# 13 Rec 13: Miscellaneous Random Process Topics

**Directions**: Your instructor will spend the the first 40 minutes of the recitation period working some review problems and going over one or more Matlab experiments in the following. During the last 10 minutes of recitation, your proctor will give you a "Lab Form" that your recitation team completes, signs, and turns in. See the last page for an indication of what you will be asked to do on the Lab Form.

Due to time limitations, only a part of the following can be covered during the recitation period. However, you might want in the future to try some of the uncovered experiments on your own. They could give skills useful on some future homework problems and could lend insight into your understanding of the course from an experimental point of view.

**This Week's Topics.**

- Levinsion-Durbin Predictor Design Algorithm
- Simulation/Stability of Single-Server Queue
- Power Computation Via Power Spectral Density
- Introduction to Spectral Factorization
- Gaussian White Noise/Brownian Motion Realizations

## 13.1 Exp 1: Levinson-Durbin Predictor Design Algorithm

In Recitation 12, you learned how to design a linear predictor of any order, with emphasis on the first, second, and third order predictor. On Homework 10, you separately designed the best first order predictor, the best second order predictor, and the best third order predictor for a certain WSS discrete-time process $X_n$. ("Best" predictor means that the predictor achieved minimum mean square prediction error.)

In this experiment, I have you play with the so-called *Levinson-Durbin Algorithm*, which miraculously, for any positive integer $k$, simultaneously designs all $j$-th order linear predictors for $1 \leq j \leq k$, given the autocorrelation function of a WSS process $X_n$.

*Example 1.* In Problem 2 on Homework 10, you designed predictors for a WSS process $X_n$ whose autocorrelation function satisfies

$$R_X(\tau) = \begin{cases} 8, & \tau = 0 \\ -4, & \tau = \pm 1 \\ 1, & \tau = \pm 2 \\ 0, & elsewhere \end{cases}$$

Run the following Matlab script, which is an implementation of the Levinson-Durbin algorithm for the above autocorrelation function:

```
clear
RX=[8 -4 1 zeros(1,50)]; %enter in enough autocorr values
k=6;            %enter in max order of predictor you want
RX =  toeplitz(RX(1:k+1)); %creates correlation matrix
d(1)=RX(1,1);
A(1,1)=RX(1,2)/d(1);
for i=1:k-1
d(i+1)=d(i)*(1-A(i,i)^2);
u=RX(1,i+1:-1:2);
v=A(i,1:i);
A(i+1,i+1)=(RX(1,i+2)-u*v')/d(i+1);
for j=1:i
A(i+1,j)=A(i,j)-A(i+1,i+1)*A(i,i+1-j);
end
end
A
```

What you see on your computer screen is a $6 \times 6$ matrix. The first $i$ entries in row $i$ ($i = 1, 2, 3, 4, 5, 6$) are the predictor coefficients for the $i$-th order predictor. If you look at the first three rows, these should coincide with the predictor coefficients given in the solutions to Problem 2, Homework 10, on the Web, for the first, second, and third order predictors. If you could run the Levinson-Durbin algorithm for $k = \infty$, then you would discover the following as you look at the generated matrix whose rows yield the predictor coefficients for every single finite order linear predictor: as you go down each column, the predictor coefficients converge. What these columns converge to is the "IIR predictor", the predictor that uses all of the previous observations to predict what is going to happen next. This would be the very best of all linear predictors, regardless of order.

*Example 2.* Now let the autocorrelation function be

$$R_X(\tau) = \begin{cases} 12, & \tau = 0 \\ 8, & \tau = \pm 1 \\ 3, & \tau = \pm 2 \\ 1, & \tau = \pm 3 \\ 0, & elsewhere \end{cases}$$

Use the Levinson-Durbin script in Example 1 to find the predictors of order 1 thru 7. (In the second line of the script, you enter an RX vector which gives at least the first eight entries of the autocorrelation function; in the third line of the script, you enter the maximum order k=7 that you are allowing. In general, in Line 2, you need to enter in the vector of $k + 1$ entries

$$R_X(0), R_X(1), \cdots, R_X(k)$$

if your maximum predictor order is $k$.

