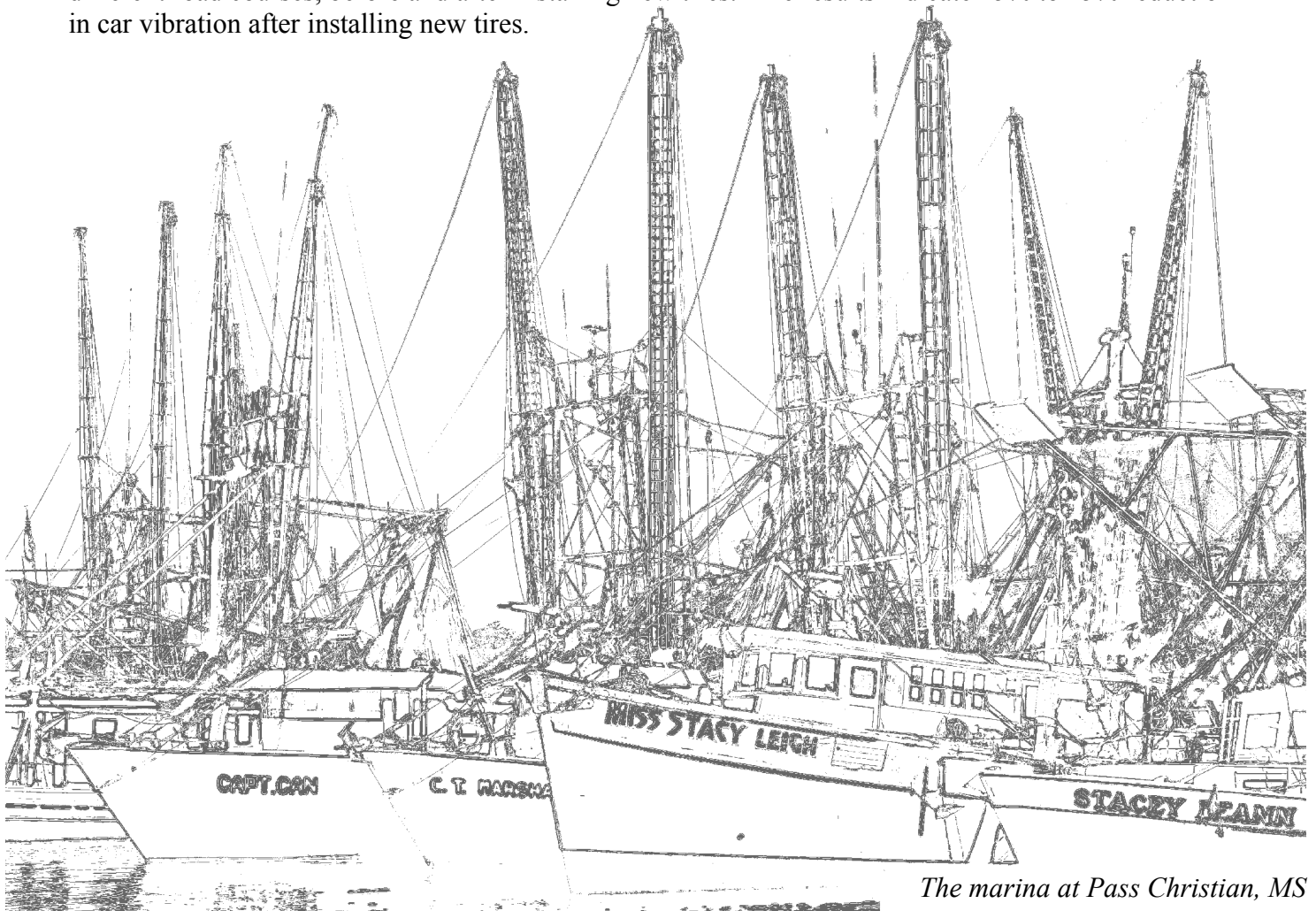


Vibration Analysis of New Tires

Test Report: TR002

Revision: New
Date: July 12, 2016

Abstract: As a car tire ages, the vulcanized rubber degrades and the tire becomes less pliable, surface cracks appear, and pieces of rubber can dislodge. Furthermore, it's common to have balancing weights shift on the rim or break off. These factors combine to cause imbalances, which makes the car vibrate, noisy, and uncomfortable to drive. This test report evaluates and compares car vibration on two different road courses, before and after installing new tires. The results indicate 15% to 25% reduction in car vibration after installing new tires.



