Experiments with Random Numbers

The following experiment will play a prominent role in this course:

Pick a (real) number between zero and one

Experiments with Random Numbers

The following experiment will play a prominent role in this course:

Pick a (real) number between zero and one

Another way of saying this is: pick a number in the interval $(0, 1)$

Experiments with Random Numbers

The following experiment will play a prominent role in this course:

Pick a (real) number between zero and one

Another way of saying this is: pick a number in the interval $(0, 1)$

The sample space $\mathcal{S}$ for this experiment is:

$$\mathcal{S} = \{x : 0 < x < 1\}$$

Experiments with Random Numbers

The following experiment will play a prominent role in this course:

Pick a (real) number between zero and one

Another way of saying this is: pick a number in the interval $(0, 1)$

The sample space $\mathcal{S}$ for this experiment is:

$$\mathcal{S} = \{x : 0 < x < 1\}$$

That is, (S) is the set of all real numbers that are greater than zero and less than one.

Experiments with Random Numbers

Of course there are many ways this could be done, but we will assume that it is done in a manner that gives each real number between zero and one the same chance of being selected.

Experiments with Random Numbers

Of course there are many ways this could be done, but we will assume that it is done in a manner that gives each real number between zero and one the same chance of being selected.

With this caveat, our experiment becomes one of a large class of experiments known as *experiments with equally likely outcomes*

Experiments with Random Numbers

The RAND function available on most spreadsheets can be thought of as performing this experiment. If we type the formula

=RAND()

in a cell and hit enter, a number between zero and one will appear.

Experiments with Random Numbers

The RAND function available on most spreadsheets can be thought of as performing this experiment. If we type the formula

=RAND()

in a cell and hit enter, a number between zero and one will appear.

Each time we refresh the spreadsheet (usually via **F9**), we get a different number.

Experiments with Random Numbers

The RAND function available on most spreadsheets can be thought of as performing this experiment. If we type the formula

=RAND()

in a cell and hit enter, a number between zero and one will appear.

Each time we refresh the spreadsheet (usually via **F9**), we get a different number.

The resulting numbers are called *psuedorandom* numbers because, while they are not truly random, they behave much like a set of truly random numbers would.

Experiments with Random Numbers

The RAND function available on most spreadsheets can be thought of as performing this experiment. If we type the formula

=RAND()

in a cell and hit enter, a number between zero and one will appear.

Each time we refresh the spreadsheet (usually via **F9**), we get a different number.

The resulting numbers are called *psuedorandom* numbers because, while they are not truly random, they behave much like a set of truly random numbers would.

The advantage of psuedorandom numbers is that they are much easier to produce than truly random numbers.

Experiments with Random Numbers

The RAND spreadsheet function is useful for generating the results of repeatedly conducting the "pick a number between zero and one" experiment.

Experiments with Random Numbers

The RAND spreadsheet function is useful for generating the results of repeatedly conducting the "pick a number between zero and one" experiment.

However, if we want to produce a larger number of replications, say 1,000,000, this method becomes awkward.

Experiments with Random Numbers

The RAND spreadsheet function is useful for generating the results of repeatedly conducting the "pick a number between zero and one" experiment.

However, if we want to produce a larger number of replications, say 1,000,000, this method becomes awkward.

Alternatively, we can use a computer program like R, which can easily generate and store millions of such numbers.

Why R?

Admitedly R is more difficult to use than most statistical packages. So, why use R?

Why R?

Admitedly R is more difficult to use than most statistical packages. So, why use R?

First, R is readily available as a free download. You can install it on your own computer, and you can take it with you when you leave Stonehill.

Why R?

Admitedly R is more difficult to use than most statistical packages. So, why use R?

First, R is readily available as a free download. You can install it on your own computer, and you can take it with you when you leave Stonehill.

Second, unlike most statistical packages that provide a fixed set of routines with an easy, graphical interface, R is a *programming language*.

Why R?

Admitedly R is more difficult to use than most statistical packages. So, why use R?

First, R is readily available as a free download. You can install it on your own computer, and you can take it with you when you leave Stonehill.

Second, unlike most statistical packages that provide a fixed set of routines with an easy, graphical interface, R is a *programming language*.

This means there is more of a learning curve with R, but it makes R very powerful and flexible.

Why R?

Admitedly R is more difficult to use than most statistical packages. So, why use R?

First, R is readily available as a free download. You can install it on your own computer, and you can take it with you when you leave Stonehill.

Second, unlike most statistical packages that provide a fixed set of routines with an easy, graphical interface, R is a *programming language*.

This means there is more of a learning curve with R, but it makes R very powerful and flexible.

