LTI Systems with Python

Prof. Seungchul Lee
iSystems (http://isystems.unist.ac.kr/)
UNIST

Table of Contents

1. Mathematical Models of LTI

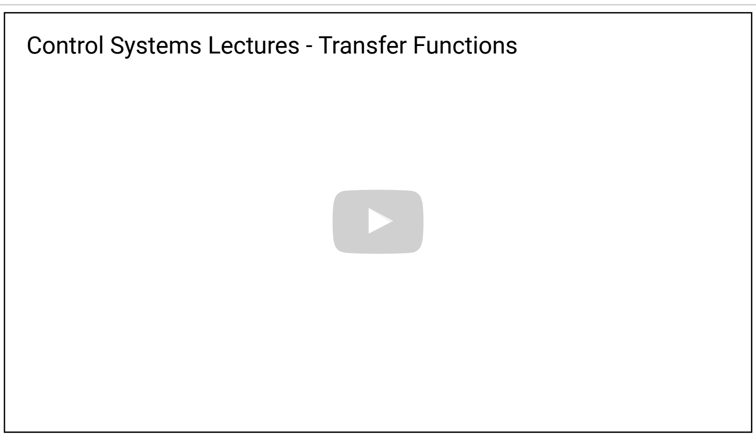
- from ebook *Linear Feedback Control Analysis and Design with MATLAB* (http://epubs.siam.org/doi/book/10.1137/1.9780898718621)

1.1. Transfer Function (TF)

- Brian Douglas youtube [Control Systems Lectures - Transfer Functions]
- Laplace Transform

In [1]:

```
%%html
<iframe src="https://www.youtube.com/embed/RJleGwXorUk"
width="560" height="315" frameborder="0" allowfullscreen></iframe>
```



$$G(s) = \frac{s+5}{s^4 + 2s^3 + 3s^2 + 4s + 5}$$

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from control import *
from scipy import *
from scipy import linalg as la
from scipy.ndimage.filters import convolve

%matplotlib inline
```

In [3]:

```
num = [1,5]
den = [1,2,3,4,5]

G = tf(num,den)
print (G)
```

```
            s + 5
-----------------------------
s^4 + 2 s^3 + 3 s^2 + 4 s + 5
```

$$G(s) = \frac{6(s+5)}{(s^2+3s+1)^2(s+6)(s^3+6s^2+5s+3)}$$

In [4]:

```
num = [1,5]
num = [x*6 for x in num]
den = np.convolve(np.convolve(np.convolve([1,3,1],[1,3,1]),[1,6]),[1,6,5,3])
den = den.tolist()
G = tf(num,den)
print G
```

```
                                  6 s + 30
---------------------------------------------------------------------------
s^8 + 18 s^7 + 124 s^6 + 417 s^5 + 740 s^4 + 729 s^3 + 437 s^2 + 141 s + 18
```

1.2. Transfer Function in zero-pole-gain model

$$G(s) = K\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

In [5]:

```
#zpk does not exist in python
```

1.3. State-space model

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

In [6]:

```
A = [[2.25,-5,   -1.25,-0.5],
     [2.25,-4.25,-1.25,-0.25],
     [0.25,-0.5, -1.25,-1],
     [1.25,-1.75,-0.25,-0.75]]

B = [[4,6],
     [2,4],
     [2,2],
     [0,2]]

C = [[0,0,0,1],
     [0,2,0,2]]

D = np.zeros((2,2))

G = ss(A,B,C,D)

print(G)
```

```
A = [[ 2.25 -5.   -1.25 -0.5 ]
 [ 2.25 -4.25 -1.25 -0.25]
 [ 0.25 -0.5  -1.25 -1.  ]
 [ 1.25 -1.75 -0.25 -0.75]]

B = [[4 6]
 [2 4]
 [2 2]
 [0 2]]

C = [[0 0 0 1]
 [0 2 0 2]]

D = [[ 0.  0.]
 [ 0.  0.]]
```

Characteristic polynomial of the system

In [7]:

```
print(G.A)

P = poly(G.A)
```

```
[[ 2.25 -5.   -1.25 -0.5 ]
 [ 2.25 -4.25 -1.25 -0.25]
 [ 0.25 -0.5  -1.25 -1.  ]
 [ 1.25 -1.75 -0.25 -0.75]]
```

In [8]:

```
print(P)
```

```
[ 1.    4.    6.25  5.25  2.25]
```

$$P(s) = s^4 + 4s^3 + 6.25s^2 + 5.25s + 2.25$$

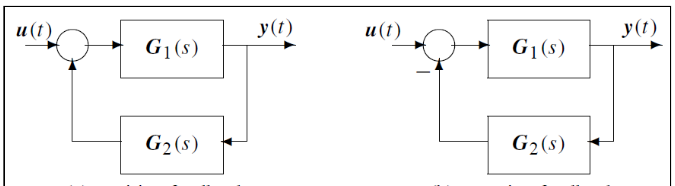
2. Interconnected Block Diagrams

series and parallel connections

In [9]:

```
#zpk does not exist in python
```

Feedback connection



(a) positive feedback (b) negative feedback

- positive feedback

$$G(s) = G_1(s)[I - G_2(s)G_1(s)]^{-1}$$

- negative feedback

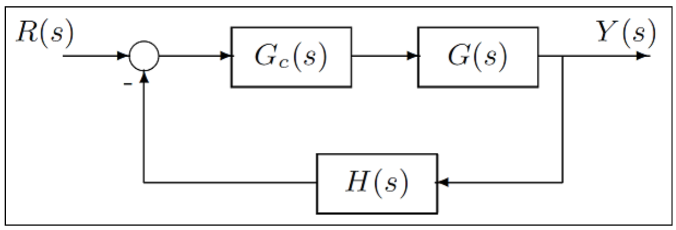
$$G(s) = G_1(s)[I + G_2(s)G_1(s)]^{-1}$$

In [10]:

```
G1 = tf(1,[1,2,1])
G2 = tf(1,[1,1])

# negative feedback
G3 = feedback(G1,G2,-1)
print(G3)

# positive feedback
G4 = feedback(G1,G2,+1)
print(G4)
```

```
        s + 1
---------------------
s^3 + 3 s^2 + 3 s + 2


      s + 1
-----------------
s^3 + 3 s^2 + 3 s
```

More complicated connections



$$G_{cl}(s) = \frac{G(s)G_c(S)}{1 + H(s)G(s)G_c(S)}$$

In [35]:

```
G = tf([1,7,24,24],[1,10,35,50,24])
Gc = tf([10,5],[1,0])
H = tf(1,[0.01,1])

Gcl = feedback(Gc*G,H,-1)
print(Gcl)

G = ss(tf([1,7,24,24],[1,10,35,50,24]))
G_a = feedback(Gc*G,H)
print(G_a)
```

```
     0.1 s^5 + 10.75 s^4 + 77.75 s^3 + 278.6 s^2 + 361.2 s + 120
--------------------------------------------------------------------
0.01 s^6 + 1.1 s^5 + 20.35 s^4 + 110.5 s^3 + 325.2 s^2 + 384 s + 120


     0.1 s^5 + 10.75 s^4 + 77.75 s^3 + 278.6 s^2 + 361.2 s + 120
--------------------------------------------------------------------
0.01 s^6 + 1.1 s^5 + 20.35 s^4 + 110.5 s^3 + 325.2 s^2 + 384 s + 120
```

3. Model Conversion

3.1. from state space to transfer function

In [12]:

```
A = [[0,1, 0,0],
     [0,0,-1,0],
     [0,0, 0,1],
     [0,0, 5,0]]
B = np.array([[0,1,0,-2]]).T
C = [1,0,0,0]
D = 0

Gss = ss(A,B,C,D)
print(Gss)
Gtf = tf(Gss)
print(Gtf)
```

```
A = [[ 0  1  0  0]
 [ 0  0 -1  0]
 [ 0  0  0  1]
 [ 0  0  5  0]]

B = [[ 0]
 [ 1]
 [ 0]
 [-2]]

C = [[1 0 0 0]]

D = [[0]]


  s^2 - 3
-----------
s^4 - 5 s^2
```

3.2. from zpk to tf

In [13]:

```
#zpk does not exist in python
```

3.3. from ss to zpk

In [14]:

```
#zpk does not exist in python
```

3.4. from tf to zpk

In [15]:

```
#zpk does not exist in python
```

3.5. Similarity Transformation of State Space Model

ss2ss

$$\begin{aligned}
\dot{x}(t) &= Ax(t) + Bu(t) \\
y(t) &= Cx(t) + Du(t) \\
z &= Tx \\
\dot{z}(t) &= TAT^{-1}z(t) + TBu(t) \\
y(t) &= CT^{-1}z(t) + Du(t)
\end{aligned}$$

In [16]:

```
#ss2ss does not exist in python
```

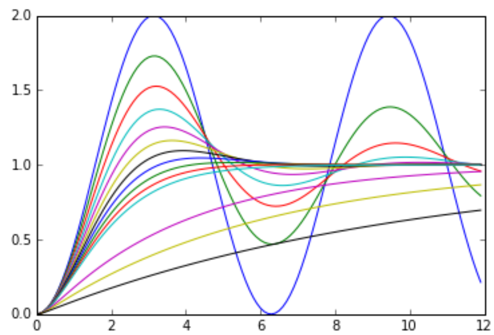
4. Time Response of LTI

4.1. Step response

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

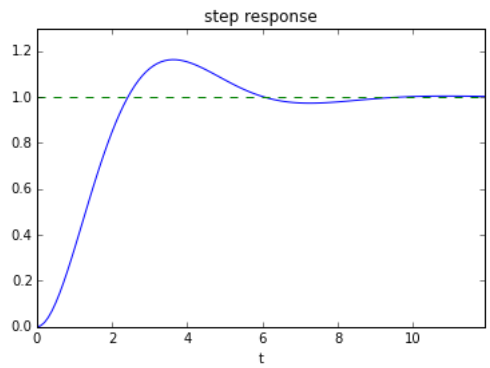
In [17]:

```
eps = np.finfo(float).eps

wn = 1
t = np.arange(0.0,12.0,0.1)
zet = np.arange(0.0,1.0,0.1).tolist() + [1+eps,2,3,5]

for i in range(0, len(zet)):
    G = tf(wn**2,[1,2*zet[i]*wn,wn**2])
    [y, tout] = step(G,t)
    plt.plot(tout,y)

plt.show()
```



In [18]:

```
z = 0.5
wn = 1

G = tf(wn**2,[1,2*z*wn,wn**2])

[y,tout] = step(G,np.arange(0.0,12.0,0.1))

plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape),'--')
plt.axis('tight')
plt.ylim([0,1.3])
plt.title('step response')
plt.xlabel('t')

plt.show()

print(G)
```

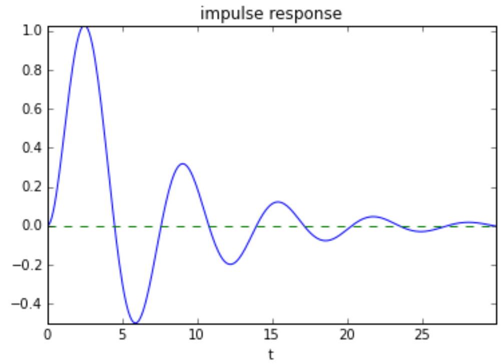



```
     1
-----------
s^2 + s + 1
```

4.2. Impluse response

In [19]:

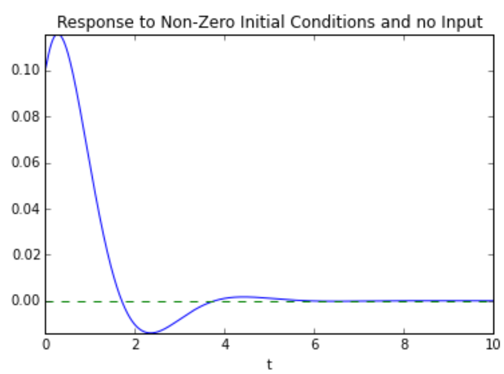
```
G = tf([10,20],[10,23,26,23,10])

[y,tout] = impulse(G,np.arange(0.0,30.0,0.1))

plt.plot(tout,y)
plt.plot(tout,np.zeros(tout.shape),'--')
plt.axis('tight')
plt.title('impulse response')
plt.xlabel('t')

plt.show()
```



4.3. General response using `lsim`

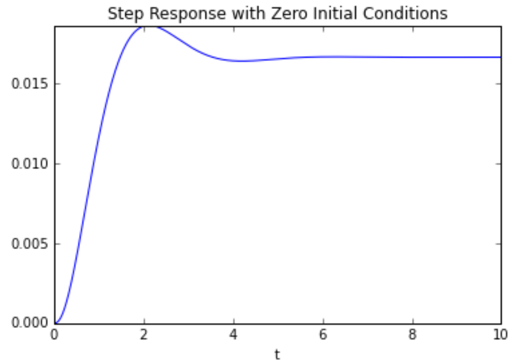
In [20]:

```
A = [[-20,-40,-60],
     [1,    0,  0],
     [0,    1,  0]]
B = [[1],[0],[0]]
C = [0,0,1]
D = 0

sys = ss(A,B,C,D)     # construct a system model

t = np.arange(0.0,10.0,0.01) # simulation time = 10 seconds
u = np.zeros(t.shape)          # no input
X0 = [0.1,0.1,0.1]           # initial conditions of the three states

[y,tout,non] = lsim(sys,u,t,X0)    # simulate and plot the response (the output)

plt.plot(tout,y)
plt.plot(tout,np.zeros(tout.shape),'--')
plt.axis('tight')
plt.xlabel('t')
plt.title('Response to Non-Zero Initial Conditions and no Input')

plt.show()
```



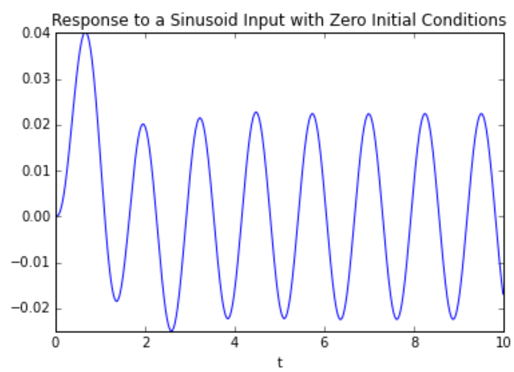
In [21]:

```
t = np.arange(0.0,10.0,0.01) # simulation time = 10 seconds
u = np.ones(t.shape)           # u = 1, a step input

[y,tout,X] = lsim(sys,u,t) # simulate

plt.plot(tout,y)
plt.axis('tight')
plt.xlabel('t')
plt.title('Step Response with Zero Initial Conditions')

plt.show()
```



In [22]:

```
t = np.arange(0.0,10.0,0.01) # simulation time = 10 seconds
u = 10*sin(5*t+1)            # input as a function of time

[y,tout,non] = lsim(sys,u,t) # simulate

plt.plot(tout,y)
plt.axis('tight')
plt.xlabel('t')
plt.title('Response to a Sinusoid Input with Zero Initial Conditions')

plt.show()
```



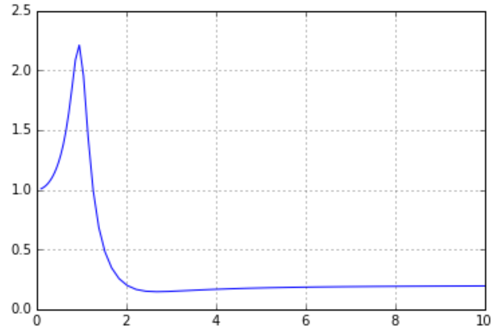
5. Frequency

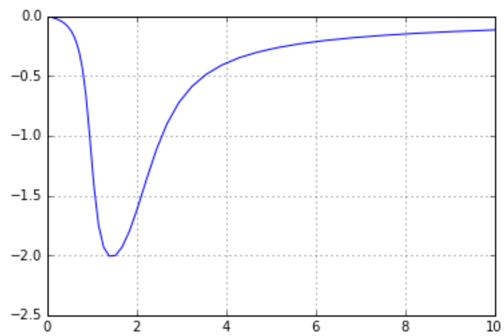
- from umich control (http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemAnalysis)

In [23]:

```
# freqs: Laplace-transform (s-domain) frequency response

a = [1,0.4,1]      # Numerator coefficients
b = [0.2,0.3,1]    # Denominator coefficients
G = tf(b,a)

w = logspace(-1,1)  # Frequency vector
[mag,phase,omega] = matlab.freqresp(G,w)
```

In [24]:
```
plt.plot(omega,mag[0][0])
plt.grid(True)

plt.show()

plt.plot(omega,phase[0][0])
plt.grid(True)