The marina at Pass Christian, MS

Table of Contents

1	Introduction.....	1
1.1	Document Conventions	1
1.2	Repeating this Experiment.....	1
1.3	Disclaimers.....	1
2	Background Information.....	2
3	Objective.....	2
4	Experiment Setup.....	2
4.1	Materials and Equipment.....	2
4.2	Equipment Setup.....	3
4.3	Software.....	4
5	Procedure.....	5
5.1	Test Procedure	5
5.2	Analysis Procedure	6
6	Results.....	7
7	Discussion.....	8
8	Conclusion.....	9
9	Appendix A: R Script – car_vibration.r.....	10

List of Figures

Figure 1:	Test Car.....	3
Figure 2:	Logger Configuration.....	3
Figure 3:	X2-2 Mounting Location.....	4
Figure 4:	Data File Selection.....	6
Figure 5:	Example R Script Input and Results.....	7
Figure 6:	Performance Comparison at 65 mph.....	7
Figure 7:	Performance Comparison at 70 mph.....	8

List of Tables

Table 1:	Materials and Equipment List.....	2
Table 2:	Test Conditions and Notes.....	5
Table 3:	Total Z-axis Vibration Comparison.....	8

1 Introduction

1.1 Document Conventions

This test report describes in detail the background information, procedure, and analysis method used to conduct an experiment using a GCDC data logger. This experiment is an example application using an accelerometer that is both educational and fun.

Each section also presents relevant tips and warnings to help the user repeat the experiment or make improvements.



This icon indicates a helpful tip that may enhance the results or add a new perspective to the objective.



This icon indicates a warning, restriction, or limitation that the user should be aware of regarding the experiment or logger operation.

1.2 Repeating this Experiment

Obviously, a GCDC data logger is required to repeat the test procedure. The exact product type is described in section 4. However, the raw data and analysis methods are provided such that a user may recreate the same results without repeating the test process. Stepping through the analysis process with the data files will help the user understand the steps and see the expected results. The raw data files, spreadsheets, and R scripts are available for download at www.gcdadataconcepts.com.

A spreadsheet is used for simple analysis so the user should be familiar with manipulating data and creating plots in a spreadsheet. We recommend Microsoft Excel or OpenOffice Calc.

“R” is used for more complex data analysis. R is a simple command line programming environment that can manipulate large data sets using common math commands as well as complex function libraries. The software is compact, free, and available at www.r-project.org for Windows, Mac, and Linux. The R scripts provided herein are designed to run on Windows and may not be fully compatible with Mac and Linux systems. Specifically, file path references will not translate properly to the Mac and Linux systems. These differences will be noted in the script comments.

1.3 Disclaimers

This test report is presented “as-is” and for informational purposes only. No claims, representations or warranties, whether expressed or implied, are made to the safety and performance of the procedures described herein. Furthermore, we do not make any guarantee that your son or daughter will get an “A” on their science project, which is probably due tomorrow morning...but that's not our problem and you should have started earlier anyway.

2 Background Information

During a recent rain storm, driving on the interstate in a slightly less than straight line, and with all manner of dash lights blinking warnings at me, I realized that I was hydroplaning and it was time to get new tires for my car. Hydroplaning isn't a fun experience and I recommend strongly against it. As I drifted further sideways into the onslaught of rain, I wondered if new tires would also smooth the ride and bring luxury comfort to my sporty Mazda3. The traction control kicked in, straightened my car, and I concluded that testing tire vibration performance is an interesting application of the X2-2 logger. New tires will definitely fix the hydroplaning but will new tires make a smoother and more comfortable ride?

Many tires today can last up to 60,000 miles but tire manufacturers recommend an age limit of about 5 years. As a tire ages, the vulcanized rubber degrades and the tire becomes less pliable, surface cracks appear, and pieces of rubber can dislodge. Furthermore, it's common to have balancing weights shift on the rim or break off. These factors combine to cause imbalances, which makes the car vibrate, noisy, and uncomfortable to drive.

3 Objective

The test analyzed data collected from an X2-2 accelerometer data logger to evaluate and compare car vibration on two different road courses, before and after installing new tires.



This test report compares the vibration between two tire conditions (new versus old) on different road surfaces. However, the procedure establishes a general method for evaluating vibration. The procedure can be simplified to compare the vibration of different road conditions, such as asphalt versus concrete, or any similar type vibration scenario.

4 Experiment Setup

4.1 Materials and Equipment

Table 1: Materials and Equipment List

Description	Quantity	Comments
2012 Mazda3 or equivalent	1	Fitted with old tires
New tires for car	4	They smell bad and they are expensive but every car needs new tires eventually
X2-2 Accelerometer Data Logger	1	Purchase online from GCDC. The X16 series logger is also a viable option.
Notepad and pen	1	High tech people will use a smartphone app
Watch or clock to monitor time of day	1	Car dashboard usually has a clock
3M Command Adhesive Strips (med)	1	Available at most retail stores in the hardware department. Medium size strip works well.



Figure 1: Test Car

4.2 Equipment Setup

4.2.1 Logger Configuration

Figure 2 shows the configuration settings of the X2-2 logger. The logger was charged on a USB port for about an hour to ensure the battery was ready prior to starting the test procedure.

The file size was set to a 2 minute length to help group the test conditions. The analysis process averaged a two minute segment of data.

```
;X2-2 configuration
;set sample rate
;available rates 4, 8, 16, 32, 64, 128, 256, 512
samplerate = 256
;record constantly
deadband = 0
deadbandtimeout = 0
;set file size to 2 minutes of data
samplesperfile = 30720
;set status indicator brightness
statusindicators = Normal
stoponvusb
;rebootOnDisconnect
microres
gain = low
;see user manual for other config options
```

Figure 2: Logger Configuration



Set the RTC so the data files will have the correct date and time. This is important to correlate the data to notes taken during the test. See the X2-2 user manual regarding the RTC initialization procedure.

