This homework is due on August 11, 2020, at 11:59PM.
Self-grades are due on Thursday, August 14, 2020, at 11:59PM.

1 Closed-Loop Control of SIXT33N

Last week, we discovered that open-loop control does not ensure that our car goes straight in the event of model mismatch. To make our control more robust, we introduce feedback, turning our open-loop controller into a closed-loop controller. In this problem, we derive the closed-loop control scheme you will use to make SIXT33N reliably drive straight.

Previously, we introduced $\delta[k] = d_L[k] - d_R[k]$ as the difference in positions between the two wheels. If both wheels of the car are going at the same velocity, then this difference δ should remain constant, since no wheel will advance by more ticks than the other. We will consider a proportional control scheme, which introduces a feedback term into our input equation in which we apply gains k_L and k_R to $\delta[k]$ to modify our input velocity at each timestep in an effort to prevent $|\delta[k]|$ from growing without bound. To do this, we will modify our inputs $u_L[k]$ and $u_R[k]$ accordingly:

We begin with our open-loop equations from last week:

$$\begin{aligned} v_L[k] &= d_L[k+1] - d_L[k] = \theta_L u_L[k] - \beta_L \\ v_R[k] &= d_R[k+1] - d_R[k] = \theta_R u_R[k] - \beta_R \end{aligned} \tag{1}$$

We want to adjust $v_L[k]$ and $v_R[k]$ at each timestep based on $\delta[k]$: instead of v^*, our desired velocities are now $v^* - k_L\delta[k]$ and $v^* + k_R\delta[k]$. As v_L and v_R go to v^*, $\delta[k]$ goes to zero.

$$\begin{aligned} v_L[k] &= d_L[k+1] - d_L[k] = v^* - k_L\delta[k] \\ v_R[k] &= d_R[k+1] - d_R[k] = v^* + k_R\delta[k] \end{aligned} \tag{2}$$

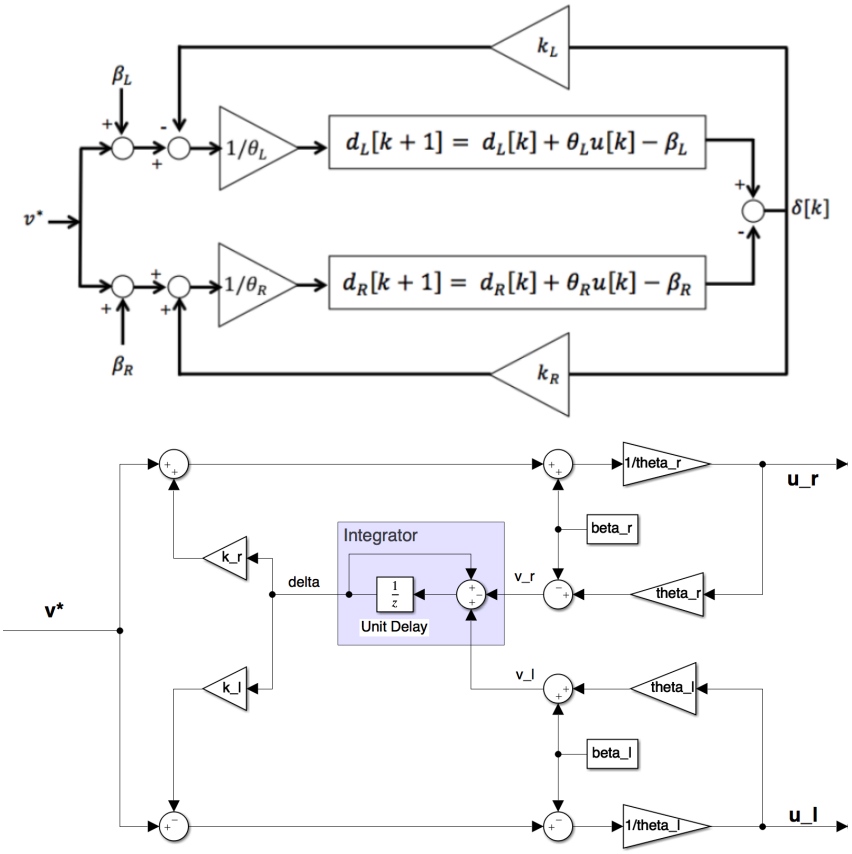
Setting the open-loop equation for v equal to our desired equation for v, we solve for the inputs u_L and u_R:

$$\begin{aligned} u_L[k] &= \frac{v^* + \beta_L}{\theta_L} - k_L\frac{\delta[k]}{\theta_L} \\ u_R[k] &= \frac{v^* + \beta_R}{\theta_R} + k_R\frac{\delta[k]}{\theta_R} \end{aligned}$$

Now, we plug these i nputs into our original equations for v_L and v_R to complete the model:

$$\begin{aligned} v_L[k] &= d_L[k+1] - d_L[k] = \theta_L\left(\frac{v^* + \beta_L}{\theta_L} - k_L\frac{\delta[k]}{\theta_L}\right) - \beta_L \\ v_R[k] &= d_R[k+1] - d_R[k] = \theta_R\left(\frac{v^* + \beta_R}{\theta_R} + k_R\frac{\delta[k]}{\theta_R}\right) - \beta_R \end{aligned}$$

Below are two block diagrams representing our control scheme. The first is the one you saw in lab: in this one, we abstract away the calculation of $u_L[k]$ and $u_R[k]$ and directly plug them into our plants, $d_L[k+1] - d_L[k] = \theta_L u[k] - \beta_L$ and $d_L[k+1] - d_L[k] = \theta_L u[k] - \beta_L$. The second shows how $u_L[k]$ and $u_R[k]$ are calculated.



