Homework 6

For this homework you will create an R Markdown file and output (PDF file) and upload both to wolfware. Be sure to include text explaining your thought process/what you are doing with your questions.

The purpose of this homework is to get practice with writing functions, vectorized functions, and parallel programming.

Basic t-test

We are going to look at the performance of the usual one-sample t-test under incorrect assumptions. That is, we are going to see how well the t-test works at controlling the significance level (proportion of falsely rejected null hypotheses).

For more background, we often want to test competing hypotheses about population means. Call that population mean μ. We often test

$$H_0 : \mu = \mu_0 \text{ vs } H_A : \mu \neq \mu_0 \text{ (or } \mu > \mu_0 \text{ or } \mu < \mu_0)$$

We then make assumptions about the data generating process and conduct the test. The null hypothesis is assumed and we see if the data provides evidence against it.

We want to control against making too many errors. There are two types of errors:

- Type 1: Rejecting H_0 when H_0 is true
- Type 2: Failing to reject H_0 when H_A is true

The type 1 error is usually considered the worse type of error so tests are set up to control the probability of making this type of error (called the significane level). We define

$$\alpha = P(\text{making a type I error}) = P(\text{Rejecting } H_0 \text{ given } H_0 \text{ is true})$$

The most commonly used test for this situation is the t-test. The t-test makes a number of assumptions:

- The data is a random sample
- The data is sampled from a normal distribution

Under these assumptions we can use the t-statistic as our test statistic

$$t_{obs} = \frac{\bar{y} - \mu_0}{s/\sqrt{n}}$$

If we use the following rejection rules, we will control the type I error rate at α (where α is chosen prior to the experiment, often chosen to be 0.05):

$$\begin{aligned}
H_A : \mu \neq \mu_0 &\text{ reject if } |t_{obs}| > 0.975 \text{ quantile of a t-distribution with n-1 degrees of freedom} \\
H_A : \mu < \mu_0 &\text{ reject if } t_{obs} < 0.05 \text{ quantile of a t-distribution with n-1 degrees of freedom} \\
H_A : \mu > \mu_0 &\text{ reject if } t_{obs} > 0.95 \text{ quantile of a t-distribution with n-1 degrees of freedom}
\end{aligned}$$

This all works if our assumptions are true! In the simulation study, we'll look at how well the significance level is controlled at α when our data is a random sample from another distribution.

Writing your own functions to do a t-test

Let's do this from scratch since it isn't too difficult!

Write a function to calculate the test statistic.

- Inputs should be a vector of numeric data and the mean value to compare against (μ_0)
- The output should be the calculated t_{obs} value

Write a function to determine whether you reject H_0 for fail to reject it.

- Inputs should be the test statistic value, the sample size, the significance level (α), and the direction of the alternative hypothesis (left, right, or two-sided).
- The output should be a `TRUE` or `FALSE` value depending on whether or not you reject the null hypothesis. Note: `qt()` will give quantiles from the t-distribution.

Test how well your function works! Use the built-in `iris` data set and run a few hypothesis tests. Using the data, determine if the true mean

- sepal length differs from 5.5 (i.e. test $H_0 : \mu = 5.5$ vs $H_A : \mu \neq 5.5$ and report whether or not we reject H_0)
- sepal width is greater than 3.5
- petal length is less than 4

Quick Monte Carlo study

A Monte Carlo Simulation is one where we generate (pseudo) random values using a random number generator and use those random values to judge properties of tests, intervals, algorithms, etc.

We'll generate data from a **gamma** distribution using different `shape` parameters and sample sizes. We'll then apply the t-test functions from above and see how well the α level is controlled under the incorrect assumption about the distribution our data is generated from.

Pseudocode:

- generate a random sample of size n from the gamma distribution with given `shape` and `rate` parameters specified (see `rgamma()`)
- find the test statistic using the actual mean of the randomly generated data for μ_0 (i.e $\mu_0 = shape * rate$ - implying the null hypothesis H_0 is true)
- using the given alternative (left, right, or both) determine whether you reject or not
- repeat many times
- observe the proportion of rejected null hypothesis from your repetitions (these are all incorrectly rejected since the null hypothesis is true). This proportion is our Monte Carlo estimate of the α value under this data generating process.

Write code to do the above when sampling from a gamma distribution with `shape = 0.5` and `rate = 1` with a sample size of `n = 10` for a two-sided test. Use the `replicate()` function (a wrapper for the `sapply()` function) to do the data generation and determination of reject/fail to reject with relative ease (with the number of replications being 10000)! Then find the mean of the resulting `TRUE/FALSE` values as your estimated α level under these assumptions.

Parallel Computing

Now, we may want to do the above Monte Carlo simulation for several settings of sample size, shape, and rate parameter. As it turns out, we really don't need to worry about the rate parameter, it just scales the distribution.

Ok, to do:

- create a vector of sample size values (10, 20, 30, 50, 100)
- create a vector of shape values (0.5, 1, 2, 5, 10, 20)

- create a rate vector with the value 1

We want to be able to use `parLapply()` to execute the above Monte Carlo study for combinations of sample sizes and shape values (with rate always 1). To use `parLapply()`, we need to pass

- our cluster set up via `makeCluster()`
  - We must also use `clusterExport()` function to pass a list of our functions that we created
  - If you use any packages, you'll need to use `clusterEvalQ()` to pass the package
- `X` a list we will parallelize our computations over (each list element would be a combination of sample size and shape parameter)
- `fun` a function that does the replication of gamma samples, finds the decisions for each sample, and calculates and returns the proportion
  - Create a function to do this that just uses the `replicate()` and `mean()` code above
- any other arguments needed for our functions to run (for instance `rate = 1`)

Here is an example of part of the list you want to parallelize over but you'll need to figure out a way to create it (think `expand.grid()` and `lapply()`)

```
## [[1]]
## [[1]]$n
## [1] 10
##
## [[1]]$shape
## [1] 0.5
##
##
## [[2]]
## [[2]]$n
## [1] 20
##
## [[2]]$shape
## [1] 0.5
##
##
## [[3]]
## [[3]]$n
## [1] 30
##
## [[3]]$shape
## [1] 0.5
##
##
## [[4]]
## [[4]]$n
## [1] 50
##
## [[4]]$shape
## [1] 0.5
```

Finally, print out a table giving the results in a reasonably palatable manner (here I used `pivot_wider()` on the results).

```
## # A tibble: 5 x 7
##       n `shape = 0.5` `shape = 1` `shape = 2` `shape = 5` `shape = 10`
##   <dbl>         <dbl>       <dbl>       <dbl>       <dbl>        <dbl>
## 1    10         0.138      0.0938      0.0794      0.0595       0.0559
## 2    20         0.105      0.0823      0.0658      0.0575       0.0581
## 3    30        0.0931      0.0717      0.0608      0.0543       0.0502
## 4    50        0.0718      0.0615      0.0573      0.0496       0.054
## 5   100        0.0675      0.0574      0.0518      0.0516       0.0477
## # ... with 1 more variable: `shape = 20` <dbl>
```