Laboratory 2

Signal Correlation and Detection II

2.1 Introduction

In Lab 1, we designed an energy-based signal/no-signal detector for determining when a desired signal is present. This type of detector has a wide variety of applications, from speech analysis to communication, but it has two weaknesses. First, an energy-based detector is very susceptible to noise, especially when the signal's energy is small compared to the energy of the noise. Second, such a detector cannot distinguish between different types of signals that are mixed together.

In this laboratory, we will examine an alternative detection method that addresses these concerns. It uses a computation called *correlation* to detect the presence of a signal *with a known form*. In general, correlation measures the similarity between two signals. Using correlation for detection has significant applications. For instance, it allows several signals to be sent over a single communications channel simultaneously. It also allows the use of radar and sonar in noisy environments. Later in this course, we will see that correlation forms the basis for one of the most important tools in signals and systems engineering, the *spectrum*.

2.1.1 "The Questions"

- How can we transmit and receive bits from several different users on the same communication channel?
- How can we develop a radar detection scheme that is robust to noise, and how do we characterize its performance?

2.2 Background

2.2.1 Correlation

Suppose that we have two discrete-time signals, $x[n]$ and $y[n]$. We compute the *correlation*[1] between these two signals, $C(x, y)$, using the formula

$$C(x, y) = \sum_{n=n_1}^{n_2} x[n]y[n] \tag{2.1}$$

[1] We will occasionally refer to this operation as "in-place" correlation to distinguish it from "running" correlation. Sometimes this is also called an "inner product" or "dot product."

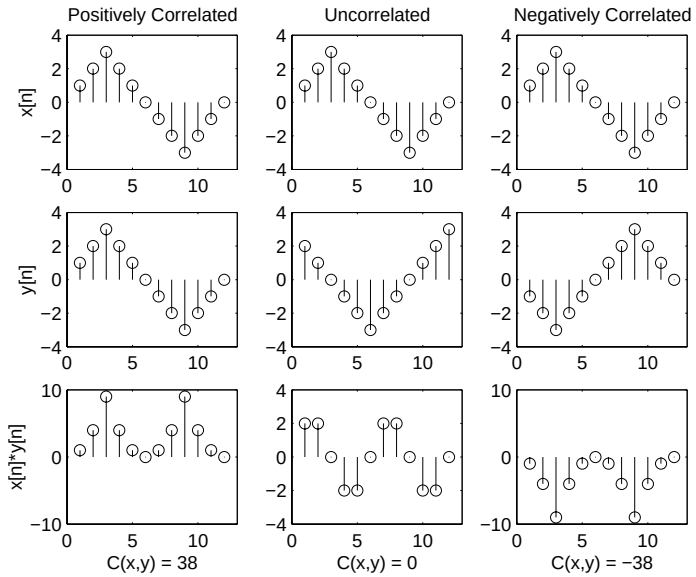

Figure 2.1: Examples of positively correlated, uncorrelated, and anticorrelated signals.

where n_1 and n_2 define the interval over which we are calculating the correlation. In words, we compute a correlation by multiplying two signals together and then summing the product. The result is a single number that indicates the similarity between the signals $x[n]$ and $y[n]$.

What values can $C(x, y)$ take on, and what does this tell us about the signals $x[n]$ and $y[n]$? Let us consider the examples in Figure 2.1. For the signals shown in the first column, $C(x, y) > 0$, in which case, the signals are said to be *positively correlated*. Basically, this means that the signals are more similar than they are dissimilar. In the second column, we can see an example where $C(x, y)$ is zero. In this case, the two signals are *uncorrelated*. One might say that uncorrelated signals are "equally" similar and dissimilar. Notice, for instance, that the signal $x[n] \times y[n]$ is positive as often as it is negative. Knowledge of the value of signal $x[n]$ at time n indicates little about the value of $y[n]$ at time n. Finally, in the third column we see an example where $C(x, y) < 0$, which means that $x[n]$ and $y[n]$ are *negatively correlated*. This means the signals are mostly dissimilar.

Note that the positively correlated signals given in Figure 2.1 are actually identical. This is a special case; from equation (2.1), we can see that in this case the correlation is simply the energy of $x[n]$, i.e.

$$C(x, x) = E(x) \; . \tag{2.2}$$

Sometimes, it is more useful to work with *normalized correlation*, as defined by

$$C_N(x, y) = \frac{C(x, y)}{\sqrt{E(x)E(y)}} = \frac{1}{\sqrt{E(x)E(y)}} \sum_{n=n_1}^{n_2} x[n]y[n]. \tag{2.3}$$

Normalized correlation is somewhat easier to interpret. The well known Cauchy-Schwartz inequality shows that the normalized correlation varies between -1 and +1. That is, for any two signals

$$-1 \leq C_N(x, y) \leq 1. \tag{2.4}$$

Thus, signals that are as positively correlated as possible have normalized correlation 1 and signals that are as negatively correlated as possible have normalized correlation -1. Moreover, it is known that two signals have normalized

correlation equal to 1 when and only when one of the signals is simply the other multiplied by a positive number. In this case, the signals are said to be *perfectly correlated*. Similarly, two signals have normalized correlation equal to -1 when and only when one is simply the other multiplied by a negative number, in which case they are said to be *perfectly anticorrelated*.