R is modeled after the S language which was developed at Bell Labs.

Why R?

Third, because of its modular structure, it is easy to contribute new routines to R. Cutting edge statistical techniques often appear in R before they are available in commercial packages.

Why R?

Third, because of its modular structure, it is easy to contribute new routines to R. Cutting edge statistical techniques often appear in R before they are available in commercial packages.

For a list of the major contributors, type *contributors()* and hit enter.

Why R?

Third, because of its modular structure, it is easy to contribute new routines to R. Cutting edge statistical techniques often appear in R before they are available in commercial packages.

For a list of the major contributors, type *contributors()* and hit enter.

Finally, and perhaps most important, R is catching on.

Why R?

Third, because of its modular structure, it is easy to contribute new routines to R. Cutting edge statistical techniques often appear in R before they are available in commercial packages.

For a list of the major contributors, type *contributors()* and hit enter.

Finally, and perhaps most important, R is catching on.

In a number of specialized areas such as bioinformatics, R has become the preferred tool for many researchers.

Why R?

Third, because of its modular structure, it is easy to contribute new routines to R. Cutting edge statistical techniques often appear in R before they are available in commercial packages.

For a list of the major contributors, type *contributors()* and hit enter.

Finally, and perhaps most important, R is catching on.

In a number of specialized areas such as bioinformatics, R has become the preferred tool for many researchers.

This is not suprising because bioinformatics combines biology, computer science, and statistics.

For an example, see: bioinformatics.oxfordjournals.org/cgi/content/full/26/1/139

Getting Started with R

R is essentially a command line program, although there are a number of graphical front-end programs available.

Getting Started with R

R is essentially a command line program, although there are a number of graphical front-end programs available.

For starters we will be using R in command line mode.

Getting Started with R

R is essentially a command line program, although there are a number of graphical front-end programs available.

For starters we will be using R in command line mode.

On a UNIX platform, start R by opening a terminal window (command prompt) and typing: R

Getting Started with R

R is essentially a command line program, although there are a number of graphical front-end programs available.

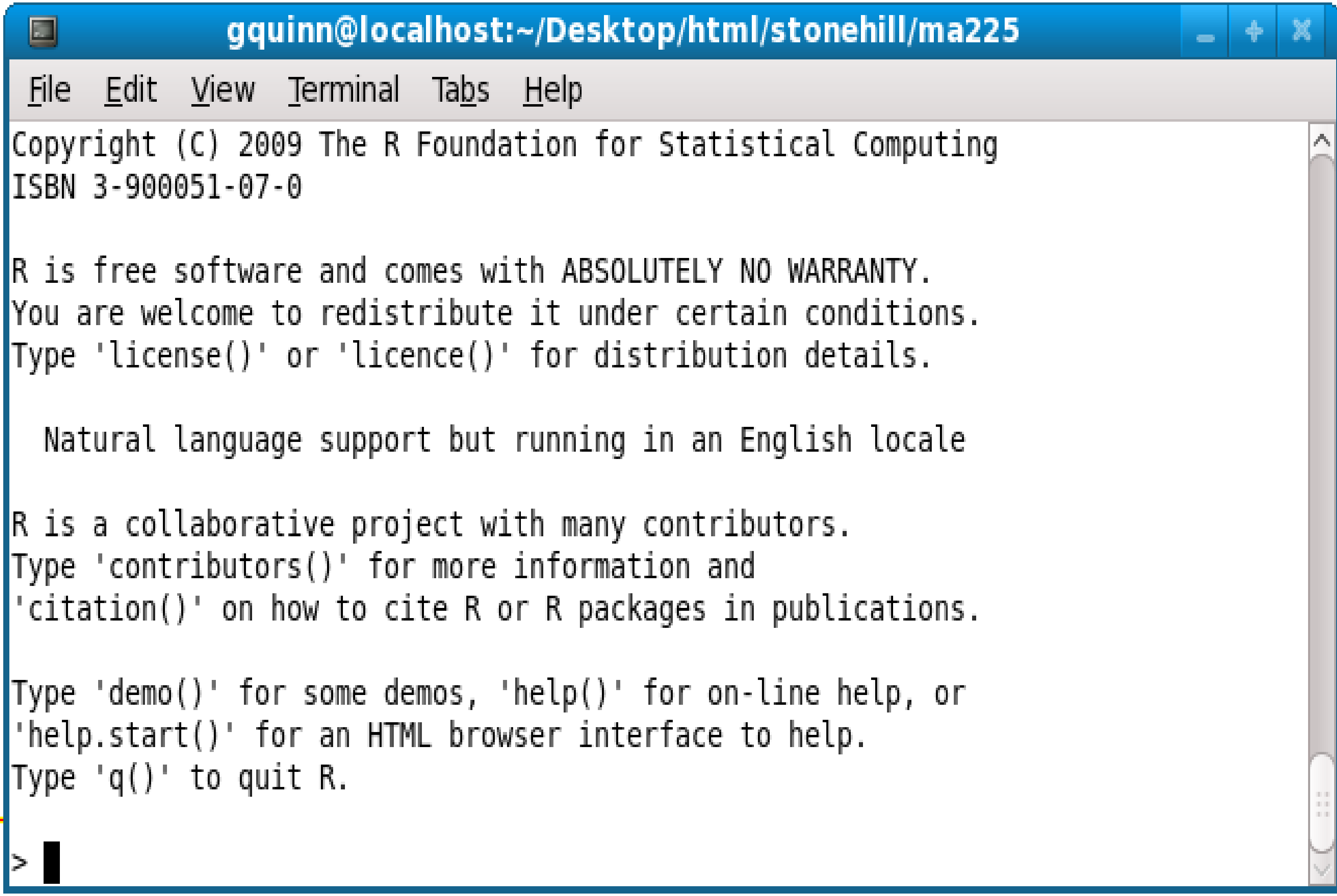
For starters we will be using R in command line mode.

