Practical 1 - Julia

Hand in: 4 March 2022

Michael Altshuler
ALTMIC003

Taine De Buys
DBYTAI001

I. Introduction

This practical investigates the execution speed of 2 functions createwhiten() and corr() against 2 built in functions of the julia programming language.As stated above, white noise will be generated and written to a WAV file by the rand() function, and the same will occur for the white noise generator script. The functionality of the white noise generator script and the rand() function will then be compared based on their execution time and Pearson's Correlation between the data generated by the two functions. Not only will the functions be compared in their ability to create white noise, but a version of the Pearson's Correlation function will be made and comapared to the functionality of the Statistics.cor function offered by Julia. Lastly, we will be using the the Statistics.cor function to theorise the correlation of time-shifted sinusoids.

II. Method

A. Software

This assignment was completed using the Julia programming language , an open source, high level language, well known for it's useful statistical operations. The focus of this practical is to perform various statistical operations on data generated by the Julia language - with the use of the rand() function to generate white noise, and a white noise generator script created in the assignment. Hence, Julia is the perfect language for this task.

B. Implementation

There were various parts of this assignment that needed to be completed in order to grasp an understanding of how white noise is generated, how statistical operations can be applied to these data sets generated, and lastly how these statistical operations work themselves. The tasks needed to be completed are described above in the introduction section of this report. Here, the ways in which these white noise data sets were generated and the ways in which they were meaningfully compared using correlation functions will be discussed.

Identify applicable funding agency here. If none, delete this.

As stated above, the main task of this assignment is to compare the white noise generated by the rand() function and the white noise generated by a written script using the function createwhiten(). Thus, the ways in which these two different white noise data sets were generated will be discussed below.

First, the rand() function will be looked at. The rand() function generates uniformly distributed random values in the interval [0,1). In order to generate white noise using this function, a sound wave needs to be created using the wavwrite() function. The wavwrite() function expects values in [-1.0, 1.0), so the rand() output must be multiplied by 2 and shifted down by 1. The following generated 10 seconds of white noise sampled at 48 kHz. The code below describes this process:

1.2.1 Measuring Execution Time of rand()

```
using WAV

whiteNoise = (rand(48000)*2).-1
#sample freq is 4800Hz
WAV.wavwrite(whiteNoise, "whiteNoise.wav",
    Fs=4800)
```

Secondly, in order to compare the white noise generated by this rand() function with the white noise generated by the createwhiten() function, a script creating this function has to be made. createwhiten() is a function with a for loop that generates a white noise signal, one sample at a time, comprising N duration in seconds. The white noise is also sampled at 48kHz so that the white noise generated by the rand() and the white noise generated by createwhiten() have the same sample size. The code below shows how createwhiten() is created:

1.2.2 White Noise Generator Script

```
#white noise generator script:

function createwhiten(n)
   whiten = Vector{Float64}();
   for i in 1:n*48000
      append!(whiten, (rand()*2. - 1));
   end
   return whiten
```

```
end

noise = createwhiten(1);
WAV.wavwrite(noise, "white_noise_sound2.wav",
    Fs=4800);
print("Number of samples: ", length(noise))
```

Once the createwhiten() function was created, we need a way to confirm that WAV file created by it has a uniform distribution. This is done using the Plots.histogram() function in the Plots package. The code and the output of the output of the Plots.histogram() function are shown below:

1.2.3 Visual Confirmation of Uniform Distribution

```
using Plots
h = Plots.histogram(noise)
Plots.display(h)
readline() #this will stop the program at
    this point till you press enter
```

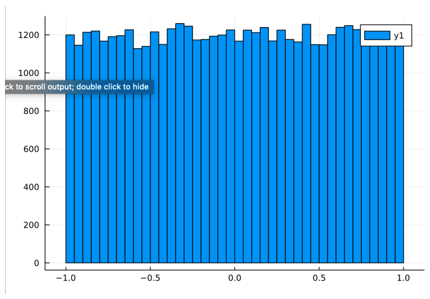


Fig. 1. Figure 1: Output of Plots.histogram(whiteNoiseArray)

Now that there are two ways of generating white noise using the rand() and createwhiten() functions, these functions can be compared by timing how long each one takes to run and generate white noise for varying sample sizes. In order to control the experiment, white noise was generated for 10, 20 and 30 seconds for both the rand() and createwhiten() functions. The TickTock Package was used to perform these tests. The tests are run multiple times to ensure that the cache is loaded. The code performing these tests and its results are shown below:

1.2.4 Timing Execution

```
using TickTock

# comparing the performance of the two
    approaches to generate white noise.

# Measuring excecution time of rand()
    generating 10 seconds of white noise:
tick()
whiteNoise = (rand(48000)*2).-1
#WAV.wavwrite(whiteNoise,
    "whiteNoise_test1_rand().wav", Fs=4800)
    #sample freq is 4800Hz
#tok()
#x = tok()
print("Time taken for rand() function to
    execute: ", tok(), "\n")

# Measuring excecution time of rand()
    generating 20 seconds of white noise:
tick()
whiteNoise = (rand(96000)*2).-1
#WAV.wavwrite(whiteNoise,
    "whiteNoise_test1_rand().wav", Fs=4800)
    #sample freq is 4800Hz
#tok()
#x = tok()
print("Time taken for rand() function to
    execute: ", tok(), "\n")

# Measuring excecution time of rand()
    generating 30 seconds of white noise:
tick()
whiteNoise = (rand(180000)*2).-1
#WAV.wavwrite(whiteNoise,
    "whiteNoise_test1_rand().wav", Fs=4800)
    #sample freq is 4800Hz
#tok()
#x = tok()
print("Time taken for rand() function to
    execute: ", tok(), "\n")

# Measuring the execution time of
    createwhiten(n) for n = 10;
tick()
noise = createwhiten(10);
#WAV.wavwrite(noise,
    "white_noise_sound2_test2_createwhiten.wav",
    Fs=4800);
#tok()
#y = tok()
print("Time taken for createwhiten(10)
    function to execute: ", tok(), "\n")

# Measuring the execution time of
    createwhiten(n) for n = 20;
tick()
noise = createwhiten(20);
#WAV.wavwrite(noise,
    "white_noise_sound2_test3_createwhiten.wav",
    Fs=4800);
print("Time taken for createwhiten(20)
    function to execute: ", tok(), "\n")

# Measuring the execution time of
    createwhiten(n) for n = 30;
tick()
noise = createwhiten(30);
#WAV.wavwrite(noise,"white_noise_sound2_test4_createwhit
    ", Fs=4800);
print("Time taken for createwhiten(30)
    function to execute: ", tok(), "\n")
```

The next part of the task entails comparing the data from the rand() function and the createwhiten() function respectively by calculating the correlation between them. Julia does have its own Pearson's correlation function with the use of Statistics.cor, however part of the task is to implement our own Pearson's correlation function. This function is called carr() and the code for it is displayed below:

1.2.5 Implementing Pearson's Correlation

```
# Implement the Pearsons correlation formula
    a new function call it corr().

function corr(x = Vector{Float64}(), y =
    Vector{Float}())

   #Setting up summation of xy:
   xyVal = 0
   for i in 1:length(x)
      #xyVal = 0;
      for z in 1:length(y)
         xyVal = xyVal + x[z]*y[z]
      end
      #return xyVal
   end

   #Setting up summation of x values:
   for n in 1:length(x)
      xVal = 0;
      for j in 1:length(x)
         xVal = xVal + x[j]
      end
      #return xVal
   end

   #Setting up the summation of y values:
   for c in 1:length(y)
      yVal = 0;
      for p in 1:length(y)
         yVal = yVal + y[p]
      end
      #return yVal
   end

   #Setting up summation of x^2
   x2Val = 0
   for q in 1:length(x)
      #x2Val = 0;
      for g in 1:length(x)
         x2Val = x2Val + x[g]*x[g]
      end
      #return x2Val
   end

   #Setting up summation of y^2
   y2Val = 0
   for h in 1:length(y)
      #y2Val = 0;
      for w in 1:length(y)
         y2Val = y2Val + y[w]*y[w]
      end
      #return y2Val
   end

   r = xyVal/(sqrt(x2Val)*sqrt(y2Val));
   return r

end
```

1.2.6 Comparing Your Correlation Function to the Statistics Package's Correlation Function

In this part of the assignment the results from the corr() function created are compared to that of the results of the Statistics.cor function built within Julia.

This was done by using the output from the create-whiten() function and comparing its correlation against itself using the correlation function and the Statistics.cor function. Once this has been done the correlation between the two different approaches to generate white noise, using the correlation function and the Statistics.cor function are compared.

1.2.7 Correlation of Shifted Signals

In the last part of the experiment, sinusoidal signals that have been shifted in time are compared to one another. This is done by generating sin curves of 3 different frequencies of 2, 20 and 50 Hz. An array was made and populated with the correlation coefficients of a regular sine wave against one that was shifted to the left 0from 0 to 1 second, the results were then plotted.

```
using Base
using MTH229
using Plots
using SymPy

#defining variables used
#time
t=0:0.01:1 #100 samples
u=0:0.005:1 #200 samples used to plot and
#angular frequency
freq1=20*pi
freq2=50*2pi
freq3=2*pi
#empty array for correlation coefficients
k=zeros(0)

#plot of graph and shifted graphs from above
f(t)= sin.(freq1*(t))
g(t)=sin.(freq1*(t).-pi/2)
i(t)=sin.(freq1*(t).-pi)
plot(t,f(t),
title="Graph displaying normal and shifted
    plots",
label=true,
lw=2,
xlabel= "t",
ylabel= "Ampltude",
)
plot!(t,g(t))
plot!(t,i(t))
```

```
#Correlation coefficient vs shift, f = 2 Hz
h=zeros(0)
for i in 0:0.005:1
   f(t)=sin.(freq3*(t))
   #shifted function
   g(t)=sin.(freq3*(t.-i))
   push!(h,cor(g(t),f(t)))
```

```
end
plot(u,h ,
title="Corelation Coefficient vs shift, f= 2
    Hz",
label=false,
lw=2,
xlabel= "Shift in time",
ylabel= "Correlation Coefficient")
```

```
#Correlation coefficient vs shift, f = 20 Hz
for i in 0:0.005:1
   f(t)=sin.(freq1*(t))
   #shifted function
   g(t)=sin.(freq1*(t.-i))
   push!(k,cor(g(t),f(t)))
end

plot(u,k,
title="Corelation Coefficient vs shift, f= 20
    Hz",
label=false,
lw=2,
xlabel= "Shift",
ylabel= "Correlation Coefficient")
```

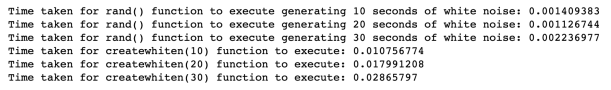
```
#Correlation coefficient vs shift, f = 50 Hz
l=zeros(0)
t=0:0.001:1 #1000 samples
u=0:0.001:1 #200 samples used to plot and
for i in 0:0.001:1
   f(t)=sin.(freq2*(t))
   #shifted function
   g(t)=sin.(freq2*(t.-i))
   push!(l,cor(g(t),f(t)))
end
plot(u,l ,
title="Corelation Coefficient vs shift, f= 50
    Hz",
label=false,
lw=2,
xlabel= "Shift in time",
ylabel= "Correlation Coefficient")
```

III. Discussion and Results

A. Createwhiten

This function generates white noise and takes in one argument that specifies the duration in seconds of the white noise created. In order to test the efficiency of createwhiten() to generate white noise, run time tests were run on this function for various durations and compared to the run time of the rand() function to generate white noise for the same inputted durations. Durations of 10, 20 and 30 seconds were chosen to run these tests. The results yielded are depicted below:

As it can be seen above, the rand() function produced run time results that are better for every inputted duration. There is a one order of magnitude difference in the run time



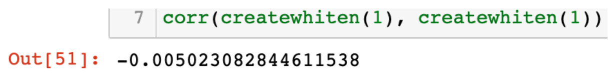
```
Time taken for rand() function to execute generating 10 seconds of white noise: 0.001409383
Time taken for rand() function to execute generating 20 seconds of white noise: 0.001126744
Time taken for rand() function to execute generating 30 seconds of white noise: 0.002236977
Time taken for createwhiten(10) function to execute: 0.010756774
Time taken for createwhiten(20) function to execute: 0.017991208
Time taken for createwhiten(30) function to execute: 0.02865797
```

Fig. 2. Results for comparing the run time of the rand() and createwhiten() functions for varying times of white noise generated

of the rand() function compared to that of the createwhiten() function. This can be attributed to the fact that the createwhiten() function makes use of a for loop, meaning that the number of iterations needing to be run as a result of the duration input slows down the run time of the function as a whole, where in the case of the rand() function, cleaner and more efficient methods of generating the white noise may have been used when writing the code for it. Furthermore, the rand() function and the createwhiten() function were inputted into the Statistics.cor function, yielding a result of -0.006128. It was expected that the correlation coefficient would not be zero, but the fact that it is this close to zero is promising. This is confirmation that the createwhiten() function generates white noise very similarly to the way in which the rand() function does, however due to the run time tests above it cannot be concluded that the createwhiten() function works more efficiently/better than the rand() function.

B. Correlation

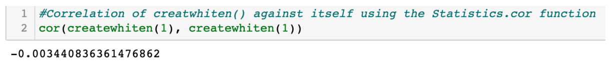
In order to test the accuracy of the correlation function created (corr()), its results were compared to the results of the built in Statistics.cor function offered by the Julia language. Below are the results showing the correlation coefficients when inputting createwhiten(1) and comparing it against itself for bothe the corr() function and the Statistics.cor function:



```
corr(createwhiten(1), createwhiten(1))
```

```
Out[51]: -0.005023082844611538
```

Fig. 3. Results of corr() function with inputs of createwhiten(1)



```
#Correlation of creatwhiten() against itself using the Statistics.cor function
cor(createwhiten(1), createwhiten(1))
```

```
-0.003440836361476862
```

Fig. 4. Results of Statistics.cor function with inputs of createwhiten(1)

As it can be seen, the corr() function produces a result of -0.00502 and the Statistics.cor function produces a result of -0.00344. To ensure accuracy of results, the tests were run multiple times, loading the cache. It was expected that the results would be different due to the fact that createwhiten() produced random results each time. However, the fact that the results only differ by 0.002, it can be concluded that the corr() function operates correctly. As for it being a better correlation function than the Statistics.cor function, one cannot conclude this. Although the results were very close, the Statistics.cor function produced results closer to 0 than the corr() function did for every test. Thus, it can be said that the corr() function works, but it cannot be said that it works in a manner that is better than the Statistics.cor function.

C. Shifted signals

As expected, the shift in time of a sinusoidal waveform resulted in the graph moving left along the x axis when both frequency and sample size were kept constant.

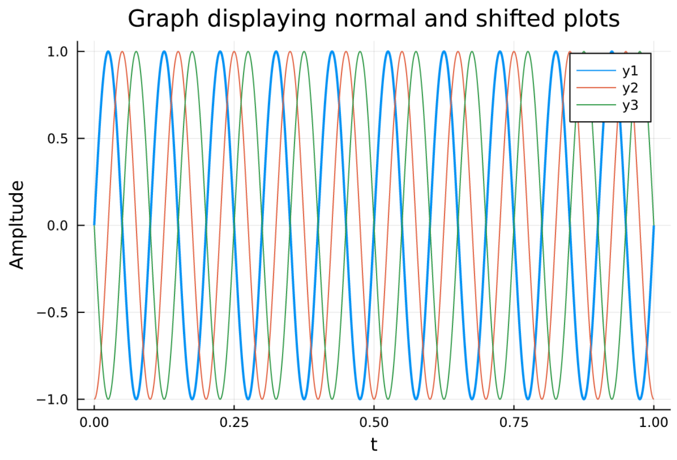


Fig. 5. Graph displaying normal and shifted plots.

Upon plotting the correlation coefficient between the original sinusoidal and 200 samples of graphs shifted to the left in the time domain from 0-1 second, the resulting outputs were as expected. As the frequency increased, the time shift had a greater effect on the graphs correlations, resulting in more oscillations from a perfect correlation (1) to and perfect negative correlation (-1) for the given time frame.These are shown below.

It was also noted that as frequency increased, the sample rate also needed to increase in order to ensure a high enough resolution to extract meaningful information from the plots. A change in sample size did not effect the correlation in any way as all sample rates for the given frequencies were above the Nyquist rate, sample rate merely changed the resolution of the graphs plotted.

References

[1] Mathematics · The Julia Language (2022). Available at: https://docs.julialang.org/en/v1/base/math/ (Accessed: 3 March 2022).
[2] Julia Manual - Function List and Reference Julia Functions (2022). Available at: http://www.jlhub.com/julia/manual/en/ (Accessed: 4 March 2022).

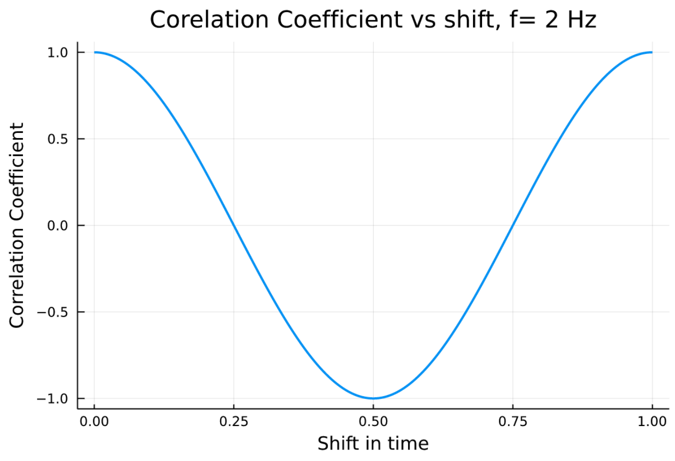


Fig. 6. Graph displaying Correlation Coefficient vs shift, f= 2 Hz.

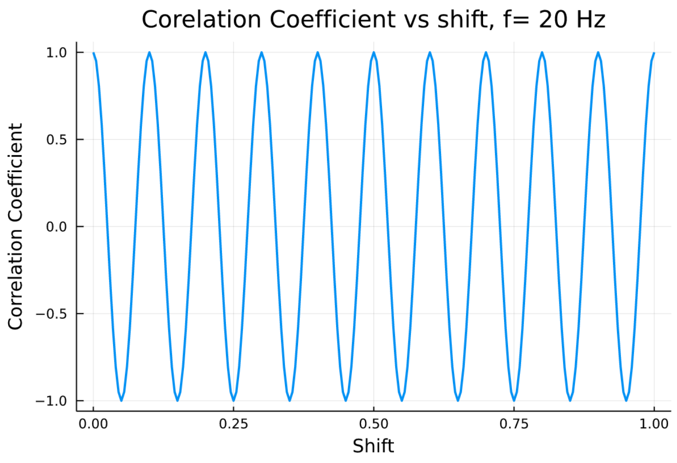


Fig. 7. Graph displaying Correlation Coefficient vs shift, f= 20 Hz.

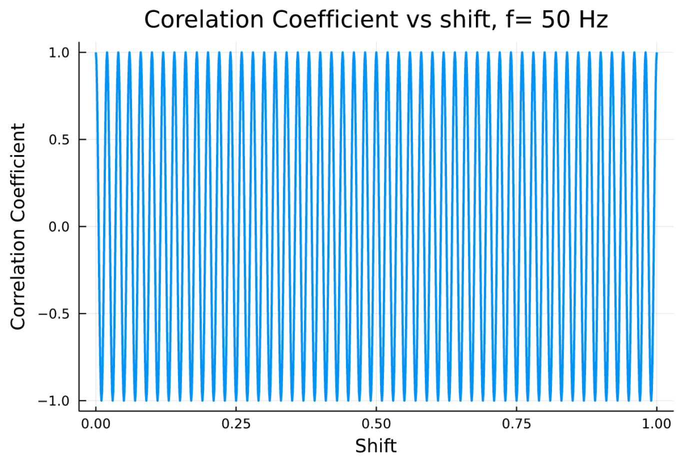


Fig. 8. Graph displaying Correlation Coefficient vs shift, f= 50 Hz.