2.2.2 Running correlation

In many situations, it is quite useful to correlate a signal $y[n]$ with a sequence of delayed versions of another signal $x[n]$. That is, we wish to correlate $y[n]$ with $x[n]$, with $x[n-1]$, with $x[n-2]$, etc. In such cases, we perform *running correlation* of $y[n]$ with $x[n]$, which produces the *correlation signal*

$$\begin{aligned} r[k] &= C(x[n-k], y[n]) \, , k = 0, 1, 2, \ldots && (2.5) \\ &= \sum_n x[n-k]y[n] \, , k = 0, 1, 2, \ldots && (2.6) \end{aligned}$$

Note that since n was used as the time variable for x and y, we have introduced a new time variable, k, for r.

Suppose, for example, that we want to know the distance to a certain object, like an airplane. We transmit a radar pulse, $x[n]$, and receive a signal, $y[n]$, that contains the reflection of our pulse off of the object. For simplicity, let's assume that we know $y[n]$ is simply a delayed version of $x[n]$, that is[2],

$$y[n] = x[n - n_0], \tag{2.7}$$

However, we do not know the delay factor, n_0. Since n_0 is proportional to the distance to our object, this is the quantity that we wish to estimate. We can use correlation to help us determine this delay, but we need to use running correlation rather than simply in-place correlation.

Suppose that we first guess that n_0 is equal to zero. We correlate $y[n]$ with $x[n]$ and record the resulting correlation value as one sample of a new signal, $r[0]$. Then, we guess that n_0 is equal to one, shift $x[n]$ over by one sample, and correlate $y[n]$ with $x[n-1]$. We record this correlation value as $r[1]$. We can continue this shift-and-correlate procedure, building up the new signal $r[k]$ according to the formula

$$r[k] = C(x[n-k], y[n]) = \sum_{n=-\infty}^{\infty} x[n-k]y[n]. \tag{2.8}$$

Once we find a value of $r[k]$ that equals $E(x)$, we have found the value of n_0. This procedure of building up the signal $r[k]$ is known as *running correlation* or *sliding correlation*. We will refer to the resulting signal ($r[k]$ above) as the *correlation signal*.

As an example, Figure 2.2 shows a radar pulse, a received signal containing two delayed versions of the radar pulse (one without noise and one with noise), and the running correlation produced by correlating the pulse with the received signal.

Let us note a couple important features of the correlation signal. First, the limits of summation in equation (2.8) are infinite. Usually, though, the support of $x[n]$ and $y[n]$ will be finite, so we do not actually need to perform an infinite summation. Instead, the duration of the correlation signal will be equal to the sum of the durations of $x[n]$ and $y[n]$ minus one[3]. There will also be *transient effects* (or *edge effects*) at the beginning and end of the correlation signal. These transient effects result from cases where $x[n-k]$ only partially overlaps $y[n]$. Finally, notice that the value of the correlation signal at time $k = 0$ is just the in-place correlation $C(x[n], y[n])$.

[2] Recall that a signal $x[n - n_0]$ is equal to the signal $x[n]$ shifted n_0 samples to the right.

[3] Suppose that support interval of $x[n]$ is $n_{x_1} \leq n \leq n_{x_2}$, while the support interval of $y[n]$ is $n_{y_1} \leq n \leq n_{y_2}$. For this general case, we can see that the first nonzero sample of $r[k]$ will occur at $k = n_{y_1} - n_{x_2}$. Similarly, the last nonzero sample will fall at $k = n_{y_2} - n_{x_1}$. Thus, the duration of $r[k]$ is $(n_{y_2} - n_{y_1}) + (n_{x_2} - n_{x_1}) + 1 = (n_{y_2} - n_{y_1} + 1) + (n_{x_2} - n_{x_1} + 1) - 1 = \text{duration}(x) + \text{duration}(y) - 1$.

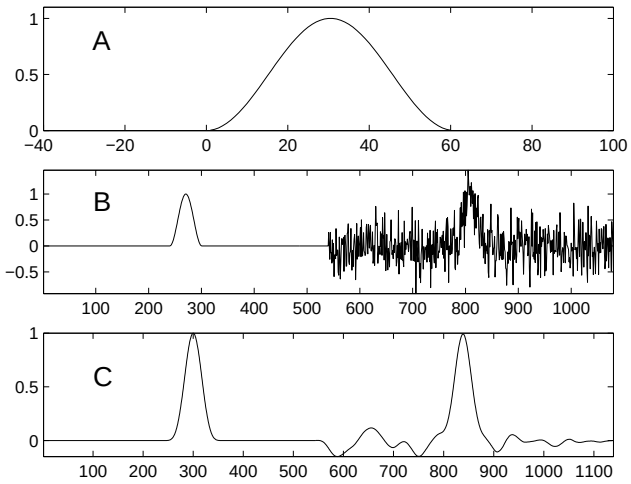

Figure 2.2: (A) A radar pulse. (B) A received sequence from the radar system, containing two pulses and noise. (C) The running correlation produced by correlating the radar pulse with the received signal.

2.2.3 Using correlation for signal detection

Whenever we wish to use correlation for signal detection, we use a two-part system. The first part of the system performs the correlation and produces the correlation value or correlation signal, depending upon whether we are doing in-place or running correlation. The second part of the system examines the correlation or correlation signal and makes a decision or sequence of decisions. See the block diagram given in Figure 2.3.

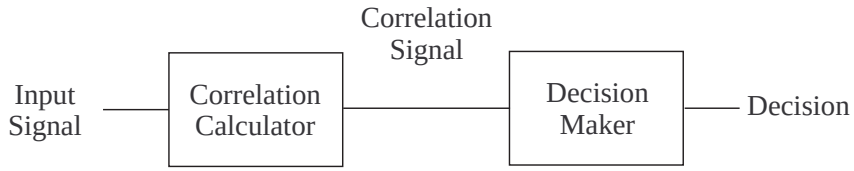

Figure 2.3: A generalized block diagram for a correlation-based detection system.

In the radar example used to motivate running correlation in Section 2.2.2, we simply checked to see if the correlation signal at a given point equals the energy of the transmitted signal. While this will work for the idealized system presented, real systems are usually much less ideal. We may have multiple reflections, distorted reflections, reductions in reflection amplitude, and various kinds of environmental noise. In order to address such problems in a wide variety of systems, we commonly use a simple threshold comparison as our decision maker. For instance, if we compute a running correlation signal $r[n]$, we might choose a constant c called a *threshold* and make a decision for each sample based on the following formula:

$$r \underset{0}{\overset{1}{\gtrless}} c \tag{2.9}$$

That is, when the correlation value r is greater than the threshold, c, we *decide* 1, or "signal present." If the value is less than the threshold, we decide 0, or "signal absent." In our radar example, for instance, we might select the threshold to be $c = E(x)/2$.

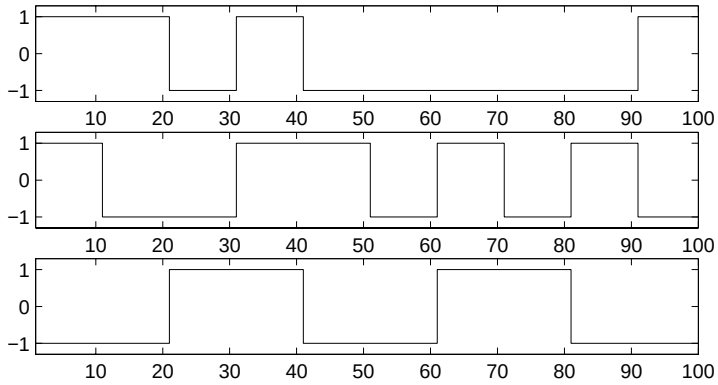

Figure 2.4: Example code signals for simultaneous communications

2.2.4 Using correlation for detection of signals transmitted simultaneously with other signals

Suppose that we wish to design a multi-user wireless communication system that permits several users to simultaneously transmit a sequence of message bits. That is, each user will transmit a signal across a common communication channel (for example, a wire or a small portion of the electromagnetic spectrum) that conveys his/her message bits, and despite the fact that these signals are received on top of one another, it should be possible to decode the message bits produced by any one of the users. The users of this system are completely uncoordinated; so no user has any idea who else might be using the system at any given time. How can we design a system so that each user can use the system without experiencing interference from the other users? This is the problem faced by the designers of cell phones and cordless phones, for example.

It turns out that we can use a correlation-based detector to address this problem. To begin, suppose that each user is trying to send a one bit message to a friend, and suppose each user has a distinct *code signal*, like those shown in Figure 2.4. Each code signal is made up of some number of binary *chips*, which are regions of constant signal value; the signals shown here each consist of ten chips.[4] To send a "one" message bit, the user transmits his or her code signal. To send a "zero" message bit, the user instead transmits a negated version of his or her code signal (which is is perfectly anticorrelated with the code signal).

Now, to send a *sequence* of message bits, the user concatenates these positive and negative versions of the code signal into a *sequence of code signals*, which is called the *transmitted signal* and which is input to the communication channel. Other users transmit their own message bits in the same fashion, except that, of course, they use different code signals. For example, Figure 2.5 shows a transmitted signal conveying eight message bits using the top code signal from Figure 2.4. It also shows this signal with the transmitted signals from three other users added to it. Notice that the signal in the upper panel is obscured in the lower panel.

When someone, say the *user's friend*, receives the signal from the communication channel[5] and wishes to *decode* the user's message bits, the friend correlates the received signal with the user's code signal. Specifically, in-place correlation of the received signal with the code signal produces a value with which a decision about the first message bit can be decided. Then in-place correlation of the received signal with a delayed version of the code signal produces a value from which the second message bit can be decided, and so on. Since each of these correlation values would equal plus or minus the energy of the code signal if there were no other signals or noise present, it is natural to have

[4] Though the code signals clearly have a binary nature, we use the term "chip" to distinguish binary segments of the code signal from binary message elements, which we call "bits".

[5] For simplicity, we assume the communication channel does not attenuate or otherwise distort the transmitted signal.

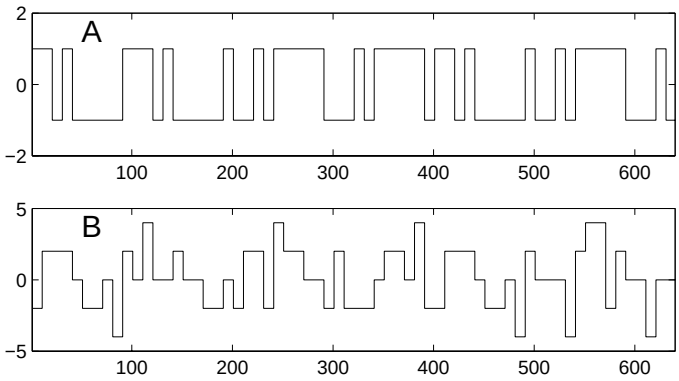

Figure 2.5: (A) Example of a transmitted signal. (B) The sum of the transmitted signals from four users.

the decision maker decide that a message bit is "one" if the correlation value is positive and "zero" if the correlation value is negative. That is, a threshold of zero is chosen.

A communication system of this form is said to be a *code-division, multiple-access* (CDMA) system or a *direct-sequence, spread-spectrum* (DSSS) system. They are used, for example, in 900 Mhz cordless telephones. Such systems work best when the code signals are as different as possible, i.e. when the normalized correlation between the code signals of distinct users are as near to zero as possible, which is what system designers typically attempt to do. Consider the examples in Figure 2.4. The first two code signals are completely uncorrelated, as are the second two. The first and third signals are slightly anticorrelated. The normalized correlation between these signals is only -0.2, which is small enough that these two code signals will not interfere much with one another.

Above, we've indicated that our detection system uses in-place correlation. This means that this system is *synchronous*; that is, the receiver knows when bits are sent. However, we can actually save ourselves some work by using running correlation, and then sampling the resulting correlation signal at the appropriate times. This is how we will implement this communication system in the laboratory assignment. Using the running correlation algorithm presented in this lab, the "appropriate times" occur in the correlation signal at the end of each code signal. That is, if our code signals are N samples long, we want to pick off the $(k \times N)^{th}$ sample out of the running correlator to decide the k^{th} message bit.

The threshold used to decode bits in this detection system, which we have chosen to be zero, is actually a design parameter of the system. If it should happen, for instance, that the system's noise is biased in a way that we tend to get slightly positive correlations when no signal is sent, then we would be able to improve performance of the system by using a positive threshold, rather than a threshold of zero. Alternatively, we might want to decide that no bit has been sent if the magnitude of the correlation is below some threshold. In this case, we actually have two thresholds. One separates "no signal" from a binary "one;" the other separates "no signal" from a binary "zero."

2.2.5 Noise, detector errors, and setting the threshold

Detectors, such as the radar and DSSS detectors we have discussed, must typically operate in the presence of noise. Here, we begin by discussing the radar example of Section 2.2.2, and conclude with a brief discussion of the DSSS detector.

When the radar pulse is $x[n]$, the typical received radar signal has the form

$$y[n] = x[n - n_0] + w[n] \tag{2.10}$$

where $x[n-n_0]$ is the reflected radar pulse and $w[n]$ is noise, i.e. an unpredictable, usually wildly fluctuating signal that

normally is little correlated with $x[n]$ or any delayed version of $x[n]$. To estimate n_0, we perform running correlation of $y[n]$ with $x[n]$, and the resulting correlation signal is

$$\begin{aligned} r[k] &= \sum_{n=-\infty}^{\infty} x[n-k]y[n] \\ &= \sum_{n=-\infty}^{\infty} x[n-k](x[n-n_0]+w[n]) \\ &= \sum_{n=-\infty}^{\infty} x[n-k]x[n-n_0] + \sum_{n=-\infty}^{\infty} x[n-k]w[n] \\ &= r_0[k] + r_w[k]\ , \end{aligned} \tag{2.11}$$

where $r_0[k]$ is the running correlation of $x[n-n_0]$ with $x[n]$. Note that $r_0[k]$ is what $r[k]$ would be if there were no noise, as given in equation (2.8). $r_w[k]$ is the running correlation of the noise $w[n]$ with $x[n]$, which is added to $r_0[k]$. This shows that the effect of noise is to add $r_w[k]$ to $r_0[k]$. Though in a well designed system $r_w[k]$ is usually close to zero, it will occasionally be large enough to influence the decision made by the decision maker.

In Section 2.2.3, we argued that a threshold-based decision maker was useful for such systems. Then, for example, when $r[k] > c$, the decision is that a radar pulse is present at time k, whereas when $r[k] < c$, the decision is that no radar pulse is present at time k. Since in the absence of noise $r[k] = E(x)$ when there is a radar pulse at time k, and since $r[k] = 0$ when there is no pulse at time k, it is natural, as mentioned in Section 2.2.3 to choose threshold $c = E(x)/2$.

Though it makes good sense to use a threshold detector, such a detector will nevertheless occasionally make an *error*, i.e. the wrong decision. Indeed, there are two types of errors that a detector can make. First, it could detect a reflection of the transmitted signal where no actual reflection exists. This is called a *false alarm*. It occurs at time k when $r_0[k] = 0$ and $r_w[k] > c$, i.e. when the part of the correlation due to noise is larger than the threshold. The other type of error occurs when the detector fails to detect an actual reflection because the noise causes the correlation to drop below the threshold even though a signal is present. This type of error is called a *miss*. It occurs when $r_0[k] = E(x)$ and $r[k] = E(x) + r_w[k] < c$, which in turn happens when $r_w[k] < c - E(x)$. In summary, a false alarm occurs when there is no radar pulse present, yet the noise causes $r_w[k] > c$, and a miss occurs when there is a radar pulse present, yet the noise causes $r_w[k] < c - E(x)$.

Depending on the detection system being developed, these two types of error could be equally undesirable or one could be more undesirable than another. For instance, in a defensive radar system, false alarms are probably preferable to misses, since the former are decidedly less dangerous. We can trade off the likelihood of these two types of error by adjusting the threshold. Raising the threshold decreases the likelihood of a false alarm, while lowering it decreases the likelihood of a miss.

It is often useful to know the frequency of each type of error. There is a simple way to empirically estimate these frequencies. First, we perform an experiment where we do not send any radar pulses, but simply record the received signal $y[n]$, which contains just environmental noise $w[k]$. We then compute its running correlation $r[k]$ with the radar pulse $x[n]$, which is just $r_w[k]$. We count the number of times that $r_w[k]$ exceeds the threshold c and divide by the total number of samples. This gives us an estimate the *false alarm rate*, which is the frequency with which the detector will decide a radar pulse is present when actually there is none. We can also use this technique to estimate the *miss rate*. When a radar pulse is present, an error occurs when $r_w[k] < c - E(x)$. Thus, we can estimate the *miss rate* by counting the number of times the already computed correlation signal $r_w[k]$ is less than $c - E(x)$, and dividing by the total number of samples.

The signal value distribution is also useful here. If we plot the histogram of values in $r_w[k]$, we can use this plot to determine the error rate estimates. The estimate of false alarm rate is the area of the histogram above values that

exceeds c, divided by the total area of the histogram[6]. Similarly, the estimate of the miss rate is the area of the hisogram above values that are less than $c - E(x)$

Assuming that the distribution of $r_w[k]$ is symmetric about 0, we can minimize the *total error rate* (which is simply the sum of the false alarm and miss rates) by setting a threshold that yields the same number of false alarms as misses. Since the distribution of $r_w[k]$ is assumed to be symmetric, we get an equal number of false alarms and misses when $c = E(x)/2$, which is the threshold value suggested earlier.

Next, it is important to note how the error rates depend on the energy of the radar pulse. Consider first the false alarm rate, which corresponds to the frequency with which the noise induced correlation signal $r_w[k]$ exceeds $c = E(x)/2$. Suppose for example that the radar pulse is amplified by a factor of two. Then its energy increases by a factor of four, and consequently, the threshold c increases by a factor of four. On the other hand, one can see from the formula $r_w[k] = \int y[n]x[n-k]\,dx$ that the noise term $r_w[k]$ will be doubled. Since the threshold is quadrupled but the noise term is only doubled, the frequency with which the noise term exceeds the threshold will be greatly decreased, i.e. the false alarm rate is greatly decreased. A similar argument shows that the miss rate is also greatly decreased. Thus, we see that what matters is the energy of the radar pulse, in relation to the strength of the noise. If the energy of the signal increases, but the typical values of the noise $w[n]$ remain about the same, the system will make fewer errors. By making the energy sufficiently large, we can make the error rate as small as we like. In the lab assignments to follow, we will observe situations where the noise $w[n]$ is so strong that it completely obscures the radar pulse $x[n - n_0]$, yet the radar pulse is long enough that it has enough energy that a correlation detector will make few errors.