*Example 3.* Now use the Levinson-Durbin algorithm to generate the predictors of the first few orders for the autocorrelation function

$$R_X(\tau) = 10(1/2)^{|\tau|}.$$

Do the rows of the generated matrix look kind of strange? What do you think is happening here? (Note: If you are not seeing something "strange", you possibly did something wrong, and you should ask your recitation instructor to help you.)

## 13.2 Exp 2: Simulation/Stability of Single-Server Queue

We are going to examine the "single-server queueing system model". You can conceptualize this system via the block diagram

$$\text{arrivals} \rightarrow \boxed{\text{server}} \rightarrow \text{departures}$$

To help you fix the ideas in your mind, you can think of think of the queueing system in the following way:

- Think of the "arrivals" as message packets arriving at different random times. These arrival times are modeled by a Poisson process with an averate rate of $\lambda$ arrivals per second. Each message packet, upon arrival to the system, goes to the end of a queue, and is not processed by the system server until it reaches the beginning of the queue.
- Think of the "server" as an e-mail server or router which receives each message packet and then processes it when the packet reaches the front of the queue. The server is assumed to process packets at the rate of $\mu$ packets per second.
- Think of the "departures" as the message packets leaving the system at various times after being processed by the server.

Each message packet has an "arrival time" and a "departure time". These two times are related by the formula:

$$\text{departure time} = (\text{arrival time}) + (\text{waiting time}) + (\text{service time})$$

The "waiting time" is the length of time that it takes for the packet to move to the front of the queue, and the "service time" is the length of time that it takes for the packet to be processed by the server.

We want to use Matlab to simulate arrival times, waiting times, service times, and departure times. Here is how we can do it:

- The first packet arrives at time

  ```
  arrivaltime(1)=-log(rand(1,1))/lambda;
  ```

- The waiting time for the first packet is then

  ```
  waitingtime(1)=0;
  ```

  This is because the queue is empty when the first customer arrives.

- The service time for the first packet is

```
servicetime(1)=-log(rand(1,1))/mu;
```

- The departure time for the first packet is therefore

```
departuretime(1)=arrivaltime(1)+waitingtime(1)+servicetime(1);
```

- The arrival time for the second packet is computed as:

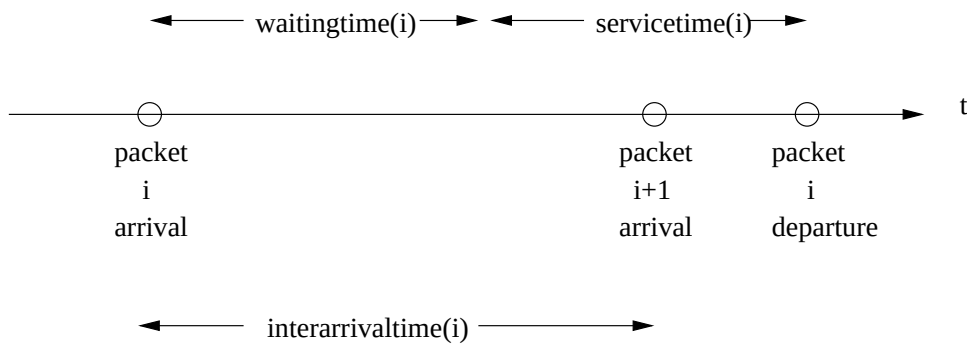
```
interarrivaltime(1)=-log(rand(1,1))/lambda; %time between packets 1 and 2
arrivaltime(2)=arrivaltime(1)+interarrivaltime(1);
```

- The waiting time, service time, and departure time for the second packet are:

```
waitingtime(2)=max(0,waitingtime(1)+servicetime(1)-interarrivaltime(1));
servicetime(2)=-log(rand(1,1))/mu;
departuretime(2)=arrivaltime(2)+waitingtime(2)+servicetime(2);
```

- The arrival time, waiting time, service time, and departure time for the third packet would then be Matlab simulated as follows:

```
interarrivaltime(2)=-log(rand(1,1))/lambda;
arrivaltime(3)=arrivaltime(2)+interarrivaltime(2);
waitingtime(3)=max(0,waitingtime(2)+servicetime(2)-interarrivaltime(2));
servicetime(3)=-log(rand(1,1))/mu;
departuretime(3)=arrivaltime(3)+waitingtime(3)+servicetime(3);
```