$u_L[k]$ and $u_R[k]$ are represented as outputs because they are essentially the outputs of the system you implemented in your code: all the calculations you go through at each timestep lead up to the calculation of $u_L[k]$ and $u_R[k]$, which are then input into the **physical** system via the actuators that generate the PWM signal and make the wheels turn (i.e., the Launchpad's signal pins and the motors/motor controller circuitry). Do not be alarmed by the integrator block: it merely accumulates the velocities to determine distance, and calculates $\delta[k]$ from the differences of the distances. You do not have a direct representation of this integration in your code because the Launchpad stores your distance traveled, but we need it in the block diagram representation because your chosen $v*$, the primary input to the system, is a *velocity*, and is independent of total distance traveled: the block diagram is a direct representation of the system created by setting the equations in (1) equal to their respective equations in (2). Delay blocks in discrete-time systems act like memory overwritten at each timestep, and we can use them to create discrete-time integrators, representing persistent memory, like the one shown in the second system above. Keep in mind, $v_L[k]$ and $v_R[k]$ are the *modeled* $v_L[k]$ and $v_R[k]$: according to the equations of our model, this is how $v_L[k]$ and $v_R[k]$ should evolve, and we do not attempt to mathematically model the full complexity of the physical actuator system but instead account for its influence with θ and β.

a) Let's examine the feedback proportions k_L and k_R more closely. Should they be positive or negative? What do they mean? Think about how they interact with $\delta[k]$.

b) Let's look a bit more closely at picking k_L and k_R. First, we need to figure out what happens to $\delta[k]$ over time. Find $\delta[k+1]$ in terms of $\delta[k]$.

c) Given your work above, what is the eigenvalue of the system defined by $\delta[k]$? For discrete-time systems like our system, $\lambda \in (-1, 1)$ is considered stable. Are $\lambda \in [0, 1)$

and $\lambda \in (-1, 0]$ identical in function for our system? Which one is "better"? (*Hint:* Preventing oscillation is a desired benefit.)

Based on your choice for the range of λ above, how should we set k_L and k_R in the end?

d) Let's re-introduce the model mismatch from last week in order to model environmental discrepancies, disturbances, etc. How does closed-loop control fare under model mismatch? Find $\delta_{ss} = \delta[k \to \infty]$, assuming that $\delta[0] = \delta_0$. What is δ_{ss}? (To make this easier, you may leave your answer in terms of appropriately defined c and λ obtained from an equation in the form of $\delta[k+1] = \delta[k]\lambda + c$.)

Check your work by verifying that you reproduce the equation in part (c) if all model mismatch terms are zero. Is it better than the open-loop model mismatch case from last week?

$$v_L[k] = d_L[k+1] - d_L[k] = (\theta_L + \Delta\theta_L)u_L[k] - (\beta_L + \Delta\beta_L)$$
$$v_R[k] = d_R[k+1] - d_R[k] = (\theta_R + \Delta\theta_R)u_R[k] - (\beta_R + \Delta\beta_R)$$

$$u_L[k] = \frac{v^* + \beta_L}{\theta_L} - k_L\frac{\delta[k]}{\theta_L}$$
$$u_R[k] = \frac{v^* + \beta_R}{\theta_R} + k_R\frac{\delta[k]}{\theta_R}$$

2 DFT properties

a) Suppose the DFT of a length-N sequence $x[n]$, $n = 0, 1, \ldots, N-1$, is given by $X[k]$, $k = 0, 1, \ldots, N-1$ and a new sequence is defined as

$$y[n] = e^{jm\frac{2\pi}{N}n} x[n].$$

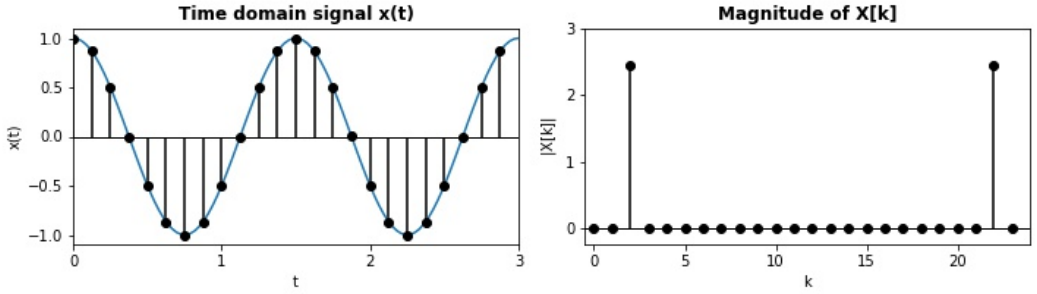
Show that

$$Y[k] = X[(k-m) \mod N]$$

b) Let $N = 4$ and $y[n] = (-1)^n x[n]$, $n = 0, 1, 2, 3$. Write $Y[0]$, $Y[1]$, $Y[2]$, $Y[3]$ in terms of $X[0]$, $X[1]$, $X[2]$, $X[3]$.

