12-778 Fall 2022: Assignment #1 Solutions

Mario Bergés

Here are the solutions to the assignment along with a grading rubric for each item.

What are sensors? (5%)

Pick an analog sensor from an online provider (one that you can easily purchase and would be interested in understanding better for your final project) and answer the following questions:

Since both of these questions require input from the student on the selected sensor, the answers here will depend entirely on that selection. A representative answer for the first question is provided here, but it is to be treated as a reference guide only. The point of this exercise is for us to better understand how much you have assimilated about the course content regarding these issues. There is not a single right answer here.

a) What are the potential applications for this sensor? Describe two.

Answer

Current application: The ADXL330 is a high-sensitive triaxial accelerometer, and it has been used in smart phones, game consoles, digital cameras, structure vibration tests and GPS systems. For smartphone applications, it is usually used to measure the motion of the phone and facilitate interaction with the phone (e.g., turn on the screen when the phone is picked up). The ADXL330 could also help game consoles equipped with motion controllers to transfer our actions with the gamepad to the system. In digital cameras, ADXL330 can capture the small vibrations occuring when we take a picture and report this to a anti-shock system to provide compensation. In GPS systems and software such as Google Maps, it may keep track of our motion and orientation.

b) How does this sensor operate, physically? In other words, how does the sensor transform energy in the physical phenomenon being measured into an electrical signal?
c) What is this sensor's accuracy? What is its transfer function? What other static and/or dynamic properties are worth discussing?

d) When considering the specific application that you have in mind, what are the advantages and disadvantages of this specific sensor compared to other alternative sensing technologies for the same physical stimulus?
e) How much does it cost, and where can you buy it?
f) What kind of interface circuit would be needed to connect it to your Raspberry Pi Pico W? Please sketch the circuit.

Working with sensors and your RPi Pico W (10%)

The ADC in your RPi Pico W has 3 available channels for you to supply whatever inputs you like, but there is a fourth channel that is directly connected to an on-board temperature sensor. You can easily access this fourth channel in the same way that you do the other three, but you won't need to connect any analgo sensor to it as it is hard-wired to the temperature sensor. Can you leverage the information on the Raspberry Pi Pico Python SDK (specifically, page 14) to interface with this sensor and answer the following questions?

a) What is the sensor's transfer function?

Answer

The sensor converts temperature (in Fahrenheit) into a voltage in a linear fashion (at least in some range near ambient temperature), using the following linear model:

$$V = (27 - T)0.001721 + 0.706 = 0.752467 - 0.001721T$$

Where V is the output voltage, and T is the temperature in Fahrenheit.
Conversely, and according to Page 14 of the Python SDK for the Pico, one can find the temperature given the voltage measurement by inverting the transfer function: $T = 27 - \frac{V-0.706}{0.001721}$.

b) Why do we need to take the ADC measurements and divide them by 65535? Why do we multiply them by 3.3?

Answer

According to the documentation (including Page 14 of the Python SDK for the Pi Pico), the Pico's ADC measures a voltage value between 0 and 3.3V using a 12 bit resolution. However, when calling the ADC through the `read_u16()` method of the ADC class in Mycropython, this value is then cast into an integer in the range (0 to 65535) corresponding to 2 bytes (16 bits). Thus, range from 0V to max input voltage gets converted into a number between 0 and 65535 or, in other words, each increment of one integer obtained through the `read_u16()` method corresponds to 1/65535 the maximum input range of

the ADC. Given all this, the values that one obtains from the ADC need to be divided by 65535 and multiplied by the maximum input (3.3V) for them to be converted into voltage values.

c) Create a Pico program that allows you to collect temperature data for a 10 minute interval, at 1s resolution.

Answer

```
import machine
import time

fcsv = open('data.csv','w', encoding="utf-8")
fbin = open('data.bin','wb')

temp_sensor = machine.ADC(4)

temperature = lambda v: 27 - (v-0.706)/0.001721

for t in range(1,600):
    voltage = temp_sensor.read_u16() * 3.3/65535
    temp = temperature(voltage)
    fscv.write(f'{temp}, ')
    fbin.write(temp)
    time.sleep(1)
```

d) Estimate the memory footprint of the data that you will collect using this program.

Answer

For the binary format, we would expect 2 bytes per measurement (since `read_u16()` returns an unsigned short), and a total of 1200 bytes.
For the CSV formate, we expect a much larger number given that the unsigned short is first converted into a string of characters for the decimal representation of the value which has a maximum of 5 characters, and then a comma (and a space) are added to the string. This represents at most 7 characters per measurement stored. Since we are using UTF-8 encoding, this would mean (in the general case) 4 bytes per character, which turns into 28 bytes per measurement stored (7×4), or a total of $600 \times 28 = 16800$ bytes for the whole set of measurements (14 times more than in binary format).

e) Collect data for 10 minutes and save it (either locally, or remotely). What is the file size? Compare it with the estimated memory footprint in the last question and comment on this.

> Answer
>
> If we did things correctly, the answers from the previous question should match what we find save for some additional considerations such as the size of the file on disk (which depends on the filesystem).

