Limpet force meter analysis code

2017-10-27

#################################################################################  
## Function loadCalibFile  
## This function imports a calibration data file (.csv) with 4 columns of data  
## that should be titled 'axis', 'direction', 'mass.grams', 'analogValue'.  
## The input should be the file name and directory of the file  
## The output will be a dataframe holding the data.   
loadCalibFile = function(fname = 'CalibrationFiles\_Apr202016.csv' ,  
 fdir = 'Dropbox/Limpet\_force\_meter/'){  
 # Determine which computer we're working on so we can put the appropriate  
 # file path prefix on the Dropbox file directory  
 platform = .Platform$OS.type  
 if (platform == 'unix'){  
 prefixDrive = '~/'  
 } else if (platform == 'windows'){  
 prefixDrive = 'D:/'  
 }  
   
 fdir = paste0(prefixDrive,fdir)  
   
 # Import the calibration data file  
 calib = read.csv(paste0(fdir,fname))   
}  
#################################################################################  
  
  
################################################################################  
# Function plotSeparateAxesCalib  
# A function to plot the individual positive and negative calibration data for  
# each of the axes (X, Y, Z). This includes the zero values for a particular   
# axis in each plot of the positive or negative direction. Note that the Z   
# axis only has one direction (positive = downwards)  
# Input a data frame imported from the raw calibration data file  
  
plotSeparateAxesCalib = function(calib = calib){  
# Pull apart the separate axes calibration data, and include the zero values  
 xpos = calib[calib$axis == 'X' & (calib$direction == 'zero' |   
 calib$direction == 'positive'),]  
 xpos = droplevels(xpos)  
   
 xneg = calib[calib$axis == 'X' & (calib$direction == 'zero' |   
 calib$direction == 'negative'),]  
 xneg = droplevels(xneg)  
   
 ypos = calib[calib$axis == 'Y' & (calib$direction == 'zero' |   
 calib$direction == 'positive'),]  
 ypos = droplevels(ypos)  
   
 yneg = calib[calib$axis == 'Y' & (calib$direction == 'zero' |   
 calib$direction == 'negative'),]  
 yneg = droplevels(yneg)  
   
 zneg = calib[calib$axis == 'Z' & (calib$direction == 'zero' |   
 calib$direction == 'negative'),]  
 zneg = droplevels(zneg)  
  
# Plot the raw data for each axis  
 par(mfrow = c(3,2))  
 #############################  
# Positive X-axis  
 plot(x = xpos$analogValue, y = xpos$mass.grams, type='p',   
 xlab = 'Analog count',  
 ylab = 'Mass, g',  
 main = 'X axis positive')  
 mod = lm(mass.grams~analogValue, data = xpos)  
 modSum = summary(mod)  
 abline(mod)  
 r2 = modSum$adj.r.squared  
 mylabel = bquote(italic(R)^2 == .(format(r2, digits = 3)))  
 legend('topleft', legend = mylabel, bty = 'n')  
 ###########################  
# Negative X-axis  
 plot(x = xneg$analogValue, y = xneg$mass.grams, type='p',   
 xlab = 'Analog count',  
 ylab = 'Mass, g',  
 main = 'X axis negative')  
 mod = lm(mass.grams~analogValue, data = xneg)  
 modSum = summary(mod)  
 abline(mod)  
 r2 = modSum$adj.r.squared  
 mylabel = bquote(italic(R)^2 == .(format(r2, digits = 3)))  
 legend('topright', legend = mylabel, bty = 'n')  
 #######################  
# Positive Y axis  
 plot(x = ypos$analogValue, y = ypos$mass.grams, type='p',   
 xlab = 'Analog count',  
 ylab = 'Mass, g',  
 main = 'Y axis positive')  
 mod = lm(mass.grams~analogValue, data = ypos)  
 modSum = summary(mod)  
 abline(mod)  
 r2 = modSum$adj.r.squared  
 mylabel = bquote(italic(R)^2 == .(format(r2, digits = 3)))  
 legend('topleft', legend = mylabel, bty = 'n')  
 ###########################  
# Negative Y-axis  
 plot(x = yneg$analogValue, y = yneg$mass.grams, type='p',   
 xlab = 'Analog count',  
 ylab = 'Mass, g',  
 main = 'Y axis negative')  
 mod = lm(mass.grams~analogValue, data = yneg)  
 modSum = summary(mod)  
 abline(mod)  
 r2 = modSum$adj.r.squared  
 mylabel = bquote(italic(R)^2 == .(format(r2, digits = 3)))  
 legend('topright', legend = mylabel, bty = 'n')  
 #######################  
# Z-axis  
 plot(x = zneg$analogValue, y = zneg$mass.grams, type='p',   
 xlab = 'Analog count',  
 ylab = 'Mass, g',  
 main = 'Z axis negative')  
 mod = lm(mass.grams~analogValue, data = zneg)  
 modSum = summary(mod)  
 abline(mod)  
 r2 = modSum$adj.r.squared  
 mylabel = bquote(italic(R)^2 == .(format(r2, digits = 3)))  
 legend('topleft', legend = mylabel, bty = 'n')   
} # end of plotSeparateAxesCalib  
  