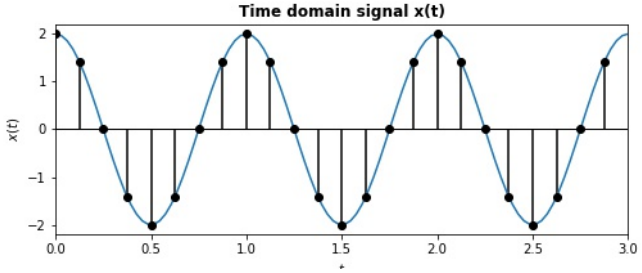
3 DFT Sampling Matching

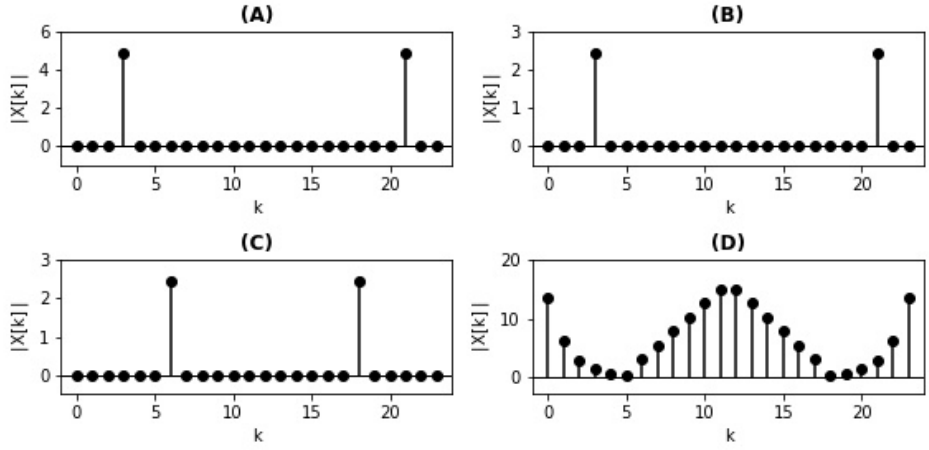
a) A sampled sinusoidal time domain signal and its DFT coefficients are given below:



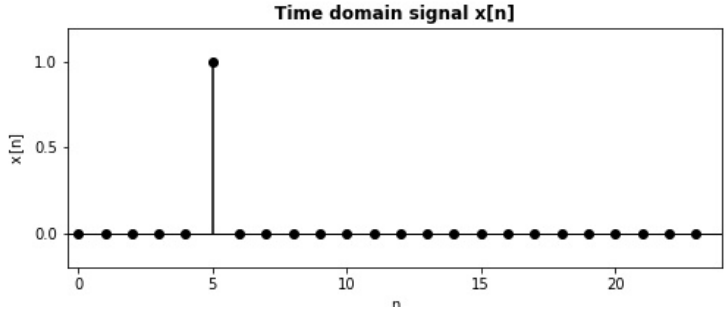
Remember that we saw in discussion that the DFT of a pure harmonic of the form $x \mapsto A_l \cos(2\pi l f_0 + \theta_l)$ has a very particular structure, and that its non-zero coefficients are related to A_l and θ_l.

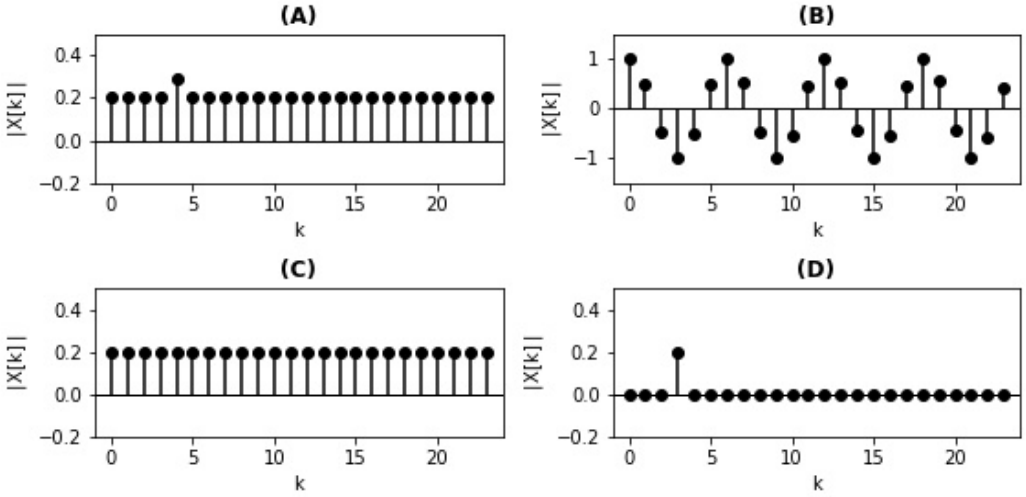
Now given the following sinusoidal time domain signal, **which of the options below shows the correct DFT coefficient magnitudes?**



