Prelab 3 : Filters

Omkar H. Ramachandran

February 1, 2018

Note: I do not have Mathematica installed on my work laptop - which is an 8+ year old Lenovo flex 10. Thus, I have completed all the prelab activities in Python and have written it up in LaTeX. In addition, all the code present in this document is available on my Github repository

1 Low and High-pass filters

1.1 Define Python functions to calculate the cutoff

For both low-pass and high-pass RC filters, the cut off frequency is the same:

$$f_C = \frac{1}{2\pi RC}$$

The code to return f_C given R and C is as follows:

```
def compute_cutoff_RC(R,C):
    """ Returns cutoff frequency for an RC filter """
    return 1./(2*np.pi*R*C)
```

This function can simply be called as follows:

```
R = 10e3
C = 1e-9
Fc = compute_cutoff_RC(R,C)
```

Evaluating this function for the values in the schematic yeilds $F_c = 15.9154\,kHz$

1.2 Bode plots for each filter

A bode plot is simply a plot of the transfer function $H(\omega)$ on log -log axes. We know that, for a low-pass filter,

$$|H(\omega)| = \frac{1}{\sqrt{1 + (RC\omega)^2}}$$

The code to generate these plots is as follows:

```
def plot_bode_RC(R,C,omega_min,omega_max,flag):
    """ Plots the bode function between the given limits. The flag
    decides whether the transfer function used is for the high-pass
    or low-pass filter"""
    Nomega = 1000
    H = np.zeros(Nomega)
    omega = np.linspace(np.log(omega_min),np.log(omega_max),Nomega)
```

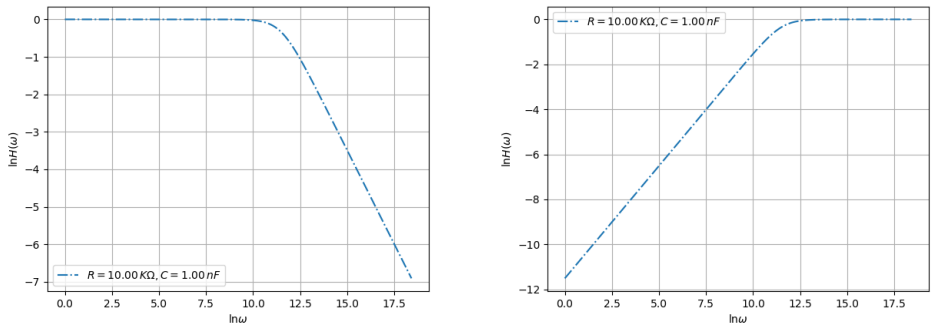

Figure 1: (left) Plot of low-pass and (right) high-pass bode curves for $R = 10\,K\Omega$ and $C = 1000\,pF$

```
    if(flag == 0):
        # Low pass
        H = 1./np.sqrt(1+(R*C*np.exp(omega))**2)
    elif(flag == 1):
        # High pass
        H = (R*C*np.exp(omega))/np.sqrt(1+(R*C*np.exp(omega))**2)
    plt.figure(1)
    plt.plot(omega,H,'r-.',label='$R = %.2f\, K\Omega, C = %.2f\, pF$' % (R
        /1e3,C/1e-9))
    plt.xlabel(r"$\ln\omega$")
    plt.ylabel(r"$H(\omega)$")
    plt.legend()
    plt.grid()
    plt.draw()
    return H
```

To call one of these functions, one simply needs to execute the following:

```
R = 10000
C = 10**(-9)
omega_min = 1.0
omega_max = 10**8
# For low pass, execute the following:
H = plot_bode_RC(R,C,omega_min,omega_max,0)
# And for high pass,
H = plot_bode_RC(R,C,omega_min,omega_max,1)
plt.show()
```

The program will automatically generate and label plots that look as in Figure 1:

1.3 Generate mock data

The following code snippet chooses 50 points uniformly on the log x scale and then plots them in the appropriate place. Note: This function must be called only **after** the main curve plotting function has already been called

```
def generate_mock_RC(R,C,omega_min,omega_max,flag):
    """ Generates lots of mock data that follows bode curve """
    Npoints = 50
    Nomega = 1000
```

