# Basins of Attraction in Human Balance

by

Victoria Smith

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2016

## ABSTRACT

According to the CDC[1] in 2010, there were 2.8 million emergency room visits costing $7.9 billion dollars for treatment of nonfatal falling injuries in emergency departments across the country. Falls are a recognized risk factor for unintentional injuries among older adults, accounting for a large proportion of fractures, emergency department visits, and urgent hospitalizations.[2] The objective of this research was to identify and learn more about what factors affect balance using analysis techniques from nonlinear dynamics. Human balance and gait research traditionally uses linear or qualitative tests to assess and describe human motion; however, it is growing more apparent that human motion is neither a simple nor a linear task. In the 1990s Collins,[3,4] first started applying stochastic processes to analyze human postural control system. Recently, Zakynthinaki[5] et al. modeled human balance using the idea that humans will remain erect when perturbed until some boundary, or physical limit, is passed. This boundary is similar to the notion of basins of attraction in nonlinear dynamics and is referred to as the basin of stability. Human balance data was collected using dual force plates and Vicon marker position data for leans using only ankle movements and leans that were unrestricted. With this dataset, Zakynthinaki's work was extended by comparing different algorithms used to create the critical curve (basin of stability boundary) that encloses the experimental data points as well as comparing the differences between the two leaning conditions.

## ACKNOWLEDGMENTS

Special thanks are in order for both of my advisors Dr. Mark Spano and Dr. Thurmon Lockhart for all of their support, advice, and work throughout the entire process. I would also like to thank my committee member Dr. Claire Honeycutt for her knowledge and advice on how to move my project forward in the future. I would also like to thank Dr. Maria S. Zakynthinaki for all of the work she has done in this area and for kindly sharing her data to help me start my project in this field. While working on this project I was funded by the NSF Louis Stokes Alliance for Minority Participation (LSAMP) Fellowship program.

TABLE OF CONTENTS

APPENDIX Page

# LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 The Need

Falls are a well-recognized risk factor for unintentional injuries among older adults, accounting for a large proportion of fractures, emergency department visits, and urgent hospitalizations.[2] According to the CDC's Web-based Injury Statistics Query and Reporting (WISQARS),[1,6] in 2010 3.7 million people, over the age of 50, reported non-fatal fall-related injuries and 24,000 people in this age bracket died from falling or from their injuries. Also in 2010, there were 2.8 million emergency room visits costing $7.9 billion dollars for treatment of nonfatal falling injuries across the country.[7] While interventions to reduce falls have been tested in community-dwelling populations, they have produced mixed success.

Generally a person's level of fall risk is determined by musculoskeletal and sensory functions, fatigue, and training. However these factors are often viewed only with linear measures such as time to walk a predetermined distance, walking velocity, step length, sway area, and sway velocity. Every year, almost a third of people over the age of 65 fall at least once and 10-15% of those falls cause serious injuries or result in death.[8] Early prevention and intervention are paramount in reducing the number of falls.[9] We want to identify at-risk fallers before the first fall to reduce the overall number of falls and the number of injuries that occur when a fall occurs. Some of the more promising predictors of fall risk have sprouted from analyzing stability with nonlinear dynamical tools such as Lyapunov Exponents,[10,11] stochastic resonance,[12] and entropy.[13,14,15] The current stability and gait measures that are often used include a Sit-to-

Stand test, Timed Up and Go test, Tinetti's Mobility index, Roberg tests, gait velocity, center of pressure, and center of pressure velocity. All of these tests are very simple diagnostic tests and easily calculable or are strictly qualitative in nature. The more that is learned and observed during these studies, the more apparent it becomes that standing still and walking are not simple tasks. Thus a more complex algorithm or model may better describe the observed innate motion in order to separate individuals that have a normal and stable stance from individuals who are unstable and are at risk of falling. Postural stability and quiet standing metrics are capable of identifying the differences between individuals who have a history of falling, "fallers", and those who do not have a history of falling, "non-fallers." [16,17,18] Despite being able to identify these individuals after they have fallen, it is still inconclusive if these measures can be used as clinical indications for being at risk of falling and can justify clinical intervention.[19]

## 1.2 Background

Currently fall risk assessment is based on questionnaires and functional tests in the clinical setting. Fall history and questionnaires are mostly qualitative in nature. Functional tests such as Timed-Up and Go, Tinetti tests, or other balance scales are objective and more quantitative, but they lack the ability to discriminate between healthy and fall risk populations.20 The output factors these tests are often only views in terms of a linear system.

Nonlinear dynamics describes the irregular or unpredictable outcomes from a nonlinear system and it appears in almost every discipline from solid mechanics, fluid dynamics, chemical reactions, and even in human movement.[21] One fundamental source of this irregularity is the uncontrolled response to the set of initial conditions of the system and that such irregularity is an intrinsic dynamic of the system and is not due to outside forces. The first step when looking at a dynamical system is to create a geometric phase space, which represents all of the possible trajectories for that system in terms of position and momentum (or velocity). In our case, the dynamical system we are interested in learning more about is human standing balance. A phase space shows all the possible states of the desired system in order to identify all of the possible "attractors". There are three main types of attractors: point, limit cycle, and chaotic. Different types of attractors can coexist and compete in the same phase space for nonlinear systems.[22] A point attractor (repellor) draws (repels) the nearby trajectories to (from) a single point while limit cycles attract or repel periodic motions, as shown schematically in Figure 1. Settling onto an attractor is reaching a steady state of the dynamical system[23].

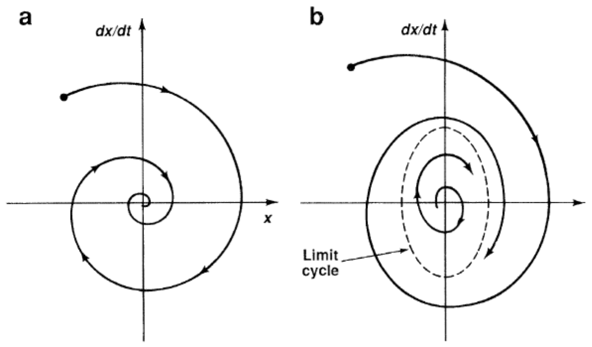


Figure 1: Examples of Attractors. Fig. 1a is an example of a point attractor, pulling the trajectories around itself to a single point. Fig 1b shows an attracting limit cycle in the phase space. All of the trajectories within and outside the limit cycle are approaching the limit cycle. Figure was taken from C. Grebogi et al., *Science* **238**, 4827 (1987).[24]

Attractors can combine attracting and repelling properties, repelling in one direction and attracting in another. Points that do this are called *saddle points*. In a phase space with a single or multiple attractor(s), there also exists a basin of attraction. A *basin of attraction* is the set of all starting points that lead onto a specific attractor and is contained inside a *basin boundary* that describes the edge of that influence. Figure 2 shows an example phase space that contains two attractors, *A* and *B,* with a basin boundary (Σ) separating the space.

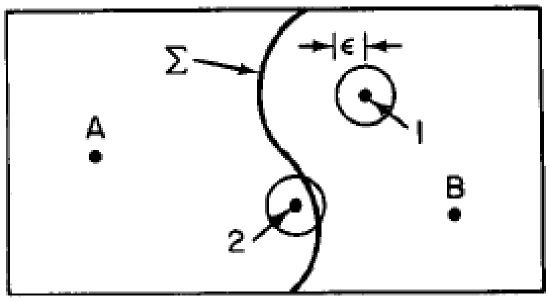


Figure 2: Basin Boundary Example. A region of phase space divided by the basin boundary Σ into basins of attraction for the two attractors A and B. 1 and 2 represent two initial conditions. Taken from C. Gregogi et al., Phys. Lett. A **99**, 9 (1983).[24]

One application of nonlinear dynamics in the field of biomechanics is the use of Lyapunov exponents (LyExp) and entropy measures. Finding the maximum LyExp of a data series evaluates the rate of divergence from neighboring trajectories and the sensitivity of the system to initial conditions (or perturbations).[11,21,25] In a phase space, trajectories will begin to diverge as small changes (or perturbations) are made to the initial conditions. For instance in walking, we take very similar steps from right to left in terms of step size, walking velocity, etc. but not identical. These small changes are due to slightly different initial conditions before we take a step. LyExp evaluates the changes (divergences) between initial conditions and is used as a stability measure for dynamical systems. We can evaluate the dynamic stability of walking by looking at the maximum LyExp, and several papers have already used it as metric for walking stability.[10,11] Another way to use these exponents is to look at the entire LyExp field instead of just the maximum. The field of exponents quantifies the trajectory divergence rate at different locations in the phase space. Tanaka and Ross[26] used this method to study torso stability and sway using a wobble chair. They created boundaries that separate the stable state versus a failure mode in their experiment.

While standing upright, a person is not actually remaining in a fixed position but is swaying in the anteroposterior (front to back) and mediolateral (side to side) directions. This motion is regulated by two independent subsystems and requires a nonlinear system of equations to describe it.[29] As such, postural sway is an example of physiological chaos in the human postural control system.[27,28] For postural stability, a phase space can be created by plotting the center of pressure (COP) and center of mass (COM) in the anteroposterior and mediolateral directions and calculating the angle between the ground reaction forces and the vertical in the anteroposterior and mediolateral directions.[3,27,28,29,30] When one looks at these plots, it is evident that there is a region (in the phase space) where the subject is able to maintain their balance without falling, which can be considered a basin of attraction.

Figure 3 shows what an idealized phase space for postural stability would look like. Maintaining an upright position has a pair of stable attractors, a point attractor at the center representing vertical stance and an infinite number of point attractors that represent falling.[32] The basin boundary, or separatrix, is an infinite number of saddle points, meaning that the closer a trajectory approaches the boundary, the more unpredictable it is to determine in which basin it will end.[25]

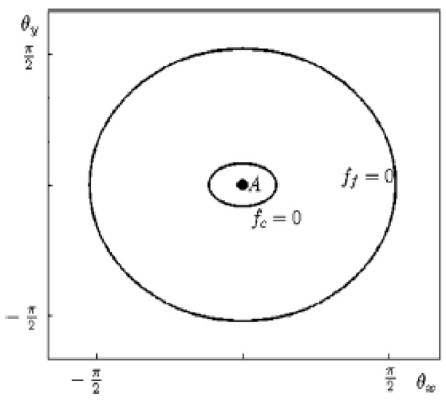


Figure 3: Idealized Phase Space of Postural Stability. Point A is the point attractor for vertical stance and the outermost ring (labeled $f_f$) represents the infinite number of point attractors for falling. The basin of stability (labeled $f_c$ for the critical curve) is an unstable limit cycle that separates the basins of attraction of remaining balanced and falling. Figure was taken from M. S. Zakynthinaki et al., *Appl. Math. Comput*. **219**, 8910 (2013)[32].

## 1.3 Objectives

Maintaining an upright posture is a complex control sensorimotor system involving our visual, vestibular, and proprioceptive systems in the central nervous system as well as information from our neuromuscular system.[3] The integration of all of these inputs allows people to balance, walk, and go about their daily lives without having to concentrate on the movements themselves. Without the correct integration of all these systems or if an individual system starts to fail or degrade, performing those simple daily motions becomes difficult and can lead to falls.

Currently there are very few models that attempt to describe human postural stability, or the ability to remain standing even when an individual is perturbed. One of the more promising models, by Zakynthinaki and Stirling,[28] describes postural stability as a dynamical system that has a basin of attraction, which is responsible for maintaining erect. In essence, the upright standing position is an attractor and balance is maintained as long as the body's position remains within this basin of attraction. If the body moves outside of this basin an individual will lose their balance and fall or require a reactionary response in order to stop from falling. It is important to note that this group did not create a true phase space, which requires position and momentum, but actually only use a portion of the phase space including just the positions. We will be adapting the definition of the phase space to their interpretation from this point forward.

The model designed by Zakynthinaki and collaborators[31,32] displayed the basin of attraction as contours fitted to their experimental data. however, the depth (3rd dimension) of these basins does not have any correlation to a physiological event or action. This lack of connection limits their model's ability to be used to evaluate balance. We propose to improve upon their basin of attraction model by creating a link between the third dimension of the 3D contour basin of attraction model and a physical phenomenon, namely the energy required to regain balance from a specific position. The objective of the current study is to recreate, revise, and compare the basin of stability models by accomplishing the following:

1. Recreate the computer simulation (algorithms) to determine the basin of stability from the original 2004 Zakynthinaki et al. paper.
2. Develop more clinically useful algorithms that describe the basin of stability.
3. Compare the models' accuracy to describe postural behavior.

# CHAPTER 2

# EXPERIMENTAL DESIGN

## 2.1 Experimental Procedures

The experiment consisted of leaning in eight distinct directions and used the GRAIL: Gait Real-time Analysis Interactive Lab, from Motek Medical. This system contains two force plates (Bertec), 10 motion analysis cameras (Bonita), and Vicon Nexus 2.2 software program which captured the marker position and force plate data. We took all of our data directly from Vicon. Our subject was a healthy, 28-year old male. Twenty-five reflective markers were attached to the subject, placement of the tracking spheres can be seen in Figure 4 and the names and their abbreviations for each sphere is included in Table 1. They were placed on anatomically significant landmarks to represent and track the lower extremities and trunk movement.

The standard procedure for the experiment was adapted from Zakynthinaki 2010[31]. The subject's initial and final position were standing on a force plate with both feet together on the floor and with hands on their hips. The subject was asked to lean as far and as fast as he could from this initial position in eight cardinal directions: forward, backward, left, right, forward-left, forward-right, backward-left, and backward-right. The trial was deemed successful if the subject was able to regain his balance and return to the initial position. This was done for two different leaning conditions: rigid body, allowing for movement only at the ankles and non-rigid body, allowing the subject to use all of his lower extremity joints at the subject's discretion. The order of the lean directions was randomized before the experiment started and three trials for each direction were taken, totaling 24 time series for each condition. Additionally, the subject was asked to stand still for 30 seconds at three separate times during the experiment, after every eight leans. In order to avoid fatigue, the subject was asked to come back at a later time to perform the other condition.

The ten-camera (Bonita) system collected 3-dimensional position data of the participant. Position data were sampled and recorded at 100 Hz using Nexus 2.2 software. The ground reaction forces were collected using two force plates (Bertec) at 1000 Hz and were also recorded using Nexus 2.2.

## 2.2 Data Processing

The ground reaction forces were filtered using a low-pass 4th order Butterworth filter at 7 Hz. The force plate data collected was used to calculate the models (described in the next section) and the marker position data was used to calculate the joint angles at the ankle, knee, and hip joints, for both the left and right legs.

## 2.3 3D Marker Locations and Definitions

The marker locations and the naming scheme for each marker were based on the human body model developed by A.J. van den Bogert et al. in 2013.[33] We reduced the number of markers from 47 to 25 for simplicity and to capture only the lower body motions. The locations can be seen in Figure 4 and the abbreviations are explained in Table 1.

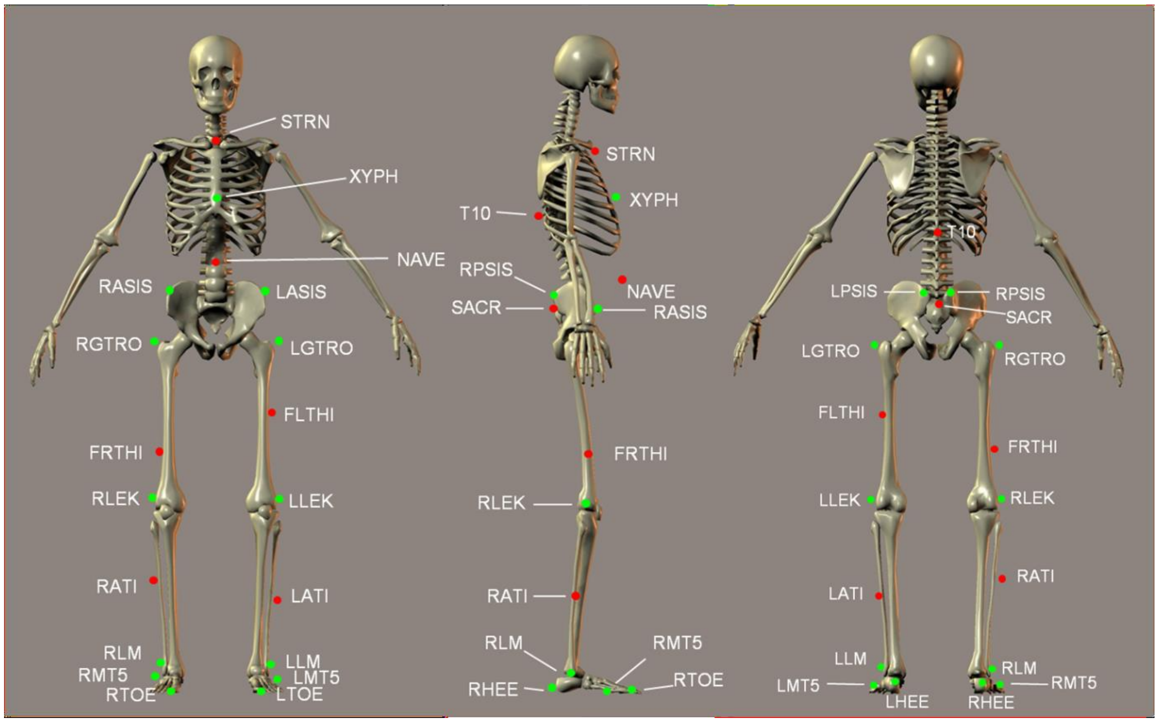


Figure 4: Lower Limb Human Body Model Marker Positions. This figure shows the placement of all 25 markers used to collect data. Taken from “Grail System Manual” by Motek Medical.

Table 1: Lower Limb Human Body Model Marker Details. The table shows the abbreviation or name of the marker used in model and the details about its location on the body.

| No. | Name | Position | Details |
|---|---|---|---|
| 1 | T10 | T10 | On the 10th thoracic vertebrae |
| 2 | SACR | Sacrum bone | On the sacral bone |
| 3 | NAVE | Navel | On the navel |
| 4 | XYPH | Xiphoid process | Xiphoid Process of the sternum |
| 5 | STRN | Sternum | On the jugular notch of the sternum |
| 6 | LASIS | Pelvic bone left front | Left anterior superior iliac spine |
| 7 | RASIS | Pelvic bone right front | Right anterior superior iliac spine |
| 8 | LPSIS | Pelvic bone left back | Left posterior superior iliac spine |
| 9 | RPSIS | Pelvic bone right back | Right posterior superior iliac spine |
| 10 | LGTRO | Left greater trochanter of the femur | On the center of the left greater trochanter |
| 11 | FLTHI | Left thigh | 1/3 of the way between LGTRO and LLEK |
| 12 | LLEK | Left lateral epicondyle of the knee | On the lateral side of the joint |
| 13 | LATI | Left anterior of the tibia | 2/3 of the way between LLEK and LLM |
| 14 | LLM | Left lateral malleolus of the ankle | The center of left lateral malleolus |
| 15 | LHEE | Left heel | Center of the heel, at the same height as the toe |
| 16 | LTOE | Left toe | Tip of big toe |
| 17 | LMT5 | Left 5th meta tarsal | Caput of the 5th meta tarsal bone |
| 18 | RGTRO | Right greater trochanter of the femur | On the center of the right greater trochanter |
| 19 | FRTHI | Right thigh | 2/3 of the way between RGTRO and RLEK |
| 20 | RLEK | Right lateral epicondyle of the knee | On the lateral side of the joint |
| 21 | RATI | Right anterior of the tibia | 1/3 of the way between RLEK and RLM |
| 22 | RLM | Right lateral malleolus of the ankle | The center of right lateral malleolus |
| 23 | RHEE | Right heel | Center of the heel, at the same height as the toe |
| 24 | RTOE | Right toe | Tip of big toe |
| 25 | RMT5 | Right 5th meta tarsal | Caput of the 5th meta tarsal bone |

# CHAPTER 3

# THEORY AND CALCULATIONS

## 3.1 General Description

In all of the models, we assume that the initial position of the subject is the vertical position of quiet stance and that the initial perturbation is away from the vertical (refer to the experimental protocol for perturbation details).

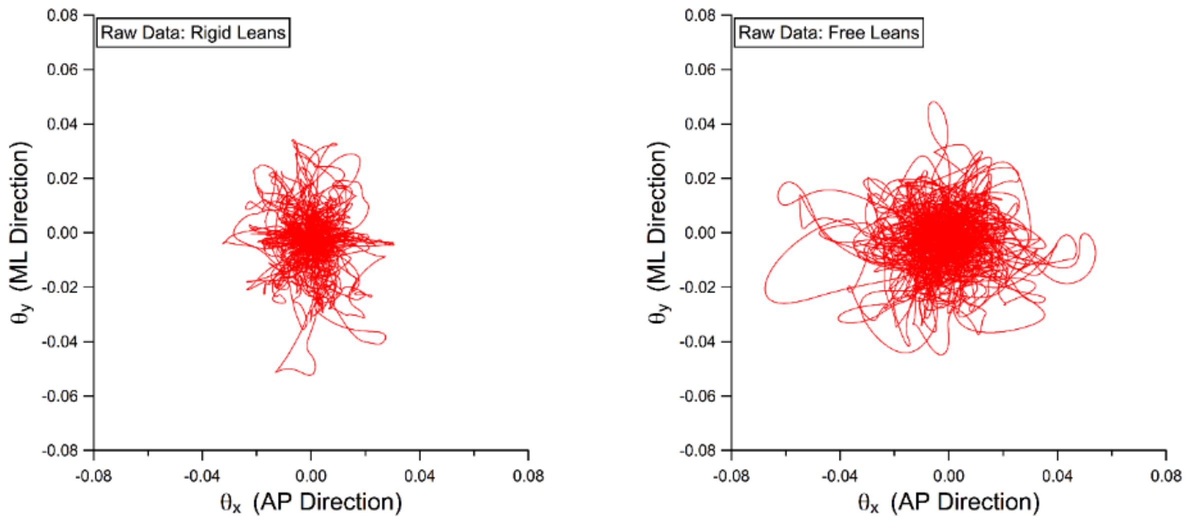


Figure 5: Phase Space of Rigid (left) and Free (left) Leans. Each data set contains all 8 leans with 3 replicates each and was filtered using a 4th order Butterworth filter at 7 Hz.

The angles, $\theta_x$ and $\theta_y$, are the movements in the x (anteroposterior) and y (mediolateral) directions, respectively, in relation to the ground reaction forces recorded from the force plate. These values are calculated using the filtered force plate data. In the original paper[28] only a single force plate was used to calculate these angles. In order to simulate the effect of having only a single force plate from a dual force plate system the following equations were adapted:

$$\theta_x = \tan^{-1}\left(\frac{F_{x1}+F_{x2}}{F_{z1}+F_{z2}}\right) \tag{1}$$

$$\theta_y = \tan^{-1}\left(\frac{F_{y1}+F_{y2}}{F_{z1}+F_{z2}}\right) \tag{2}$$

Using this notation, *Fz* is the vertical component of the ground reaction force and *Fx*, *Fy* are the components of the resultant force in the anteroposterior and mediolateral direction, respectively. Additionally the force plate under the left foot and right foot are assigned to be force plate 1 and 2, respectively. The experimental phase space of both the rigid and free leans is shown in Figure 5. Furthermore we color-coded the individual leans of the free lean data to see how each lean traverses the phase space in Figure 6.