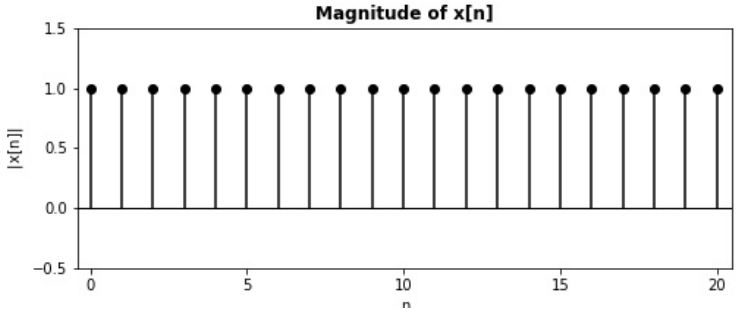


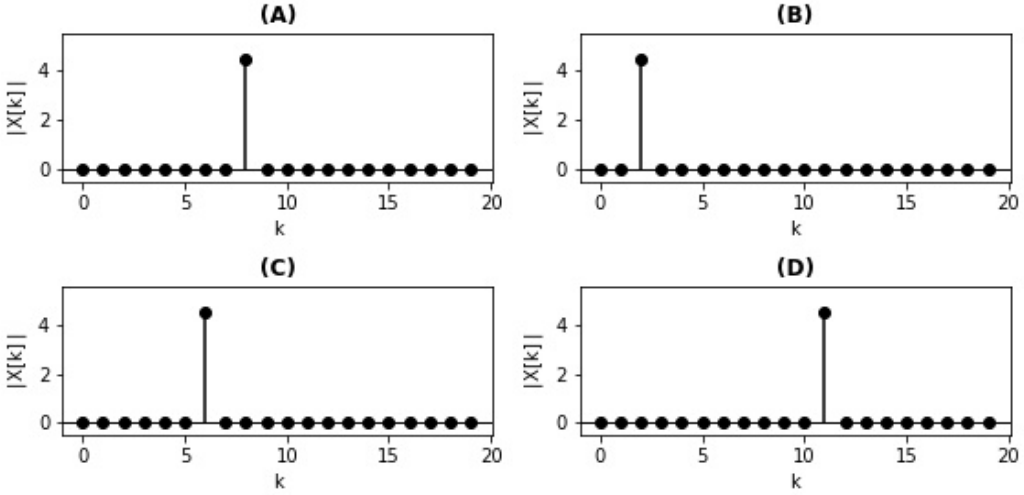
b) Given the time domain signal below, **which of the options below shows the correct DFT coefficient magnitudes?**



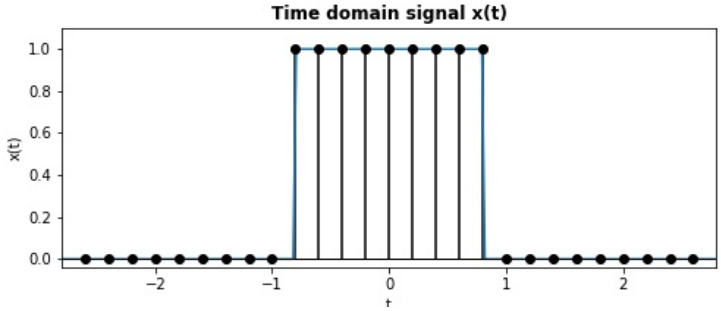


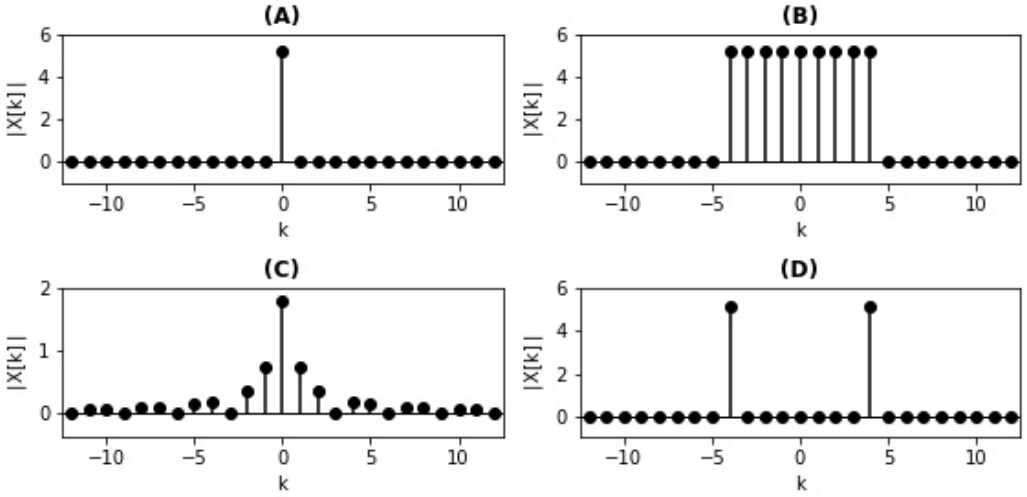
c) Given the magnitude of the (complex) time domain signal below, **which of the options below shows the correct DFT coefficient magnitudes?** *Hint: this is a trick question!*





