4 Rec 4: Means/Variances and Applications

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.

- Verification of Means/Variances of Common Distributions
- Application to Monte Carlo Integration
- Application to Quantizer Design
- Application to Text Compression

4.1 Exp 1: Verification of Means/Variances

In Recitation 2, you saw how to simulate observations from a uniform, Gaussian, binomial, Poisson, and exponential distribution. Each of these distributions has a theoretical mean μ and variance σ^2 given in Appendix A. In this experiment, we verify that the Appendix A expressions for μ and σ^2 are correct by applying the Matlab functions "`mean`" and "`var`" to a large number of simulated observations, respectively.

- For a vector `x` of observations $(x_i : i = 1, 2, \cdots, n)$ from a prob distribution with mean μ, `mean(x)` is the arithmetic average

$$\frac{\sum_{i=1}^{n} x_i}{n}.$$

  It should provide a good estimate of μ if number of observations n is large.

- The variance of the observations is `var(x)` and is almost[1] the same thing as

$$\frac{\sum_{i=1}^{n}(x_i - \text{mean}(\mathbf{x}))^2}{n},$$

[1] Actually, the variance `var(x)` of observations $\mathbf{x} = (x_i : i = 1, \cdots, n)$ is $\frac{\sum_{i=1}^{n}(x_i - \text{mean}(\mathbf{x}))^2}{n-1}$ instead of dividing by n. We explain later in this course why some people divide by $n-1$ instead of n. For large n, it makes very little difference.

the average square deviation of each observation from `mean(x)`. `var(x)` should be a good estimate of σ^2, the variance of the probability distribution, if number of observations n is large.

Example 1: The Uniform(a, b) distribution has mean and variance

$$\begin{aligned} \mu &= \frac{a+b}{2} \\ \sigma^2 &= \frac{(a-b)^2}{12} \end{aligned}$$

Suppose $a = 3$ and $b = 7$. Then the mean of the Uniform(a, b) distibution should be 5 and the variance should be $16/12 = 4/3$. Run the following Matlab code to verify this:

```
a=3; b=7;
x=(b-a)*rand(1,100000)+a; %simulated Uniform(a,b) observations
mean(x) %estimate mean of distribution
var(x) %estimate variance of distribution
```

Are your mean and variance estimates approximately correct?

Example 2: The Gaussian(μ, σ) distribution has mean μ and variance σ^2. Run the following Matlab script, which verifies that we do indeed have this mean and variance for $\mu = 4$, $\sigma = 3$ (which means the variance is $\sigma^2 = 9$).

```
mu=4; sigma=3;
x=sigma*randn(1,100000)+mu; %simulated Gaussian(mu,sigma) observations
mean(x) %estimate mean of distribution
var(x) %estimate variance of distribution
```

Are your mean and variance estimates approximately correct?

Example 3: The Binomial(n, p) distribution has mean and variance

$$\begin{aligned} \mu &= np \\ \sigma^2 &= np(1-p) \end{aligned}$$

We verify this with $n = 6$, $p = 1/3$, in which case the mean is $np = 2$ and the variance is $np(1-p) = 12/9 = 1.333$. See what happens when you run the following Matlab script.

```
n=6; p=1/3;
x=sum(rand(n,50000)>1-p); %simulated Binomial(n,p) observations
mean(x) %estimate mean of distribution
var(x) %estimate variance of distribution
```

Are your mean and variance estimates approximately correct?

Example 4: A curious property of the Poisson(α) probability distribution is that its mean and variance are both α! In the following script you test this for $\alpha = 0.5$.

```
clear;
alpha=0.5;
%generate 10000 Poisson(alpha) samples
for i=1:10000
t=0;
count=-1;
while t<1
t=t-log(rand(1,1))/alpha;
count=count+1;
end
x(i)=count;
end
mean(x) %estimate mean of distribution
var(x) %estimate variance of distribution
```

Are your mean and variance estimates approximately correct?

Example 5: The Exponential(a) probability distribution has mean $1/a$ and variance $1/a^2$. Let us take $a = 1/70$. Then the mean should be 70 and the variance should be 4900. Test this with the following Matlab script.

```
clear;
a=1/70;
x=-log(rand(1,50000))/a; %simulated Exponential(a) samples
mean(x) %estimate mean of distribution
var(x) %estimate variance of distribution
```

Are your mean and variance estimates approximately correct?