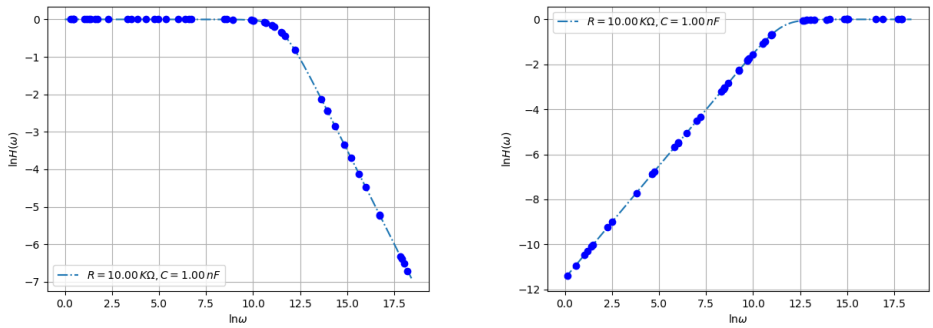

Figure 2: (left) Plot of low-pass and (right) high-pass bode curves for $R = 10\,K\Omega$ and $C = 1000\,pF$. Each curve has further been populated with fake data points

```
    omega = np.linspace(np.log(omega_min),np.log(omega_max),Nomega)
    P_omega = (1.0*np.ones(Nomega))/Nomega
    plt.figure(1)
    for i in range(Npoints):
        omega_choice = np.random.choice(omega,p=P_omega)
        if(flag == 0):
            # Low pass
            H = 1./np.sqrt(1+(R*C*np.exp(omega_choice))**2)
        elif(flag == 1):
            # High pass
            H = (R*C*np.exp(omega_choice))/np.sqrt(1+(R*C*np.exp(
                omega_choice))**2)
        plt.plot(omega_choice,H,'bo')
        plt.draw()
```

Running this function creates plots as in Figure 2

2 Band-pass Filters

2.1 Define functions that calculate f_0

For the band-pass filter, the required parameters are defined as follows:

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

$$Q = 2\pi f_0 RC$$

$$Z_0 = 2\pi f_0 L$$

Thus, the code to compute these is as follows:

```
def compute_parameters_BP(R,L,C):
    """ Returns the required parameters for a band-pass filter """
    f0 = 1/(2*np.pi*np.sqrt(L)*np.sqrt(C))
    Q = 2*np.pi*f0*R*C
    Z0 = 2*np.pi*f0*L
    return np.array([f0,Q,Z0])
```

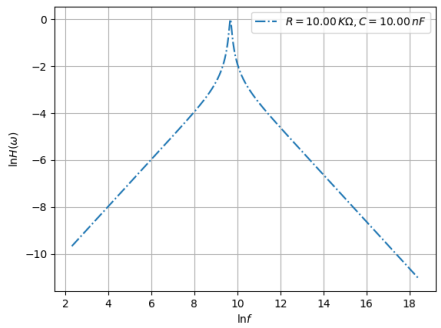

Figure 3: Plot of band-pass bode curves for $R = 10\,K\Omega$ and $C = 1000\,pF$

To run this, we simply use the following:

```
R = 10e3
L = 10e-3
C = 1e-8
f0,Q,Z0 = compute_parameters_BP(R,L,C)
```

For the values in the schematic, the output from the code is $f_0 = 15.9154\,kHz$, $Q = 9.999$ and $Z_0 = 999.9\,\Omega$

2.2 Bode plots for band-pass filters

The code to generate bode plots for the band pass filter is as follows:

```
def plot_bode_BP(R,L,C,omega_min,omega_max):
    """ Plots the bode function between the given limits. The flag
    decides whether the transfer function used is for the high-pass
    or low-pass filter"""
    Nomega = 1000
    H = np.zeros(Nomega)
    omega = np.linspace(np.log(omega_min),np.log(omega_max),Nomega)
    H = R/np.sqrt(R**2+(np.exp(omega)*L-1/(np.exp(omega)*C))**2)

    plt.figure(1)
    plt.plot(omega,H,'r-.',label='$R = %.2f\, K\Omega, C = %.2f\, pF$' % (R
        /1e3,C/1e-9))
    plt.xlabel(r"$\ln\omega$")
    plt.ylabel(r"$H(\omega)$")
    plt.legend()
    plt.grid()
    plt.draw()
    return H
```

It generates plots as in Figure 3.

2.3 Mock Data for the Band Pass filter

The code to generate mock data on the bode-curve is as follows:

```
def generate_mock_BP(R,L,C,omega_min,omega_max):
    """ Generates lots of mock data that follows bode curve """
```

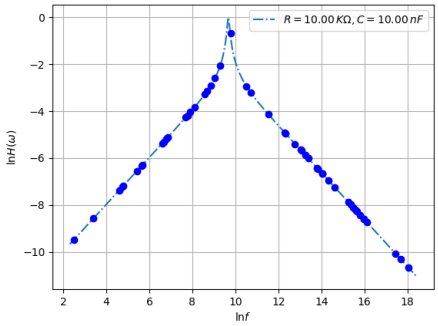

Figure 4: Plot of band-pass bode curves for $R = 10\,K\Omega$ and $C = 1000\,pF$. Each curve has further been populated with fake data points

```
Npoints = 50
Nomega = 1000
omega = np.linspace(np.log(omega_min),np.log(omega_max),Nomega)
P_omega = (1.0*np.ones(Nomega))/Nomega
plt.figure(1)
for i in range(Npoints):
    omega_choice = np.random.choice(omega,p=P_omega)
    H = R/np.sqrt(R**2+(np.exp(omega_choice)*L-1/(np.exp(omega_choice)*
        C))**2)
    plt.plot(omega_choice,H,'bo')
    plt.draw()
```

Once again, it must only be called after the curve is already plotted, since it inherits the axial labeling. Once called, it generates values as in Figure 4

3 Lab Activities

- Step 6 looks to be the hardest, especially point b, given that we need to use the prelab model in conjunction with the oscilloscope measurement
- I'm still not completely sure as to why we needed to generate the mock data.