#############################################################################  
# Produce a list of calibration coefficients for the three axes.   
# The output will be a list with entries X, Y, Z, each containing a field  
# 'intercept','slope', and 'R2' (R-squared)  
# The regression coefficients intercept and slope will convert an input   
# analogValue into an estimate of Force (Newtons), with a positive or negative  
# value depending on the direction of the force application.   
  
calibCoefficients = function(calib = calib){  
 #########################################  
 xax = calib[calib$axis == 'X',]  
# Convert mass into force (Newtons) by multiplying by gravity acceleration  
 xax$Force.N = (xax$mass.grams/1000) \* 9.8066  
# Make negative-direction values into negative forces  
 xax$Force.N[xax$direction == 'negative'] = -1 \*   
 xax$Force.N[xax$direction=='negative']  
 #########################################  
# Y-axis all data, converted to force in Newtons  
 yax = calib[calib$axis == 'Y',]  
# Convert mass into force (Newtons) by multiplying by gravity acceleration  
 yax$Force.N = (yax$mass.grams/1000) \* 9.8066  
# Make negative-direction values into negative forces  
 yax$Force.N[yax$direction == 'negative'] = -1 \*   
 yax$Force.N[yax$direction=='negative']   
 ############################################  
# Z-axis all data, converted to force in Newtons  
 zax = calib[calib$axis == 'Z',]  
# Convert mass into force (Newtons) by multiplying by gravity acceleration  
 zax$Force.N = (zax$mass.grams/1000) \* 9.8066  
# Make negative-direction values into negative forces  
 zax$Force.N[zax$direction == 'negative'] = -1 \*   
 zax$Force.N[zax$direction=='negative']  
 ################################################  
 # Fit regressions for each axis  
 modX = lm(Force.N~analogValue, data = xax)  
 modXSum = summary(modX)  
 # Extract intercept, slope, R^2 of regression  
 myinterceptX = coef(modXSum)[1,1]  
 myslopeX = coef(modXSum)[2,1]  
 r2X = modXSum$adj.r.squared  
 ################  
 modY = lm(Force.N~analogValue, data = yax)  
 modYSum = summary(modY)  
 # Extract intercept, slope, R^2 of regression  
 myinterceptY = coef(modYSum)[1,1]  
 myslopeY = coef(modYSum)[2,1]  
 r2Y = modYSum$adj.r.squared  
 ################  
 modZ = lm(Force.N~analogValue, data = zax)  
 modZSum = summary(modZ)  
 # Extract intercept, slope, R^2 of regression  
 myinterceptZ = coef(modZSum)[1,1]  
 myslopeZ = coef(modZSum)[2,1]  
 r2Z = modZSum$adj.r.squared  
 ###############################################  
 # Combine data into an output list  
 output = list(X = data.frame(intercept = myinterceptX, slope = myslopeX,   
 R2 = r2X),  
 Y = data.frame(intercept = myinterceptY, slope = myslopeY,R2 = r2Y),  
 Z= data.frame(intercept = myinterceptZ, slope = myslopeZ,R2 = r2Z))  
 output  
}  
################################################################################  
  