A convenient location on the driver's seat was chosen for easy placement of the X2-2 logger. Specifically, the bracket that mounts the seat to the car body as seen in Figure 3. The +X axis pointed towards the front of the car and the +Z axis pointed downward. This experiment did not consider the X and Y axes.



Figure 3: X2-2 Mounting Location

4.2.2 Road Course

The analysis procedure evaluated 2 minutes of data. Therefore, road locations were chosen that allowed a consistent speed for at least 6 minutes. This ensured one complete data file covered a consistent speed and road condition. The road tests spanned several days but care was taken to select days with similar weather conditions.

Table 2 lists the location, speed, surface conditions, and time when each course was driven.



Temperature, wind, and wet road conditions will have some influence to the results. Minimizing these variables will make a better comparison between test conditions.

4.3 Software

An R script was developed to analyze the test data (see Appendix A). The script converted the accelerometer data to the frequency domain using a Fast Fourier Transform (FFT) function. The FFT function analyzed 1024 samples, or 4 seconds of data, and created a spectral response between 0 Hz and 128 Hz. Consecutive segments of data were taken from the file until the entire 2 minute file was processed. All the FFT results were averaged together into a single spectral plot summarizing the entire 2 minutes. This method averaged out intermittent changes in driving behavior and increased the signal-to-noise ratio of the results.

The script used the average spectral response to calculate the total RMS acceleration $G_{rms}/\sqrt{\text{Hz}}$ between 0.5 Hz and 128 Hz. Establishing the lower bandwidth at 0.5Hz removed the affects of gravity.

The script saved the output data as a text file, which was later imported into a spreadsheet to access better plotting capabilities.



The script and the raw data files used in this report are available to download from the GCDC website. Copy the script file to the root directory of the X2-2 logger to allow easy access from the R console. Copy the data files to a temporary directory on the data logger. The script will prompt the user for the drive and directory location of the data.

5 Procedure

5.1 Test Procedure

The following steps outline the procedures taken to collect vibration data on two different road conditions before and after the tires were changed. The test notes are summarized in Table 2.

1. Started the X2-2 data logger and drove to the first road course segment.
2. Noted the date and time when the first road course started.
3. Drove for at least 6 minutes on the first road course using the cruise control to maintain a consistent speed.
4. Noted when the course completed.
5. Continued to the second road course and noted the start time. Again, a consistent speed and driving habit was maintained.
6. Noted when the course was completed.
7. Parked the car and stopped the logger.
8. Bought new tires at a reputable car service center. Sat in the waiting room for hours. Watched other customers receive service, even though they arrived an hour later. Got frustrated. Gave the manager the “evil eye”. Rejoiced when the car was finally done. Cried after paying the bill.
9. Repeated the two test courses according to steps 1 through 7.



Don't try to drive and take notes. Designate a passenger to help log time and road conditions so the driver can focus on driving safely.

Table 2: Test Conditions and Notes

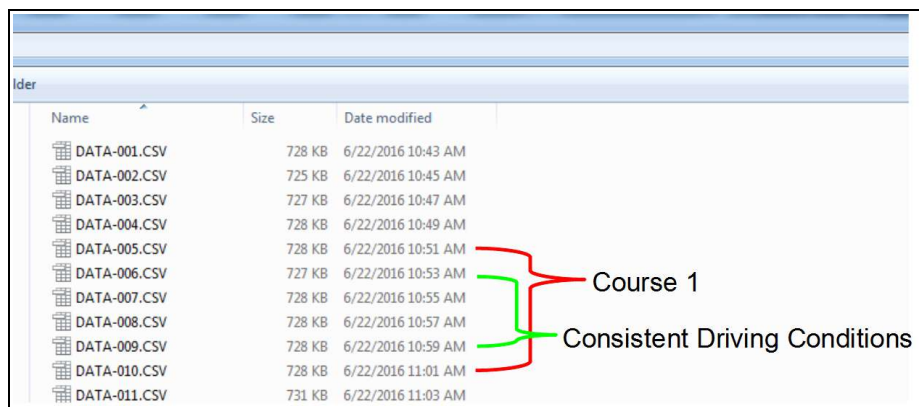
Location	Speed	Surface	Test Number	Tires	Start Time	Finish Time
Hwy 607 West	65 mph	10 yr old asphalt	1	Old	6/22 10:50 am	6/22 11:00 am
			3	New	6/23 3:20 am	6/23 3:11 am
I-10 East	70 mph	2 yr old smooth asphalt	2	Old	6/22 11:10 am	6/22 11:18 am
			4	New	6/23 3:35 am	6/23 3:40 am

5.2 Analysis Procedure

5.2.1 Data Selection

The tests created dozens of files but only four files were needed for the analysis. The files were chosen such that the data represented consistent driving conditions on the specific course. Each data file has a “last modified” date that indicates when the file closed. Therefore, the two-minute data segment covered the time before the “last modified” date. The start/finish times recorded in the test notes determined the appropriate data files for each of the four test condition.

