HW7 - solution

1. state and output feedback

Purpose: apply analytical and computational techniques to observe and control the state of a *nonlinear* system.

Consider the following model of a robot arm from Problem 2 of Homework 6 (hw6p2) that consists of a rigid rod of length ℓ attached to a point mass m at one end and a rotational joint at the other end,

$$m\ell^2\ddot{\theta} = mg\ell\sin\theta - \gamma\dot{\theta} + \tau,\ y = \ell\sin\theta,$$

where γ is a coefficient of friction for the rotational joint and τ is a torque applied by a motor attached to the rotational joint; use parameter values $m = 1$ kg, $\ell = 1$ m, $g = 9.81$ m sec^{-2}, $\gamma = 1$ in this problem.

a. Implement a simulation where you apply the observer from (hw6p2g.) and controller from (hw6p2c.) to the ***nonlinear*** system (nonlinear dynamics and nonlinear output equation); provide overlaid plots of states versus time for the nonlinear system and linear controller to verify that (i) the state of the observer converges to the state of the nonlinear system and (ii) the state of the nonlinear system converges to the equilibrium.

Note: the nonlinear system and linear controller should be initialized at non-equal non-equilibrium initial conditions.

Hint: the nonlinear system and linear controller should not be initialized too far from the equilibrium.

Bonus: Demonstrate that the controller in (a.) fails to stabilize the nonlinear system to the equilibrium by initializing the nonlinear system and/or linear observer sufficiently far from the equilibrium.

Takeaway: we can apply linear systems tools to synthesize a controller that stabilizes a nonlinear system ***near an equilibrium***.

In [ ]:
```
# import useful python packages
!pip install control
import numpy as np
import pylab as plt
from control import place as place
```

```
Requirement already satisfied: control in /usr/local/lib/python3.7/dist-packages (0.9.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from control) (1.19.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from control) (1.4.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from control) (3.2.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->control) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->control) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->control) (3.0.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->control) (1.3.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib->control) (1.15.0)
```

In [ ]:

```python
# build ODE solver
def numerical_simulation(f,t,x,t0=0.,dt=1e-4,ut=None,ux=None,utx=None,returr
  """
  simulate x' = f(x,u)

  input:
    f : R x X x U --> X - vector field
      X - state space (must be vector space)
      U - control input set
    t - scalar - final simulation time
    x - initial condition; element of X

    (optional:)
    t0 - scalar - initial simulation time
    dt - scalar - stepsize parameter
    return_u - bool - whether to return u_

    (only one of:)
    ut : R --> U
    ux : X --> U
    utx : R x X --> U

  output:
    t_ - N array - time trajectory
    x_ - N x X array - state trajectory
    (if return_u:)
    u_ - N x U array - state trajectory
  """
  t_,x_,u_ = [t0],[x],[]

  inputs = sum([1 if u is not None else 0 for u in [ut,ux,utx]])
  assert inputs <= 1, "more than one of ut,ux,utx defined"

  if inputs == 0:
    assert not return_u, "no input supplied"
  else:
    if ut is not None:
```

```
      u = lambda t,x : ut(t)
    elif ux is not None:
      u = lambda t,x : ux(x)
    elif utx is not None:
      u = lambda t,x : utx(t,x)

  while t_[-1]+dt < t:
    if inputs == 0:
      _t,_x = t_[-1],x_[-1]
      dx = f(t_[-1],x_[-1]) * dt
    else:
      _t,_x,_u = t_[-1],x_[-1],u(t_[-1],x_[-1])
      dx = f(_t,_x,_u) * dt
      u_.append( _u )

    x_.append( _x + dx )
    t_.append( _t + dt )

  if return_u:
    return np.asarray(t_),np.asarray(x_),np.asarray(u_)
  else:
    return np.asarray(t_),np.asarray(x_)
```