Harmonic Oscillators (10%)

In class we discussed dynamic characteristics of sensors by looking into the response of single degree of freedom systems to harmonic loading. Unfortunately, we did not have enough time to solve the equations of motion for the damped forced oscillator case, or to play around with the resulting solutions. Thus, for this task, and to make sure the concepts are more intuitive to you, I ask that you play with a simulated harmonic oscillator, borrowed and later modified from here.

The problem is defined as finding the solutions to the following differential equation:

$$\ddot{x} + c\dot{x} + kx = \frac{F(t)}{m}$$

or, if we know that $\omega_0 = \sqrt{\frac{k}{m}}$, which is the angular frequency of the oscillator when undamped; and $\zeta = \frac{c}{2\sqrt{mk}}$ is the so-called damping ratio, we can rewrite it as such:

$$\ddot{x} + 2\zeta\omega_0\dot{x} + {\omega_0}^2 x = \frac{F(t)}{m}$$

We will start with the case where $\frac{F(t)}{m} = F_m sin(\omega_d t)$, i.e. the oscillator is driven by a sinusoidal force of amplitude F_m and frequency ω_d.

```
%matplotlib notebook
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import integrate
import ipywidgets as ipw
```

```
def ode(X, t, zeta, omega0):
    """
    Free Harmonic Oscillator ODE
    """
    x, dotx = X
    ddotx = -2*zeta*omega0*dotx - omega0**2*x
    return [dotx, ddotx]

def odeDrive(X, t, zeta, omega0, omegad_omega0):
    """
    Driven Harmonic Oscillator ODE
    """
    x, dotx = X
    omegad = omegad_omega0 * omega0
    ddotx = -2*zeta*omega0*dotx - omega0**2*x + F_m * np.sin(omegad * t)
    return [dotx, ddotx]

def update(zeta = 0.05, omega0 = 2.*np.pi, omegad_omega0 = 1.): #def update(c=1,m=10,k=0.5
    """
    Update function.
    """
    #zeta = c/(2.*np.sqrt(m*k))
    #omega0 = np.sqrt(k/m)
    #omegad_omega0 = omegad/omega0
    X0 = np.zeros(2)
    sol = integrate.odeint(odeDrive, X0, t, args = (zeta, omega0, omegad_omega0))
    line0.set_ydata(sol[:, 0])
    fig.canvas.draw()
    fig.canvas.flush_events()

Nt = 1000
F_m = 1.
t = np.linspace(0., 10., Nt)
dummy = np.zeros_like(t)
fig = plt.figure()
line0, = plt.plot(t, dummy, label = "position")
plt.grid()
plt.ylim(-1., 1.)
plt.xlabel("Time, $t$")
plt.ylabel("Amplitude, $a$")
plt.legend()
```

```
ipw.interact(update, zeta = (0., .2, 0.01),
             omega0 = (2.*np.pi*0.5, 2.*np.pi*5, 2.*np.pi*0.01),
             omegad_omega0 = (0.1, 2., 0.05));
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
interactive(children=(FloatSlider(value=0.05, description='zeta', max=0.2, step=0.01), FloatS
```

To start, take the code above and copy/paste it to a Jupyter Notebook so that you can play around with the interactive interface that it provides. Currently, the interface is set up to allow you to set three parameters, namely ζ, ω_0 and $\frac{\omega_d}{\omega_0}$. That said, it is relatively easy to change the interface so that you can directly alter c, m, k and ω_d. Your task is to leverage the interactive interface and get familiarized with the response of the system as you change the damping, stiffness, mass and frequency of the harmonic loading. Try answering these questions for yourself:

a) What is the relationship between the amplitude and frequency of the harmonic loading, and the amplitude and frequency of the system's response?

> Answer
>
> This relationship is entirely described by the frequency response function (FRF). In general, the higher the amplitude of the output is a scaled copy of the input and the scaling is inversely related to the mass, frequency, spring constant and damping factor. This relationship is complex and leads to different behaviors depending on the values of these parameters (especially ω_d and c, but can be described as $X(\omega) = \frac{1}{\sqrt{(k-m{\omega_d}^2)^2+c^2{\omega_d}^2}} F_m$.

b) What happens when you drive the system at the resonant frequency?

> Answer
>
> It depends on the damping ratio, but in the underdamped case there would be resonance.

Other questions you may want to ask yourself about this system:

c) In practice, one can assume that the natural frequency of a damped system can be taken to be equal to the undamped natural frequency, because the damping ratio is less than 10%. Does this make sense given the simulation results you can observe? How large of a damping ratio can you expect to have while still maintaining this assumption?

Answer

As shown in Chapter 2 of Paz, for an underdamped system ($c < c_{cr}$), the damped frequency ω_D is equivalent to $\omega_0\sqrt{1-\zeta^2}$.

d) If someone sets the damping ratio randomly, would you be able to estimate its value simply from the simulated response of the system?

Answer

If the input signal were a step function, then yes. But discerning the difference between the transient and steady-state response when the input is a periodic function becomes hard.

Once you are done, please write a brief summary of your overall findings (2 or 3 paragraphs of thoughts, not just about the last few questions) as your answer to this part of the assignment.