Figure 4 shows the files created driving the first course. Files 5-10 span the course time but files 6-9 represent the most consistent driving conditions. Any of the files between 6 and 9 are acceptable but the analysis used DATA-007.



Name	Size	Date modified
DATA-001.CSV	728 KB	6/22/2016 10:43 AM
DATA-002.CSV	725 KB	6/22/2016 10:45 AM
DATA-003.CSV	727 KB	6/22/2016 10:47 AM
DATA-004.CSV	728 KB	6/22/2016 10:49 AM
DATA-005.CSV	728 KB	6/22/2016 10:51 AM
DATA-006.CSV	727 KB	6/22/2016 10:53 AM
DATA-007.CSV	728 KB	6/22/2016 10:55 AM
DATA-008.CSV	728 KB	6/22/2016 10:57 AM
DATA-009.CSV	728 KB	6/22/2016 10:59 AM
DATA-010.CSV	728 KB	6/22/2016 11:01 AM
DATA-011.CSV	731 KB	6/22/2016 11:03 AM

Figure 4: Data File Selection

5.2.2 R Script Analysis

From the R console window, the following command started the analysis script:

```
> source("{drive}:car_vibration.r")
```

where {drive} is the letter designation for the logger. For example, if the logger mounted as drive D then the command would be: `source("d:car_vibration.r")`

The R script prompted the user for further information regarding the location of the data files. The script can handle data files in other drive locations but this procedure will continue with the example “D” drive. Figure 5 shows the typical input to process file DATA-007.CSV and the resulting plot.

The script generated a plot summarizing the average spectral response for an entire data file. The plot presented in a separate window within the R workspace. The script saved the FFT output to a text file located on the logger, which was later imported into OpenOffice Calc spreadsheet for better plot presentation.

The script also generated a total z-axis RMS vibration acceleration value in the terms of G_{rms}/\sqrt{Hz} . The analysis method removed the DC offset effects caused by gravity so the lower bandwidth starts at 0.5 Hz. The total vibration value is used as a basis for comparison between the different test conditions.

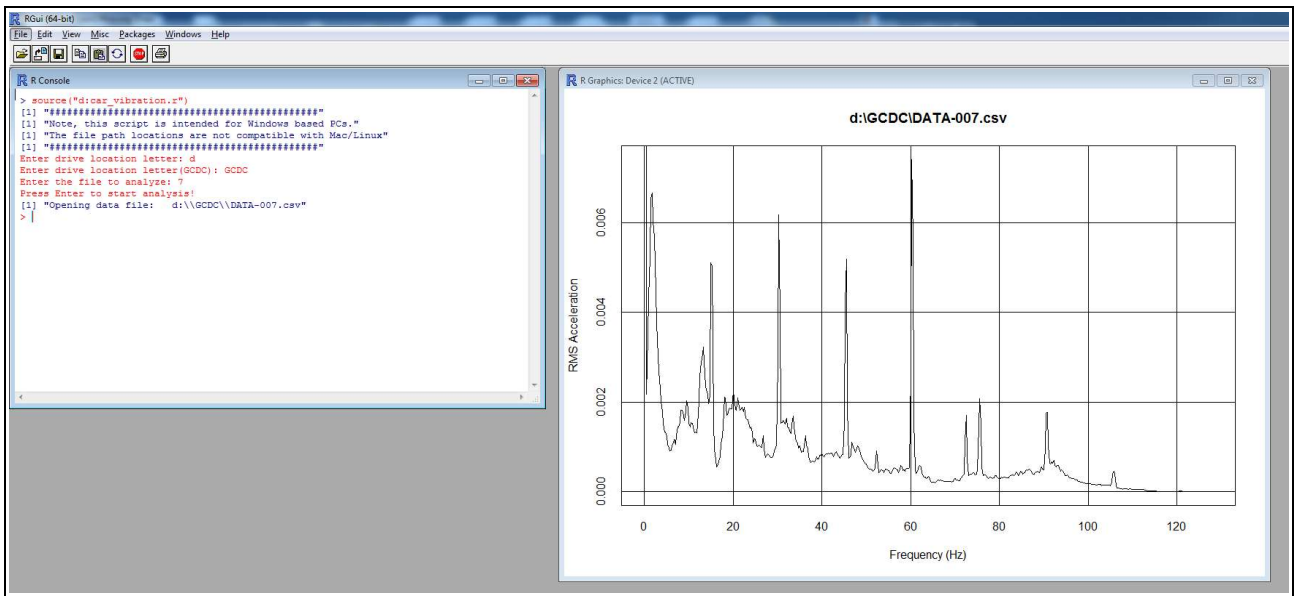


Figure 5: Example R Script Input and Results

6 Results

R is not particularly flexible with formatting plots so the figures below were generated using OpenOffice Calc. Table 3 compares the total RMS acceleration for each test condition.

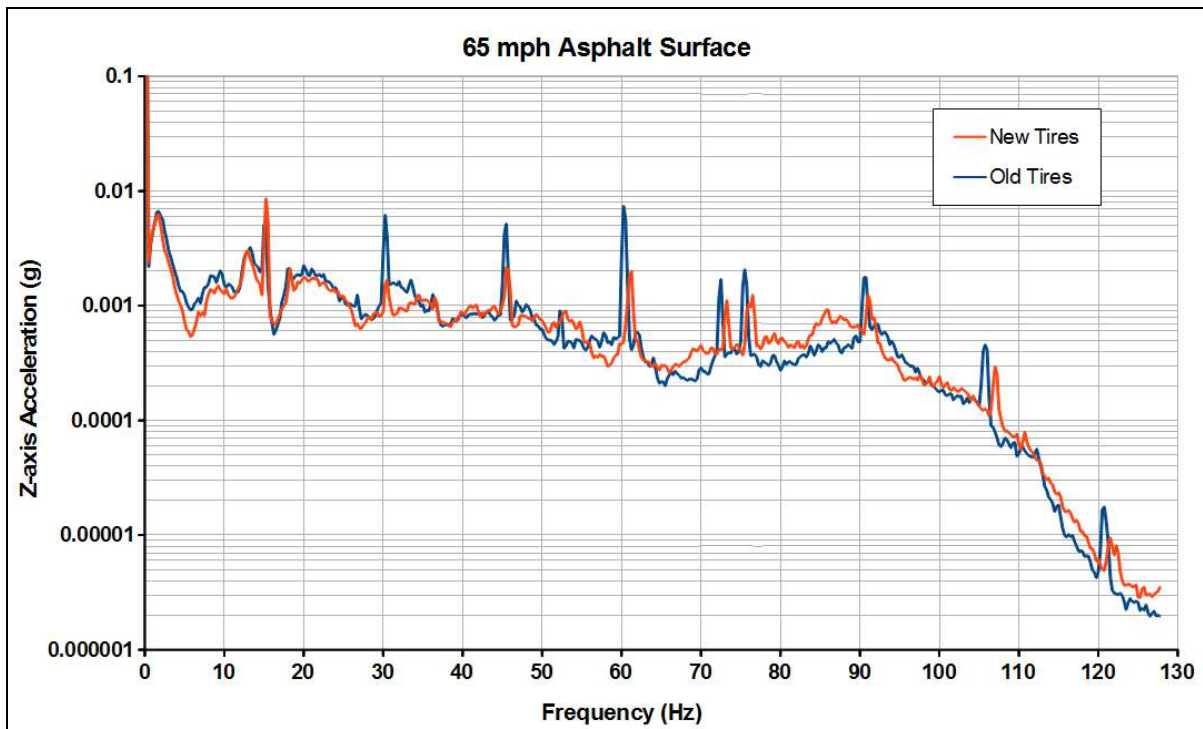


Figure 6: Performance Comparison at 65 mph

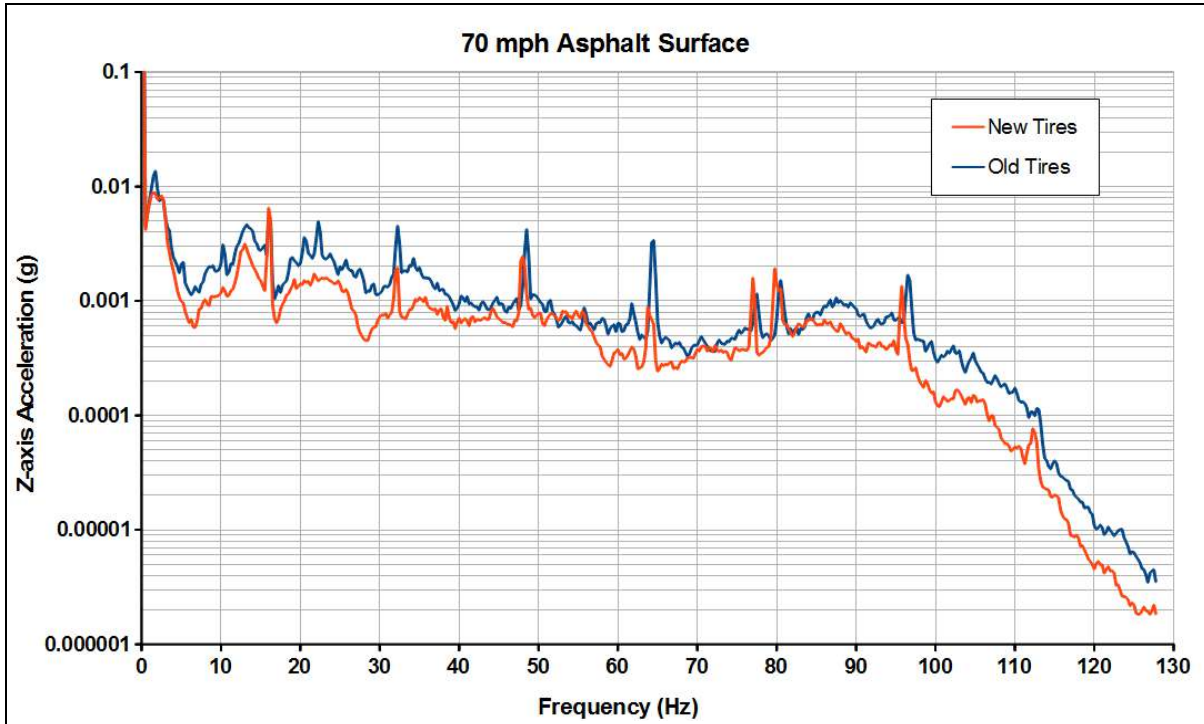


Figure 7: Performance Comparison at 70 mph

Table 3: Total Z-axis Vibration Comparison

Road Condition	Old Tires	New Tires	Change in Total Vibration
	Grms/ $\sqrt{\text{Hz}}$	Grms/ $\sqrt{\text{Hz}}$	%
65 mph Asphalt	0.002698	0.002275	-15.7
70 mph Asphalt	0.003797	0.002832	-25.4

7 Discussion

Figures 6 and 7 illustrate the performance differences between the old and new tires. The distinct “spiked” peaks are due to the rotating wheels. Notice that the primary frequencies are 15.2 Hz at 65mph and 16.3 Hz at 70mph, which corresponds to the 24” diameter wheel spinning at each speed. Subsequent peaks are harmonics of the primary frequency. The new tires (red line) produced a reduction in the vibration peaks over the old tires (blue line). This improvement was evident to the driver as smoother feel at the steering wheel. The slight shift in frequency peaks between test runs is likely due to slight differences in the speed control setting.

Table 3 lists the total RMS vibration in the z-axis across the bandwidth of 0.5 to 128 Hz. A significant reduction in total vibration was observed at both driving speeds, which was observed as a noticeable improvement in the cabin noise.

8 Conclusion

The new tires made a significant improvement to the ride experience. The most obvious change was the reduction in tire harmonics but this could have been due to the re-balanced rims (a standard practice after installing tires) rather than the new tires. The overall reduction in spectral vibration illustrated in Figure 7 reduced the cabin noise and made the Mazda3 more enjoyable at interstate driving speeds.

New tires not only improve safety but also can provide a more comfortable and quieter ride.

9 Appendix A: R Script – car_vibration.r

```
#Gulf Coast Data Concepts
#June 22, 2016
#file name: "car_vibration.r"
#R script for processing X2 accelerometer data into
#a frequency spectrum plot
#-----
#This script takes a segment of data (a block) and converts
#the time series into a frequency series (Fast Fourier Transform)
#The FFT output represents the energy level for each frequency bin
#The analysis repeats by pulling successive blocks of data
#throughout the data set. Each block is plotted during the analysis.
#The results of each FFT are summed and averaged. The output data is
#saved to a text file
#-----

#####
#Functions must be declared first before the script can utilize them #
#####

#fileList is a simple function to build an array of text strings containing
#the files to analyze. This particular script will only use one data file
#as outlined in the Helmholtz test procedure. The fileList is kept here to
#maintain continuity with other scripts. The list will contain only one file
#####
#these lines will build a list using the path, directory, and file numbers defined
#at the beginning of the script
fileList<-function(start, finish, dr, dir){
  dataFiles<-array()
  #prefix name of the data files
  filename<-"DATA-"
  g<-as.numeric(start)
  n<-1
  while(g<(as.numeric(finish)+1)){
    #add leading zeros if need to make valid file name
    if(g<10)
      temp<-paste("00",as.character(g),sep="")
    else if(g<100)
      temp<-paste("0",as.character(g),sep="")
    else
      temp<-as.character(g)
    name<-paste(filename, temp, ".csv", sep="")
    path<-paste(dr,":", "\\\"", dir, "\\\"", name, sep="")
    dataFiles[n]<-path
    g<-g+1
    n<-n+1
  }
  return(dataFiles)
}

#the next few lines of code build a Hann Window filter
#for processing successive FFTs. It's good to filter the
#input data such that there are no discontinuities in the
#transitions between blocks of data. Hann Window is a common
#way to do this.
hannWindow<-function(block){
  #first, initialize the array with a zero
  hann.window<-0
  #the first element of the hann.window array has a 0 so the
  #index will be initialized to '2' to address the next array element
  z<-2
  #start a loop that will build an element array of factors
  #representing a Hann filter
  while(z<(block+1)){
    #this is the formula for a Hann filter
    temp<-0.5*(1-cos((2*pi*z)/(block-1)))
    #take the new factor and append it to the hann.window array
    hann.window<-append(hann.window, temp)
  }
}
```

```

    #increment the index by 1
    z<-z+1
  }
  return(hann.window)
}

#####
#Function definitions are complete. Before starting the main #
#script, we need to collect information from the user regarding #
#files and analysis parameters. #
#####

#First, print a note to the user that this script is intended
#for Windows PCs and the file directory path is not compatible
#with Mac or Linux.
print("#####")
print("Note, this script is intended for Windows based PCs.")
print("The file path locations are not compatible with Mac/Linux")
print("#####")

#ask for the drive location of the data files
#this is the drive or path that contains the GCDC directory
drive<-(-1)
while(drive<0){
  drive<-readline("Enter drive location letter: ")
  drive<-ifelse(grepl("[a-z,A-Z]", drive), as.character(drive), -1)
}

#the directory containing the data files
#ask for the drive location of the data files
#this is the drive or path that contains the GCDC directory
directory<-(-1)
while(directory<0){
  directory<-readline("Enter drive location letter(GCDC): ")
  if(directory==""){
    directory="GCDC"
    break
  }
  directory<-as.character(directory)
}

#prefix name of the data files
filename<-"DATA-"
#the data logger creates files sequentially numbered
#the following parameters define the file numbers to analyze
#for example, files 377 through 380
#ask for the sequential files to analyze
fileNumStart<-(-1)
while(fileNumStart<0){
  fileNumStart<-readline("Enter the file to analyze: ")
  fileNumStart<-ifelse(grepl("[0-9]", fileNumStart), as.numeric(fileNumStart),-1)
}
#the script uses only one file so the fileList will include only
#one entry
fileNumFinish<-fileNumStart

#build a vector list of files to analyze
#the first file is used to determine the start_time
dataFiles<-fileList(fileNumStart, fileNumFinish, drive, directory)

#open the first file and read the first five lines
#the third line contains the start_time
lines<-readLines(dataFiles[1],6)
#parse the lines
date<-unlist(strsplit(lines[3],","))
#combine the 2 and 3 entries into a date/time
d<-paste(date[2],date[3])
#convert the date/time text into a R type date
startDate<-as.POSIXlt(d)

#read the sample rate setting from the first data file

```

```

#the X2 logger records sample rate on line 6 of the file header
temp<-unlist(strsplit(lines[6],","))
SR<-as.numeric(temp[2])

#is user ready to start analysis
test<-(-1)
while(test<0){
  test<-readline("Press Enter to start analysis! ")
  if(test==""){
    break
  }else{
    test<-(-1)
  }
}

#####
#the following parameters affect the analysis algorithm #
#####

#block is a segment of data processed through the FFT algorithm
#The script will process 4 seconds of data thru the FFT
block<-4*SR
#the FFT function can't handle blocks of data larger than 4096
if(block<0 | block>4096){
  print("Invalid block size")
  block<-4096
}

#the script progresses through data set pulling
#blocks of data. Each block of data can overlap the
#previous block of data. Overlapping the data improves
#the signal-to-noise of the results but increases
#computations and time
#define the FFT overlap, 1=0%, 2=50%, 4=75%, etc
overlap<-4 #75% of data will overlap previous block of data
#and 25% will be new data from the file

#####
# Analysis parameters finished. Now, set up variables before # #
# starting the main part of script # #
#####

#check system clock for start of analysis
startTime<-Sys.time()

#cfactor is used to convert the raw data into "g". The factor
#is different depending on the logger type and sensor range.
#X16 series -> 2048
#X2, low gain -> 6554
#X2, high gain -> 13108
#X200 -> 163.8
cfactor<-6554

#initialize three arrays to store the xyz data
dataX<-array(0)
dataY<-array(0)
dataZ<-array(0)

#a time value is calculated for each FFT, time increases with
#each new block of data as the script proceeds through a file.
#The time values are stored to timearray. When a new file is opened,
#timeF is used to carry the last time value into the new file data
time<-0
timeF<-time

#timearray stores all the FFT time values and is used later for
#plotting purposes
timearray<-array(0)

#build a history of FFTs
fftHistory<-array(0)

```

```

#build a Hann Window filter
#when processing successive FFTs it's good to filter the
#input data such that there are no discontinuities in the
#transitions. Hann Window is a common way to do this.
hann.window<-hannWindow(block)

#build an array of frequency values based on fft size
#this is used for plotting the spectral output
x.axis<-0:(block/2-1)
x.axis<-x.axis*((SR/2)/(block/2))

#####
# START OF MAIN SECTION #
#####

#this loop will read the input file into three arrays
#there is only one data file so this loop will complete once
f<-1
while(f<(length(dataFiles)+1)){
  print(paste("Opening data file: ", dataFiles[f]))
  colNames<-c("time", "Ax", "Ay","Az")
  dataIn<-read.table(dataFiles[f], sep="," , comment=";", col.names=colNames, fill=TRUE)

  #initialize three arrays to store the xyz data
  dataX<-array(0)
  dataY<-array(0)
  dataZ<-array(0)

  #use spline to resample the data into evenly spaced samples. The accelerometer
  #sensor streams data at the set sample rate but this may vary depending on
  #the accuracy of the sensor's clock. For example, configuring to sample at 400Hz
  #may actually result in data recorded at 410Hz. We need the resulting data to be
  #very accurate and consistent. Therefore, as data arrives to the logger CPU, it is time
  #stamped using an independent and accurate real time clock. These time stamps
  #are assumed to be accurate. We will resample the data based on
  #the time stamps to ensure the time series data is consistent and accurate.
  #####

  #determine the total time recorded in the file by finding the
  #largest time stamp and subtracting the smallest time stamp
  timeFile=max(dataIn[,1])-min(dataIn[,1])
  #total number of samples that should be in the file assuming
  #the sample rate used. In reality, the sensor may produce data
  #slower or faster than the set rate, that's why we are resampling
  numberSamples=timeFile*SR
  #The time stamps are based on the real time clock so these values are
  #accurate. However, the arrival of the data could be different than the
  #set sample rate due to the sensor's clock error. Therefore, we will
  #use a cubic spline algorithm to resample each axis such that the
  #sample timing is consistent. This will improve the accuracy of the FFT
  #by correcting the sample rate error from the sensor.
  resample<-spline(dataIn[,1],dataIn[,2],n=numberSamples,xmin=min(dataIn[,1]),xmax=max(dataIn[,1]))
  dataTime<-unlist(resample[1])
  dataX<-unlist(resample[2])
  resample<-spline(dataIn[,1],dataIn[,3],n=numberSamples,xmin=min(dataIn[,1]),xmax=max(dataIn[,1]))
  dataTime<-unlist(resample[1])
  dataY<-unlist(resample[2])
  resample<-spline(dataIn[,1],dataIn[,4],n=numberSamples,xmin=min(dataIn[,1]),xmax=max(dataIn[,1]))
  dataTime<-unlist(resample[1])
  dataZ<-unlist(resample[2])

  #Convert the raw data into g's
  dataX<-(dataX)/cfactor
  dataY<-(dataY)/cfactor
  dataZ<-(dataZ)/cfactor

  #Evaluate the z-axis data which is vertical to the road surface
  data<-dataZ
  #Another way to look at the data is to use the RMS magnitude all three axis

```

```

#data<-sqrt(dataX^2+dataY^2+dataZ^2)

#this is the main loop to calculate each FFT.  First, a block data points
#is pulled from the input data and the Hann filter applied to it.  An FFT is
#performed on the filtered data.  The "Mod" function extracts the real portion
#of the FFT.  The results of the FFT are summed and then averaged.
i<-1
while((i+block)<length(data)) {
  datablock<-data[i:(i+block-1)] #get the data
  data.hann<-hann.window*datablock #apply the filter
  data.fft<-fft(data.hann) #perform FFT
  #data.fft<-fft(datablock) #no filter
  data.mod<-Mod(data.fft) #Mod separates the real and imaginary values
  data.mod.half<-data.mod[1:(length(data.mod)/2)] #use the real half of FFT

  #correct output so values are in acceleration units (g)
  fftOutput<-data.mod.half/(block/2)

  #calculate the time stamp for this particular block of data
  time<-timeF+(i+block)/SR/60

  #combine the FFT and x.axis into one array for plotting purposes
  output<-array(c(x.axis, fftOutput), dim=c(length(x.axis),2)) #combine arrays
  #plot the output in log scale on the y-axis
  #this plot shows the spectral response of the resonating bottle
  plot(output, type="l", xlim=c(0,SR/2), ylim=c(1e-6,1e6), log="y", tck=1,
        xlab="Frequency", ylab="Total RMS Acceleration (g)")
  text(round(time, 1), x=70, y=50)
  text("elapsed minutes", x=70, y=10)
  #Sys.sleep(0.1) #pause so the plots don't flash by too quickly

  #add the fft result to the history
  fftHistory<-fftHistory+fftOutput

  #save the time stamp associated with this block of data
  timearray<-append(timearray, time)

  #increment the index by a percentage of the block
  #defined by the overlap variable.
  i<-i+(block/overlap)

  #back to the top of loop and repeat everything for
  #the next block of data
}

#update timeF to track time into the next data file
timeF<-time

#move to the next file
f<-f+1 #increment f

#back to top of main loop to repeat everything
#for next data file
}
#end of main loop

#convert time into date
timeDate<-startDate+timearray*60

#average out the FFT history
fftHistory<-fftHistory/(length(timearray))

#plot averaged spectral response for the entire data set
output<-array(c(x.axis, fftHistory), dim=c(length(x.axis),2))
plot(output, type="l", tck=1, xlab="Frequency (Hz)", ylab="RMS Acceleration", xlim=c(0,SR/2),
ylim=c(0,max(fftHistory[5:length(fftHistory)])), main=dataFiles[1])
text(paste("Data File:",dataFiles[1]), x=SR/4, y=0.005, col="red")

#finished analysis
finishTime<-Sys.time()
#print(paste("total time for analysis: ",(finishTime-startTime)/60," minutes"))

```



```
#write the results to the output file
#uncomment the next two lines to have the results stored to a file
outputDataFile<-paste(drive,":\\", "analysis_output_of_File_", as.character(fileNumStart), ".txt",
  sep="")
columnOutput=c("freq", "accel (g)")
write.table(output, outputDataFile, sep="\t", row.names=FALSE, col.names=columnOutput)
```

End of Document