HW: Week 7

36-350 – Statistical Computing

Week 7 – Spring 2021

Name: Cherie Hua

Andrew ID: cxhua

You must submit **your own** HW as a PDF file on Gradescope.

HW Length Cap Instructions

- If the question requires you to print a data frame in your solution e.g. `q1_out_df`, you must first apply **head(q1__out__df, 30)** and **dim(q1__out__df)** in the final knitted pdf output for such a data frame.
- Please note that this only applies if you are knitting the `Rmd` to a `pdf`, for Gradescope submission purposes.
- If you are using the data frame output for visualization purposes (for example), use the entire data frame in your exploration
- The **maximum allowable length** of knitted pdf HW submission is **30 pages**. Submissions exceeding this length *will not be graded* by the TAs. All pages must be tagged as usual for the required questions per the usual policy
- For any concerns about HW length for submission, please reach out on Piazza during office hours

Question 1

(20 points)

Let's say you have data sampled randomly from an unknown distribution. How can you estimate the probability that the next datum x_o that you observe will lie between the values $x = a$ and $x = b$?

One technique is to guess the distribution from a set of named ones, go through the optimization process, and then perform the necessary integral, by hand or with `integrate()`. But if you cannot guess the distribution, you could also use density estimation, i.e., nonparametrically estimate the probability density function using a kernel density estimator. Given the estimated pdf, you would then compute the required integral.

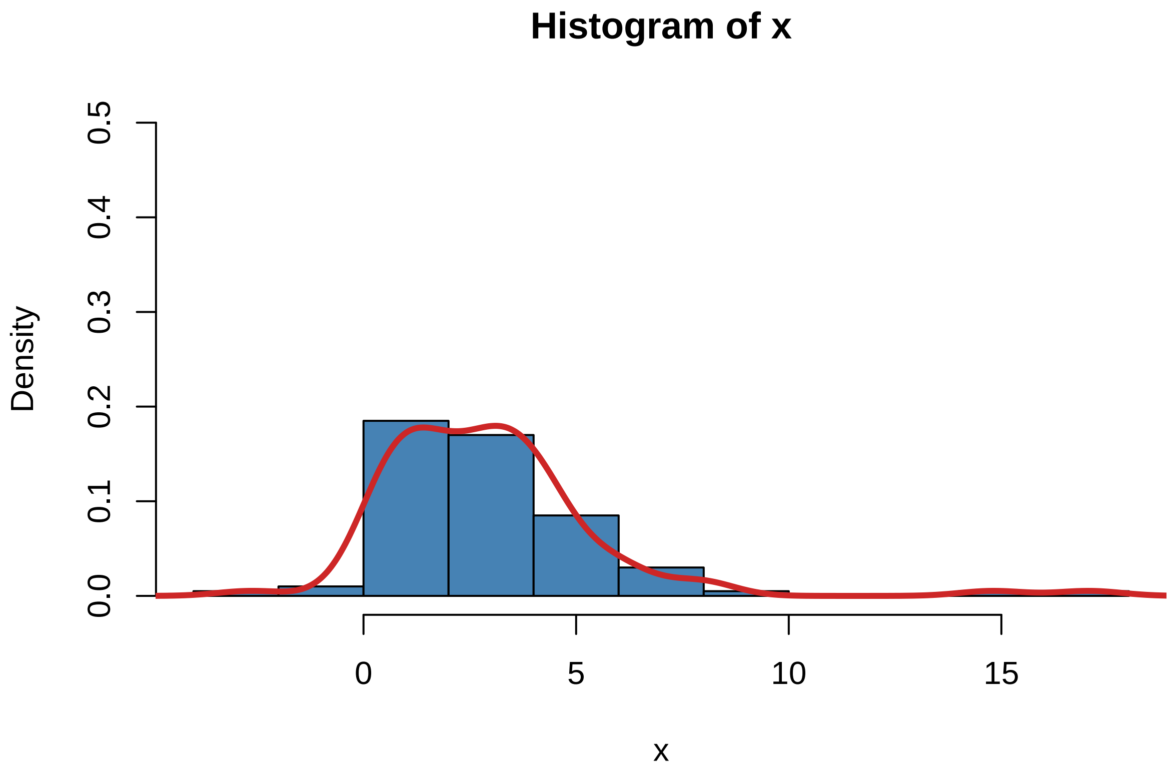
Below is a histogram for a data sample of size 100. Use the `density()` function to estimate the underlying pdf. By default, the weighting kernel is `"gaussian"` and the width of the kernel is computed via a "plug-in rule"; you do not need to change these default values. Overlay the estimated density function onto the histogram. Then compute the probability between $x = 0$ and $x = 4$, using the output from `density()`. You can code your own integrator, based on summations (trapezoid rule!), or utilize an appropriate function in the `sfsmisc` package that will integrate a function that is defined as a series of (x, y) pairs and not as a parametrized function. Note that my answer is approximately 0.666; you do not need to match this *exactly*, but your own answer should be pretty close. (Remember: in the end, this is only an estimate of the probability. Changing the width and/or form of the kernel will change your answer. Also, a kernel density estimate smooths data, so you can never actually recover the true underlying pdf anyway, just an approximation of it that is going to be slightly too wide. Still, this whole process beats having to fall back on Tchebysheff's theorem.)

```r
if ( require(sfsmisc) == FALSE ) {
  install.packages("sfsmisc",repos="https://cloud.r-project.org")
  library(sfsmisc)
}
```

```
## Loading required package: sfsmisc
```

```r
set.seed(303)
x = rgamma(100,2.5,scale=1.25)
s = sample(c(-1,1),100,replace=TRUE)
x = x + s*rexp(100,rate=3/abs(x))
hist(x,prob=TRUE,ylim=c(0,0.5),col="steelblue")

d = density(x)
lines(d, col="firebrick3",lwd=3)
```



```r
integrate.xy(d, x, 0, 4)
```

```
## [1] 0.6657188
```

Question 2

(20 points)

The `integrate()` function is used to perform univariate integrals of parametrized functions. What if you have a multi-dimensional integral instead, like

$$\int_0^{\pi/2} \int_0^{\pi/4} \sin(2x+y)\cos(x+2y) dx dy\,?$$

One option is to use the `cubature` package ("adaptive multivariate integration over hypercubes"). Install and use a function or functions of the `cubature` package to compute the integral given above. (Note: this package is useful for *hypercubes*; if the bounds of integration are variable [like when you dealt with triangular regions of integration in 225], then the functions of the `cubature` package are not appropriate. At that point, you would start moving towards, e.g., Monte Carlo integration.)

```
if ( require(cubature) == FALSE ) {
  install.packages("cubature",repos="https://cloud.r-project.org")
  library(cubature)
}
```

```
## Loading required package: cubature
```

```
hcubature(function(x){sin(2*x[1] + x[2])*cos(x[1]+2*x[2])}, lowerLimit = c(0, 0), upperLimit = c(pi/2, p
```

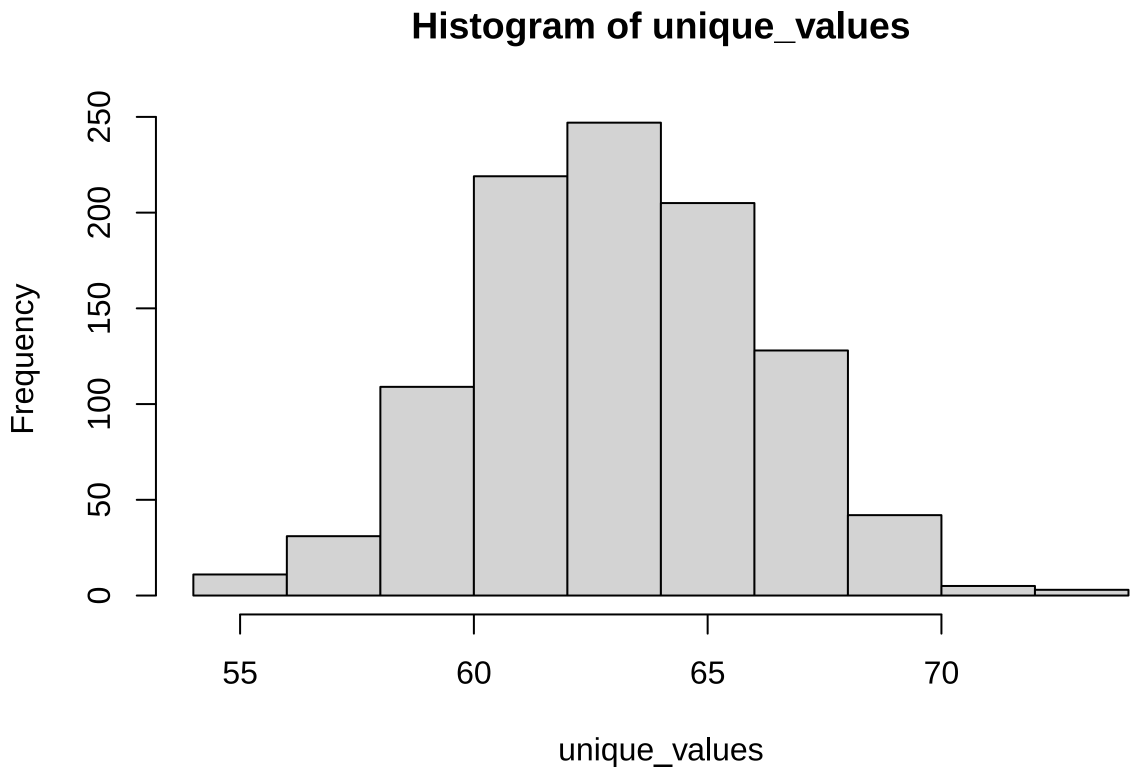
```
## $integral
## [1] 0.1515512
##
## $error
## [1] 1.50009e-06
##
## $functionEvaluations
## [1] 799
##
## $returnCode
## [1] 0
```

Question 3

(20 points)

If you sample 100 integers between 1 and 100 inclusive with replacement, how many unique integers are selected, on average? To answer this, do the following: (1) sample 100,000 integers between 1 and 100 inclusive with replacement; (2) place these samples into a 1000 × 100 matrix (so you've effectively simulated 1000 datasets with 100 samples each; (3) determine the number of unique values in each row and save the result to a vector; (4) histogram the result, properly labeled (base `R` or `ggplot()`); and (5) compute the sample mean and sample standard deviation of vector generated in step (3). If the mean is much different from 63.397, you've made a misstep. (Now, why are you doing this? Because the result tells you that in any one bootstrap sample of your data, there is a roughly 63% chance that a particular datum will be sampled one or more times. Or, alternatively, that roughly 37% of your data will not be sampled. Google, e.g., "out-of-bag error".)

```r
set.seed(123)
x = sample(1:100, 100000, replace = TRUE)
m = matrix(x, nrow = 1000, ncol = 100)
unique_values = apply(m, MARGIN = 1, FUN = function(x) {length(unique(x))})
hist(unique_values)
```



```r
cat("mean", mean(unique_values))
```

```
## mean 63.558
```

```r
cat("\nstandard deviation", sd(unique_values))
```

```
##
## standard deviation 3.054172
```

Question 4

(20 points)

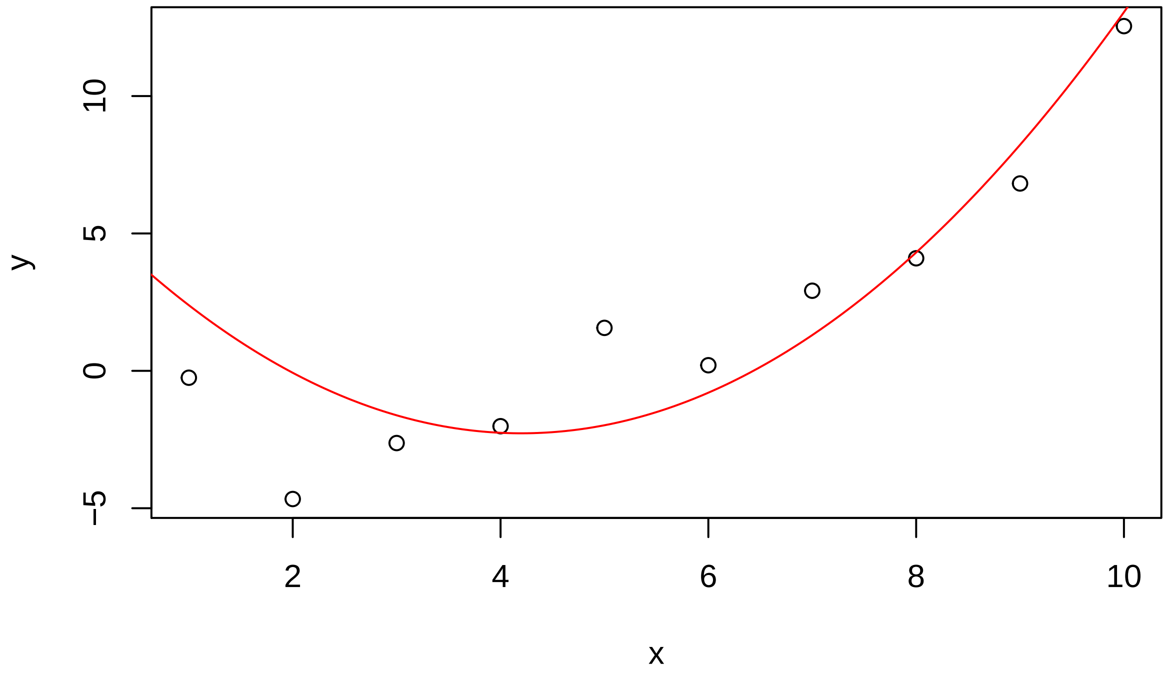
Below we load in a set of (x, y) data pairs:

```r
load(url("http://www.stat.cmu.edu/~pfreeman/HW_07_Q7.Rdata"))
```

Fit a quadratic function $f(x) = ax^2 + bx + c$ to these data using `optim()`, with a gradient function specified. Use `method = BFGS`, and instead of the negative log-likelihood, minimize the residual sum of squares (i.e., minimize the squared distance from the model to the data). Plot the data and overplot the best-fit line. Note that if that line does not match the data well, you might have a situation in which your initial guess of the parameter values was incorrect…so try fitting again with a new guess. (Hint: remember that the gradient is the gradient of the fit metric, not the derivative of $f(x)$, and that the gradient function returns a vector of length three: the partial derivative of the fit metric with respect to a, then with respect to b, then with respect to c. You'll need to derive the gradient vector by hand. Ask us sooner rather than later how to do this if you've forgotten…but you should have covered this in 21-259 or the equivalent.)

```
my.fit.fun = function(my.par, my.data) {
  y_hat <- my.par[1]*my.data$x^2 + my.par[2]*my.data$x + my.par[3]
  sum((my.data$y - y_hat)^2)
}
my.fit.gradient = function(my.par, my.data) {
  y_hat <- my.par[1]*my.data$x^2 + my.par[2]*my.data$x + my.par[3]
  c(sum(-2*my.data$x^2*(my.data$y - y_hat)), sum(-2*my.data$x*(my.data$y - y_hat)), -2)
}

my.par = c(1,1,1)
my.data = data.frame(x = x, y = y)
optim.out = suppressWarnings(optim(my.par, my.fit.fun, gr=my.fit.gradient, my.data=my.data, method="BFGS
plot(y ~ x, data = my.data)
x = seq(min(my.data),max(my.data),by=0.01)
lines(x, optim.out$par[1]*x^2 + optim.out$par[2]*x + optim.out$par[3], col = "red")
```



Question 5

(20 points)

Jumble, the scrambled-word game that is syndicated in many newspapers, can be hard. If we can have the computer play Jumble for us, well…it would be easier. Maybe not as fulfilling, but certainly easier.

Construct a function called `matchDict` that utilizes the `permn()` function of the `combinat` package and the 120,000+ length character vector `GradyAugmented` in the `qdapDictionaries` package so as to return possible unscrambled solutions given a scrambled input. `permn()` will output a list of all permutations of the letters in the input word; you need to combine these letters back into candidate words and determine which of the candidate words is in the `GradyAugmented` vector. (`permn()` is a useful tool for generating permutations, hence the idea for this exercise.) For instance, if you input "nidkr", you should get "drink" as your only output. Show output for "rneup", "srsets", and "LYPELU". Note: if your input is not a single string, have your function return `NULL`, and if your input is an upper-case string, convert it to lower-case: the `GradyAugmented` vector consists of lower-case strings. Also, return only unique instances of words. Hint: if you need to paste letters together, use `paste()` and look closely at its arguments to make sure you set the right argument to the right value.

```r
if ( require(combinat) == FALSE ) {
  install.packages("combinat",repos="https://cloud.r-project.org")
  library(combinat)
}
```

```
## Loading required package: combinat
```

```
##
## Attaching package: 'combinat'
```

```
## The following object is masked from 'package:utils':
##
##     combn
```

```r
if ( require(qdapDictionaries) == FALSE ) {
  install.packages("qdapDictionaries",repos="https://cloud.r-project.org")
  library(qdapDictionaries)
}
```

```
## Loading required package: qdapDictionaries
```

```r
matchDict = function(x) {
  if (is.character(x)) {
    x = tolower(x)
    all = permn(strsplit(x, "")[[1]])
    perms = lapply(all, paste, collapse = '')
    unique(perms[perms %in% GradyAugmented])
  } else {
    NULL
  }
}

matchDict("rneup")
```

```
## [[1]]
## [1] "prune"
```

```r
matchDict("srsets")
```

```
## [[1]]
## [1] "stress"
```

```r
matchDict("LYPELU")
```

```
## [[1]]
## [1] "pulley"
```

Question 6

(20 points)

The bootstrap is one type of what are called "resampling tests." Another type of resampling test is the permutation test. Assume you observe a series of (x, y) pairs; for instance, x might be a factor variable with two levels representing treatment groups, and y might be the observed responses. In this context, you might be interested in seeing if the difference in the mean values of y for each treatment group is either significantly different from zero (a two-sided hypothesis test) or significantly less than or greater than zero (a one-sided test). In a permutation test, you estimate the p-value by randomly shuffling the x vector (while leaving the y vector intact!), then computing the difference, and repeating until you build up a vector of differences, and seeing, e.g., how many of the differences generated via permutation are greater than the actual difference you observe.

In the code chunk below, x and y are the observed data. Generate the observed difference between the means for the groups $x = 0$ and $x = 1$. Then code a permutation test. Note that you cannot use `permn()` here to generate all the possible permutations for the x vector, as that would be $\sim 10^{18}$ permutations. So: generate 10,000 permutations, save the differences for each, and see how many of the 10,000 differences are greater than what you observe. In the end, do you reject or fail to reject the null hypothesis that the actual difference in population means is zero?

```r
set.seed(1002)
x = c(rep(0,10),rep(1,10))
y = x+rnorm(20,sd=1.5)
observed.diff = diff(tapply(y,x,mean))

indices = sample(length(y), 10000*length(y), replace = TRUE)
data.array = matrix(y[indices], nrow = 10000)
sample.diffs = apply(data.array, MARGIN = 1, FUN = function(y){diff(tapply(y,x,mean))})
num_greater = sum(sample.diffs > observed.diff)
p = num_greater/10000
p
```

```
## [1] 0.0321
```

```
Since the p value is 0.0321, I reject the null hypothesis that the
actual difference in population means is zero.
```