EECS 16A Designing Information Devices and Systems I
Spring 2022 Lecture Notes Note 24

24.1 Introduction: Trilateration with multiple beacons

In the previous notes we learned how to find the distance from a beacon to a receiver using cross-correlation. Then we used trilateration to combine several of these measurements and find the the location of the receiver. However, we only looked at cross-correlation for finding the distance to *one* beacon, but we know that we need at least 3 beacons to find our location in 2D. How can we detect our distances from many beacons simultaneously?

In this note, we'll introduce an algorithm called **orthogonal matching pursuit (OMP)** that will allow us to unmix the signals from multiple beacons.

This algorithms isn't only useful for estimating distances – in addition, the beacons can encode messages in the signals they send, and using OMP we can recover these messages even when multiple beacons are transmitting at the same time.

24.2 Orthogonal Matching Pursuit (OMP)

We'll walk through the OMP algorithm using an example with three beacons. At the end of the note we'll provide a general formulation.

Consider three beacons (in GPS, these would be satellites) which are each transmitting their own unique periodic code. We'll denote these codes $\vec{s}_1$, $\vec{s}_2$, and $\vec{s}_3$. Each of these is length $N = 10$. For concreteness, let's say that the codes are as follows:

$$\vec{s}_1 = \begin{bmatrix} -4 \\ -1 \\ 1 \\ -1 \\ 1 \\ 0 \\ -3 \\ 4 \\ -4 \\ -3 \end{bmatrix} \qquad \vec{s}_2 = \begin{bmatrix} -4 \\ -5 \\ -4 \\ -4 \\ -2 \\ -2 \\ 4 \\ 2 \\ 1 \\ 3 \end{bmatrix} \qquad \vec{s}_3 = \begin{bmatrix} -2 \\ -1 \\ -1 \\ -4 \\ -4 \\ 5 \\ 2 \\ 4 \\ -4 \\ 1 \end{bmatrix}$$

Beacon 1 transmits $\vec{s}_1$ repeatedly, beacon 2 transmits $\vec{s}_2$ repeatedly, and so on. In addition, the beacons can each multiply their code by a scalar to encode additional information. For example, each beacon could tell the receiver the temperature at its location by multiplying its code by the temperature. (This could be helpful in determining which parts of the walls or doors need more insulation).

Our receiver is a different distance away from each beacon, so it receives a circularly shifted version of each code. It can't distinguish between the signals sent by the different beacons, so it measures the sum of the three incoming signals. We measure the first $N = 10$ samples at the receiver. Putting all of this together, we can model our received signal, $\vec{y}$, as

$$\vec{y} = a_1\vec{s}_1^{(\tau_1)} + a_2\vec{s}_2^{(\tau_2)} + a_3\vec{s}_3^{(\tau_3)} \tag{1}$$

where $\vec{s}_1^{(\tau_1)}$ is $\vec{s}_1$ circularly shifted by τ_1 samples. The a values are the scalars that the beacons can use to send messages.

From our known codes $\vec{s}_1, \ldots, \vec{s}_3$ and our measured receiver signal $\vec{y}$, we'd like to find the coefficients $a_1, \ldots, a_3$ and the shifts $\tau_1, \ldots, \tau_3$.

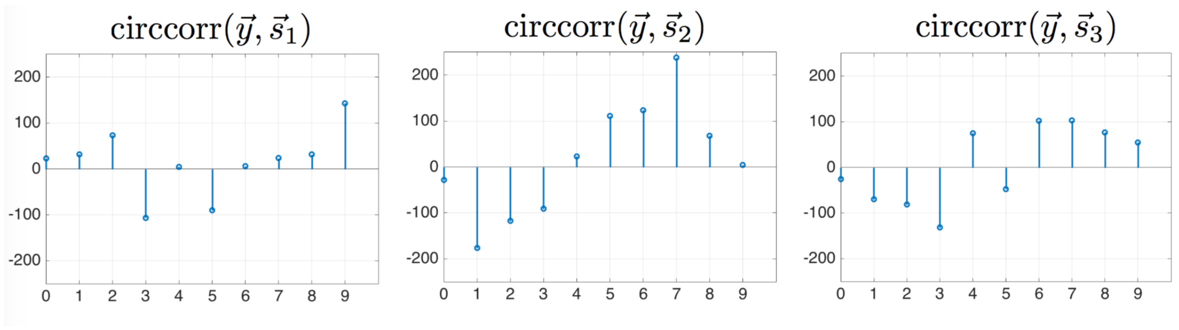
Suppose our received signal is:

$$\vec{y} = \begin{bmatrix} -11 & 2 & -3 & 12.5 & 4 & 0.5 & 6.5 & -11 & -12 & -2.5 \end{bmatrix}^T$$

We will now decompose it into the scaled and shifted versions of the beacon codes with OMP. OMP is an *iterative* method, which means we will repeat the same set of steps several times before reaching an answer.