  
################################################################################  
# Function plotAxesCalib  
# This function will produce three plots (one per axis X, Y, Z) to show the   
# calibration data and regression fit to those data. The regression will use  
# analogValue as the x-axis, and force in newtons (converted from mass in grams)  
# for the y-axis.   
# Input should be a data frame of calibration data, with a column 'axis',   
# 'direction', 'mass.grams', and 'analogValue', produced by importing the  
# raw calibration data file   
  
plotAxesCalib = function(calib = calib){  
 ######################################################  
 # Combine all data for a single axis into one set, convert to force units,  
 # estimate regression fit.   
 ######################################################  
 # X-axis all data, converted to force in Newtons  
 par(mfrow=c(3,1))  
 xax = calib[calib$axis == 'X',]  
 # Convert mass into force (Newtons) by multiplying by gravity acceleration  
 xax$Force.N = (xax$mass.grams/1000) \* 9.8066  
 # Make negative-direction values into negative forces  
 xax$Force.N[xax$direction == 'negative'] = -1 \*   
 xax$Force.N[xax$direction=='negative']  
   
 plot(x = xax$analogValue, y = xax$Force.N,las = 1,  
 xlab = 'Analog Count',  
 ylab = 'Force, N',  
 main = 'X axis')  
 abline(h = 0, lty = 2, col = 'grey70')  
 # Fit regression  
 modX = lm(Force.N~analogValue, data = xax)  
 modXSum = summary(modX)  
 abline(modX)  
 myinterceptX = coef(modXSum)[1,1]  
 myslopeX = coef(modXSum)[2,1]  
 r2 = modXSum$adj.r.squared  
   
 # Start by making an expression vector to hold the 2 lines of output:  
 rp = vector('expression',2)  
   
 # Write the first line, which will give R-squared and   
 # pull the value from the data frame wt.fits  
 # The double == prints an equal sign when used inside expression()  
 rp[1] = substitute(expression(italic(R)^2 == MYVALUE),  
 list(MYVALUE = format(r2,digits = 4)))[2]  
   
 # Write the 2nd line, which will pull 2 values from data frame wt.fits:  
 rp[2] = substitute(expression(italic(Y) == MYVALUE2 + MYVALUE3\*x),  
 list(MYVALUE2 = format(myinterceptX, digits = 3),  
 MYVALUE3 = format(myslopeX,digits = 3)))[2]  
 # Finally, simply plot with legend() function:  
 legend('topleft', legend = rp, bty = 'n')  
   
 ############################################  
 # Y-axis all data, converted to force in Newtons  
 yax = calib[calib$axis == 'Y',]  
 # Convert mass into force (Newtons) by multiplying by gravity acceleration  
 yax$Force.N = (yax$mass.grams/1000) \* 9.8066  
 # Make negative-direction values into negative forces  
 yax$Force.N[yax$direction == 'negative'] = -1 \*   
 yax$Force.N[yax$direction=='negative']  
   
 plot(x = yax$analogValue, y = yax$Force.N,las = 1,  
 xlab = 'Analog Count',  
 ylab = 'Force, N',  
 main = 'Y axis')  
 abline(h = 0, lty = 2, col = 'grey70')  
 # Fit regression  
 modY = lm(Force.N~analogValue, data = yax)  
 modYSum = summary(modY)  
 abline(modY)  
 myinterceptY = coef(modYSum)[1,1]  
 myslopeY = coef(modYSum)[2,1]  
 r2 = modYSum$adj.r.squared  
   
 # Start by making an expression vector to hold the 2 lines of output:  
 rp = vector('expression',2)  
   
 # Write the first line, which will give R-squared and   
 # pull the value from the data frame wt.fits  
 # The double == prints an equal sign when used inside expression()  
 rp[1] = substitute(expression(italic(R)^2 == MYVALUE),  
 list(MYVALUE = format(r2,digits = 4)))[2]  
   
 # Write the 2nd line, which will pull 2 values from data frame wt.fits:  
 rp[2] = substitute(expression(italic(Y) == MYVALUE2 + MYVALUE3\*x),  
 list(MYVALUE2 = format(myinterceptY, digits = 3),  
 MYVALUE3 = format(myslopeY,digits = 3)))[2]  
 # Finally, simply plot with legend() function:  
 legend('topleft', legend = rp, bty = 'n')  
   
 ############################################  
 # Z-axis all data, converted to force in Newtons  
 zax = calib[calib$axis == 'Z',]  
 # Convert mass into force (Newtons) by multiplying by gravity acceleration  
 zax$Force.N = (zax$mass.grams/1000) \* 9.8066  
 # Make negative-direction values into negative forces  
 zax$Force.N[zax$direction == 'negative'] = -1 \*   
 zax$Force.N[zax$direction=='negative']  
   
 plot(x = zax$analogValue, y = zax$Force.N, las = 1,  
 xlab = 'Analog Count',  
 ylab = 'Force, N',  
 main = 'Z axis')  
 abline(h = 0, lty = 2, col = 'grey70')  
 # Fit regression  
 modZ = lm(Force.N~analogValue, data = zax)  
 modZSum = summary(modZ)  
 abline(modZ)  
 myinterceptZ = coef(modZSum)[1,1]  
 myslopeZ = coef(modZSum)[2,1]  
 r2 = modZSum$adj.r.squared  
   
 # Start by making an expression vector to hold the 2 lines of output:  
 rp = vector('expression',2)  
   
 # Write the first line, which will give R-squared and   
 # pull the value from the data frame wt.fits  
 # The double == prints an equal sign when used inside expression()  
 rp[1] = substitute(expression(italic(R)^2 == MYVALUE),  
 list(MYVALUE = format(r2,digits = 4)))[2]  
   
 # Write the 2nd line, which will pull 2 values from data frame wt.fits:  
 rp[2] = substitute(expression(italic(Y) == MYVALUE2 + MYVALUE3\*x),  
 list(MYVALUE2 = format(myinterceptZ, digits = 3),  
 MYVALUE3 = format(myslopeZ,digits = 3)))[2]  
 # Finally, simply plot with legend() function:  
 legend('topright', legend = rp, bty = 'n')   
}  
###############################################################################

We have the following calibration files and trial files. Note that the April 2016 trial data should not be analyzed, as the z-axis was not working.

if (.Platform$OS.type == 'windows'){  
 fdir = "D:/Dropbox/Force Meter Files with Luke/Force Meter Data and Calibration Files/"  
} else if (.Platform$OS.type == 'unix'){  
 fdir = "~/Dropbox/Force Meter Files with Luke/Force Meter Data and Calibration Files/"  
}  
  
  
fnames = dir(fdir)  
fnames # print file names

## [1] "CalibrationFiles\_Apr202016.csv"   
## [2] "CalibrationFiles\_Dec132016.csv"   
## [3] "CalibrationFiles\_Dec142016.csv"   
## [4] "CalibrationFiles\_Jan42017.csv"   
## [5] "CalibrationFiles\_Jan52017.csv"   
## [6] "ForceMeterData\_Apr262016.csv"   
## [7] "ForceMeterData\_Dec142016.csv"   
## [8] "ForceMeterData\_Dec152016.csv"   
## [9] "ForceMeterData\_Jan52017.csv"   
## [10] "ForceMeterData\_Jan62017.csv"   
## [11] "ForceMeterData\_May182016.csv"   
## [12] "ForceMeterData\_May192016.csv"   
## [13] "Pound\_ForceData\_Final.csv"   
## [14] "Pound\_ForceData\_withXYZ\_forces.csv"

# Do not bother with Apr 26 2016 data, the Z-axis wasn't working.   
  
#[1] "CalibrationFiles\_Apr202016.csv"  
#[2] "CalibrationFiles\_Dec132016.csv"  
#[3] "CalibrationFiles\_Dec142016.csv"  
#[4] "CalibrationFiles\_Jan42017.csv"   
#[5] "CalibrationFiles\_Jan52017.csv"   
#[6] "ForceMeterData\_Apr262016.csv"   
#[7] "ForceMeterData\_Dec142016.csv"   
#[8] "ForceMeterData\_Dec152016.csv"   
#[9] "ForceMeterData\_Jan52017.csv"   
#[10] "ForceMeterData\_Jan62017csv.csv"  
#[11] "ForceMeterData\_May182016.csv"   
#[12] "ForceMeterData\_May192016.csv"   
  
# subset names by file type (calibration or force trials)  
calibfiles = fnames[grep(x = fnames, pattern = 'Calibration')]  
forcefiles = fnames[grep(x = fnames, pattern = 'ForceMeterData')]  
eventFile = fnames[grep(x = fnames, pattern = 'Pound\_ForceData\_Final.csv')]  
  
# Load the event file  
events = read.csv(paste0(fdir,eventFile))  
events$Date = as.Date(events$Date, format = '%m/%d/%Y')  
  