On a UNIX platform, start R by opening a terminal window (command prompt) and typing: R

On a windows system set up with a shorcut, start R by double-clicking on the R icon

Getting Started with R

Usually something like the following set of messages will appear:



Now What?

Once we have the command prompt, R will sit there waiting for us to tell it to do something.

This can be rather intimidating if you are new to R

Now What?

Once we have the command prompt, R will sit there waiting for us to tell it to do something.

This can be rather intimidating if you are new to R

We'll start with something easy. Type:

```
contributors()
```

Now What?

Once we have the command prompt, R will sit there waiting for us to tell it to do something.

This can be rather intimidating if you are new to R

We'll start with something easy. Type:

contributors()

A list of contributors should appear, maybe in a separate window or maybe not.

Now What?

Once we have the command prompt, R will sit there waiting for us to tell it to do something.

This can be rather intimidating if you are new to R

We'll start with something easy. Type:

```
contributors()
```

A list of contributors should appear, maybe in a separate window or maybe not.

The local HELP function in R is built into the command:

```
help.start()
```

Now What?

Once we have the command prompt, R will sit there waiting for us to tell it to do something.

This can be rather intimidating if you are new to R

We'll start with something easy. Type:

```
contributors()
```

A list of contributors should appear, maybe in a separate window or maybe not.

The local HELP function in R is built into the command:

```
help.start()
```

This should start a browser window and open the local HELP webpage. Depending on local restrictions you may have to paste the filename into the browser window.

Pick a Number

Now we are ready to use R to perform the "pick a number between zero and one" experiment

As we will see, the probability model associated with this experiment is called the *uniform distribution*

Pick a Number

Now we are ready to use R to perform the "pick a number between zero and one" experiment

As we will see, the probability model associated with this experiment is called the *uniform distribution*

At the R command prompt, type:

```
runif(1)
```

Pick a Number

Now we are ready to use R to perform the "pick a number between zero and one" experiment

As we will see, the probability model associated with this experiment is called the *uniform distribution*

At the R command prompt, type:

```
runif(1)
```

This should produce a "random" number between zero and one.

Pick a Number

Now we are ready to use R to perform the "pick a number between zero and one" experiment

As we will see, the probability model associated with this experiment is called the *uniform distribution*

At the R command prompt, type:

```
runif(1)
```

This should produce a "random" number between zero and one.

You can recall the previous command(s) by pressing the up arrow key. Recall the runif(1) command and run it again.

Pick a Number

Now we are ready to use R to perform the "pick a number between zero and one" experiment

As we will see, the probability model associated with this experiment is called the *uniform distribution*

At the R command prompt, type:

runif(1)

This should produce a "random" number between zero and one.

You can recall the previous command(s) by pressing the up arrow key. Recall the runif(1) command and run it again.

You should get a different result, but still a number between zero and one.

Pick a Number

We are interested in what happens when we repeat this experiment over and over. The runif() function allows us to repeat the experiment multiple times with a single call. This time type:

runif(10)

Pick a Number

We are interested in what happens when we repeat this experiment over and over. The runif() function allows us to repeat the experiment multiple times with a single call. This time type:

runif(10)

You can make the number of repetitions as large as you like (subject to available memory).

Pick a Number

We would like to use R to examine the results of the runif() command, so we need to store them in a variable. Type:

```
x<-runif(50)
```

Pick a Number

We would like to use R to examine the results of the runif() command, so we need to store them in a variable. Type:

```
x<-runif(50)
```

We should be back to an empty command prompt.

What happened was that R ran the runif() function 50 times and stored the results in a variable called x.

Pick a Number

We would like to use R to examine the results of the runif() command, so we need to store them in a variable. Type:

x<-runif(50)

We should be back to an empty command prompt.

What happened was that R ran the runif() function 50 times and stored the results in a variable called x.

This is another feature of R that new users find frustrating: R often only shows results if you ask for them. Type:

x

Pick a Number

We would like to use R to examine the results of the runif() command, so we need to store them in a variable. Type:

x<-runif(50)

We should be back to an empty command prompt.

What happened was that R ran the runif() function 50 times and stored the results in a variable called x.

This is another feature of R that new users find frustrating: R often only shows results if you ask for them. Type:

x

This will display the contents of the x variable, allowing us to see the results of the 50 "pick a number" experiments.

Pick a Number

Now type:

hist(x)

Pick a Number

Now type:

hist(x)

This should produce a graphical display of the results called a **histogram**

Each bar represents a range of values, and the area of the bar is proportional to the number of values in that range.

Pick a Number

Now type:

hist(x)

This should produce a graphical display of the results called a **histogram**

Each bar represents a range of values, and the area of the bar is proportional to the number of values in that range.

If you are willling to live with the defaults R chooses with regard to the number of bars and ranges of values, it is very easy to get a histogram.

Pick a Number

Now type:

hist(x)

This should produce a graphical display of the results called a **histogram**

Each bar represents a range of values, and the area of the bar is proportional to the number of values in that range.

If you are willling to live with the defaults R chooses with regard to the number of bars and ranges of values, it is very easy to get a histogram.

If not, you can customize the histogram. Be forewarned, hist() has a *lot* of parameters - type

help(hist)

Pick a Number

The histogram for the results of 50 "pick a number" experiments is usually rather "bumpy"

If increase the number of repetitions, it should "smooth out". Enter:

```
x<-runif(10000)
hist(x)
```

Pick a Number

The histogram for the results of 50 "pick a number" experiments is usually rather "bumpy"

If increase the number of repetitions, it should "smooth out". Enter:

```
x<-runif(10000)
hist(x)
```

This time the bars should be more nearly equal.

This illustrates one of the central ideas in probability theory, known as the **law of large numbers**

The Law of Large Numbers

Notice that to draw the histogram, R chose to use 20 bars. Consequently, it divided the outcomes into 20 groups, with boundaries at multiples of 0.05.

The Law of Large Numbers

Notice that to draw the histogram, R chose to use 20 bars. Consequently, it divided the outcomes into 20 groups, with boundaries at multiples of 0.05.

If each number between zero and one has an equal chance of being chosen each time we perform the experiment, intuition tells us that we expect about $1/20^{th}$ of the results in each group.

The Law of Large Numbers

Notice that to draw the histogram, R chose to use 20 bars. Consequently, it divided the outcomes into 20 groups, with boundaries at multiples of 0.05.

If each number between zero and one has an equal chance of being chosen each time we perform the experiment, intuition tells us that we expect about $1/20^{th}$ of the results in each group.

Of course the actual counts in each group are random and will vary.

The Law of Large Numbers says that if we perform the experiment many times, the *proportion* of outcomes represented by a given bar should get closer and closer to $1/20$, the probability that x falls in the range represented by the bar in a *single trial*.

The Law of Large Numbers

If we have 20 groups, each representing an interval of length 0.05, and every number between zero and one has an equal chance of being chosen, on a single trial we expect the probability associated with each of the 20 events

x falls in the i^{th} interval, $i = 1, 2, \ldots, 20$

to be $1/20$.

The Law of Large Numbers

If we have 20 groups, each representing an interval of length 0.05, and every number between zero and one has an equal chance of being chosen, on a single trial we expect the probability associated with each of the 20 events

x falls in the i^{th} interval, $i = 1, 2, \ldots, 20$

to be $1/20$.

The Law of Large Numbers says that if we repeat the experiment many times, the *proportion* of outcomes corresponding to each of these events should be close to the probability of the event.

The Law of Large Numbers

If we have 20 groups, each representing an interval of length 0.05, and every number between zero and one has an equal chance of being chosen, on a single trial we expect the probability associated with each of the 20 events

x falls in the i^{th} interval, $i = 1, 2, \ldots, 20$

to be $1/20$.

The Law of Large Numbers says that if we repeat the experiment many times, the *proportion* of outcomes corresponding to each of these events should be close to the probability of the event.

In this case, it says the number of times we expect each event to occur is $1/20^{th}$ of the total number of repetitions.

The Law of Large Numbers

If we have 20 groups, each representing an interval of length 0.05, and every number between zero and one has an equal chance of being chosen, on a single trial we expect the probability associated with each of the 20 events

x falls in the i^{th} interval, $i = 1, 2, \ldots, 20$

to be $1/20$.

The Law of Large Numbers says that if we repeat the experiment many times, the *proportion* of outcomes corresponding to each of these events should be close to the probability of the event.

In this case, it says the number of times we expect each event to occur is $1/20^{th}$ of the total number of repetitions.

(500 in the case of 10,000 trials)

The Law of Large Numbers

Now we'll repeat the exercise with even more trials - 1,000,000. Enter:

```
x<-rep(0,1000000)
x<-runif(1000000)
hist(x)
```

The Law of Large Numbers

Now we'll repeat the exercise with even more trials - 1,000,000. Enter:

```
x<-rep(0,1000000)
x<-runif(1000000)
hist(x)
```

This time, if we classify the outcomes into 20 equally likely events, we expect about 50,000 occurrences for each.

The Law of Large Numbers

Now we'll repeat the exercise with even more trials - 1,000,000. Enter:

```
x<-rep(0,1000000)
x<-runif(1000000)
hist(x)
```

This time, if we classify the outcomes into 20 equally likely events, we expect about 50,000 occurrences for each.

The bars in the histogram of 1,000,000 trials should appear quite uniform, in accordance with the Law of Large Numbers.

The Coin Toss Experiment

The sample space for the "pick a number between zero and one" experiment is an **interval**

Now consider the "coin toss" experiment: We toss a fair coin, and observe one of two possible outcomes: heads or tails

For simplicity, we will label the outcomes 0 and 1.

The Coin Toss Experiment

The sample space for the "pick a number between zero and one" experiment is an **interval**

Now consider the "coin toss" experiment: We toss a fair coin, and observe one of two possible outcomes: heads or tails

For simplicity, we will label the outcomes 0 and 1.

We will adapt the runif() function to simulate this experiment as follows:

- Pick a number between zero and one, as before
- Multiply the number by 2
- Truncate the result to an integer with the FLOOR function

The Coin Toss Experiment

The R syntax for this is (for 100 repititions):

```
floor(2*runif(100))
```

The Coin Toss Experiment

The R syntax for this is (for 100 repititions):

```
floor(2*runif(100))
```

Now let's simulate the experiment of tossing a fair coin 1,000,000 times. Enter:

```
x<-rep(0,1000000)
x<-floor(2*runif(1000000))
hist(x)
```

The Coin Toss Experiment

The R syntax for this is (for 100 repititions):

```
floor(2*runif(100))
```

Now let's simulate the experiment of tossing a fair coin 1,000,000 times. Enter:

```
x<-rep(0,1000000)
x<-floor(2*runif(1000000))
hist(x)
```

This time, our histogram has only two bars, representing zero and one.

R still chose to use an interval from zero to one, but we can live with this.

The Coin Toss Experiment

Consider the results of this experiment in light of the Law of Large Numbers.

We assumed a fair coin, so each of the two outcomes should have probability $1/2$

The Coin Toss Experiment

Consider the results of this experiment in light of the Law of Large Numbers.

We assumed a fair coin, so each of the two outcomes should have probability $1/2$

That is, the events "0" and "1" each have probability of $1/2$ of occurring on a *single trial*

The Coin Toss Experiment

Consider the results of this experiment in light of the Law of Large Numbers.

We assumed a fair coin, so each of the two outcomes should have probability $1/2$

That is, the events "0" and "1" each have probability of $1/2$ of occurring on a *single trial*

The Law of Large Numbers says that if we perform 1,000,000 trials, we expect the *proportion* of trials corresponding to each of these events to be close to their probability on a *single trial*: $1/2$

The Coin Toss Experiment

With 1,000,000 trials, we expect about 500,000 each of "0" and "1" to occur.

The Coin Toss Experiment

With 1,000,000 trials, we expect about 500,000 each of "0" and "1" to occur.

To get a count for the number of zeros and ones, enter:

table(x)

The Coin Toss Experiment

With 1,000,000 trials, we expect about 500,000 each of "0" and "1" to occur.

To get a count for the number of zeros and ones, enter:

table(x)

Note that the counts are quite close to 500,000, even though there is nothing to prevent "1" coming up, say, 800,000 times in a million tosses of a fair coin. It's just *highly* unlikely.