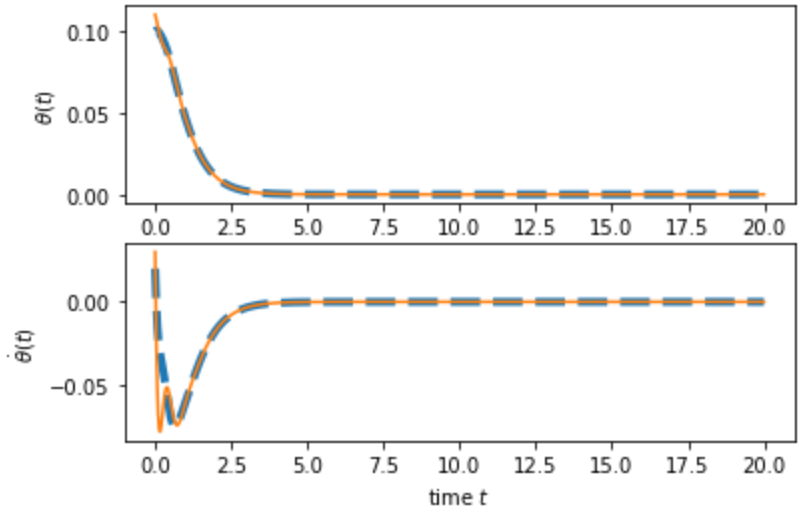
In [ ]:

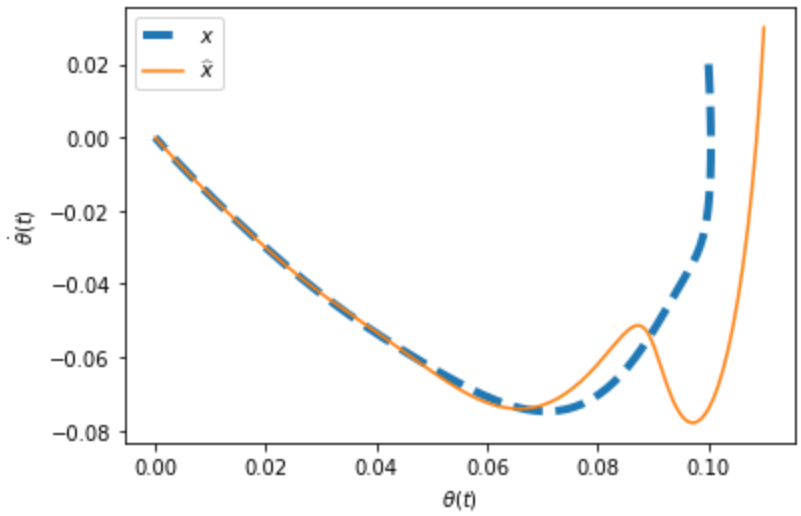
```
# define parameters
m = 1 # kg
ell = 1 # m
g = 9.81 # m/sec/sec
gamma = 1

A = np.array([[0,1],[g/ell, -gamma/(m*ell**2)]])
B = np.array([[0],[1/(m*ell**2)]])
C = np.array([[ell, 0]])
D = np.array([[0]])
roots = [-5+8.66*1j, -5-8.66*1j] #Enter value

k1 = m*ell*g+4*m*ell**2
k2 = 4*m*ell**2 - gamma
K = np.asarray([k1,k2])

L = -place(A.T,C.T,roots).T


def f_NL(t,x,u): # nonlinear system state dynamics
  theta = x[0]
  dtheta = x[1]
  ddtheta = g/ell*np.sin(theta) - gamma/(m*(ell**2))*dtheta + u/(m*(ell**2))
  dx = np.array([dtheta, ddtheta]).squeeze()
  return dx

def h_NL(t,x,u): # nonlinear system output dynamics
  theta = x[0]
  dtheta = x[1]
  y = ell * np.sin(theta)
  return np.array([[y]])

def f_cl(t,z,u): # closed-loop system
```

```
  # extract controlled system and observer system state from full system sta
  x,x_hat = z[:2],z[2:]
  # compute control input using observer state
  u = -np.dot(x_hat,K.T)

  # output from controlled system and observer system
  y = h_NL(t,x,u)
  y_hat = ell * x_hat[0] # linearized

  # dynamics of controlled system and observer system
  dx = f_NL(t,x,u)
  dx_hat = np.dot(x_hat,A.T) + np.dot(u,B.T) - np.dot(y-y_hat,L.T)
  dx_hat = np.array(dx_hat)
  dx_hat = dx_hat[0]
  # print( "dx.shape, dx_hat.shape", dx.shape, dx_hat.shape ) # debug

  # combine dynamics of controlled system and observer system
  dz = np.hstack((dx,dx_hat))
  return dz

# non-zero initial state
theta0 = 0.1
dtheta0 = 0.02
theta_hat0 = 0.11
dtheta_hat0 = 0.03
x0 = np.array([theta0, dtheta0, theta_hat0, dtheta_hat0])
T = 20
# zero input
def u(t):
  ut = 0
  return ut

# numerical simulation returns:
# t_ - size N array
# x_ - size N x 4 array (since the state is 2-dimensional)
t_,z_ = numerical_simulation(f_cl,T,x0,ut=u,dt=dt)
z_ = np.squeeze(z_)
print(z_.shape)
# the first column contains theta(t), the second column contains dtheta(t)
# (I append an underscore "_" to remind myself these are time series)
x_ = z_[:,:2]
x_hat_ = z_[:,2:]
theta_,dtheta_, = x_.T
theta_hat_,dtheta_hat_, = x_hat_.T

# we'll use the Matplotlib library for plots
# (as the name suggests, it has a similar interface to MATLAB)
import pylab as plt
plt.figure()

# plot theta, theta_hat
plt.subplot(2,1,1)
plt.plot(t_,theta_,'--',lw=4,label=r'$\theta$')
plt.plot(t_,theta_hat_,label=r'$\widehat{\theta}$')
plt.ylabel(r'$\theta(t)$')
```

```
# plot dtheta, dtheta_hat
plt.subplot(2,1,2)
plt.plot(t_,dtheta_,'--',lw=4,label=r'$\dot{\theta}$')
plt.plot(t_,dtheta_hat_,label=r'$\dot{\widehat{\theta}}$')
plt.ylabel(r'$\dot{\theta}(t)$')
plt.xlabel(r'time $t$')

plt.figure()
plt.subplot(1,1,1)
plt.plot(theta_,dtheta_,'--',lw=4,label=r'$x$')
plt.plot(theta_hat_,dtheta_hat_,label=r'$\widehat{x}$')
plt.ylabel(r'$\dot{\theta}(t)$')
plt.xlabel(r'${\theta}(t)$')
plt.legend();
```

```
(2000, 4)
```