Circuit Analysis (25%)

Task a (10%)

If you think carefully about what we've learned about complex impedances and how they work in AC circuits, you'll quickly realize that almost any component in your system can be seen as a filter. It may not be evident, but even the wire you are using to connect components together, itself, can act as a filter. Let's explore that a bit.

Virtually all cables have a very small, but detectable capacitance. This is because the insulation material around each of the wires closing the circuit acts as a dielectric and can accumulate charge when a voltage is present.

Figure 1 shows a diagram of how this works out in practice. If, for example, we had an ideal microphone as the voltage source (V_{in}) in this circuit, and wanted to measure the signal coming out (V_{out}), we would find that the values of Z_1 and Z_2, namely the resistance and capacitance of the cable itself, would influence the signal we receive.

To study the effect of this filter, let's analyze the ratio of the magnitudes for V_{in} and V_{out}. In other words, let's study how the voltage we measure is related to the voltage being supplied by the microphone, as described below:

$$\frac{V_{out}}{V_{in}}$$

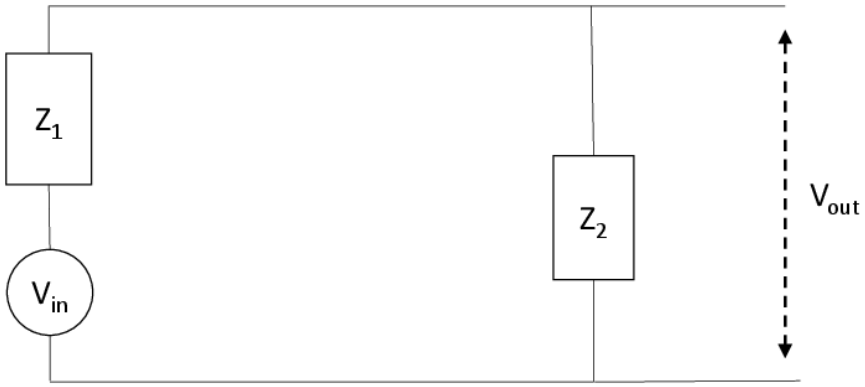


Figure 1: Simple circuit generalizing the effect of cable resistance and capacitance.

Task a: If $Z_1 = R$ and $Z_2 = \frac{1}{j\omega C}$, where C is the capacitance of the cable, and R is the resistance, then what is the expression for $\frac{V_{out}}{V_{in}}$?

Answer

You may recognize the circuit as a voltage divider with two impedances, which means that: $V_{out} = V_{in}\frac{Z_2}{Z_1+Z_2}$. In this particular case, $H(\omega) = \frac{V_{out}}{V_{in}} = \frac{1}{1+j\omega RC}$. The magnitude of this complex quantity, which we also call the Frequency Response Function (FRF) or $H(\omega)$ due to its dependence on ω once R and C are fixed, is as follows: $|H(\omega)| = \frac{1}{\sqrt{1+\omega^2R^2C^2}}$. This is the FRF for a low-pass filter, and it is the essence of the answer to this queestion. In other words, we expect this cable to act as a filter, attenuating high-frequency signals. The cut-off frequency for this filter is defined as the value of ω at which the magnitude drops by 3dB (i.e., $|H(\omega) = \frac{1}{\sqrt{2}}$). You can calculate this by simple algebra:

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

$$\omega_c = \frac{1}{RC}$$

Task b: The capacitance of the wire (as well as the resistance) increase with its length l. In

other words, $C \propto l$ and $R \propto l$. What will happen to the dynamic properties of the measured signal from a microphone as we increase the length of the wire?

Answer

We know that the magnitude of any frequency component ω that is present in the input signal V_{in} is amplified/attenuated by $|H(\omega)|$. If we assume that R and C are variable, and proportional to l then generally speaking what we are saying is that:

$$|H(\omega, l)| = \frac{1}{\sqrt{1 + \omega^2(\alpha_R l + \beta_R)^2(\alpha_C l + \beta_R)^2}}$$

Where α_R and α_C are proportionality constants for R and C with respect to l (i.e., their corresponding sensitivity to l), respectively; and β_R and β_C are intercept terms for this proportionality (though we can assume them to be zero since when l is zero we have no resistance or capacitance. We can now investigate how $|H(\omega, R, C, l)|$ changes as a function of l. As you can assess from the equation, for any given ω its magnitude will decrease as we increase l. Essentially, as l increases, the cut-off frequency ($\frac{1}{\alpha_R \alpha_C l^2}$) decreases.

Task b (5%)

For the circuit in Figure 2, find the steady-state voltage across R_1, R_2 and C, if $V_s = 10$ V DC, $R_1 = 1$ kΩ, $R_2 = 1$ kΩ and $C = 0.01\mu$F.

Answer

The circuit is available for simulation and inspection on CircuitLab here
The voltage drops are $V(R_1) = 0$ volts, $V(R_2) = 10$ volts and $V(C) = 10$ volts. You can see that by moving your mouse over the specific elements on the CircuitLab model (after running the DC Solver under the Simulate tab).

Task c (10%)

Solve Exercise 2.24 from Chapter 2 (Analyis of Circuits) from Instrumentation for Engineering Measurements by Dally, Riley and McConnell.