plt.show()
```



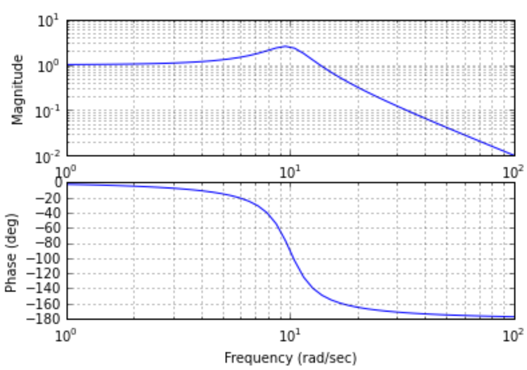


5.1. Bode plot

- Good reference from Mathworks
    - understanding Bode plots (https://www.youtube.com/playlist?list=PLn8PRpmsu08poVEWzpqKXpj_c7aSwVDdm)
    - using Bode plots (https://www.youtube.com/playlist?list=PLn8PRpmsu08qbUh-mLHxDYAW8ClPdE1H6)
- A serise of Bode plot lectures by Brian Douglas (https://www.youtube.com/watch?v=_eh1conN6YM&index=9&list=PLUMWjy5jgHK1NC52DXXrriwihVrYZKqjk)

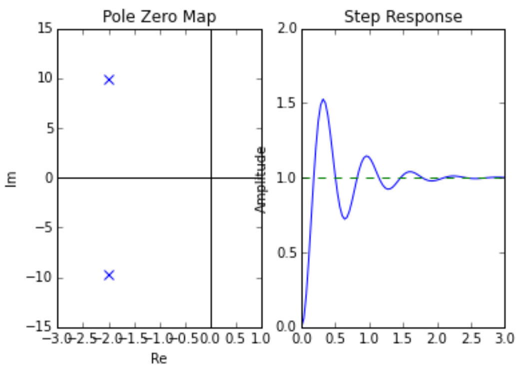
$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

In [25]:
```
w_n = 10
zeta = 0.2

G1 = tf(w_n**2,[1,2*zeta*w_n,w_n**2])

bode(G1)
plt.show()
```



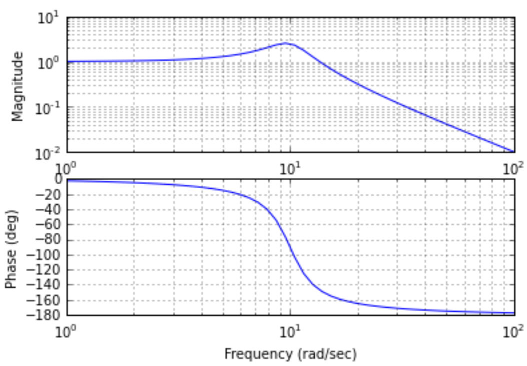
In [26]:

```
plt.subplot(1,2,1)
pzmap.pzmap(G1)
plt.axis([-3,1,-15,15])
plt.subplot(1,2,2)

[y,tout] = step(G1)

plt.plot(tout,y)
plt.plot(tout, np.ones(tout.shape),'--')
plt.axis([0,3,0,2])
plt.title('Step Response')
plt.ylabel('Amplitude')

plt.show()
```



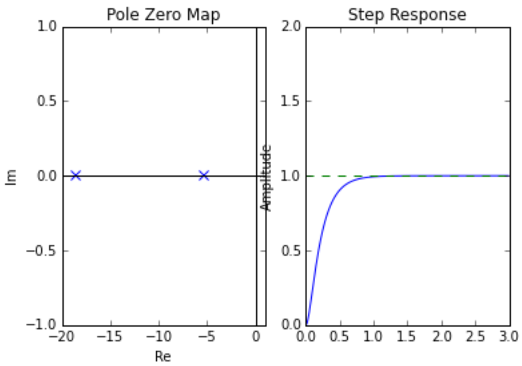
In [27]:

```
bode(G1)
plt.show()
```



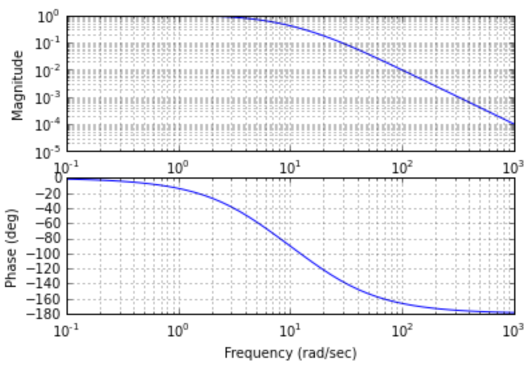
In [28]:

```
w_n = 10
zeta = 1.2

G1 = tf(w_n**2,[1,2*zeta*w_n,w_n**2])

plt.subplot(1,2,1)
pzmap.pzmap(G1)
plt.axis([-20, 1, -1, 1])
plt.subplot(1, 2, 2)

[y, tout] = step(G1,np.arange(0.0,3.0,0.01))

plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape),'--')
plt.axis([0,3,0,2])
plt.title('Step Response')
plt.ylabel('Amplitude')

plt.show()
```



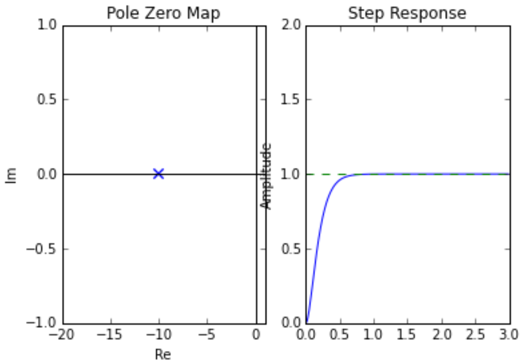
In [29]:

```
bode(G1)
plt.show()
```



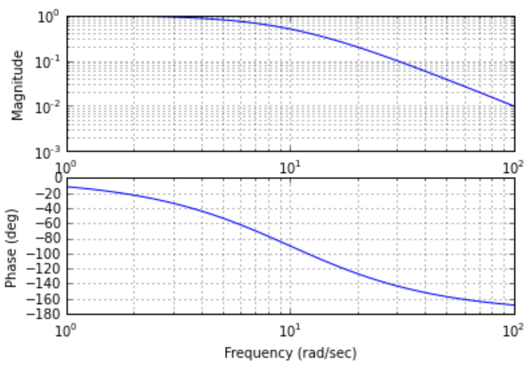
In [30]:

```
w_n = 10
zeta = 1

G1 = tf(w_n**2,[1,2*zeta*w_n,w_n**2])

plt.subplot(1,2,1)
pzmap.pzmap(G1)
plt.axis([-20,1,-1,1])
plt.subplot(1,2,2)

[y, tout] = step(G1,np.arange(0.0,3.0,0.01))

plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape),'--')
plt.axis([0,3,0,2])
plt.title('Step Response')
plt.ylabel('Amplitude')

plt.show()
```



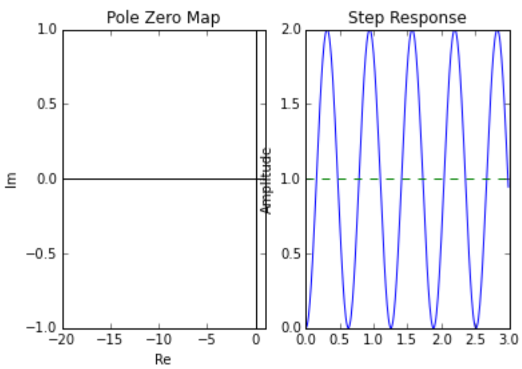
In [31]:

```
bode(G1)
plt.show()
```



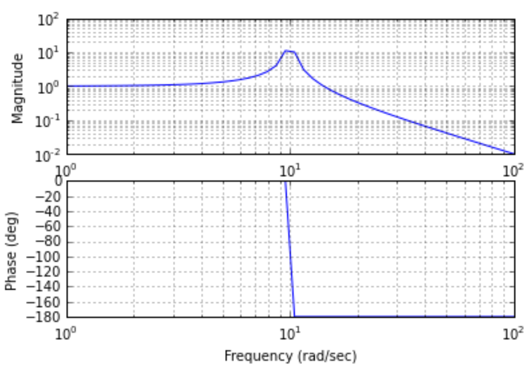
In [32]:

```
w_n = 10
zeta = 0

G1 = tf(w_n**2,[1,2*zeta*w_n,w_n**2])

plt.subplot(1,2,1)
pzmap.pzmap(G1)
plt.axis([-20,1,-1,1])
plt.subplot(1,2,2)

[y,tout] = step(G1,np.arange(0.0,3.0,0.01))

plt.plot(tout,y)
plt.plot(tout,np.ones(tout.shape),'--')
plt.axis([0,3,0,2])
plt.title('Step Response')
plt.ylabel('Amplitude')

plt.show()
```



In [33]:

```
bode(G1)
plt.show()
```



In [34]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js')
```