Continuing in this way, we can simulate the arrival time, service time, waiting time, and departure time of each message packet. Using the preceding figure, one can show that the waiting times are generated recursively by the following equation in Matlab syntax:

```
waitingtime(i+1)=max(0,waitingtime(i)+servicetime(i)-interarrivaltime(i));
```

For the scenario in the figure, it is clear the packet $i+1$'s waiting time is

```
waitingtime(i)+servicetime(i)-interarrivaltime(i)
```

On the other hand, the departure of packet $i$ might occur before packet $i+1$ arrives. In this case, the waiting time for packet $i+1$ is zero and

```
waitingtime(i)+servicetime(i)-interarrivaltime(i) < 0
```

so that

```
waitingtime(i+1)=max(0,waitingtime(i)+servicetime(i)-interarrivaltime(i));
```

gives the correct waiting time in all cases.

**Stability of the Queue.** Does the expected length of the queue remain bounded as time $t$ goes to infinity? This is called a *stable queue*. Or, does the expected length of the queue blow up as $t \to \infty$? This is an *unstable queue*. Using mathematics that is beyond the scope of EE 3025, one can establish the following result concerning stability:

**Case 1: Stable Queue.** The single-server queueing system is stable if $\lambda < \mu$ (that is, the arrival rate is less than the service rate).

**Case 2: Unstable Queue.** The single-server queueing system is unstable if $\mu \leq \lambda$.

We are now going to do a Matlab simulation that will suggest to you that the preceding result is true. Our simulation will involve two staircase functions called `In(t)` and `Out(t)`. The function `In(t)` is the realization of the Poisson arrival process: the value of `In(t)` at each time $t \geq 0$ is the number of arrivals that have taken place up to and including time $t$. If you look back at Experiment 5 of Recitation 11, you will see Matlab code for simulating and plotting the function `In(t)`. At each time $t \geq 0$, the function `Out(t)` is defined to be the number of departures from the queueing system that have taken place up to and including time $t$. When you plot the function `In(t)` and the function `Out(t)` on the same set of coordinate axes, you will see that the staircase given by `In(t)` lies above the staircase given by `Out(t)`, that is,

$$\texttt{In(t)} \geq \texttt{Out(t)}$$

It is the gap between these two staircase plots that determines stability. The difference is

```
In(t)-Out(t)
```

which has the interpretation of being the length of the queue as a function of time $t$. You either have:

**Case 1: Stable Queue.** E[`In(t)-Out(t)`] remains bounded as $t \to \infty$. (That is, on average, the gap between the two plots can only become so large and no larger. Equivalently, the expected length of the queue is leveling off with time.)

**Case 2: Unstable Queue.** E[`In(t)-Out(t)`] blows up as $t \to \infty$. (That is, on average, the gap between the two plots is getting bigger and bigger with time. Equivalently, the expected length of the queue is blowing up with time.)

