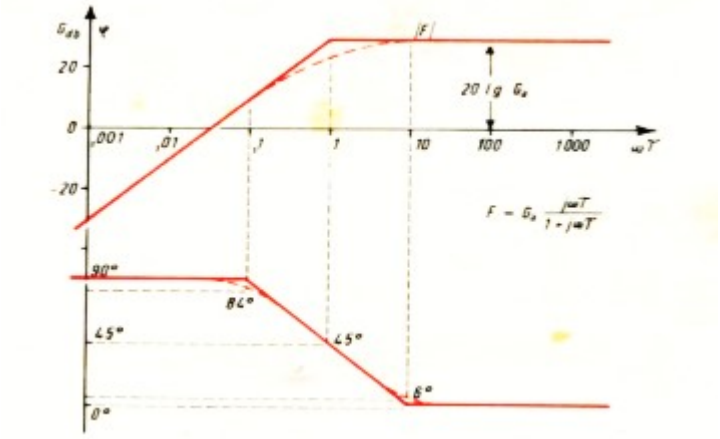


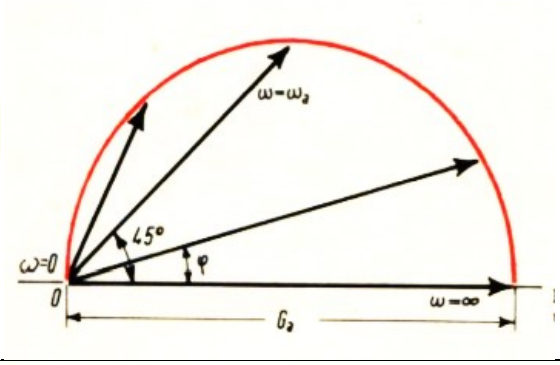




Blocco con una costante di tempo in anticipo







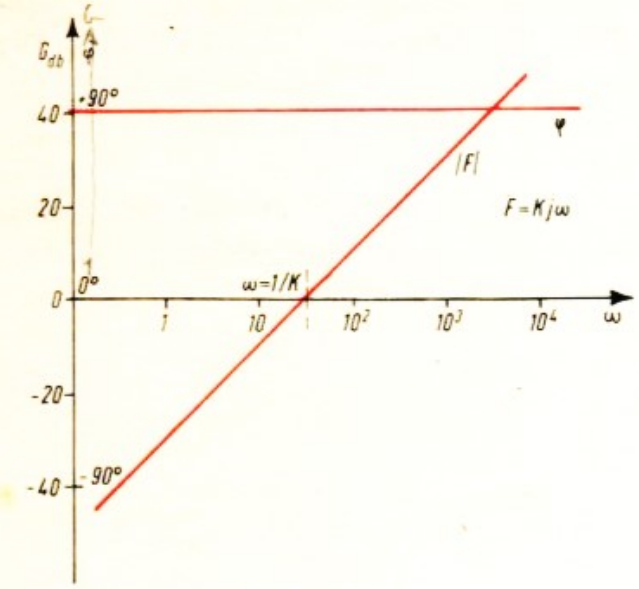


Diagramma di Bode per blocco derivatore puro.

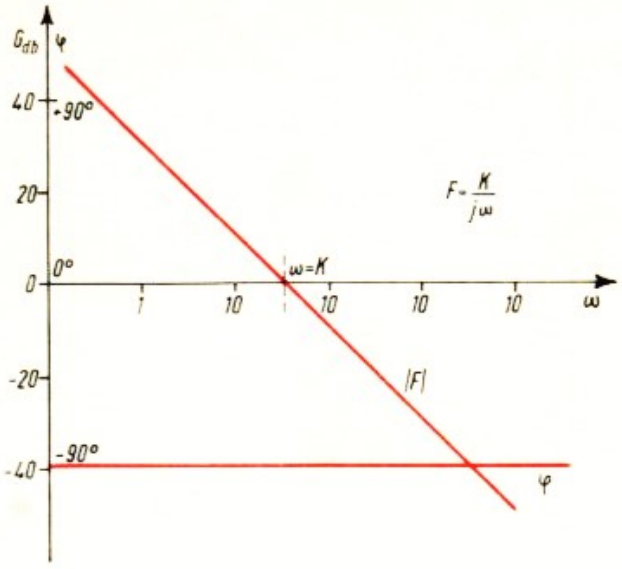


Diagramma di Bode per blocco integratore puro.

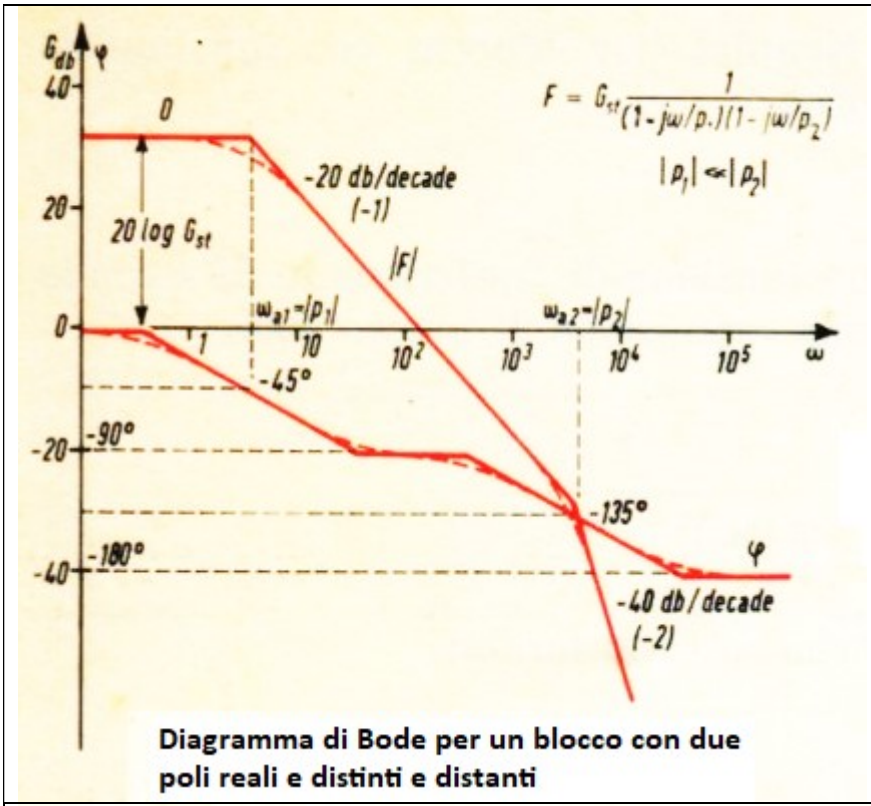
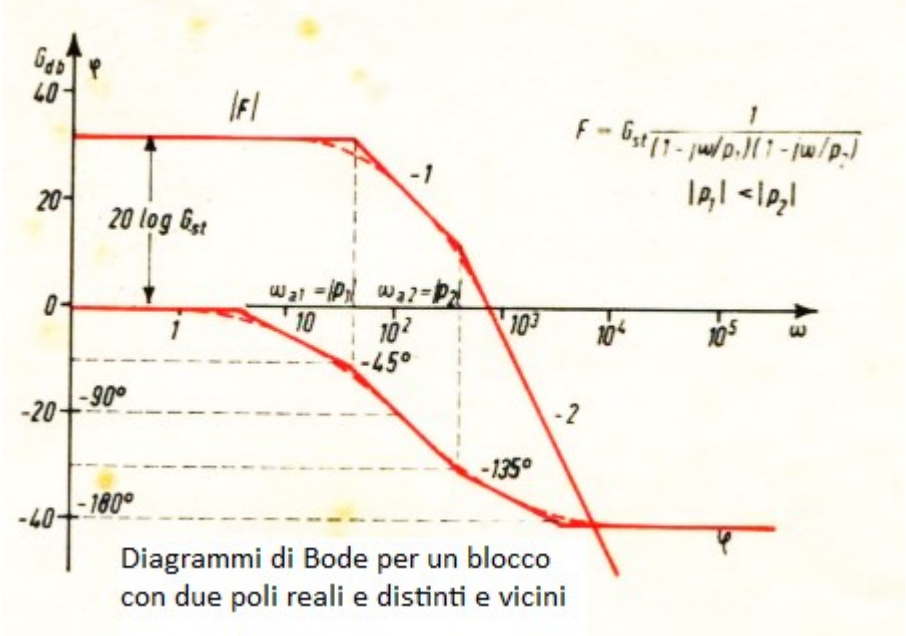