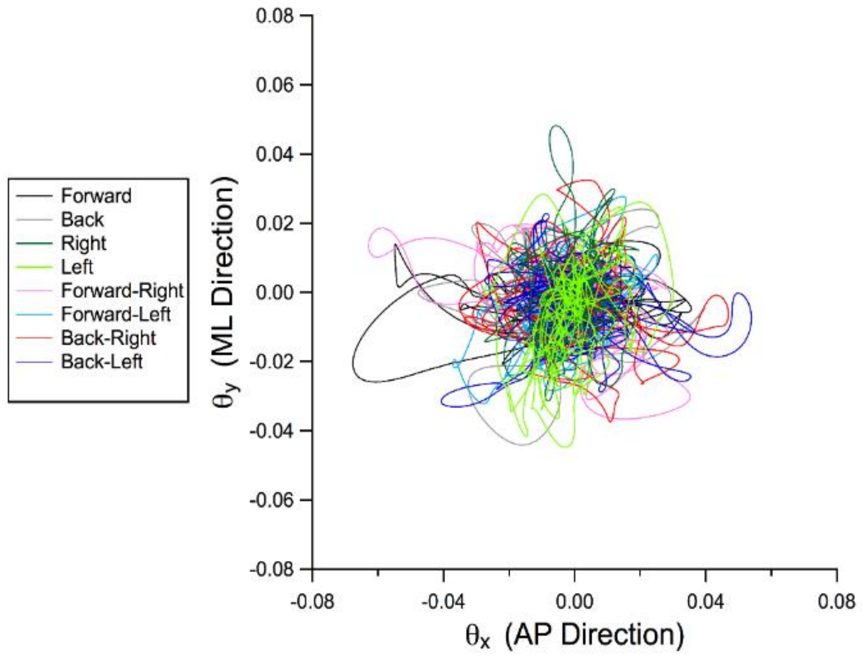


Figure 6: Colored Lean Directions.

The upright attractor in the phase space, i.e. the vertical state, corresponds to the condition that $\theta_x = \theta_y = 0$. The phase space of $\theta_x$ and $\theta_y$ includes a region where upright balance can be maintained and bounded by the “critical curve”. If the body is perturbed beyond this critical curve, it will be “attracted” to the “fallen” attractor and the body must either take a step to regain balance or fall. The important difference between all of the models is the algorithm that is used to calculate the critical curve. For simplicity, all of the equations used to describe these methods will refer to $\theta_x$ as $x$ and $\theta_y$ as $y$.

## 3.2 Zakynthinaki Methods

### 3.2.1 2004 Zakynthinaki Method

Stirling, Zakynthinaki and colleagues[28] developed Equation 3 to describe the critical curve or basin of stability boundary in their 2004 paper. The constants *(A-J)* use the maximum lean angles in the forward, back, left, and right directions found in the in experimental data collected over all 24 leaning trials where $I$ is a scaling factor and was chosen to be 0.3 based on the original papers. These maximum angles are denoted as$\phi_f$, $\phi_b$, $\phi_l$, $\phi_r$, respectively.

$$Ax^2 + Bx + Cy^2 + Dy + Gxy^2 + Hx^2y + Ix^2y^2 + Jxy - E = 0 \tag{3}$$

$$\mathrm{A} = -\phi_l\phi_r \tag{4}$$

$$\mathrm{C} = \phi_r\phi_b \tag{5}$$

$$\mathrm{B} = \phi_l\phi_r(\phi_f + \phi_b) \tag{6}$$

$$\mathrm{D} = \phi_f\phi_b(\phi_l + \phi_r) \tag{7}$$

$$\mathrm{E} = \phi_l\phi_r\phi_f\phi_b \tag{8}$$

$$\mathrm{H} = \phi_l + \phi_r \tag{9}$$

$$\mathrm{I} = 0.3$$

$$\mathrm{G} = \phi_b + \phi_f \tag{10}$$

$$\mathrm{J} = -(\phi_l + \phi_r)(\phi_f + \phi_b) \tag{11}$$

These equations make the following assumptions:

1. The values of the maximum left and backward leans must be less than zero and the maximum right and forward leans must be greater than zero;
2. The maximum forward lean angle must be greater than the maximum backward lean angle;
3. Maximum lean in the left and right directions are approximately equal. This method we have recreated and included in the comparison of all the models.

### 3.2.2 2008 Greek Method and Beyond

In Zakythinaki and collaborators' 2008 paper, they changed their method of evaluating and creating the basin boundary. As you will see in Section VI, the 2004 critical curve equation does not include all of the experimental points due to the asymmetrical nature of these kinds of datasets. The model also breaks when the first assumption, that a person cannot lean farther back than forward, is breached which can happen in the rigid leans and frequently occurs during free leans. In order to correct the fit to the critical curve they redefined the optimal curve to fulfill the five following criteria:

1. A closed curve defined on the $(\theta_x, \theta_y)$ plane
2. A curve that encloses all of the data points
3. A curve with minimum weighted squared distance from the border points of the data set
4. The critical curve passes through specific turning points, there can be one to four different turning points
5. The curve turns after passing through the turning points

In order to find the best fit, they maximize a utility function, Equation 13, using ALOPEX IV –an algorithm of pattern extraction that employs stochastic optimization. This method is summarized in Figure 7.

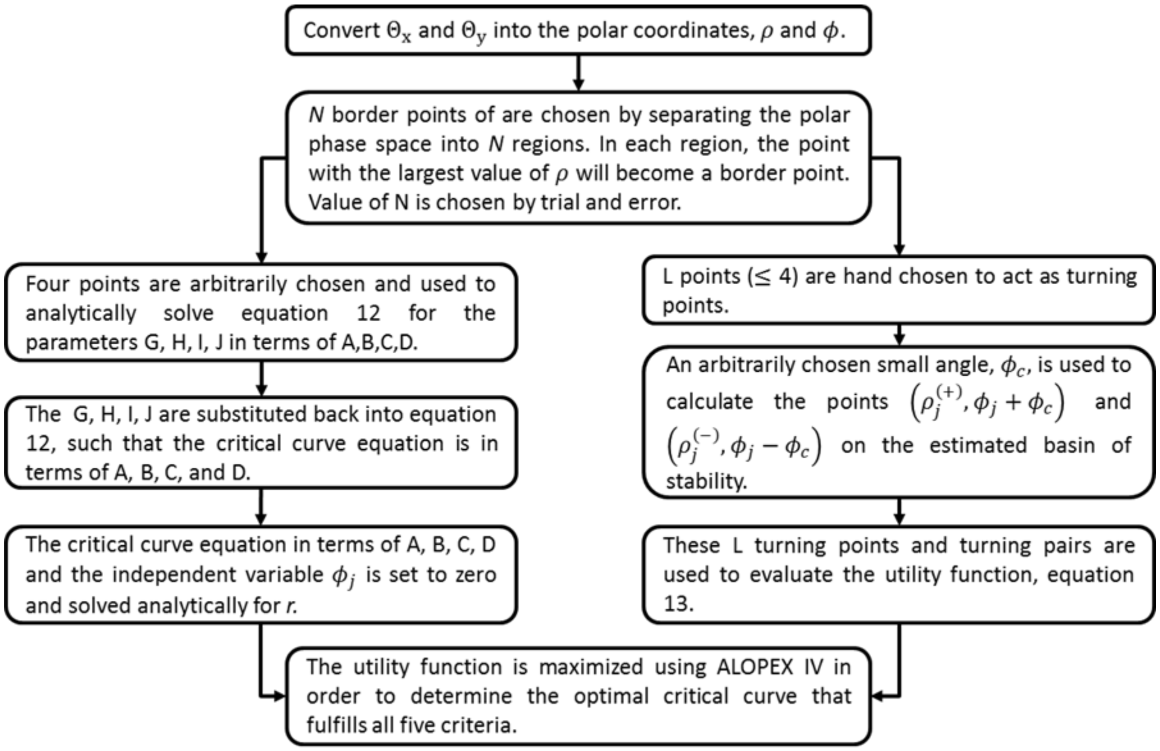

Convert $\Theta_x$ and $\Theta_y$ into the polar coordinates, $\rho$ and $\phi$.
$N$ border points of are chosen by separating the polar phase space into $N$ regions. In each region, the point with the largest value of $\rho$ will become a border point. Value of N is chosen by trial and error.
Four points are arbitrarily chosen and used to analytically solve equation 12 for the parameters G, H, I, J in terms of A,B,C,D.
The G, H, I, J are substituted back into equation 12, such that the critical curve equation is in terms of A, B, C, and D.
The critical curve equation in terms of A, B, C, D and the independent variable $\phi_j$ is set to zero and solved analytically for *r*.
L points ($\leq$ 4) are hand chosen to act as turning points.
An arbitrarily chosen small angle, $\phi_c$, is used to calculate the points $\left(\rho_j^{(+)}, \phi_j + \phi_c\right)$ and $\left(\rho_j^{(-)}, \phi_j - \phi_c\right)$ on the estimated basin of stability.
These L turning points and turning pairs are used to evaluate the utility function, equation 13.
The utility function is maximized using ALOPEX IV in order to determine the optimal critical curve that fulfills all five criteria.

Figure 7: Schematic for Zakynthinaki 2008 Critical Curve Method. This schematic explains the calculation method of the critical curve in the 2008 and later papers written by Zakynthinaki and colleagues.

This method still uses the foundational critical curve expression (Equation 3) but does not use the maximum lean angles to solve for the variables *A*-*J*. Instead the method first converts all of the data into polar coordinates $(\rho, \psi)$. The new critical curve equation is shown below. In polar, *N* border points are chosen by dividing the polar phase plot into *N* wedges, where the value of *N* is chosen by trial and error. Within each wedge, the border point is defined as the data point with the largest $\rho$ value. Four of the border points are hand-picked to analytically evaluate parameters *G*, *H*, *I*, and *J*, using Equation 12, in terms of *A*, *B*, *C*, and *D*.

$$\begin{gathered} I\cos^2(\psi_j)\sin^2(\psi_j)\,\rho_j^4 + \left[H\cos^2(\psi_j)\sin(\psi_j) + G\cos(\psi_j)\sin^2(\psi_j)\right]\rho_j^3 \\ +\left[C\sin^2(\psi_j) + A\cos^2(\psi_j) + J\cos(\psi_j)\sin(\psi_j)\right]\rho_j^2 \\ +\left[B\cos(\psi_j) + D\sin(\psi_j)\right]\rho_j - E = 0 \end{gathered} \tag{12}$$

The expression of *G*, *H*, *I* and *J* are substituted back into equation 12 such that the critical curve equation is now in terms of *A*, *B*, *C*, and *D*. These four values will be the parameters that will be changed to maximize the first part of the utility function.

The four $\rho$ values from the four selected border points are now functions of *r*. The critical curve equation in terms of *A*, *B*, *C*, *D* and the independent variable $\psi_j$ is solved analytically for *r*. The *r* expressions are now ready to be used in the utility function. Note that every $\psi_j$ can generate four *r* values. If any $r_j$ gives more than 2 real solutions it is thrown away because only one solution can be given to the cost function to evaluate. If any of the solutions of $r_j$ is not in the real space they are thrown away. Only an $r_j$ value that is real and has only one real solution is used in the cost function. However, we need to first satisfy the final criterion before running the utility function through an optimization algorithm. Criterion five states that there needs to be four or less, with a minimum of one (in the case of a circle), turning points (L). These points are again chosen by hand to act as turning points. An arbitrarily chosen small angle, $\psi_c$, is used to calculate the points $\left(\rho_j^{(+)}, \psi_j + \psi_c\right)$ and $\left(\rho_j^{(-)}, \psi_j - \psi_c\right)$, where j = 1, …, *L*. These *L* turning points are used in the second half of the utility function.

The utility function, shown below, minimizes the distance between $\rho_i$ and $r_j$ while forcing the critical curve to turn at the *L* turning points.

$$F = -\sum_{i=1}^{M} w_i(\rho_i - r_i)^2 - \sum_{j=1}^{L}\left[\left(\rho_j^{(+)} - \rho_j^{(-)}\right)^2 - \left(\rho_j - \rho_j^{(+)}\right)\left(\rho_j - \rho_j^{(-)}\right)\right] \quad (13)$$

This utility function is maximized by changing the values of the parameters *A*, *B*, *C*, *D* in the expression *r* via the ALOPEX IV or in later papers LMA (Levenberg-Marquardt nonlinear equation solving Algorithm).

We decided to not recreate this optimization scheme for several reasons, such as the high number of arbitrarily picked values and hand-picked data points as well as the time consuming undertaking of analytically solving a quartic equation. We do recognize that the authors fixed the early issues in the 2004 method and their current methods would perform much better under the condition of including all of the data points. Unfortunately because we did not recreate this optimization function we will be unable to directly compare Zakynthinaki's and collaborators most recent work on the basin boundaries of stability with the models that we have created.

## 3.3 Developed Encapsulation Algorithms

A different algorithm for the basin of stability is needed because the definition of a basin of attraction is violated using the original method on experimental data. The basin of attraction is supposed to enclose a region of trajectories that all converge on the attractor. As previously explained in Chapter 1 of this text, the attractor for remaining upright is a point attractor at the center of the basin and falling has an infinite number of point attractors. The basin boundary is the critical separator between the vertical state and falling. If a point falls outside of the basin of stability, it should mean that the subject would be unable to recover from their lean and fall. In the case of the 2004 critical curve method, as seen in Figure 1, not all of the points lie within the basin of stability. The algorithms that we will be presenting in this section are optimized to fit all of the points inside the generated basin of stability.

The first three algorithms that were developed to encapsulate the data were based on the equation of an ellipse, Equation 14.

$$1 = \frac{[(x-h)\cos(\phi)+(y-k)\sin(\phi)]^2}{a^2} + \frac{[(x-h)\cos(\phi)-(y-k)\sin(\phi)]^2}{b^2} \tag{14}$$

where (h,k) is the center of the ellipse and $\phi$ is the angle at which the ellipse is tilted with respect to the x-axis. Now let

$$\lambda = (x-h)\cos(\phi) + (y-k)\sin(\phi), \tag{15}$$

and

$$\eta = (x-h)\cos(\phi) - (y-k)\sin(\phi) \tag{16}$$

such that the equation becomes $1 = \frac{\lambda^2}{a^2} + \frac{\eta^2}{b^2}$ which is the more recognizable equation of an ellipse.

In order to determine the angle $\phi$, the least squares regression line of a session's lean data was calculated. The angle of tilt was calculated to be the inverse tangent of the slope of the regression line, Equation 17. This angle is based purely on the experimental data and is the same angle regardless of which algorithm is being used.

$$\phi = \tan^{-1}(\text{slope}) \tag{17}$$

An ellipse was chosen as the first attempt to create a new basin of stability based on viewing the experimental data, like that shown in Figure 9.

### 3.3.1 Basic ellipse

The first algorithm, Basic Ellipse fit, tries to create the simplest ellipse based on the data. The center point is defined as the midpoint between the maximum and minimum values of *x* and *y*, Equations 18 and 19. The semiaxes of the ellipse were defined as the difference between the center point of the ellipse and the maximum *x* and *y* points, Equations 20 and 21.

$$h = \frac{x_{max}+x_{min}}{2} \tag{18}$$

$$k = \frac{y_{max}+y_{min}}{2} \tag{19}$$

$$a = x_{max} - h \tag{20}$$

$$b = y_{max} - k \tag{21}$$

This algorithm is very fast and simple which is ideal if this data was to be processed on a handheld device for clinicians to use. But is not able to include all of the experimental data within the boundary it creates, meaning that people aren't falling at the points outside of the boundary.

## 3.3.2 One point ellipse

In the second algorithm, called One-Point Ellipse, we solved the minor semi-axis analytically given that we know the value of the major semi-axis and a point on the curve. Just as in the Basic Ellipse fit, the center point is the midpoint between the minimum and maximum *x* and *y* values. In this fit we are using the maximum lean angles in the forward and backward direction to determine the length of the semi axis length, $a$. We project these points onto the regression line and let $a$ be the greatest distance between the center point and either extremum.

Once the length of $a$, major axis, is known we can solve for the length of the minor axis using the following equation:

$$\mathrm{b} = \mathrm{a}\eta\sqrt{\frac{1}{(\lambda^2 - \mathrm{a}^2)}} \tag{22}$$

This method was able to include a greater percentage of data points within its boundary but not every point. The shape does not change much from the BE fit so there was minimal deviation in describing the shape of the data.

## 3.3.3 Two point ellipse

The Two-Point Ellipse fit algorithm requires the center point (*h*,*k*), as defined in the two previous algorithms, the slope of the regression line (ϕ), and all of the maximum lean angles. This algorithm tries to create an ellipse knowing two points that must lie on the boundary line. The algorithm optimizes which points among the maximum lean angles to use by minimizing how many data points fall outside of the boundary. Two of the maximum lean angles are used at a time to determine both the major and minor semi-axes' lengths and every permutation of the four maximum lean angle coordinates is used. For each pair of coordinates, the λ and η are calculated from Equation 15 and 16, these are the numerators of the ellipse equation. This yields the following variables: $\lambda_1, \eta_1, \lambda_2$, and $\eta_2$. The semi-axes are calculated using the following equations, which were solved analytically given that the following was known: two points on the ellipse, center point, and tilt angle. The least number of points that fall outside of the drawn ellipse determines the best ellipse fit.

$$a^2 = \frac{\lambda_1^2}{\left[1 - \frac{\left(\frac{\lambda_2}{\lambda_1}\right)^2 \eta_1^2 - \eta_1^2}{\left(\frac{\lambda_2}{\lambda_1}\right)^2 \eta_1^2 - \eta_2^2}\right]} \tag{23}$$

$$b^2 = \frac{\left(\frac{\lambda_2}{\lambda_1}\right)^2 \eta_1^2 - \eta_2^2}{\left(\frac{\lambda_2}{\lambda_1}\right)^2 - 1} \tag{24}$$

This algorithm includes 99.98% of data points, but lost the ability to describe the shape of the data. The area of the basin is much greater than the other ellipse algorithms. This algorithm is still more complicated than the BE and OPE fits but would still be able to run on a simple device for clinical use.

### 3.3.4 Convex hull

In the final algorithm we wanted to ensure our goal that no data point could ever lie outside of the basin boundary. Through a literature search we chose to use a convex hull approach. A convex hull is the smallest convex curve containing all of the points in a given set.[34] The algorithm used in this paper was based on the Graham scan[35] which has a worst case optimal run time of O(*n* log *n*). The algorithm is used to find the "extreme" or outermost points in the given 2D coordinate dataset that can completely encapsulate the data by creating a polygon.  This algorithm finds the lowest "y-coordinate," point P, in the paired data point set and then sorts the entire dataset in increasing order of the angle they form with P with respect to the x-axis. It then considers each of the sorted points and determines if a "right" or "left" turn is taken. The algorithm looks for a left turn because the points are sorted in a roughly counter clockwise order, so right turns mean that the current path of the polygon is incorrect and other points need to be considered as the outermost points.

If a right turn occurs, the second to last point is not part of the convex hull and each of the preceding points are revaluated until a left turn can be made, as shown in Figure 8. All of the points that were found to be inside of the hull are removed from the sorted data set and the outside points of the hull are saved in a separate data array.

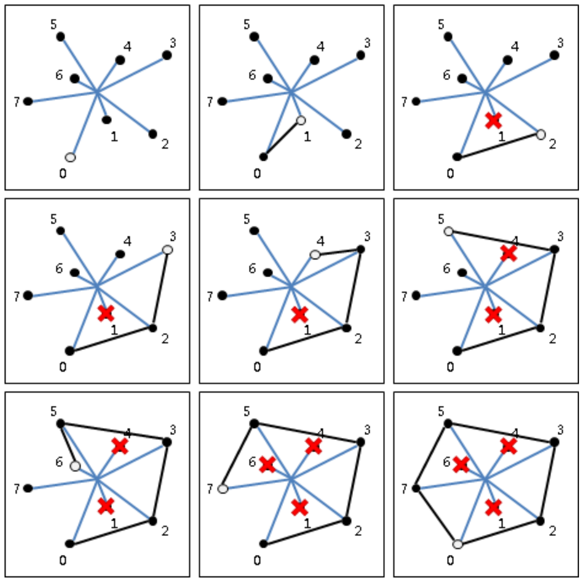


Figure 8: Graham Scan Schematic. Credit to: http://www.csie.ntnu.edu.tw/~u91029/ConvexHull.html

# CHAPTER 4

# MODELING BASINS OF STABILITY

The 2004 Zakynthinaki method for calculating a basin boundary using rigid lean experimental data is shown in Figure 10. A critical curve could not be generated by their method for the free lean data because it broke the assumption that a person could not lean farther backwards than forwards. It is important to note that the critical curve could not be generated for all of the rigid lean data; only one of the three sets of all 8 leans could be used to create the critical curve depicted in Figure 10. This was due to the fact that the maximum backwards-lean angles were greater in magnitude than the forward-leans maximums.

Figure 9 through Figure 13 give a side by side comparison of how the Basic Ellipse (BE) fit, One Point Ellipse (OPE) fit, Two Point Ellipse (TPE) fit, and Convex Hull (CH) fit, respectively, generate the basin of stability for rigid and free leans. Before looking at the differences among the models, it is essential to understand the raw data first. The rigid and free leans do not show much of a difference between mediolateral (left and right) leans, but there is a significant difference in the maximum lean angles in the anteroposterior directions (back and front).

In Figure 9 you can see that the BE fit does not enclose all of the data for either lean type. When the data is close to together, e.g. rigid lean data, the fit does well in not having excess white space but when there are far excursions from the center more white space is inevitably included to capture the movement.

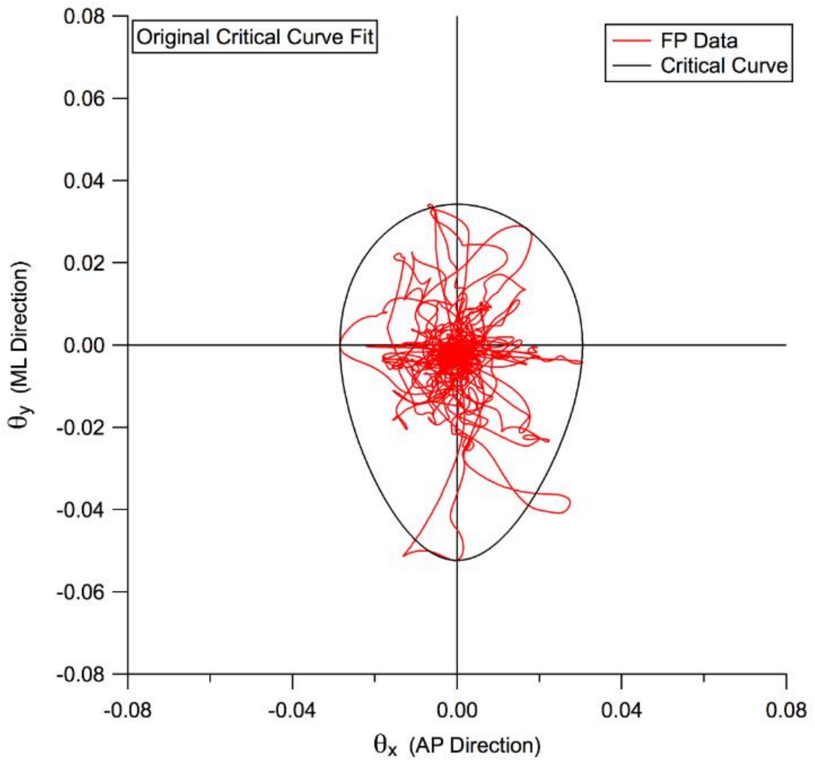


Figure 10: 2004 Zakynthinaki Method using Rigid Lead Data. It is important to note that the data uses only one lean in each of the 8 directions to make this plot and not all 24 leans collected. The force plate data for the other backward and forward leans broke the model due to the fact that the backwards leans were farther than the forward leans.

