**Lab 2: Random Data**

**Must create an R notebook in jupyter or RStudio notebook of the following material and code and to the exercises.**

**2  Random Data**

Although Einstein said that god does not play dice, R can. For example

> sample(1:6,10,replace=T)

[1] 6 4 4 3 5 2 3 3 5 4

or with a function

> RollDie = function(n) sample(1:6,n,replace=T)

> RollDie(5)

[1] 3 6 1 2 2

In fact, R can create lots of different types of random numbers ranging from familiar families of distributions to specialized ones.

**2.1  Random number generators in R - the “r” functions.**

As we know, random numbers are described by a distribution. That is, some function which specifies the probability that a random number is in some range. For example *P*(*a* < *X* *b*). Often this is given by a probability density (in the continuous case) or by a function *P*(*X*=*k*) = *f*(*k*) in the discrete case. R will give numbers drawn from lots of different distributions. In order to use them, you only need familiarize yourselves with the parameters that are given to the functions such as a mean, or a rate. Here are examples of the most common ones. For each, a histogram is given for a random sample of size 100, and density (using the d'' functions) is superimposed as appropriate.

**Uniform.**

Uniform numbers are ones that are "equally likely" to be in the specified range. Often these numbers are in [0,1] for computers, but in practice can be between [a,b] where a,b depend upon the problem. An example might be the time you wait at a traffic light. This might be uniform on [0,2].

> runif(1,0,2) # time at light

[1] 1.490857 # also runif(1,min=0,max=2)

> runif(5,0,2) # time at 5 lights

[1] 0.07076444 0.01870595 0.50100158 0.61309213 0.77972391

> runif(5) # 5 random numbers in [0,1]

[1] 0.1705696 0.8001335 0.9218580 0.1200221 0.1836119

The general form is runif(n,min=0,max=1) which allows you to decide how many uniform random numbers you want (n), and the range they are chosen from ([min,max])  
  
To see the distribution with min=0 and max=1 (the default) we have

> x=runif(100) # get the random numbers

> hist(x,probability=TRUE,col=gray(.9),main="uniform on [0,1]")

> curve(dunif(x,0,1),add=T)

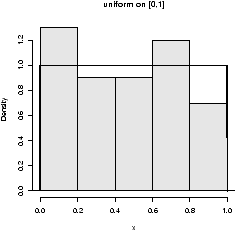


Figure 25: 100 uniformly random numbers on [0,1]

The only tricky thing was plotting the histogram with a background color''. Notice how the dunif function was used with the curvefunction.

**Normal.**

Normal numbers are the backbone of classical statistical theory due to the central limit theorem The normal distribution has two parameters a mean μ and a standard deviation . These are the location and spread parameters. For example, IQs may be normally distributed with mean 100 and standard deviation 16, Human gestation may be normal with mean 280 and standard deviation about 10 (approximately). The family of normals can be standardized to normal with mean 0 (centered) and variance 1. This is achieved by "standardizing" the numbers, i.e.

*Z*=(*X*- μ)/.  
  
Here are some examples

> rnorm(1,100,16) # an IQ score

[1] 94.1719

> rnorm(1,mean=280,sd=10)

[1] 270.4325 # how long for a baby (10 days early)

Here the function is called as rnorm(n,mean=0,sd=1) where one specifies the mean and the standard deviation.  
  
To see the shape for the defaults (mean 0, standard deviation 1) we have (figure [26](http://www.math.csi.cuny.edu/Statistics/R/simpleR/stat007.html#fig:random-data-normal-100-16))

> x=rnorm(100)

> hist(x,probability=TRUE,col=gray(.9),main="normal mu=0,sigma=1")

> curve(dnorm(x),add=T)

## also for IQs using rnorm(100,mean=100,sd=16)

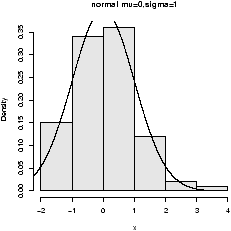
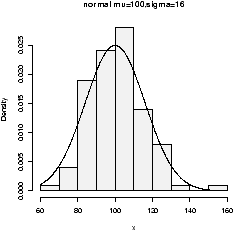
  

Figure 26: Normal(0,1) and normal(100,16)

**Binomial.**

1. **Compute probability according to binomial distribution**

The **binomial distribution**is a discrete probability distribution. It describes the outcome of *n*independent trials in an experiment. Each trial is assumed to have only two outcomes, either success or failure. If the probability of a successful trial is *p*, then the probability of having *x*successful outcomes in an experiment of *n* independent trials is as follows.

      (    )
        n    x      (n− x)
f(x) =  x   p (1− p)      where x = 0,1,2,...,n


#### Problem

Suppose there are twelve multiple choice questions in an English class quiz. Each question has five possible answers, and only one of them is correct. Find the probability of having four or less correct answers if a student attempts to answer every question at random.

#### Solution

Since only one out of five possible answers is correct, the probability of answering a question correctly by random is 1/5=0.2. We can find the probability of having exactly 4 correct answers by random attempts as follows.

> dbinom(4, size=12, prob=0.2)   
[1] 0.1329

To find the probability of having four or less correct answers by random attempts, we apply the function dbinom with *x*= 0*,…,*4.

> dbinom(0, size=12, prob=0.2) +   
+ dbinom(1, size=12, prob=0.2) +   
+ dbinom(2, size=12, prob=0.2) +   
+ dbinom(3, size=12, prob=0.2) +   
+ dbinom(4, size=12, prob=0.2)   
[1] 0.9274

Alternatively, we can use the cumulative probability function for binomial distribution pbinom.

> pbinom(4, size=12, prob=0.2)   
[1] 0.92744

2) **Generate binomial random numbers**

The binomial random numbers are discrete random numbers. They have the distribution of the number of successes in *n* independent Bernoulli trials where a Bernoulli trial results in success or failure, success with probability *p*.  
  
A single Bernoulli trial is given with n=1 in the binomial

> n=1, p=.5 # set the probability

> rbinom(1,n,p) # different each time

[1] 1

> rbinom(10,n,p) # 10 different such numbers

[1] 0 1 1 0 1 0 1 0 1 0

A binomially distributed number is the same as the number of 1's in *n* such Bernoulli numbers. For the last example, this would be 5. There are then two parameters *n* (the number of Bernoulli trials) and *p* (the success probability).  
  
To generate binomial numbers, we simply change the value of n from 1 to the desired number of trials. For example, with 10 trials:

> n = 10; p=.5

> rbinom(1,n,p) # 6 successes in 10 trials

[1] 6

> rbinom(5,n,p) # 5 binomial number

[1] 6 6 4 5 4

The graphs (figure [27](http://www.math.csi.cuny.edu/Statistics/R/simpleR/stat007.html#fig:random-data-binomials)) show 100 binomially distributed random numbers for 3 values of *n* and for *p*=.25. Notice in the graph, as *n* increases the shape becomes more and more bell-shaped. These graphs were made with the commands

> n=5;p=.25 # change as appropriate

> x=rbinom(100,n,p) # 100 random numbers

> hist(x,probability=TRUE,)

## use points, not curve as dbinom wants integers only for x

> xvals=0:n;points(xvals,dbinom(xvals,n,p),type="h",lwd=3)

> points(xvals,dbinom(xvals,n,p),type="p",lwd=3)

... repeat with n=15, n=50

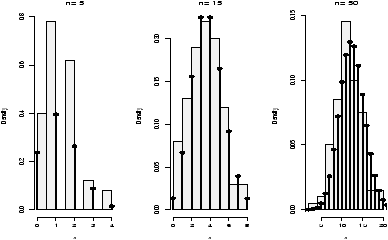


Figure 27: Random binomial data with the theoretical distribution

**Exponential**

The exponential distribution is important for theoretical work. It is used to describe lifetimes of electrical components (to first order). For example, if the mean life of a light bulb is 2500 hours one may think its lifetime is random with exponential distribution having mean 2500. The one parameter is the rate = 1/mean. We specify it as follows rexp(n,rate=1/2500). Here is an example with the rate being 1/2500 (figure [28](http://www.math.csi.cuny.edu/Statistics/R/simpleR/stat007.html#fig:random-data-exponetial)).

> x=rexp(100,1/2500)

> hist(x,probability=TRUE,col=gray(.9),main="exponential mean=2500")

> curve(dexp(x,1/2500),add=T)

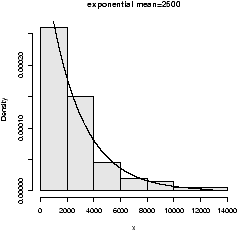


Figure 28: Random exponential data with theoretical density

There are others of interest in statistics. Common ones are the Poisson, the Student *t*-distribution, the *F* distribution, the beta distribution and the 2 (chi squared) distribution.

**2.2  Sampling with and without replacement using sample**

R has the ability to sample with and without replacement. That is, choose at random from a collection of things such as the numbers 1 through 6 in the dice rolling example. The sampling can be done with replacement (like dice rolling) or without replacement (like a lottery). By default sample samples without replacement each object having equal chance of being picked. You need to specify replace=TRUE if you want to sample with replacement. Furthermore, you can specify separate probabilities for each if desired.  
  
Here are some examples 

## Roll a die

> sample(1:6,10,replace=TRUE)

[1] 5 1 5 3 3 4 5 4 2 1 # no sixes!

## toss a coin

> sample(c("H","T"),10,replace=TRUE)

[1] "H" "H" "T" "T" "T" "T" "H" "H" "T" "T"

## pick 6 of 54 (a lottery)

> sample(1:54,6) # no replacement

[1] 6 39 23 35 25 26

## pick a card. (Fancy! Uses paste, rep)

> cards = paste(rep(c("A",2:10,"J","Q","K"),4),c("H","D","S","C"))

> sample(cards,5) # a pair of jacks, no replacement

[1] "J D" "5 C" "A S" "2 D" "J H"

## roll 2 die. Even fancier

> dice = as.vector(outer(1:6,1:6,paste))

> sample(dice,5,replace=TRUE) # replace when rolling dice

[1] "1 1" "4 1" "6 3" "4 4" "2 6"

The last two illustrate things that can be done with a little typing and a lot of thinking using the fun commands paste for pasting together strings, rep for repeating things and outer for generating all possible products. 

**2.3  A bootstrap sample**

Bootstrapping is a method of sampling from a data set to make statistical inference. The intuitive idea is that by sampling, one can get an idea of the variability in the data. The process involves repeatedly selecting samples and then forming a statistic. Here is a simple illustration on obtaining a sample. 

The examples in this section are based on a built-in [data frame](http://www.r-tutor.com/node/10) named **faithful**. It consists of a collection of observations of the Old Faithful geyser in the USA Yellowstone National Park. The following is a preview via the head function.

> head(faithful)   
  eruptions waiting   
1     3.600      79   
2     1.800      54   
3     3.333      74   
4     2.283      62   
5     4.533      85   
6     2.883      55

There are two observation variables in the data set. The first one, called eruptions, is the duration of the geyser eruptions. The second one, called waiting, is the length of waiting period until the next eruption.

The built in data set faithful has a variable “eruptions'' that measures the time between eruptions at Old Faithful. It has an unusual distribution. A bootstrap sample is just a sample with replacement from the given values. It can be found as follows

> data(faithful) # part of R's base

> names(faithful) # find the names for faithful

[1] "eruptions" "waiting"

> eruptions = faithful[['eruptions']] # or attach and detach faithful

> sample(eruptions,10,replace=TRUE)

[1] 2.03 4.37 4.80 1.98 4.32 2.18 4.80 4.90 4.03 4.70

> hist(eruptions,breaks=25) # the dataset

## the bootstrap sample

> hist(sample(eruptions,100,replace=TRUE),breaks=25)

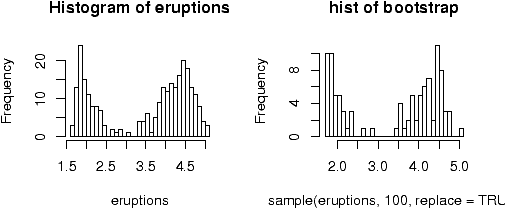


Figure 29: Bootstrap sample

Notice that the bootstrap sample has a similar histogram, but it is different (figure [29](http://www.math.csi.cuny.edu/Statistics/R/simpleR/stat007.html#fig:random-data-bootstrap-eg)). 

**2.4  d, p and q functions**

The d functions were used to plot the theoretical densities above. As with the “r” functions, you need to specify the parameters, but differently, you need to specify the *x* values (not the number of random numbers *n*).

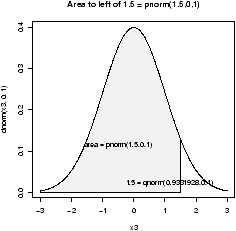
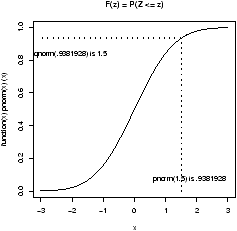
  

Figure 30: Illustration of 'p' and 'q' functions

The p and q functions are for the cumulative distribution functions and the quantiles. As mentioned, the distribution of a random number is specified by the probability that the number is between *a* and *b* for arbitrary *a* and *b*, *P*(*a* < *X* *b*). In fact, the value *F*(*x*) = *P*(*X* =*b*) is enough.  
  
The p functions answer what is the probability that a random variable is less than *x*. Such as for a standard normal, what is the probability it is less than .7?

> pnorm(.7) # standard normal

[1] 0.7580363

> pnorm(.7,1,1) # normal mean 1, std 1

[1] 0.3820886

Notationally, these answer *P*(*Z* .7) where *Z* is a standard normal variable or normal(1,1). To answer *P*(*Z* > .7) is also easy. You can do the work by noting this is 1 - *P*(*Z*  .7) or let R do the work, by specifying lower.tail=F as in:

> pnorm(.7,lower.tail=F)

[1] 0.2419637

The q function are inverse to this. They ask, what value corresponds to a given probability. This the quantile or point in the data that splits it accordingly. For example, what value of *z* has .75 of the area to the right for a standard normal? (This is *Q*3)

> qnorm(.75)

[1] 0.6744898

Notationally, this is finding *z* which solves 0.75 = *P*(*Z*  *z*). 

**2.5  Standardizing, scale****and *z* scores**

To standardize a random variable you subtract the mean and then divide by the standard deviation. That is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Z* = | |  | | --- | | *X* - μ | | |  | | --- | |  | | |  | | . |

To do so requires knowledge of the mean and standard deviation.   
  
You can also standardize a sample. There is a convenient function scale that will do this for you. This will make your sample have mean 0 and standard deviation 1. This is useful for comparing random variables which live on different scales.  
  
Normal random variables are often standardized as the distribution of the standardized normal variable is again normal with mean 0 and variance 1. (The standard'' normal.) The *z*-score of a normal number is the value of it after standardizing.   
  
If we have normal data with mean 100 and standard deviation 16 then the following will find the *z*-scores

> x = rnorm(5,100,16)

>

> x

[1] 93.45616 83.20455 64.07261 90.85523 63.55869

> z = (x-100)/16

> z

[1] -0.4089897 -1.0497155 -2.2454620 -0.5715479 -2.2775819

The *z*-score is used to look up the probability of being to the right of the value of *x* for the given random variable. This way only one table of normal numbers is needed. With R, this is not necessary. We can use the pnorm function directly

> pnorm(z)

[1] 0.34127360 0.14692447 0.01236925 0.28381416 0.01137575

> pnorm(x,100,16) # enter in parameters

[1] 0.34127360 0.14692447 0.01236925 0.28381416 0.01137575

**2.6  Problems**

**2.1**

Generate 10 random numbers from a uniform distribution on [0,10]. Use R to find the maximum and minimum values.x

**2.2**

Generate 10 random normal numbers with mean 5 and standard deviation 5 (normal(5,5)). How many are less than 0? (Use R)

**Counting**

Frequently it is useful to count how many elements within a vector satisfy

some condition. For example, if we wanted to know how many of the integers

between 1 and 100 are divisible by 7, we could use

A = seq(100)

B = (A %% 7 == 0)

x = sum(B)

Note that when summing the Boolean vector B, true values are counted as 1

and false values are counted as 0.

**2.3**

Generate 100 random normal numbers with mean 100 and standard deviation 10. How many are 2 standard deviations from the mean (smaller than 80 or bigger than 120)?

**2.4**

Toss a fair coin 50 times (using R). How many heads do you have?

**2.5**

Roll a die'' 100 times. How many 6's did you see?

**2.6**

Select 6 numbers from a lottery containing 49 balls. What is the largest number? What is the smallest? Answer these using R.

**2.7**

For normal(0,1), find a number *z*\* solving *P*(*Z*  *z\**) = .05 (use qnorm).

**2.8**

For normal(0,1), find a number *z*\* solving *P*(-*z*\* *Z*  *z*\*) = .05 (use qnorm and symmetry).

**2.9**

How much area (probability) is to the right of 1.5 for a normal(0,2)?

**2.10**

Make a histogram of 100 exponential numbers with mean 10. Estimate the median. Is it more or less than the mean?

**2.11**

Can you figure out what this R command does?

> rnorm(5,mean=0,sd=1:5)

**2.12**

Use R to pick 5 cards from a deck of 52. Did you get a pair or better? Repeat until you do. How long did it take?