Diagramma di Bode per un blocco con due poli reali e distinti e distanti



Diagrammi di Bode per un blocco con due poli reali e distinti e vicini

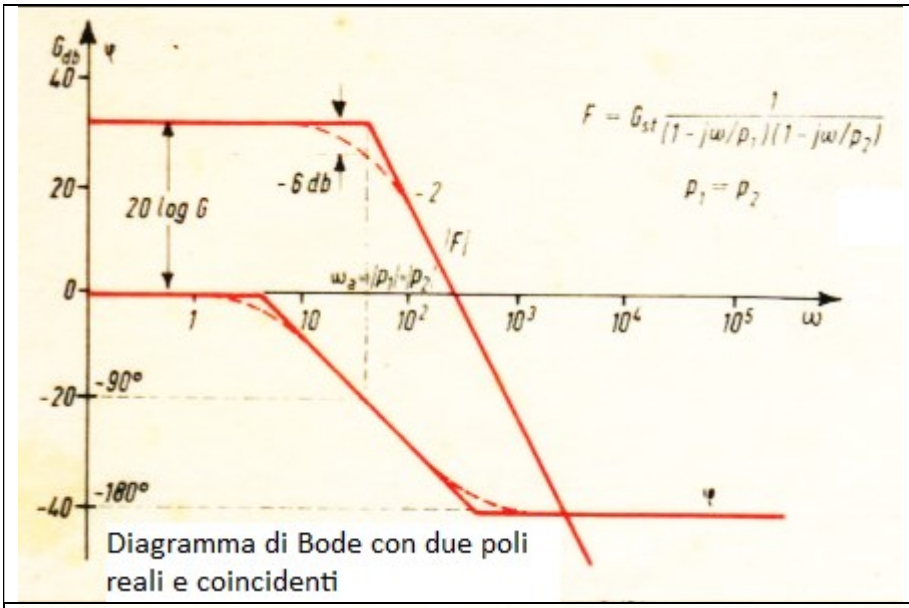


Diagramma di Bode con due poli reali e coincidenti

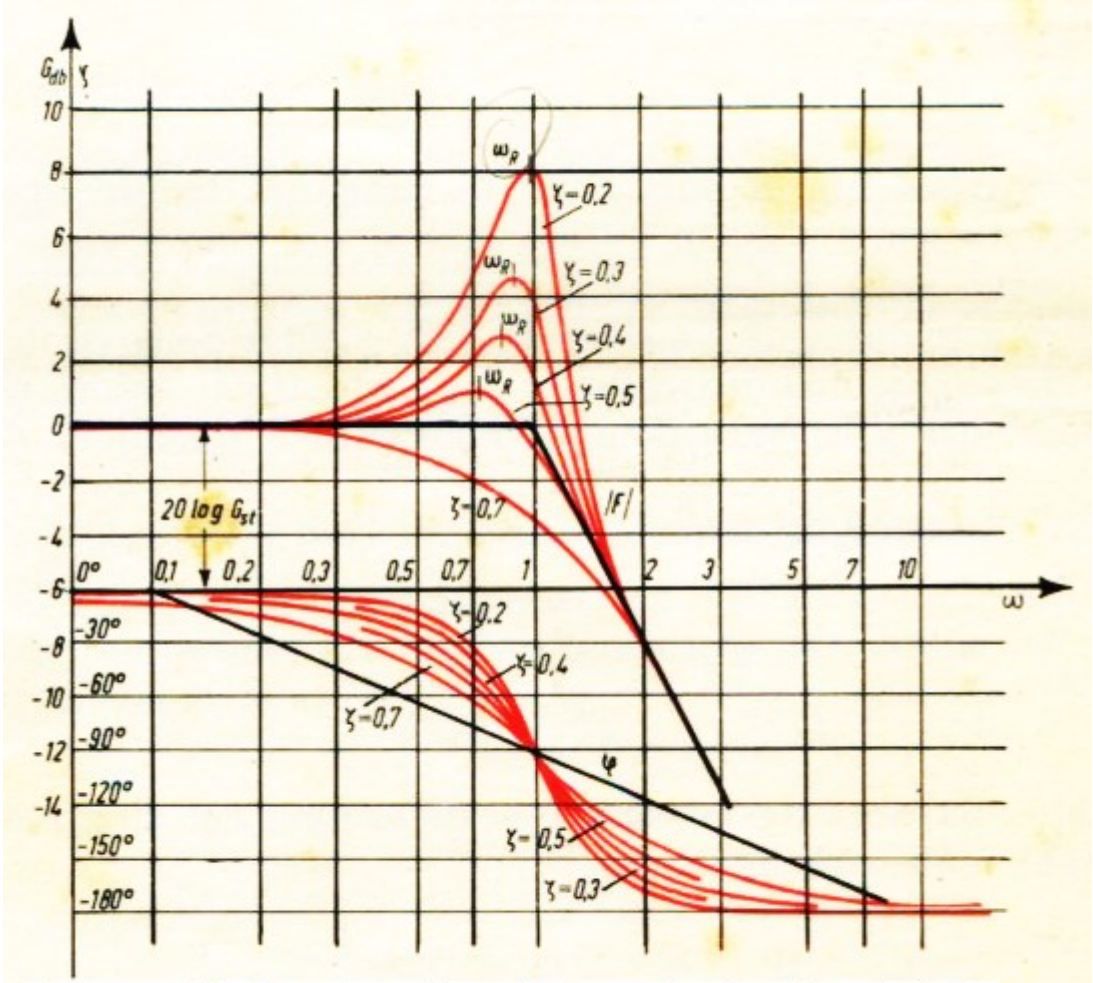


Diagramma di Bode per due poli complessi e coniugati per diversi valori del fattore di smorzamento

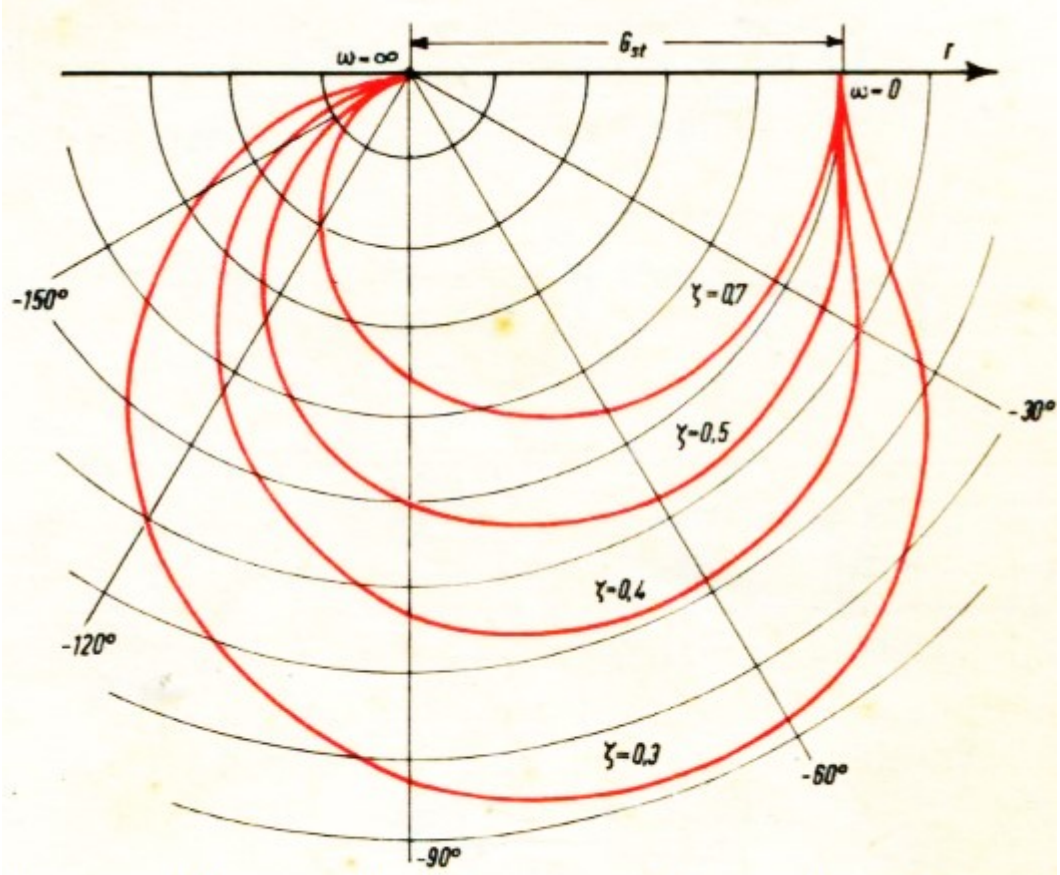
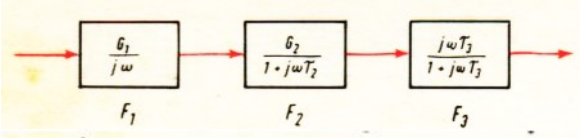


Diagramma di Nyquist per due poli complessi e coniugati per diversi valori dello smorzamento



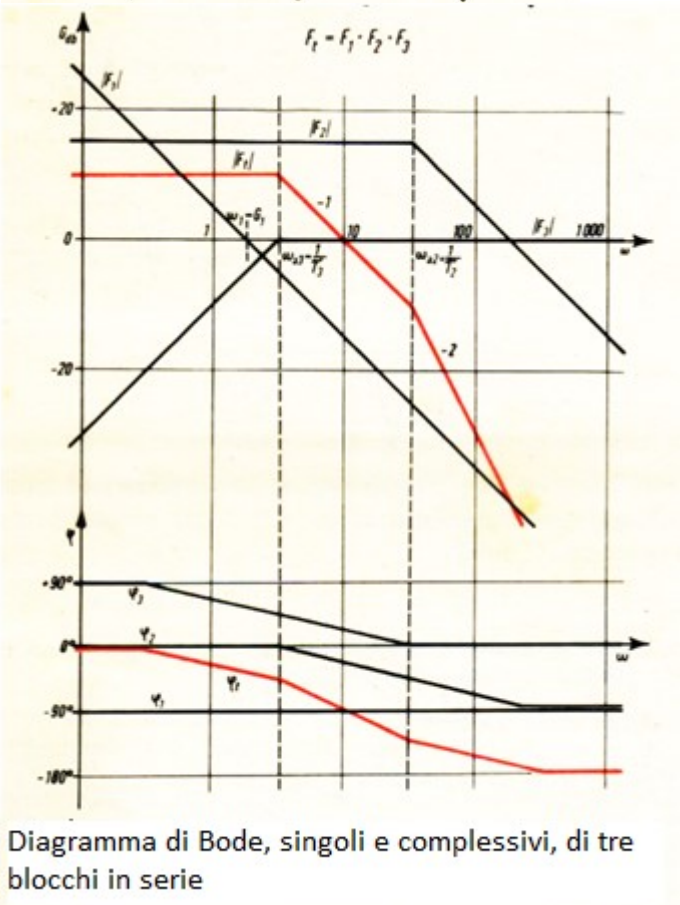
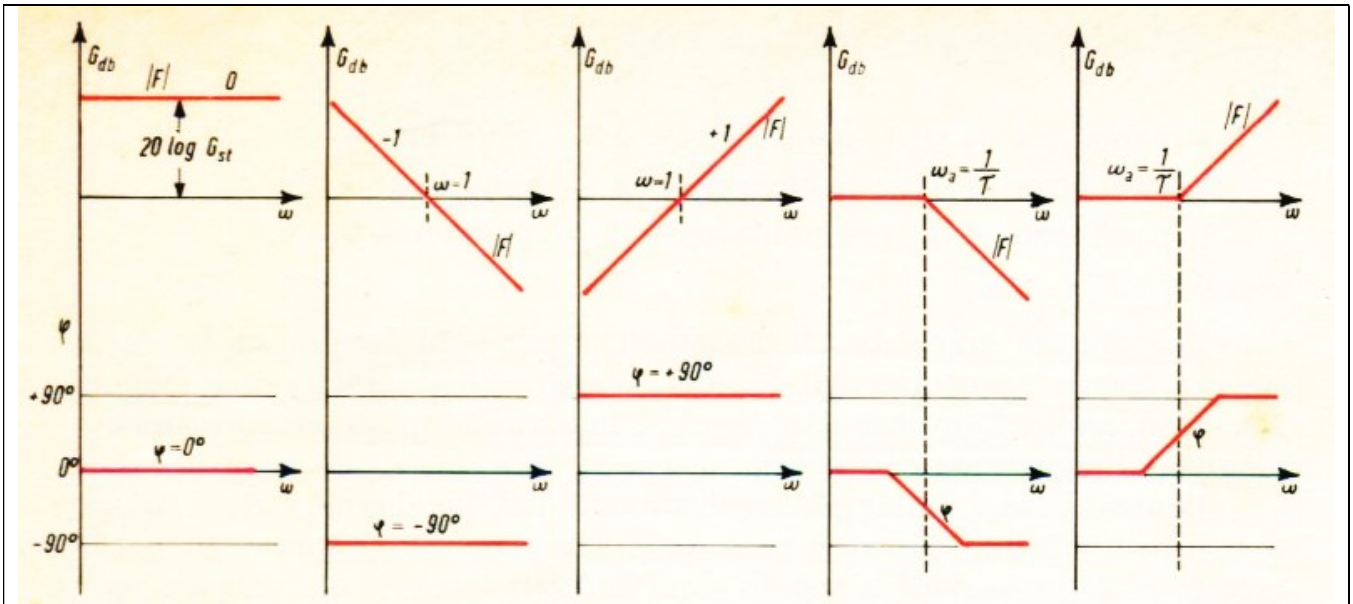


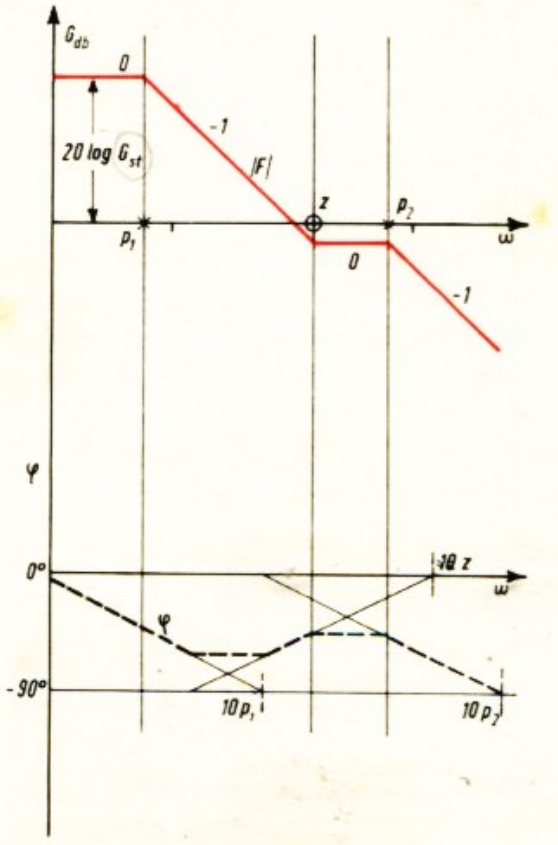
Diagramma di Bode, singoli e complessivi, di tre blocchi in serie



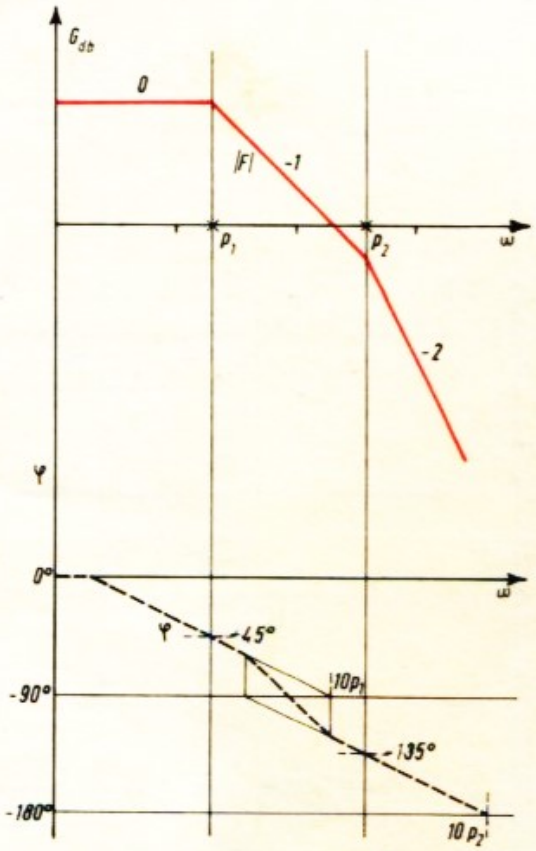
$$F = G_{st} \qquad F = \frac{1}{j\,\omega} \qquad F = j\,\omega \qquad F = \frac{1}{1 + j\,\omega\,\tau} \qquad F = 1 + j\,\omega\,\tau$$