# Concatenate all calibration data into one data frame  
for (i in 1:length(calibfiles)){  
 temp = read.csv(paste0(fdir,calibfiles[i]))  
 # Extract date from file name  
 locs = regexpr(text = calibfiles[i],  
 pattern='[[:upper:]][[:lower:]]{2}[[:digit:]]\*.csv')  
 month = substr(calibfiles[i],start = locs[[1]][1],   
 stop = locs[[1]][1] + 2)  
 nums = regexpr(text = calibfiles[i],  
 pattern='[[:digit:]]+')  
 if(attr(nums,'match.length')>5){  
 # 2-digit day value  
 day = substr(calibfiles[i],start = nums[[1]][1],  
 stop = nums[[1]][1]+1)  
 yr = substr(calibfiles[i], start = nums[[1]][1]+2,  
 stop = nums[[1]][1]+5)  
 } else {  
 # 1- digit day value  
 day = substr(calibfiles[i],start = nums[[1]][1],  
 stop = nums[[1]][1])  
 yr = substr(calibfiles[i], start = nums[[1]][1]+1,  
 stop = nums[[1]][1]+4)  
 }  
 # Assemble date  
 temp$Date = as.Date(paste0(month,'-',day,'-',yr),format = "%b-%d-%Y")  
   
 if (i == 1){  
 calibs = temp  
 } else {  
 calibs = rbind(calibs,temp)  
 }  
}  
  
if (!exists('forces')) rm(forces)  
  
# Concatenate all force data into one data frame  
for (i in 1:length(forcefiles)){  
 temp = read.csv(paste0(fdir,forcefiles[i]))  
  
 # Extract date from file name  
 locs = regexpr(text = forcefiles[i],  
 pattern='[[:upper:]][[:lower:]]{2}[[:digit:]]\*.csv')  
 month = substr(forcefiles[i],start = locs[[1]][1],   
 stop = locs[[1]][1] + 2)  
 nums = regexpr(text = forcefiles[i],  
 pattern='[[:digit:]]+')  
 if(attr(nums,'match.length')>5){  
 # 2-digit day value  
 day = substr(forcefiles[i],start = nums[[1]][1],  
 stop = nums[[1]][1]+1)  
 yr = substr(forcefiles[i], start = nums[[1]][1]+2,  
 stop = nums[[1]][1]+5)  
 } else {  
 # 1- digit day value  
 day = substr(forcefiles[i],start = nums[[1]][1],  
 stop = nums[[1]][1])  
 yr = substr(forcefiles[i], start = nums[[1]][1]+1,  
 stop = nums[[1]][1]+4)  
 }  
 # Assemble date   
 thisDate = as.Date(paste0(month,'-',day,'-',yr),format = "%b-%d-%Y")  
 if (thisDate == as.Date('2016-04-26')){  
 # skip any file from 2016-04-26  
 } else {  
 # Remove columns that don't match these names:  
 keepCols = c('Time.msec.','JOY\_X\_signal','JOY\_Y\_signal','BEAM\_Z\_signal')  
 temp = temp[,keepCols]  
 names(temp)[1] = 'Time.msec'  
 temp$Date = thisDate  
   
 if (!exists('forces')){  
 forces = temp  
 } else {  
 forces = rbind(forces,temp)  
 }  
 }  
   
}  
  
# Rename the Time.msec. column to Time.msec  
#names(forces)[1] = 'Time.msec'

The relevant calibration file for the May 2016 trials is the Apr202016 file.

During the import process, the calibration files and force trial files are concatenated together into two data frames, calibs and forces, each with a Date column that can be used to separate different days. The data frame events contains the identified peck and push events, with dates and millisecond timestamps.

The goal is to go through each identified event and extract the X-axis and Y-axis forces (which need to be estimated based on the associated calibration values). The events data frame currently only has the net euclidean norm (total for all 3 axes).

# For each date in the 'calibs' data frame, fit regressions to each axis  
# and determine conversions from raw counts to force in Newtons.   
  
dates = unique(calibs$Date)  
  
for (i in 1:length(dates)){  
 # subset the calibs by date  
 temp = calibs[calibs$Date == dates[i],]  
   
 tempCalib = calibCoefficients(temp)  
 # Assemble into a list based on date  
 if (i == 1){  
 calibCoeffs = list()  
 calibCoeffs[[1]] = tempCalib   
 calibCoeffs[[i]]$Date = dates[i]  
 # names(calibCoeffs[1]) = dates[i]  
 } else if (i > 1) {  
 calibCoeffs[[i]] = tempCalib  
 calibCoeffs[[i]]$Date = dates[i]  
 }  
}  
  
  
for (j in 1:length(calibCoeffs)){  
 if (j == 1){  
 calibDates = calibCoeffs[[j]]$Date   
 } else {  
 calibDates = c(calibDates,calibCoeffs[[j]]$Date)  
 }  
}

# Go through each events in the events data frame, pull out the relevant time  
# which should be in the Time\_msec column and the   
# associated Date. Then go find the relevant rows in the forces data frame  
# to get the raw JOY\_X\_signal, JOY\_Y\_signal, and BEAM\_Z\_signal for a chunk of   
# just before and after the event. Use the relevant calibration data to convert  
# each channel to forces in Newtons,   
events2 = events # make a copy  
events2$X.N = NA # make empty columns  
events2$Y.N = NA  
events2$Z.N = NA  
for (i in 1:length(events$Event)){  
 dateval = events[i,'Date']  
 msecval = events[i,'Time\_msec']  
# Go into forces and find the time point  
# Start by subsetting by Date  
 temp = forces[forces$Date == dateval,]  
 timeMatch = which.min(abs(temp$Time.msec - msecval))  
# Grab some rows ahead and after the timeMatch  
 temp2 = temp[(timeMatch-100):(timeMatch+100),c('JOY\_X\_signal','JOY\_Y\_signal',  
 'BEAM\_Z\_signal','Time.msec','Date')]  
# Use the initial rows as a local zero offset reading to apply the calibration.  
# Recall that the relevant calibration for 201605 dates is Apr202016.  
 tempcalibs = calibCoeffs[[which.min(abs(calibDates-dateval))]]  
 ##############################################  
# Convert raw count data to forces  
 temp2$X.N = (temp2$JOY\_X\_signal \* tempcalibs$X$slope) +   
 tempcalibs$X$intercept  
 temp2$Y.N = (temp2$JOY\_Y\_signal \* tempcalibs$Y$slope) +   
 tempcalibs$Y$intercept  
 temp2$Z.N = (temp2$BEAM\_Z\_signal \* tempcalibs$Z$slope) +   
 tempcalibs$Z$intercept  
 ##################  
# Calculate an offset from 'zero' for each axis using the first few samples  
# where there is presumably no force being exerted  
 xoffset = mean(temp2$X.N[1:50])  
 temp2$X.N = temp2$X.N - xoffset  
 yoffset = mean(temp2$Y.N[1:50])  
 temp2$Y.N = temp2$Y.N - yoffset  
 zoffset = mean(temp2$Z.N[1:50])  
 temp2$Z.N = temp2$Z.N - zoffset   
# Take the focal time match and write it into the new columns of the events2   
 # data frame.  
 events2[i,'X.N'] = temp2$X.N[101]  
 events2[i,'Y.N'] = temp2$Y.N[101]  
 events2[i,'Z.N'] = temp2$Z.N[101]  
   
# Also generate a list containing an entry for each event that will hold the  
 # relevant time points  
 if (i == 1){  
 forceList = list()  
 }   
 forceList[[i]] = temp2  
 forceList[[i]]$Event = events[i,'Event']  
 # Calculate the euclidean norm of all three axes forces  
 xyz = as.matrix(forceList[[i]][,c('X.N','Y.N','Z.N')])  
 for (k in 1:nrow(xyz)){  
 # norm() function computes the norm. Needs to work on one row of data at  
 # a time  
 forceList[[i]]$Norm[k] = norm(xyz[k,],"2")   
 }  
   
}   
# Save output  
# write.csv(events2,file='Pound\_ForceData\_withXYZ\_forces.csv',row.names=FALSE)

## DEFINTIONS

* A given force can only be assigned to one category (cannot be a push and peck)
* Forces discussed below refer to Euclidean forces

### Peck:

* Magnitude: ≥ 2N and MUST BE greater than or equal to two times the intensity of the force data points immediately before AND after a given point.
* Duration: Force MUST BE sustained for ≤ 2 consecutive milliseconds
* Both magnitude and duration conditions must be met in order for a force to be categorized as a peck

### Push:

* Magnitude: ≥ 2N and may or may not be greater than or equal to two times the intensity of the force before and after a given force data point
* Duration: May be sustained for any amount of time
* Forces categorized as pushes do not meet conditions of Peck and Touch

### Touch:

* Magnitude: < 2N (e.g. the bird seems to be playing with the ‘limpet’ mimic rather than actively attempting to remove it)
* We are not considering ‘Touches’ in our force meter analyses

# Code to run through a data set and have the user identify probably push or   
# peck events.