Answer

Verifying Equation 2.45:

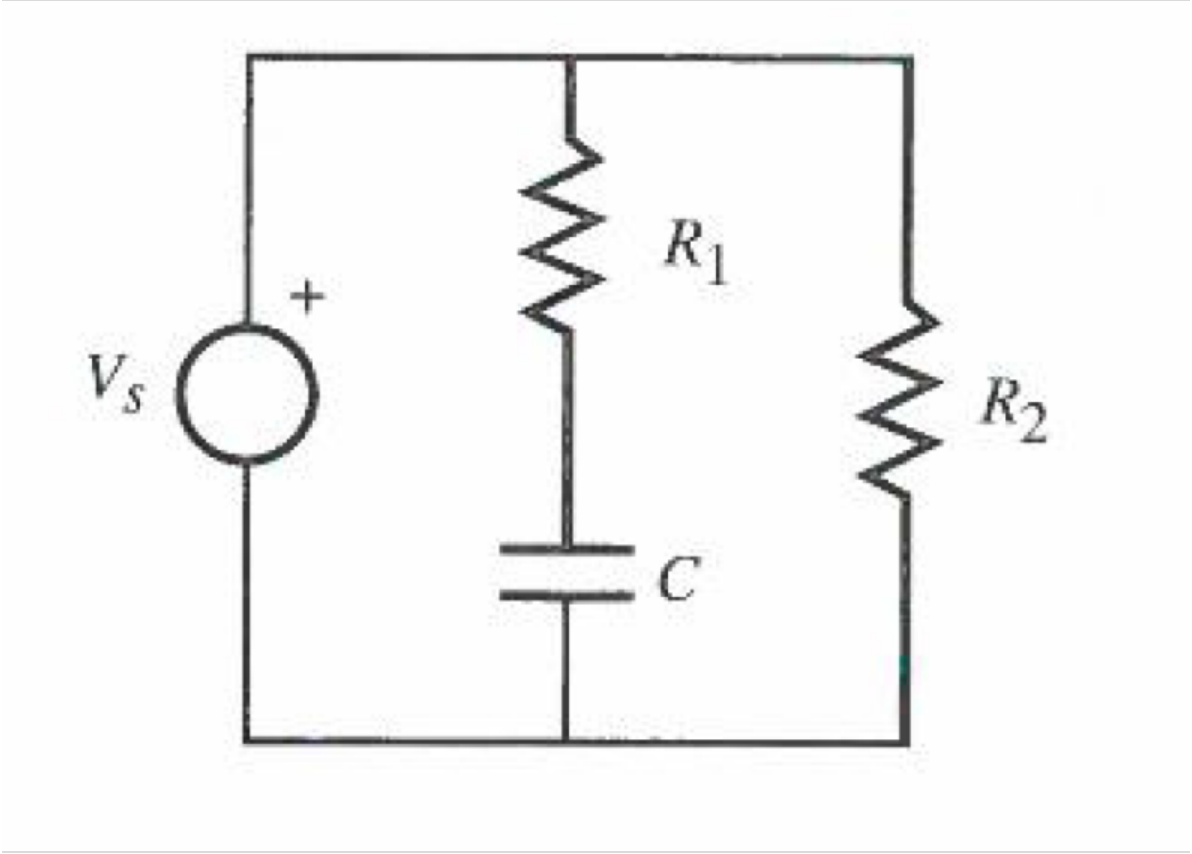


Figure 2: Simple DC circuit

$$Z_R = R$$

$$Z_C = \frac{-j}{\omega C}$$

$$v_o(t) = \frac{Z_C}{Z_C + Z_R} v_i e^{j\omega t}$$

$$v_o(t) = \frac{1}{1 + j\omega RC} v_i e^{j\omega t}$$

This is the same circuit we saw in Task a of the Circuit Analysis section of this assignment. The output voltage is a complex periodic function and we can write it down in polar form. We already computed the magnitude of the frequency response function (FRF) (i.e., $|\frac{v_o}{v_i}|$) but we need to also calculate the phase angle:

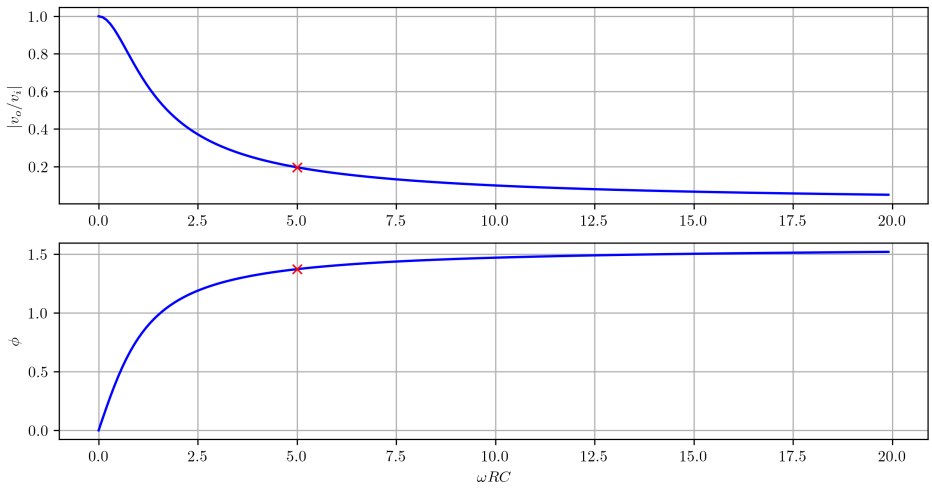
$$|v_o(t)| = \frac{v_i}{\sqrt{1 + (\omega RC)^2}}$$