2. transfer functions

Consider the model of a series RLC circuit from lecture,

$$L\ddot{q} + R\dot{q} + q/C = v,$$

where q denotes the charge on the capacitor, (R, L, C) denote the (resistor, inductor, capacitor) parameters, and v denotes a series voltage source.

Purpose: practice determining transfer functions from state-space representations of LTI systems.

a. Determine the transfer function from input voltage to capacitor charge $G_{qv}(s)$.

Solution:

Laplace transform to obtain: $s^2Lq + sRq + \frac{q}{c} = v$

$$\boxed{G_{qv} = \frac{q}{v} = \frac{1}{s^2L + sR + 1/C}}$$

---

b. With $x = (q, \dot{q})$ as the circuit's state, $u = v$ as the circuit's input, and $y = q$ as the circuit's output, determine matrices A, B, C, D such that

$$\dot{x} = Ax + Bu, \quad y = Cx + Du.$$

Solution:

By determining that $\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = A \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + Bv$ we can compare this against our equation for $\ddot{q} = \frac{v}{L} - \frac{R}{L}\dot{q} - \frac{q}{LC}$ to obtain:

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-1}{LC} & \frac{-R}{L} \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} v$$

and similarly for $y = q = C \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + Dv = \begin{bmatrix} 1 & 0 \end{bmatrix} + 0v$

$$\Leftrightarrow \boxed{A = \begin{bmatrix} 0 & 1 \\ \frac{-1}{LC} & \frac{-R}{L} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, D = 0}$$

---

c. Verify that the transfer function of the state-space system in (b.) is the same as the transfer function determined in (a.). (***Hint:*** you just need to compute $C(sI - A)^{-1}B + D$ and verify that it equals $G_{qv}(s)$).

