Random Experiments with R

Corrections: (check the class message board)

This exercise uses `R` to simulate a random experiment and make a histogram of the results. The sample code `ListAndHistogram.R` illustrates the features of `R` you need for this.

Example 1: The `R` function `rnorm(n)` produces a *list* of n independent standard normal random variables. Here `x` is a *list* and `x[k]` is element k of the list. Now we know several kinds of objects in `R`. If `x` is an object in `R`, then `x` could be a *number*, or `x` could be a `string` (character string), or `x` could be a list. The `x` here is a list of number, but it is possible to make a list of strings or even a list of lists. The `for` loop prints the numbers `x`. Then, a single `cat` statement prints the whole object `x`, which is a list of n numbers.

Example 2: This creates a list of n standard normals in a more complicated way. You don't have to understand this, but if you want to ... . The statement `rnorm(1)` first creates a list of one number. Then `x_k = rnorm(1)[1]` gets the first (and only) number in this list and gives the variable `x_k` that value. The variable name `x_k` is supposed to remind you of the mathematical notation x_k. The `R` function `c` makes a list out of the previous list `x` and the new element `x_k` by putting `x_k` on the end. So, if x is the list $(4, 6, 9)$, and x_k is 7, then `c( x, x_k)` is the list $(4, 6, 9, 7)$. The statement `x = c(x,x_k)` results in `x` being a list object with value $(4, 6, 9, 7)$. It *appends* the value of `x_k` to the end of the list `x`. The code in Example 2 first creates a list of length 1, then appends $n - 1$ random numbers to the list to make a list of length n. It seems more complicated in this case, but sometimes you don't know how long a list will be when you start, so you have to build the list by appending.

Example 3: This is the same as Example 1, except that the *seed* has been set to 1. Note that these are not the numbers you got the first two times.

Example 4: This is the same as Example 3, with the same seed. The numbers should be the same as Example 3.

Example 5: This is the same as Examples 3 and 4, but with a different seed. The numbers should be different from the Example 3 and Example 4 numbers.

Example 6: Here we create a big list with a lot of independent standard normal random variables. The `R` command `hist` creates a histogram plot. The code gives several arguments to this function, each on its own line for clarity. The first argument is the numbers, which are in the list `x`. The second argument is the `R` variable `breaks`, which is the number of bins to use. Using more bins makes each bin smaller. Experiment with different numbers of bins, say 10 and 100. The `R` variable `main` is what you want printed at the top of the histogram plot. You can see that we created a string called `Title` for this purpose. The `R` command that creates `Title` embeds the number of samples into the text. That's what `sprintf` is for. The last argument, `probability`, tells `R` to plot the estimated probability density instead of the bin counts. The picture should be approximately $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$. To check this, the code computes and prints $f(0) = \frac{1}{\sqrt{2\pi}}$. Check whether the printed number matches the plotted value for $x = 0$. Note that the `hist` statement appears twice in (almost) exactly the same form. The first time it makes a popup window on your screen with the histogram plot. The second time it makes a `.pdf` file called `NormalHistogram.pdf` with the same plot. The titles of the popup and the .pdf are slightly different. Look for the difference in the `R` code.

Example 7: This is a probability experiment in `R`. Suppose Y_j are independent standard normals, and X is the max of L of them. What is the PDF of X? This code does the probability experiment n times to create n *samples* of the X distribution. These numbers X_k are recorded in the `R` list `x`. Experiment with L. For large L the distribution is more narrow, as you can see by looking at the labels on the x axis. Try, for example, $L = 2$, $L = 10$, and $L = 100$. For $L = 2$, it isn't so unlikely for the maximum to be negative. That's very unlikely if $L = 10$, and nearly impossible for $L = 100$ (but not mathematically impossible). The bigger L is, the longer it takes the code to run. If the code is too slow with your desired L, reduce the number of samples, which is n.

Example 8: This shows how to generate *exponential* random variables. A random variable T is *exponential* with *rate constant* equal to λ if it has the PDF

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & \text{if } t > 0 \\ 0 & \text{if } t < 0 \,. \end{cases}$$

The code generates n independent exponential times, T_k, using the formula $T_k = -\frac{1}{\lambda}\log(U_k)$, where the U_k are independent and uniformly distributed in $[0, 1]$. You might know why this works, but you can see that it does work by looking at the histogram. The histogram will become more clear if you increase n. Try it. The sample code also uses the T_k to estimate $\Pr(T > t)$ for two values of t. The theoretical formula is

$$\Pr(T > t) = \int_t^\infty f(t')dt' = \lambda \int_t^\infty e^{-\lambda t'} dt' = e^{-\lambda t} \,.$$

The empirical estimate from the data is

$$\Pr(T > t) \approx \frac{\#\left\{T_k > t\right\}}{n}$$

On top on the right is the number of samples T_k so that $T_k > t$. The code prints the theoretical and empirical probability for two t values. The agreement is not great for $n = 1000$, but it gets better for larger n values.