$$\phi = tan^{-1}(\omega RC)$$

A graph of $|\frac{v_o}{v_i}|$ as a function of ωRC

```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as ipw

magnitude = lambda wrc: 1 / np.sqrt(1+np.square(wrc))

phase = lambda wrc: np.arctan(wrc)


def update_plot(omega, R, C):
    wrc_now = omega*R*C
    testmag.set_data((wrc_now, magnitude(wrc_now)))
    testphase.set_data((wrc_now, phase(wrc_now)))
    fig.canvas.draw()
    fig.canvas.flush_events()

wrc = np.arange(0,20,0.1)

omega = 1
R = 1
C = 5

plt.rcParams.update({
    "text.usetex": True,
    "font.family": "Helvetica"
})
fig, ax = plt.subplots(figsize=(10,5))
plt.subplot(2,1,1)
plt.plot(wrc,magnitude(wrc),'-b')
testmag, = plt.plot(omega*R*C, magnitude(omega*R*C), 'rx')
plt.grid()
plt.ylabel('$|{v_o}/{v_i}|$')
plt.subplot(2,1,2)
plt.plot(wrc,phase(wrc),'-b')
testphase, = plt.plot(omega*R*C, phase(omega*R*C), 'rx')
plt.grid()
plt.ylabel('$\phi$')
plt.xlabel('$\omega R C$')
plt.savefig('_ex-circuits-6_solution.png')

ipw.interact(update_plot, omega = (0., 2., 0.1), R = (0., 5., 0.1), C = (0., 5., 0.1))
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

interactive(children=(FloatSlider(value=1.0, description='omega', max=2.0), FloatSlider(val

<function __main__.update_plot(omega, R, C)>
```



What happens to the impedance Z_C as ω becomes very large?
As ω becomes very large, the impedance Z_C (more specifically, its magnitude $|Z_C|$) decreases significantly, tending to zero.

Impedance (20%)

In chapter 2 of Fraden, Figure Figure 3 shows up (Figure 2.15 in the book):

Task a: The concept of impedance is very important for circuit analysis. Please define, as best as you can (i.e., using technical concepts and mathematical notation), what impedance is.

Answer

Impedance is a concept used to extend the idea of *resistance* in DC circuits into AC circuits. Because, unlike in the DC case, the impedance of an element is dependent frequency of the voltage, the impedance is a complex number.

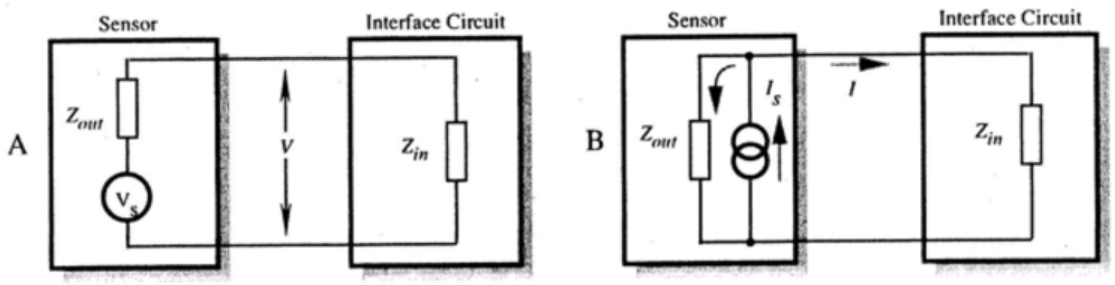


Figure 3: Sensor connections to an interface circuit. (a) shows a sensor with voltage output, while (b) shows a sensor with current output.

Task b: Impedance matching is also an important concept, as described by Fraden in page 40 of Chapter 2. Can you explain why it is recommended that a sensor with voltage output should have a much smaller impedance (Z_{out}) compared to the impedance (Z_{in}) of the interface circuit? (see Figure 3 A). Why is it different for Figure 3 B?

Answer

As we saw in class, this concept of impedance matching is very related to another concept called interstage loading (Chapter 6 of Figliola). There, we learned that if we want to have maximum voltage between stages, we need to set the Z_{out} should be much smaller than the input impedance Z_{in} because:

$$V - V_S = V_S \left(\frac{1}{1 + \frac{Z_{out}}{Z_{in}}} - 1 \right)$$

And we would like to reduce that error (i.e., reduce $V - V_S$).
For similar reasons, when the signal is driven by current, then we want the opposite.

Task c: Find a good video online explaining the concept of impedance matching (one whose explanation you find intuitive and clear) and provide the URL to that video as the answer to this question.

Answer

I'm very interested in seeing your solutions to this!