4.2 Exp 2: Application to Monte Carlo Integration

Monte Carlo Integration is a technique via which you approximate the integral of a function over an interval by making use of pseudorandomly generated points in that interval. Specifically, suppose we want to evaluate the integral

$$\int_a^b g(x)dx.$$

We can re-write this integral as

$$(b-a)\int_a^b g(x)\left(\frac{1}{b-a}\right)dx.$$

Note that the quantity $\frac{1}{b-a}$ in the integrand is the Uniform(a,b) probability density, and so we can interpret this integral as $E[g(X)]$, where X is a RV uniformly distributed in the interval $[a,b]$. We have proved the formula

$$\int_a^b g(x)dx = (b-a)E[g(X)], \quad X \sim \ Uniform(a,b). \tag{1}$$

To estimate $E[g(X)]$, you can perform the following three steps:

Step 1: Generate a large number of pseudorandom observations of $X \sim \text{Uniform}(a, b)$.

Step 2: Apply function g to transform the observations from Step 1.

Step 3: Applying Matlab function `mean` to the transformed observations, which gives the estimate for $E[g(X)]$.

Having found your estimate for $E[g(X)]$, you then multiply it by $b-a$ according to equation (1) in order to obtain the Monte Carlo estimate of $\int_a^b g(x)dx$.

We illustrate the Monte Carlo technique in several examples.

Example 6: Evaluate the integral

$$\int_0^1 x^{1/3} dx$$

by hand. Then, execute the following Matlab script to get the Monte Carlo estimate:

```
a=0; b=1; %enter in the limits of integration
x=(b-a)*rand(1,50000)+a; %generate Uniform(a,b) random data points
y=x.^(1/3);
Monte_Carlo_estimate=(b-a)*mean(y)
```

Does your estimated value of the integral approximately agree with the actual value? Re-run the script several times to see whether the fluctuations about the actual value of the integral are small.

Example 7: Find the exact value of the integral

$$\int_{-1}^1 \sqrt{1-x^2} dx$$

using geometric reasoning (it's the area of a semicircular region). Then, execute the following Matlab script to get the Monte Carlo estimate:

```
a=-1; b=1;
x=(b-a)*rand(1,50000)+a; %uniform dist samples from -1 to 1
y=sqrt(1-x.^2);
Monte_Carlo_estimate=(b-a)*mean(y)
```

Does your estimated value of the integral approximately agree with the actual value? Re-run the script several times to see whether the fluctuations about the actual value of the integral are small.

Example 8: Using the table on page 123 of your textbook, show that

$$\int_0^1 \left(\frac{1}{\sqrt{2\pi}}\right) \exp(-x^2/2) dx = 0.3413.$$

Obtain a Monte Carlo estimate of this same integral by running the following Matlab script:

```
a=0; b=1;
x=(b-a)*rand(1,50000)+a; %uniform dist samples from 0 to 1
y=exp(-x.^2/2)/sqrt(2*pi);
Monte_Carlo_estimate=(b-a)*mean(y)
```

Example 9: Using the Matlab scripts from the preceding Examples as a guide, obtain a Monte Carlo estimate of the integral

$$\int_2^5 \sin(\log_e(4+9x^{3.5}))dx$$

using 50000 pseudorandomly generated data points.

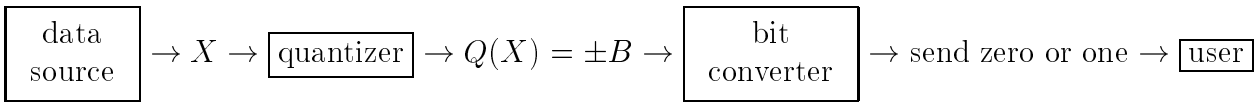
4.3 Exp 3: Application to Quantizer Design

Suppose we have a RV X which we assume for simplicity to have a PDF $f_X(x)$ which is symmetric about $x = 0$ (such as a mean 0 Gaussian random variable). A two-level quantizer for X would be a step function $Q(x)$ of the form

$$Q(x) = \begin{cases} -B, & -\infty < x \leq 0 \\ B, & 0 < x < \infty \end{cases}$$

where B is a positive constant.

The quantizer $Q(x)$ is sometimes called a *one bit quantizer* because depending upon whether $Q(x)$ is equal to B or $-B$, you can send a bit (a zero or a one) to the user to indicate which of these two cases occurs. In other words, we have a kind of primitive communication system:



Depending upon whether the user's received bit is 0 or 1, the user will estimate X as either B or $-B$. (If you are allowed to send more bits to the user, you can make a finer quantization for greater accuracy. Using just one bit, I am considering the simplest such communication system.)

In order to evaluate how good a particular quantizer is, you compute the so-called *signal-to-quantizing-noise ratio*, called SQNR for short and defined by

$$SQNR \triangleq 10 \log_{10} \left(\frac{\sigma_X^2}{E[(X - Q(X))^2]} \right).$$

The units of SQNR are called "decibels". (Typical ranges of SQNR for two-level quantization are in the 4 to 6 decibel range.) Our design goal is to select the value of the parameter B for which the resulting two-level quantizer $Q(x)$ will make the SQNR as big as possible. During one of our EE 3025 class lectures, I will derive what the best choice of B is. For the purposes of a Matlab demonstration, we can easily estimate the SQNR for different choices of B. For example, via Matlab, we can easily determine which of two possibilities for $Q(x)$ yields the bigger SQNR. Here is how you can use Matlab to obtain a SQNR estimate:

Step 1: Generate a vector `x` consisting of a large number of pseudorandom observations of the value of the random variable X.

Step 2: Then estimate the SQNR for a given positive B value via the following Matlab script:

```
%enter the value of B you are testing
y=B*(x>0)-B*(x<=0); %yields quantizer output sequence
error=mean((x-y).^2);
SQNR_estimate=10*log10(var(x)/error)
```

Now you are ready to design a quantizer by seeing which quantizer yields the biggest estimated SQNR performance.

Example 10: Let X be a standard Gaussian random variable. Let's quantize X with the two-level quantizer in which $B = 1$. Using 50000 simulated observations of X, estimate the resulting SQNR using the following Matlab code:

```
x=randn(1,50000);
B=1;
y=B*(x>0)-B*(x<=0);
error=mean((x-y).^2);
SQNR_estimate=10*log10(var(x)/error)
```

Run the script several times. Did you obtain an SQNR estimate in the 3.9 to 4.0 decibel range?

Example 11: Again, let X be the standard Gaussian RV. You are now going to use a different B than $B = 1$ to see if you can improve the SQNR performance of your two-level quantizer. To obtain the B value that we are going to use, run the following script:

```
x=randn(1,50000);
t=find(x>0);
B=mean(x(t))
```

Did you get a B value different from 1? Now estimate the SQNR performance of your new quantizer with the following Matlab script:

```
x=randn(1,10000);
%enter in the B value you just found
y=B*(x>0)-B*(x<=0);
error=mean((x-y).^2);
SQNR_estimate=10*log10(var(x)/error)
```

Run the script a few times to see if there is not much change in the estimated SQNR. Is your SQNR estimate around 4.4 decibels? Notice that this is an improvement over the SQNR performance you obtained in Example 10 using the two-level quantizer in which $B = 1$.

Example 12: Now let X be a uniformly distributed RV in the interval $[-3, 3]$. Choose B for your two-level quantizer design by running the following Matlab script:

```
a=-3;b=3;
x=(b-a)*rand(1,50000)+a; %generate Uniform samples in [-3,3]
t=find(x>0);
B=mean(x(t))
```

Now run the following script to see what SQNR performance you are getting for this choice of B:

```
a=-3;b=3;
x=(b-a)*rand(1,50000)+a; %generate Uniform samples in [-3,3]
%enter in the B value you just found
y=B*(x>0)-B*(x<=0);
error=mean((x-y).^2);
SQNR_estimate=10*log10(var(x)/error)
```

Now re-run this script with $B = 1$ in order to convince yourself that your first choice of B gives the bigger SQNR. What is the approximate difference in the SQNR performance yielded by these two quantizers? Is the difference more than one decibel? (In communication system design, a difference of one decibel or more is pretty significant. If you can achieve a one decibel or more improvement by changing your design, then you would typically change your design.)

In closing, think about the following: suppose you have a vector `x` consisting of a large number of observations of a RV X. The following Matlab script would generate a real number B:

```
t=find(x>0);
B=mean(x(t))
```

Can you see any possible relationship between the B that is generated by this script and the real number

$$\frac{\int_0^\infty x f_X(x)dx}{\int_0^\infty f_X(x)dx} \; ?$$

4.4 Exp 4: Application to Text Compression

A text file typically consists of a long sequence of characters from the ascii alphabet of size 256. For simplicity, we suppose that we have a text file formed as a sequence of $2^{16} = 65536$ characters from the following character alphabet of size 8:

$$\{0, 1, 2, 3, 4, 5, 6, 7\}.$$

For example, the text file might start off as:

$$1266310557240051\cdots \tag{2}$$

Suppose we randomly select a character from the sequence of characters forming our text file; let random variable X denote this randomly selected character. Suppose X has the following PMF:

x	0	1	2	3	4	5	6	7
$P^X(x)$	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/128

For example, the probability

$$P^X(3) = 1/16$$

means that 1/16 of the 65536 characters in the text file (a total of 4096 characters) are equal to 3.