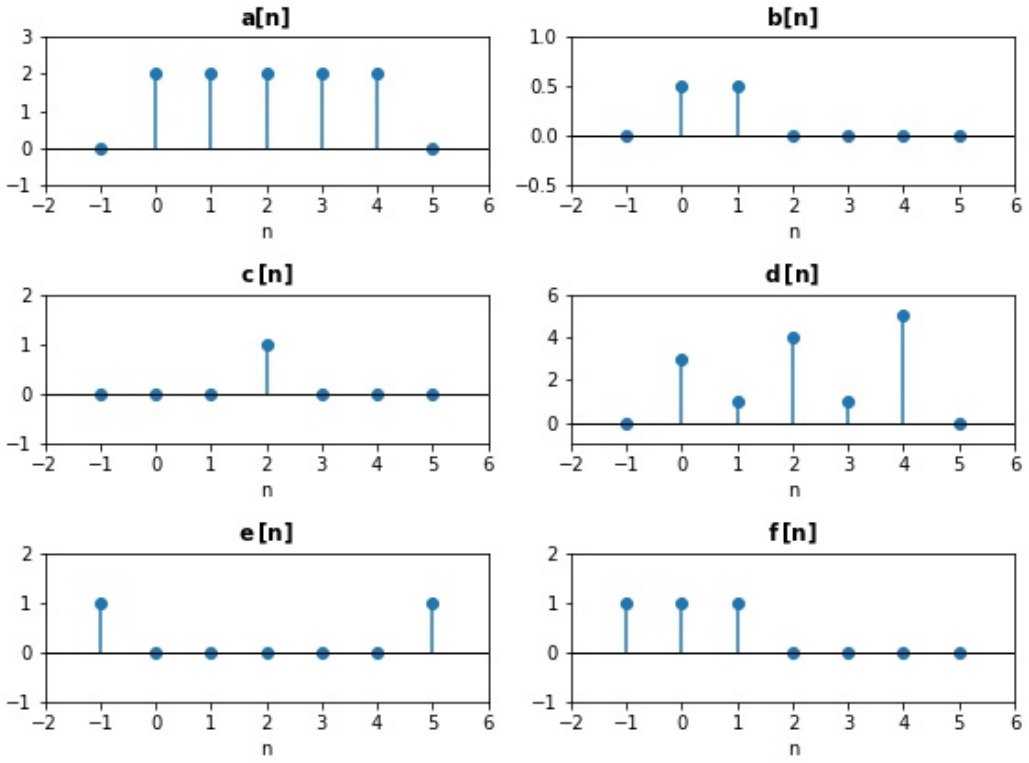
d) Given the time domain signal below, **which of the options below shows the correct DFT coefficient magnitudes?**