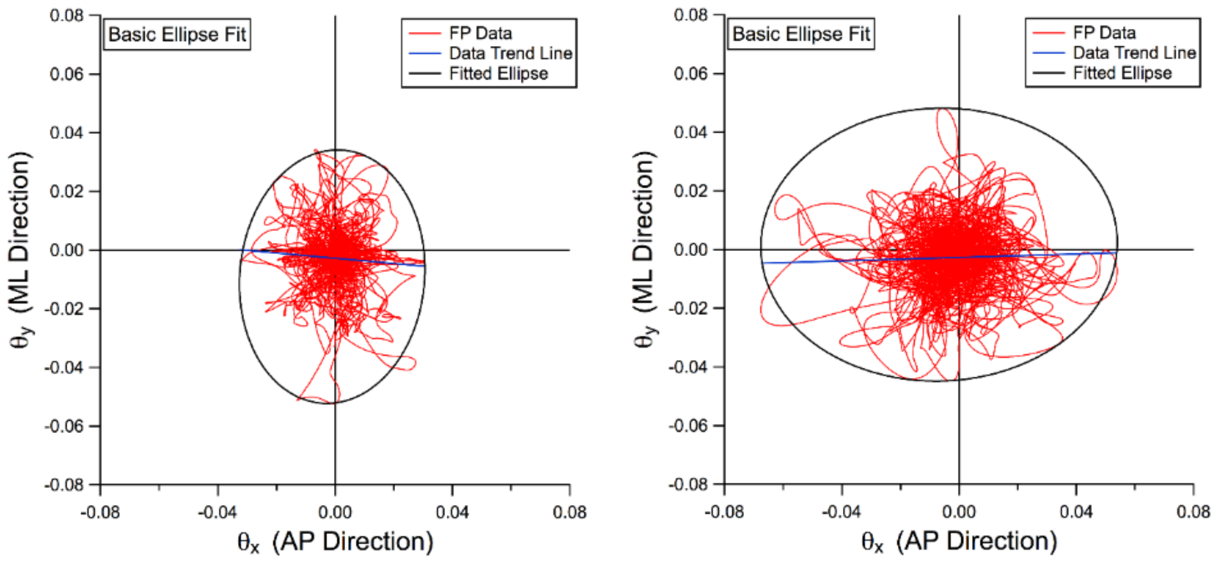


Figure 9: Basic Ellipse Fit. The rigid (right) and free (left) lean data are compared using the same fit. Data is not completely included within the basin of stability fit for either lean data set.

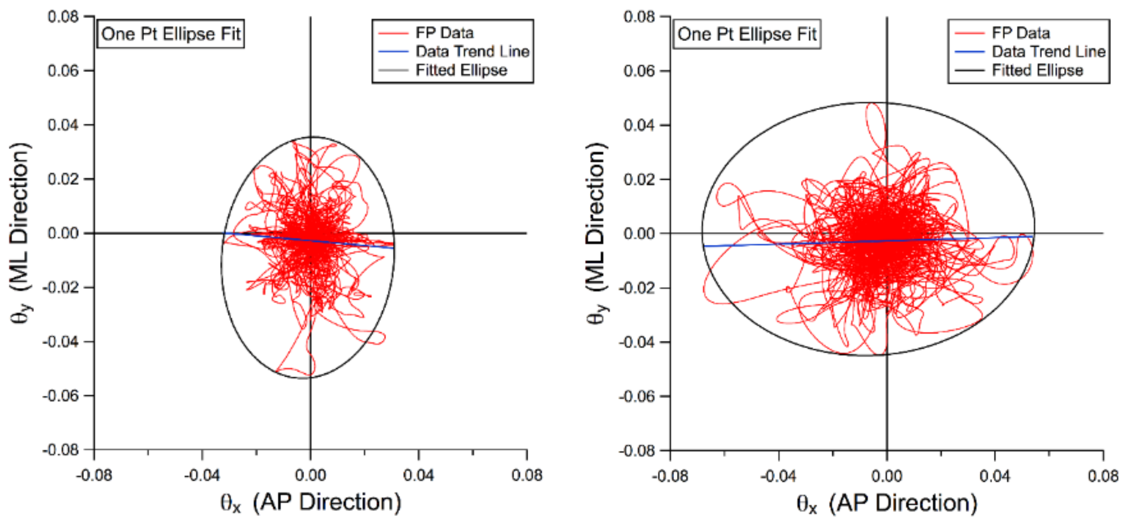


Figure 11: One Point Ellipse fit. This method calculates the linear regression of the force plate (FP) data to tilt the ellipse along.

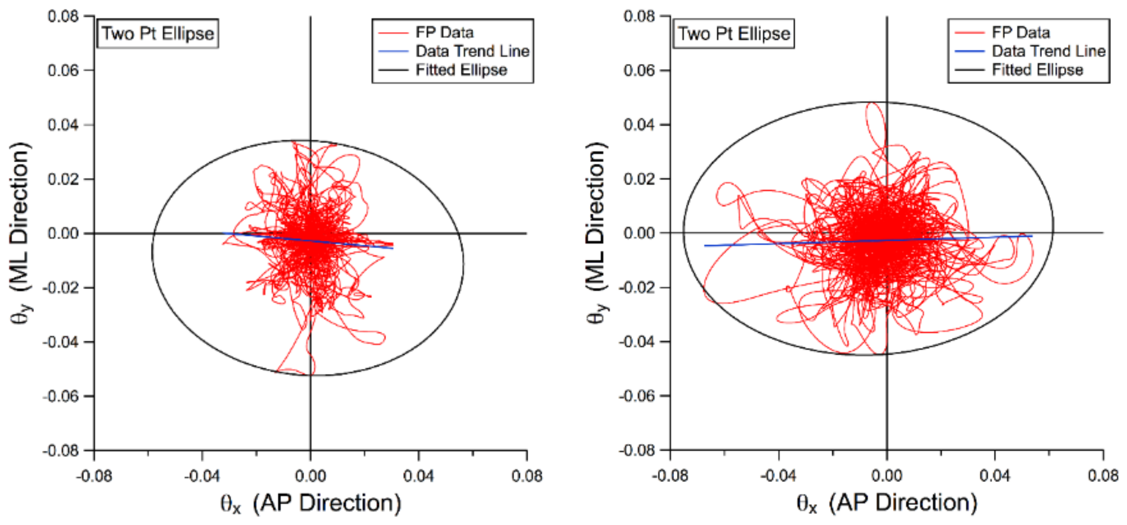


Figure 12: Two Point Ellipse fit. The four maxima of the data sets (in the front, back, left, and right direction) are found. The semi-axes are then calculated using two of the four maxima; all of the pairs are used to calculate the possible axes. The best pair is selected by minimizing the number of points that lay outside of the ellipse generated.

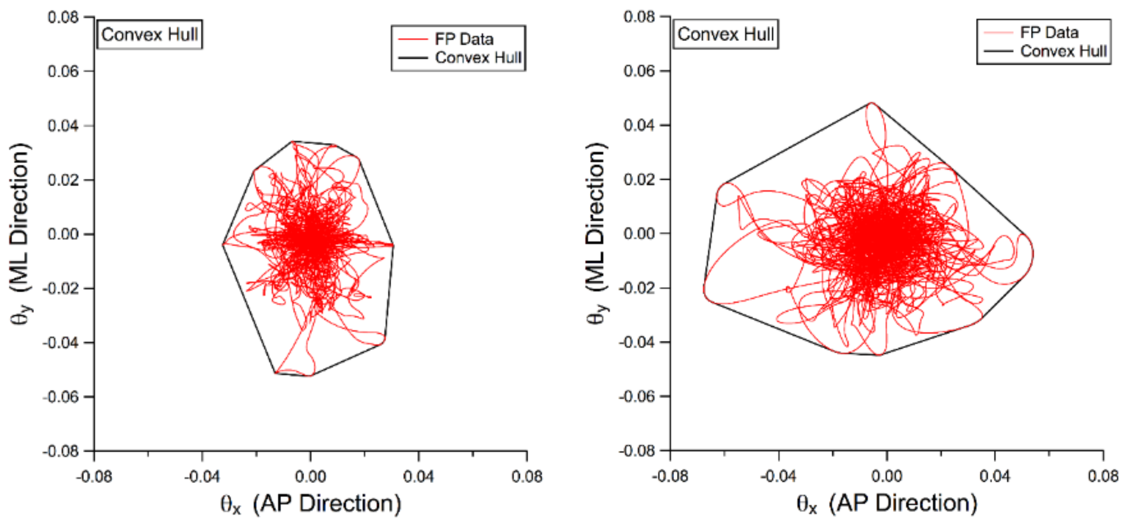


Figure 13: Convex Hull fit. This fit ensures that all of the data points are included within the created polygon. This method also finds the minimum number of outside points to make the final shape.

The OPE, Figure 11, and TPE, Figure 12, fits are similar but have distinct differences as well. The OPE maintains the true shape of the data better than the TPE, however, the TPE was able to include more of the data inside compared to the OPE. If one looks at the OPE for the rigid data, you can see that the major axis is actually orthogonal to the data trend line (blue), whereas in the TPE it is along the trend line. This is due to how the major and minor axes are calculated for each model. In the TPE we expected more leaning to the front and back and made an assumption that the major axis should be along that axis, which as we see in the rigid leans is not always true. Both of these methods improve the number of points that is enclosed compared to the BE fit, but the area also increases. The CH fit, Figure 13, maintained the integrity of the shape of the data and enclosed every data point, improving upon all of the previous models.

The area, perimeter, and the number of points that are outside the generate boundary line were calculated for all five of the basin of stability algorithms. The circumference of the ellipse was estimated using Ramanujan's method.[36] The eccentricity, or the "roundness", for the three ellipse-based algorithms was also calculated. Table 2 and Table 3 summarize these features. All of the fits improved on enclosing more points within the basin of stability, however the area and perimeter also increased. The Convex Hull approach kept the area the smallest and closest to the original algorithm while the Basic Ellipse fit had the smallest perimeter. The Two Point Ellipse had the best inclusion of all the data points, excluding the convex hull; however the area and perimeter of the fit is much larger than the original methods.

Table 2: Rigid Lean Parameter Summary.

| **Summary Table for Rigid Leans** | | | | | |
|---|---|---|---|---|---|
| **Models / Parameters** | **Basic Ellipse** | **One Point Ellipse** | **Two Point Ellipse** | **Convex Hull** | **2004 Stirling, Zakynthinaki et al Model** |
| **Area** | 4.28E-3 | 4.46E-3 | 7.82E-3 | 4.01E-3 | 3.85E-3 |
| **Perimeter** | 236.37E-3 | 241.84E-3 | 318.29E-3 | 237.69E-3 | 228.02E-3 |
| **% Pts Outside** | 0.430% | 0.163% | 0.021% | 0% | 1% |
| **Major Axis** | 31.50E-3 | 31.82E-3 | 57.61E-3 | NA | NA |
| **Minor Axis** | 46.50E-3 | 44.62E-3 | 46.63E-3 | NA | NA |
| **Eccentricity** | 685.71E-3 | 701.02E-3 | 730.79E-3 | NA | NA |
| **Computational Cost** | Low | Low | Low | Medium | High |

Table 3: Free Lean Parameter Summary

| **Summary Table for Free Leans** | | | | | |
|---|---|---|---|---|---|
| **Models / Parameters** | **Basic Ellipse** | **One Point Ellipse** | **Two Point Ellipse** | **Convex Hull** | **2004 Stirling, Zakynthinaki et al Model*** |
| **Area** | 8.88E-3 | 9.00E-3 | 10.01E-3 | 7.47E-3 | - |
| **Perimeter** | 338.61E-3 | 341.16E-3 | 364.34E-3 | 327.91E-3 | - |
| **% Pts Outside** | 0.076% | 0.062% | 0.017% | 0% | - |
| **Major Axis** | 60.80E-3 | 61.44E-3 | 68.32E-3 | NA | NA |
| **Minor Axis** | 43.28E-3 | 46.64E-3 | 46.63E-3 | NA | NA |
| **Eccentricity** | 0.6857059 | 0.65101 | 730.79E-3 | NA | NA |
| **Computational Cost** | Low | Low | Low | Medium | High |

*Method was unable to create a basin boundary for this dataset.

# CHAPTER 5

# EXPERIMENTAL RESULTS & DISCUSSION

The objective of this thesis was to develop new algorithms for finding the basin of stability from experimental data and to create a more substantial link between the basin and physiological movements. We collected force plate and the 3D locations of 25 markers for eight lean types and 30 seconds of quiet stance, with three replicates each. The first lean condition was a free lean where no other instructions other than the direction and to lean as far as you can and return to the initial position as fast as possible without falling. The second lean condition was a rigid lean where only ankle movements were allowed while keeping the rest of the body as straight as possible. This rigid lean type is what is usually seen in the literature[37,38] but it is not a very natural motion. Initially this style was used by researchers because it reducing the degrees of freedom making it easier to do certain calculations as well as reduced possible confounding factors. We wanted to see how much information about a person's balance is lost when using this paradigm by comparing it to what a person would normally do when asked to lean in a given direction.

Our first step at looking at the differences between these two different kinds of leans is to compare the differences between the magnitude of leaning angles during quiet stance and the leaning motions, seen in Figure 14 and Figure 15. The rigid lean style reduces how far forward and back a person leans, while it doesn't seem to affect the left and right leans. Looking more closely at the free lean plot, there doesn't seem to be as big of a difference between leaning backwards and forwards as one might expect since most people can lean farther forward and backwards. While this subject could possibly be more skilled at leaning back than the normal population it does not seem likely. In truth more weight and thus the center of mass of the person is actually transferred to the back of the feet during a backwards lean. When asked to lean forward as far as possible most people bend at the waist, not shifting the center of mass very far, and then try to lean further by going up on their toes, which actually shift the center of mass. Additionally, the successful act of leaning backwards and returning to a standing position involves more joints, i.e. hips, knees, and ankles, than leaning forward. It is also possible that we have more control leaning forward because we do it naturally in some normal daily activities, such as bending over and picking something off the ground, whereas leaning backwards is an unusual motion unless you are trying to recover from a fall or playing limbo. Figure 15 (right) looked more closely at the magnitude difference of leans in quiet stance (black) and leaning (red). This was found by calculating the distance between the center (0,0) and each data point. Phenomenologically, a log normal fit describes both distributions well. The reason for this is not understood at present.

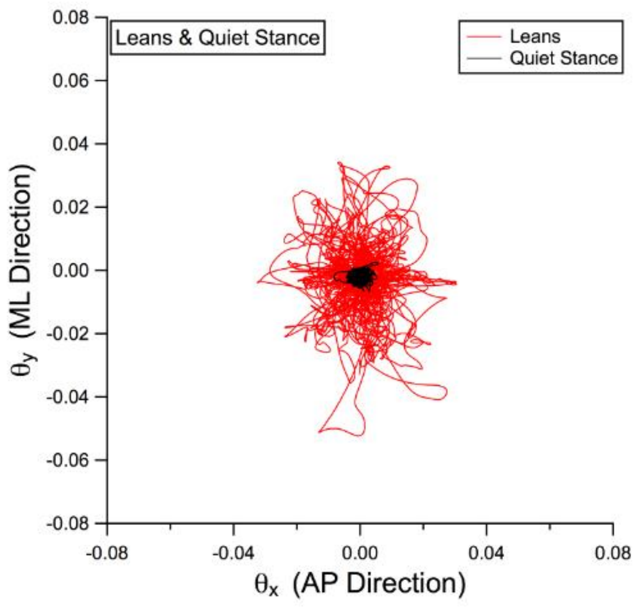


Figure 14: Rigid Lean and Quiet Stance Comparison. We are looking at the magnitude of leaning under rigid conditions to the natural sway we see in quiet stance.

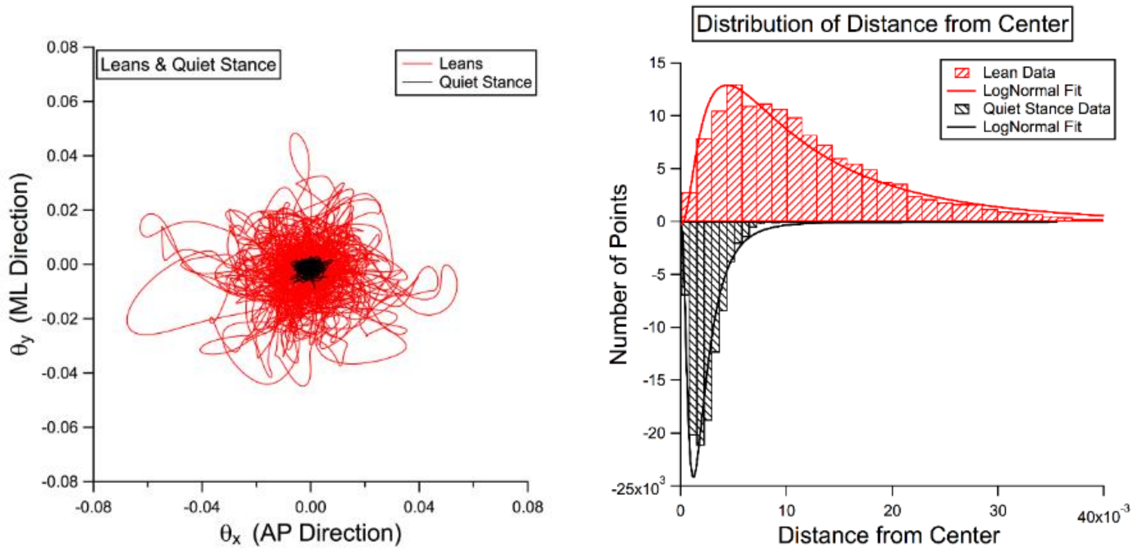


Figure 15: Free Lean and Quiet Stance Comparison. (Left) Comparing the difference in magnitude of the angles calculated from the free lean force plate data and quiet stance force plate data. (Right) Histogram showing the distribution of the distances of each data point from the center (0,0) point for the free lean and quiet stance data.

We previously defined the basin of stability to mean that all the points that are enclosed within that region will converge back onto the center attractor. If you are beyond the boundary then you are in a state of falling or must act (such as taking a step) in order to recover to the standing position. The 2004 Zakynthinaki method used the maximum lean angles in the left, right, back, and forward directions but was unable to encompass all of the data points. Their later methods were able to encompass all of the points but required “special” points for making the final basin of stability, but were more computationally expensive to reproduce and would not be ideal for a clinical test. Therefore we only recreated the 2004 method and developed methods of encapsulation that do not require hand-picked points and that allow the collected data to be able to be quickly processed for clinical use.

Each of the four methods we developed – BE fit, OPE fit, TPE fit, and CH fit – created different basins that included more of the data compared to the recreated Zakynthinaki model, Table 2. The 2004 Zakynthinaki model was unable to create a basin boundary for the free leaning data because the data breaks an assumption. The ellipse algorithms do improve in inclusion, but some lose the ability to adequately describe the overall shape of the data. We calculated the area, perimeter/circumference, and eccentricity for all of the algorithms, when appropriate, in order to quantify how the fit changed from algorithm to algorithm. This is most easily seen in the TPE fit algorithm, Figure 12. It was able to include all but 0.021% of the data points but became extremely elongated along the x-axis with an eccentricity greater than 0.9. The BE fit had similar areas and circumference values to the original method and kept the empty space between the data and the basin of stability to a minimum. The OPE was able to enclose more data into their basin but elongated along the y-axis in comparison to the other methods. This was quantified in the eccentricity of the ellipse, which was greater than the eccentricity of the basic ellipse. The CH fit was the best algorithm of the four new basin of stability methods. It was able to encapsulate all points as well as maintain the integrity of the shape of the data.

The significant differences between the two lean types can be seen when you compare them on the same plots in Figure 16.The dotted lines in the left plot of Figure 16 are the CH fits for each lean data set with the raw data and the right plot shows only the CH fits. In the right Figure 16 plot, you can see that the range of mediolateral leans is similar while the anteroposterior leans are distinctly different. The free leans had a much larger lean angle range, shown in radians, than the rigid leans as we expected. One important point to make about the shape of the overall basin of stability is that we do not know how close to the data the basin of stability should be because it is impossible to collect information about every location within this phase space.

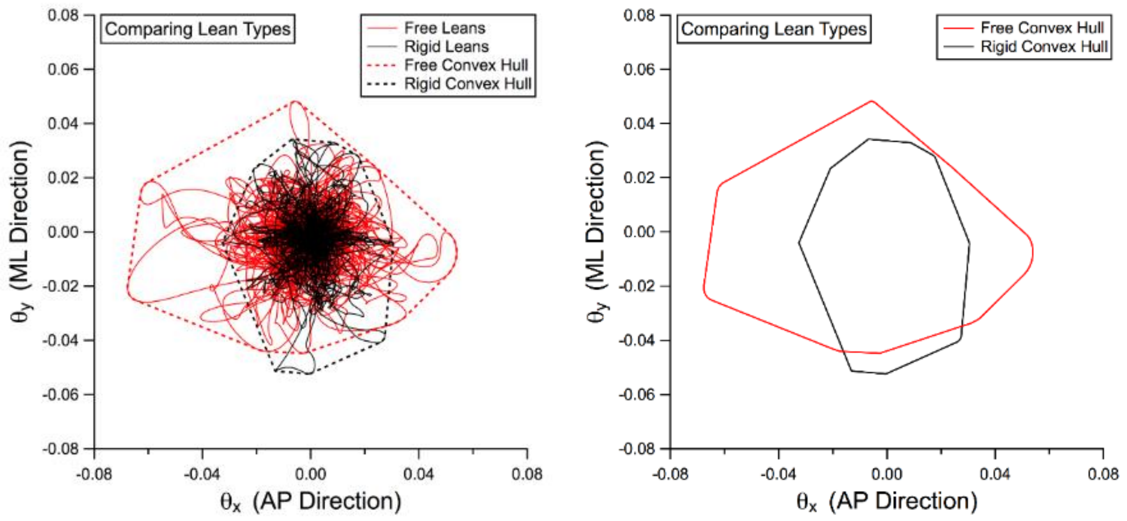


Figure 16: Convex Hull Comparison of Rigid and Free leans. Force data from rigid (black) and free (red) leans are shown on the left plot with the CH fit of each data set in dotted lines with their respective colors. The right plot shows just the convex hulls of the rigid and free leans.

Traditionally, a phase space is a plot comparing position and momentum. However, what we have called a phase space thus far compares the position in the anteroposterior direction and mediolateral direction. Since we have two positions, a correct phase space should have momenta in each direction as well, thus creating a 4 dimensional space. In order to come closer to producing a proper phase space we tried to relate each position (in x and y directions) to the magnitude of the velocity via a calculating information density.

We calculated the information density by dividing the $\theta_x$ and $\theta_y$ axes into a grid of 100 by 100 boxes. The number of data points found in each box was counted, this value corresponds to the density of information found in that box. The density of information in each box is then displayed using different colors to represent the ranging density values, we called this a dwell time plot. The denser the number of points in a given box (shown in purple) means that the subject spent more time in that location. When the color of the box is lighter or red the subject spent less time in that region and moved quickly through the area. Thus it essentially depicts the magnitude of the velocity (which is proportional to the momentum).

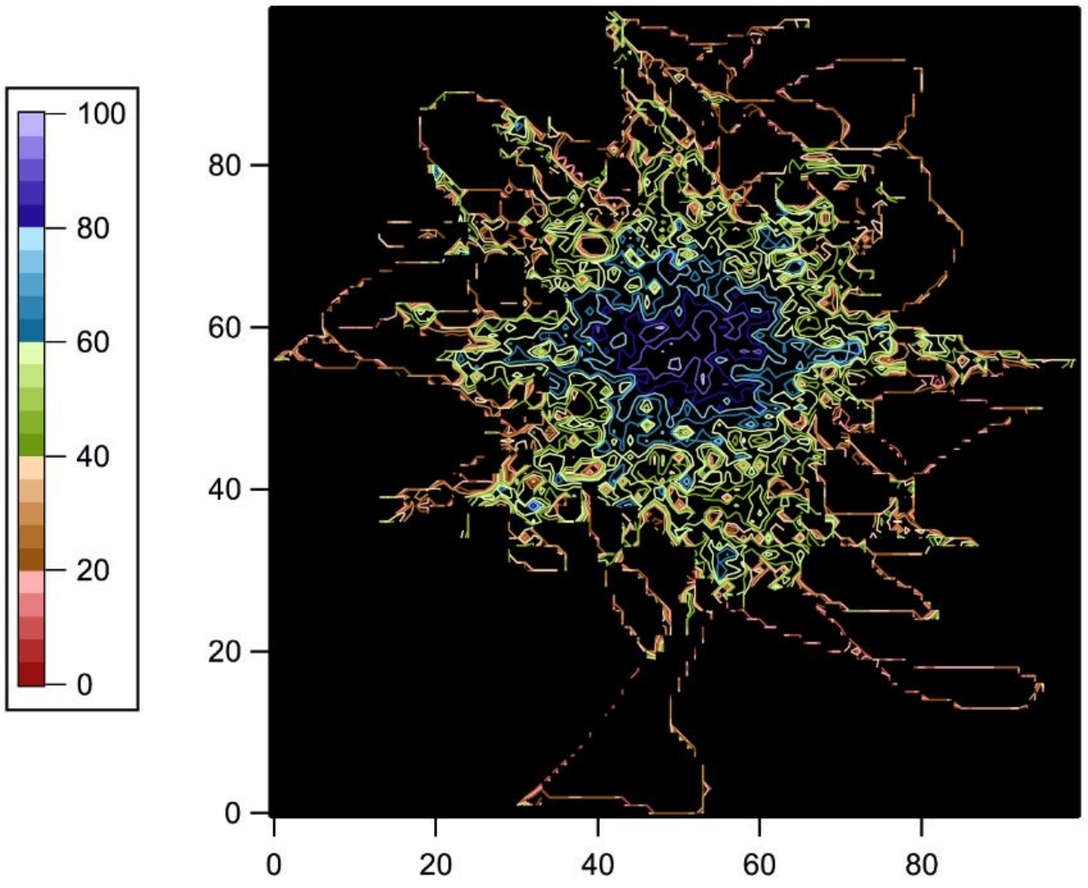


Figure 17: Dwell Time for Rigid Leans. This plot includes all 24 leans collected for the rigid condition. The purples and blues indicate that a lot of time is spent in that region, where areas of red are where the subject spent the least amount of time during a lean motion. The scaling in these plots is based on the number of boxes used to divide the space. The maxima along the x and y-axes were divided into 100 equal lengths with respect to each axis. The color scale represents the number of points that were within each box, purple – up to 100 data points, red – less than 20, and black – no points.

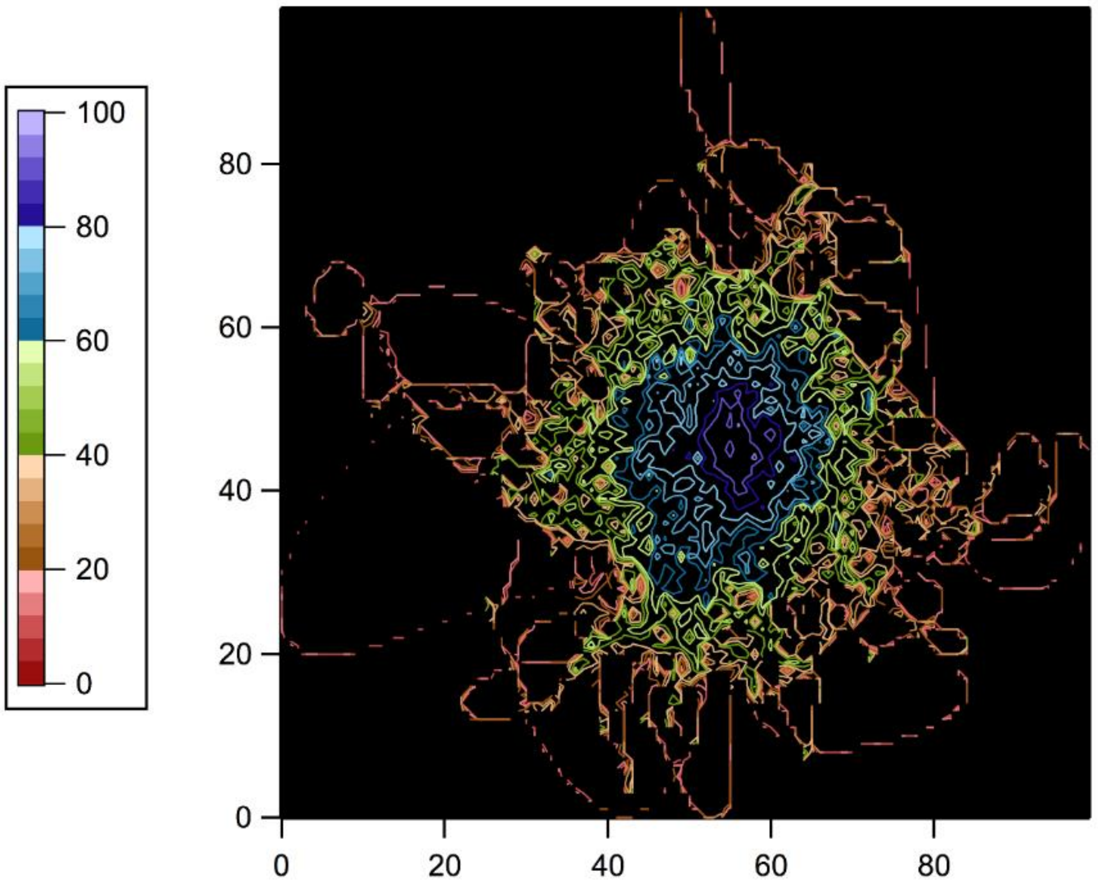


Figure 18: Dwell Time for Free Leans. This plot includes all 24 leans collected for the free condition. The purples and blues indicate that a lot of time is spent in that region, where areas of red are where the subject spent the least amount of time during a lean motion. The scaling in these plots is based on the number of boxes used to divide the space. The maxima along the x and y-axes were divided into 100 equal lengths with respect to each axis. The color scale represents the number of points that were within each box, purple – up to 100 data points, red – less than 20, and black – no points.

Figure 17 depicts the dwell time map for rigid leans and Figure 18 depicts the dwell time for free leans. The dwell time plots were made using different scales based on the maxima points along the x and y axes, which explains why the plots look similar in size when in reality they are not, refer to Figure 16. Both plots have a range of purples in the center, which is expected since this is the start, and end position for the subject. The major difference between the two plots is the differences in the color range. The rigid lean has a lot of green and blue colors with very few paths of red. This is indicative that more time is spent in general at every stage of the lean when the subject is only allowed to lean with their ankles. This physical restriction makes it more difficult for the subject to control their movements thus the subject moves more carefully and slowly to reach their farthest lean before returning to the initial position. Additionally when visually comparing the two lean styles during the experiment, the free lean motion was much more natural where the rigid lean motion made the subject hesitate more and even ask to practice the motion more before continuing with the experiment. Looking at Figure 18, we see a more gradual transition between each color gradient. A lot more of the leaning action is done very quickly and occurs farther from the initial and final position states. It is possible that the more distinctive color regions in a free lean dwell time plot could be an indicator of the individual's ability to balance. Each color could be a specific phase of the movement, such as purple – simple stance; blue – start of the transition to a lean; green – comfortable leaning region; red – pushing the boundaries of leaning before starting to fall. In the future, we would like to quantify this further by calculating the joint torques and power to estimate the energy required during each phase of a lean. We hypothesize that these kinetic measurements might hold significance in determining differences between different age groups and may help identify people who are at risk for falling.

In the future, we would like to add more algorithms as well as make some changes to the current algorithms, such as changing the optimization scheme used to pick the semi-axes lengths for the TPE fit in an attempt to improve the closeness of fit to the experimental data. Two of the major additions to the algorithm lineup would be adapting the CH fit to allow for some “right turns” to further reduce the white space in the fit and create asymmetrical ellipses using singular value decomposition (SVD) methods[39]. Our next steps also including calculating joint angles and moments and seeing how they change during a free and rigid lean. We will be looking into different energy calculations as well as estimating the movement of the center of mass. If the current marker set is not sufficient for accurate estimations changes to the marker locations may be necessary. The current marker layout does not capture much information about the movements of the upper extremities and arms, which could be problematic for calculating center of mass and estimating energy expenditure.

The application of basin of attraction to postural stability may help quantify previously subjective clinical measures. Our intent is to use these algorithms to create a fall risk assessment tool. We plan to apply these measures to several different populations of people including healthy young, healthy elderly, and fall-prone elderly to characterize their differences. We hypothesize that the size of the basin will decrease from young to elderly populations and the basins of healthy and fall prone elderly adults will have different, possibly asymmetric, shapes. For instance, a healthy individual may be able to lean evenly in all direction while a fall-prone individual may only be able to lean well in specific directions. In these future population comparison analyses, all of the basin of stability models should be used to verify the effectiveness and ability of each model to define clinical differences between individuals. We will also be collecting more personal information from the participants such as medical history, questionnaires for quality of daily living and specific motor actions, and a record of their falls or use fall journals. We intend to develop a tool that can take all of these pieces of information along with the experimental data that would be collected to find the best factors or features to look at for determining fall risk, such as a *k*-means clustering algorithm. This data mining method uses the given observations (or parameters) and partitions them into a number (*k)* of clusters; within each cluster is a collection of similar observations. In our case we want to see if the algorithm can separate and differentiate between the different patient populations and yield the best set of parameters to create a comprehensive fall risk assessment for clinics to use.

# CHAPTER 6

# CONCLUSION

The objective of this thesis was to develop new algorithms for creating the basin of stability from experimental data to be used in a clinical setting as a way to assess an individual's risk of falling and possibly to distinguish different causal conditions for that risk. We defined the basin of stability to mean that all the points that enclosed within that region will converge back onto the center attractor. If you are beyond the boundary then you are in a state of falling or must act in order to recover to the standing position. We also wanted to better understand under what physical conditions we approach or cross this basin boundary.

The original method created by Zakynthinaki and collaborators built the framework for this vein of research and analysis. The 2004 Zakynthinaki method used the maximum lean angles in the left, right, back, and forward directions but was unable to encompass all of the data points. Their later methods were able to encompass all of the points but required "special" points for making the final basin of stability making this method unrealistic for a clinical diagnostic test. Therefore we developed methods of encapsulation that do not require hand-picked points while allowing the collected data to be quickly processed and used as a clinical measure.

We developed four methods of encapsulation, Basic Ellipse (BE) fit, One Point Ellipse (OPE) fit, Two Point Ellipse (TPE) fit, and Convex Hull (CH) fit. Each of the four methods tried to include all of the data points within the shape in order to fulfill the definition of a basin boundary We calculated the area, perimeter/circumference, and eccentricity for all of the algorithms, when appropriate, in order to quantify how the fit changed from algorithm to algorithm. We believe that the CH fit was the best algorithm of the four new basin of stability methods because it was able to encapsulate all points as well as maintain the integrity of the shape of the data. One important point to make about the shape of the overall basin of stability is that we do not know how close to the data the basin of stability should be because it is impossible to collect information about every location within this phase space. In future analyses when comparing different populations of people (e.g. young to old, non-fallers to fallers) all of the basin of stability models should be used to verify the effectiveness and ability of each model to define differences between groups.

## REFERENCES

[1] CDC Web-based Injury Statistics Query and Reporting System (WISQARS), "Unintentional Fall Nonfatal Injuries and Rates per 100,000 United States, All Cases, Ages 50 to 85+," 2010, http://www.cdc.gov/injury/wisqars/, (2015).

[2] M. E. Tinetti, "Preventing falls in elderly persons," *N. Engl. J. Med.* **348** (1), 42 (2003).

[3] J. J. Collins and C. J. De Luca, "Open-loop and closed-loop control of posture: A random-walk analysis of center-of-pressure trajectories," *Exp. Brain Res.* **95**, 308 (1993).

[4] J. J. Collins and C. J. De Luca, "Upright, correlated random walks: A statistical-biomechanics approach to the human postural control system," *Chaos* **5**, 57 (1995).

[5] M. S. Zakynthinaki, J. Stirling, C. Martinez, A. Durana, M. Quintana, G. Romo, and J. Molinuevo, "Modeling the basin of attraction as a two-dimensional manifold from experimental data: Applications to balance in humans," *Chaos* **20**, 013119 (2010).

[6] CDC Web-based Injury Statistics Query and Reporting System (WISQARS) , "Fatal Injuries, Both Sexes , Ages 50 to 85 + , United States, 2010 Intent Deaths and Type of Cost Unintentional Average," 2010, http://www.cdc.gov/injury/wisqars/, (2015).

[7] CDC Web-based Injury Statistics Query and Reporting System (WISQARS), "Nonfatal Emergency Department Treated and Released Injuries , Both Sexes , Ages 50 to 85 + , United States , 2010 Intent ED Visits and Type of Cost Unintentional Mechanism Number of ED Visits Fall Average Total," 2010, http://www.cdc.gov/injury/wisqars/ (2015).

[8] A. J. Milat, W. L. Watson, C. Monger, M. Barr, M. Giffin and M. Reid, "Prevalence, circumstances and consequences of falls among community-dwelling older people: results of the 2009 NSW Falls Prevention Baseline Survey." *NSW Public Health Bull.* **22** (4), 43 (2011).

[9] I. D. Cameron, G.R. Murray, L. D. Gillespie, M.C. Robertson, K. D. Hill, R. G. Cumming, and N. Kerse, "Interventions for preventing falls in older people in nursing care facilities and hospitals. *Cochane DB. Syst. Rev.* (2010).

[10] T. E. Lockhart and J. Liu, "Differentiating fall-prone and healthy adults using local dynamic stability," *Ergonomics* **51** (12), 1860 (2008).

[11] A. R. Armiyoon and C. Q. Wu, "A novel method to identify boundaries of basins of attraction in a dynamical system using Lyapunov exponents and Monte Carlo techniques," *Nonlinear Dyn.* **79** (1), 275 (2014).

[12] A. Priplata, J. Niemi, M. Salen, J. Harry, L. A. Lipsitz, and J. J. Collins, "Noise-enhanced human balance control," *Phys. Rev. Lett.* **89** (23), (2002).

[13] P.C. Fino, A.R. Mojdehi, K. Adjerid, M. Habibi, T.E. Lockhart, and S.D. Ross, "Comparing postural stability entropy analyses to differentiate fallers and non fallers," *Ann. Biomed. Eng.* **44** (5), 1636 (2015).

[14] F. G. Borg, and G. Laxåback, "Entropy of balance – some recent results," *J. NeuroEngineering Rehabil.* **7**, 38 (2010).

[15] R. Soangra, and T. E. Lockhart, "Comparison of intra-individual physiological sway complexity from force plate and inertial measurement unit," *Biomed. Sci. Instrum.* **49,** 180 (2013).

[16] L. K. Boulgarides, S. M. McGinty, J. A. Willett, and C. W. Barnes, "Use of clinical and impairment-based tests to predict falls by community-dwelling older adults," *Phys. Ther.* **83**, 328 (2003).

[17] J. A. Norris, A. P. Marsh, I. J. Smith, R. I. Kohut, and M. E. Miller, "Ability of static and statistical mechanics posturographic measures to distinguish between age and fall risk," *J. Biomech.* **38**, 1263 (2005).

[18] P. B. Thapa, P. Gideon, K. G. Brockman, R. L. Fought, and W. A. Ray, "Clinical and biomechanical measures of balance fall predictors in ambulatory nursing home residents," *J. Gerontol. A-Biol.* **51A**, (1996).

[19] M. Piirtola, and P. Era, "Force platform measurements as predictors of falls among older people—a review," *Gerontology* **52** (1)**,** 1 (2006).

[20] D. Hamacher, N. B. Singh, J. H. Van Dieen, M. O. Heller, and W. R. Taylor, "Kinematic measures for assessing gait stability in elderly indivudals: a systematic review" *J. R. Soc. Interface* **8** (65), 1682 (2011).

[21] G. L. Baker and J. P. Gollub, Chaotic Dynamics, an Introduction, 2nd ed. (Cambridge University Press, New York, 1996).

[22] J. M. T. Thompson and H. B. Stewart, Nonlinear Dynamics and Chaos, (John Wiley & Sons Ltd., Chichester, 1986).

[23] H. Kantz and T. Schreiber, Nonlinear Time Series Analysis, 2nd ed. (Cambridge University Press, Cambridge, 2000)

[24] C. Grebogi, E. Ott, and J.A. Yorke, "Chaos, strange attractors, and fractal basin boundaries in nonlinear dynamics," *Science* **238** (4827), 632 (1987).

[25] C. Grebogi, S. W. McDonald, E. Ott, and J. A. Yorke, “Final state sensitivity: an obstruction to predictability,” *Phys. Lett. A* **99**, 415 (1983).

[26] M. L. Tanaka and S.D. Ross, “Separatrices and basins of stability from time series data: an application to biodynamics,” *Nonlinear Dyn.* **58**, 1 (2009).

[27] J. J. Collins and C. J. De Luca, “Random walking during quiet stance,” *Phys. Rev. Lett.* **73**, 764 (1994).

[28] J. R. Stirling and M. S. Zakynthinaki, “Stabiity and the maintenance of balance following a perturbation from quiet stance,” *Chaos* **14**, 96 (2004).

[29] D. A. Winter, F. Prince, J. S. Frank, C. Powell, and K. F. Zabjek, “Unified theory regarding A/P and M/L balance in quiet stance,” *J. Neurophysiol.* **75**, 2334 (1996).

[30] J. Liu, X. Zhang, and T. E. Lockhart, “Fall risk assessment based on postural and dynamic stability using inertial measurement units,” *Safety and Health at Work* **3**, 192 (2012).

[31] M. S. Zakynthinaki, J. Stirling, C. Martinez, A. Durana, M. Quintana, G. Romo, and J. Molinuevo, “Modeling the basin of attraction as a two-dimensional manifold from experimental data: Applications to balance in humans,” *Chaos* **20**, 013119 (2010).

[32] M. S. Zakynthinaki, A. López, C. A. Cordente, J. A. Ospina Betancurt, M. S. Quintana, and J. Sampedro, “Detecting changes in the basin of attraction of a dynamical system: Application to the postural restoring system,” *Appl. Math. Comput.* **219** (17), 8910 (2013).

[33] A. J. van den Bogert, T. Geijtenbeek, O. Even-Zohar, F. Steenbrink, and E.C. Hardin, “A real-time system for biomechanical analysis of human movement and muscle function,” *Med. Biol. Eng. Comput.* **154**, 1069 (2013).

[34] J. O'Rourke, Computational Geometry in C, (Cambridge, UK: Cambridge UP, 1998). Print.

[35] R. L. Graham. “An efficient algorithm for determining the convex hull of a finite planar set,” *Information Processing Letters* **1**, 132 (1972).

[36] G. Almkvist and B. Berndt, “Gauss, Landen, Ramanujan, the arithmetic-geometric mean, ellipses, $\pi$, and the Ladies Diary,” *Amer. Math. Monthly* **95** (7), 585 (1988).

[37] T. M. Owings, M. J. Pavol, K. T. Foley, and M. D. Grabiner, “Measures of postural stability are not predictors of recovery from large postural disturbances in healthy older adults,” *J. Am. Geriatr. Soc.* **48**, 42 (2000).

[38] H. B. Menz, M. E. Morris, and S. R. Lord, “Foot and ankle characteristics associated with impaired balance and functional ability in older people,” *J. Gerontol. A-Biol.* **60**, 1546 (2005).

[39] Tim Sauer, private communication.

# APPENDIX A

# IGOR PRO CODE: MODELS

```
// BasinOfStability_v4.2.pxp
//
//  Created By:             Victoria Smith, vasmith5@asu.edu
//  Created On:             Jan 19, 2016
//  Last Updated On:        Mar 29, 2016
//
//

#pragma rtGlobals=3             // Use modern global access method and strict wave access.
#include <All Gizmo Procedures>
#include <FillBetweenContours>,menus=0

Menu "Macros"
        "Clean up graphs", CleanWindows()
        Submenu "Main Fits"
                "Basic Ellipse", DisplayEllipseFit()
                "1pt Ellipse", Display1PtEllipse()
                "2pt Ellipse", Display2PtEllipse()
                "Convex Hull", DisplayConvexHullFit()
                Submenu "2004 Greek Fit"
                        "All Leans", DisplayGreekMethod()
                        "Individual Groups", DisplayGreekByGroup()
                End
        End
        Submenu "Other Graphs"
                Submenu "Phase Space"
                "All Leans", DisplayPhaseSpace()
                "Grouped Leans", DisplaySingleGroupPhaseSpace()
                End
                Submenu "Lean Directions"
                        "All Leans", DisplayLeanDirections()
                        "Grouped Leans", DisplayGroupedLeans()
                End
                Submenu "Quiet Stance"
                        "Only Queit Stance", DisplayQuietStance()
                        "Quiet Stance w Leans", DisplayLeansAndQS()
                End
                Submenu "Information Density"
                        "All Data", DisplayInfoDensity()
                        "Single Sets", DisplaySingleGroupInfoDensity()
                End
        End
        Submenu "Utility Functions"
                "Print Information", WriteInfoToFile()
                "Load Dataset", InitSession()
                "Delete Dataset", CleanDataFolder()
        End
        "HistTest", HistTest()

End
```

```
//
//
// Utility Functions
//
//

Function CleanDataFolder()                                          //CleanDataFolder
            String folder = SelectDataFolder()

            //Checks if a folder has been selected, if not operation is aborted
            if (strlen(folder) == 0)
                        return -1
            endif

            //Creates a warning that the user is about to delete a folder. If user presses yes, then action will be
            //executed if user presses no, action is aborted.
            String warning = "Do you wish to delete everything in:\t" + folder
            DoAlert/T="Caution" 1, warning

            if (V_flag == 1) //Meaning yes clicked
                        KillDataFolder root:$(folder)
                        //Kills all the Session data folder and all the children df in that session
                        printf "Deleted folder: %s\r", folder
            else //V_flag = 2 (No clicked)
                        printf "%s folder deletion aborted\r", folder
                        return -1
            endif

End


//Function that clears all the Graphs currently being displayed
Function CleanWindows()                                             //CleanWindows
            String windows
            String graphName
            Variable i

            // Kill layouts
            i = 0
            windows = WinList("*",",","WIN:4")
            do
                        graphName = StringFromList(i,windows,",")
                        if (strlen(graphName) == 0)
                                    break
                        endif
                        KillWindow $graphName
                        i += 1
            while(1)
            // Kill graphs
            i = 0
            windows = WinList("*",",","WIN:1")
            do
                        graphName = StringFromList(i,windows,",")
                        if (strlen(graphName) == 0)
                                    break
```

```
		endif
		KillWindow $graphName
		i += 1
	while(1)
End


//Loads multiple data files. These are specifically formatted .txt files from Matlab (postProcessing_v2)
Function/S DoLoadMultipleFiles(type)							//DoLoadMultipleFiles
	String type
	String message = "Select one or more files"
	String outputPaths
	String fileFilters = "Data Files (*.txt,*.dat,*.csv):.txt,.dat,.csv;"
	fileFilters += "All Files:.*;"

	Open /D /R /MULT=1 /F=fileFilters /M=message refNum
	outputPaths = S_fileName

	if (strlen(outputPaths) == 0)
		Print "Cancelled"
	else
		Variable numFilesSelected = ItemsInList(outputPaths, "\r")
		Variable i
		for(i=0; i<numFilesSelected; i+=1)
			String path = StringFromList(i, outputPaths, "\r")
			Printf "%d: %s\r", i, path
			// Load the waves and set the local variables.
			//This command is if there are only 2 waves in the imported file.
			if( stringMatch(type, "Thetas"))
				LoadWave/W/A/D/G/L={2, 0, 0, 0,2} path
			else //if type = "Angles"
				Printf "Loading Angles --Not ready yet"
			endif

			if (V_flag==0)	// No waves loaded. Perhaps user canceled.
				return "-1"
			endif

		endfor
	endif

	return outputPaths		// Will be empty if user canceled
End


//This functions initializes the waves that are to be used for later use. In this function, we will load
//multiple files with thetaX and thetaY data from .txt files created in Matlab. After loading all the files from
//a single trial type, all of the thetaX and thetaY data will be concatenated to create an Xwave and Ywave,
//respectively. These output waves will be used for further calculations in other routines.
//(This version was created in multipleFileLoad_v4)
Function InitSession()							//InitSession
	//Outputs: Xwave, Ywave, Forward, Left, Back, Right
	//Outputs: ForwardLeft, ForwardRight, BackLeft, BackRight

	//Sets data folder to root
```

```
SetDataFolder root:

//Creates a prompt for the user to input the names of the waves for the final concatenated waves
String sessionNum
Prompt sessionNum, "Enter Session Number: " //Set prompt for userInput1 param
DoPrompt "Session Number", sessionNum

if(V_Flag)
            return -1 //User canceled
endif

//Creates the Session folder name and sets the current folder to the session folder
String folderName = "Session" + sessionNum
NewDataFolder/O/S root:$folderName

//Creates a prompt for the user to input what kind of data is being loaded and appropriately load
//files into the correct waves
String dataType
String menuList = "Thetas; Angles;"
Prompt dataType, "Pick Data Type:", popup, menuList
DoPrompt "Select Loaded Data File Type", dataType

//Function that enables user to select files from the computer. Creates a new data folder for those
// files to be saved in
DoLoadMultipleFiles(dataType)
if (V_Flag)
            return -1                                                   // User canceled
endif

//Data Folder reference to the data folder we are in, after DoLoadMultipleFiles()
DFREF sessionDFR = GetDataFolderDFR()
String sessionFolder = GetDataFolder(1)

// Puts the names of specific waves in the current data folder into a List for ConcatWaves to read.
//The waves are specified by the first string in WaveList. The waves are being separated by data
//type (theta X or theta Y).
String list1x, list1y
list1x = WaveList("*_X", ";", "")
list1y= WaveList("*_Y", ";", "")

//Based on the answer chosen in the above prompt different load protocols are used
strswitch(dataType)
            case "Thetas":
                        String thetaFolder = sessionFolder + "Thetas"
                        NewDataFolder/O $thetaFolder
                        InitThetas(list1x, list1y, sessionNum)
                        break

            case "Angles":
                        String anglesFolder = sessionFolder + "Angles"
                        NewDataFolder/O $anglesFolder
                        InitAngles(list1x, list1y, sessionNum)

                        break
endswitch
```

```
		//Create Model datafolder for all the display functions to utilize
		String modelFolder = sessionFolder + "Models"
		NewDataFolder/O/S $modelFolder
		//Basic Ellipse model parameters
		String/G ebParam = "area=-1;perim=-1;pout=-1;semiA=-1;semiB=-1;eccen=-1;"
		//One Pt Ellipse model parameters
		String/G e1Param = "area=-1;perim=-1;pout=-1;semiA=-1;semiB=-1;eccen=-1;"
		//Two Pt Ellipse model parameters
		String/G e2param = "area=-1;perim=-1;pout=-1;semiA=-1;semiB=-1;eccen=-1;"
		//Convex Hull model parameters
		String/G chParam  = "area=-1;perim=-1;pout=-1;semiA=-1;semiB=-1;eccen=-1;"
		//Greek 2004 model parameters
		String/G grParam  = "area=-1;perim=-1;pout=-1;semiA=-1;semiB=-1;eccen=-1;"
End


//This function receives a list of x and y waves that were just selected by the user. The desired sets of the
//list are parsed into new lists and given to the function InitWaves where the desired waves are created and
//saved into the predetermined data folder
Function InitThetas(xList, yList, sessionNum)					//InitTheta
		String xList, yList
		String sessionNum

		//String thetaFolder = GetDataFolder(1)
		//Makes the nessary folder paths and then creates the folders
		String tFolder		= ":Thetas"
		String sFolder		= ":Thetas:SingleLeans"
		String gFolder	= ":Thetas:GroupedLeans"
		String qsFolder = ":Thetas:QuietStance"
		NewDataFolder/O $sFolder
		NewDataFolder/O $gFolder
		NewDataFolder/O $qsFolder

		//Creating the main wave that contains all of the leans from all trials
		// Puts the names of specific waves into the Theta. The waves are being separated by data type
		String trialListX = ListMatch(xList, "!q*")
		String trialListY = ListMatch(yList, "!q*")
		String name

		if (strlen(trialListX) != 0)
				name = "trial"
				InitWaves(trialListX, trialListY, name, sessionNum, tFolder)
		endif

		//Separating the x and y lists of all waves imported into the individual lean directions and then
		// creating a wave that only has those leans. Then creating waves that contain all of the same trial
		// numbers, i.e.all 8 first leans in a wave, all 8 leans from take 2 in a wave, etc. Putting them into
		// their respective folders: "Single Leans", "Grouped Lean", "Quiet Stance"

		//Getting only forward leans
		String flistx = ListMatch(xList, "f*")
		flistx =  ListMatch(flistx, "!*R*")
		flistx = ListMatch(flistx, "!*L*")
		String flisty = ListMatch(yList, "f*")
```

```
flisty =  ListMatch(flisty, "!*R*")
flisty = ListMatch(flisty, "!*L*")
if (strlen(flistx) != 0)
            name = "forward"
            InitWaves(flistx, flisty, name, sessionNum, sFolder)
endif

//Getting only backward leans
String blistx = ListMatch(xList, "b*")
blistx =  ListMatch(blistx, "!*R*")
blistx = ListMatch(blistx, "!*L*")
String blisty = ListMatch(yList, "b*")
blisty =  ListMatch(blisty, "!*R*")
blisty = ListMatch(blisty, "!*L*")
if (strlen(blistx) != 0)
            name = "back"
            InitWaves(blistx, blisty, name, sessionNum, sFolder)
endif

//Getting only forwardRight leans
String fRlistx =  ListMatch(xList, "fr*")
String fRlisty =  ListMatch(yList, "fr*")
if (strlen(fRlistx) != 0)
            name = "forwardRight"
            InitWaves(fRlistx, fRlisty, name, sessionNum, sFolder)
endif

//Getting only forwardLeft leans
String fLlistx =  ListMatch(xList, "fl*")
String fLlisty =  ListMatch(yList, "fl*")
if (strlen(fLlistx) != 0)
            name = "forwardLeft"
            InitWaves(fLlistx, fLlisty, name, sessionNum, sFolder)
endif

//Getting only backRight leans
String bRlistx =  ListMatch(xList, "br*")
String bRlisty =  ListMatch(yList, "br*")
if (strlen(bRlistx) != 0)
            name = "backRight"
            InitWaves(bRlistx, bRlisty, name, sessionNum, sFolder)
endif

//Getting only backLeft leans
String bLlistx =  ListMatch(xList, "bl*")
String bLlisty =  ListMatch(yList, "bl*")
if (strlen(bLlistx) != 0)
            name = "backLeft"
            InitWaves(bLlistx, bLlisty, name, sessionNum, sFolder)
endif

//Getting only right leans
String rlistx = ListMatch(xList, "r*")
String rlisty = ListMatch(yList, "r*")
if (strlen(rlistx) != 0)
```

```
                name = "right"
                InitWaves(rlistx, rlisty, name, sessionNum, sFolder)
        endif

        //Getting only left leans
        String llistx = ListMatch(xList, "l*")
        String llisty = ListMatch(yList, "l*")
        if (strlen(llistx) != 0)
                name = "left"
                InitWaves(llistx, llisty, name, sessionNum, sFolder)
        endif

        //Creating list for group leans
        String gListX, gListY
        String testX = "*1_X;*2_X;*3_X"
        String testY = "*1_Y;*2_Y;*3_Y"
        String nameList = "group1;group2;group3"
        Variable limitNum = 3
        Variable i = 0
        for (i = 0; i < limitNum; i += 1)
                gListx = ListMatch(xlist, StringFromList(i, testX))
                gListy = ListMatch(ylist, StringFromList(i, testY))
                if(strlen(gListx) == 0) //Checks if list is empty before continuing
                        break
                endif
                InitWaves(gListx, gListy, StringFromList(i, nameList), sessionNum, gFolder)
        endfor

        //Getting quiet stance (qS) data
        String qSlistx = ListMatch(xList, "q*")
        String qSlisty = ListMatch(yList, "q*")

        if (strlen(qSlistx) != 0)
                name = "qs"
                InitWaves(qSlistx, qSlisty, name, sessionNum, qsFolder)
        endif

        //Killing all the individual leans trials from the Session2 folder
        KillWaves/A/Z
End


//This function takes a list of waves that need to be concatenated and with the provided names and
//session number it names the wave accordingly. Lastly, these 2 new waves (x and y) are moved into
//a data folder given by the dfpath that is supplied by the calling function.
Function InitWaves(xList, yList, baseName, sessionNum, dfpath)                //InitWaves
        String xList, yList, baseName                                         //Inputs
        String sessionNum, dfpath                                             //Inputs

        //Need to add the colon to the path or it thinks it should rename the wave in MoveWaves to the
        //last folder name
        dfpath = dfpath +":"

        Variable count = 0
        count = PntsInWaveList(xList)
        //Defining wave name
        String xName = baseName + "_X" + sessionNum
        String yName = baseName + "_Y" + sessionNum
        //Makes waves
        Make/O/D/N=(count) $xName, $yName
        ConcatWaves(xList, $xName)
        ConcatWaves(yList, $yName)
        //Moving waves into the desired data folder given dfpath
        MoveWave $xName, $dfpath
        MoveWave $yName, $dfpath
End


//Initializes the angles data sets for all types of leans and quiet stance. They will be saved into their
//respective folders.
Function InitAngles(xList, yList, sessionNum)                                    //InitAngles
        String xList, yList, sessionNum

        //String thetaFolder = GetDataFolder(1)
        //Makes the necessary folder paths and then creates the folders
        String aFolder      = ":Thetas"
        String ankleFolder          = ":Thetas:SingleLeans"
        String hipFolder   = ":Thetas:GroupedLeans"
        String kneeFolder = ":Thetas:QuietStance"
        NewDataFolder/O $ankleFolder
        NewDataFolder/O $hipFolder
        NewDataFolder/O $kneeFolder
End


//This function creates a list of the trials that have been Loaded into the program. The user can select
//which trial data set they want to choose. The trial name chosen, will be returned as a string to be
//used to create different displays
Function/S SelectDataSet(folder)                                                  //SelectDataSet
        DFREF folder
        SetDataFolder folder

        //Counts the number of different trials that are in the given folder
        String list = WaveList("*_X*", ";","DP:1")
        Variable count = ItemsInList(list)

        //Creates a list of trials based on how many trials are found in count (above)
        Variable i
        String temp
        String groupList = ""
        for (i = 0; i < count; i+=1)
                temp = "Group" + num2str(i+1) +";"
                groupList = groupList + temp
        endfor

        //Dialog box for selecting which trial is to be used
        String group
        Prompt group, "Select Data Set:", popup groupList
        DoPrompt "Graph Selected Data", group
```

```
		if(V_flag)
			return ""						//user cancelled
		endif
		//Return answer to function caller
		return group								//Output
End


//This function allows the user to select what data folder to pull waves from in the display functions
Function/S SelectDataFolder()							//SelectDataFolder
		//Gets the names of all the folders in root
		SetDataFolder root:
		DFREF dfr = GetDataFolderDFR() //dfr = data folder reference

		//Defining variables to create list of folders
		String objName
		String folderList = ""
		Variable index = 0

		do
			objName = GetIndexedObjNameDFR(dfr, 4, index)
			if (strlen(objName) == 0 ) //If returns a null string
				break
			endif
			//Creates the folder list
			folderList = folderList + objName + ";"
			index +=1
		while(1)

		//Remove the folder "Packages" from user folder choices
		folderList = RemoveFromList("Packages", folderList)
		//Sorts the folder list into alphanumeric order
		folderList = SortList(folderList)
		//Dialog box for selecting which folder is to be used
		String folderName
		Prompt folderName, "Select Folder:", popup folderList
		DoPrompt "Retrieve Data from Folder", folderName

		if(V_flag)
			return "" //user cancelled
		endif
		//Return answer to function caller
		return folderName
End

//This function is used in all of the main functions. It extracts specific strings in each function that hold the
//desired parameters for writing to a txt file
Function WriteInfoToFile()							//WriteInforToFile
//User chooses data folder to use
		String folder
		folder = SelectDataFolder()			//note that the name of the folder is the session name
		DFREF mDFR = GetModelsDFR(folder)
		String path = GetDataFolder(1)
		SetDataFolder mDFR
```

```
	//Set up for making parameter table
	Variable i = 0, j = 0, k= 0
	String var, param, paramPath
	Make/O/T/N=(6,7) ModelParams
	String modelList = StringList("*", ";")
	Make/O/T/N=5 modelNames = {"Basic Ellipse","1pt Ellipse","2pt Ellipse","Convex Hull","2004
	Greek"}

	Make/O/T/N=6 keywords = {"area", "perim", "pout", "semiA", "semiB", "eccen"}

	//Making table labels
	ModelParams[0][0] = folder
	for (k=0; k<numPnts(keywords); k+=1)
		ModelParams[0][k+1] = keywords[k]
	endfor
	for (k=0; k<numPnts(modelNames); k+=1)
		ModelParams[k+1][0] =  modelNames[k]
	endfor

	//Filling in data into table
	for (i=0; i < ItemsInList(modelList); i+=1)
		var = StringFromList(i, modelList)
		paramPath = path + var
		//Global key string with a single model's parameters
		SVAR model = $paramPath

		for (j=0; j<numPnts(keywords); j+=1)
			param = StringByKey( keywords[j], model, "=", ";")
			ModelParams[i+1][j+1] = param
		endfor
	endfor
End
```

```
//
//
//Display Functions
//
//

//This function creates a graph based on the data from the user selected folder and creates a
//Basic Ellipse to fit the data.
Function DisplayEllipseFit()                                              //DisplayEllipseFit
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(folder)
        DFREF tDFR                          = GetThetasDFR(folder)
        DFREF mDFR              = GetModelsDFR(folder)
        SetDataFolder mDFR

        //Setting up the wave names that will be used to check if they already exist or to build waves if
        //they do not exist. Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum
        //name = "trial"; number = "#" based on user choice

        //Create the wave names
        String XYtrialList, inY, inX //used later in the if-else statement
        String ellipseWaveX = "basicElliX" + sessionNum
        String ellipseWaveY = "basicElliY" + sessionNum
        String trendWave = "basicElliTrend" + sessionNum
        String outPtsWave = "basicElliOutPts" + sessionNum
        String waveCheck = ellipseWaveX+";" + ellipseWaveY+";" +trendWave + ";" + outPtsWave + ";"

        //The current folder is still :Models. This creates
        String objList, str
        str = "*basic*"
        objList = GetWavesInFolder(mDFR, str)
        print objList

        //Sets the folder to Thetas in order to get the trials waves
        SetDataFolder tDFR
        //From the folder selected the corresponding X and Y waves is used
        XYtrialList = waveList("trial*", ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Check to see if the waves for creating this display already exist so you don't have to
        //recalculate everything.
        if(stringMatch(waveCheck, objList) == 1)
                Printf "say Hi!!"
        else  //Make the waves needed for ellipseFit
                Make/O/D/N=501 $ellipseWaveX, $ellipseWaveY     //holds the ellipse points
                Make/O/D/N=(2,2) $trendWave                     //will contain the angled trend line
                Make/O/D/N=(numpnts($inY),3) $outPtsWave
```

```
				//Main Function
				ellipseFit($inX, $inY, $ellipseWaveX, $ellipseWaveY, $trendWave, $outPtsWave)

				//Moving the generated waves into the Model folder
				//Setting the folder to session and then adding a new folder and getting that pathway
				SetDataFolder mDFR
				String mFolder =  GetDataFolder(1)

				//Setting the path back to theta folder where all the waves are located so MoveWave can
				//access those waves to move them to the desired data folder given dfpath
				SetDataFolder tDFR
				MoveWave $ellipseWaveX, $mFolder
				MoveWave $ellipseWaveY, $mFolder
				MoveWave $trendWave, $mFolder
				MoveWave $outPtsWave, $mFolder

		endif

		//Start Plotting
		SetDataFolder root:
		String wn = "BasicEllipseFit_" + folder		//name of the graph window
		//Laying out the traces
		Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX
		//The linear regression line calculated in CurveFit
		AppendToGraph/W=$wn mDFR:$trendWave[][1] vs mDFR:$trendWave[][0]
		ModifyGraph/W=$wn rgb($trendWave)=(0,12800,52224), lsize($trendWave)=1.5; DelayUpdate
		AppendToGraph/W=$wn mDFR:$ellipseWaveY vs mDFR:$ellipseWaveX;DelayUpdate
		ModifyGraph/W=$wn rgb($ellipseWaveY)=(0,0,0),lsize($ellipseWaveY)=1.5;DelayUpdate
		//Axes modification
		SetAxis/W=$wn left -0.08,0.08;DelayUpdate
		SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
		ModifyGraph axThick=1.5 //Axes thickness
		ModifyGraph/W=$wn standoff=0;DelayUpdate
		ModifyGraph/W=$wn zero=1;DelayUpdate
		ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
		ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
		ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
		//Labels
		Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
		Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
		TextBox/W=$wn/C/N=text0/D=1.5 "\\Z18Basic Ellipse Fit"
		TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate
		Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) FP Data\r\\s(#1) Data Trend Line\r"
		AppendText/W=$wn "\\Z16\\s(#2) Fitted Ellipse";DelayUpdate
		Legend/W=$wn/C/N=text1/J/A=RT/X=-1.88/Y=0.33;DelayUpdate

End
```

```
//This function creates a graph based on the data from the user selected folder and uses the
//OnePtEllipse to fit the data.
Function Display1PtEllipse()                                            //Display1PtEllipse
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()                     //note that the name of the folder is the session name

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(folder)
        DFREF tDFR              = GetThetasDFR(folder)
        DFREF mDFR              = GetModelsDFR(folder)
        SetDataFolder mDFR

        //Setting up the wave names that will be used to check if they already exist or to build
        //waves if they do not exist.
        //Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum
        //Create the wave names
        String XYtrialList, inY, inX //used later in the if-else statement
        String ellipseWaveX = "onePtFitX" + sessionNum
        String ellipseWaveY = "onePtFitY" + sessionNum
        String trendWave = "onePtTrend" + sessionNum
        String outPtsWave = "onePtFitOutPts" + sessionNum
        String waveCheck =ellipseWaveX + ";" + ellipseWaveY+ ";" + trendWave + ";" +outPtsWave+";"

        //The current folder is still :Models. This creates
        String objList, str
        str = "*one*"
        objList = GetWavesInFolder(mDFR, str)

        //Sets the folder to Thetas in order to get the trials waves
        SetDataFolder tDFR
        //From the folder selected the corresponding X and Y waves is used
        XYtrialList = waveList("trial*", ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Check to see if the waves for creating this display already exist so you don't have to
        //recalculate everything.
        if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
                Printf "say Hi!!"
        else //Makes the waves from above using the desired Main Function and then graphs the results

                //Make the waves needed for ellipseFit
                Make/O/D/N=501 $ellipseWaveX, $ellipseWaveY     //holds the ellipse points
                Make/O/D/N=(2,2) $trendWave                     //will contain the angled trend line
                Make/O/D/N=(numpnts($inY),3) $outPtsWave

                //Main Function
                OnePtEllipse($inX, $inY, $trendWave, $ellipseWaveX, $ellipseWaveY, $outPtsWave)

                //Moving the generated waves into the Model folder
```

```
				//Setting the folder to session and then adding a new folder and getting that pathway
				SetDataFolder mDFR
				String mFolder =  GetDataFolder(1)

				//Setting the path back to theta folder where all the waves are located so MoveWave can
				//access those waves to move them to the desired data folder given dfpath
				SetDataFolder tDFR
				MoveWave $ellipseWaveX, $mFolder
				MoveWave $ellipseWaveY, $mFolder
				MoveWave $trendWave, $mFolder
				MoveWave $outPtsWave, $mFolder
		endif

		//Start Plotting
		SetDataFolder root:
		String wn = "OnePtEllipseFit_" + folder				//name of the graph window
		//Adding traces
		Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX
		AppendToGraph/W=$wn mDFR:$trendWave[][1] vs mDFR:$trendWave[][0];DelayUpdate
		ModifyGraph/W=$wn rgb($trendWave)=(0,12800,52224), lsize($trendWave)=1.5; DelayUpdate
		AppendToGraph/W=$wn mDFR:$ellipseWaveY vs mDFR:$ellipseWaveX;DelayUpdate
		ModifyGraph/W=$wn rgb($ellipseWaveY)=(0,0,0),lsize($ellipseWaveY)=1.5;DelayUpdate
		//Axes modification
		SetAxis/W=$wn left -0.08,0.08;DelayUpdate
		SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
		ModifyGraph axThick=1.5 //Axes thickness
		ModifyGraph/W=$wn standoff=0;DelayUpdate
		ModifyGraph/W=$wn zero=1;DelayUpdate
		ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
		ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
		ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
		//Labels
		Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
		Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
		TextBox/W=$wn/C/N=text0/D=1.5 "\\Z18One Pt Ellipse Fit"
		TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate
		Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) FP Data\r\\s(#1) Data Trend Line\r"
		AppendText/W=$wn "\\Z16\\s(#2) Fitted Ellipse";DelayUpdate
		Legend/W=$wn/C/N=text1/J/A=RT/X=-1.88/Y=0.33;DelayUpdate

End


//This function creates a graph based on the data from the user selected folder and uses
//FitConvexHull to fit the data.
Function Display2PtEllipse()							//Display2PtEllipse
		//User chooses data folder to use
		String folder
		folder = SelectDataFolder()				//note that the name of the folder is the session name

		//Setting up all of the data folder references we will need for creating the display
		DFREF sessionDFR		= GetSessionDFR(folder)
		DFREF tDFR			= GetThetasDFR(folder)
		DFREF mDFR			= GetModelsDFR(folder)
		SetDataFolder mDFR
		//Setting up the wave names that will be used to check if they already exist or to build
		//waves if they do not exist.
		//Splitting the string to extract the trial number
		String name, sessionNum
		String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
		SplitString/E=regExpr folder, name,

		//Create the wave names
		String XYtrialList, inY, inX //used later in the if-else statement
		String ellipseWaveX = "twoPtFitX" + sessionNum
		String ellipseWaveY = "twoPtFitY" + sessionNum
		String trendWave = "twoPtTrend" + sessionNum
		String outPtsWave = "twoPtFitOutPts" + sessionNum
		String waveCheck=ellipseWaveX + ";" +ellipseWaveY +";" +trendWave + ";" + outPtsWave + ";"

		//The current folder is still :Models. This creates
		String objList, str
		str = "*two*"
		objList = GetWavesInFolder(mDFR, str)

		//Sets the folder to Thetas in order to get the trials waves
		SetDataFolder tDFR
		//From the folder selected the corresponding X and Y waves is used
		XYtrialList = waveList("trial*", ";", "DP:1")
		inX = StringFromList(0, XYtrialList)
		inY = StringFromList(1, XYtrialList)

		//Check to see if the waves for creating this display already exist so you don't have to
		//recalculate everything.
		if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
			Printf "say Hi!!"
		else //Makes the waves from above using the desired Main Function and then graphs the results
			//Make the waves needed for ellipseFit
			Make/O/D/N=501 $ellipseWaveX, $ellipseWaveY	//holds the ellipse points
			Make/O/D/N=(2,2) $trendWave			//will contain the angled trend line
			Make/O/D/N=(numpnts($inY),3) $outPtsWave

			//Main Function
			TwoPtEllipse($inX, $inY, $trendWave, $ellipseWaveX, $ellipseWaveY, $outPtsWave)

			//Moving the generated waves into the Model folder
			//Setting the folder to session and then adding a new folder and getting that pathway
			SetDataFolder mDFR
			String mFolder =  GetDataFolder(1)

			//Setting the path back to theta folder where all the waves are located so MoveWave can
			//access those waves to move them to the desired data folder given dfpath
			SetDataFolder tDFR
			MoveWave $ellipseWaveX, $mFolder
			MoveWave $ellipseWaveY, $mFolder
			MoveWave $trendWave, $mFolder
			MoveWave $outPtsWave, $mFolder
		endif
```

```
	//Start Plotting
	SetDataFolder root:
	String wn = "TwoPtEllipseFit_" + folder				//name of the graph window
	//Traces
	Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX;DelayUpdate
	AppendToGraph/W=$wn mDFR:$trendWave[][1] vs mDFR:$trendWave[][0];DelayUpdate
	ModifyGraph/W=$wn rgb($trendWave)=(0,12800,52224), lsize($trendWave)=1.5; DelayUpdate
	AppendToGraph/W=$wn mDFR:$ellipseWaveY vs mDFR:$ellipseWaveX;DelayUpdate
	ModifyGraph/W=$wn rgb($ellipseWaveY)=(0,0,0), lsize($ellipseWaveY)=1.5; DelayUpdate
	//Axes modification
	SetAxis/W=$wn left -0.08,0.08;DelayUpdate
	SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
	ModifyGraph axThick=1.5 //Axes thickness
	ModifyGraph/W=$wn standoff=0;DelayUpdate
	ModifyGraph/W=$wn zero=1;DelayUpdate
	ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
	ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
	ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
	//Labels
	Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
	Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
	TextBox/W=$wn/C/N=text0/D=1.5 "\\Z18Two Pt Ellipse Fit"
	TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate
	Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) FP Data\r\\s(#1) Data Trend Line\r"
	AppendText/W=$wn "\\Z16\\s(#2) Fitted Ellipse";DelayUpdate
	Legend/W=$wn/C/N=text1/J/A=RT/X=-1.88/Y=0.33;DelayUpdate

End


//This function creates a graph based on the data from the user selected folder and uses the
//OnePtEllipse to fit the data.
Function DisplayConvexHullFit()						//DisplayConvexHullFit
	//User chooses data folder to use
	String folder
	folder = SelectDataFolder()			//note that the name of the folder is the session name

	//Setting up all of the data folder references we will need for creating the display
	DFREF sessionDFR		= GetSessionDFR(folder)
	DFREF tDFR				= GetThetasDFR(folder)
	DFREF mDFR		= GetModelsDFR(folder)
	SetDataFolder mDFR

	//Setting up the wave names that will be used to check if they already exist or to build
	//waves if they do not exist.
	//Splitting the string to extract the trial number
	String name, sessionNum
	String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
	SplitString/E=regExpr folder, name, sessionNum

	//Create the wave names
	String XYtrialList, inY, inX //used later in the if-else statement
	String Xhull = "Xhull" + sessionNum
	String Yhull = "Yhull" + sessionNum
```

```
        String waveCheck = Xhull + ";" + Yhull + ";"

        //The current folder is still :Models. This creates
        String objList, str
        str = "*hull*"
        objList = GetWavesInFolder(mDFR,str)

        //Sets the folder to Thetas in order to get the trials waves
        SetDataFolder tDFR
        //From the folder selected the corresponding X and Y waves is used
        XYtrialList = waveList("trial*", ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Check to see if the waves for creating this display already exist so you don't have to
        //recalculate everything.
        if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
                Printf "say Hi!!"
        else //Makes the waves from above using the desired Main Function and then graphs the results
                //Sets the folder to Thetas in order to get the trials waves
                //Set up of waves to be passed into ConvexHull
                variable length = numPnts($inY)
                Make/O/D/N=(length) $Xhull, $Yhull        //will contain the ConvexHull points

                //Main Function
                FitConvexHull($inX, $inY, $XHull, $Yhull)

                //Moving the generated waves into the Model folder
                //Setting the folder to session and then adding a new folder and getting that pathway
                SetDataFolder mDFR
                String mFolder =  GetDataFolder(1)

                //Setting the path back to theta folder where all the waves are located so MoveWave can
                //access those waves to move them to the desired data folder given dfpath
                SetDataFolder tDFR
                MoveWave $XHull, $mFolder
                MoveWave $Yhull, $mFolder

        endif

        //Start Plotting
        SetDataFolder root:
        String wn = "ConvexHullFit_" + folder                        //name of the graph window

        //Traces
        Display/K=1/N=$wn mDFR:$YHull vs mDFR:$XHull;DelayUpdate
        AppendToGraph/W=$wn tDFR:$inY vs tDFR:$inX;DelayUpdate
        ModifyGraph/W=$wn rgb($YHull)=(0,0,0);DelayUpdate
        ModifyGraph/W=$wn lsize($YHull)=1.5;DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
```

```
	ModifyGraph/W=$wn zero=0;DelayUpdate
	ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
	ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
	ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
	//Labels
	Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
	Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
	TextBox/W=$wn/C/N=text0/D=1.5 "\\Z18Convex Hull"
	TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate

End


Function DisplayGreekMethod()						//DisplayGreekMethod
	//User chooses data folder to use
	String folder
	folder = SelectDataFolder()				//note that the name of the folder is the session name

	//Setting up all of the data folder references we will need for creating the display
	DFREF sessionDFR		= GetSessionDFR(folder)
	DFREF tDFR				= GetThetasDFR(folder)
	DFREF mDFR			= GetModelsDFR(folder)
	SetDataFolder mDFR

	//Setting up the wave names that will be used to check if they already exist or to build
	//waves if they do not exist.
	//Splitting the string to extract the trial number
	String name, sessionNum
	String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
	SplitString/E=regExpr folder, name, sessionNum

	//Create the wave names
	String XYtrialList, inY, inX //used later in the if-else statement
	String ellipseWaveX = "greekCritX" + sessionNum
	String ellipseWaveY = "greekCritY" + sessionNum
	String waveCheck = ellipseWaveX + ";" + ellipseWaveY + ";"

	//The current folder is still :Models. This creates
	String objList, str
	str = "*greek*"
	objList = GetWavesInFolder(mDFR, str)

	//Sets the folder to Thetas in order to get the trials waves
	SetDataFolder tDFR
	//From the folder selected the corresponding X and Y waves is used
	XYtrialList = waveList("trial*", ";", "DP:1")
	inX = StringFromList(0, XYtrialList)
	inY = StringFromList(1, XYtrialList)

	//Check to see if the waves for creating this display already exist so you don't have to
	//recalculate everything.
	if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
		Printf "say Hi!!"
	else //Makes the waves from above using the desired Main Function and then graphs the results
```

```
                Variable leng, doubleLeng
                leng = numPnts($inX)
                doubleLeng = 2*leng
                Make/O/D/N=(doubleLeng) $ellipseWaveX, $ellipseWaveY //holds the ellipse points

                //Main Function
                GreekMethod($inX, $inY, $ellipseWaveX, $ellipseWaveY)

                //Moving the generated waves into the Model folder
                //Setting the folder to session and then adding a new folder and getting that pathway
                SetDataFolder mDFR
                String mFolder =  GetDataFolder(1)

                //Setting the path back to theta folder where all the waves are located so MoveWave can
                //access those waves to move them to the desired data folder given dfpath
                SetDataFolder tDFR
                MoveWave $ellipseWaveX, $mFolder
                MoveWave $ellipseWaveY, $mFolder
        endif

        //Start Plotting
        SetDataFolder root:
        String wn = "GreekPlot_" + folder                               //name of the graph window
        //Traces
        Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX;DelayUpdate
        AppendToGraph/W=$wn mDFR:$ellipseWaveY vs mDFR:$ellipseWaveX;DelayUpdate
        ModifyGraph/W=$wn rgb($ellipseWaveY)=(0,0,0),lsize($ellipseWaveY)=1.5;DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
        ModifyGraph/W=$wn zero=0;DelayUpdate
        ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
        ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
        ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
        //Labels
        Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
        Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
        TextBox/W=$wn/C/N=text0/D=1.5 "\\Z182004  Zakynthinaki Method"
        TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate
        Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) FP Data\r\\s(#1) 2004 Basin\r"
        AppendText/W=$wn "\\s              Boundary";DelayUpdate
        Legend/W=$wn/C/N=text1/J/A=RT/X=-1.88/Y=0.33;DelayUpdate

End
```

```
Function DisplayGreekByGroup()                                          //DisplayGreekByGroup
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()                      //note that the name of the folder is the session name

        if(strlen(folder)==0)
                return -1  //User Cancelled
        endif
        //Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR         = GetSessionDFR(folder)
        DFREF gDFR               = GetGroupedLeansDFR(folder)
        DFREF mDFR               = GetModelsDFR(folder)

        //Gets the folder where the data is extra
        String group = SelectDataSet(gDFR)

        if(strlen(group)==0)
                return -1   //User Cancelled
        endif

        String groupMatchStr = lowerStr(group) + "*"
        Printf "str = %s\r", groupMatchStr
        //Sets the folder to GroupedLeans in order to get the desired waves
        SetDataFolder gDFR
        //From the folder selected the corresponding X and Y waves is used
        String XYtrialList, inX, inY
        XYtrialList = waveList(groupMatchStr, ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        SetDataFolder mDFR
        //Create the wave names
        String ellipseWaveX = "greekCritX" + sessionNum + group
        String ellipseWaveY = "greekCritY" + sessionNum + group
        String waveCheck = ellipseWaveX + ";" + ellipseWaveY + ";"

        //The current folder is still :Models. This creates
        String objList, str
        str = "*greek*"
        objList = GetWavesInFolder(mDFR, str)

        //Check to see if the waves for creating this display already exist so you don't have to
        //recalculate everything.
        if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
                Printf "say Hi!!"
        else //Makes the waves from above using the desired Main Function and then graphs the results
                Variable leng, doubleLeng
                leng = numPnts(gDFR:$inX)
                doubleLeng = 2*leng
```

```
			Make/O/D/N=(doubleLeng) $ellipseWaveX, $ellipseWaveY //holds the ellipse points
			//Main Function
			GreekMethod(gDFR:$inX, gDFR:$inY, $ellipseWaveX, $ellipseWaveY)
		endif

		//Start Plotting
		SetDataFolder root:
		String wn = "GreekPlot" + sessionNum + "_" + group		//name of the graph window
		//Traces
		Display/K=1/N=$wn gDFR:$inY vs gDFR:$inX;DelayUpdate
		AppendToGraph/W=$wn mDFR:$ellipseWaveY vs mDFR:$ellipseWaveX;DelayUpdate
		ModifyGraph/W=$wn rgb($ellipseWaveY)=(0,0,0),lsize($ellipseWaveY)=1.5;DelayUpdate
		//Axes modification
		SetAxis/W=$wn left -0.08,0.08;DelayUpdate
		SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
		ModifyGraph axThick=1.5 //Axes thickness
		ModifyGraph/W=$wn standoff=0;DelayUpdate
		ModifyGraph/W=$wn zero=0;DelayUpdate
		ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
		ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
		ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
		//Labels
		Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
		Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
		TextBox/W=$wn/C/N=text0/D=1.5 "\\Z182004  Zakynthinaki Method"
		TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33;DelayUpdate
		Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) FP Data\r\\s(#1) 2004 Basin\r"
		AppendText/W=$wn "\\s			Boundary";DelayUpdate
		Legend/W=$wn/C/N=text1/J/A=RT/X=-1.88/Y=0.33;DelayUpdate
End


Function DisplayInfoDensity()							//DisplayInfoDensity()
		//User chooses data folder to use
		String folder
		folder = SelectDataFolder()				//note that the name of the folder is the session name

		//Setting up all of the data folder references we will need for creating the display
		DFREF sessionDFR		= GetSessionDFR(folder)
		DFREF tDFR				= GetThetasDFR(folder)
		DFREF mDFR		= GetModelsDFR(folder)
		SetDataFolder mDFR

		//Setting up the wave names that will be used to check if they already exist or to build
		//waves if they do not exist.
		//Splitting the string to extract the trial number
		String name, sessionNum
		String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
		SplitString/E=regExpr folder, name, sessionNum

		//Create the wave names
		String XYtrialList, inY, inX //used later in the if-else statement
		String densityMatrix = "densityMatrix" + sessionNum
		String logDensity		= "logDensity" + sessionNum
		String waveCheck = densityMatrix + ";" + logDensity + ";"
```

```
	//The current folder is still :Models. This creates
	String objList, str
	str = "*sity*"
	objList = GetWavesInFolder(mDFR,str)

	//Sets the folder to Thetas in order to get the trials waves
	SetDataFolder tDFR
	//From the folder selected the corresponding X and Y waves is used
	XYtrialList = waveList("trial*", ";", "DP:1")
	inX = StringFromList(0, XYtrialList)
	inY = StringFromList(1, XYtrialList)

	//Check to see if the waves for creating this display already exist so you don't have to
	//recalculate everything.
	if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
		Printf "say Hi!!"
	else //Makes the waves from above using the desired Main Function and then graphs the results
		//Folder is already set to Thetas in order to get the trials waves
		//Set variables and waves to pass into boxCounting
		Variable numBoxes = 100 // number of boxes in the r & c to make an NxN matrix
		Make/O/N=(numBoxes,numBoxes) $densityMatrix
		//In order to visualize both the very dense and minimally dense boxes, we log10 the
matrix.
		Make/O/N=(numBoxes,numBoxes) $logDensity

		//Main Function
		boxCounting($inX, $inY, numBoxes, $densityMatrix, $logDensity)

		//Moving the generated waves into the Model folder
		//Setting the folder to session and then adding a new folder and getting that pathway
		SetDataFolder mDFR
		String mFolder =  GetDataFolder(1)

		//Setting the path back to theta folder where all the waves are located so MoveWave can
		//access those waves to move them to the desired data folder given dfpath
		SetDataFolder tDFR
		MoveWave $densityMatrix, $mFolder
		MoveWave $logDensity, $mFolder
	endif

	//Start Plotting
	SetDataFolder root:
	String wn = "Data_Log_Density_" + folder				//name of the graph window

	Display/K=1/N=$wn;AppendMatrixContour/W=$wn mDFR:$logDensity;DelayUpdate
	ModifyContour/W=$wn $logDensity ctabLines={*,*,Classification,0},labels=0;DelayUpdate
	ColorScale/W=$wn/C/N=text0/E  ctab={0,100,Classification,0};DelayUpdate
	ModifyGraph/W=$wn standoff=0;DelayUpdate
	ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
	//Go to Graph --> Packages --> Fill between Contours --> Setting: continous, classification,
	//log scale colors
End
```

```
Function DisplaySingleGroupInfoDensity()                    //DisplaySingleGroupInfoDensity
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()               //note that the name of the folder is the session name

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(folder)
        DFREF gDFR              = GetGroupedLeansDFR(folder)

        //Gets the folder where the data is extra
        String group = SelectDataSet(gDFR)

        //Setting up the wave names that will be used to check if they already exist or to build
        //waves if they do not exist.
        //Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum //name = "trial"; number = "#" based on user
choice

        //The current folder is still :Models. This is the set up for the if statement check below
        String objList, str
        str = "*sity*"
        objList = GetWavesInFolder(mDFR,str)

        //Create the wave names
        String XYtrialList, inY, inX //used later in the if-else statement
        String densityMatrix = "densityMatrix" + sessionNum + "_" + group
        String logDensity              = "logDensity" + sessionNum + "_" + group
        String waveCheck = densityMatrix + ";" + logDensity + ";"

        String groupMatchStr = lowerStr(group) + "*"
        Printf "str = %s\r", groupMatchStr
        //Sets the folder to GroupedLeans in order to get the desired waves
        SetDataFolder gDFR
        //From the folder selected the corresponding X and Y waves is used
        XYtrialList = waveList(groupMatchStr, ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Check to see if the waves for creating this display already exist so you don't have to
        //recalculate everything.
        if(stringMatch(waveCheck, objList) == 1) // these waves already exist break to straight to
graphing
                Printf "Waves already exist, graphing now ..."
        else //Makes the waves from above using the desired Main Function and then graphs the results
                //Folder is already set to Thetas in order to get the trials waves

                //Set variables and waves to pass into boxCounting
                Variable numBoxes = 100 // number of boxes in the rows & cols to make an NxN matrix
                Make/O/N=(numBoxes,numBoxes) $densityMatrix
                //In order to visualize both the very dense and minimally dense boxes, we log10 the
matrix.
                Make/O/N=(numBoxes,numBoxes) $logDensity
                //Main Function
```

```
		boxCounting($inX, $inY, numBoxes, $densityMatrix, $logDensity)
	endif

	//Start Plotting
	SetDataFolder root:
	String wn = "Data_LogDensity_" + group				//name of the graph window

	Display/K=1/N=$wn;AppendMatrixContour/W=$wn gDFR:$logDensity;DelayUpdate
	ModifyContour/W=$wn $logDensity ctabLines={*,*,Classification,0},labels=0;DelayUpdate
	ColorScale/W=$wn/C/N=text0/E  ctab={0,100,Classification,0};DelayUpdate
	ModifyGraph/W=$wn standoff=0;DelayUpdate
	ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
	//Go to Graph --> Packages --> Fill between Contours --> Setting: continous, classification,
	// log scale colors
End


//Function that will display all the waves based on the lean type with different colors
Function DisplayPhaseSpace()							//DisplayPhaseSpace
	//User chooses data folder to use
	String folder
	folder = SelectDataFolder()			//note that the name of the folder is the session name

	//Setting up all of the data folder references we will need for creating the display
	DFREF sessionDFR		= GetSessionDFR(folder)
	DFREF tDFR				= GetThetasDFR(folder)
	//Sets the folder to Thetas in order to get the trials waves
	SetDataFolder tDFR

	//From the folder selected the corresponding X and Y waves is used
	String XYtrialList, inX, inY
	XYtrialList = waveList("trial*", ";", "DP:1")
	inX = StringFromList(0, XYtrialList)
	inY = StringFromList(1, XYtrialList)

	//Start Plotting
	SetDataFolder root:
	String wn = "PhaseSpace_" + folder				//name of the graph window
	//Traces
	Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX
	//Axes modification
	SetAxis/W=$wn left -0.08,0.08;DelayUpdate
	SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
	ModifyGraph axThick=1.5 //Axes thickness
	ModifyGraph/W=$wn standoff=0;DelayUpdate
	ModifyGraph/W=$wn zero=0;DelayUpdate
	ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
	ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
	ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
	//Labels
	TextBox/W=$wn/C/N=text0/D=1.5 /A=LT/X=0.37/Y=0.33 "\\Z18Raw Data"
	Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
	Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
End
```

```
Function DisplaySingleGroupPhaseSpace()                    //DisplaySingleGroupPhaseSpace
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()                //note that the name of the folder is the session name

        //Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(folder)
        DFREF gDFR              = GetGroupedLeansDFR(folder)
        //Gets the folder where the data is extra
        String group = SelectDataSet(gDFR)

        String groupMatchStr = lowerStr(group) + "*"
        Printf "str = %s\r", groupMatchStr
        //Sets the folder to GroupedLeans in order to get the desired waves
        SetDataFolder gDFR
        //From the folder selected the corresponding X and Y waves is used
        String XYtrialList, inX, inY
        XYtrialList = waveList(groupMatchStr, ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Start Plotting
        SetDataFolder root:
        String wn = "PhaseSpace" + sessionNum + "_" + group          //name of the graph window
        //Traces
        Display/K=1/N=$wn gDFR:$inY vs gDFR:$inX;DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
        ModifyGraph/W=$wn zero=0;DelayUpdate
        ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
        ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
        ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
        //Labels
        TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33 "//Z18Raw Data";DelayUpdate
        Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
        Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate

End


Function DisplayLeanDirections()                                         //DisplayLeanDirections
        //User chooses data folder to use
        String folder
        folder = SelectDataFolder()                //note that the name of the folder is the session name

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(folder)
        DFREF slDFR                          = GetSingleLeansDFR(folder)
        SetDataFolder slDFR

        //Splitting the string to extract the trial number
        String name, sessionNum
        String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
        SplitString/E=regExpr folder, name, sessionNum

        String xStr = "*_X" + sessionNum + "*"
        String yStr = "*_Y" + sessionNum + "*"

        //Create the wave names
        String XdirectionList, YdirectionList
        String fX, bX, fRX, fLX,  bRX, bLX, rX, lX
        String fY, bY, fRY, fLY,  bRY, bLY, rY, lY
        XdirectionList = waveList(xStr, ";", "DP:1")
        YdirectionList = waveList(yStr, ";", "DP:1")
        fX = StringFromList(0, XdirectionList)
        fY = StringFromList(0, YdirectionList)
        bX = StringFromList(1, XdirectionList)
        bY = StringFromList(1, YdirectionList)
        fRX = StringFromList(2, XdirectionList)
        fRY = StringFromList(2, YdirectionList)
        fLX = StringFromList(3, XdirectionList)
        fLY = StringFromList(3, YdirectionList)
        bRX= StringFromList(4, XdirectionList)
        bRY = StringFromList(4, YdirectionList)
        bLX = StringFromList(5, XdirectionList)
        bLY = StringFromList(5, YdirectionList)
        rX = StringFromList(6, XdirectionList)
        rY = StringFromList(6, YdirectionList)
        lX = StringFromList(7, XdirectionList)
        lY = StringFromList(7, YdirectionList)
        //Start Plotting
        SetDataFolder root:
        String wn = "LeanDirection_" + folder                    //name of the graph window
        //Traces
        Display/K=1/N=$wn slDFR:$fY vs slDFR:$fX;DelayUpdate;
        AppendToGraph/W=$wn slDFR:$bY vs slDFR:$bX;DelayUpdate
        AppendToGraph/W=$wn slDFR:$fRY vs slDFR:$fRX; DelayUpdate
        AppendToGraph/W=$wn slDFR:$fLY vs slDFR:$fLX; DelayUpdate
        AppendToGraph/W=$wn slDFR:$bRY vs slDFR:$bRX; DelayUpdate
        AppendToGraph/W=$wn slDFR:$bLY vs slDFR:$bLX; DelayUpdate
        AppendToGraph/W=$wn slDFR:$rY vs slDFR:$rX; DelayUpdate
        AppendToGraph/W=$wn slDFR:$lY vs slDFR:$lX; DelayUpdate
        ModifyGraph/W=$wn rgb($fY)=(0,0,0),rgb($bY)=(39168,39168,39168);DelayUpdate
        ModifyGraph/W=$wn rgb($fRY)=(65280,32768,58880);DelayUpdate
        ModifyGraph/W=$wn rgb($fLY)=(0,43520,65280),rgb($rY)=(0,26112,13056);DelayUpdate
        ModifyGraph/W=$wn rgb($lY)=(32768,65280,0),rgb($bLY)=(0,0,52224);DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
        ModifyGraph/W=$wn zero=0;DelayUpdate
```

```
		ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
		ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
		ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
		//Labels
		Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
		Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
		Legend/W=$wn/C/N=text0/J/D=1.5/A=LC/E "\\Z16\\s(#0) Forward\r\\s(#1) Back";DelayUpdate
		AppendText/W=$wn "\\Z16\\s(#6) Right\r\\s(#7) Left\r\\s(#2) Forward-Right";DelayUpdate
		AppendText/W=$wn "\\Z16\\s(#3) Forward-Left\r\\s(#4) Back-Right";DelayUpdate
		AppendText/W=$wn "\\Z16\\s(#5) Back-Left\r";DelayUpdate

End


Function DisplayQuietStance()								//DisplayQuietStance
		//User chooses data folder to use
		String session = SelectDataFolder()		//note that the name of the folder is the session name

		//Setting up all of the data folder references we will need for creating the display
		DFREF sessionDFR	= GetSessionDFR(session)
		DFREF qsDFR		= GetQuietStanceDFR(session)

		//Sets the folder to QuietStance in order to get the trials waves
		SetDataFolder qsDFR
		//From the folder selected the corresponding X and Y waves is used
		String XYtrialList, inX, inY
		XYtrialList = waveList("qs*", ";", "DP:1")
		inX = StringFromList(0, XYtrialList)
		inY = StringFromList(1, XYtrialList)

		//Start Plotting
		SetDataFolder root:
		String wn = "QuietStance_" + session			//name of the graph window
		//Traces
		Display/K=1/N=$wn qsDFR:$inY vs qsDFR:$inX;DelayUpdate
		//Axes modification
		SetAxis/W=$wn left -0.08,0.08;DelayUpdate
		SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
		ModifyGraph axThick=1.5 //Axes thickness
		ModifyGraph/W=$wn standoff=0;DelayUpdate
		ModifyGraph/W=$wn zero=1;DelayUpdate
		ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
		ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
		ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
		//Labels
		Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
		Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
		TextBox/W=$wn/C/N=text0/D=1.5 "\\Z18Quiet Stance Data"

End
```

```
Function DisplayLeansAndQS()                                              //DisplayLeansAndQS
        //User chooses data folder to use
        String session = SelectDataFolder()          //note that the name of the folder is the session name

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR         = GetSessionDFR(session)
        DFREF qsDFR              = GetQuietStanceDFR(session)
        DFREF tDFR               = GetThetasDFR(session)

        //Sets the folder to QuietStance in order to get the trials waves
        SetDataFolder tDFR
        //From the folder selected the corresponding X and Y waves is used
        String XYtrialList, inX, inY
        XYtrialList = waveList("trial*", ";", "DP:1")
        inX = StringFromList(0, XYtrialList)
        inY = StringFromList(1, XYtrialList)

        //Sets the folder to QuietStance in order to get the trials waves
        SetDataFolder qsDFR
        //From the folder selected the corresponding X and Y waves is used
        String QStrialList, qsX, qsY
        QStrialList = waveList("qs*", ";", "DP:1")
        qsX = StringFromList(0, QStrialList)
        qsY = StringFromList(1, QStrialList)

        //Start Plotting
        SetDataFolder root:
        String wn = "Leans_QuietStance_" + session //name of the graph window
        //Traces
        Display/K=1/N=$wn tDFR:$inY vs tDFR:$inX;DelayUpdate
        AppendToGraph/W=$wn qsDFR:$qsY vs qsDFR:$qsX;DelayUpdate
        ModifyGraph rgb($qsY)=(0,0,0);DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
        ModifyGraph/W=$wn zero=0;DelayUpdate
        ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
        ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
        ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
        //Labels
        TextBox/W=$wn/C/N=text0/A=LT/X=0.37/Y=0.33 "\\Z14Leans & Quiet Stance";DelayUpdate
        Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
        Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
        Legend/W=$wn/C/N=text1/J/D=1.5 "\\Z16\\s(#0) Leans\r\\s(#1) Quiet Stance";DelayUpdate

End
```

```
Function DisplayGroupedLeans()                                  //DisplayGroupedLeans
        //User chooses data folder to use
        String session = SelectDataFolder()        //note that the name of the folder is the session name

        //Setting up all of the data folder references we will need for creating the display
        DFREF sessionDFR        = GetSessionDFR(session)
        DFREF gDFR                      = GetGroupedLeansDFR(session)
        //Sets the folder to Thetas in order to get the trials waves
        SetDataFolder gDFR
        //From the folder selected the corresponding X and Y waves is used
        //okay to only have 3 groups becuase it is limited to only 3 groups in initThetas()
        String groupsList, x1,y1,x2,y2,x3,y3
                groupsList = waveList("group*", ";", "DP:1")
        x1 = StringFromList(0, groupsList)
        y1 = StringFromList(1, groupsList)
        x2 = StringFromList(2, groupsList)
        y2 = StringFromList(3, groupsList)
        x3 = StringFromList(4, groupsList)
        y3 = StringFromList(5, groupsList)

        //Start Plotting
        SetDataFolder root:
        String wn = "Grouped_Leans_" + session     //name of the graph window

        Display/K=1/N=$wn gDFR:$y1 vs gDFR:$x1;DelayUpdate
        //Axes modification
        SetAxis/W=$wn left -0.08,0.08;DelayUpdate
        SetAxis/W=$wn bottom -0.08,0.08;DelayUpdate
        ModifyGraph axThick=1.5 //Axes thickness
        ModifyGraph/W=$wn standoff=0;DelayUpdate
        ModifyGraph/W=$wn zero=1;DelayUpdate
        ModifyGraph/W=$wn lblMargin(left)=10,lowTrip=0.01;DelayUpdate
        ModifyGraph/W=$wn fSize=16;DelayUpdate //font size of ticks
        ModifyGraph/W=$wn width=432,height=432//graph area is square at 6inx 6in
        //Labels
        Label/W=$wn left "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'y \\M\\]0 (ML Direction)";DelayUpdate
        Label/W=$wn bottom "\\Z24\\[0\\F'Symbol'q\\B\\F'Arial'x \\M\\]0 (AP Direction)";DelayUpdate
        TextBox/W=$wn/C/N=text0/D=1.5/A=LT/X=0.37/Y=0.33 "\\Z18Lean Sets"

        //If loop for appending the other optional groups traces
        if(strlen(x2) !=0 && strlen(x3) != 0)
                AppendToGraph/W=$wn gDFR:$y2 vs gDFR:$x2;DelayUpdate
                ModifyGraph/W=$wn rgb($y2)=(0,0,65535);DelayUpdate
                AppendToGraph/W=$wn gDFR:$y3 vs gDFR:$x3;DelayUpdate
                ModifyGraph/W=$wn rgb($y3)=(0,0,0);DelayUpdate
                Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) Group 1\r"
                AppendText/W=$wn "\\s(#1) Group 2\r\\s(#2) Group 3";DelayUpdate
                Legend/W=$wn/C/N=text1/J/A=RT/X=30.99/Y=35.03
        elseif(strlen(x2) !=0)
                AppendToGraph/W=$wn gDFR:$y2 vs gDFR:$x2;DelayUpdate
                ModifyGraph/W=$wn rgb($y2)=(0,0,65535);DelayUpdate
                Legend/W=$wn/C/N=text1/J/D=1.5/A=MC "\\Z16\\s(#0) Group 1\r\\s(#1) Group 2";
                Legend/W=$wn/C/N=text1/J/A=RT/X=-3.47/Y=0.33
        endif
End
```

```
//
//
// Main Fit Functions
// (in order seen in menu)
//
//

Function EllipseFit(Xwave, Ywave, fitElliX, fitElliY, trend, outOfBounds)                    //EllipseFit
            Wave Xwave, Ywave                                                                  //Inputs
            Wave fitElliX, fitElliY, trend, outOfBounds                                       //Output
            Variable phi,  x_max, x_min, y_max, y_min
            printf "Function: ellipseFit\r"

            //Finding the linear trend line of all the data
            Make/O/D/N=2 Coef
            MakeLinearTrend(Xwave, Ywave, trend, Coef)

            //Determine the value of phi, the angle of the trendline wrt the xaxis
            phi = DetermineAngle(trend, Coef)

            //Looking for the max and min x and y values of all the data
            x_max = WaveMax(Xwave)
            x_min = WaveMin(Xwave)
            y_max = WaveMax(Ywave)
            y_min = WaveMin(Ywave)

            //Finding the midpoint of the Ellipse
            variable midX, midY
            midX = (x_max + x_min)/2
            midY = (y_max + y_min)/2

            //Defining the semi-axis values. We are creating a horizontal ellipse, thus the a = major axis
            variable a, b
            a = x_max - midX
            b = y_max - midY

            //Creating the pnts for the ellipse
            MakeEllipse2(midX, midY, a,b, phi, fitElliX, fitElliY)

            //Finding the eccentricity, perimenter and area of an ellipse
            Variable elliArea, perimeter, eccen
            eccen = EllipseEccentricity(a,b)
            perimeter = EllipsePerimeter(a,b)
            elliArea = CalcElliArea(a, b)

            //Check for how many points lay outside of th ellipse
            variable length = numpnts(Ywave)
            variable pntsOut, nPnts, percent
            nPnts = numPnts(outOfBounds)
            pntsOut = 0
            pntsOut = PntCheck(Xwave, Ywave, midX, midY, a, b, phi, outOfBounds)
            printf "# pts Outside = %d\r", pntsOut
            percent = (pntsOut/length)*100
            printf "Percentage of pts Outside = %.5f%\r", percent
```

```
        //Get the string path and Fill in the parameters into the given global string
        String currentDF = GetDataFolder(1)
        string temp =  stringFromList(1, currentDF, ":")
        String strName = "root:" + temp + ":Models:ebParam";
        print strName
        FillParamStr(strName, elliArea, perimeter, percent, a, b, eccen)

        //Resizes the outOfBounds matrix such that all values are pts that are outside of the ellipse
        DeletePoints/M=0 pntsOut, (nPnts-1), outOfBounds

        //Kill excessive waves
        KillWaves Coef
End


//This is the main function that contains all of the sub-routines necessary to create an ellipse using a
//single maximum or minimum pt in the x or y-axis
Function OnePtEllipse(Xwave, Ywave, trend, fitX, fitY, outsidePts)                    //OnePtEllipse
        Wave Xwave, Ywave                                                        //Inputs
        Wave trend, fitX, fitY, outsidePts                                       //Outputs
        printf "Function: OnePtEllipse\r"

        //Finding the linear trend line of all the data
        Make/O/D/N=2 Coef
        MakeLinearTrend(Xwave, Ywave, trend, Coef)

        //Determine the value of phi, the angle of the trendline wrt the xaxis
        variable Phi
        phi = DetermineAngle(trend, Coef)

        //Finding all of the mins and maxes of data
        Make/O/D/N=(4,2) orig4Corners
        FindingFourCorners(Xwave, Ywave, orig4Corners)
        //Row names: [0] xmin; [1]xmax; [2]ymin; [3]ymax
        //Column names: [0] x values [1] y values

        //Finding the Center point of all the data: Pt(h,k)
        variable h, k
        Make/O/D/N=2 centerWave
        //Function that finds the midpoint between the mins and maxes to be the center pt of the data
        FindMidPoints(orig4Corners, centerWave)
        h = centerWave[0]
        k = centerWave[1]

        //Projecting all of the points on the trend line, including the midpoint
        Make/O/D/N=(5,2) projPoints
        //note that the output wave has 5 points, the first point pair is the projected center,
        // the the x & y mins and maxes: [0] midpt [1]xmin [2] xmax [3] ymin [4] ymax
        KeyPointProjections(centerWave, orig4Corners, Coef, projPoints)

        //Set up and function to find the semiAxes
        Make/O/D/N=2 semiAxes
        FindAxesWith1Pt(orig4Corners, projPoints, h, k, phi, semiAxes)
        variable a,b
```

```
	a = semiAxes[0]
	b = semiAxes[1]

	//Error check to make sure b is a real value
	if (b == numtype(2)) //if b == nan
		printf "Error: b = nan; Cannot use only a single point from the Ellipse\r Try using
TwoPtEllipse"

	else //if b is a real number
		//Function for drawing the ellipse
		MakeEllipse2( h, k, a, b, phi, fitX, fitY)

		//Finding the eccentricity, perimenter and area of an ellipse
		variable elliArea, perimeter, eccen
		eccen = EllipseEccentricity(a,b)
		perimeter = EllipsePerimeter(a,b)
		elliArea = CalcElliArea(a, b)

		//Check for how many points lay outside of th ellipse
		variable pntsOut, length2, percent
		length2 = DimSize(outsidePts, 0)
		pntsOut = PntCheck(Xwave, Ywave, h, k, a, b, phi, outsidePts)
		percent = pntsOut/(numPnts(Ywave))*100

		//Get the string path and Fill in the parameters into the given global string
		String currentDF = GetDataFolder(1)
		string temp =  stringFromList(1, currentDF, ":")
		String strName = "root:" + temp + ":Models:e1Param";
		print strName
		FillParamStr(strName, elliArea, perimeter, percent, a, b, eccen)

		//Resizing outsidePts wave to only contain pnts that are outside of the Ellipse
		DeletePoints/M=0 pntsOut, length2, outsidePts

	endif

	//Kill waves
	Killwaves orig4Corners, projPoints, semiAxes, centerWave, Coef
End


Function TwoPtEllipse(Xwave, Ywave, trend, fitX, fitY, outPts)				//TwoPtEllipse
	Wave Xwave, Ywave								//Inputs
	Wave trend, fitX, fitY, outPts							//Outputs
	printf "Function: TwoPtEllipse\r"

	//Finding the linear trend line of all the data
	Make/O/D/N=2 Coef
	MakeLinearTrend(Xwave, Ywave, trend, Coef)

	//Determine the value of phi, the angle of the trendline wrt the xaxis
	variable Phi
	phi = determineAngle(trend, Coef)

	//Finding all of the mins and maxes of data
	Make/O/D/N=(4,2) orig4Corners
	FindingFourCorners(Xwave, Ywave, orig4Corners)
	//Row names: [0] xmin; [1]xmax; [2]ymin; [3]ymax

	//Finding the Center point of all the data: Pt(h,k)
	variable h, k
	Make/O/D/N=2 centerWave

	//Function for calculating the midPoints between the xMin & xMax and then between yMin &
yMax
	FindMidPoints(orig4Corners, centerWave)
	h = centerWave[0]
	k = centerWave[1]

	Make/O/D/N=2 semiAxes
	FindAxesWith2Pts(Xwave, Ywave, centerWave, orig4Corners, phi, semiAxes)
	variable a,b
	a = semiAxes[0]
	b = semiAxes[1]

	//Function for drawing the ellipse
	MakeEllipse2( h, k, a, b, phi, fitX, fitY)

	//Finding the eccentricity, perimenter and area of an ellipse
	variable elliArea, perimeter, eccen
	eccen = EllipseEccentricity(a,b)
	perimeter = EllipsePerimeter(a,b)
	elliArea = CalcElliArea(a, b)

	//Check for how many points lay outside of th ellipse
	variable pntsOut, length2, percent
	length2 = DimSize(outPts, 0)
	pntsOut = PntCheck(Xwave, Ywave, h, k, a, b, phi, outPts)
	percent = (pntsOut/(numPnts(Ywave)))*100

	//Get the string path and Fill in the parameters into the given global string
	String currentDF = GetDataFolder(1)
	string temp =  stringFromList(1, currentDF, ":")
	String strName = "root:" + temp + ":Models:e2Param";
	print strName
	FillParamStr(strName, elliArea, perimeter, percent, a, b, eccen)

	//Resizing outsidePts wave to only contain pnts that are outside of the Ellipse
	DeletePoints/M=0 pntsOut, length2, outPts

	//Kill waves
	Killwaves orig4Corners, semiAxes, centerWave, Coef
End
```

```
//This function creates a convex hull around the points provided in Xwave and Ywave and puts them into
//outXhull, and outYhull. Additionally, it calculates the area and perimeter of the hull and writes it into
//the global string chParam
Function FitConvexHull(Xwave, Ywave, outXHull, outYhull)                          //FitConvexHull
        Wave Xwave, Ywave                                                     //Inputs
        Wave outXhull, outYhull                                               //Outputs

        Convexhull/C Xwave, Ywave
        Wave W_YHull, W_XHull //output waves of Convexhull

        Variable length, length2
        length = numPnts(W_YHull)
        length2 = numPnts(outYhull)

        DeletePoints/M=0 length, length2+1, outYhull
        DeletePoints/M=0 length, length2+1, outYhull

        Duplicate/O W_YHull, outYhull
        Duplicate/O W_XHull, outXHull

        Variable n = numPnts(outXhull)
        Variable i = 0
        Make/O/D/N=(n) CHLoc
        for (i=0; i<n; i+=1)
                Variable temp = outXHull[i]
                Variable V_value
                FindValue/V=(temp) Xwave
                CHLoc[i] = V_value

        endfor

        //Moving CHLoc wave into the model folder
        DFREF mDFR = GetModelsDFR
        MoveWave CHLoc, mDFR


        Variable area1, perim1
        area1 = AreaCH(outXhull, outYhull)
        perim1 = PerimeterCH(outXhull, outYHull)
        printf "CH Area = %f\r", area1
        printf "CH Perimeter = %f\r", perim1

        //Put parameters into the global parameter wave for ConvexHulls
        String currentDF = GetDataFolder(1)
        string temp1 =  stringFromList(1, currentDF, ":")
        String strName = "root:" + temp1 + ":Models:chParam";
        print strName
        Variable percent = 0
        Variable a=-1, b=-1, e = -1
        FillParamStr(strName, area1, perim1, percent, a, b, e)

        Killwaves W_Yhull, W_XHull
End
```

```
//Function recieves a string that contains the session folder. This will create a DFR to the SingleLeans
//folder
Function/DF GetGroupedLeansDFR(sessionFolder)                                    //GetGroupedLeansDFR
            String sessionFolder                                          //Input
            SetDataFolder root:$(sessionFolder):Thetas:GroupedLeans
            DFREF gDFR = GetDataFolderDFR()
            return gDFR
End


//Function recieves a string that contains the session folder. This will create a DFR to the Models folder
Function/DF GetSessionDFR(sessionFolder)                                                     //GetSessionDFR
            String sessionFolder                                                      //Input
            SetDataFolder root:$(sessionFolder)
            DFREF sDFR = GetDataFolderDFR()
            return sDFR
End

//Function recieves a string that contains the session folder. This will create a DFR to the session folder
Function/DF GetModelsDFR(sessionFolder)                                          //GetModelsDFR
            String sessionFolder                                          //Input
            SetDataFolder root:$(sessionFolder):Models
            DFREF mDFR = GetDataFolderDFR()
            return mDFR
End

//Function recieves a str that contains the session folder. This will create a DFR to the SingleLeans folder
Function/DF GetSingleLeansDFR(sessionFolder)                                     //GetSingleLeansDFR
            String sessionFolder                                          //Input
            SetDataFolder root:$(sessionFolder):Thetas:SingleLeans
            DFREF sDFR = GetDataFolderDFR()
            return sDFR

End

//Function recieves a string that contains the session folder. This will create a DFR to the Theta folder
Function/DF GetThetasDFR(sessionFolder)                                                      //GetThetasDFR
            String sessionFolder                                                      //Input
            SetDataFolder root:$(sessionFolder):Thetas
            DFREF tDFR = GetDataFolderDFR()
            return tDFR

End

//Function recieves a string that contains the session folder. This will create a DFR to the Theta folder
Function/DF GetQuietStanceDFR(sessionFolder)                                     //GetQuietStanceDFR
            String sessionFolder                                                      //Input
            SetDataFolder root:$(sessionFolder):Thetas:QuietStance
            DFREF qsDFR = GetDataFolderDFR()
            return qsDFR
End
```

```
//This function is passed a data folder reference and sets the data folder.
//The string str containis inclusion/exclusion criteria for selecting the desired waves
//and putting them into a List.
Function/S GetWavesInFolder(dfr, str)                                   //GetWavesInFolder
        DFREF dfr                                                   //Input
        String str                                                  //Input
