Stat 133, Fall 2014
Homework 5: Web Caching PART 2: Functions and simulation
Due Thursday Oct 9 at 11:55pm on bspace.

About the data

This homework is a continuation of the previous homework. We will be using the same websites data, `Cache500`. Load the data into R from the site `http://www.stat.berkeley.edu/users/nolan/data/Cache500.rda`

In this assignment, we'll look at using simulation to help us determine which websites have changes that fall very far from what we'd expect from a uniform distribution. Specifically, we'll look at using the maximum difference from theoretical uniform quantiles as a statistic to measure deviation from the uniform distribution. We'll simulate what how this statistic behaves when the data is actually uniform, to get a sense of how much variability we can expect by chance. We'll then compare our real data results to the simulated distribution of max differences, and pick an extreme website to examine more closely by quantile plots.

What to turn in

Like last week, I have provided a skeleton .R file for you to work from. Turn in just a completed version of this .R file with the name `[last name][first name]_HW5.R`, e.g. "`IbserHank_HW5.R`." As you will check again in problem 5, your .R file needs to run error-free from top to bottom and produce two .png files.

Assignment

Problem 0

Just like in HW 4, before starting the assignment, create a separate folder and save your .R file in there. Set this to be your working directory in R. In RStudio, you can do this by going to `Tools > Change Working Directory`. If you are more comfortable with specifying the filepath, you can run `setwd("filepath_of_my_folder")` in the console, but do not put this in your .R file.

As before, we are doing this because we'll be writing plots straight to files in your working directory; this makes sure you don't lose track of where all of this output is going.

Problem 1: Understanding calcMaxDiff, a function to compute the largest difference from theoretical uniform quantiles

In your skeleton file, there is a function called calcMaxDiff. Leave it there. You don't have to do anything for this question except read the function in and understand what it does, which we'll explain here:

- `calcMaxDiff` creates a vector that takes the original data for a website and divides it by 720. The result is a vector where the last value is close to 1 and the first value is close 0.

- `calcMaxDiff` creates a vector of the same length as the previous, which goes from 0 to 1 in equal increments.
- It takes the maximum (absolute) difference between these two vectors. If the data follow the uniform distribution closely, these differences should be small, and thus the maximum distance should be small.
- It uses `return()` to give the maximum difference as output.

Again, you don't have to do anything for this question.

Problem 2: Write a function to simulate the null distribution of maximum differences and compare a website to it

Write a function called `simMaxDiff`. This function should do the following:

2a. Take two arguments as input. The first argument, `x`, will be a numeric vector (e.g., an entry of Cache500, like Cache500[[8]]). The second argument `n` will be a number, and its default value should be set to 1000.

2b. Create a matrix called `simUnifMatrix` which contains random uniform integer-valued data from 1 to 720 (alternatively, you can think of this as drawing tickets from a box with numbers labeled with the integers from 1 to 720). There should be `n` columns, and as many rows as the length of the data vector `x`. Essentially, each column is one simulation of that website's data as though it came from a uniform distribution.

2c. Use your answer from 2b, `apply()`, and the function `calcMaxDiff()` to obtain a vector of length `n` which gives the maximum difference from the uniform quantiles for each column of simulated data. Assign this to `simMaxDiffs`.

2d. Find the maximum difference of the data vector `x` from the uniform quantiles and assign this to `actualMaxDiff`.

2e. Use `rank()` to find the rank of the data's maximum difference (`actualMaxDiff`) among the simulated ones (`simMaxDiffs`). The answer to this question should be a number between `1` and `n+1`: the website should have rank 1 if its maximum difference is smaller than all of the simulated ones, and n+1 if its maximum difference is bigger than all of the simulated ones. Assign this value to `simRank`.

As a concrete example, suppose my website's maximum difference is 0.4, and I simulate the differences (0.2, 0.3, 0.5). My website's rank should be 3, because it is the third largest out of the total four differences I have.

2f. Use `return()` to return the value of `simRank` as output from this function.

Problem 3: Write a function to apply the simulation to a set of websites, and examine whether these websites follow a uniform distribution on the whole

Write another function called `histRanks()`. This function should do the following:

3a. Take two arguments. The first argument `x` will be a list of numeric vectors, for example `Cache500`. The second argument `n` will be a number, and its default value should be set of 1000, as before.

3b. Use `sapply()` and the function you wrote in Problem 2, `simMaxDiff()`, to obtain a vector of simulated ranks for each website in `x`. Name this vector `websiteRanks`.

3c. Scale these ranks by dividing by `n+1`, and assign the resultant vector to `scaledRanks.`

3d. Plot a histogram of the scaled ranks to a PNG with the filename `LastnameFirstname_Hist.png.` Use the `breaks` argument in `hist()` to make it so that one bar has an edge at 0.95. Further, add a vertical red line through 0.95.

3e. Use `return()` to pass the vector of scaled ranks as output.

Problem 4: Apply your functions to a subset of Cache500 data

A vector called `whichOK` has already been defined for you in the skeleton .R file. This is the same `whichOK` vector as in HW 4, but I gave it to you here so you wouldn't have to do that part of the assignment again. :)

Apply `histRanks()` to `whichOK` and assign the output to `whichOKRanks`.

Problem 5: Examine a website with non-uniform changes more closely with a quantile plot

We wrote these functions to help us establish which websites have changes that follow a uniform distribution. Those to the right of the red line in your histogram are so extreme that, roughly speaking, you'd expect to see results like that less than 5% of the time if the data were truly generated from a uniform distribution.

From the output of Problem 4, choose a website that has a scaled rank of 1. This means that the largest difference in the website's quantiles from the theoretical uniform quantiles was bigger than those found in `n` simulations from an actual uniform, We'll make the same quantile plots as in HW 4, so we can examine how this website's changes differ from a uniform distribution.

5a. Assign the index of the site you chose to `hiRankSite`, i.e. if this is the 8th site in `Cache500`, `hiRankSite` should be 8.

5b. Create a vector of 0.01 uniform quantiles called `unifQuantiles`, just as in HW 4.

5c. Find the 0.01 quantiles of the website changes, and assign them to `hiRankQuantiles`.

5d. Plot the site quantiles against the uniform quantiles. Add two blue horizontal lines indicating the range of time where the website's changes looked very different from a distribution. Use `text()` to write "more than unif." or "less than unif." in blue to indicate if the website changed more or less than expected under a uniform distribution, respectively. The text can be anywhere there's white space in the plot. Write the plot to a PNG with the filename `LastnameFirstname_QQ.png`.

Problem 6: Check code reproducibility

Check that your code is reproducible by making sure it can be run from top to bottom. There are NO commands to turn in for this problem, but you will lose points if your code produces errors or does not pass the basic checks in the test file when the grader runs your code.

6a. As in HW 4, save your .R file and clear your workspace by running `rm(list=ls())` (do not include this in your .R file!). Delete any .pngs that were rendered to your working directory.

6b. Set your working directory to the folder where your .R file lives (and where you've been saving your .pngs). Run your .R file by typing `source("name_of_my_R_file.R")`. If this gives you errors, you'll need to go back and address them before you can proceed to the next part. Also check that your code produces two .png files: `LastnameFirstname_Hist.png` and `LastnameFirstname_QQ.png`

6c. At this point, your .R file should run error-free from top to bottom, producing three .png files in the working directory along the way. Download `HW5_tests.R` and save it in the same directory. Clear your workspace by running `rm(list=ls())`. Source this file by running `source("HW5_tests.R")`. Run `testHW5("name_of_my_R_file.R")`. As in HW 4, check that each test passes. Submit only your completed .R file, no plots: don't forget to change the name to `LastnameFirstname_HW5.R`.