UNIVERSITY OF UTAH
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

ECE1020 **COMPUTING ASSIGNMENT 3**
N. E. COTTER **MATLAB® ARRAYS: RECEIVED SIGNALS PLUS NOISE**

READING
Matlab® Student Version: learning Matlab 6, Ch 4.14-4.26
Mastering Matlab® 6, Ch 5

TOPICS
Creating and performing calculations with arrays
Noise power calculation

OVERVIEW
When our rover receives a signal from the base station antenna, it uses a correlation calculation to determine whether a one or zero is being sent. In this assignment you will create arrays containing signals with noise added to them, simulating what the robot vehicle would receive from the base transmitter.

In our simulation, we encode ones and zeros as short segments of sinusoids of two different frequencies—a low frequency signals a zero, while a high frequency signals a one. To keep matters simple, we use the following frequencies:
f_0 = 1.0 MHz (i.e., 1,000,000 Hz or 1,000,000 cycles per second)
f_1 = 2.0 MHz

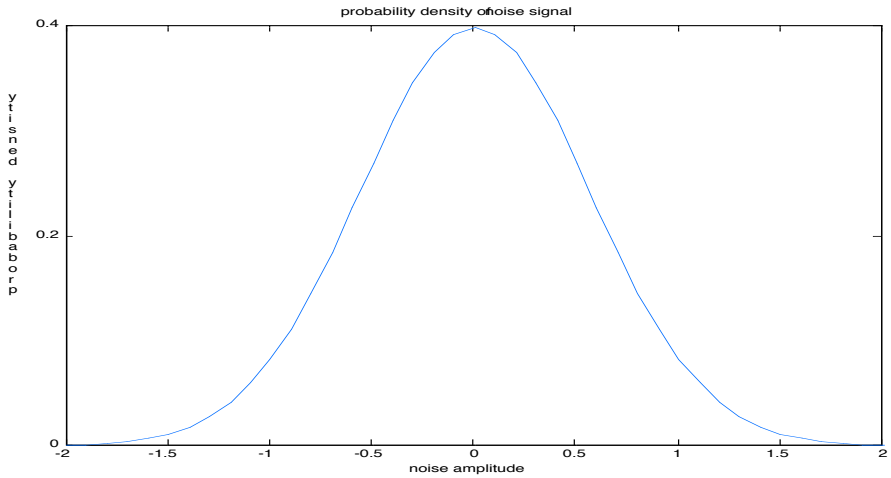
Each bit's waveform lasts 2 μsec, i.e. 0.000002 seconds, and its amplitude is 10 mV, i.e. 0.010 Volts. In the absence of noise, our signal would be a cosine wave:

$$s(t) = A\cos(2\pi ft)$$

where t ≡ time ($0 \le t \le 2$ msec)
$s(t)$ ≡ received signal
A ≡ amplitude of desired signal (0.010 V)
f ≡ frequency of desired signal (f_0 or f_1)

Unfortunately, environmental noise (from sunspots, for example) is added to the waveform for each bit. Because it is easy to analyze mathematically and is a reasonable representation of a noise signal, we use white noise. White noise is characterized by signal values that jump randomly from moment to moment. The average or mean value of the noise is zero—it is positive as often as it is negative.

The average power in the noise (or the average squared value of the noise) is called the variance. We will assume the variance is 1.0 V^2. If we collect many samples of white noise values, we find that they follow a gaussian or normal distribution. This is the so-called bell curve illustrated in the figure below. The probability that a given sample of the noise signal is between, say, 0 and 1 is equal to the integral of the area under the bell curve (or probability density function) between $t = 0$ and $t = 1$. As expected, the total area under the probability density function is equal to unity. This is equivalent to saying that we have 100 % probability of getting *some* value for a given noise sample.



Matlab commands used to make above figure:

```
x= -1.5:0.1:1.5
pdf = exp(-pi * x.^2);
plot(x,pdf,'-')
xlabel('noise amplitude')
ylabel('probability density')
title('probability density of noise signal')
axis([-1.5,1.5,0,1])
```

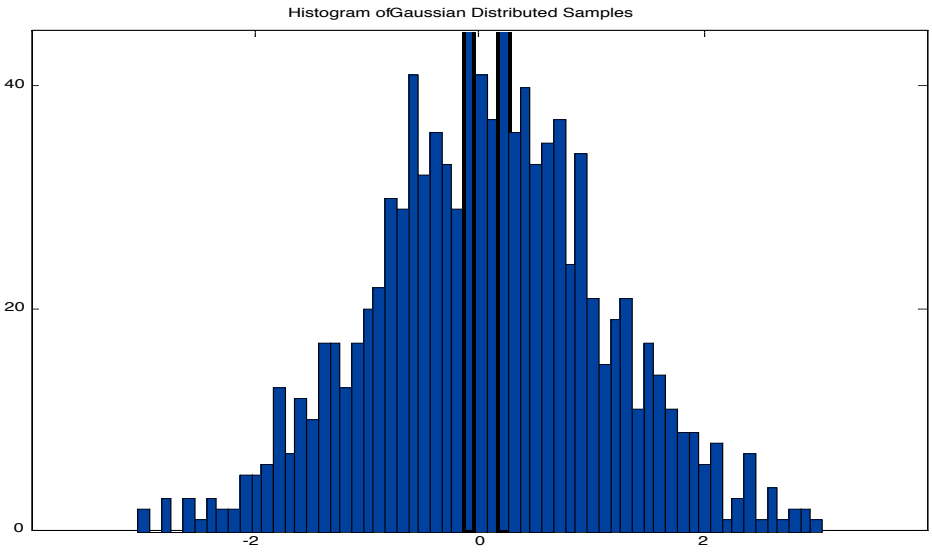
Tick marks edited by double clicking on axes in figure and using dialog windows.
Labels moved inside plot by clicking and dragging them on figure.

Fortunately, Matlab has a function called randn() (see *Mastering Matlab 6* Section 5.7) that creates arrays of gaussian or normally distributed numbers—as though we had samples from a noise signal. The variance of these random numbers is unity—just what we want. Consider the following Matlab session transcript:

```
>> randn(1,6)
ans =
   1.1909    1.1892   -0.0376    0.3273    0.1746   -0.1867
>> randn(1,6)
ans =
   0.7258   -0.5883    2.1832   -0.1364    0.1139    1.0668
```

Notice that the random numbers are different each time we use the randn() function. The two arguments of randn() are the number of rows and the number of columns of random numbers that will be produced. In this assignment, we will just require one row.

The plot below shows what happens when we generate many random values with randn() and then sort them into bins, according to their size. The result resembles the perfect bell curve shown above, but the curve is jagged because we have only a finite number of trials. We get the bell curve only if we average over infinitely many trials.



Matlab commands used to make above figure:

```
t = -3:0.1:3
y = randn(1000,1);
hist(y,t)
title('Histogram of Gaussian Distributed Samples')
```

Tick marks edited by double clicking on axes in figure and using dialog windows.

When we add the noise to the signal waveform we have the following equation:

$s(t) = A\cos(2\pi ft) + n(t)$

where t ≡ time ($0 \le t \le 2$ μsec)
$s(t)$ ≡ received signal
A ≡ amplitude of desired signal (0.01 V)
f ≡ frequency of desired signal (f_0 or f_1)
$n(t)$ ≡ gaussian noise added to desired signal (mean = 0 V, variance = 1 V^2)

The receiver's task is to determine whether $s(t)$ looks more like it arose from $f = f_0$ or from $f = f_1$. We use correlation to make this decision. First, we sample the $s(t)$ waveform at 0.1 μsec intervals for $0 \le t \le 2$ μsec. To generate an array of these sample times in Matlab, we would use the following command:

```
>>t = 1e-6 * (0:0.1:2.0)  % time array = beginning time : increment : final time
```

This gives us 21 equally spaced time values for t = 0.0 μsec, t = 0.1 μsec, ..., t = 2.0 μsec.

Our next step is to simulate the value of $s(t)$ at these sample times (see Procedure). Assume we have done this and created an array called svec. This array represents a signal plus noise. We next create arrays for signals without noise (see Procedure). These arrays represent perfectly clean signals, one called s0vec for $f = f_0$ and one called s1vec for $f = f_1$.

To decide whether the actual, noisy, received signal looks more like s0vec or s1vec, we compute the dot product of svec with s0vec and the dot product of svec with s1vec. The dot product is the mathematical way of computing how similar two waveforms are, something we can do visually when the noise is small. The dot product of two arrays is the sum of an element by element multiply. For example, the dot product of vectors x

and y is the first element of x times the first element of y, plus the second element of x times the second element of y, and etc. Matlab has a convenient operator called .* and the sum() function that serve this purpose. (Thus, there is more than one way to the same thing.) Alternatively, we can multiply the first vector by the transpose of the second vector. The transpose of a vector means that a horizontal vector becomes a vertical vector or vice versa. (This is accomplished by the apostrophe operator in Matlab. s0vec' is the transpose of s0vec.)

Whichever dot product is bigger determines which signal we decide was sent. Because of the noise, we sometimes make mistakes. In this assignment, you will estimate how often we make these mistakes. This will help determine how much transmitter power is needed in order to overcome noise. Later on, we will explore ways to correct errors, (just as your cell phone does).

PROCEDURE

In this assignment, you will determine whether a signal represents a zero or one.

+5 pts Script File for calculations

Using a text editor program on your PC, create a script file called **sdecide.m** containing matlab commands to perform the calculations in the first part of this assignment.

+5 pts Do **not** use semicolons at the ends of commands in your script files.

+5 pts Array of Sample Times

Using the colon operator, create an array called t containing 21 time values starting at 0.0 μsec, spaced by 0.1 μsec, and ending with 2.0 μsec.

(Note: To check your values, use plot commands such as plot(t) to see if the waveforms have the shape you would expect.)

+20 pts Simulated Received Signals

Using the array of sample times and vector processing, create arrays (containing 21 values) of the following signal waveforms for zero and one without noise. Call the arrays s0vec and s1vec.

$$s_0(t) = A\cos(2\pi f_0 t)$$

$$s_1(t) = A\cos(2\pi f_1 t)$$

where t ≡ time ($0 \le t \le 2$ μsec)
$s(t)$ ≡ received signal
A ≡ amplitude of desired signal (0.01 V)
f_0 ≡ frequency of zero (1 MHz)
f_1 ≡ frequency of one (2 MHz)

+5 pts Simulated Noise Signal

Using the randn() function, create an array called nvec having one row of 21 random values drawn from a gaussian (or normal) distribution.

+5 pts Simulated Signal Plus Noise
Add s0vec and nvec to create svec.

+10 pts Dot product of signal with waveform for zero
Calculate r0 = svec dot s0vec by multiplying the entries of svec and s0vec element by element and then operating on the result with the sum() function.

+5 pts Dot product of signal with waveform for one
Calculate r1 = svec dot s1vec by multiplying svec by the transpose of s1vec.
This completes **sdecide.m**

+5 pts Diary Command
When you have finished typing the script file for the above equations, delete the current diary file and start a new diary.
>>diary on % turns on diary

Change the working directory to the c: drive where you stored your script file.
This an alternative to the addpath command if everything you write is stored in one directory.
>> cd c:\

+10 pts Run Script File 10 Times
Run your script file ten times by typing the name of the file without the .m
>> sdecide
If you make any changes in your sdecide.m file, be sure to run the following Matlab command to insure that Matlab reads your file again the next time you run it:
>> clear all

Each time you run the file successfully, record which of r0 and r1 is larger. If r0 is larger we interpret the transmitted signal as a zero. If r1 is larger we interpret the transmitted signal as a one.

+5 pts Create Array of Received 0's or 1's
In a new script file called **tally.m**, create an array called recv01 containing the ten values, 0 or 1, that you recorded in the previous step.

+5 pts Count Number of 1's Received
In tally.m, use the sum function to count the number of ones in recv01. Put the result in a variable called nerr. Since we started with the waveform for a zero and added noise to it every time we ran sdecide, the ones are errors. Thus, nerr is the number of errors we had in ten trials.

+10 pts Select Results of Odd-Numbered Trials
For the sake of illustration, suppose we have reason to believe that there is a pattern to the noise. In particular, we wish to extract odd-numbered trials to see if they are more likely to have errors. We proceed as follows.

In tally.m, use the colon operator to create an array called odd_array containing odd numbers 1,3,5,7, and 9. Then use odd_array as the index of recv01 to extract the results of odd-numbered trials. Place the result in a variable odd_results.

By counting the number of ones in odd_results we could determine how often the odd trials produce errors. (You are only required to calculate the variable called odd_results and not the number of ones in it, however.)

+5 pts End of Diary
>> diary off % Close the diary file. Look for the diary in e.g., c:\matlab\work directory.

E-mail your script file (sdecide.m), your results file (tally.m), and your diary file to your TA, (as three separate e-mails). In the Subject line of your e-mail, be sure to put Your Name, "ECE1020 Comp3," and the file name, (e.g. sdecide.m). Also, print out the files and hand them in to the TA or the ECE1020 locker.