EECS 16A Designing Information Devices and Systems I
Spring 2018 Homework 11

This homework is due April 18, 2018, at 23:59.
Self-grades are due April 21, 2018, at 23:59.

Submission Format
Your homework submission should consist of **two** files.

- `hw11.pdf`: A single PDF file that contains all of your answers (any handwritten answers should be scanned) as well as your IPython notebook saved as a PDF.

  If you do not attach a PDF of your IPython notebook, you will not receive credit for problems that involve coding. Make sure that your results and your plots are visible.

- `hw11.ipynb`: A single IPython notebook with all of your code in it.

  In order to receive credit for your IPython notebook, you must submit both a "printout" and the code itself.

Submit each file to its respective assignment on Gradescope.

1. Cauchy-Schwarz Inequality

The Cauchy-Schwarz inequality states that for two vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$:

$$|\langle \vec{x}, \vec{y} \rangle| = |\vec{x}^T \vec{y}| \leq \| \vec{x} \| \cdot \| \vec{y} \|$$

Use the Cauchy-Schwarz inequality to verify (i.e. prove or derive) the triangle inequality:

$$\| \vec{x} + \vec{y} \| \leq \| \vec{x} \| + \| \vec{y} \|$$

Hint: Start with $\| \vec{x} + \vec{y} \|^2$.

Solution:

We consider the Euclidean 2-norm here. By Cauchy-Schwarz inequality, we have

$$|\langle \vec{x}, \vec{y} \rangle| \leq \|\vec{x}\|_2 \cdot \|\vec{y}\|_2$$
$$|\langle \vec{y}, \vec{x} \rangle| \leq \|\vec{y}\|_2 \cdot \|\vec{x}\|_2$$

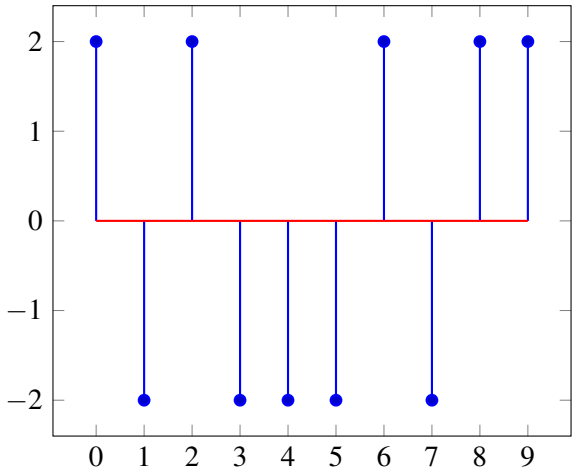
Therefore,

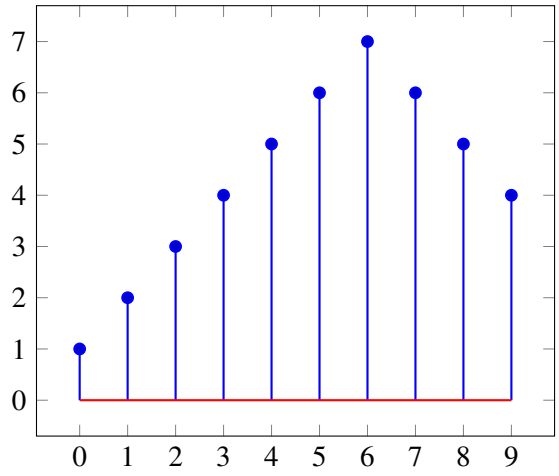
$$\begin{aligned} \|\vec{x} + \vec{y}\|_2^2 &= (\vec{x} + \vec{y})^T (\vec{x} + \vec{y}) \\ &= \|\vec{x}\|_2^2 + \|\vec{y}\|_2^2 + \langle \vec{x}, \vec{y} \rangle + \langle \vec{y}, \vec{x} \rangle \\ &\leq \|\vec{x}\|_2^2 + \|\vec{y}\|_2^2 + 2\|\vec{y}\|_2 \cdot \|\vec{x}\|_2 \\ &= (\|\vec{x}\|_2 + \|\vec{y}\|_2)^2 \end{aligned}$$

Taking square root on both sides, we get

$$\|\vec{x} + \vec{y}\|_2 \leq \|\vec{x}\|_2 + \|\vec{y}\|_2$$

2. Mechanical Correlation

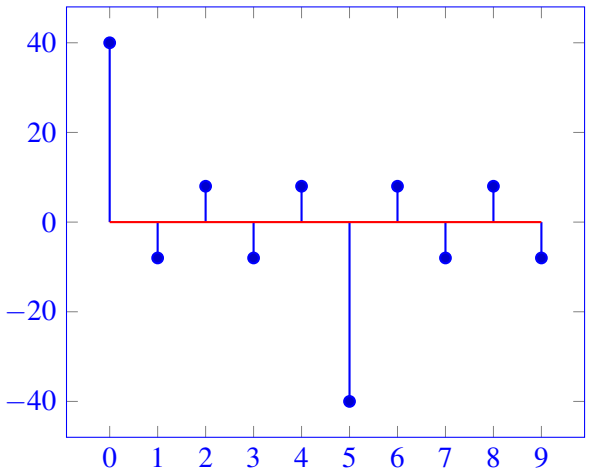




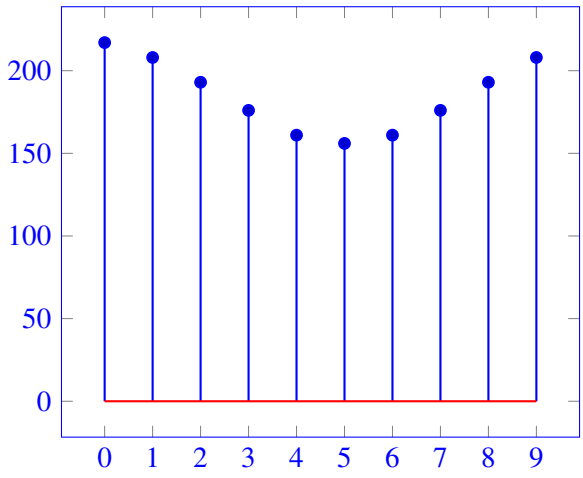
(a) Calculate and plot the **autocorrelation** (the inner products of one period of the signal with all the possible shifts of one period of the same signal) of each of the above signals. Each signal is periodic with a period of 10 (one period is shown). You may use iPython for this question, please use the notebook provided to you for all work.

Solution:

$$\text{Autocorrelation of } \vec{s}_1: \begin{bmatrix} 40 \\ -8 \\ 8 \\ -8 \\ 8 \\ -40 \\ 8 \\ -8 \\ 8 \\ -8 \end{bmatrix}$$



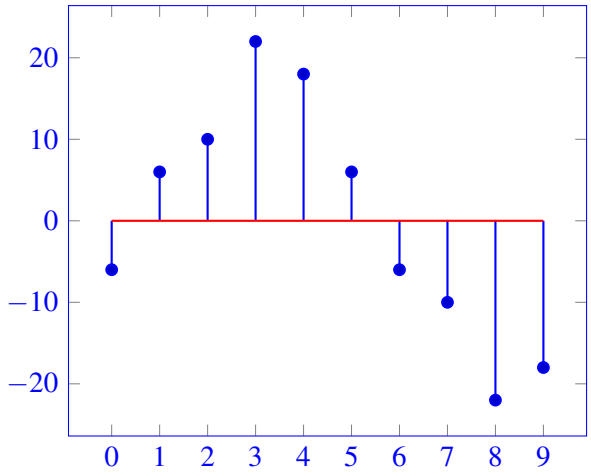
$$\text{Autocorrelation of } \vec{s}_2: \begin{bmatrix} 217 \\ 208 \\ 193 \\ 176 \\ 161 \\ 156 \\ 161 \\ 176 \\ 193 \\ 208 \end{bmatrix}$$



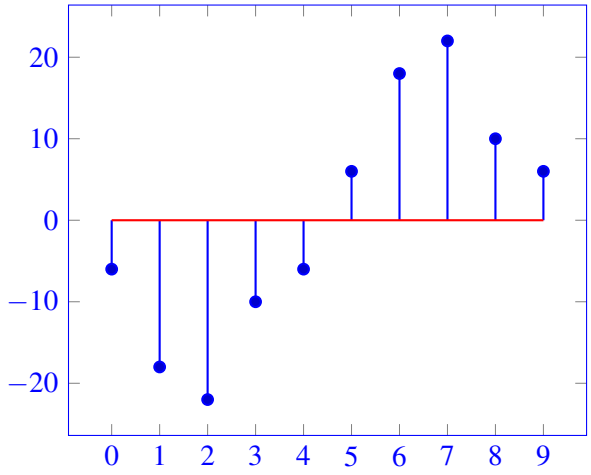
(b) Calculate and plot the **cross-correlation** (the inner products of one period of the first signal with all possible shifts of one period of the second signal) of the two signals. Each signal is periodic with a period of 10 (one period is shown).

Solution:

$$\vec{s}_1 \text{ with } \vec{s}_2: \begin{bmatrix} -6 \\ 6 \\ 10 \\ 22 \\ 18 \\ 6 \\ -6 \\ -10 \\ -22 \\ -18 \end{bmatrix}$$



$\vec{s}_2$ with $\vec{s}_1$: $\begin{bmatrix} -6 \\ -18 \\ -22 \\ -10 \\ -6 \\ 6 \\ 18 \\ 22 \\ 10 \\ 6 \end{bmatrix}$



3. Audio File Matching

Lots of different quantities we interact with every day can be expressed as vectors. For example, an audio clip can be thought of as a vector. The series of numbers in the clip determines the sounds we hear. An audio segment or a sound wave is a continuous function of time, but this can be sampled at regular intervals to make a discrete sequence of numbers that can be represented as a vector.

This problem explores using inner products to measure similarity. The ideas here are similar to the themes of the Acoustic Positioning System in the lab.

Let us consider a very simplified model for an audio signal, one that is just composed of two tones. One is represented by -1 and the other by $+1$. A vector of length N makes up the audio file.

(a) Say we want to compare two audio files of the same length N to decide how similar they are. First, consider two vectors that are exactly identical $\vec{x}_1 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ and $\vec{x}_2 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$. What is the dot product of these two vectors? What if $\vec{x}_1 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ and $\vec{x}_2 = \begin{bmatrix}1 & -1 & 1 & -1 & \cdots & 1 & -1\end{bmatrix}^T$ (where the length of the vector is an even number)? Can you come up with an idea to compare two general vectors of length N now?

Solution:

The dot product $\vec{u} \cdot \vec{v} = \sum_{i=1}^{N} u_i v_i$ is the same thing as the Euclidean Inner Product. The dot product of $\vec{x}_1 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ and $\vec{x}_2 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ is $\vec{x}_1 \cdot \vec{x}_2 = N$. The dot product of $\vec{x}_1 = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ and $\vec{x}_2 = \begin{bmatrix}1 & -1 & 1 & -1 & \cdots & 1 & -1\end{bmatrix}^T$ is $\vec{x}_1 \cdot \vec{x}_2 = 0$ when the vector length is even. To compare two vectors of length N composed of 1 and -1, we first take the dot product of the two vectors. The larger the magnitude of the dot product, the more similar the two vectors are. The smaller the magnitude of the dot product, the more dissimilar the two vectors are. This is related to the ideas of correlation that you will see later on. In many circumstances, a dot product with a very large negative value would mean the vectors are very different, but it turns out that humans are unable to perceive the sign of sound, so two sounds vectors $\vec{x}$ and $-\vec{x}$ sound exactly the same. As a result, for this problem we are interested in is the **absolute value** of the dot product, but in many other problems, we will interpret a large negative dot product as very different vectors. Don't take off points in parts (a), (b), or (c) if you didn't mention the absolute value.

(b) Next suppose we want to find a short audio clip in a longer one. We might want to do this for an application like *Shazam*, which is able to identify a song from a signature tune. Consider the vector of length 8, $\vec{x} = \begin{bmatrix}-1 & 1 & 1 & -1 & 1 & 1 & -1 & 1\end{bmatrix}^T$. Let us label the elements of $\vec{x}$ so that $\vec{x} = \begin{bmatrix}x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8\end{bmatrix}^T$. We want to find the short segment $\vec{y} = \begin{bmatrix}1 & 1 & -1\end{bmatrix}^T$ in the longer vector, i.e. we want to find i, such that the sequence represented by $\begin{bmatrix}x_i & x_{i+1} & x_{i+2}\end{bmatrix}^T$ is the closest to $\vec{y}$. How can we find this? Applying the same technique, what i gives the best match for $\vec{y} = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$?

Solution:

For each sub-sequence of length 3 of $\vec{x}$, say $\vec{x}_i = \begin{bmatrix}x_i & x_{i+1} & x_{i+2}\end{bmatrix}^T$ (the sub-sequence which starts at position i), take the dot product of $\vec{x}_i$ and $\vec{y}$. The sub-sequences $\vec{x}_i$ with the largest dot product magnitude with $\vec{y}$ gives us the closest match with $\vec{y}$. For $\vec{y} = \begin{bmatrix}1 & 1 & -1\end{bmatrix}^T$, the list of i with maximum dot product between $\vec{x}_i$ and $\vec{y}$ are $\{2,5\}$ with each dot product being 3. For $\vec{y} = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$, every dot product is 1, so there is no good match.

We can also implement the 6 dot products in one matrix multiplication. For instance, if $\vec{x} = \begin{bmatrix}x_1 & x_2 & \ldots & x_8\end{bmatrix}^T$ and $\vec{y} = \begin{bmatrix}y_1 & y_2 & y_3\end{bmatrix}^T$, the dot products (or correlation) $z_i = \vec{x}_i \cdot \vec{y}$ can be represented as:

$$\begin{bmatrix} y_1 & y_2 & y_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & y_1 & y_2 & y_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & y_1 & y_2 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & y_1 & y_2 & y_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & y_1 & y_2 & y_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix}$$

We then pick the z_i with the largest magnitude (there can be multiple as we've just seen) and the corresponding $\vec{x}_i$ gives us the required substrings.

(c) Now suppose our vector is represented using integers and not just by 1 and -1. Say we want to locate the sequence closest to $\vec{y} = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$ in $\vec{x} = \begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{bmatrix}^T$. What happens if you apply the technique of part (b)? How would you modify this technique for the problem here?

Solution:

Applying the technique in part (b), we get the best match to be $\begin{bmatrix}6 & 7 & 8\end{bmatrix}^T$ as this has the largest dot product with $\vec{y} = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$.

One way to modify the previous approach is by considering how close the "directions" of $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$ and any sub-sequence of length 3 of $\vec{x}$ are. The unit vector in the direction of $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$ is $\vec{y}_u = \frac{1}{\sqrt{14}}\begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$, and the unit vector in the direction of any sub-sequence of length 3 $\vec{x}_i = \begin{bmatrix}x_i & x_{i+1} & x_{i+2}\end{bmatrix}^T$ is then given by $\vec{u}_i = \frac{1}{\|\vec{x}_i\|}\begin{bmatrix}x_i & x_{i+1} & x_{i+2}\end{bmatrix}^T$. The unit vector $\vec{u}_i$, which has the largest dot product magnitude with $\vec{y}_u$ (i.e. $\vec{u}_i \cdot \vec{y}_u$), is the one most aligned with the vector $\vec{y}$. Thus, in our example for $\vec{x} = \begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{bmatrix}^T$ and $\vec{y} = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$, the unit vector of the sub-sequence $\vec{x}_1 = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$ has the largest dot product ($\vec{u}_1 \cdot \vec{y}_u = 1$).

What are other interesting ways of achieving this?

(d) Answer part (d) in the provided IPython notebook.

Solution:

See `sol10.ipynb`.

(e) Answer part (e) in the provided IPython notebook.

Solution:

See `sol10.ipynb`.

4. Finding Signals in Noise

Disclaimer: This problem looks long. However, almost all of the parts involve only running the provided IPython code and commenting on its output.

In this problem, we will explore how to use correlation and least squares to find signals, even in the presence of noise and other interfering signals.

(a) Suppose that there is a transmitter sending a known signal $\vec{s}_1$, which is periodic with length $N = 1000$. Say $\vec{s}_1$ is chosen to be a random $\{+1, -1\}$ vector (for example, by tossing a coin N times and replacing every heads with $+1$ and every tails with -1). For convenience, let us also normalize $\vec{s}_1$, so that $\vec{s}_1$ has a norm of 1. That is, the vector $\vec{s}_1$ looks like:

$$\vec{s}_1 = \frac{1}{\sqrt{n}}\begin{bmatrix}+1 & -1 & -1 & +1 & -1 & \cdots\end{bmatrix}^T,$$

where the ± 1 entries are chosen randomly.

We claim that such a vector $\vec{s}_1$ is "approximately orthogonal" to circular shifts of itself. That is, if $\vec{s}_1^{(j)}$ denotes circularly shifting $\vec{s}_1$ by j, then for all $j \neq 0$:

$$\left|\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle\right| \lessapprox \varepsilon$$

for some small epsilon.

Run the provided IPython code to generate a random $\vec{s}_1$ and plot its autocorrelation (all inner products with shifted versions of itself). Run this a few times, for different random vectors $\vec{s}_1$. Around how small are the largest inner products $\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle$, $j \neq 0$?

Recall that we normalized $\vec{s}_1$, so $\langle \vec{s}_1, \vec{s}_1 \rangle = 1$.

Solution:

After running the provided code a few times, you should have observed that the individual inner products $\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle, j \neq 0$ are usually less than 0.06 and that the maximum magnitude of the inner product $\left| \left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle \right|$ for all $j \neq 0$ is usually around 0.1. (You could have done this by looking at the maximum of the autocorrelation plot at indices $j \neq 0$.) Therefore, we can let

$$\varepsilon = 0.1$$

Let us also define

$$\varepsilon_2 = 0.06$$

as a bound on the deviation of an individual inner product. (It is fine to just use ε as a bound instead of ε_2, but we will carry ε_2 through to be slightly more precise.)

Note that $\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle$ for $j \neq 0$ is much smaller than $\langle \vec{s}_1, \vec{s}_1 \rangle = 1$.

(b) Suppose we receive a signal $\vec{y}$, which is $\vec{s}_1$ delayed by an unknown amount. That is,

$$\vec{y} = \vec{s}_1^{(j)}$$

for some unknown shift j. To find the delay, we can choose the shift j with the largest inner product $\left\langle \vec{s}_1^{(j)}, \vec{y} \right\rangle$.

Run the provided IPython code to plot the cross-correlation between $\vec{y}$ and $\vec{s}_1$ (i.e. all the shifted inner products). Can you identify the delay? Briefly comment on why this works using the findings from the previous part.

Hint: What does $\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle$ look like, for $k = j$? For $k \neq j$?

Solution:

Running the provided code, we see that the autocorrelation peaks at index $k = 10$. That is, the inner product $\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle$ is maximum for the shift $k = 10$. Thus, we identify the delay as $j = 10$.

This works because the inner product $\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle = \left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} \right\rangle$ will usually be small (at most ε) if $k \neq j$ (from the previous part) but will be exactly 1 if $k = j$. Thus, the maximum index k will usually identify the correct shift.

Note: We used the word "usually" here as a technicality since it is possible, though unlikely, that our random choice of $\vec{s}_1$ happens to have high correlation with its shifted versions. For example, it is possible that our "random" choice resulted in $\vec{s}_1 = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 \cdots & 1 \end{bmatrix}$, i.e. the normalized all-ones vector. However, this is unlikely to occur – random vectors are usually almost orthogonal to their shifted selfs, as you saw in the previous part. We will omit the "usually" for the rest of the solutions for the sake of readability.

(c) Now suppose that we receive a slightly noisy signal:

$$\vec{y} = \vec{s}_1^{(j)} + 0.1\vec{n},$$

where the "noise" source $\vec{n}$ is chosen to be a random normalized vector, just like $\vec{s}_1$.

Run the provided IPython code to compute $\langle \vec{s}_1, \vec{n} \rangle$. Run this a few times for different random choices of $\vec{s}_1, n$. Around how small is $|\langle \vec{s}_1, \vec{n} \rangle|$? How does this compare to $\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle$ from your answer in part (a)?

Solution:

After running the provided code a few times, you should have observed that $|\langle \vec{s}_1, \vec{n} \rangle|$ is usually around 0.03 and almost always less than $0.06 = \varepsilon_2$.

This is similar to how the inner products $\left\langle \vec{s}_1, \vec{s}_1^{(j)} \right\rangle, j \neq 0$ looked in the previous parts. That is, the inner product of a random signal with its shifted self is roughly the same as the inner product of a signal with random noise.

(d) Can we identify the delay from this noisy reception? In this case, we do not know the noise $\vec{n}$, and we do not know the delay j. (But, as before, we know that the signal $\vec{s}_1$ being transmitted).

Run the provided IPython code to plot the cross-correlation between $\vec{y}$ and $\vec{s}_1$. Briefly comment on why this works to find the delay using the findings from the previous part.

Solution:

Yes, we can identify the delay in this case, as demonstrated by the provided code.

This works roughly because, for a fixed $k \neq j$, the inner product

$$\begin{aligned}\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle &= \left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} + 0.1\vec{n} \right\rangle \\ &= \left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} \right\rangle + (0.1)\left\langle \vec{s}_1^{(k)}, \vec{n} \right\rangle \\ &\approx \varepsilon_2 \pm (0.1)\varepsilon_2\end{aligned}$$

is small. In fact, *all* the inner products for $k \neq j$ are small; the maximum over all $k \neq j$ is roughly bounded by:

$$\begin{aligned}\max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle &= \max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} + 0.1\vec{n} \right\rangle \\ &= \max_{k \neq j} \left(\left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} \right\rangle + (0.1)\left\langle \vec{s}_1^{(k)}, \vec{n} \right\rangle \right) \\ &\leq \max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{s}_1^{(j)} \right\rangle + (0.1) \max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{n} \right\rangle \\ &\lessapprox \varepsilon + (0.1)\varepsilon,\end{aligned}$$

where we used our findings from the previous parts in the last step.

On the other hand, for $k = j$, the inner product

$$\begin{aligned}\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle &= \left\langle \vec{s}_1^{(j)}, \vec{s}_1^{(j)} + 0.1\vec{n} \right\rangle \\ &= \left\langle \vec{s}_1^{(j)}, \vec{s}_1^{(j)} \right\rangle + (0.1)\left\langle \vec{s}_1^{(j)}, \vec{n} \right\rangle \\ &\approx 1 \pm (0.1)\varepsilon_2\end{aligned}$$

is large.

(e) What if the noise is higher? For example:

$$\vec{y} = \vec{s}_1^{(j)} + \vec{n}$$

Does cross-correlation still work to find the delay? (Use the provided IPython notebook).

What about very high noise?

$$\vec{y} = \vec{s}_1^{(j)} + 10\vec{n}$$

Does cross-correlation still work to find the delay? If not, can you explain why? (use findings from previous parts).

Solution:

Noise with the same amplitude as the signal ($\vec{y} = \vec{s}_1^{(j)} + \vec{n}$) still works, as seen in the IPython notebook. We expect this since, similar to the rough calculations from the previous part, the maximum correlation for all $k \neq j$:

$$\max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle \lessapprox \varepsilon + (1)\varepsilon \approx 0.2$$

is still small compared to $\left\langle \vec{s}_1^{(j)}, \vec{y} \right\rangle \approx 1$.

However, very high noise ($\vec{y} = \vec{s}_1^{(j)} + 10\vec{n}$) no longer works – there is no clear autocorrelation peak. Again, we expect this because the maximum correlation for $k \neq j$ is now roughly:

$$\max_{k \neq j} \left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle \approx 10\varepsilon \approx 1.1,$$

which is larger than $\left\langle \vec{s}_1^{(j)}, \vec{y} \right\rangle \approx 1$. Thus, it is likely that some inner product $\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle, k \neq j$ is greater than the "correct" inner product $\left\langle \vec{s}_1^{(j)}, \vec{y} \right\rangle$.

In other words, the noise masks the signal in this case.

(f) Now suppose that there are two transmitters, sending known signals $\vec{s}_1$ and $\vec{s}_2$ at two unknown delays. That is, we receive

$$\vec{y} = \vec{s}_1^{(j)} + \vec{s}_2^{(k)}$$

for unknown shifts j, k. Both signals $\vec{s}_1$ and $\vec{s}_2$ are chosen as random normalized vectors, as before.

We can try to find the first signal delay by cross-correlating $\vec{y}$ with $\vec{s}_1$ (as in the previous parts). Similarly, we can try to find the second signal delay by cross-correlating $\vec{y}$ with $\vec{s}_2$. Run the provided IPython code to estimate the delays j, k. Does this method work to find both delays? Briefly comment on why or why not.

Solution:

Yes, this method works in this case, as demonstrated by the provided code.

We may expect this because when we try to find the delay of $\vec{s}_1$ by computing inner products $\left\langle \vec{s}_1^{(k)}, \vec{y} \right\rangle$, the signal $\vec{s}_2$ looks like "noise". In fact, $\vec{s}_2$ is chosen as an independent random normalized vector, just like $\vec{n}$ in the previous part. Thus, this works for exactly the same reason that the "medium noise" case ($\vec{y} = \vec{s}_1^{(j)} + \vec{n}$) worked in the previous part.

(g) Now, suppose the second transmitter is very weak, so we receive:

$$\vec{y} = \vec{s}_1^{(j)} + 0.1\vec{s}_2^{(k)}$$

Does the method of the previous part work reliably to find signal $\vec{s}_1$? What about $\vec{s}_2$? (Run the provided code a few times to test for different choices of random signals). Briefly justify why or why not.

Hint: $\vec{s}_1$ looks like "noise" when we are trying to find $\vec{s}_2$. Based on the previous parts, would you expect to be able to find $\vec{s}_2$ under such high noise?

Solution:

This still works to find $\vec{s}_1$ since the signal $\vec{s}_2$ looks like random noise with a very low amplitude. However, this no longer works reliably to find $\vec{s}_2$ for the same reason that the "very high noise" case in part (e) failed. That is, when we are trying to find $\vec{s}_2$, the signal $\vec{s}_1$ appears as random noise, at 10 times the amplitude of $\vec{s}_2$. Thus, as in part (e), we may expect that the "noise" masks the signal in this case.

(h) To address the problem of the previous part, suppose that we use the following strategy: First, cross-correlate to find the delay j of the strongest signal (say, $\vec{s}_1$). Then, subtract this out from the received $\vec{y}$, to get a new signal $\vec{y}' = \vec{y} - \vec{s}_1^{(j)}$. Then cross-correlate to find the second signal in $\vec{y}'$.

Run the provided IPython code to test this strategy for the setup of the previous part (with a strong and weak transmitter). Does it work? Briefly comment on why or why not.

Solution:

Yes, this new strategy works to find both signals. The received signal is:

$$\vec{y} = \vec{s}_1^{(j)} + 0.1\vec{s}_2^{(k)}$$

If we know the coefficient of $\vec{s}_1$ exactly (1 in this case) and we find the shift of $\vec{s}_1$ correctly, then when we subtract the contribution of $\vec{s}_1$, we find

$$\vec{y}' = \vec{y} - \vec{s}_1^{(j)} = 0.1\vec{s}_2^{(k)},$$

which only contains signal $\vec{s}_2$. Thus, we can cross-correlate it to find its shift.

5. Homework Process and Study Group

Who else did you work with on this homework? List names and student ID's. (In case of homework party, you can also just describe the group.) How did you work on this homework?

Solution:

I worked on this homework with...

I first worked by myself for 2 hours, but got stuck on problem 5, so I went to office hours on...

Then I went to homework party for a few hours, where I finished the homework.