Solution:

$$C(sI - A)^{-1}B + D = \begin{bmatrix} 1 & 0 \end{bmatrix} (\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ \frac{-1}{LC} & \frac{-R}{L} \end{bmatrix})^{-1} \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} + 0 = \begin{bmatrix} 1 & 0 \end{bmatrix} (\begin{bmatrix} s \\ \frac{1}{LC} \end{bmatrix}$$

Use your favorite computational tool to compute this inverse (I actually calculated by hand since for any 2×2 matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{det(M)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$). Use this to determine that this matrix multiplication comes out to:

$$C(sI - A)^{-1}B + D = \begin{bmatrix} 1 & 0 \end{bmatrix} (\begin{bmatrix} s & -1 \\ \frac{1}{LC} & s + \frac{R}{L} \end{bmatrix})^{-1} \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} + 0 = \frac{1}{s^2 + \frac{R}{L}s + \frac{1}{LC}} (\begin{bmatrix} 1 & 0 \end{bmatrix} (\begin{bmatrix} s$$

$$\boxed{G_{qv} = \frac{1}{s^2 L + sR + 1/C}}$$

---

d. With $\tilde{x} = (\frac{1}{2}(q + \dot{q}), \frac{1}{2}(q - \dot{q}))$ as the state, $u = v$ as the input, and $y = q$ as the output, determine matrices $\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{D}$ such that

$$\dot{\tilde{x}} = \widetilde{A}\tilde{x} + \widetilde{B}u, \quad y = \widetilde{C}\tilde{x} + \widetilde{D}u.$$

Solution:

$$\widetilde{A} = TAT^{-1}, \widetilde{B} = TB$$

We determine T using the relation

$$\tilde{x} = Tx$$

$$\begin{bmatrix} \frac{1}{2}(q + \dot{q}) \\ \frac{1}{2}(q - \dot{q}) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$$

Applying this to find $\widetilde{A}$ and $\widetilde{B}$,

$$\widetilde{A} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ \frac{-1}{LC} & \frac{-R}{L} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} \frac{-(RC - LC + 1)}{2LC} & \frac{RC - LC - 1}{2LC} \\ \frac{RC + LC + 1}{2LC} & \frac{-(RC + LC - 1)}{2LC} \end{bmatrix}$$

$$\widetilde{B} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} = \begin{bmatrix} \frac{1}{2L} \\ \frac{-1}{2L} \end{bmatrix}$$

$$\Leftrightarrow \boxed{\widetilde{A} = \begin{bmatrix} \frac{-(RC - LC + 1)}{2LC} & \frac{RC - LC - 1}{2LC} \\ \frac{RC + LC + 1}{2LC} & \frac{-(RC + LC - 1)}{2LC} \end{bmatrix}, \widetilde{B} = \begin{bmatrix} \frac{1}{2L} \\ \frac{-1}{2L} \end{bmatrix}}$$

$$y = q = \widetilde{C} \begin{bmatrix} \frac{1}{2}(q + \dot{q}) \\ \frac{1}{2}(q - \dot{q}) \end{bmatrix} + \widetilde{D}u = \begin{bmatrix} 1 & 1 \end{bmatrix} + 0u$$

$$\boxed{\widetilde{C} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \widetilde{D} = 0}$$

---

e. Verify that the transfer function of the state-space system in (d.) is the same as the transfer functions obtained in (a.), (c.). (***Hint:*** you just need to compute $\widetilde{C}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{D}$ and verify that it equals $G_{qv}(s)$).

Solution:

Transfer function for the state-space system is $\widetilde{C}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{D}$

$$\Leftrightarrow \begin{bmatrix} 1 & 1 \end{bmatrix} \left(\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} \frac{-(RC-LC+1)}{2LC} & \frac{RC-LC-1}{2LC} \\ \frac{RC+LC+1}{2LC} & \frac{-(RC+LC-1)}{2LC} \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{1}{2L} \\ \frac{-1}{2L} \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} 1 & 1 \end{bmatrix} \left(\begin{bmatrix} s + \frac{(RC-LC+1)}{2LC} & -\frac{RC-LC-1}{2LC} \\ -\frac{RC+LC+1}{2LC} & s + \frac{(RC+LC-1)}{2LC} \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{1}{2L} \\ \frac{-1}{2L} \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} 1 & 1 \end{bmatrix} \frac{1}{s^2 + (\frac{1}{2LC} - \frac{1}{2} + \frac{R}{2L} - \frac{1}{2LC} + \frac{1}{2} + \frac{R}{2L})s + ((\frac{1}{2LC} - \frac{1}{2} + \frac{R}{2L})(-\frac{1}{2LC} + \frac{1}{2} +}$$

$$\Leftrightarrow \frac{1}{s^2 + \frac{R}{L}s + \frac{1}{LC}} \begin{bmatrix} s + 1 + \frac{R}{L} & s - 1 + \frac{R}{L} \end{bmatrix} \begin{bmatrix} \frac{1}{2L} \\ -\frac{1}{2L} \end{bmatrix}$$

$$\boxed{G(s) = \frac{\frac{1}{L}}{s^2 + \frac{R}{L}s + \frac{1}{LC}}}$$

This matches the transfer function we get for G_{qv}.

Bonus: Show that the transfer function is the same regardless of the choice of coordinates for the state vector. (***Hint:*** Let $\tilde{x} = Tx$ be an arbitrary different choice of state vector (assume T is invertible), determine $\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{D}$ such that $\dot{\tilde{x}} = \widetilde{A}\tilde{x} + \widetilde{B}u,\ y = \widetilde{C}\tilde{x} + \widetilde{D}u$, and verify that $\widetilde{C}(sI - \widetilde{A})^{-1}\widetilde{B} + \widetilde{D} = C(sI - A)^{-1}B + D$.)

Takeaway: the transfer function of a state-space system is the same regardless of what coordinates you choose for the state vector. Thus, if we are primarily concerned with a system's input/output response (i.e. how input signal u transforms to output signal y) and less concerned with the details of what happens "inside" the system (i.e. how the state x is influenced by the input), the transfer function provides a convenient representation.

3. Bode plots

Purpose: visualize and manipulate transfer functions and their interconnections using ***Bode plots***, that is, plots that show how sinusoidal inputs are amplified/attenuated and phase-shifted by a transfer function G as a function of the input frequency ω.

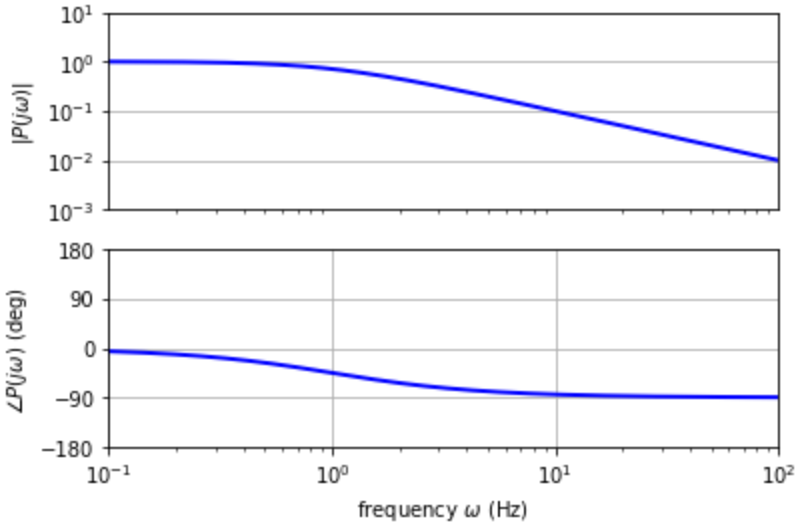
Consider the transfer function process model for an RC circuit with a voltage source as the input and capacitor voltage as the output,

$$P(s) = \frac{1}{1+sRC}.$$

Use parameter values $R = 1\text{M}\Omega$, $C = 1\mu\text{F}$ (i.e. $RC = 1$) in the remainder of the problem.

a. Create the ***Bode plot*** for P, that is, plot $|P(j\omega)|$ and $\angle P(j\omega)$ versus ω. (***Hint:*** refer to the RLC Circuit example in Section 2 of the lecture examples notebook for plotting code; you should use the `plt.subplot`, `plt.loglog`, and `plt.semilogx` functions in your solution.)

In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt

R = 1 # mOhm
C = 1 # uF

P = lambda s : 1. / (1+s*R*C)

omega = np.logspace(-1,2,1000)
xlim = omega[[0,-1]]

abs_P = np.abs(P(1.j*omega))
angle_P = np.unwrap(np.angle(P(1.j*omega)))*180./np.pi

plt.figure(); axs = []
ax = plt.subplot(2,1,1); ax.grid(True)
ax.loglog(omega,abs_P,'b-',lw=2)
ax.set_ylabel(r'$|P(j\omega)|$')
ax.set_yticks([10**-3,10**-2,10**-1,10**0,10**1])
ax.set_xticks([])
ax.set_xlim(xlim)
axs.append(ax)

ax = plt.subplot(2,1,2); ax.grid(True)
ax.semilogx(omega,angle_P,'b-',lw=2)
ax.set_xlabel(r'frequency $\omega$ (Hz)')
ax.set_ylabel(r'$\angle P(j\omega)$ (deg)')
ax.set_yticks([180,90,0,-90,-180])
ax.set_xlim(xlim)
axs.append(ax)
```

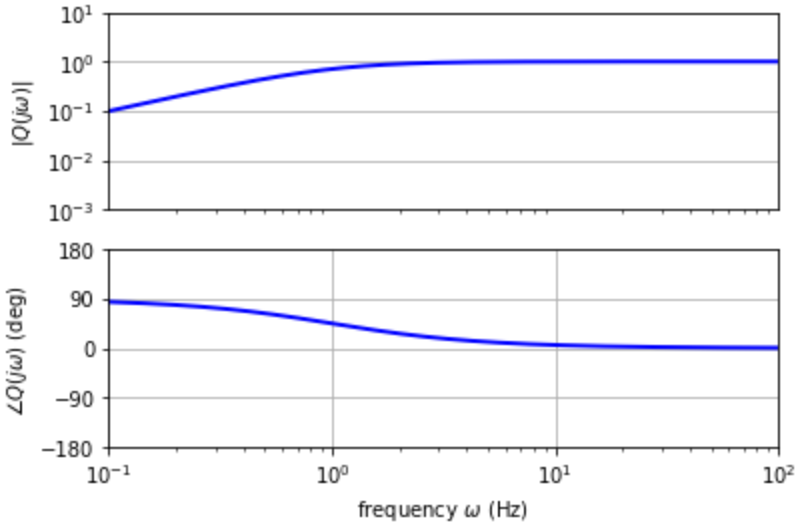


If we instead consider the transfer function from the input voltage to the resistor voltage, we obtain the transfer function

$$Q(s) = \frac{sRC}{1+sRC}.$$

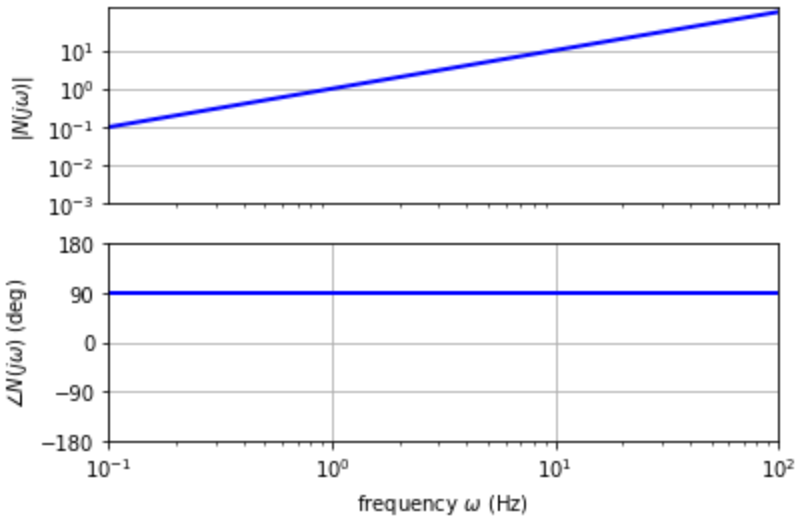
b. Create the Bode plot for Q.

In [ ]:
```
Q = lambda s : s*R*C/(1+s*R*C)

abs_Q = np.abs(Q(1.j*omega))
angle_Q = np.unwrap(np.angle(Q(1.j*omega)))*180./np.pi

plt.figure(); axs = []
ax = plt.subplot(2,1,1); ax.grid(True)
ax.loglog(omega,abs_Q,'b-',lw=2)
ax.set_ylabel(r'$|Q(j\omega)|$')
ax.set_yticks([10**-3,10**-2,10**-1,10**0,10**1])
ax.set_xticks([])
ax.set_xlim(xlim)
axs.append(ax)

ax = plt.subplot(2,1,2); ax.grid(True)
ax.semilogx(omega,angle_Q,'b-',lw=2)
ax.set_xlabel(r'frequency $\omega$ (Hz)')
ax.set_ylabel(r'$\angle Q(j\omega)$ (deg)')
ax.set_yticks([180,90,0,-90,-180])
ax.set_xlim(xlim)
axs.append(ax)
```



c. Create the Bode plot for the numerator $N(s) = sRC$ of Q.

In [ ]:
```
N = lambda s : s*R*C

abs_N = np.abs(N(1.j*omega))
angle_N = np.unwrap(np.angle(N(1.j*omega)))*180./np.pi

plt.figure(); axs = []
ax = plt.subplot(2,1,1); ax.grid(True)
ax.loglog(omega,abs_N,'b-',lw=2)
ax.set_ylabel(r'$|N(j\omega)|$')
ax.set_yticks([10**-3,10**-2,10**-1,10**0,10**1])
ax.set_xticks([])
ax.set_xlim(xlim)
axs.append(ax)

ax = plt.subplot(2,1,2); ax.grid(True)
ax.semilogx(omega,angle_N,'b-',lw=2)
ax.set_xlabel(r'frequency $\omega$ (Hz)')
ax.set_ylabel(r'$\angle N(j\omega)$ (deg)')
ax.set_yticks([180,90,0,-90,-180])
ax.set_xlim(xlim)
axs.append(ax)
```



d. Noting that $Q(s) = P(s)N(s)$, determine how the Bode plot of Q is related to the Bode plots of P and N. (***Note:*** the magnitude component of the Bode plot should be represented logarithmically.)

Solution:

Comparing the bode plot of P and N to the bode plot of Q, we can see that $PN = Q$. Specifically, the angles and magnitudes of P and N add to form the angles and magnitudes of Q

---

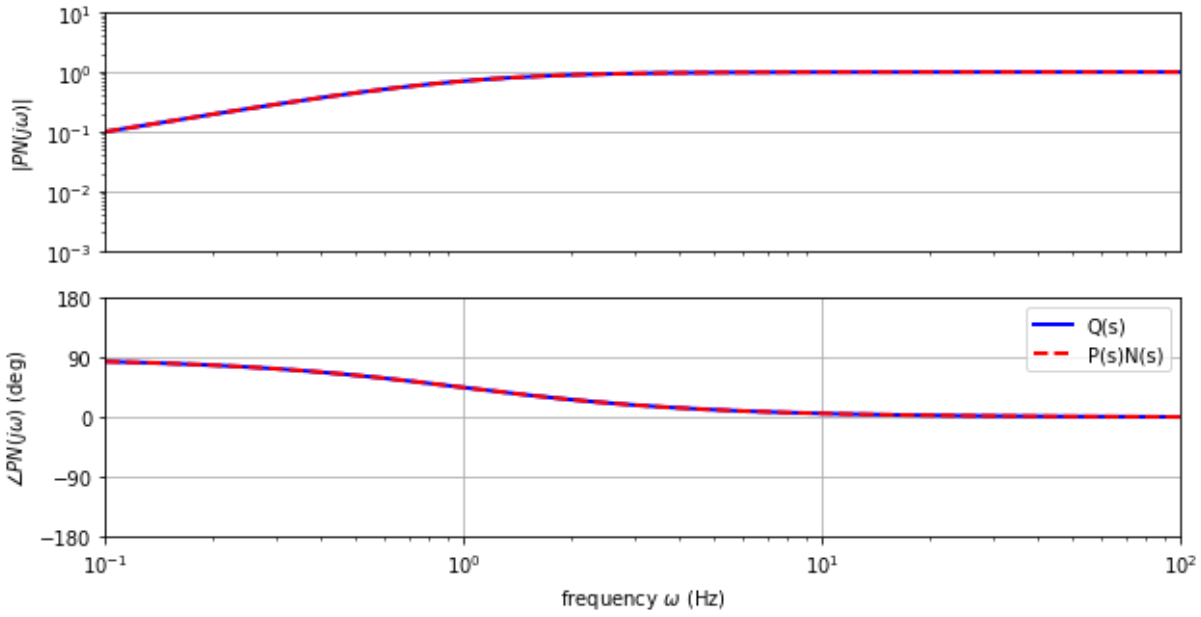
e. Verify the relationship you derived in (d.) by overlaying the Bode plot of $P(s)N(s)$ on your Bode plot of Q from (b.).

In [ ]:
```
# Compute P(s)N(s)
PN = lambda s : P(s)*N(s)