$$F = \frac{1}{1 - j\,\frac{\omega}{p}} \qquad F = 1 - j\,\frac{\omega}{z}$$

Diagrammi di Bode per funzioni elementari

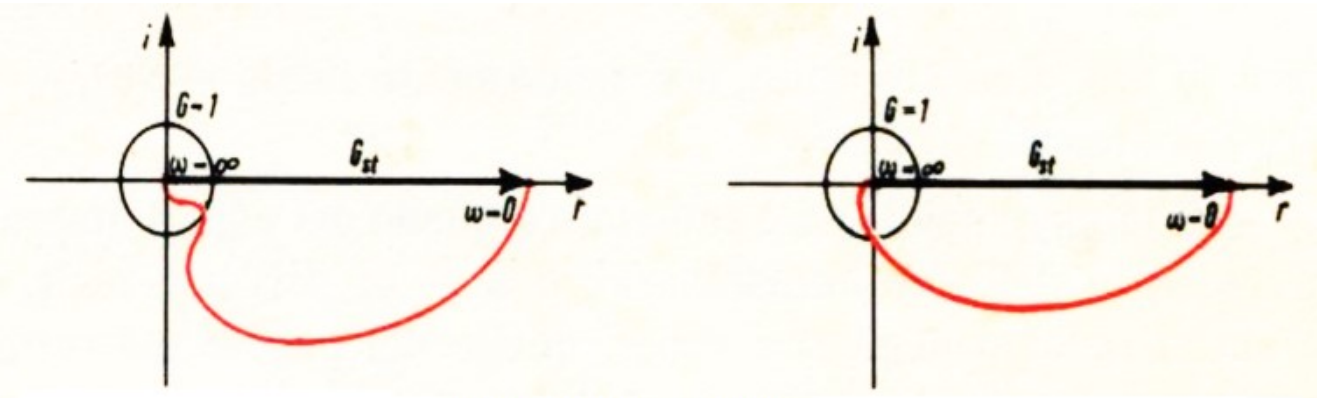


$$F(j\omega) = G_{st}\,\frac{1 - j\,\omega/z}{\left(1 - j\,\frac{\omega}{p_1}\right)\left(1 - j\,\frac{\omega}{p_2}\right)}$$

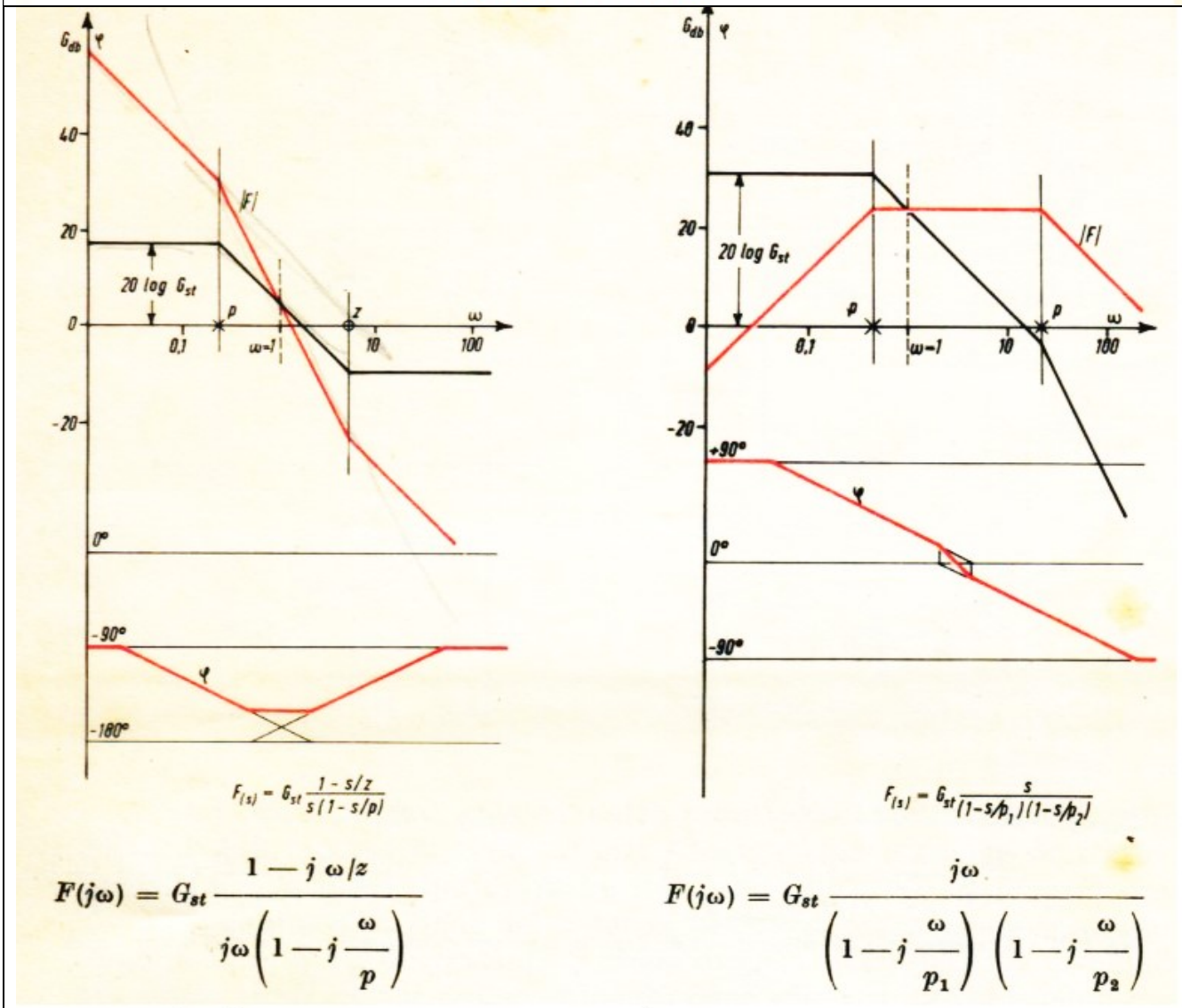


$$F(j\omega) = G_{st}\,\frac{1}{\left(1 - j\,\frac{\omega}{p_1}\right)\left(1 - j\,\frac{\omega}{p_2}\right)}$$

Tracciamento dei diagrammi di Bode in funzione della posizione dei poli e degli zeri



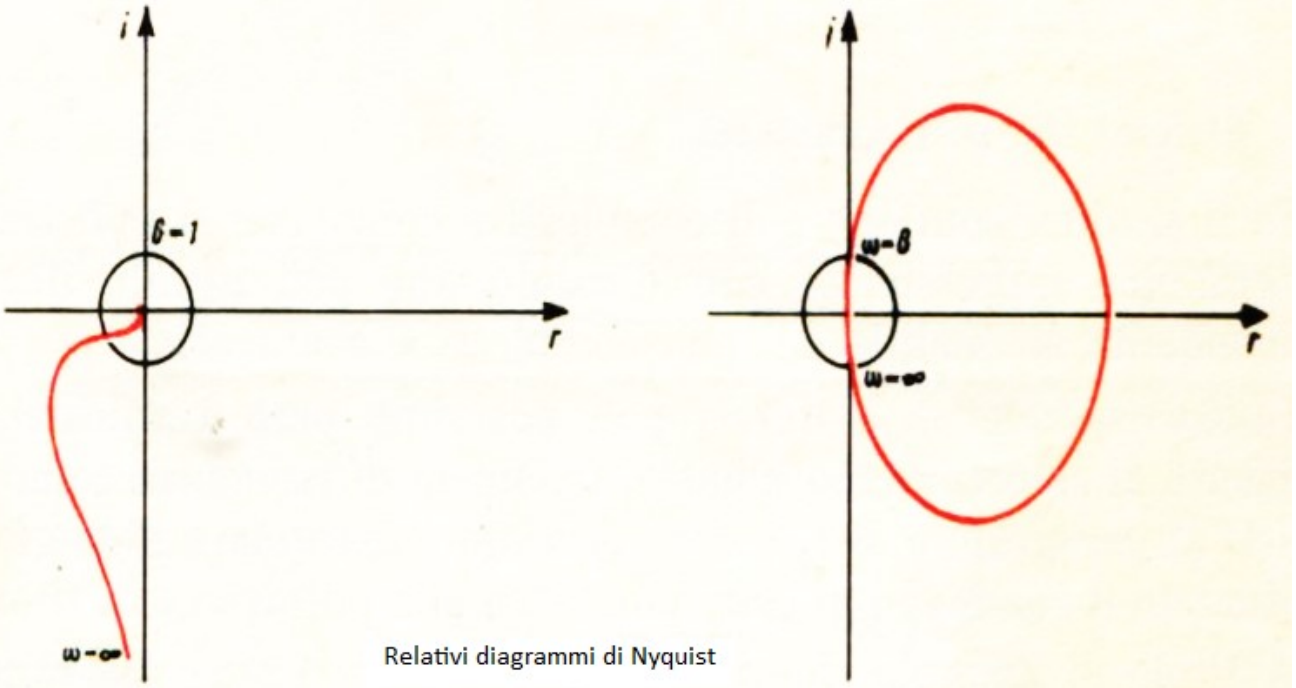
Relativi diagrammi di Nyquist



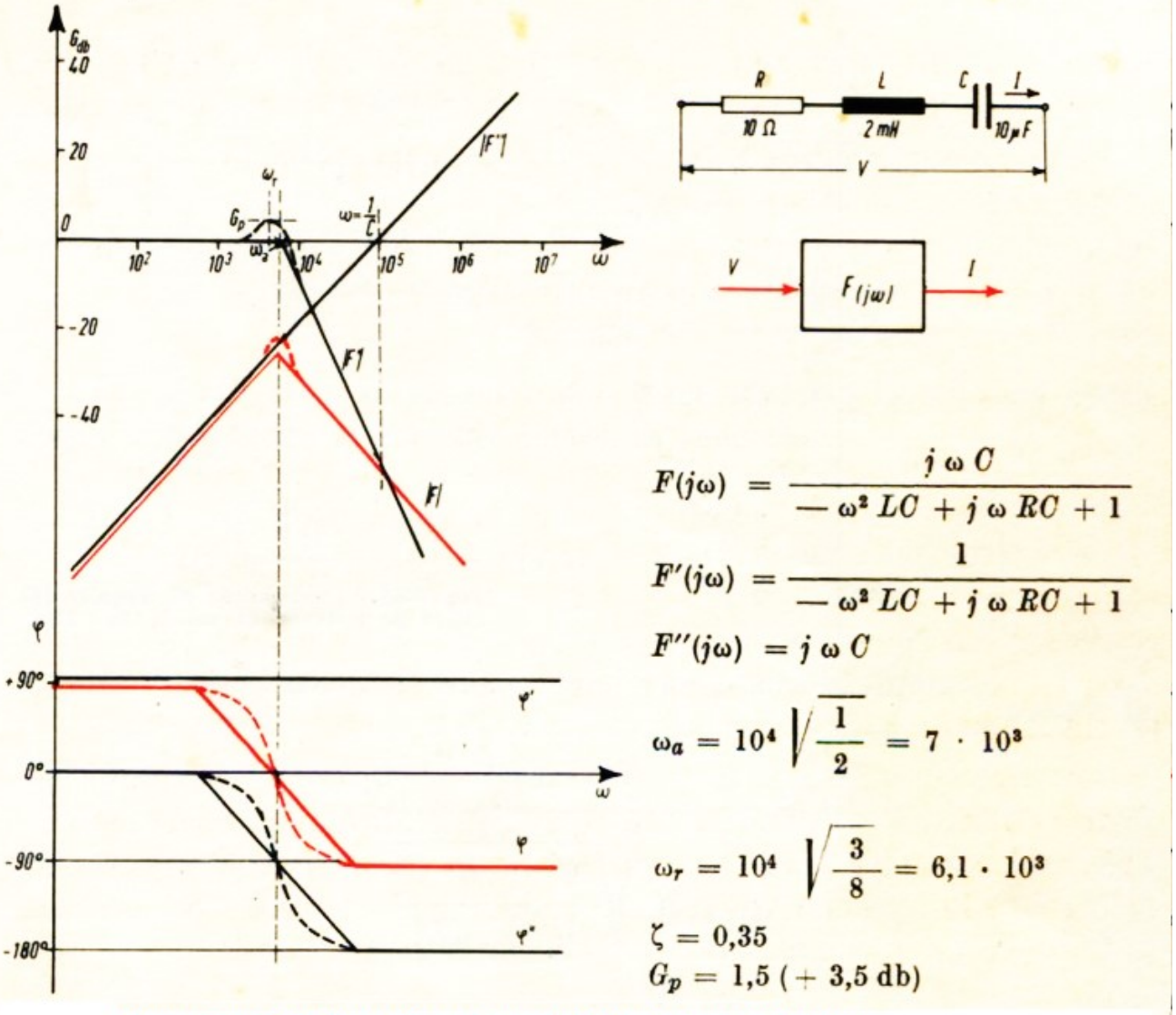
$$F(j\omega) = G_{st}\,\frac{1 - j\,\omega/z}{j\omega\left(1 - j\,\frac{\omega}{p}\right)}$$

$$F(j\omega) = G_{st}\,\frac{j\omega}{\left(1 - j\,\frac{\omega}{p_1}\right)\left(1 - j\,\frac{\omega}{p_2}\right)}$$

Diagrammi di Bode per sistemi con una integrazione e una derivazione



Relativi diagrammi di Nyquist



$$F(j\omega) = \frac{j\,\omega\,C}{-\,\omega^2\,LC + j\,\omega\,RC + 1}$$

$$F'(j\omega) = \frac{1}{-\,\omega^2\,LC + j\,\omega\,RC + 1}$$

$$F''(j\omega) = j\,\omega\,C$$

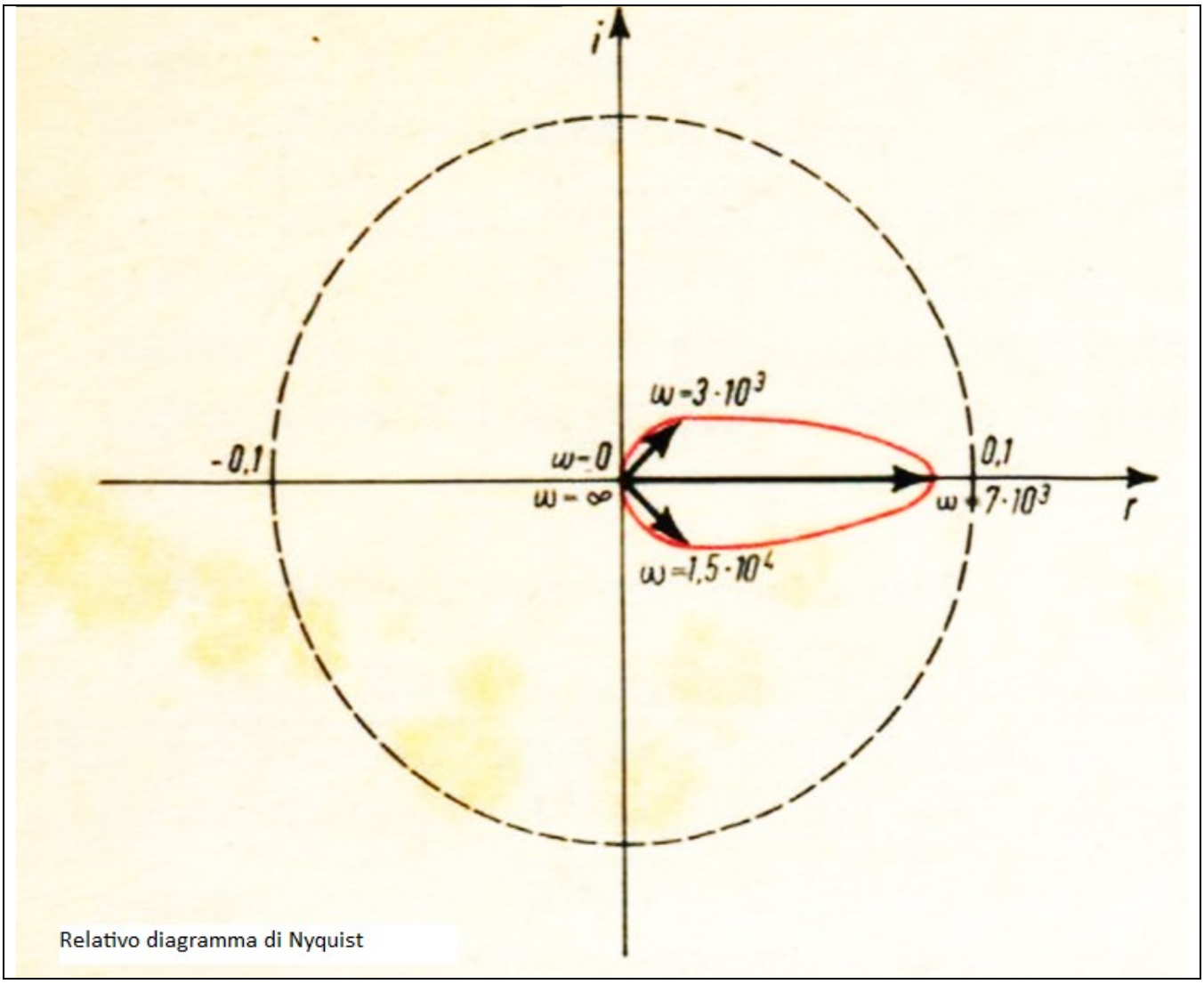
$$\omega_a = 10^4 \sqrt{\frac{1}{2}} = 7 \cdot 10^3$$

$$\omega_r = 10^4 \sqrt{\frac{3}{8}} = 6{,}1 \cdot 10^3$$

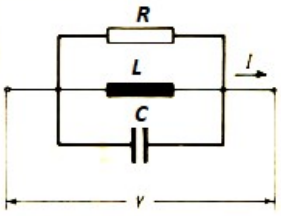
$$\zeta = 0{,}35$$

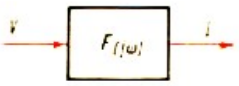
$$G_p = 1{,}5\ (+\ 3{,}5\ \text{db})$$

Diagrammi di Bode di un sistema del secondo ordine con una derivazione



Relativo diagramma di Nyquist

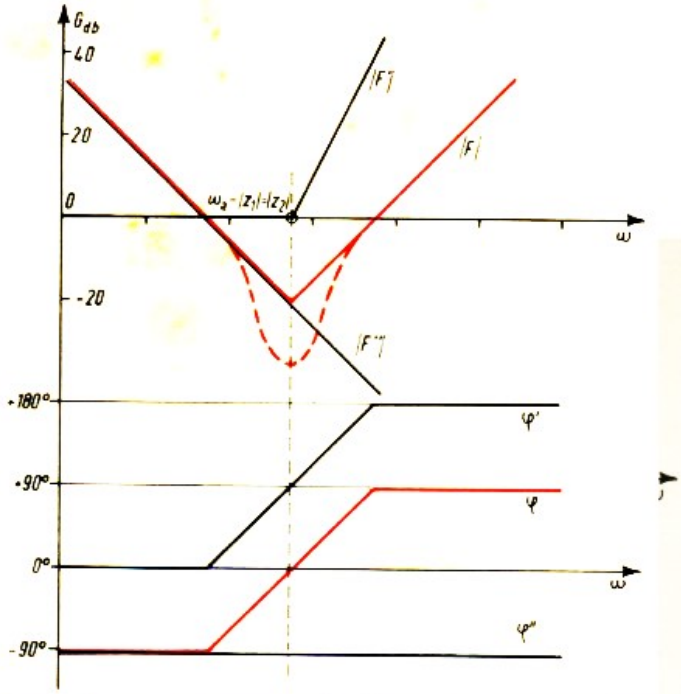




$$F(j\omega) = \frac{1}{R} \cdot \frac{-\omega^2 \tau_1 \tau_2 + j\omega\tau_1 + 1}{j\omega\tau_1}$$

$$F'(j\omega) = -\omega^2 \tau_2 \tau_1 + j\omega\tau_1 + 1$$

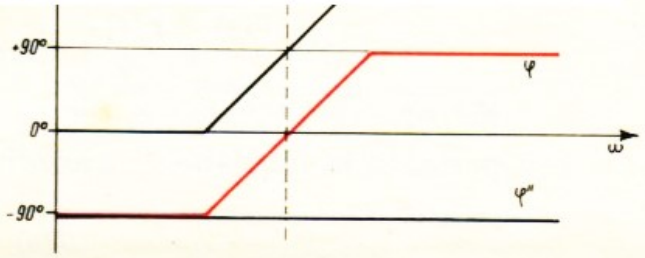
$$F''(j\omega) = \frac{1}{R} \cdot \frac{1}{j\omega\tau_1}$$

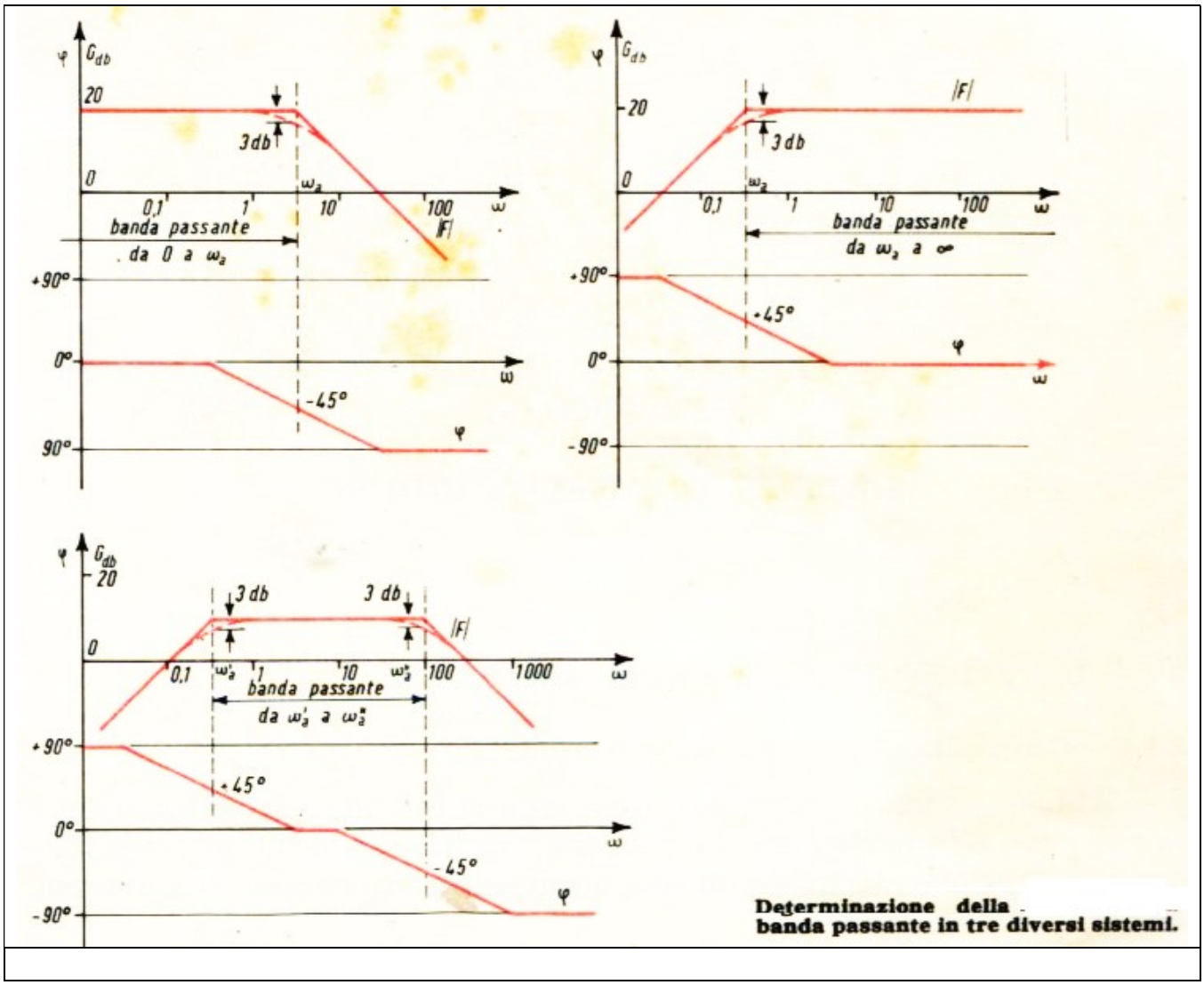


$$F(j\omega) = \frac{1}{R} \cdot \frac{-\omega^2 \tau_1 \tau_2 + j\omega\tau_1 + 1}{j\omega\tau_1}$$

$$F'(j\omega) = -\omega^2 \tau_2 \tau_1 + j\omega\tau_1 + 1$$

$$F''(j\omega) = \frac{1}{R} \cdot \frac{1}{j\omega\tau_1}$$





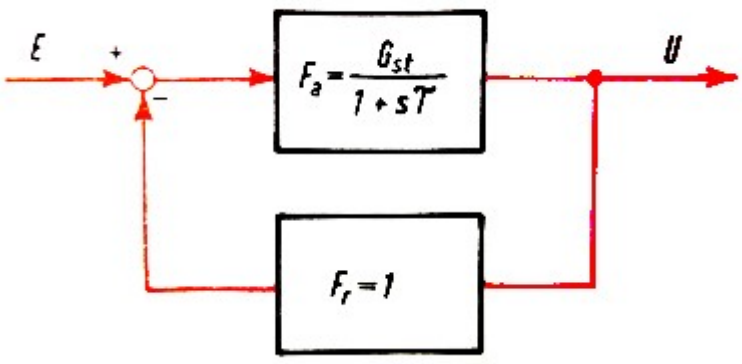
Determinazione della banda passante in tre diversi sistemi.

SISTEMI AD ANELLO CHIUSO

Dato un blocco del primo ordine con f.d.t.:

$$F_a(s) = \frac{G_{st}}{1 + s\tau}$$

Retroazionato con un blocco unitario $F_r(s)=1$



La f.d.t. del sistema retroazionato e':

$$G(s) = \frac{F_a(s)}{1 + F_a(s) \cdot F_r(s)}$$

Sostituendo Fa e Fr:

$$G(s) = \frac{\frac{G_{st}}{1 + s\tau}}{1 + \frac{G_{st}}{1 + s\tau} \cdot 1} = \frac{\frac{G_{st}}{1 + G_{st}}}{1 + s \cdot \frac{\tau}{1 + sG_{st}}}$$

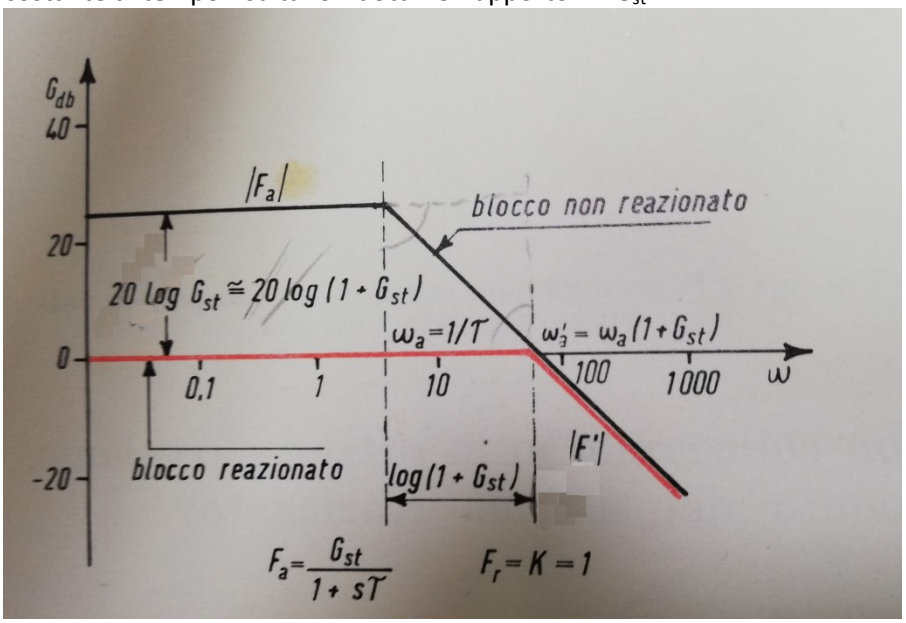
Cioe', la f.d.t. del sistema retroazionato e' dello stesso tipo del blocco non retroazionato, con un nuovo guadagno statico:

$$\frac{G_{st}}{1+G_{st}}$$

Ed una nuova costante di tempo:

$$\frac{\tau}{1+sG_{st}}$$

Cioe' guadagno e costante di tempo risultano ridotti nel rapporto: $1+G_{st}$



IL CRITERIO DI STABILITA' DI ROUTH-HURWITZ

• Si ponga l'equazione caratteristica in forma polinomiale:

$$D(s) = a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0 = 0$$

• Condizione necessaria affinche le radici dell'equazione caratteristica abbiano tutte parte reale negativa e' che tutti i coefficienti siano positivi: a_0>, a_1>0,…, a_{n-1}, a_n>0

• Criterio di Routh: ad ogni variazione di segno che presentano i termini della prima colonna della tabella di Routh corrisponde una radice a parte reale positiva, ad ogni permanenza una radice a parte reale negativa

• La Tabella di Routh si costruisce nel modo seguente:

$$\mathbf{R}(p) = \begin{bmatrix} a_n & a_{n-2} & a_{n-4} & a_{n-6} & \cdots \\ a_{n-1} & a_{n-3} & a_{n-5} & \cdots & \\ b_{n-1} & b_{n-2} & \cdots & & \\ c_{n-2} & c_{n-3} & & & \end{bmatrix}$$

Vedi per un esempio:https://www.andreaminini.org/sistemi/stabilita/criterio-di-routh

Esempio - Verificare la stabilità del sistema la cui equazione caratteristica è data da:

$s^4 + 2s^3+6s^2+4s+1 = 0$

```
#Codice Python
import sympy
def routh(p):
    """ Construct the Routh-Hurwitz array given a polynomial in s
    Input: p - a sympy.Poly object
    Output: The Routh-Hurwitz array as a sympy.Matrix object
    """
    coefficients = p#.all_coeffs()
    N = len(coefficients)
    M = sympy.zeros(N, (N+1)//2 + 1)

    r1 = coefficients[0::2]
    r2 = coefficients[1::2]
```

```
    M[0, :len(r1)] = [r1]
    M[1, :len(r2)] = [r2]
    for i in range(2, N):
        for j in range(N//2):
            S = M[[i-2, i-1], [0, j+1]]
            M[i, j] = sympy.simplify(-S.det()/M[i-1,0])
    return M[:, :-1]

print  routh([1,2,6,4,1])
>>>
[ 1.00   6.00   1.00
  2.00   4.00   0.00
  4.00   1.00   0.00
  3.50   0.00   0.00
  1.00   0.00   0.00   ]
```

Poiché tutti i coefficienti nella prima colonna sono dello stesso segno, cioè positivo, l'equazione data non ha radici con parti reali positive; pertanto, il sistema e' stabile.

Esempio

$s^3 - 4s^2 + s + 6 = 0$

La corrispondente tabella di Routh e':

```
[  1.00    1.00
  -4.00    6.00
   2.50    0.00
   6.00   -0.00   ]
```

La prima colonna della tabella presenta due variazioni di segno: si hanno pertanto due radici a parte reale positiva (le radici dell'equazione sono -1, 2,3).

APPENDICE

Libreria Control -Riferimento

La Python Control Systems Library `control`fornisce funzioni comuni per l'analisi e la progettazione di sistemi di controllo del feedback.

Creazione del sistema

`ss`(LA, SI, DO, RE[, dt])	Creare un sistema spaziale degli stati.
`tf`(num, den[, dt])	Creare un sistema di funzioni di trasferimento.
`frd`(d, w)	Costruire un modello di dati di risposta in frequenza
`rss`([stati, uscite, ingressi])	Crea un oggetto nello spazio degli stati casuale *continuo stabile.*
`drss`([stati, uscite, ingressi])	Creare un oggetto stabile dello spazio degli stati casuali *discreto* .

Interconnessioni di sistema

`append`(sys1, sys2, …, sysn)	Raggruppa i modelli aggiungendo i loro input e output
`connect`(sys, Q, inputv, outputv)	Interconnessione basata su indici di un sistema LTI.

`feedback`(sys1[, sys2, segno])	Interconnessione di feedback tra due sistemi I/O.
`negate`(sistema)	Restituisce il negativo di un sistema.
`parallel`(sys1, *sysn)	Restituisce la connessione parallela sys1 + sys2 (+ .
`series`(sys1, *sysn)	Restituisce la connessione in serie (sysn * .

Vedere anche il modulo Sistemi di input/output , che può essere utilizzato per creare e interconnettere sistemi di input/output non lineari.

Grafico nel dominio della frequenza

`bode_plot`(syslist[, omega, trama, …])	Bode plot per un sistema
`nyquist_plot`(syslist[, omega, trama, …])	Trama di Nyquist per un sistema
`gangof4_plot`(P, C[, omega])	Tracciare le funzioni di trasferimento "Gang of 4" per un sistema
`nichols_plot`(sys_list[, omega, griglia])	Nichols trama per un sistema
`nichols_grid`([cl_mags, cl_phases, line_style])	Griglia del grafico di Nichols

Nota: per i comandi di stampa che creano più assi sullo stesso grafico, i singoli assi possono essere recuperati utilizzando l'etichetta degli assi (recuperata utilizzando il metodo *get_label per l'oggetto assi matplotliib).* Attualmente sono definite le seguenti etichette:

- Grafici di Bode: *control-bode-magnitudine* , *control-bode-phase*
- Banda di 4 trame: *control-gangof4-s* , *control-gangof4-cs* , *control-gangof4-ps* , *control-gangof4-t*

Simulazione nel dominio del tempo

`forced_response`(sys[, T, U, X0, trasposizione, …])	Simula l'output di un sistema lineare.
`impulse_response`(sys[, T, X0, input, …])	Risposta all'impulso di un sistema lineare
`initial_response`(sys[, T, X0, input, …])	Risposta alla condizione iniziale di un sistema lineare
`input_output_response`(sis, T[, U, X0, …])	Calcola la risposta di output di un sistema a un dato input.
`step_response`(sys[, T, X0, ingresso, uscita, …])	Risposta al gradino di un sistema lineare
`phase_plot`(odefun[, X, Y, scala, X0, T, …])	Diagramma di fase per sistemi dinamici 2D

Algebra dei diagrammi a blocchi

`series`(sys1, *sysn)	Restituisce la connessione in serie (sysn * .
`parallel`(sys1, *sysn)	Restituisce la connessione parallela sys1 + sys2 (+ .
`feedback`(sys1[, sys2, segno])	Interconnessione di feedback tra due sistemi I/O.
`negate`(sistema)	Restituisce il negativo di un sistema.

Analisi del sistema di controllo

`dcgain`(sistema)	Restituisce il guadagno a frequenza zero (o CC) del sistema specificato
`evalfr`(sistema, x)	Valutare la funzione di trasferimento di un sistema LTI per un singolo numero complesso x.
`freqresp`(sistema, omega)	Risposta in frequenza di un sistema LTI a più frequenze angolari.
`margin`(dati di sistema)	Calcola guadagno e margini di fase e frequenze di crossover associate
`stability_margins`(sysdata[, returnall, epsw])	Calcola i margini di stabilità e le frequenze di crossover associate.
`phase_crossover_frequencies`(sistema)	Calcola frequenze e guadagni alle intersezioni con l'asse reale nel grafico di Nyquist.
`pole`(sistema)	Poli del sistema di calcolo.
`zero`(sistema)	Calcola zeri di sistema.
`pzmap`(sys[, trama, griglia, titolo])	Traccia una mappa polo/zero per un sistema lineare.
`root_locus`(sys[, kvect, xlim, ylim, …])	Trama del luogo della radice
`sisotool`(sys[, kvect, xlim_rlocus, …])	Collezione di trame in stile Sisotool ispirata al sisotool di MATLAB.

Calcoli matriciali

`care`(A, B, Q[, R, S, E, stabilizzante])	(X, L, G) = care(A, B, Q, R=None) risolve l'equazione di Riccati algebrica in tempo continuo
`dare`(A, B, Q, R[, S, E, stabilizzante])	(X, L, G) = osare(A, B, Q, R) risolve l'equazione di Riccati algebrica a tempo discreto
`lyap`(LA, Q[, C, E])	X = lyap(A, Q) risolve l'equazione di Lyapunov a tempo continuo
`dlyap`(LA, Q[, C, E])	dlyap(A,Q) risolve l'equazione di Lyapunov a tempo discreto
`ctrb`(A, B)	Matrice di controllabilità
`obsv`(CORRENTE ALTERNATA)	Matrice di osservabilità
`gram`(sistema, tipo)	Gramiano (controllabilità o osservabilità)

Sintesi del sistema di controllo

`acker`(A, B, poli)	Posizionamento dei poli con metodo Ackermann
`h2syn`(P, nmeas, ncon)	Sintesi di controllo H_2 per l'impianto P.
`hinfsyn`(P, nmeas, ncon)	H_{inf} sintesi di controllo per l'impianto P.
`lqr`(LA, B, Q, R[, N])	Design del regolatore quadratico lineare
`lqe`(A, G, C, QN, RN, [, N])	Progettazione di uno stimatore quadratico lineare (filtro di Kalman) per sistemi a tempo continuo.

`mixsyn`(g[, w1, w2, w3])	Sintesi H-infinito a sensibilità mista.
`place`(A, B, p)	Posizionare autovalori ad anello chiuso

Strumenti di semplificazione del modello

`minreal`(sys[, tol, verbose])	Elimina gli stati incontrollabili o non osservabili nei modelli dello spazio degli stati o annulla le coppie polo-zero nelle funzioni di trasferimento.
`balred`(sistema, ordini[, metodo, alfa])	Modello di ordine ridotto bilanciato di sys di un determinato ordine.
`hsvd`(sistema)	Calcola i valori singolari di Hankel.
`modred`(sistema, ELIM[, metodo])	Modella la riduzione di *sys* eliminando gli stati in *ELIM* utilizzando un determinato metodo.
`era`(AA, m, n, nono, no, r)	Calcola un modello ERA dell'ordine *r* basato sui dati di risposta all'impulso *YY* .
`markov`(Y, U[, m, trasposizione])	Calcolare i primi *m* parametri Markov [D CB CAB …] dall'ingresso *U* , uscita *Y* .

Supporto per sistemi non lineari

`find_eqpt`(sys, x0[, u0, y0, t, parametri, iu, …])	Trova il punto di equilibrio per un sistema di input/output.
`linearize`(sys, xeq[, ueq, t, parametri])	Linearizzare un sistema di input/output in un dato stato e input.
`input_output_response`(sis, T[, U, X0, …])	Calcola la risposta di output di un sistema a un dato input.
`ss2io`(*args, **kw)	Creare un sistema I/O da un sistema lineare nello spazio degli stati.
`tf2io`(*args, **kw)	Converti una funzione di trasferimento in un sistema I/O
`flatsys.point_to_point`(sys, x0, u0, xf, uf, Tf)	Calcolare la traiettoria tra una condizione iniziale e una finale.

Funzioni di utilità e conversioni

`augw`(g[, w1, w2, w3])	Impianto di potenziamento per problemi di sensibilità mista.
`canonical_form`(xsys[, modulo])	Converti un sistema in forma canonica
`damp`(sys[, doprint])	Calcola la frequenza naturale, il rapporto di smorzamento e i poli di un sistema
`db2mag`(db)	Converti un guadagno in decibel (dB) in una grandezza
`isctime`(sys[, strict])	Verificare se un sistema è un sistema a tempo continuo
`isdtime`(sys[, strict])	Verificare se un sistema è un sistema a tempo discreto
`issiso`(sys[, strict])	Verificare se un sistema è a ingresso singolo, uscita singola
`issys`(oggetto)	Restituisce True se un oggetto è un sistema, altrimenti False

`mag2db`(mag)	Converti una magnitudine in decibel (dB)
`observable_form`(xsys)	Converti un sistema in forma canonica osservabile
`pade`(T[, n, numdeg])	Creare un sistema lineare che approssima un ritardo.
`reachable_form`(xsys)	Converti un sistema in una forma canonica raggiungibile
`reset_defaults`()	Reimposta i valori di configurazione sui valori predefiniti (iniziali).
`sample_system`(sysc, Ts[, metodo, alfa, …])	Converti un sistema a tempo continuo in tempo discreto
`ss2tf`(sistema)	Trasforma un sistema nello spazio degli stati in una funzione di trasferimento.
`ssdata`(sistema)	Restituisce oggetti dati dello spazio di stato per un sistema
`tf2ss`(sistema)	Trasforma una funzione di trasferimento in un sistema nello spazio degli stati.
`tfdata`(sistema)	Restituisce oggetti dati della funzione di trasferimento per un sistema
`timebase`(sys[, strict])	Restituisce la base dei tempi per un sistema LTI
`timebaseEqual`(sistema1, sistema2)	Verificare se due sistemi hanno la stessa base dei tempi
`unwrap`(angolo[, punto])	Scartare un angolo di fase per dare una curva continua
`use_fbs_defaults`()	Utilizzare le impostazioni compatibili con i sistemi di feedback (FBS).
`use_matlab_defaults`()	Usa le impostazioni di configurazione compatibili con MATLAB.
`use_numpy_matrix`([segnala, avverte])	Attiva/disattiva l'uso della classe *matrice* Numpy per le operazioni nello spazio degli stati.

RIFERIMENTI:
Cerri …– Nuovo Corso di Sistemi Automatici, Hoepli ed.
http://www.controlsystemsacademy.com/0019/0019.html

http://www.rosatelli.edu.it/seconde-prove-ind-elettornica-elettrotecnica-automazione
https://online.scuola.zanichelli.it/provatecnici/sistemi-automatici