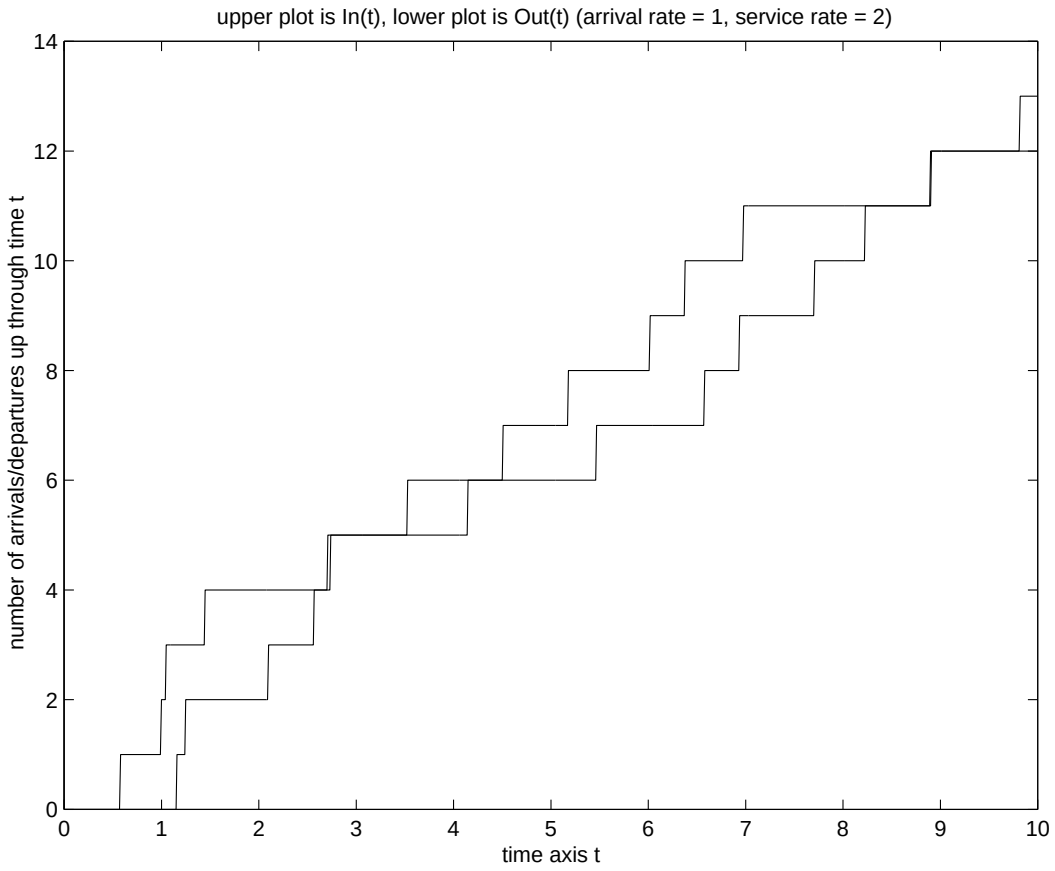
We will first do a simulation of the queueing system with $\lambda < \mu$ to see that Case 1 holds, and then we will do a simulation with $\lambda \geq \mu$ to see that Case 2 holds.

*Example 4.* In this example, we take the arrival rate to be $\lambda = 1$ arrivals/second and the service rate to be $\mu = 2$ packets/second. Since $\lambda < \mu$, our queueing system will be stable. Running the following Matlab script, you simulate the queueing system from time $t = 0$ to time $t = 10$, and obtain plots of `In(t)` and `Out(t)` on the same set of axes:

```
lambda=1; mu=2;
T=10;
n=1000;
interarrivaltimes=-log(rand(1,n))/lambda;
servicetimes=-log(rand(1,n))/mu;
w(1)=0;
for i=2:n;
w(i)=max(0,-interarrivaltimes(i)+servicetimes(i-1)+w(i-1)); end
arrivaltimes=cumsum(interarrivaltimes);
waitingtimes=w;
departuretimes=arrivaltimes + waitingtimes + servicetimes;
t=0:.01:T;
for i=1:length(t)
count1(i)=max(round(100*(departuretimes-t(i)))==0);
count2(i)=max(round(100*(arrivaltimes-t(i)))==0); end
Out=cumsum(count1);
In=cumsum(count2);
plot(t,In,t,Out)
```

Looking at the gap between the two staircase functions in the preceding figure, we see that at any time, the length of the queue appears to either 0, 1, or 2. The length of the queue does not appear to be blowing up, and so the system appears to be stable. To make the result more convincing, we obtained the following In(t) versus Out(t) plot when the time axis is expanded to go from $t = 0$ to $t = 100$ (change the second line of the Matlab script to `T=100;`):

*Example 5.* In this example, we examine the unstable queue in which the arrival rate is $\lambda = 2$ and the service rate is $\mu = 1$. We simply changed the first line of code in the Example 4 Matlab script to obtain the plots of In(t) and Out(t):




The two plots clearly seem to be diverging from one another. This does indeed suggest that we have an unstable queue.

## 13.3 Exp 3: Power Computation Via Power Spectral Density

Let $X_n$ be a discrete-time WSS process. The *power spectral density* $S_X(f)$ of this process is the discrete-time Fourier transform of the autocorrelation function $R_X(\tau)$. The power $P_X$ generated by the $X$ process can be computed as follows:

$$P_X = \int_0^1 S_X(f)df. \tag{1}$$

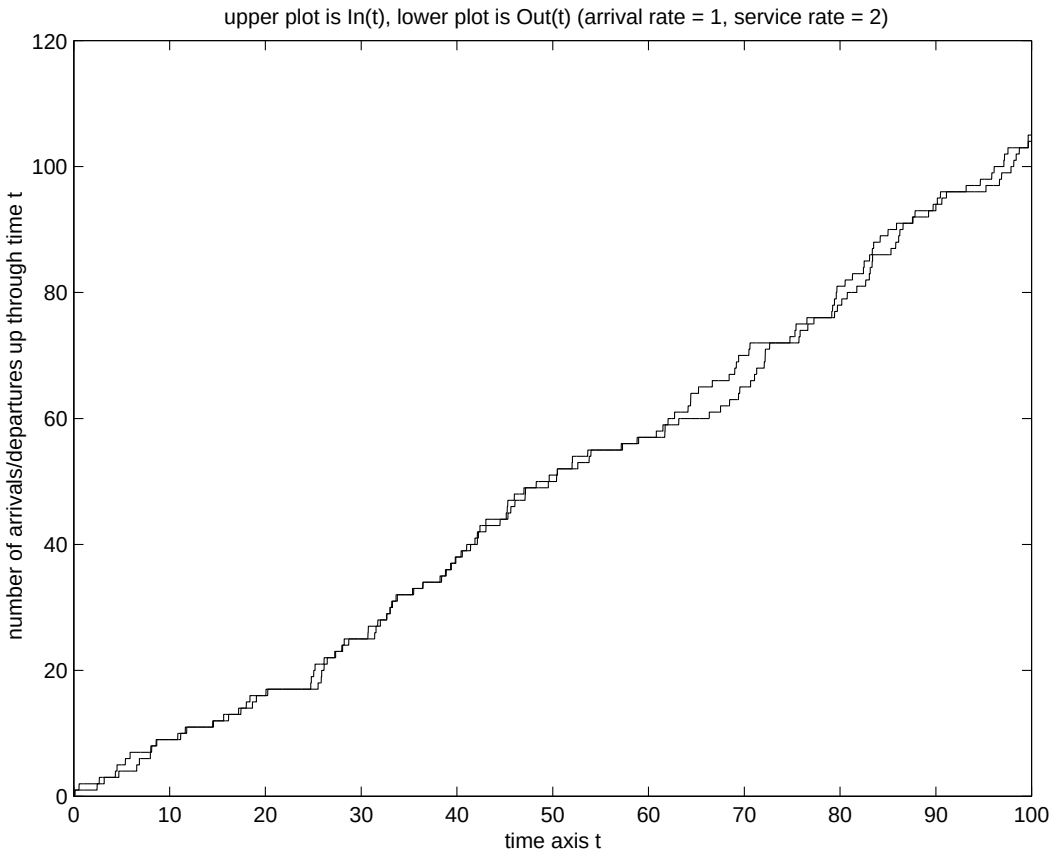
The following example illustrates this fact.

*Example 6.* Let $Z_n$ be the Gaussian white noise process with unit variance. Let $X_n$ be the process defined by filtering the random signal $Z_n$ as follows:

$$X_n = (0.5)X_{n-1} + (0.5)Z_n \tag{2}$$

You are going to compute the power $P_X$ generated by the $X$ process using $S_X(f)$. First you have to find $S_X(f)$. In Chapter 11, you learn that

$$S_X(f) = |H(f)|^2 S_Z(f) = |H(f)|^2,$$




where $H(f)$ is the frequency response function of the filter given by (2), and you are using the fact that $S_Z(f) = 1$ for all frequencies $f$. Fourier transforming the equation (2), you get the equation

$$X(f) = (0.5)\exp(-j2\pi f)X(f) + (0.5)Z(f).$$

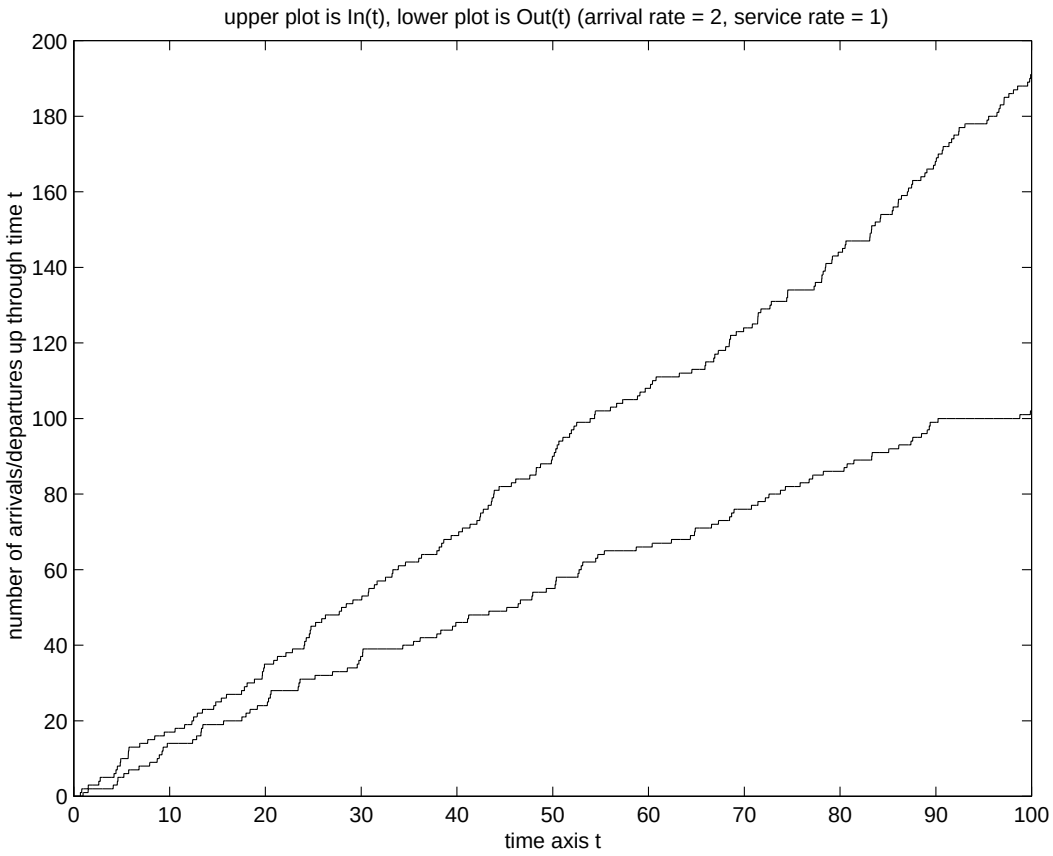
- Use pencil and paper to argue that the frequency response $H(f) = X(f)/Z(f)$ is given by
$$H(f) = \frac{1}{2 - \exp(-j2\pi f)}.$$

- Use pencil and paper to do the algebraic manipulations involved in computing $|H(f)|^2$, showing that
$$S_X(f) = |H(f)|^2 = \frac{1}{5 - 4\cos(2\pi f)}.$$

- Run the following Matlab code to compute $P_X$ according to the formula (1):

```
syms f
PX=eval(int(1/(5-4*cos(2*pi*f)),f,0,1))
```

Here is a way you can check your work. In the course of covering Chapter 11, we will show that since the $Z$ process is white noise, then we have the following formula relating $P_X$ and




$P_Z$:

$$P_X = P_Z[\sum_{n=-\infty}^{\infty} h[n]^2], \tag{3}$$

where $h[n]$ is the impulse response of the filter given by (2). (Warning: Don't use this formula unless $Z$ is white!) In our case here, $h[n]$ takes the form

$$h[n] = A(1/2)^n u[n]$$

for some constant $A$. What is $A$? (You can find out from studying equation (2).) In equation (3), sum the geometric series on the right hand side, and use the fact that $P_Z = 1$ to deduce what the value of $P_X$ is. Did you obtain the same value for $P_X$ as found using formula (1)?

## 13.4 Exp 4: Introduction to Spectral Factorization

We consider the simplest possible filter design problems that can be solved by a technique called *spectral factorization*. Let $Z_n$ be a discrete-time white noise process with unit variance. Suppose we filter the random signal $Z_n$ using a discrete-time causal stable linear time-invariant filter with transfer function $H(z)$:

$$Z_n \rightarrow \boxed{H(z)} \rightarrow X_n$$

As indicated in the block diagram, the filter output is WSS random signal $X_n$. We want to design the filter (that is, find $H(z)$) so that the autocorrelation function $R_X(\tau)$ is the following:

$$R_X(\tau) = \begin{cases} 5, & \tau = 0 \\ 2, & \tau = \pm 1 \\ 0, & elsewhere \end{cases}$$

I will guide you through the following steps for accomplishing this goal.

**Step 1:** We take the $z$ transform of $R_X(\tau)$. We get $S_X(z)$, the power spectral density of $X$ process in $z$ domain:

$$S_X(z) = 5 + 2z + 2z^{-1}.$$

Do you understand that the preceding equation gives the $z$ transform of $R_X(\tau)$? If you don't understand, ask your recitation instructor.

**Step 2:** This step is a "root finding" step. Before I tell you what to do, you need some background. From a result we will get in Chapter 11, we have

$$S_X(z) = S_Z(z)H(z)H(z^{-1}).$$

In our case, since the input is white, we have $S_Z(z) = 1$, and this equation becomes

$$5 + 2z + 2z^{-1} = H(z)H(z^{-1}).$$

Let us attempt a solution of this equation of the form $H(z) = a + bz^{-1}$ for some unknown real constants $a, b$. We then have

$$5 + 2z + 2z^{-1} = (a + bz^{-1})(a + bz). \tag{4}$$

For Step 2, write the left side as

$$\frac{2z^2 + 5z + 2}{z},$$

(by factoring out a $z^{-1}$), and then find the two roots of the polynomial $2z^2 + 5z + 2$ by executing the Matlab line

```
roots([2 5 2])
```

Look at the two roots you see on your computer screen. Are they real? Are they reciprocals of one another?

**Step 3:** Pick either root from Step 2 and call it $r$. Write the equation (4) as

$$5 + 2z + 2z^{-1} = c(1 - rz^{-1})(1 - rz),$$

where $c$ is an unknown positive constant. For Step 3, multiply out the right side of the preceding equation and compare it to the left side in order to figure out what $c$ is.

**Step 4:** Take your filter transfer function as

$$H(z) = \sqrt{c}(1 - rz^{-1}).$$

*Example 7.* Suppose we now want to design the filter transfer function $H(z)$ so that the filter output power spectral density is

$$S_X(f) = \frac{1}{7 - 4\cos(2\pi f)}.$$

Design $H(z)$ as a causal filter so that this will be true. I will get you started. In $z$ domain, $\cos(2\pi f)$ becomes

$$\cos(2\pi f) = (1/2)[\exp(2\pi f j) + \exp(-2\pi f j) = (1/2)[z + z^{-1}].$$

Therefore, the power spectral density in $z$ domain is

$$S_X(z) = \frac{1}{7 - 2z - 2z^{-1}}.$$

Factor $7 - 2z - 2z^{-1}$ according to the method in Steps 2-3 above. You will obtain something of the form

$$7 - 2z - 2z^{-1} = c(1 - rz^{-1})(1 - rz).$$

Then take

$$H(z) = \frac{1}{\sqrt{c}(1 - rz^{-1})}.$$

In Step 2, the "root finding step", you will have two choices for the root $r$. Be sure to choose the one that makes the filter stable.

## 13.5 Exp 5:Gaussian White Noise/Brownian Motion Realizations

In this experiment, you see how to simulate realizations of the continuous-time Gaussian white noise process (GWN process), and the Brownian motion process (also called Wiener process).

The GWN process $X(t)$ satisfies

$$R_X(\tau) = A\delta(\tau) \tag{5}$$

for some positive constant $A$. Because the delta function blows up at $\tau = 0$, the GWN process has infinite power. Therefore, the GWN process is not really physically realizable. But, it can be approximated (because a delta function can be approximated as a rectangular pulse with very high amplitude and very short duration). The following example uses Matlab to create plots of realizations of a process that is approximately GWN.

*Example 8.* In this example, you simulate realizations of the GWN process $X(t)$ with autocorrelation function (5). The basic idea behind simulating a realization of $X(t)$ for $0 \le t \le T$ (where $T$ is a positive integer) goes as follows: Pick a large positive integer $n$, and form a vector `x` of $n + 1$ independent pseudorandom samples from a gaussian distribution with mean 0 and variance $A/\Delta$, where $\Delta = T/n$. Then, form a "time axis", which is a vector `t` consisting of $n + 1$ equally spaced entries from 0 to $T$ inclusively; executing the command "`plot(t,x)`" then gives the desired GWN realization. Run the code:

```
A=1; T=10; n=10000;
Delta=T/n;;
t=0:Delta:T;
white_noise=sqrt(A)*Delta^(-0.5)*randn(1,length(t));
plot(t,white_noise)
title('Gaussian white noise realization')
```

If you pass GWN through an integrator, you obtain the Brownian motion process:

$$\text{GWN} \rightarrow \boxed{\int_0^t} \rightarrow X(t) = \text{ Brownian motion process}$$

Even though GWN is physically unrealizable, the Brownian motion process is physically realizable. In fact, the realizations of the Brownian motion process are continuous functions of $t$. The following example allows you to use Matlab to plot realizations of a Brownian motion process.

*Example 9.* In obtaining Brownian motion process from integration of GWN, you can approximate the continuous-time integrator by a discrete-time integrator implemented by the "`cumsum`" command in Matlab. Run the following script which will generate the plot of a Brownian motion process realization:

```
A=1; T=10; n=10000;
Delta=T/n;
t=0:Delta:T;
x=Delta^(-0.5)*randn(1,n);
w=[0 cumsum(Delta*x)];
plot(t,w)
title('Brownian motion process realization')
```

The factor of $\Delta$ in the discrete-time integrator is because

$$\int_{(i-1)\Delta}^{i\Delta} x(t)dt \approx \Delta x(i\Delta).$$

Although Brownian motion process realizations are continuous, they are also nowhere differentiable! This is why your plot might look find of funny. In fact, even if you put the realization curve under a microscope, it will still look "spiky". (Have you heard of fractals? Brownian motion process realizations are fractals—they are not 1-D curves but rather have a dimension somewhere between 1-D and 2-D.)

The Brownian motion process is very important for applications. For example, there is an extension of the Brownian motion process to 2-D that can be used to solve Laplace's equation

$$\partial^2 V/\partial x^2 + \partial^2 V/\partial y^2 = 0$$

in a bounded region of the xy-plane, in which a boundary condition is placed on the boundary of the region. (To obtain $V(x, y)$ at an interior point $(x, y)$ of the region, you simply start the 2-D Brownian motion process there and let it run until it hits the boundary. The expected boundary value is equal to $V(x, y)$.)

EE 3025 Recitation 13 Lab Form

- NAME AND ID NUMBER OF TEAM MEMBER 1:
- NAME AND ID NUMBER OF TEAM MEMBER 2:
- NAME AND ID NUMBER OF TEAM MEMBER 3:

Let $Z_n$ be a white noise process with unit variance. In this report, you find filter coefficients $A, B$ so that the filtering operation

$$X_n = AZ_n + BZ_{n-1} \tag{6}$$

yields WSS process $X_n$ satisfying

$$R_X(0) = 6, \quad R_X(\pm 1) = 1, \quad R_X(\tau) = 0 \text{ elsewhere.}$$

**(a)** The desired $A, B$ must satisfy the equations

$$\begin{aligned} A^2 + B^2 &= 6 \\ AB &= 1 \end{aligned}$$

Run the following Matlab script which finds a solution for $A, B$.

```
syms a b
[a,b] = solve('a^2+b^2=6' , 'a*b=1');
A=double(a(1))
B=double(b(1))
```

Fill in the blanks for $A, B$ at right (four decimal places).

**(b)** Using pencil and paper and high school algebra, verify that

$$(A + Bz^{-1})(A + Bz) = 6 + z + z^{-1} \tag{7}$$

Write down your work below (or on the back if you run out of room).

**(c)** You will learn in Chapter 11 of text that the "spectral factorization" in equation (7) means that the filtering operation (6) will give us the desired autocorrelation function $R_X(\tau)$. Here you do a Matlab simulation to verify this. Run the script:

```
A =      ;   B =      ;  %enter the A,B values you found in (a)
n=1000000;
z=randn(1,n); %white noise inputs to filter
x=A*z(2:n) + B*z(1:n-1); %filter outputs
RX0_hat = mean(x.^2)
RX1_hat = mean(x(1:n-2).*x(2:n-1))
```

Write down the autocorrelation estimates (four decimal places) yielded by Matlab: