Lab 6: Simulations

INSERT YOUR NAME HERE (INSERT YOUR UW NETID HERE)

Due by 23:59pm on Feb 29, 2024

Total Points: 45

Part 1. Random Number Generation and ECDFs (4+5+5 pts)

1. Generate the following objects, save them to variables (with names of your choosing), and call `head()` on those variables.
   - A vector with 60 draws from Beta$(0.1, 0.1)$.
   - A vector of 200 characters sampled with replacement uniformly from "A", "G", "C", and "T".
   - A data frame with a column `x` that contains 100 draws from Unif$(0, 3)$, and a column `y` that contains 100 draws of the form $y_i \sim \text{Unif}(0, x_i)$, where x_i is the i-th element of column `x`. Do this without using explicit iteration.

```
set.seed(123)  # Don't change this line
# Your code goes here
```

2. You are given a function called `plotCumMeans()` to plot the cumulative sample mean as the sample size increases.

- The first argument `rfun` stands for a function which takes one argument `n` and generates this many random numbers when called as `rfun(n)`.
- The second argument `n.max` is an integer which tells the number samples to draw. As a side effect, the function plots the cumulative mean against the number of samples.

```
# plotCumMeans: plot cumulative sample mean as a function of sample size
# Inputs:
# - rfun: function which generates random draws
# - n.max: number of samples to draw
# Ouptut: none
plotCumMeans = function(rfun, n.max) {
  samples = rfun(n.max)
  plot(1:n.max, cumsum(samples) / 1:n.max, type = "l",
       xlab = "Cumulative sample size", ylab = "Sample mean")
}
```

Use this function to make plots for the following distributions, with `n.max = 20000`. Then answer: do the sample means start concentrating around the appropriate value as the sample size increases?

- $N(-3, 10)$.
- Exp(mean $= 5$).
- Cauchy(location $= -1$, scale $= 2$).

Hint: for each case, you should construct a new anonymous function to pass as the `rfun` argument to `plotCumMeans()`. For instance, you should pass `function(n) rnorm(n, mean = -3, sd = sqrt(10))` as the `rfun` argument to `plotCumMeans()` for the first case.

```r
set.seed(123)  # Don't change this line
# Your code goes here
```

3. For the same distributions in Part 1-2 above, we will do the following.
   - Generate 10, 100, 1000 random samples from each of the following distributions.
     - $N(-3, 10)$, where 10 is the variance.
     - Exp(mean $= 5$).
   - Generate 10, 50, 100 random samples from the following distribution.
     - Cauchy(location $= -1$, scale $= 2$).
   - On a single plot, display the ECDFs (empirical cumulative distribution functions) from each set of samples, and the true CDF, with each curve being displayed in a different color.

In order to do this, we will write a function `plotECDF(rfun, pfun, sizes)` which takes as its arguments the single-argument random number generating function `rfun`, the corresponding single-argument cumulative distribution function `pfun`, and a vector of sample sizes `sizes` for which to plot the ecdf.

We've already started to define `plotECDF()` below, but we've left it incomplete. Fill in the definition by editing the lines with "##" and "??", and then run it on the same distributions as in Part 1-2.

- Examine the plots and discuss how the ECDFs converge as the sample size increases.

Note: make sure to remove `eval=FALSE`, after you've edited the function, to see the results.

```r
# plotECDF: plots ECDFs along with the true CDF, for varying sample sizes
# Inputs:
# - rfun: function which generates n random draws, when called as rfun(n)
# - pfun: function which calculates the true CDF at x, when called as pfun(x)
# - sizes: a vector of sample sizes
# Output: none

plotECDF = function(rfun, pfun, sizes) {
  # Draw the random numbers
  ## samples = lapply(sizes, ??)

  # Calculate the grid for the CDF
  grid.min = min(sapply(samples, min))
  grid.max = max(sapply(samples, max))
  grid = seq(grid.min, grid.max, length=1000)

  # Calculate the ECDFs
  ## ecdfs = lapply(samples, ??)
  evals = lapply(ecdfs, function(f) f(grid))

  # Plot the true CDF
  ## plot(grid, ??, type="l", col="black", xlab="x", ylab = "P(X <= x)", lwd=3)

  # Plot the ECDFs on top
  n.sizes = length(sizes)
  cols = rainbow(n.sizes)
  for (i in 1:n.sizes) {
    lines(grid, evals[[i]], col=cols[i], lwd=1.5)
  }
  legend("topleft", legend=sizes, col=cols, lwd=3)
}
```

```r
set.seed(123)  # Don't change this line
```

```
# Your code for plotting goes here
```

Part 2. Drug Effect Simulation (2+2+3+4+2+3 pts)

We continue studying the drug effect model that was discussed in Lecture 6 Slides. Recall that we believe those who are not given the drug experience a reduction in tumor size of percentage as:

$$X_{\text{no drug}} \sim 100 \cdot \text{Exp}(\text{mean} = R), \quad R \sim \text{Unif}(0, 1),$$

whereas those who were given the drug experience a reduction in tumor size of percentage as:

$$X_{\text{drug}} \sim 100 \cdot \text{Exp}(\text{mean} = 2).$$

1. Look the code chunk in the lecture that generated data according to the above model. Write a function around this code called `simulateData()` that takes two arguments:

- `n`: the sample size (number of subjects in each group) with a default value of 60;
- `mu_drug`: the mean for the exponential distribution that defines the drug tumor reduction measurements with a default value of 2.

Your function should return a list with two vectors called `no_drug` and `drug`. Each of these two vectors should have length `n`, containing the percentage reduction in tumor size under the appropriate condition (not taking the drug or taking the drug).

```
# Your code goes here
```

2. Run your function `simulateData()` without any arguments (hence, relying on the default values of `n` and `mu_drug`), and store the output in `results1`. Print out the first 6 values in both the `results1$no_drug` and `results1$drug` vectors. Now, run `simulateData()` again, and store its output in `results2`. Again, print out the first 6 values in both the `results2$no_drug` and `results2$drug` vectors. We have effectively simulated two hypothetical datasets. Note that we shouldn't expect the values from `results1` and `results2` to be the same.

```
set.seed(123)  # Don't change this line
# Your code goes here
```

3. Compute the following three numbers: the absolute difference in the mean values of `no_drug` between `results1` and `results2`, the absolute difference in the mean values of `drug` between `results1` and `results2`, and the absolute difference in mean values of `no_drug` and `drug` in `results1`.

Answer in words: Of these three numbers, which one is the largest, and does this make sense?

```
# Your code goes here
```

4. Now, we want to visualize the simulated data. Fortunately, the code to visualize the data is already provided for you in our Lecture 6 slides. Write a function around this code, called `plotData()` that takes just one argument `data`, which is a list with components `drug` and `no_drug`.

To be clear, this function should create a single plot, with two overlaid histograms, one for`data$no_drug` (in gray) and one for `data$drug` (in red), with the same 20 bins. It should also overlay a density curve for each histogram in the appropriate colors, and produce a legend. Once written, call `plotData()` on both `results1` and `results2`.

```
# Your code goes here
```

5. Generate a new simulated dataset using `simulateData()` where `n=1000` and `mu_drug=1.1`, and plot the results using `plotData()`. In one or two sentences, explain the differences that you see between this plot and the two you produced in the last problem.

```
# Your code goes here
```

6. Combine the `simulateData()` function, where `n=2000` and `mu_drug=1.6`, with the `replicate()` function that we learned in Lecture 6 slides to generate 3000 simulated datasets and compute the absolute difference in mean values of `no_drug` and `drug` for each of the simulated dataset. Output both the mean and variance of all these absolute differences.

```
set.seed(123)  # Don't change this line
# Your code goes here
```

Part 3. Bootstrap (3+7+5 pts)

In this problem, you will use the nonparametric/empirical and parametric bootstrap to estimate the mean of an exponential distribution Exponential(λ) with density

$$f(x) = \lambda \exp(-\lambda x) \quad \text{for} \quad x \geq 0.$$

Suppose that you are given some data `X_dat` from an exponential distribution with rate $\lambda = \frac{1}{3}$ as follows. You may need to read about the relation between the rate λ of the exponential distribution Exponential(λ) and its mean by typing `?rexp`.

```
# Don't change this chunk
set.seed(123)
X_dat = rexp(100, rate = 1/3)
```

1. Compute and output the empirical mean $T = \bar{X}_n$ and its standard error of `X_dat`. In addition, plot the histogram of `X_dat` with `freq=FALSE`.

```
# Your code goes here
```

2. Use the nonparametric bootstrap with the re-sampling times as $B = 3000$ to estimate and output the standard error for T. In addition, output the normal, pivotal, and percentile confidence intervals with the nominal level **90%**.

- Answer in words: Do all the confidence intervals cover the empirical mean $T = \bar{X}_n$ of `X_dat`?
- Plot the histogram of these $B = 3000$ bootstrap statistics with `freq=FALSE`. Describe in words about its shape.

```
set.seed(123)  # Don't change this line
# Your code goes here
```

3. Assume that you know the data `X_dat` come from an exponential distribution Exponential(λ), but its rate λ is unknown. Estimate the rate λ by the empirical mean $T = \bar{X}_n$. Then, use the parametric bootstrap with the re-sampling times as $B = 3000$ to estimate and output the standard error for T. In addition, output the normal, pivotal, and percentile confidence intervals with the nominal level **90%**.

```
set.seed(123)  # Don't change this line
# Your code goes here
```

Part 4. Validity of the Bootstrap Confidence Intervals (Extra Credit: 5 pts)

Note: The extra credit in the problem can only be applied to any of your lost points in this Lab 6 assignment. The total points that you can earn for Lab 6 are still capped at 45.

We again use the exponential distribution with rate $\lambda = \frac{1}{3}$ to check the validity of normal, pivotal, and percentile confidence intervals based on the nonparametric bootstrap. Taking the normal confidence interval as an example, we do the following procedure.

1. Generate `X_dat` from an exponential distribution with $n = 100$ and rate $\lambda = \frac{1}{3}$ and compute the mean statistic $T = \bar{X}_n$ of `X_dat`.
2. Use the nonparametric bootstrap with the re-sampling times as $B = 3000$ to construct the normal confidence interval with nominal level 90%.
3. Repeat Steps 1 and 2 for $N = 600$ times and compute the proportion of normal confidence intervals covering the true mean $\frac{1}{\lambda} = 3$. Is this proportion close to 90%? (Hint: You may need two nested `replicate()` functions or use a `replicate()` function within the `for` loop.)

Do the same procedure for the pivotal and percentile confidence intervals with nominal level 90%. Which type of the confidence interval is closest to 90%?

```
set.seed(123)  # Don't change this line
# Your code goes here
```