CS 504 – Programming Languages for Data Analysis

Assignment 3

Lin, Junjia 002268506

Nie, Luyun 002268087

1. Julia

In [1]:
```
using Random, Distributions, HypothesisTests, Plots; pyplot()
```

Out[1]:
```
Plots.PyPlotBackend()
```

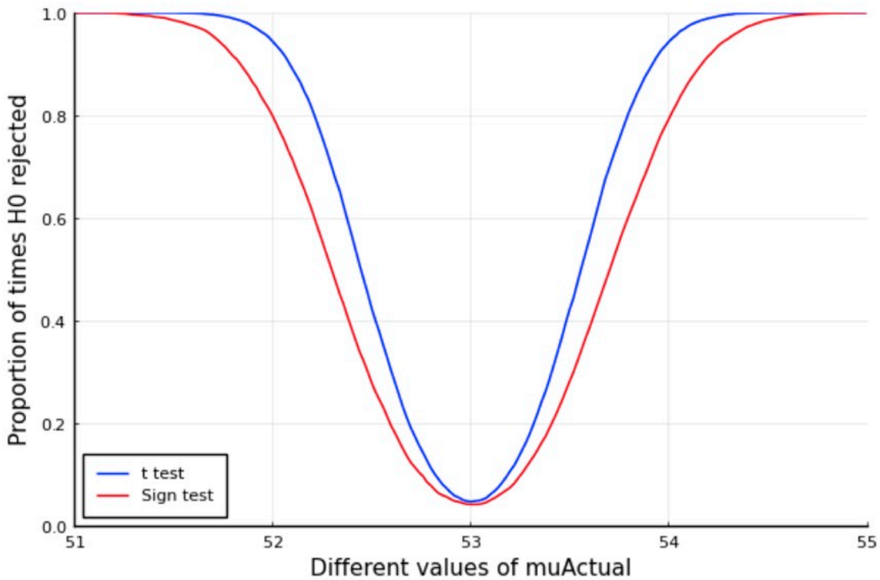
In [2]:
```
muRange = 51:0.02:55
n = 20
N = 10^4
mu0 = 53.0
powerT, powerU = [], []
```

Out[2]:
```
(Any[], Any[])
```

In [3]:
```
for muActual in muRange
    dist = Normal(muActual, 1.2)
    rejectT, rejectU = 0, 0
    Random.seed!(1)
    for _ in 1:N
        data = rand(dist,n)
        xBar, stdDev = mean(data), std(data)
        tStatT = (xBar - mu0)/(stdDev/sqrt(n))
        pValT = 2*ccdf(TDist(n-1), abs(tStatT))
        xPositive = sum(data .> mu0)
        uStat = max(xPositive, n-xPositive)
        pValSign = 2*pdf(Binomial(n,0.5), uStat)
        rejectT += pValT < 0.05
        rejectU += pValSign < 0.05
    end
    push!(powerT, rejectT/N)
    push!(powerU, rejectU/N)
end
```

In [4]:
```
plot(muRange, powerT, c=:blue, label="t test")
plot!(muRange, powerU, c=:red, label="Sign test",
xlims=(51,55), ylims=(0,1),
xlabel="Different values of muActual",
ylabel="Proportion of times H0 rejected", legend=:bottomleft)
```

Out[4]:



Proportion of times H0 rejected
Different values of muActual
t test
Sign test
0.0
0.2
0.4
0.6
0.8
1.0
51
52
53
54
55

In [ ]:

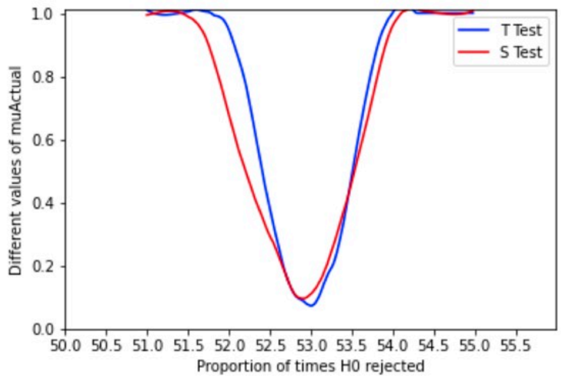
2. Python

In [1]:
```
import matplotlib.pyplot as plt
import scipy.stats as st
import numpy as np
from scipy.signal import savgol_filter
```

