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 $\vec{x}, \vec{y} \in \mathbb{R}^n$:

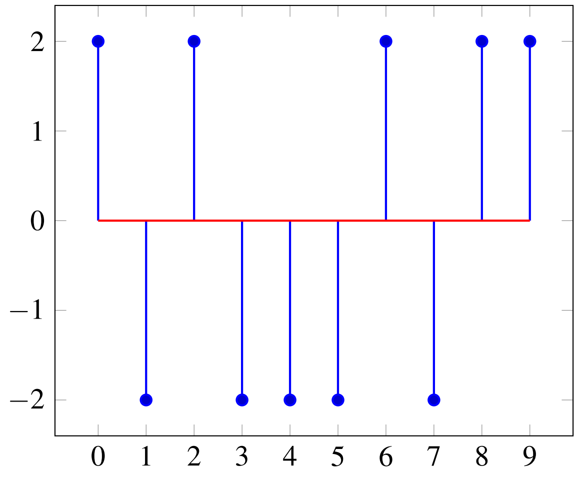
$$|\langle \vec{x}, \vec{y} \rangle| = |\vec{x}^T \vec{y}| \leq \| \vec{x} \| \cdot \| \vec{y} \|$$

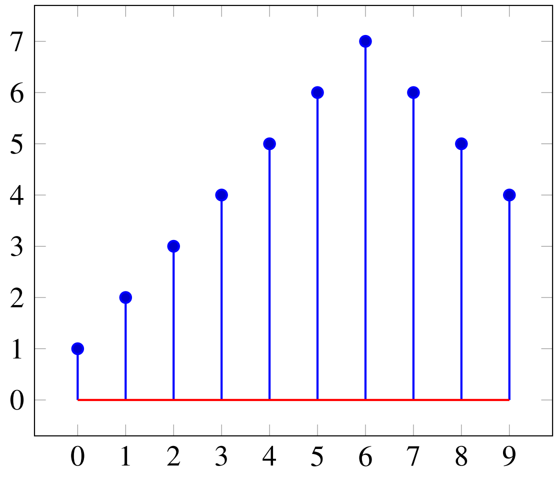
Use the Cauchy-Schwarz inequality to verify (i.e. prove or derive) the triangle inequality:

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

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

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.

(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).

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 Locationing and GPS.

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?

(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 $\vec{y} = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$?

(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?

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

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

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 $\langle \vec{s}_1, \vec{s}_1 \rangle = 1$.

(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 $k \neq j$?

(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)?

(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.

(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).

(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.

(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?

(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.

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?