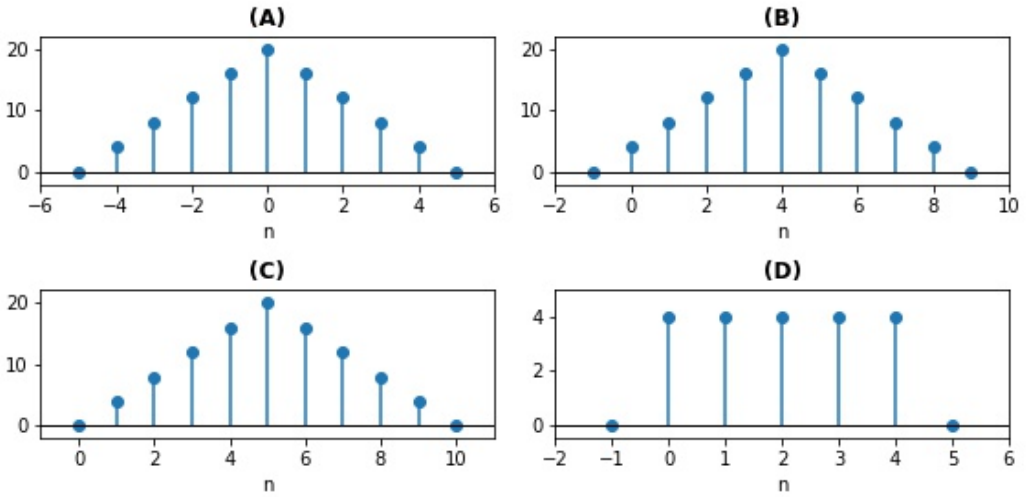


4 Convolution Matching

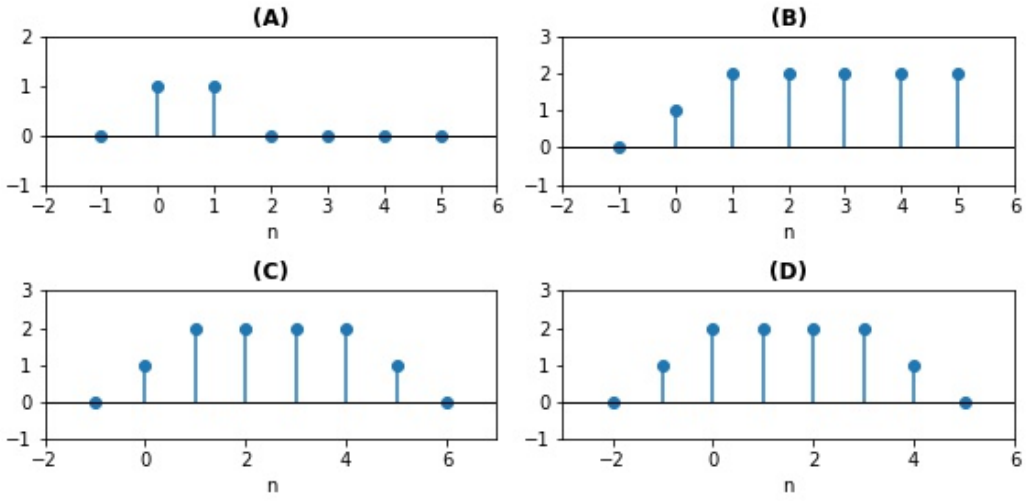
Consider the following discrete time signals:



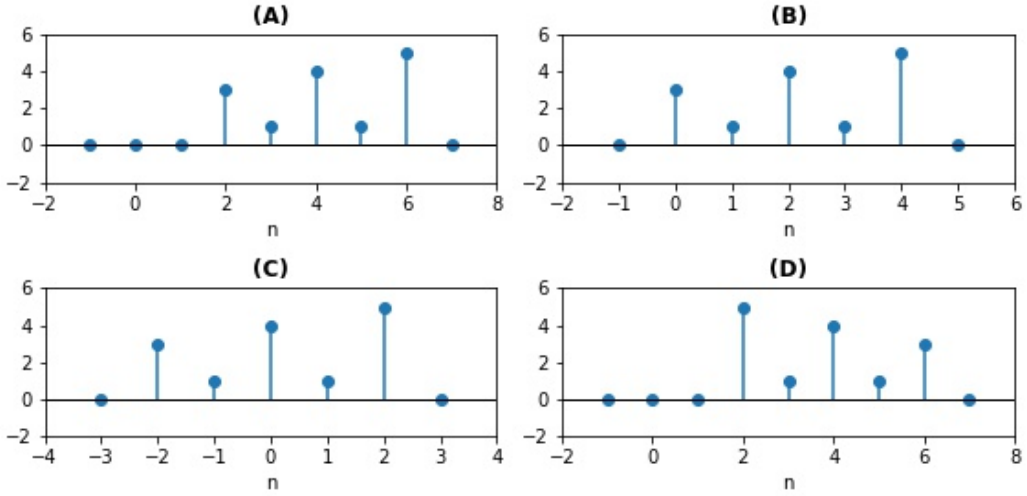
a) Which of the options below shows the correct plot for the convolution $a[n] * a[n]$?



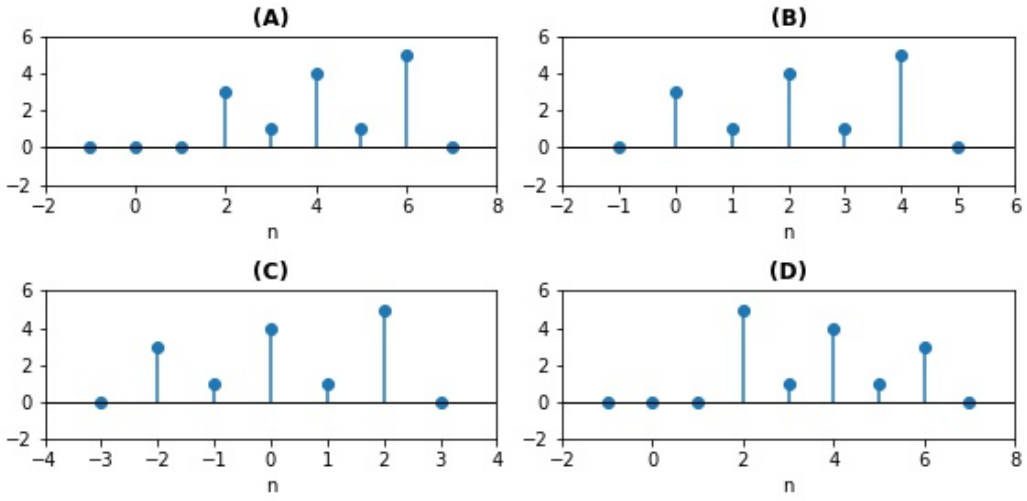
b) Which of the options below shows the correct plot for the convolution $a[n] * b[n]$?