Iteration 1

We know that our receiver signal consists of the sum of several shifted and scaled codes. First we will find the code (and it's corresponding shift) that is most similar to the received signal. To do this, we'll **take the circular cross correlation of $\vec{y}$ with each of the different possible codes** ($\vec{s}_1, \vec{s}_2, \vec{s}_3$). These values are shown below graphically.



The shifted code that is most similar to the received signal is the one with largest cross-correlation with the received signal. Therefore, we **find the code and shift with the maximum (in absolute value) cross-correlation.** Why absolute value? The transmitted code could have been scaled by a negative number, which will result in a negative value of the cross-correlation when these signals are similar. In this example, the maximum cross-correlation is given by $\vec{s}_2^{(7)}$, i.e. code $\vec{s}_2$ at shift 7.

Now we will guess that this shifted code is the only one in the received signal. In other words, we'll create an estimate of $\vec{y}$, denoted $\hat{\vec{y}}$, which takes the form

$$\hat{\vec{y}} = x_1\vec{s}_2^{(7)}.$$

We want to understand how well the signal $\vec{s}_2$ at shift 7 can explain the received signal $\vec{y}$. For this, we **set up and solve a least squares problem**. We define

$$A = \vec{s}_2^{(7)} \qquad \vec{x} = x_1 \qquad \hat{\vec{y}} = A\vec{x}.$$

To get the best estimate of $\vec{x}$, we try to find $\vec{x}$ that minimizes the difference between $\vec{y}$ and $\hat{\vec{y}}$. We do this with least squares:

$$\vec{x} = (A^T A)^{-1} A^T \vec{y}$$

Plugging in the values in our example gives

$$\vec{x} = 2.14.$$

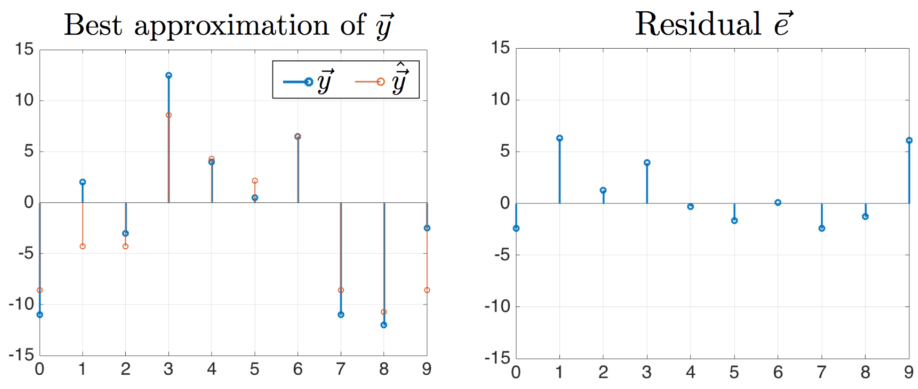
so our estimate of the received signal is

$$\hat{\vec{y}} = A\vec{x} = 2.14\,\vec{s}_2^{(7)}$$

However, our estimate of the received signal $\hat{\vec{y}}$, does not match our actual received signal. Next we will **calculate the difference between the actual received signal and estimated received signal, called the residual or error.** We'll denote the error signal $\vec{e}$.

$$\vec{e} = \vec{y} - \hat{\vec{y}}$$
$$\vec{e} = \vec{y} - A\vec{x}$$

The values of $\vec{y}$, $\hat{\vec{y}}$, and $\vec{e}$ for our example are shown below:

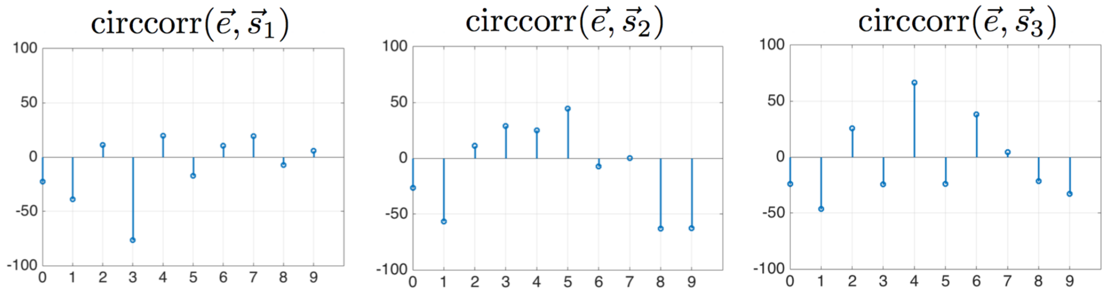


The residual is not zero, so we know that there are more codes from different beacons in the received signal. To find them, we will repeat the same process, but instead of starting with our received signal, we'll start with the residual. The logic is that the first shifted code we find is the strongest and will possibly block us from seeing other weaker codes. We first found this strongest code, and then removed it from the receiver signal to give us a better chance of finding the other weaker codes.

We're now ready for the second iteration of OMP.

Iteration 2

In the first iteration, the first thing we did was cross-correlate the receiver signal $\vec{y}$ with each code. We'll do the same thing this time but use the residual $\vec{e}$ instead of the receiver signal. Below are the circular cross correlations between the residual and each code:



Once again, we will find the shifted code that is most similar to the residual by taking the maximum (in absolute value) of the cross correlation. In this example, we find that the maximum is code $\vec{s}_1$ at shift 3 (it's a negative value, but biggest in magnitude). Note that the cross-correlation with $\vec{s}_2$ at shift 7 (the maximum from the first iteration) is 0. This is because we completely removed this component from the residual, forcing $\vec{e}$ to be orthogonal to $\vec{s}_2^{\,(7)}$. This is where the "orthogonal" in "orthogonal matching pursuit" comes from.

Now we update our estimate of the received signal to include the two shifted codes that we've found. (Recall that in the first iteration we found $\vec{s}_2$ at shift 7, and in this iteration we found $\vec{s}_1$ at shift 3).

$$\hat{\vec{y}} = x_1 \vec{s}_2^{\,(7)} + x_2 \vec{s}_1^{\,(3)}$$

We don't assume that x_1 is the same value as before, since we want to account for the additional codes that we've found. Instead we solve for both coefficients, x_1 and x_2, using least squares. We setup our least squares problem by defining

$$A = \begin{bmatrix} | & | \\ \vec{s}_2^{\,(7)} & \vec{s}_1^{\,(3)} \\ | & | \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \hat{\vec{y}} = A\vec{x}$$

so that

$$\vec{x} = (A^T A)^{-1} A^T \vec{y}$$

If we plug in the values in our example, we get

$$\vec{x} = \begin{bmatrix} 2.0 \\ -1.12 \end{bmatrix}$$

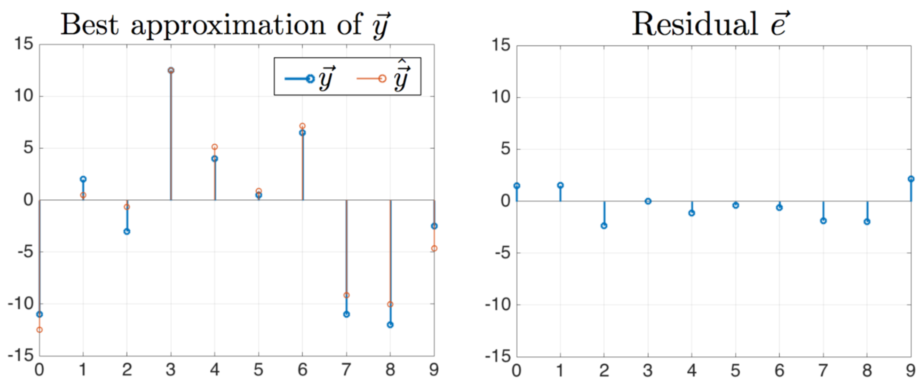
and our estimate of our received signal is now

$$\hat{\vec{y}} = 2.0\,\vec{s}_2^{\,(7)} - 1.1\,\vec{s}_1^{\,(3)}$$

.

Finally, we calculate our residual using our updated estimate $\hat{\vec{y}}$.