abs_PN = np.abs(PN(1.j*omega))
angle_PN = np.unwrap(np.angle(PN(1.j*omega)))*180./np.pi

abs_Q = np.abs(Q(1.j*omega))
angle_Q = np.unwrap(np.angle(Q(1.j*omega)))*180./np.pi

# plot Q
plt.figure(figsize=(10,5)); axs = []
ax = plt.subplot(2,1,1); ax.grid(True)
ax.loglog(omega,abs_Q,'b-',lw=2)
ax.set_ylabel(r'$|Q(j\omega)|$')
ax.set_yticks([10**-3,10**-2,10**-1,10**0,10**1])
ax.set_xticks([])
ax.set_xlim(xlim)
axs.append(ax)

ax = plt.subplot(2,1,2); ax.grid(True)
ax.semilogx(omega,angle_Q,'b-',lw=2,label='Q(s)')
ax.set_xlabel(r'frequency $\omega$ (Hz)')
ax.set_ylabel(r'$\angle Q(j\omega)$ (deg)')
```

```
ax.set_yticks([180,90,0,-90,-180])
ax.set_xlim(xlim)
axs.append(ax)

# plot P(s)N(s)
ax = plt.subplot(2,1,1); ax.grid(True)
ax.loglog(omega,abs_PN,'r--',lw=2)
ax.set_ylabel(r'$|PN(j\omega)|$')
ax.set_yticks([10**-3,10**-2,10**-1,10**0,10**1])
ax.set_xticks([])
ax.set_xlim(xlim)
axs.append(ax)

ax = plt.subplot(2,1,2); ax.grid(True)
ax.semilogx(omega,angle_PN,'r--',lw=2,label='P(s)N(s)')
ax.set_xlabel(r'frequency $\omega$ (Hz)')
ax.set_ylabel(r'$\angle PN(j\omega)$ (deg)')
ax.set_yticks([180,90,0,-90,-180])
ax.set_xlim(xlim)
axs.append(ax)

plt.legend()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: MatplotlibD
eprecationWarning: Adding an axes using the same arguments as a previous axe
s currently reuses the earlier instance.  In a future version, a new instanc
e will always be created and returned.  Meanwhile, this warning can be suppr
essed, and the future behavior ensured, by passing a unique label to each ax
es instance.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:37: MatplotlibD
eprecationWarning: Adding an axes using the same arguments as a previous axe
s currently reuses the earlier instance.  In a future version, a new instanc
e will always be created and returned.  Meanwhile, this warning can be suppr
essed, and the future behavior ensured, by passing a unique label to each ax
es instance.
```

Out[ ]: <matplotlib.legend.Legend at 0x7f1e6260cc90>



f. Describe the Bode plots of N, Q, and P as ***filters***, that is, explain how each system responds to "low frequency" inputs (specify a range of frequencies you regard as "low") and "high frequency" inputs (specify a range of frequencies you regard as "high"). Could you predict the combined response of $Q = PN$ based on the individual responses of N and P?

Solution:

N is a high-frequency amplifier that rejects (attenuates) low < 1 Hz signals and amplifies high > 1 Hz signals.

Q is a high-pass filter that rejects (attenuates) low < 1 Hz signals and passes (multiplies by 1) high > 1 Hz signals.

P is a low-pass filter that passes (multiplies by 1) low < 1 Hz signals and rejects (attenuates) high > 1 Hz signals.

We predict that $Q = PN$ will reject low < 1 Hz signals and passes high > 1 Hz signals

Takeaway: Bode plots provide a convenient visualization of a transfer function that enables us to determine how systems respond to input frequency components, individually and through interconnection.

In [ ]: