CS 504 – Programming Languages for Data Analysis

Assignment 3: Non-parametric Sign Test

I. Problem: Non-parametric Sign Test

The validity of the T-test relies heavily on the assumption that the sample $X_1, \dots, X_n$ is comprised of independent normal random variables. This is because only under this assumption does the T-statistic follow a T-distribution. This assumption may often be safely made, however in certain cases we cannot assume a normal population and we need an alternative test.

Here we present a particular type of non-parametric test known as the sign test. The phrase "*non-parametric*" implies that the distribution of the test statistic does not depend on any particular distributional assumption for the population. For the sign test, we begin by denoting the random variables,

$$X^+ = \sum_{i=1}^{n} 1\{X_i > \mu_0\} \;\; and \;\; X^- = \sum_{i=1}^{n} 1\{X_i < \mu_0\} = n - X^+$$

where $1\{.\}$ is the indicator function. The variable X^+ is a count of the number of observations that exceed μ_0, and similarly, X^- is a count of the number of observations that are below μ_0.

Observe that under $H_0: \mu = \mu_0$, it holds that $P(X_i > \mu_0) = P(X_i < \mu_0) = 1/2$. Note that here we are actually taking μ_0 as the median of the distribution and assuming that $P(X_i = \mu_0) = 0$ as is the case for a continuous distribution. Hence under H_0 the random variables X^+ and X^- both follow a binomial$(n, 1/2)$ distribution. Given the symmetry of this binomial distribution we define the test statistic to be,

$$U = max\{X^+, X^-\}$$

Hence with observed data, and an observed test statistic *u*, the *p-value* can be calculated via,

$$p = 2P(B > u)$$

where B is a binomial$(n, 1/2)$ random variable. Here, under H_0, p is the probability of getting an extreme number of signs greater than u (either too many via X^+ or a very small number via X^-). The test procedure is then to reject H_0 if $p < \alpha$.

In the following code we present an example where we calculate the value of the test statistic and its corresponding p-value manually using Julia. We then use these to make conclusions about the null hypothesis at the 5% significance level. We compare two hypothetical cases. In the first case $\mu_0 = 52.2$, and the second $\mu_0 = 53.0$. As can be observed from the output, the former case is significant (H_0 is rejected) while the latter is not, as the test statistic of 11 is not non-plausible under H_0.

```
1 using CSV, Distributions, HypothesisTests
2
3  data = CSV.read("../data/machine1.csv", header=false)[:,1]
4 n = length(data)
5 mu0A, mu0B = 52.2, 53
6
7 xPositiveA = sum(data .> mu0A)
8 testStatisticA = max(xPositiveA, n-xPositiveA)
9
10 xPositiveB = sum(data .> mu0B)
11 testStatisticB = max(xPositiveB, n-xPositiveB)
12
13 binom = Binomial(n,0.5)
14 pValA = 2*ccdf(binom, testStatisticA)
15 pValB = 2*ccdf(binom, testStatisticB)
16
17 println("Binomial mean: ", mean(binom))
18
19 println("Case A: mu0: ", mu0A)
20 println("\tTest statistc: ", testStatisticA)
21 println("\tP-value: ", pValA)
22
23 println("Case B: mu0: ", mu0B)
24 println("\tTest statistc: ", testStatisticB)

25 println("\tP-value: ", pValB)
```

Outputs:

```
Binomial mean: 10.0

Case A: mu0: 52.2

Test statistc: 15

p-value: 0.011817932128906257

Case B: mu0: 53
```

```
Test statistc: 11

p-value: 0.5034446716308596
```

In line 5 the value of the population mean under the null hypothesis for both cases, mu0A and mu0B, is specified. In lines 7–11 the observed test statistics for both cases are calculated via $U = max\{X^+, X^-\}$. Note the use of .> for comparing the array data element wise with the scalars *mu0A* and mu0B. In lines 13–15, *Binomial()* and *ccdf()* are used used to compute the *p-values* for both cases in via $p = 2P(B > u)$. The results are printed in lines 19–25. As there are *n = 20* observations the binomial mean is *10*.

Sign Test vs. T-Test

With the sign test presented as a robust alternative to the T-test, one may ask why not simply always use the sign test. After all, the validity of the T-test rests on the assumption that $X_1, \dots, X_n$ are normally distributed. Otherwise, T of ($T = \frac{\bar{X}-\mu_0}{S/\sqrt{n}}$) does not follow a *T-distribution*, and conclusions drawn from the test may be potentially imprecise.

One answer is due to the statistical power of the test. As we show in the example below, the *T-test* is a more powerful test than the sign test when the normality assumption holds. That is, for a fixed α, the probability of detecting H_1 is higher for the T-test than for the sign test. This makes it a more effective test to use, if the data can be assumed normally distributed.

In the following code we perform a two-sided hypothesis test for $H_0: \mu = 53$ vs. $H_1: \mu \neq 53$ via both the T-test and sign test. We consider a range of scenarios where we let the actual μ vary over $[51.0, 55.0]$. When $\mu = 53$, H_0 is the case, however all other cases fall in H_1. On a grid of such cases we use Monte Carlo to estimate the power of the tests (for $\sigma = 1.2$). The resulting curves in Figure 1 show that the T-test is more powerful than the sign test.

```
1 using Random, Distributions, HypothesisTests, Plots; pyplot()
2
3 muRange = 51:0.02:55
4 n = 20
5 N = 10^4
6 mu0 = 53.0
7 powerT, powerU = [], []
8
9 for muActual in muRange
10
```

```
11 dist = Normal(muActual, 1.2)
12 rejectT, rejectU = 0, 0
13 Random.seed!(1)
14
15 for _ in 1:N
16 data = rand(dist,n)
17 xBar, stdDev = mean(data), std(data)
18
19 tStatT = (xBar - mu0)/(stdDev/sqrt(n))
20 pValT = 2*ccdf(TDist(n-1), abs(tStatT))
21
22 xPositive = sum(data .> mu0)
23 uStat = max(xPositive, n-xPositive)
24 pValSign = 2*cdf(Binomial(n,0.5), uStat)
25
26 rejectT += pValT < 0.05
27 rejectU += pValSign < 0.05
28 end
29
30 push!(powerT, rejectT/N)
31 push!(powerU, rejectU/N)
32
33 end
34
35 plot(muRange, powerT, c=:blue, label="t test")
36 plot!(muRange, powerU, c=:red, label="Sign test",
37                 xlims=(51,55), ylims=(0,1),
38                 xlabel="Different values of muActual",
39                 ylabel="Proportion of times H0 rejected", legend=:bottomleft)
```

In lines 3–6 we setup the basic parameters. The sample size n is 20. The range muRange represents the range $[51.0,55.0]$ in discrete steps of 0.02. The number N is the number of simulation repetitions to carry out for each value of μ in that range. The value *mu0* is the value under H_0. In line 7 we initialize empty arrays that are to be populated with power estimates. Lines 9–33 contain the main loop where each discrete step in *muRange* is tested. In line 11 for each value a distribution *dist* is created with the same standard deviation, 1.2, but with a different mean, *muActual*. The inner loop of lines 15–28 is a repetition of the sampling experiment *N* times where for the same data we compute *tStat*, *uStat*, and the corresponding p-values. Rejection counts are accumulated in lines 26 and 27, where *pValT<0.05* and *pValSign<0.05* constitutes a rejection. Note *Random.seed!()* is used in Line 13 so to obtain a smoother curve via common random numbers. Lines 30–31 record the power for the respective *muActual* by appending to the arrays using *push!()*. Lines 35–49 plot the power curves showing the superiority of the T-test. Observe that at = 53 the power is identical to $\alpha = 0.05$.

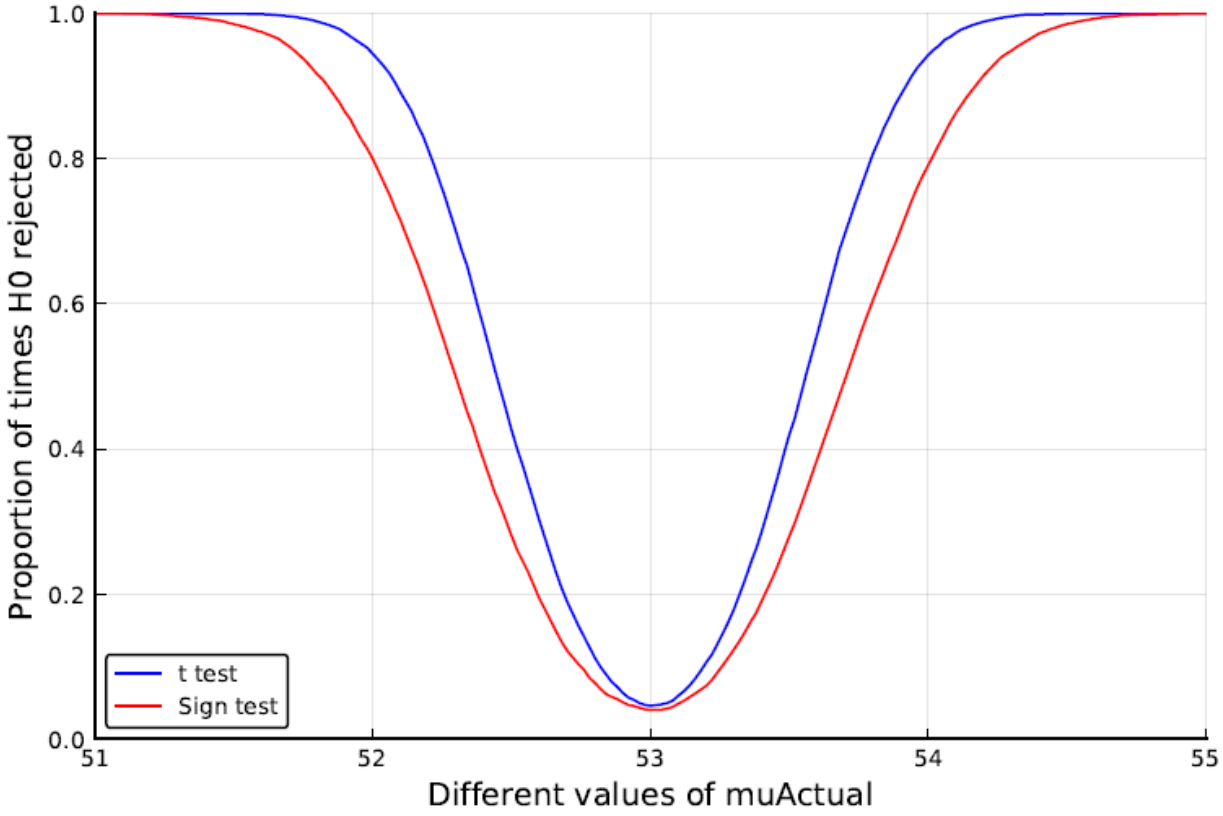

Figure 1: Power of the T-test vs. power of the sign-test.

Your task in this assignment is to write programs in Julia, Python, and R to plot Figure 1 and to study the possibility of the implementation of comparison between T-Test vs. power of the sign test using Octave and Scala. The Julia code is already provided. You are recommended to rely on the concept of this implementation on Julia to implement the comparison of *Power of the T-test vs. power of the sign-test* in Python and R.

Which programming language that provided relevant solution for this assignment? Justify your answer.

Which difficulties will you face if you want to solve this problem using Scala and Octave programming languages?

II. Useful links

1. For loop with Julia
2. Python For Loops

3. R for loop
4. For loop with Octave
5. Plotting with Julia
6. Line Plots - R Base Graphs
7. Python Matplotlib
8. 2D & 3D Plots with Octave
9. Online Scala Editor
10. Programming in Scala
11. Scala Tutorial

III. Submission

You have to submit a pdf file that contains four outputs that are related to Julia, Python, and R. In addition, you have to submit the sources files in Julia, Python, and R. ***Please, do not use .zip nor .rar file in your submission***.

Good luck :)