Finally, we comment on the effects of noise on the DSSS detector. In this case, instead of deciding whether a pulse is present or not, the detector decides whether a positive or negative code signal is present. As with the radar example, this must ordinarily be accomplished in the presence of noise. However, in this case there are two kinds of noise: environmental noise, similar to that which affects radar, and multiple user noise, which is due to other users transmitting their own code signals. In the absence of any noise, the in-place correlation $r(x, y)$ computed by the detector will be $+E(x)$ when the message bit is "one" and $-E(x)$ when the message bit is "zero" , where $x[n]$ is the user's code signal. For this reason, using a decision threshold $c = 0$ is natural. When the message bit is zero, an error occurs when $r(x, y) > 0$, which happens when the correlation term r_w due to noise exceeds $E(x)$. Similarly, when the message bit is one, an error occurs when $r(x, y) < 0$, which happens when $r_w < -E(x)$. As with the radar example, errors occur less frequently when the signal energy becomes larger. This will be evident in the lab assignment, when code signals of different lengths, and hence different energies, are used.

2.2.6 An algorithm for running correlation

Here, we provide an algorithm for running correlation. One of its primary benefits is that it is easy to understand. In this algorithm, we imagine the filter as a box into which we drop one new sample of the "incoming" signal and a corresponding new sample of the correlation signal comes out. This allows the algorithm to be used in real-time: as samples of our signal arrive (from a radar detector, for instance), we can process the resulting signal with almost no delay.

In this algorithm we refer to the signal we are looking for (i.e., the transmitted radar signal) as $x[n]$, following (2.8). The algorithm goes like this:

1. Initialize an *input buffer*, which is simply an array with length equal to the duration of $x[n]$, to all zeros.
2. For each sample that comes in:
   (a) Update the buffer by doing the following:
      i. Discard the sample at the beginning of the buffer.
      ii. Shift the rest of the samples one place towards the beginning of the buffer.

[6] That is, we sum the values in this region of the histogram and divide by the sum of all values in the histogram

iii. Insert the incoming sample at the end of the buffer.

(b) Initialize a running sum variable to zero.

(c) For each position, n, in the buffer:

i. Multiply the n^{th} position in the input buffer by the n^{th} sample of $x[n]$.

ii. Add the resulting product to the running sum.

(d) Output the running sum as the next sample of the correlation signal.

In the laboratory assignment, you will be asked to complete an implementation of this algorithm. Note that significant portions of this algorithm can be implemented very simply in MATLAB. For instance, all of (a) can be accomplished using a single line of code. Similarly, parts (b) through (d) can all be accomplished in a single line using one of MATLAB's built-in functions and its vector arithmetic capabilities.

2.3 Some MATLAB commands for this lab

- **Calculating in-place correlation:** If you have two signals, `x` and `y`, that you wish to correlate, simply use the command

  ```
  >> c_xy = sum(x.*y);
  ```

  Note that `x` and `y` must be the same size; otherwise MATLAB will return an error.

- **The `subplot` command:** To put several plots on the same figure in MATLAB, we use the `subplot` command. `subplot` creates a rectangular array of axes in a figure. Figure 2.6 has an example figure with such an array. Each time you call `subplot`, you activate one of the axes. `subplot` takes three input parameters. The first and second indicate the number of axes per row and the number of axes per column, respectively. The third parameter indicates which of the axes to activate by counting along the rows[7]. Thus the command:

  ```
  >> subplot(2,3,5)
  ```

  activates the plot with the circle in Figure 2.6.

- **The `axis` command:** The command `axis([x_min, x_max, y_min, y_max])` allows us to set the axis display range for a particular plot. If you wish to change the display range of the currently active plot (or subplot) so that the x-axis ranges from 5 and 10 and the y-axis ranges from -100 to 100, simply execute the command

  ```
  >> axis([ 5, 10, -100, 100]);
  ```

  Other useful forms of the `axis` command include `axis tight`, which fits the axis range closely around the data in a plot, and `axis equal`, which assures that the x- and y-axes have the same scale.

- **Buffer operations in MATLAB:** It is often useful to use MATLAB's vectors as *buffers*, with which we can shift values in the buffer towards the beginning or end of the buffer by one position. Such an operation has two parts. First, we discard the number at the beginning or end of the buffer. If our buffer is a vector `b`, we can do this using either `b = b(2:end)` or `b = b(1:end-1)`. Then, we append a new number to the opposite end of the buffer using a standard array concatenation operation. Note that we can easily combine these two steps into a single command. For instance, if `b` is a row vector and we wish to shift towards the end of the buffer, we use the command

[7] Note in particular that this is the opposite of MATLAB's usual convention!

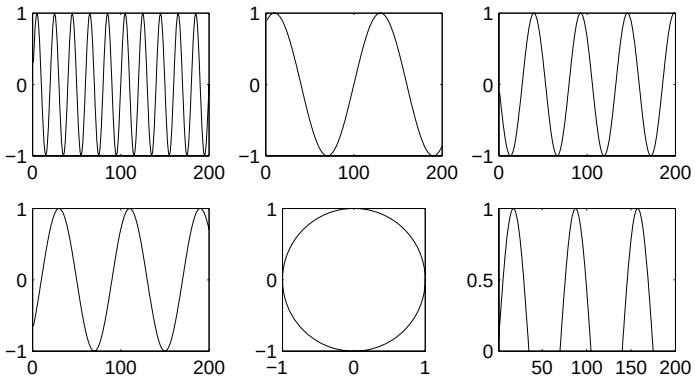

Figure 2.6: Example of a MATLAB figure with subplots.

```
>> b = [ new_sample, b(1:end-1) ];
```

- **Counting elements that meet some condition:** Occasionally we may want to determine how many elements in a vector meet some condition. This is simple in MATLAB because of how the conditional operators are handled. Recall that for a vector, `v`, `(v == 3)` will return a vector with the same size as `v`, the elements of which are either 1 or 0 depending upon the truth of the conditional statement. Thus, to count the number of elements in `v` that equal 3, we can simply use the command

  ```
  >> count = sum(v == 3);
  ```

2.4 Demonstrations in the Lab Section

- Detector error types
- Why use correlation? Or, when energy detectors break down.
- "In-place" correlation as a similarity measure
- Running correlation
- Multi-user communication

2.5 Laboratory Assignment

In this lab assignment, all signals are discrete-time and their support is assumed to be of the form $\{1, 2, \ldots, N\}$.

1. (Computing and interpreting in-place correlations) Download the file `code_signal.m` and use it to create the following signals:

   ```
   >> code1 = code_signal(75,10);
   >> code2 = code_signal(50,10);
   >> code3 = code_signal(204,10);
   ```


(a) (Plotting code signals) Use `subplot` and `stairs` to plot the three code signals on three separate axes in the same figure. After plotting each signal, make sure that the signal is visible by calling `axis([1, 100, -1.5, 1.5])`.

- [4] Include your figure, with axis labels on *each subplot*, a figure number and caption, and the generating code in your report.

(b) (Calculate statistics) For each of the three signals generated above, calculate:

- [3] Their mean values.
- [3] Their energies.

(c) (Calculate correlations) Calculate the "in-place" correlation and normalized correlation for the following pairs of signals.

- [2] `code1` and `code2`
- [2] `code1` and `code3`
- [2] `code2` and `code3`

(d) (Classify correlations) For each of the signal pairs given in problem 1c:

- [3] Identify each pair as positively correlated, uncorrelated, or negatively correlated.

2. (Implementing and interpreting running correlation) Download the file `run_corr.m`, which is a "skeleton" file for an implementation of the "real-time" running correlation algorithm described in Section 2.2.2. It accepts two input signals, performs running correlation on them, and produces the correlation signal with a length equal to the sum of the lengths of the input signals minus one.

(a) (Write the code) Complete the function, following the algorithm given in Section 2.2.2. You can use the completed demo version of the function, `run_corr_demo.dll` to check your function's output[8].

- [10] Include your code in the MATLAB appendix of your report.

(b) (Compute running correlations) Use `run_corr.m` to compute the running correlation between the following pairs of signals, and plot the resulting correlation signals on the same figure using `subplot`.

- [2] `code1` and `code2`.
- [2] `code3` and itself.

(c) (Interpret a running correlation) When performing running correlation with a signal and itself, the resulting correlation signal has some special properties. Look at the correlation signal that you computed between `code3` and itself.

- [1] Is the correlation signal symmetric? (It can be shown that it should be.)
- [2] What is the maximum value of the correlation signal? How does the maximum value relate relate to the energy of `code3`?

3. (Using correlation to decode DSSS signals transmitted simultaneously with other signals.) Download the file `lab2_data.mat` and load it into your workspace. The file contains the variable which represents a received signal that is the sum of several message carrying signals, one from each of four users. The message carrying signal from each user conveys a sequence of bits using the direct-sequence spread-spectrum technique, described in Section 2.2.4, in which each bit is conveyed by sending a code signal or its negative. Each user has a different code signal. One of the code signals is the ten chip signal corresponding to the integer 170, while another is the six chip signal corresponding to the integer 25. The other two code signals are unknown to us. In this problem, we will try to extract the bit sequences conveyed by the known code signals from `dsss`. Start by generating the following code signals:

[8] If you cannot get your function working properly, you may use `run_corr_demo.dll` to complete the rest of the assignment.


```
>> cs1 = code_signal(170,10);
>> cs2 = code_signal(25,6);
```

(a) (Plot the signals) First, let's look at the signals we're given.

- [3] Use `subplot` and `stairs` to plot `dsss`, `cs1`, and `cs2` on three separate axes of the same figure.

(b) (Decoding the bits of the user with the longer code signal) Start by using `run_corr` to correlate the received signal `dsss` with the longer code signal `cs1`. Call the resulting signal `cor1`. Now, to decode the sequence of message bits from this user, we need to extract the appropriate samples from `cor1`. That is, we need to extract just those samples of the running correlation that correspond to the appropriate in-place correlations. We can do this in MATLAB using the following command:

```
>> sub_cor1 = cor1(length(cs1):length(cs1):length(cor1));
```

Each sample of `sub_cor1` is used to make the decision about one of the user's bits. When it is greater than zero, i.e. the correlation of the received signal with the code signal is positive, the decoder decides the bit is 1. When it is less than zero, the decoder decides the user's bit is 0.

- [3] On two subplots of the same figure, use `plot` to plot `cor1`, and `stem` to plot `sub_cor1`.
- [4] Decode the sequence of bits. (You can do this visually or with MATLAB) (Hint: The sequence is 10 bits long, and the first 3 bits are "011".)

(c) (Decoding the bits of the user with the shorter code signal) Repeat the procedure in a and b above, this time using the code signal `cs2`. Call your correlation signal `cor2`, and the vector of extracted values `sub_cor2`.

- [3] On two subplots of the same figure, use `plot` to plot `cor2`, and `stem` to plot the signal `sub_cor2`.
- [4] Decode the sequence of bits. (Hint: there are 17 bits in this sequence.)
- [2] Since the code signal `cs2` has less energy (because it is shorter), there is a greater chance of error. Are there any decoded bits that you suspect might be incorrect? Which ones? Why?

4. (Using running correlation to detect reflected radar pulses) `lab2_data.mat` also contains three other signals: `radar_pulse`, `radar_received`, and `radar_noise`. The received signal contains several reflections of the transmitted radar pulse and noise. The signal `radar_noise` contains noise with similar characteristics to the noise in the received signal without the reflected pulses.

(a) (Examining the radar signals) First, let's take a look at the first two signals.

- [2] Calculate the energy of `radar_pulse`, $E(x)$.
- [3] Use `subplot` to plot `radar_pulse` and `radar_received` in separate axes of the same figure.
- [1] Can you identify the reflected pulses in the received signal by visual inspection alone?

(b) (Perform running correlation) Use `run_corr` to correlate `radar_received` with `radar_pulse`.

- [3] Plot the resulting correlation signal.
- [2] Where are the received pulses? Visually identify sample locations of each pulse in the correlation signal.
- [2] Compare the heights of the peaks to the energy of the radar pulse and explain why the peaks are larger/smaller than the energy.

- [4] Given that the speed of light is 3×10^8 m/s and the sampling frequency of the detector is 10^7 samples per second, what is the approximate distance to each object[9]?

(c) (Estimating error rates, as in Section 2.2.5) In a real radar detector, the correlation signal would be compared to a threshold to perform the detection. To estimate the error rates for such a detector, let's consider a threshold that is equal to one-half the energy of the transmitted pulse, i.e. $c = E(x)/2$. Perform running correlation between `radar_pulse` and `radar_noise` call the resulting correlation signal `noise_c`.

- [2] Plot `noise_c`.
- [3] For how many samples is `noise_c` *greater* than this threshold? Use this value to estimate the false alarm rate.
- [3] For how many samples is `noise_c` *less* than this threshold minus the energy of the transmitted pulse? Use this value to estimate the miss rate.
- [2] What is the total error rate for this threshold?

(d) (Determining error rates from a histogram) As discussed in Section 2.2.5, we can use a histogram to judge the number of errors as well.

- [3] Plot the histogram of `noise_c` using 100 bins.
- [3] Describe how you could derive the error numbers in problem 4c from the histogram.

(e) (Setting the threshold to achieve a particular error rate) Suppose that detector false alarms are considered to be more serious than detector misses. Thus, we have determined that we want to raise the threshold so that we achieve a false alarm rate of approximately 0.004. Find a threshold that satisfies this requirement.

- [4] What is your threshold?
- [3] What is the false alarm rate on this noise signal with your threshold?
- [3] What is the miss rate on this noise signal with your threshold?
- [2] What is the total error rate for the new threshold? Compare this to the total error rate of the threshold used in problem 4c.

5. On the front page of your report, please provide an estimate of the average amount of time spent outside of lab by each member of the group.

[9] Remember that the radar pulse must travel to the object and then back again.