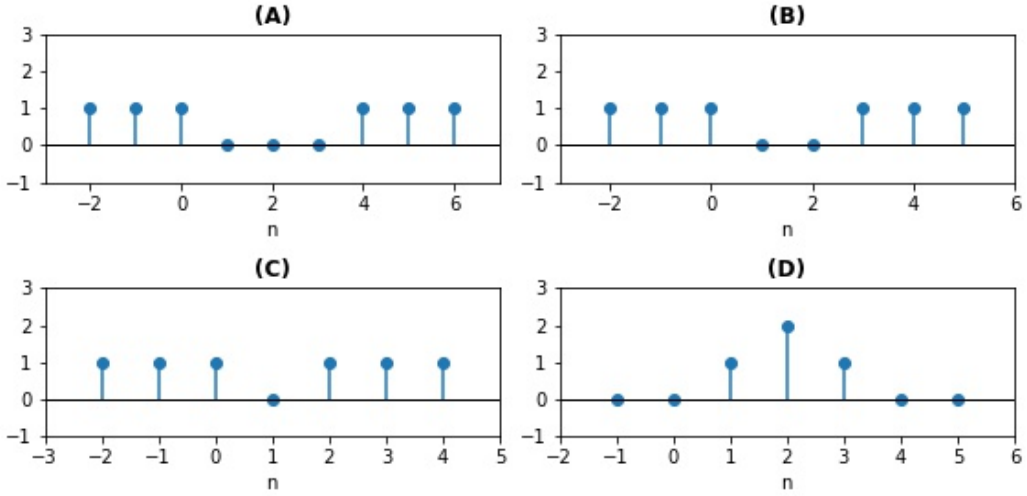
c) Which of the options below shows the correct plot for the convolution $c[n] * d[n]$?



d) Which of the options below shows the correct plot for the convolution $d[n] * c[n]$?



e) Which of the options below shows the correct plot for the convolution $e[n] * f[n]$?



5 Denoising signals using the DFT

Simon is sad. He just managed to create a beautiful audio clip consisting of a couple pure tones with beats and he wants Emily to listen to it. He calls Emily on a noisy phone and plays the message through the phone. Emily then tells him that the audio is very noisy and that he is unable to truly appreciate the music. Unfortunately, Simon has no other means of letting Emily listen to the message. Luckily, they have you! You propose to implement a denoiser at Emily's end.

a) In the IPython notebook, listen to the noisy message. Plot the time signal and comment on visible structure, if any.

b) Take the DFT of the signal and plot the magnitude. In a few sentences, describe what the spikes you see in the spectrum are.

c) There is a simple method to denoise this signal: Simply threshold in the DFT domain! Threshold the DFT spectrum by keeping the coefficients whose absolute values lie above a certain value. Then take the inverse DFT and listen to the audio. You will be given a range of possible values to test. Write the threshold value you think works best.

Yay, Simon is no longer sad!

6 Amplitude modulation (AM)

In electronic communication, transmission is achieved by varying *some aspect* of a *higher frequency signal*, ("carrier signal)", with some information-bearing waveform ("base signal") to be sent, such as an audio or video signal, a process known as **"modulation"**. Amplitude modulation is a specific scheme of modulation, where the amplitude of the carrier oscillations is varied. In this question, you will be led through the mixing of some simple base and carrier signals so you know the magic behind your traditional AM radio.

Recap of DFT: Consider a discrete time signal $x[n]$. We can represent it as a vector of discrete samples over time $\vec{x}$, of length N.

$$\vec{x} = \begin{bmatrix} x[0] & \dots & x[N-1] \end{bmatrix}^T \tag{1}$$

Let $\vec{X} = \begin{bmatrix} X[0] & \dots & X[N-1] \end{bmatrix}^T$ be the signal $\vec{x}$ represented in the frequency domain

$$\vec{X} = U^{-1}\vec{x} = U^*\vec{x} \tag{2}$$

where U is a matrix of the DFT basis vectors.

$$U = \begin{bmatrix} | & & | \\ \vec{u}_0 & \cdots & \vec{u}_{N-1} \\ | & & | \end{bmatrix} \tag{3}$$

Recall that the DFT basis vectors $\vec{u}_k$ are complex exponentials that have a cyclic property

$$\vec{u}_k = \vec{u}_{k \bmod N} = \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}k \cdot n} \text{ for } n = 0, 1, \dots, N-1. \tag{4}$$

Lastly, we can represent $\vec{x} = U\vec{X}$ or more explicitly

$$\vec{x} = X[0]\vec{u}_0 + \cdots + X[N-1]\vec{u}_{N-1} \tag{5}$$

In other words, $\vec{x}$ *can be always represented as* a linear combination of the DFT basis signals $\vec{u}_k$ with coefficients $X[k]$.