In [2]:
```
muRange=np.arange(51,55,0.02)
n = 20
N = 10^4
mu0 = 53.0
sigma = 1.2
powerT, powerU = [], []
```

In [3]:
```
for muActual in muRange:
    rejectT,rejectU = 0,0
    dist = np.random.normal(muActual,sigma,1000)
    np.random.seed(1)
    for it in range(N):
        data = np.random.choice(dist,n)
        xBar, stdDev = np.mean(data), np.std(data)
        tStatT = (xBar - mu0)/(stdDev/np.sqrt(n))
        pValT = st.t.sf(np.abs(tStatT), n-1)*2

        xPositive = sum( i > mu0 for i in data )
        uStat = max(xPositive, n-xPositive)
        pValSign = st.binom.pmf(uStat,n,0.5)*2
        if pValT < 0.05:
            rejectT  = rejectT + 1
        if pValSign <0.05:
            rejectU  = rejectU + 1
    powerT.append(rejectT/N)
    powerU.append(rejectU/N)
```

In [4]:
```
smoothT = savgol_filter(powerT,53,3)
smoothU = savgol_filter(powerU,53,3)
plt.plot(muRange,smoothT,color = 'blue',ls = "-")
plt.plot(muRange,smoothU,color = 'red',ls = "-")
plt.xlim(50.99,55.99)
plt.ylim(0,1.01)
plt.xticks(np.arange(50,56,0.5))
plt.xlabel('Proportion of times H0 rejected')
plt.ylabel('Different values of muActual')
plt.legend(["T Test","S Test"],loc='best', frameon=True)
plt.show()
```

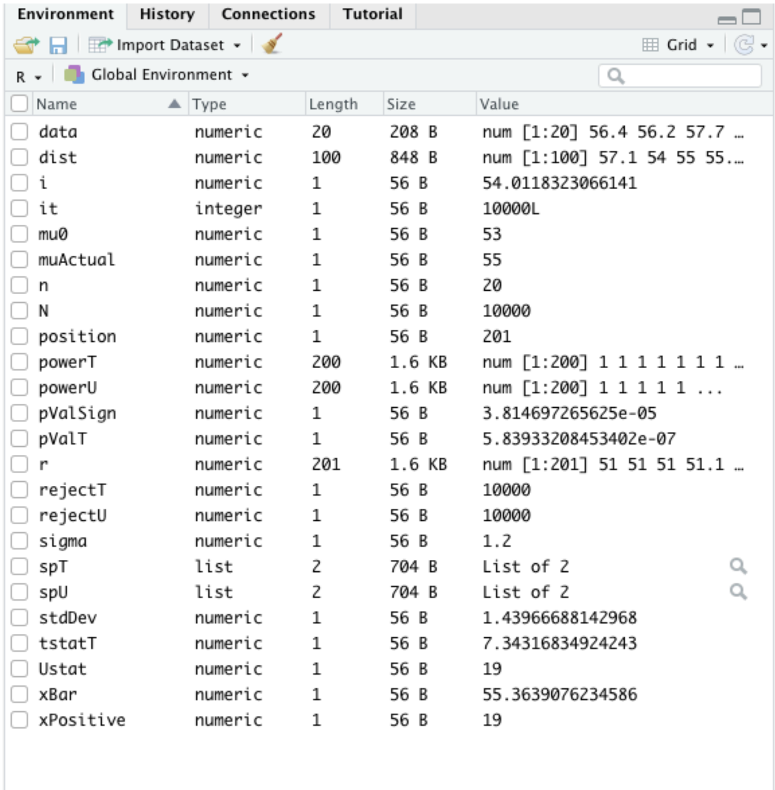

T Test
S Test
Different values of muActual
Proportion of times H0 rejected
1.0
0.8
0.6
0.4
0.2
0.0
50.0
50.5
51.0
51.5
52.0
52.5
53.0
53.5
54.0
54.5
55.0
55.5

In [ ]:

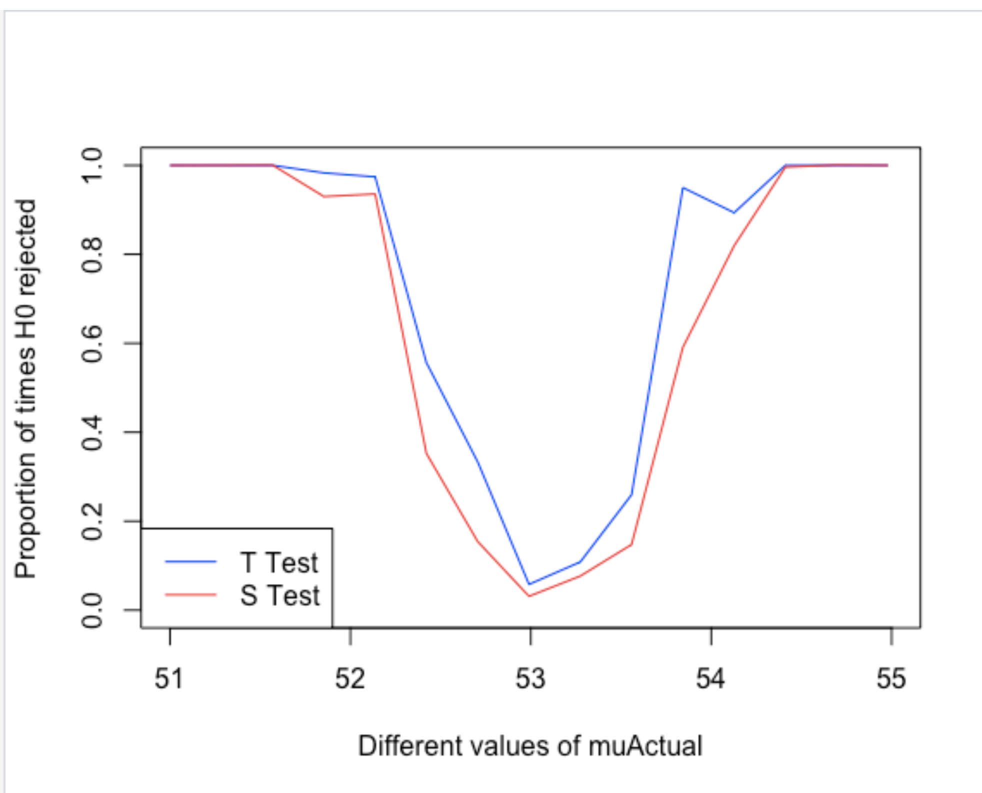
In [ ]:

3. R

```
r<-seq(51,55,0.02)
n = 20
N = 10^4
mu0 = 53.0
sigma = 1.2
powerT <- c()
powerU <- c()
position  = 0

for (muActual in r){
  dist = rnorm(100,muActual,sigma)
  rejectT = 0
  rejectU = 0
  for (it in seq(N)){
    data = sample(dist,20)
    xPositive = 0
    for (i in data){
      if (i > mu0){
        xPositive = xPositive +1}}
    Ustat = max(xPositive,n-xPositive)
    pValSign = dbinom(Ustat,n,0.5)*2

    xBar = mean(data)
    stdDev = sd(data)
    tstatT = (xBar-mu0)/(stdDev/sqrt(n))
    pValT = (1-pt(abs(tstatT),n-1))*2

    if (pValSign< 0.05){rejectU = rejectU+1}
    if (pValT<0.05){rejectT = rejectT+1}
  }
    powerT[position] <- rejectT/N
    powerU[position] <- rejectU/N
    position = position+1
}

spT=spline(seq(51,55-0.02,0.02),powerT,n=15)
spU=spline(seq(51,55-0.02,0.02),powerU,n=15)

plot(spT,type='l',col='blue',
     xlab="Different values of muActual",
     ylab="Proportion of times H0 rejected",
     xlim = c(51,55),ylim = c(0,1))
lines(spU,col = 'red')
legend("bottomleft",legend = c("T Test","S Test"),lty = c(1,1),
       col = c("blue","red"))
```



Name	Type	Length	Size	Value
data	numeric	20	208 B	num [1:20] 56.4 56.2 57.7 …
dist	numeric	100	848 B	num [1:100] 57.1 54 55 55.…
i	numeric	1	56 B	54.0118323066141
it	integer	1	56 B	10000L
mu0	numeric	1	56 B	53
muActual	numeric	1	56 B	55
n	numeric	1	56 B	20
N	numeric	1	56 B	10000
position	numeric	1	56 B	201
powerT	numeric	200	1.6 KB	num [1:200] 1 1 1 1 1 1 1 …
powerU	numeric	200	1.6 KB	num [1:200] 1 1 1 1 1 ...
pValSign	numeric	1	56 B	3.814697265625e-05
pValT	numeric	1	56 B	5.83933208453402e-07
r	numeric	201	1.6 KB	num [1:201] 51 51 51 51.1 …
rejectT	numeric	1	56 B	10000
rejectU	numeric	1	56 B	10000
sigma	numeric	1	56 B	1.2
spT	list	2	704 B	List of 2
spU	list	2	704 B	List of 2
stdDev	numeric	1	56 B	1.43966688142968
tstatT	numeric	1	56 B	7.34316834924243
Ustat	numeric	1	56 B	19
xBar	numeric	1	56 B	55.3639076234586
xPositive	numeric	1	56 B	19



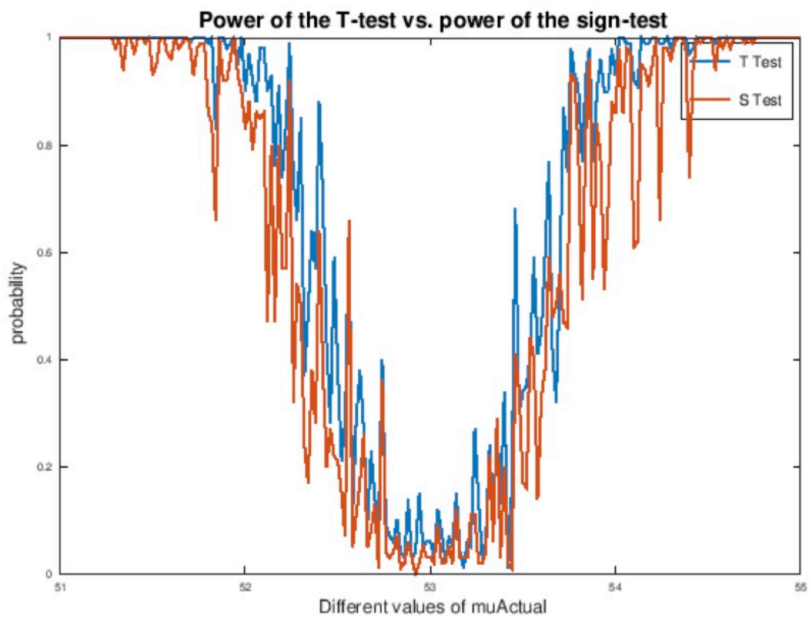
4. Octave

In [1]:
```
pkg load statistics
pkg load control
```

In [2]:
```
muRange = 51:0.02:55;
N = 100;
n = 20;
mu0 = 53;
sigma = 1.2;
powerT = size(1,length(muRange));
powerU = size(1,length(muRange));
index = 1;
```

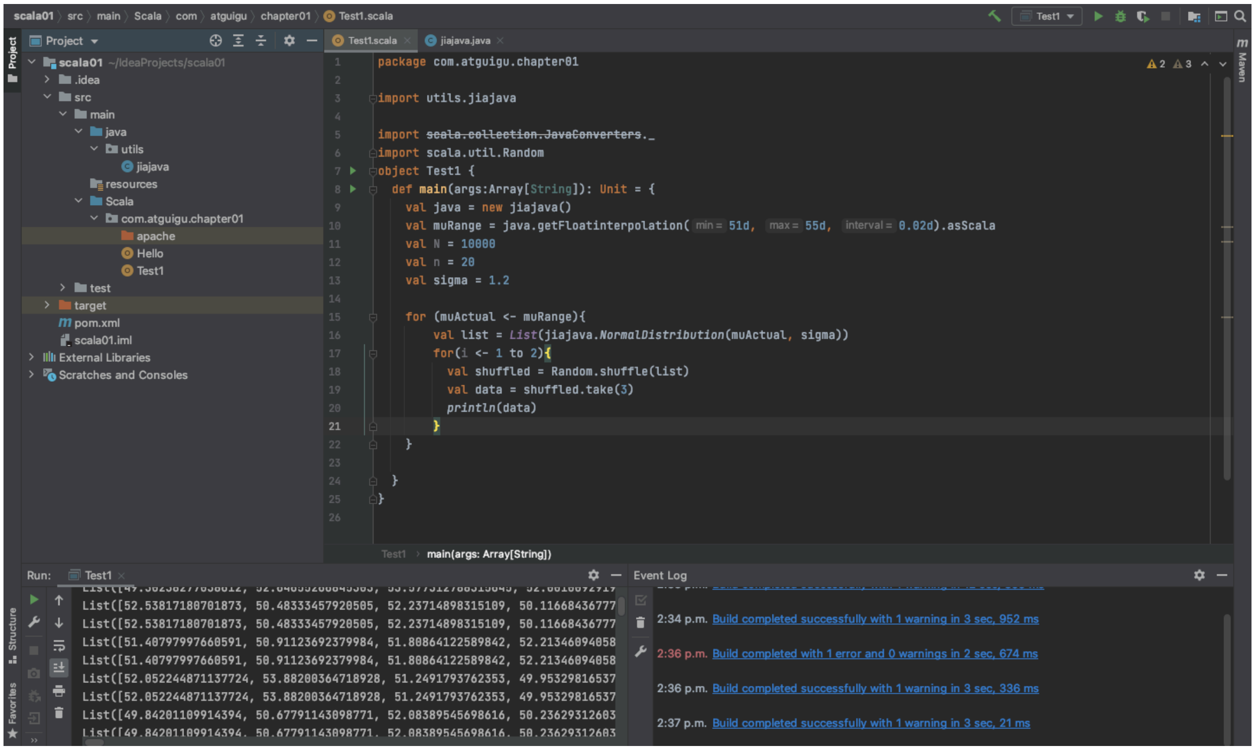
In [3]:
```
for muActual = muRange
    dist = normrnd(muActual,sigma,[1,100]);
    rejectT = 0;
    rejectU = 0;
    for i =1:N
        data = randsample(dist,n);
        xBar = mean(data);
        stdDev = std(data);
        tStatT = (xBar - mu0)/(stdDev/sqrt(n));
        pValT = (1-tcdf (abs(tStatT), n-1))*2;
        if pValT< 0.05
            rejectT = rejectT+1;
        endif

        xPositive = 0;
        for j = data
            if j > mu0
                xPositive = xPositive +1;
            endif
        endfor
        uStat = max(xPositive, n-xPositive);
        pValSign = 2*binopdf(uStat,n,0.5);
        if pValSign< 0.05
            rejectU = rejectU+1;
        endif

     endfor
     powerT(1,index) = rejectT/N;
     powerU(1,index) = rejectU/N;
     index = index+1;
endfor
```

In [5]:
```
graphics_toolkit ("gnuplot");
plot(muRange,powerT,'LineWidth',4,muRange,powerU,'LineWidth',4);
h = legend({'T Test','S Test'});
set (h, "fontsize", 14);
title('{\fontsize{30} Power of the T-test vs. power of the sign-test}')
axis([51 55 0 1])
xlabel('{\fontsize{25} Different values of muActual}');
ylabel('{\fontsize{25} probability}');
hold off;
```

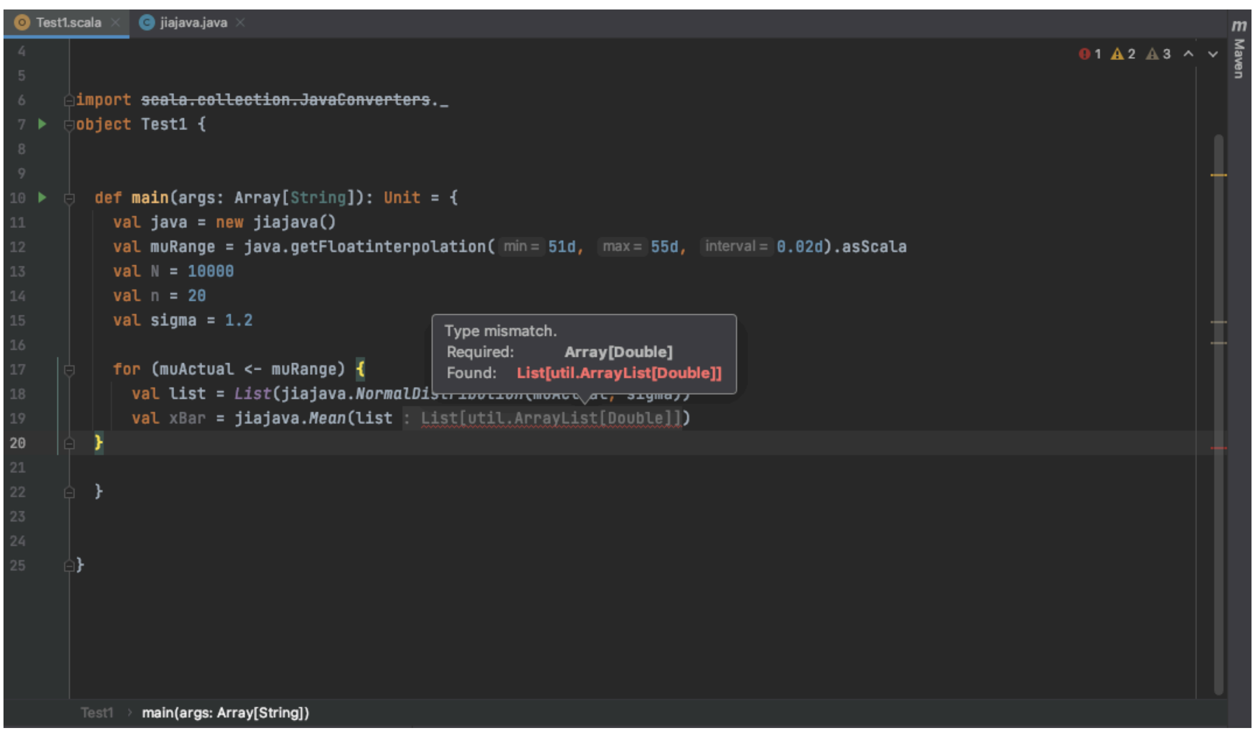


In [ ]:

5. Scala



scala01 〉 src 〉 main 〉 Scala 〉 com 〉 atguigu 〉 chapter01 〉 Test1.scala
Project
scala01 ~/IdeaProjects/scala01
.idea
src
main
java
utils
jiajava
resources
Scala
com.atguigu.chapter01
apache
Hello
Test1
test
target
pom.xml
scala01.iml
External Libraries
Scratches and Consoles
Test1.scala
jiajava.java
package com.atguigu.chapter01
import utils.jiajava
import scala.collection.JavaConverters._
import scala.util.Random
object Test1 {
def main(args:Array[String]): Unit = {
val java = new jiajava()
val muRange = java.getFloatinterpolation(min = 51d, max = 55d, interval = 0.02d).asScala
val N = 10000
val n = 20
val sigma = 1.2
for (muActual <- muRange){
val list = List(jiajava.NormalDistribution(muActual, sigma))
for(i <- 1 to 2){
val shuffled = Random.shuffle(list)
val data = shuffled.take(3)
println(data)
}
}
}
}
Test1 〉 main(args: Array[String])
Run: Test1
List([52.53817180701873, 50.48333457920505, 52.23714898315109, 50.11668436777
List([52.53817180701873, 50.48333457920505, 52.23714898315109, 50.11668436777
List([51.40797997660591, 50.91123692379984, 51.80864122589842, 52.21346094058
List([51.40797997660591, 50.91123692379984, 51.80864122589842, 52.21346094058
List([52.052244871137724, 53.88200364718928, 51.2491793762353, 49.95329816537
List([52.052244871137724, 53.88200364718928, 51.2491793762353, 49.95329816537
List([49.84201189914394, 50.67791143098771, 52.08389545698616, 50.23629312603
Event Log
2:34 p.m. Build completed successfully with 1 warning in 3 sec, 952 ms
2:36 p.m. Build completed with 1 error and 0 warnings in 2 sec, 674 ms
2:36 p.m. Build completed successfully with 1 warning in 3 sec, 336 ms
2:37 p.m. Build completed successfully with 1 warning in 3 sec, 21 ms



Test1.scala
jiajava.java
import scala.collection.JavaConverters._
object Test1 {
def main(args: Array[String]): Unit = {
val java = new jiajava()
val muRange = java.getFloatinterpolation(min = 51d, max = 55d, interval = 0.02d).asScala
val N = 10000
val n = 20
val sigma = 1.2
for (muActual <- muRange) {
val list = List(jiajava.NormalDistribution(muActual, sigma))
val xBar = jiajava.Mean(list : List[util.ArrayList[Double]])
}
}
}
Type mismatch.
Required: Array[Double]
Found: List[util.ArrayList[Double]]
Test1 〉 main(args: Array[String])

Test1.scala   jiajava.java

```java
public class jiajava {
    public ArrayList<Double> getFloatinterpolation(double min, double max, double interval) {
        ArrayList<Double> result = new ArrayList();
        for (int i = 0; i < Math.ceil((max - min) / interval); i++) {
            result.add(min + i * interval);
        }
        return result;
    }

    public static ArrayList<Double> NormalDistribution(double u, double v) {
        ArrayList<Double> result = new ArrayList();
        java.util.Random random = new java.util.Random();
        for (int i = 0; i < 100; i++) {
            result.add(Math.sqrt(v) * random.nextGaussian() + u);
        }
        return result;
    }
    public static double Sum(double[] data) {
        double sum = 0;
        for (int i = 0; i < data.length; i++)
            sum = sum + data[i];
        return sum;
    }
    public static double Mean(double[] data) {
        double mean = 0;
        mean = Sum(data) / data.length;
        return mean;
    }
    public static double POP_Variance(double[] data) {
        double variance = 0;
        for (int i = 0; i < data.length; i++) {
            variance = variance + (Math.pow((data[i] - Mean(data)), 2));
        }
        variance = variance / data.length;
        return variance;
    }
```

6. Answer Questions

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

Answer: Julia provided the most relevant solution for this assignment because of its fast speed and precise grammar. In terms of 4 succeed plotted results, curves in Julia were the most flattened and systematic which were perfect. In python, we tried to apply fixed random seed to reduce the fluctuation of data sample means and variances whereas got serrated reflected bell shape. Although at last, we found another function named wave-smooth to eliminate the serrated noises, the overall shapes were not ones we expected. By using R, as we mentioned in former assignment, the programming language executing commanding line by line, presented results for each item that we were able to observe the data changing. Also, we wanted to test our codes with random seeds in R by deleting the random seed line. Unsurprisingly, the curves were still dog-teethed, but R was warm-hearted to provide a spline function with modified parameters to help us beautify our plots. We will talk about our experience in Octave and Scala in the next question.

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

Answer: To be honest, these two languages were nightmares for us in accomplishing this assignment. Octave was very slow in executing non-matrix problems. In those screen shots, we only define the number of iteration ‘N’ as 100, where still cost us more than 1 minute to run the whole codes. If we changed the N into 10 thousand, the running time would be more than 10 minutes with timer. Without any doubts, the situation may be largely affected by the connection between the Octave and the Jupyter Notebook, while Julia behaved much better under the same circumstances.

Secondly, Scala was based on Java but not as same as Java. In a very short time to learn this Object-Oriented Language, we got really confused about the extremely strict transformation of data types, exampled in the Scala screen-shot figure 2. Some data types such as ArrayList or Double stored differently from Java, they set up a lot of difficulties in changing data types and blocking us from retrieving classes or functions. Moreover, Scala was quite different from Julia, Python and R that many useful functions such as Mean, Sum were not existed in libraries and we had to write our own functions. Sometimes, some of the Scala functions performed imperfectly like the first screen shot, in which we also need to develop our own classes of functions. In order to not give up programming in Scala, we did some research about this language. It was called the next generation of Java and chose by the Spark. If we intended to undertake a job in the Big Data industry, Scala was the key. We'd like to have more time to learn more details about this language.