Suppose you wish to store the given text file on your hard disk for use at some future time. You could just store it uncompressed as a 65536 character file. Better yet, you could store your file by compressing it so that fewer than 65536 characters are used. Here is one way that you might be able to do this:

Step 1: Use the following table to replace each character in the text file with a corresponding binary codeword, where the codeword assignment is as follows:

character	0	1	2	3	4	5	6	7
codeword	00	01	100	101	110	1110	11110	11111

We require that no codeword is a prefix of any other codeword.

Step 2: Partition the stream of bits resulting from Step 1 into three-bit blocks and then assign characters to each of these blocks as follows:

block	000	001	010	011	100	101	110	111
character	0	1	2	3	4	5	6	7

The result is a text file of possibly smaller size than 65536 characters.

To illustrate this two-step method, suppose our original text file starts as indicated in (2). Then the bit stream resulting from Step 1 starts with:

$$01100111101111010101 00\cdots$$

Performing Step 2, the compressed file then starts with:

$$317365\cdots$$

Example 13: Based upon the information given up to now, it is not hard to devise a simple Matlab script that will compute how many characters are in the compressed file. Here is such a script, which you should now run:

```
p=[1/2 1/4 1/8 1/16 1/32 1/64 1/128 1/128]; %character probs
L=[2 2 3 3 3 4 5 5]; %Step 1 codeword lengths
compressed_file_length=65536*sum(p.*L)/3
```

Does the compressed file consist of fewer than 65536 characters? Form the ratio

$$\frac{65536}{number\ of\ characters\ in\ compressed\ file}.$$

This number is called the *compression ratio*. The bigger your compression ratio is, the better the job you are doing in compressing your text file. The very best compression methods typically achieve a compression ratio of more than 2. (The simplified compression method we are using in this experiment is probability not going to do that well.)

In the following example, you are going to test another compression method in which we change the code used in Step 1.

Example 14: In Step 1, use instead the following codeword assignment:

character	0	1	2	3	4	5	6	7
codeword	0	10	110	1110	11110	111110	1111110	1111111

Make the appropriate changes in the Matlab script of Example 13 so that you can compute the length of the compressed file (the number of characters in the compressed file). (Hint: you only have to change one line of Matlab code.) Recompute the compression ratio. Is this a bigger compression ratio than obtained with the previous code?

4.5 Exp 5: Golomb Codes

For the vast majority of students, I believe Experiments 1-4 are enough work for Recitation 4. However, if you finished Experiments 1-4 during your recitation period, you can start on this Experiment.

Suppose you have a sequence of binary data (i.e., 0's and 1's) that you want to represent in more compressed form. Run-length coding is a way to do this. You partition the data into runs of zeroes and runs of ones. Each run of zeroes is replaced with a binary codeword from a so-called Golomb codeword set. Each run of ones is replaced with a binary codeword from another Golomb codeword set. Runs of zeroes and runs of ones appear randomly in the data. The probabilities with which runs of zeroes and runs of ones appear in the data are typically taken as follows:

$$P[\text{run of } n \text{ zeroes}] = p_0(n) = p^{n-1}(1-p), \quad n = 1, 2, 3, \cdots$$

$$P[\text{run of } n \text{ ones}] = p_1(n) = (1-p)^{n-1}p, \quad n = 1, 2, 3, \cdots$$

where p is the probability that a zero appears in the data (and therefore $1-p$ is the probability that a one appears in the data). The value of p determines the two codeword sets that are used. Notice that the two runlength probability distributions are geometric distributions. The Golomb codeword set that is chosen for coding the runs of zeroes minimizes expected codeword length as computed according to the geometric distribution $p_0(n)$; the Golomb codeword set that is chosen for coding the runs of ones minimizes expected codeword length as computed according to the geometric distribution $p_1(n)$.

For simplicity, let us just concentrate on the Golomb codeword set that is chosen for the runs of zeroes. Here are the first 10 codewords in the Golomb codeword set that should be used when the probability of a zero satisfies $p = 1/\sqrt{2} = 0.7071$:

Golomb codewords ($p = 1/\sqrt{2} = 0.7071$)			
runlength	codeword	runlength	codeword
1	00	6	1101
2	01	7	11100
3	100	8	11101
4	101	9	111100
5	1100	10	111101

Table 1

(The first two codewords are $0, 1$ preceded by 0, the next two codewords are $0, 1$ preceded by 10, the next two codewords after that are $0, 1$ preceded by 110, etc.)

Here are the first 10 codewords in the Golomb codeword set that should be used when the probability of a zero satisfies $p = 1/\sqrt[4]{2} = 0.8409$:

Golomb codewords ($p = 1/\sqrt[4]{2} = 0.8409$)			
runlength	codeword	runlength	codeword
1	000	6	1001
2	001	7	1010
3	010	8	1011
4	011	9	11000
5	1000	10	11001

Table 2

(The first four codewords are $00, 01, 10, 11$ preceded by 0, the next four codewords are $00, 01, 10, 11$ preceded by 10, the next four codewords after that are $00, 01, 10, 11$ preceded by 110, etc.)

Let $L(n)$ be the length of the n-th Golomb codeword. The ratio

$$\frac{E[\text{Golomb codeword length}]}{E[\text{runlength}]} = \frac{\sum_{i=1}^{\infty} L(n)p_0(n)}{\sum_{i=1}^{\infty} np_0(n)} \tag{3}$$

is the key figure of merit that tells us how good a particular Golomb codeword set is. This ratio is less than or equal to 1. The smaller it is, the better. For example, if the ratio is 2/3, this means that the codewords representing the runs of zeroes will occupy only two-thirds as much space as the space occupied by the runs of zeroes themselves.

Now you are ready to do some analysis of Golomb codeword set performance.

- *Example 15:* Use Matlab to compute the ratio (3) that arises when the probability of a zero in the data is $p = 1/\sqrt{2}$ and when the Golomb codeword set in Table 1 is used to encode the runs of zeroes in the data. In order to accomplish this, notice from Table 1 that the codeword lengths are $2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, \cdots$. The line of Matlab code "`1+ceil(n/2)`" will generate these codeword lengths. To verify this, run the Matlab script:

```
n=1:12;
1+ceil(n/2)
```

It is now clear that you can use the following Matlab code to compute the approximate value of the ratio (3):

```
n=1:10000;
p=1/sqrt(2);
L=1+ceil(n/2);
p0=p.^(n-1)*(1-p);
ratio=sum(L.*p0)/sum(n.*p0)
```

What percentage of space is saved by encoding the runs of zeroes into Golomb codewords?

- *Example 16:* Compute the ratio (3) that arises when the probability of a zero in the data is $p = \sqrt[4]{2}$ and when the Golomb codeword set in Table 2 is used to encode the runs of zeroes in the data. Notice that the codeword lengths are

$$3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, \cdots,$$

so your first task is a line of Matlab code that will generate these lengths. What percentage of space is saved by encoding the runs of zeroes into Golomb codewords?

- *Example 17:* Compute the ratio (3) that arises when the probability of a zero in the data is $p = 1/\sqrt{2}$ and when the Golomb codeword set in Table 2 is used to encode the runs of zeroes in the data. (The ratio should be bigger than you got in Example 15, because the Golomb codewords given by Table 1 give the smallest ratio when $p = 1/\sqrt{2}$.)

- *Example 18:* Compute the ratio (3) that arises when the probability of a zero in the data is $p = 1/\sqrt[4]{2}$ and when the Golomb codeword set in Table 1 is used to encode the runs of zeroes in the data. (The ratio should be bigger than you got in Example 16, because the Golomb codewords given by Table 2 give the smallest ratio when $p = 1/\sqrt[4]{2}$.)

- *Example 19:* If you had to extend Table 1, what would be the Golomb codewords for runs of zeroes of lengths $11, 12, 13, 14, 15, 16, 17, 18, 19, 20$? If you had to extend Table 2, what would be the Golomb codewords for runs of zeroes of these same lengths?

For any p, there is a Golomb codeword set appropriate to that choice of p. How one constructs the appropriate Golomb codeword set for a given p is covered in the course EE 5585, Data Compression.

EE 3025 S2007 Recitation 4 Lab Form

Name and Student Number of Team Member 1:

Name and Student Number of Team Member 2:

Name and Student Number of Team Member 3:

****

I will give you an integral to evaluate approximately using the Monte Carlo Method in Experiment 2.