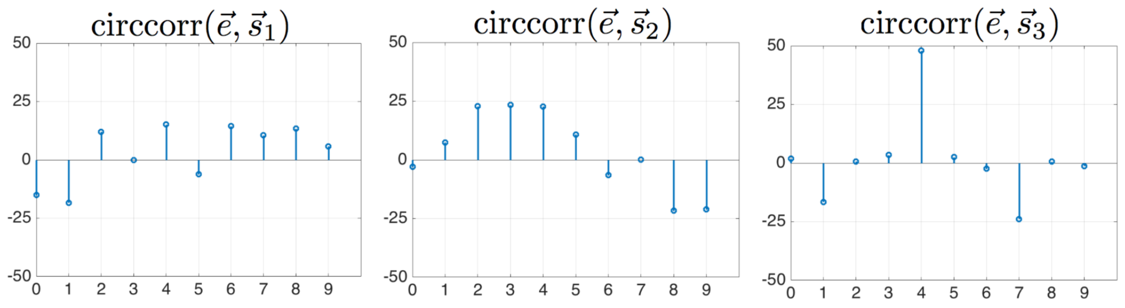
$$\vec{e} = \vec{y} - \hat{\vec{y}}$$
$$\vec{e} = \vec{y} - A\vec{x}$$



The residual is still not zero, and we know there were three beacons, so we will do a third iteration to find another beacon code.

Iteration 3

First, we take the cross-correlation between the new residual and each of the beacon codes.



Next, we find the code and shift with the maximum (in absolute value) cross-correlation. In this example, the maximum is $\vec{s}_3$ at shift 4. Once again, notice that the codes we found in previous iterations ($\vec{s}_2$ at shift 7 and $\vec{s}_1$ at shift 3) have zeros in the cross-correlation. This is because the least squares step ensures that the residual is orthogonal to all previous codes.

Now, we update our estimate of the received signal to include all three shifted codes that we've found.

$$\hat{\vec{y}} = x_1\vec{s}_2^{(7)} + x_2\vec{s}_1^{(3)} + x_3\vec{s}_3^{(4)}$$

We setup a least squares problem by defining

$$A = \begin{bmatrix} | & | & | \\ \vec{s}_2^{(7)} & \vec{s}_1^{(3)} & \vec{s}_3^{(4)} \\ | & | & | \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \hat{\vec{y}} = A\vec{x}$$

so that

$$\vec{x} = (A^T A)^{-1} A^T \vec{y}$$

If we plug in the values in our example, we get

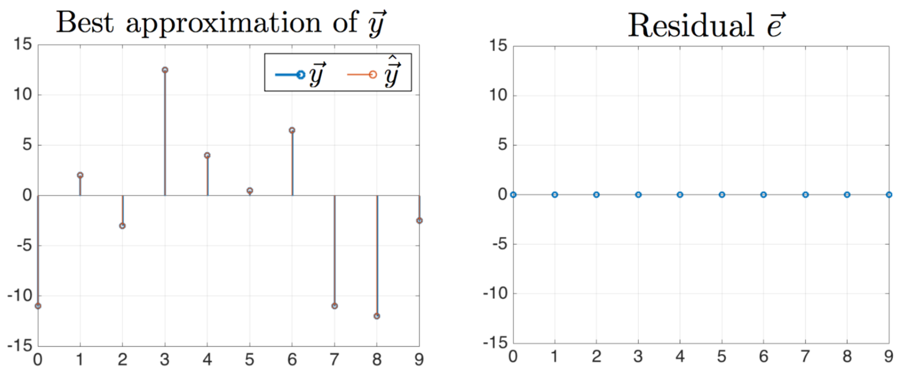
$$\vec{x} = \begin{bmatrix} 2.0 \\ -1 \\ 0.5 \end{bmatrix}$$

and our estimate of our received signal is now

$$\hat{\vec{y}} = 2.0\,\vec{s}_2^{(7)} - 1\,\vec{s}_1^{(3)} + 0.5\,\vec{s}_3^{(4)}$$

Finally, we calculate our residual using our updated estimate $\hat{\vec{y}}$.

$$\vec{e} = \vec{y} - \hat{\vec{y}}$$
$$\vec{e} = \vec{y} - A\vec{x}$$



Our residual is $\vec{0}$! We've found all of the codes:

$$\vec{y} = \hat{\vec{y}} = -1\,\vec{s}_1^{(3)} + 2.0\,\vec{s}_2^{(7)} + 0.5\,\vec{s}_3^{(4)}$$

24.3 Stopping Condition: How many iterations?

In the example above we did three iterations of OMP and at the end the residual was $\vec{0}$. However, in most examples there will be some noise in the measurement, so the residual will never completely vanish. How do we know how many iterations to run? We have two options:

1. We can run iterations until the residual is small. We set a threshold value th and when the norm of the residual is less than th, we will assume that the signal contains only noise and we'll stop iterating.
2. From the problem statement, we may already know how many beacon codes are contained in the received signal, k. Each iteration provides one new code, so the number of codes present will determine the number of iterations. In practice, we may not know the exact number, but we can guess and then do a few more iterations just in case.

We will use both of these criteria and we'll stop iterating when either is met. In other words, we'll set a number of iterations k, but we'll stop before k iterations if $||\vec{e}|| < th$.

24.4 OMP with many beacons

In the previous example, we looked at using OMP with three beacons, each with their own code. However, we can do OMP with many more beacons! These could be things like temperature sensors, light-meters, and humidity sensors. As in the previous example, each beacon has its own unique code to transmit, and the beacons send information by multiplying their codes with a scalar, a_i. The value of a_i is the message.

For OMP to work, each code should be "almost" orthogonal to all of the other codes, and to all the shifted versions of itself. (Orthogonal means that the inner product of two codes is exactly zero. "Almost" orthogonal means that the magnitude of the inner product is close to zero.) The "gold codes" that we looked at in the homework are good examples of codes that fulfill these properties and would work well for OMP.

Now imagine we have 2,000 beacons, each with its own code: $\vec{s}_0, \vec{s}_1, ..., \vec{s}_{1999}$. Each message is a real number that scales the code: $a_0, a_1, ..., a_{1999}$, for each of the 2,000 sensors. Each of the messages from the transmitters is delayed by a different amount, given by: $\tau_0, \tau_1, ..., \tau_{1999}$. Each code is periodic with period $N = 400$.

The signal received ($\vec{y} \in \mathbb{R}^{400}$) is a linear combination of shifted codes from all transmitting users given by:

$$\vec{y} = a_0 \vec{s}_0^{(\tau_0)} + a_1 \vec{s}_1^{(\tau_1)} + ... + a_{1999} \vec{s}_{1999}^{(\tau_{1999})}$$

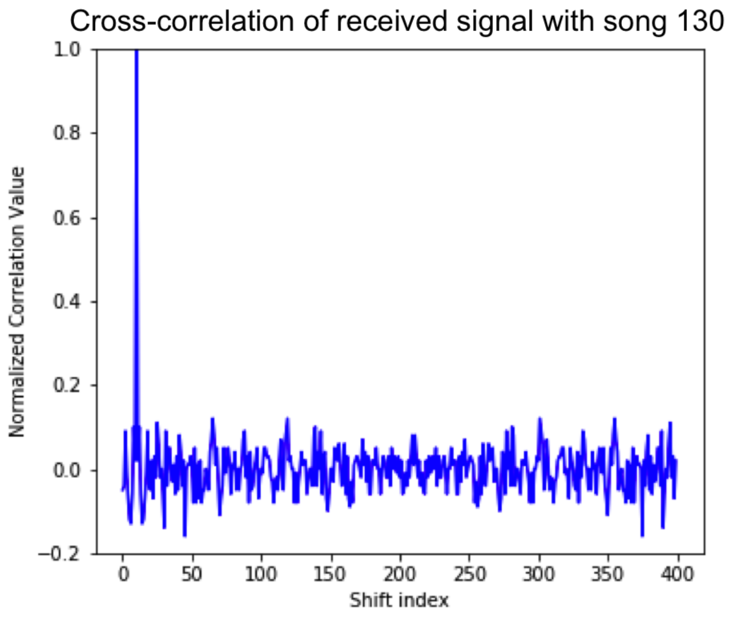
Here, the subscript of $\vec{s}$ indicates which beacon is transmitting, and the superscript in parenthesis represents the how many samples the code is shifted by.

We have 400 measurements ($\vec{y} \in \mathbb{R}^{400}$) and 2000 unknown values of a. How can we possibly determine the messages? We need more information! The additional information is that only a small number (say 10) beacons are transmitting at the same time. This means that only 10 of the beacons have non-zero a_i coefficients, but we don't know which beacons (and we don't know their shifts). If we put all of the a_i's in a vector, most of the vector would contain 0's. We call this a **sparse** vector and the **sparsity level** k is the number of non-zero elements.

In the simplest case, there will be only one beacon transmitting data to the receiver, for example. If beacon 130 is sending a message of value 1 with a delay of 10, the received signal will be:

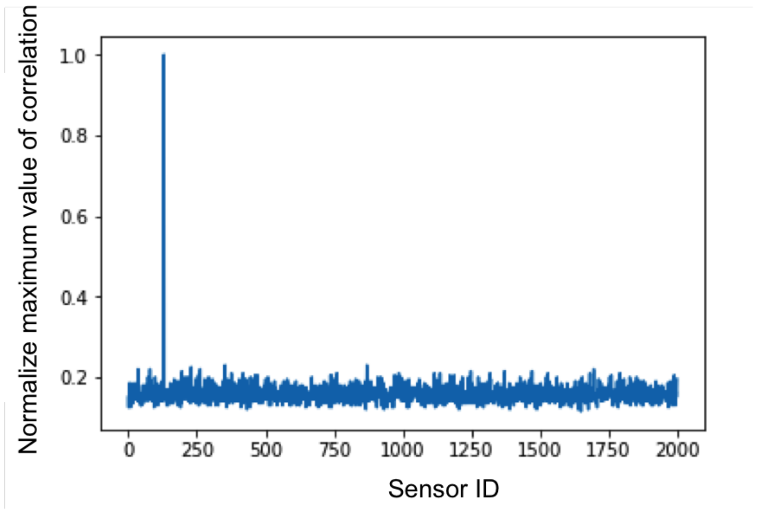
$$\vec{y} = 1 \cdot \vec{s}_{130}^{(10)}$$

Now if we perform cross-correlation of the received signal with the code from beacon 130, $\vec{s}_{130}$, the result will have a peak at shift index 10, with a normalized amplitude of 1, shown below:



Correlation of received signal with circular shifts of $\vec{S_{130}}$,
showing a peak at the true shift index of 10

If we correlate with all codes and plot the maximum from each one, we get the results shown below. There is a single peak at the beacon ID 130, telling us that beacon 130 was "on" and sending a value of 1.



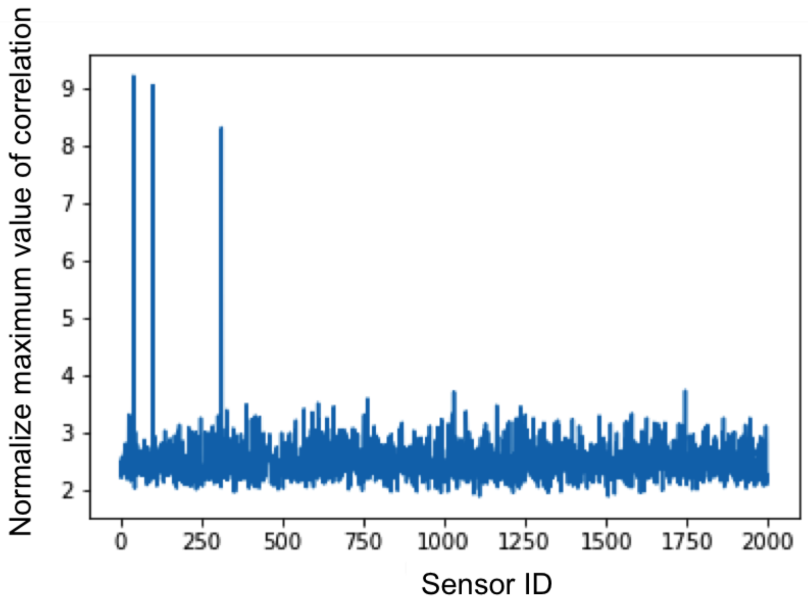
Correlation of received signal with circular shifts of all the codes.
This is only on peak, at a shift of 10 for beacon 130.

What happens when multiple transmitters are "on"?

Imagine beacons 40, 100, and 312 are simultaneously transmitting with delays of 13, 20, and 45 (respectively) and message values of 10, 10, and 8 (respectively). Our received signal is now:

$$\vec{y} = 10\,\vec{s}_{40}^{(13)} + 10\,\vec{s}_{100}^{(20)} + 8\,\vec{s}_{312}^{(45)}$$

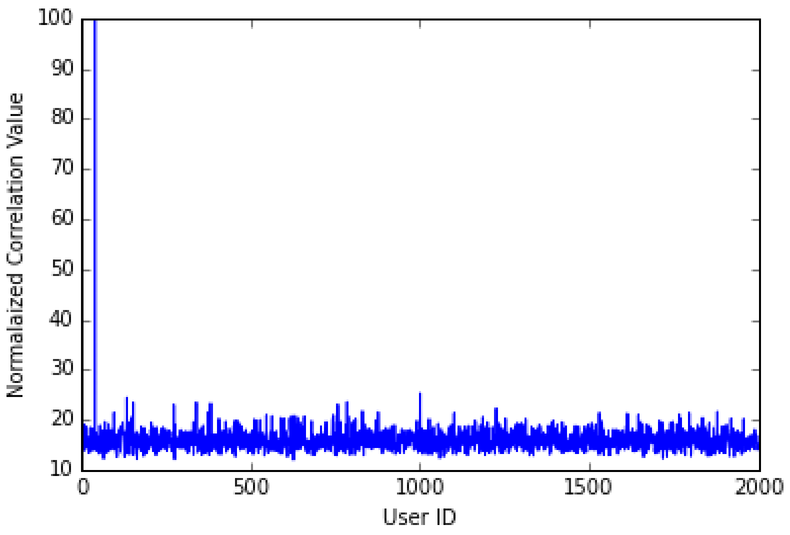
If we cross-correlate the received signal with every code, find the maximum peak of each cross-correlation, and plot it for each beacon ID (normalizing by the length of the code), we get the plot below.



Maximum value of correlation of received signal with circular shifts of all the codes.

We can see above, that beacons 40, 100, 312 are all “on”.

Let us look at one more example in which 4 beacons (beacon ID 40, 100, 312 and 350) transmit simultaneously and their corresponding messages are 100, 10, 8 and 0.02. The result of correlating the received signal with all of the different codes is shown below:



Maximum value of correlation of received signal with circular shifts of all the codes.

The peaks corresponding to sensor 100, 312 and 350 don't seem to appear at all! Even if we zoom in, we can't distinguish peaks corresponding to these sensors. What could possibly cause this? Perhaps the very high value of user 40's made it so that we cannot see the codes from users 100, 312 and 350.

However, as discussed in the previous example, OMP identifies the dominant code in the received signal, then removes it! We can use OMP in this example to recover the weaker codes as well as the dominant ones.

Here is a description of OMP with an example received signal of : $\vec{y} = a_{40}\vec{s}_{40}^{(13)} + a_{100}\vec{s}_{100}^{(8)}$ where $a_{40} = 100$ and $a_{100} = 10$.

1. Find the code with the highest correlation with $\vec{y}$, and find the shift that maximizes the correlation. We'll denote this shifted code as vector $\vec{s}_*^{(*)}$, where $*$ indicates the code ID and shift that maximize the cross-correlation. What does it mean for a vector $\vec{s}_*^{(*)}$ to have the highest correlation with $\vec{y}$? The highest correlation between two vectors means the error vector between them is the lowest. We find this vector via a cross-correlation with all the circular shifts of all the codes, the same first step we have always done. This gives us $\vec{s}_{40}^{(13)}$.

2. Use least squares to solve for $\vec{x}$ in the equation $A\vec{x} = \vec{b}$ where $\vec{b}$ is the received signal $\vec{y}$ and A is $\begin{bmatrix} | \\ \vec{s}_{40}^{(13)} \\ | \end{bmatrix}$:

   $\vec{x} = (A^TA)^{-1}A^T\vec{y}$. In this case, $\vec{x} = 100 = a_{40}$. The orthogonal projection of the least squares solution onto the subspace spanned by A (aka our 'ideal message') is given by: $A\vec{x}$.

3. Now find the residual of $\vec{y}$, denoted $\vec{e}$, left over by subtracting our ideal message from the received signal: $\vec{e} = \vec{y} - A\vec{x}$.

4. Repeat the above steps now using the residual $\vec{e}$ instead of the received signal $\vec{y}$. A new correlation between $\vec{e}$ and all of possible codes would find that the next strongest code is: $\vec{s}_{100}^{(8)}$. We then update our matrix A to be $\begin{bmatrix} | & | \\ \vec{s}_{40}^{(13)} & \vec{s}_{100}^{(8)} \\ | & | \end{bmatrix}$ Finally, we solve $\vec{x}$ via least squares. For this step, we use the received signal $\vec{y}$ (not the residual), $\vec{x} = (A^TA)^{-1}A^T\vec{y}$. We find that $\vec{x} = \begin{bmatrix} 100 \\ 10 \end{bmatrix}$. We have now recovered both of our message values!

5. We stop iterating when we have gone $k = 10$ steps, the known the sparsity of the signal, OR until the norm of the residual is below some threshold value, meaning the residual contains only noise.

24.5 OMP Summary

Setup:
Let the number of unique codes be m. Each code is length n. We represent each of the m codes with a vector $\vec{s}_i$ where i is an integer between 0 and $m-1$. Each code can potentially carry a message a_i along with it. Then the measurement at the receiver is

$$\vec{y} = a_0\vec{s}_0^{(\tau_0)} + a_1\vec{s}_1^{(\tau_1)} + \ldots + a_{m-1}\vec{s}_{m-1}^{(\tau_{m-1})}$$

We will assume that the number of codes that are being broadcast at the same time is very small (for example, 10). If a code is not being broadcast, it's coefficient a_i will be zero. Therefore, there are at most k non-zero a_i's.

Inputs:

- A set of m codes, each of length n: $\mathbf{S} = \{\vec{s}_0, \vec{s}_1, ..., \vec{s}_{m-1}\}$
- An n-dimensional received signal vector: $\vec{y}$
- The sparsity level k of the signal. This is the number of codes with non-zero coefficients.
- Some threshold, th. When the norm of the signal is below this value, the signal is assumed to contain only noise.

Outputs:

- A set of codes that were identified, F, which will contain at most k elements.
- A vector, $\vec{x}$ containing the coefficients of the codes (a_1, etc.), which will be of length k or less.
- An n-dimensional residual $\vec{e}$

Procedure:

- Initialize the following values: $\vec{e} = \vec{y}$, $j = 1$, $A = [\ ]$, $F = \{\emptyset\}$
- while $((j \leq k)$ & $(\| \vec{e} \| \geq th))$:
  1. Cross correlate $\vec{e}$ with each of the codes. Find the code index, i, and the shifted version of the code, $\vec{s}_i^{(\tau_i)}$, with which the received signal has the highest correlation value.
  2. Add i to the set of code indices, F.
  3. Column concatenate matrix A with the correct shifted version of the code: $A = [A \mid \vec{s}_i^{(\tau_i)}]$
  4. Use least squares to obtain the code coefficients: $\vec{x} = (A^T A)^{-1} A^T \vec{y}$
  5. Update the residual value $\vec{y}$ by subtracting: $\vec{e} = \vec{y} - A\vec{x}$
  6. Update the counter: $j = j + 1$
- Stop iterating when you've done k iterations (i.e. you've reached the pre-determined sparsity level of the signal) or when the norm of the residual drops below the threshold th. We'll stop at which ever one happens first.

24.6 Connections to Machine Learning

In this note, we showed you how by thinking systematically using the design philosophies taught in 16A, you could have systematically have come up with the algorithm that is commonly known as orthogonal matching pursuit or OMP. While this specific algorithm is certainly practically important in its own right as a building block, it is also a prime representative of a family of closely related algorithms that are very important in machine learning — namely what are called "boosting" algorithms (in case you want to read further on your own — there's no rush, these are topics that are usually covered if/when you take 189).

For those who want to know more, specifically "gradient boosting" algorithms can be understood as nothing more than a version of these ideas to be used when the relevant distance is not Euclidean squared distance, but something else. But all the main ideas are already present in this note about OMP. The most famous example of a gradient boosting algorithm is called AdaBoost, which is sometimes referred to as“the best out-of-the-box classificatio” algorithm for practical machine learning. In fact, modern machine learning is largely based on working through ideas whose seeds are taught to you across 16A and 16B.

To be more specific, you saw how from the hypothetical real world goal of being able to extract signals from many potential beacons, which might be at differing levels of signal strength, you could still find them by iteratively reducing interference one at a time.

The first approach you could think of was to simply ignore the interference and just search for the beacons one at a time. You saw that the problem was that a strong beacon signal could completely mask your ability to see a weaker signal.

To deal with this, you naturally thought of trying to find the signals in decreasing order of signal strength. First, you find the strongest one. Then, you remove its estimated effect from what you observed. And then, you tried to find the signal which was presumably the next strongest. And then remove its effect, and so on. This pattern is what is called “matching pursuit” in the literature. (Because it can be understood as trying to greedily reach a destination by picking the available direction that would bring you as close as possible, actually moving along that direction, and then repeating from your new vantage point.)

You saw that while this worked better, it didn’t revisit its estimate of the first found signal in light of the second signal that was found, and so on. This made errors in the residual signal potentially compound as you found more and more signals, making it harder to find weaker signals as you went on. So, you made one final version that took advantage of what you knew about projection to keep improving estimates as we found more signals — allowing us to better clean up the residual. This is what gave us orthogonal matching pursuit OMP.

While doing all this, you also noticed something else. OMP allows us to potentially break what seemed like a fundamental barrier at the beginning of 16A — namely that we need to have n equations/measurements if we want to solve for n unknowns. OMP tells us that in fact, we often can find n unknowns with fewer than n equations, if we somehow happened to know that most of the n potential unknowns were going to come out to be zero. This is the powerful idea of sparsity. This will be explored further in parts of 16B and beyond, but is actually something that is believed to be at the heart of what is sometimes considered the “unreasonable effectiveness” of machine learning algorithms in the real world.