Task d: Compute the total impedance for the AC circuit shown in Figure Figure 4. Here, v_s is an AC voltage source (i.e., it is a time-varying source of voltage, varying as follows: $v_s = v_i e^{j\omega t}$). The frequency of this AC source is ω. In the same circuit, R, L and C represent the resistance, inductance and capacitance values for those circuit elements, respectively.

Answer

$$Z_L = j\omega L$$
$$Z_R = R$$
$$Z_C = \frac{1}{j\omega C} = \frac{-j}{\omega C}$$

Using complex number notation, the complex impedance would be:

$$Z_{Total} = Z_L + Z_R + Z_C = R + j\left(\omega L - \frac{1}{\omega C}\right)$$

In polar coordinates, we would need the magnitude (i.e., $|Z_{Total}|$) and phase of Z_{Total}:

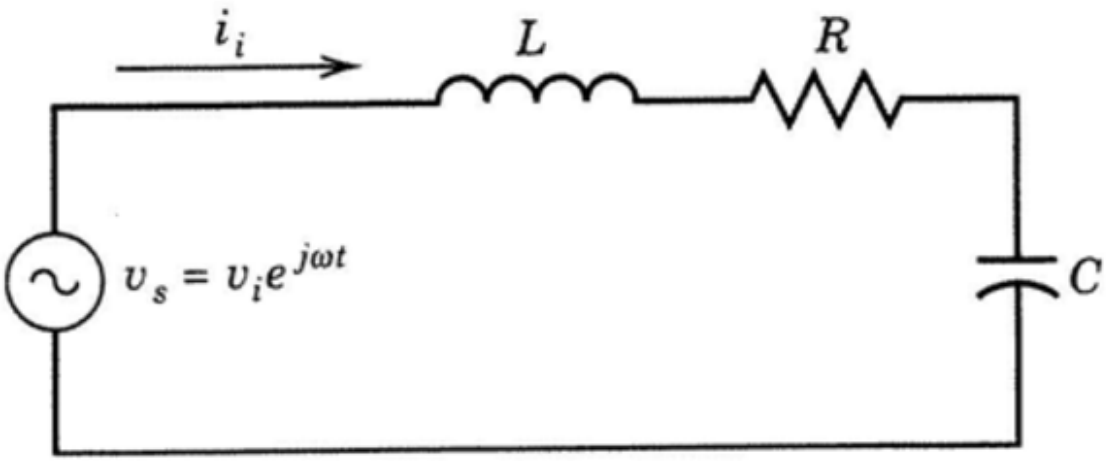


Figure 4: An AC circuit containing a resistance, capacitance and inductance in series.

$$|Z_{Total}| = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2}$$

$$\phi = \tan^{-1}\left(\frac{\omega L - 1/\omega C}{R}\right)$$

You could check what the current drawn by this total impedance would be, by remembering Ohm's law ($V = IZ$), and rearranging:

$$i_i = \frac{V_s}{Z_{Total}}$$

$$i_i = \frac{V_i}{R + j\left(\omega L - \frac{1}{\omega C}\right)} e^{j\omega t}$$

Analog-to-Digital Conversion (10%)

Suppose I set out to collect measurements about the voltage supplied by the electrical utility company to my house house for a week. I happen to know that the frequency of this voltage (in the US) is somehwere around 60Hz, but given that there is no guarantee it will maintain this frequency, and also considering the fact that the signal is not band-limited, I decide to over-sample.

Task a: Suppose I decide to sample it at 12 kHz with a 12-bit ADC. If I collect measurements for an entire week, how much memory will I need to store all of these samples?

Answer

A 12-bit value can be stored in 2 bytes (16 bits). If we are storing each value separately, then this would require $2\frac{\text{bytes}}{\text{sample}} \times 12000\frac{\text{samples}}{\text{second}} \times 8600\frac{\text{seconds}}{\text{day}} \times 7\frac{\text{days}}{\text{week}}$.

```
bytes_per_sample = 2
fs = 12000
seconds_in_day = 86400
days_in_week = 7
memory_footprint = bytes_per_sample * fs * seconds_in_day * days_in_week
print(f'The memory footprint is {memory_footprint} bytes/week.')
```

```
The memory footprint is 14515200000 bytes/week.
```

Another option is to combine the binary values (which are 12-bit), two at a time, into 3 bytes (i.e., 24 bits = 3 bytes). In that case, we could reduce the memory footprint by 1/4

at the expense of a slightly more complicated operation to read the values from memory.

Task b: Suppose now that I figure out a way to effectively make the signal band-limited, and I can guarantee that all the signal content will be below 70Hz. What would be a more efficient sampling rate in this case? How much memory would I require in this case?

Answer

In this case, we know that if we only care about being able to resolve the amplitude of the frequency components present in the signal then Nyquist-Shannon tells us that the sampling frequency should be larger than 140Hz. Assuming a value of $f_s = 150$ Hz, then we get:

```
bytes_per_sample = 2
fs = 150
seconds_in_day = 86400
days_in_week = 7
memory_footprint = bytes_per_sample * fs * seconds_in_day * days_in_week
print(f'The memory footprint is {memory_footprint} bytes/week.')
```

```
The memory footprint is 181440000 bytes/week.
```

Aliasing (10%)

In class, we learned why aliasing occurs and how it is related to the sampling frequency (or the Nyquist frequency) of the data acquisition configuration. Answer the following questions related to aliasing:

Task a: A 10Hz pure sine wave is sampled at 12 Hz. Compute the maximum frequency that can be represented in the resulting discrete signal. Compute the aliased frequency.

Answer

According to the Nyquist-Shannon sampling theorem, we know that $\frac{f_s}{2} = f_N$, where f_N is the Nyquist frequency (the maximum frequency that can be represented in the resulting discrete signal). In this case $f_N = \frac{12}{2} = 6$Hz. Since $10Hz > 6Hz$ there will be aliasing. In particular, the 10Hz signal will fold onto a 2Hz component.

Task b: Assume that the measured signal is complex periodic of the form $y(t) = A_1 sin(2\pi 25t) + A_2 sin(2\pi 75t) + A_3 sin(2\pi 125t)$. If this signal is sampled at 100Hz, determine the frequency content of the resulting discrete response signal.

Answer

When sampling this signal at 100Hz, any component that is above 50Hz will be aliased. Let's see how each of the components behaves after sampling:

- $A_1 sin(2\pi 25t)$ stays the same, as its frequency (25Hz) is under the Nyquist frequency.
- $A_2 sin(2\pi 75t)$ is aliased as a 25Hz component with amplitude $-A_2$.
- $A_3 sin(2\pi 125t)$ is aliased as another 25Hz component with amplitude A_3.

So the resulting signal has only one 25Hz component but with a combined amplitude of $(A_1 - A_2 + A_3)$.

Filters (10%)

A moving average is an filtering technique that can be applied to an analog or digital signal. A moving average is based on the concept of windowing as illustrated in Figure 5. The portion of the signal that lies inside the window in averaged and the average values are plotted as a function of time as the window moves across the signal. A 10-point moving average of the signal is plotted as well in Figure 6.

Task a: Discuss the effects of employing a moving average on the signal depicted in Figure 5. In particular, discuss the changes imparted to the dynamic characteristics of the signal. What does this say about the the transfer function for the moving average filter?

Answer

The moving average makes the resulting signal *smoother* (i.e., it removes the faster variations present in the original signal) while maintaining the general trend. It appears that the transfer function is some kind of low-pass filter.

Task b: Develop a simple Python program that computes the moving average for the following signal: $y(t) = sin(5t) + cos(11t)$, discretized by applying a 0.05second sampling train. Examine the effects of chaning the averaging window size from 4 to 30 samples.

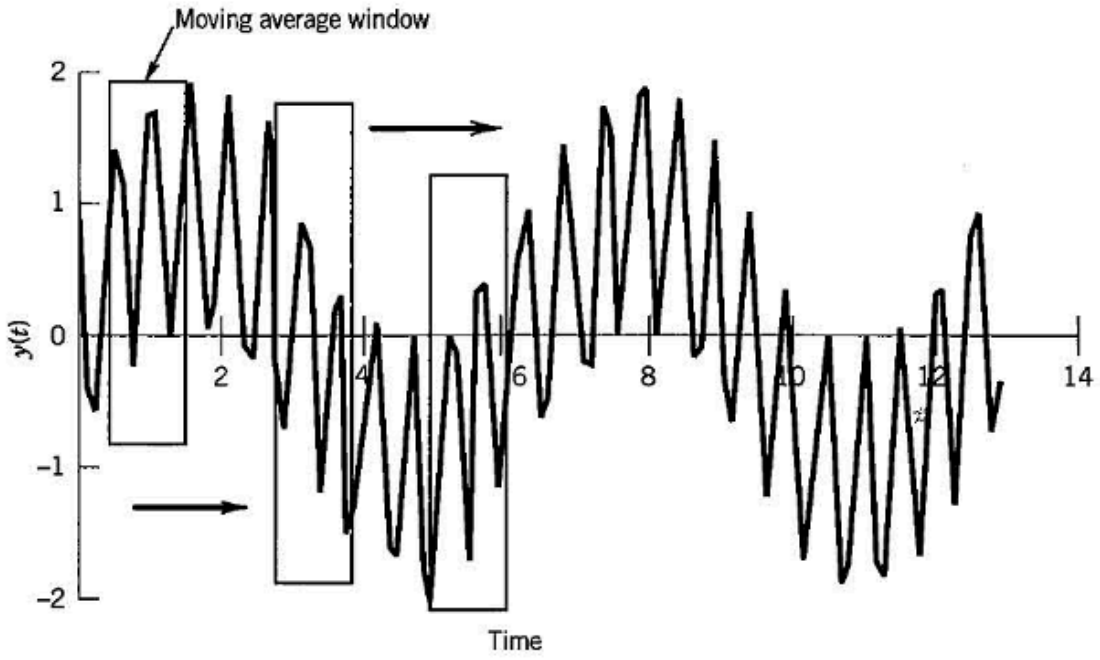


Figure 5: Moving averaging and windowing

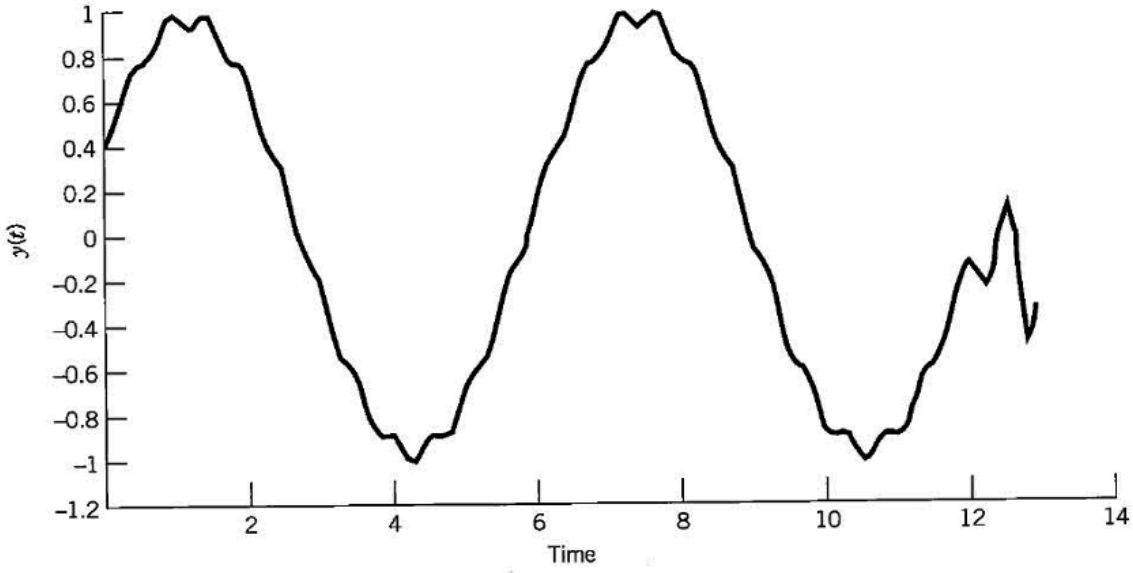
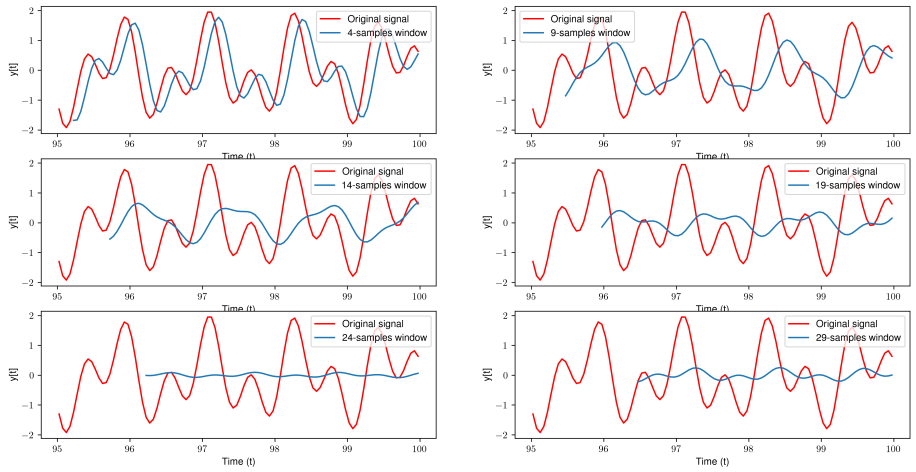


Figure 6: Effect of moving average on the signal.

Answer

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

window = 4      # samples
resolution = 0.05   # seconds

t = np.linspace(0,200,int(200/resolution))

y = np.sin(5*t) + np.cos(11*t)

def movwin(y,window=4):
    # There are way more efficient ways of doing this, but that's not the point of the e
    y_win = np.zeros(len(y))
    for i in range(len(y[:-window])):
        y_win[i] = np.sum(y[i:i+window])/window
    return y_win

plt.figure(1)
plt.subplots(figsize=(16,8))
zoom_range = (int(95/resolution),int(100/resolution))

for count, window in enumerate(range(4,30,5)):
    y_windowed = movwin(y,window)

    plt.subplot(3,2,count+1)
    plt.plot(t[zoom_range[0]:zoom_range[1]],y[zoom_range[0]:zoom_range[1]],'-r')
    plt.plot(t[zoom_range[0]+window:zoom_range[1]], y_windowed[zoom_range[0]:zoom_range[

    plt.legend(['Original signal', f'{window}-samples window'])
    plt.xlabel('Time (t)')
    plt.ylabel('y[t]')
    plt.savefig(f'_ex-filters-1_solution.png')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```



Task c: What did you learn about the effects of the width (number of samples) for the averaging window?

Answer

As the width of the window increased, the filtered signal became closer to a static signal hovering around the average of the original one. From this point of view, and loosely speaking, we can see the moving average as a low-pass filter. More technically, though, it is a special kind of finite impulse response filter, and if you want to learn more about the difference between low-pass filters and moving averages, you could start with this relatively approachable answer on StackExchange.