a) Let N be the number of samples in a vector. Suppose we have a complex exponential signal $s[n] = e^{j\frac{2\pi K}{N}n}$ with frequency $\frac{K}{N}$ for some integer K. In vector form, we can represent it as

$$e^{j\frac{2\pi K}{N}n} \Rightarrow \begin{bmatrix} 1 & e^{j\frac{2\pi K}{N}} & e^{j\frac{2\pi K}{N}\cdot 2} & \cdots & e^{j\frac{2\pi K}{N}(N-1)} \end{bmatrix}^T = \sqrt{N}\vec{u}_K \tag{6}$$

Mixing two signals is defined to be *the process of element-wise multiplying them together*, i.e. computing $e^{j\frac{2\pi K}{N}n}x[n]$. This amounts to *element-wise multiplication* $\odot$:

$$e^{j\frac{2\pi K}{N}n}x[n] = (\sqrt{N}\vec{u}_K \odot \vec{x})[n] \tag{7}$$

The idea is, instead of transmitting the signal $\vec{x}$ directly, we will transmit the mixed signal.

Let's start simple. Suppose our signal $x[n]$ is a simple complex exponential with frequency $\frac{R}{N}$, that is $\vec{x} = \vec{u}_R$.

What is $\vec{y} = \sqrt{N}\vec{u}_K \odot \vec{x}$? Then represent $\vec{y}$ in the frequency domain. What does mixing with a complex exponential do to the frequency of the signal? You may assume that $K + R < N$.

b) Now suppose that $\vec{x}$ is a cosine signal, that is $x[n] = \cos\left(\frac{2\pi R}{N} n\right)$. Mix this new $\vec{x}$ with the complex exponential $\vec{s} = \sqrt{N}\vec{u}_K$. What is the frequency domain representation of this new signal and how does it differ from $\vec{x}$? You may assume that $R < K$.

c) With the above $x[n] = \cos\left(\frac{2\pi R}{N} n\right)$, let's go one step further by mixing it with a carrier signal $\vec{s}$, where $s[n] = \cos\left(\frac{2\pi K}{N} n\right)$.

This is practically important because complex-exponentials need to be realized in a physical way that is real to build real-world AM radios.

What is the frequency domain representation of this new signal and how does it differ from $\vec{x}$? Again assume that K is much bigger than R and in-turn N is much bigger than K.

d) Now that you see the effect of mixing the base signal with a cosine wave as the carrier signal. It is natural to wonder how can we undo this process.

The technique that is used is called **de-modulation**. Now, mix $\cos\left(\frac{2\pi K}{N} n\right)$ with the signal you get from the previous question. How many resulting terms do you get in the frequency domain? Considering the fact that $K \gg R$, what are the frequencies relative positions on the frequency spectrum? Have we recovered the original signal?

e) Refer to the iPython notebook for the implementations. Explain how we can recover the original signal and comment on what you see in the plots.

7 LTI filter

Suppose we apply the length $L = 100$ input

$$x[n] = \cos(0.1\pi n) + \cos(0.4\pi n), \quad n = 0, 1, \ldots, 99 \tag{8}$$

to a finite impulse response filter whose impulse response is

$$h[n] = \begin{cases} \frac{1}{6} & n = 0, \ldots, 5 \\ 0 & \text{otherwise.} \end{cases}$$

We want to find the output $y[n],\ n = 0, \ldots, 104$ using the DFT.

a) Find the 105-point DFT of $x[n]$ by adding 5 zeros to the length-100 signal given above and using the DFT command `numpy.fft.fft(x, norm="ortho")`. Plot the magnitude $|X[k]|$ against the frequency variable $\omega \in [0, 2\pi]$. (Recall that the integer k corresponds to the frequency $\frac{2\pi}{105}k$.)

b) Find the 105-point DFT of $h[n]$ by adding 99 zeros to the length-6 impulse response given above. Plot the magnitude $|H[k]|$ against the frequency variable $\omega \in [0, 2\pi]$. Given this frequency response how do you think each frequency component of $x[n]$ will be affected when the filter is applied?

c) As will be shown in lecture, the DFT coefficients of the output are

$$Y[k] = \sqrt{105}H[k] \odot X[k],\ k = 0, \ldots 104.$$

Find $y[n]$ using the inverse DFT command `numpy.fft.ifft(Y, norm="ortho")` Plot both $x[n]$ and $y[n]$ versus time n and explain how the filter modified $x[n]$.

8 Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student! We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

a) **What sources (if any) did you use as you worked through the homework?**

b) **If you worked with someone on this homework, who did you work with?** List names and student ID's. (In case of homework party, you can also just describe the group.)

c) **How did you work on this homework?** (For example, *I first worked by myself for 2 hours, but got stuck on problem 3, so I went to office hours. Then I went to homework party for a few hours, where I finished the homework.*)

d) **Do you have any feedback on this homework assignment?**

e) **Roughly how many total hours did you work on this homework?**