        SetDataFolder dfr
        String objList = WaveList(str, ";", "")
        return objList
End


//This function is based on the 2004 critical curve method created by Stirling, Zakynthiaki et al.
//This function recieves all data points (theta values) and creates a fit around those points using the method
//they created and is within the FcritCurve() function. Then the points are resorted and given back to the
//display function that call it
Function GreekMethod(allX, allY, outX, outY)                                      //GreekMethod
        Wave allX, allY                                                        //Inputs
        Wave outX, outY                                                        //Outputs

        variable leng, doubleLeng
        leng = numPnts(allX)
        doubleLeng = 2*leng

        Make/O/D/N=(4,2) fourCorners
        //fourCorners is in the format of: col[0] - x values; col[1] - y values
        // row[0] - xMin; [1] - xMax; [2] - yMin; [3] - yMax;
        FindingFourCorners(allX, allY, fourCorners)

        Make/O/D/N=(doubleLeng) critCurveX, critCurveY
        Variable error
        //Creates all of the critical curve y points.
        FcritCurve(allX, allY, critCurveX, critCurveY, error)
        printf "error top= %d\r", error
        //Sorts out the NaN value pairs and orders the final wave set in increasing order
        SortFcritCurve(critCurveX, critCurveY, outX, outY)

        //Other Calculations and Function calls
        Variable perim, space
        perim = PerimeterCH(outX, outY)
        printf "Perimeter = %.5f\r", perim
        space = AreaCH(outX, outY)
        printf "Area = %.5f\r", space

        //Get the string path and Fill in the parameters into the given global string
        String currentDF = GetDataFolder(1)
        string temp =  stringFromList(1, currentDF, ":")
        String strName = "root:" + temp + ":Models:grParam";
        print strName
        Variable a=-1, b=-1, e=-1, percent = -1
        FillParamStr(strName, space, perim, percent, a, b, e)

        //KillWaves critCurveX, critCurveY, fourCorners
End
```

```
//This function creates an information density plot (surface plot in igor). You define the size of your NxN
//matrix by setting the variable "N". The data that is inputted will first be shifted in the x and y coordinates
//to ensure all points are contained in the 1st quadrant. Then each point pair will be used to determine it's
//xbox and ybox place and its linear BoxLocation. The box each point is put in "snaps" to left and bottom
//edges of the box to be counted. Then in the subroutine "counting" the number of points in each box in the
//NxN matrix is calculated and an NxN is returned with all of those values.
Function BoxCounting(inX, inY, N, output, logOutput)                              //BoxCounting
        Wave inX, inY                                                          //Inputs
        // number of boxes in the rows & columns to make an NxN matrix
        Variable N                                                             //Inputs
        Wave output, logOutput                                                 //Ouput

        //Duplicates the input data into a local instance of the data, so the actual waves passed into the
        // function remain unaltered during the shifting process
        Duplicate/O inX, xValues
        Duplicate/O inY, yValues

        //Shifting Set up
        variable xmin, ymin, xmax, ymax
        xmin = wavemin(xValues)
        ymin = wavemin(yValues)

        // This shifts the x and y values by their minimum value to put all points into Quadrant I
        // of a graph. Now all x and yValues are non-negative and allows for better flexibility for
        // the program to accept any kind of data set
        xValues = xValues -xmin
        yValues = yValues - ymin

        xmax = waveMax(xValues)
        ymax = wavemax(yValues)
        printf "xmin: %f\t xmax: %f\r", xmin, xmax
        printf "ymin: %f\t ymax: %f\r", ymin, ymax
        Variable yrange = ymax - ymin, xrange = xmax - xmin
        printf "xrange = %f\t yrange: %f\r", xrange, yrange

        //Scaled shift values in the x and y direction
        variable xShift, yShift
        xShift = (xmin*N)/xmax
        yShift = (ymin*N)/ymax
        Printf "xshift: %f\t yshift: %f\r", xshift, yshift

        Variable boxCount, boxSize                  //total number of boxes in all space
        boxCount = N^2
        boxSize = 1/N                                       //height and width of boxes

        //Setup for determining which box each point belongs in.
        Variable length = numpnts(xValues)          //number of variables in the xn, yn waves
        Make/O/N =(length) xplace, yplace, boxLocation
        Variable i =0

        //Calculates which box the x-coordinate belongs in
        for (i=0; i<length; i+=1)
                xplace[i] = floor(xValues[i]*(N/xmax))
                if (xplace[i] == N)
                        xplace[i] = N-1
```

```
			endif
		endfor

		//Calculates which box the y-coordinate belongs in
		for (i=0; i<length; i+=1)
			yplace[i] = floor(yValues[i]*(N/ymax))
			if (yplace[i] == N)
				yplace[i] = N-1
			endif
		endfor
		//The known xbox and ybox places can be used to calculate the linear and absolute boxLocation in
		// the NxN matrix
		for (i=0; i<length; i+=1)
			boxLocation[i] = N*yplace[i] + xplace[i]
		endfor

		Make/O/N=(N) Bx, By
		Bx = p+xShift
		By = p+yShift

		//this is the density of points in each box, in a linear vector of boxes (not NxN)
		Make/O/N=(boxCount) linDensity
		Counting(boxLocation, linDensity)

		//Redimensioning linear Density to a NxN matrix
		Duplicate/O linDensity, Density
		Redimension/N=(N,N) Density

		//Setting output wave equal to density for final output of the function
		output = Density
		logOutput = log(Density)

		//Killing all the waves that are unnecssary to see
		KillWaves Bx, By, xplace, yplace, boxLocation, xvalues, yvalues, linDensity, Density
End


Function HistTest()										//HistTest
		SetDataFolder root:
		//User chooses data folder to use
		String folder
		folder = SelectDataFolder()				//note that the name of the folder is the session name
		//Splitting the string to extract the trial number
		String name, sessionNum
		String regExpr = "([[:alpha:]]+)([[:digit:]]+)"
		SplitString/E=regExpr folder, name, sessionNum

		//Setting up all of the data folder references we will need for creating the display
		DFREF sessionDFR		= GetSessionDFR(folder)
		DFREF tDFR				= GetThetasDFR(folder)
		DFREF qDFR		= GetQuietStanceDFR(folder)
		SetDataFolder tDFR
		//From the folder selected the corresponding X and Y waves is used
		String XYtrialList, inX, inY
		XYtrialList = waveList("trial*", ";", "DP:1")
```

```
inX = StringFromList(0, XYtrialList)
inY = StringFromList(1, XYtrialList)

Variable ymax, ymin, xmax, xmin, yrange, xrange
ymax = WaveMax($inY); ymin = WaveMin($inY); yrange = ymax - ymin;
xmax = WaveMax($inX); xmin = WaveMin($inX); xrange = xmax - xmin;
printf "ymax = %.5f\t ymin = %.5f\t range: %.5f\r", ymax, ymin, yrange
printf "xmax = %.5f\t xmin = %.5f\t range: %.5f\r", xmax, xmin, xrange

SetDataFolder qDFR
//From the folder selected the corresponding X and Y waves is used
String QStrialList, inXq, inYq
QStrialList = waveList("qs*", ";", "DP:1")
print QStrialList
inXq = StringFromList(0, QStrialList)
inYq = StringFromList(1, QStrialList)

Variable qymax,qymin, qxmax, qxmin, qxrange, qyrange
qymax = WaveMax($inYq); qymin = WaveMin($inYq); qyrange = qymax - qymin;
qxmax = WaveMax($inXq); qxmin = WaveMin($inXq); qxrange = qxmax - qxmin;
printf "qymax = %.5f\t qymin = %.5f\t range: %.5f\r", qymax, qymin, qyrange
printf "qxmax = %.5f\t qxmin = %.5f\t range: %.5f\r", qxmax, qxmin, qxrange

//Making the historgram waves
SetDataFolder root:
Variable binNum = 50
Variable yBinSize = yrange/binNum;
Variable yBinSizeQ = yrange/(2*binNum);
Make/O/D/N=(binNum) histResultY, histResultQ
Histogram/B={ymin, yBinSize, binNum} tDFR:$inY, histResultY
Histogram/B={ymin, yBinSizeQ, 2*binNum} qDFR:$inYq, histResultQ
histResultQ = -histResultQ
//Plotting the histograms
String wn1 = "yPnts"+folder
Display/N=$wn1/K=1 histResultY
AppendToGraph/W=$wn1 histResultQ
ModifyGraph/W=$wn1 mode=5,rgb(histResultQ)=(0,0,0)
ModifyGraph/W=$wn1 mode=5
SetAxis bottom -0.03,0.03
//Making the historgram waves
SetDataFolder root:
Variable xBinSize = xrange/binNum
Variable xBinSizeQ = xrange/(2*binNum)
Make/O/D/N=(binNum) histResultX, histResultQx
Histogram/B={ymin, yBinSize, binNum} tDFR:$inY, histResultX
Histogram/B={ymin, yBinSizeQ, 2*binNum} qDFR:$inYq, histResultQx
histResultQx = -histResultQx
//Plotting the histograms
String wn2 = "xPnts"+folder
Display/N=$wn2/K=1 histResultX
AppendToGraph/W=$wn2 histResultQx
ModifyGraph/W=$wn2 mode=5,rgb(histResultQx)=(0,0,0)
ModifyGraph/W=$wn2 mode=5
SetAxis bottom -0.03,0.03
```

```
	//Distances from center
	Variable leng1 = numPnts(tDFR:$inX)
	Variable leng2 = numPnts(qDFR:$inXq)
	Make/O/D/N=(leng1) dist
	Make/O/D/N=(leng2) distq
	Print GetDataFolder(1)
	DistanceFromPoint(0, 0, tDFR:$inX, tDFR:$inY, root:dist)
	DistanceFromPoint(0, 0, qDFR:$inXq, qDFR:$inYq, root:distq)

	Variable distMax, distMin, distqMax, distqMin, distRange, distqRange
	distMax = WaveMax(dist); distMin = WaveMin(dist); distRange = distMax - distMin;
	distqMax = WaveMax(distq); distqMin = WaveMin(distq); distqRange = distqMax - distqMin;
	printf "distMax = %.5f\t distMin = %.5f\t range: %.5f\r", distMax, distMin, distRange
	printf "distqMax = %.5f\t distqMin = %.5f\t range: %.5f\r", distqMax, distqMin, distqRange

	//Making the historgram waves
	SetDataFolder root:
	Variable distBinSize = distRange/binNum
	Variable distQBinSize = distRange/100
	Make/O/D/N=(binNum) histDist, histDistQ
	Histogram/B={distMin, distBinSize, binNum} dist, histDist
	Histogram/B={distMin, distQBinSize, binNum} distq, histDistQ
	histDistQ = -histDistQ
	CurveFit/Q/M=2/W=0 LogNormal, histDistQ/D
	CurveFit/Q/M=2/W=0 LogNormal, histDist/D
	Wave fit_histDist, fit_histDistQ
	//Plotting the histograms
	String wn3 = "distances" +folder
	//Traces
	Display/N=$wn3/K=1 histDist
	SetAxis/W=$wn3 bottom 0,0.04
	ModifyGraph/W=$wn3 mode=5
	AppendToGraph/W=$wn3 histDistQ
	AppendToGraph/W=$wn3 fit_histDistQ
	AppendToGraph/W=$wn3 fit_histDist
	ModifyGraph/W=$wn3 mode=5,rgb(histDistQ)=(0,0,0)
	ModifyGraph/W=$wn3 hbFill(histDist)=6
	ModifyGraph/W=$wn3 hbFill(histDistQ)=7
	ModifyGraph/W=$wn3 mode(fit_histDist)=0, lsize(fit_histDistQ)=1.5
	ModifyGraph/W=$wn3 mode(fit_histDistQ)=0, rgb(fit_histDistQ)=(0,0,0), lsize(fit_histDist)=1.5
	//Axes
	SetAxis/W=$wn3 left -25000,15000
	ModifyGraph/W=$wn3 nticks(left)=7,lblMargin(left)=30,standoff=0;DelayUpdate
	ModifyGraph fSize=14
	ModifyGraph width=432,height=432
	//Labels
	TextBox/W=$wn3/C/N=text0/A=MT/E "\\Z18Distribution of Distance from Center"
	Label/W=$wn3 left "\\Z16Number of Points";DelayUpdate
	Label/W=$wn3 bottom "\\Z16Distance from Center ";DelayUpdate
	Legend/W=$wn3/C/N=text1/J/A=MT/E "\\s(histDist) Lean Data\r"
	AppendText/W=$wn3 "\\s(fit_histDist) LogNormal Fit\r\\s(histDistQ) Quiet Stance Data\r"
	AppendText/W=$wn3 "\\s(fit_histDistQ) LogNormal Fit"
	Legend/W=$wn3/C/N=text1/J/A=RT/X=-1.36/Y=-1.16/E=0
End
```

```
//
//
// Sub-routine Functions
// (in alphabetical order)
//
//


//Calculates the area of the Convex hull. This is done by finding the midpoint of all the data and defining
// it as the center point. From this center point triangles of neighboring points are created and area of
//each triangle is calculated. The sum of all the triangles is the final area of the Convex hull.
Function AreaCH(Xpts, Ypts)                                                         //AreaCH
        Wave Xpts, Ypts                                                         //Inputs
        Variable answer                                                         //Output

        WaveStats/W/Q Xpts
        Wave M_WaveStats                                  //output wave for WaveStats
        Variable xMin, Xmax, Xminmatch, Xmaxmatch, xMinLoc, xMaxLoc

        xMin = M_WaveStats[10]
        xMinLoc = M_WaveStats[9]
        xMax = M_WaveStats[12]
        xMaxLoc = M_WaveStats[11]
        xMinMatch = Ypts[xMinLoc]
        xMaxMatch = Ypts[xMaxLoc]

        Variable midX, midY
        midX = (xMin + xMax)/2
        midY = (xMinMatch + xMaxMatch)/2

        Variable length,i
        length = numPnts(Xpts)

        //Finding the distance between the pts and the midpt = (midX, midY)
        //Note that the last points of X and Y are the same as the first points
        Make/O/N=(length) d2Mid
        for( i=0; i<length; i+=1)
                d2Mid[i] = sqrt((midX - Xpts[i])^2 + (midY - Ypts[i])^2)
        endfor

        //Finding the distance between each pt pairs in Xpts & Ypts to act as base length for area of
        // triangle equation
        Make/O/N=(length) bases
        for( i=0; i<length -1; i+=1)
                bases[i] = sqrt((Xpts[i+1]-Xpts[i])^2 + (Ypts[i+1]-Ypts[i])^2)
        endfor

        //Finding the areas of each triangle slice of the polygon, using Heron's Formula
        //Heron's Formula: s = 1/2 perimeter; A = sqrt(s(s-side1)(s-side2)(s-side3))
        Make/O/N=(length) triAreas
        variable s, finalS
        for(i = 0; i < (length-1); i+=1)
                s = (d2Mid[i] + d2Mid[i+1] + bases[i])/2
                triAreas[i] = sqrt(s*(s-d2Mid[i])*(s-d2Mid[i+1])*(s-bases[i]))
        endfor
```

```
		answer = sum(triAreas)
		killWaves d2mid, triAreas, bases, M_wavestats
		return answer
End


//Calculate the Area of an Ellipse: Given the semi-axes length the area of an ellipse can be found using
// the equation: A = Pi*a*b
Function CalcElliArea(a, b)								//CalcElliArea
		Variable a, b							//Inputs
		Variable answer							//Output

		answer = PI*a*b
		return answer
End


//Double checking that all maximum angles have the correct sign
Function CheckMaximums(inWave)							//CheckMaximums
		Wave inWave							//Input

		Variable phi_f, phi_b, phi_r, phi_l
		phi_f	=	inWave[1][0]			//xmax - x value
		phi_r	=	inWave[2][1]			//ymin - y value
		phi_b	=	inWave[0][0]			//xmin - x value
		phi_l	=	inWave[3][1]			//ymax - y value

		if( phi_f + phi_b <0)
			printf "Error: phi_b > phi_f\r Check data orientation\r"
			return 1
		elseif (phi_f + phi_b == 0)
			printf "Warning: phi_b = phi_f\r"
			return 0
		else
			return 0
		endif

End


//This function takes the inputted string, which contain a list of waves, and concatenate the waves in
// the list into a new output Wave. The original waves in the given lists are not killed.
Function ConcatWaves(list, outWave)							//ConcatWaves
		String list							//Inputs
		Wave  outWave							//Outputs

		Concatenate/O/NP  list, temp
		outWave = temp

		//Kills temporary waves after outputting them as outX1and outY1
		Killwaves temp
End
```

```
//Function recieves an input wave (array) and will return a wave (output). Array is filled with intergers
// and maxWave(array) <= numpnts(output). The function will loop through the values in the array.
//Every value in the array is a value which corresponds to a location in //the output wave. Every instance
//of array will increase the count in output for that array value location in output.
Function Counting( array, output)                                                    //Counting
        Wave array                                                           //Input
        Wave output                                                          //Output

        variable i = 0
        for (i=0; i<(numpnts(array)); i+=1)
                output[array[i]] += 1
        endfor
end


//This function takes the 2points in the wave inTrend to calculate the angle between the trend line and
//x-axis inCoef is currently also passed in to see if there is a difference between atan and atan2
Function DetermineAngle(inTrend, inCoef)                                     //DetermineAngle
        Wave inTrend, inCoef                                         //Inputs
        Variable angleOut                                            //Output

        Variable x, y
        y = inTrend[1][1] - inTrend[0][1]
        x = inTrend[1][0] - inTrend[0][0]
        angleOut = atan2(y,x)

        return angleOut
End


//This function take the input wave (w1) and finds how many positive x values and negative x values
//there are. # of positive Xs = j                    # of negative Xs = k                    output = {j,k}
Function DetermineWavesSize(w1, wout)                                        //DetermineWavesSize
        wave w1                                                      //Input
        wave wout                                                    //Output

        variable leng = numpnts(w1)
        variable i, j,k
        i = 0
        j= 0
        k =0

        for (i=0; i<leng; i+=1)
                if (w1[i] > 0 || w1 == 0 )
                        j += 1
                elseif(w1[i] < 0)
                        k += 1
                endif
        endfor
        wout = {j,k}

End
```

```
//Finds the distance between the points in the waves: xPts, yPts and the given (x,y) pt. The output
//is a wave with the same length as the inputted waves
Function DistanceFromPoint(x, y, xPts, yPts, outDist)                       //DistanceFromPoint
        Variable x, y                                                //Inputs
        Wave xPts, yPts                                              //Inputs
        Wave outDist                                                 //Output

        Variable leng = numPnts(xPts); Variable leng1 = numPnts(yPts)
        Variable i

        if(leng != leng1)
                Printf "DistanceFromPoint Error: Inputted waves are different sizes"
        else
                for(i = 0; i < leng; i += 1)
                        outDist[i] = sqrt((x - xPts[i])^2 + (y - yPts[i])^2)
                endfor
        endif
End


//Calculates the eccentricity of an ellipse. The value of e increases as the ellipse is more "squashed"
//Eccentricity can be defined as how "round" the ellipse is.
Function EllipseEccentricity(a,b)                                            //EllipseEccentricity
        Variable a,b                                                 //Inputs
        Variable e //eccentricity                                    //Output
        Variable temp
        if (a > b)
                temp = b^2/a^2
                e = sqrt(1-temp)
                return e
        else //if b > a
                temp = a^2/b^2
                e = sqrt(1-temp)
                return e
        endif
End


//Return perimeter of ellipse based on Ramanujan's 1914 formula:
// P = pi*(a+b)*(1+(3*h^2)/(10+sqrt(4-3*h^2)))
//Equations found in Almkuist and Berndy 1988 paper in Amer. Math Monthly
Function EllipsePerimeter(a,b)                                               //EllipsePerimeter
        Variable a, b                                                //Inputs
        Variable perimeter                                           //Output

        Variable h = (a-b)/(a+b)
        perimeter = PI*(a+b)*(1+((3*h^2)/(10+sqrt(4 - 3*h^2))))
        return perimeter
End
```

```
//This function calculates all of the points on the critical curve using the 2004 Greek method.
//For each inX value there are two inY values.
Function FcritCurve(inX, inY, outX, outY, error1)                                    //FcritCurve
            Wave inX, inY                                                        //Input
            Wave outX, outY                                                      //Output
            Variable error1                                                      //Output

            Make/O/D/N=4 fourCorners
            FindingFourCorners(inX, inY, fourCorners)
            //fourCorners is in the format of: col[0] - x values; col[1] - y values
            // row[0] - xMin; [1] - xMax; [2] - yMin; [3] - yMax;

            //Checks if forward lean is greater than backward lean.
            //The model can't work if backward lean is greater than forward lean
            error1 = CheckMaximums(fourCorners)

            if(error1 == 0)
                        Make/O/D/N=9 coefs
                        MakeCoefs(fourCorners, coefs)
                        // coefs = {A, B, C, D, E, G, H, I, J}
                        // coefs = {0,  1, 2,   3, 4, 5,  6, 7, 8}

                        //Using the quadratic equation we are solving for y of the following equation:
                        // Ax^2 + Bx + Cy^2+ Dy + Gxy^2 + Hx^2y +Ix^2y^2 + Jxy - E =
                        //Therefore y = [-Beta +/- sqrt(Beta^2 - 4*Alpha*Gamma)]/2*Alpha
                        Variable beta1, alpha1, gamma1
                        variable n, totN, count
                        totN  = numPnts(inX)
                        count = 0

                        variable leng = numpnts(inX)
                        Make/O/D/N=(leng) tempY1, tempY2

                        //For each thetaX value, the coefs are used to find the alpha, beta, and gamma, variables
                        // used to solve the quadratic equation for thetaY
                        for (n = 0; n< totN; n+=1)
                                    beta1      = coefs[6] * (inX[n])^2 + coefs[8] * inX[n] + coefs[3]
                                    alpha1    = coefs[7] * (inX[n])^2 + coefs[5] * inX[n] + coefs[2]
                                    gamma1 = coefs[0] * (inX[n])^2 + coefs[1] * inX[n]  - coefs[4]

                                    tempY1[n] = (-beta1 + sqrt((beta1)^2 - 4*alpha1*gamma1))/(2*alpha1)
                                    tempY2[n] = (-beta1 - sqrt((beta1)^2 - 4*alpha1*gamma1))/(2*alpha1)

                                    //Counting how many values are NaN
                                    if (numtype(tempY1[n]) == 2 )
                                                count +=1
                                    endif

                        endfor

                        if (count > 0)
                                    printf "Count  = %d\r", count
                        endif

            elseif (error1 ==1)
```

```
                print "Error1 = 1\r"
                return -1
        endif

        //Since there are 2 tempYs the outY is the concatenation of the two. The same is done for
        //inX such that the same number of points and order is maintained
        Concatenate/O/NP {inX, inX}, tempX
        Concatenate/O/NP/KILL {tempY1, tempY2}, tempY

        outX = tempX
        outY = tempY
        Killwaves fourCorners, coefs, tempX, tempY

End


//This function is given a global string name and the parameter values that belong in the str key
Function FillParamStr(strName, theArea, circum, pout, semiA, semiB, eccen)          //FillParamStr
        String strName                                                      //Input
        Variable theArea, circum, pout, semiA, semiB, eccen                  //Inputs
        SVAR gS = $strName                                                  //Global
        gS =  ReplaceNumberByKey("area", gS, theArea, "=")
        gS =  ReplaceNumberByKey("perim", gS, circum, "=")
        gS =  ReplaceNumberByKey("pout", gS, pout, "=")
        gS =  ReplaceNumberByKey("semiA", gS, semiA, "=")
        gS =  ReplaceNumberByKey("semiB", gS, semiB, "=")
        gS =  ReplaceNumberByKey("eccen", gS, eccen, "=")
        Print gS
End


//This function is used to determine the semiAxes of the One Pt Ellipse algorithm. Based on the projected
//points the major axis is determined by the distance between the midpoint and the projected mins and
//maxes onto the regression line. Once the major axis is defined, the equation in MinorAxisSelection()
//is used to calculate the minor axis length. These values are outputed to the calling function.
Function FindAxesWith1Pt(inOrig4Corners, inProjPoints, inH, inK, inPhi, outputWave) //AxesWith1Pt
        Wave inOrig4Corners, inProjPoints                                   //Inputs
        variable inH, inK, inPhi                                            //Inputs
        Wave outputWave                                                     //Output

        //Calling Functions to calculate major and minor axis
        variable a
        a = MajorAxisSelection(inProjPoints)
        variable b
        b = MinorAxisSelection(inOrig4Corners, inH, inK, a, inPhi)

        //Answer
        outputWave = {a, b}
End
```

```
//This function is used to determine the semiAxes of the Two Pt Ellipse algorithm. The in points are
//the four corners (maximums) of the data with their respective x or y value. This function finds the
//pair of maximums from four corners to calcuate the semi axes. The pair that creates the semi axes that
//include the most data points is chosen and returned to the calling function in outAxes.
Function FindAxesWith2Pts(inX, inY ,inCenter, inPoints, angle, outAxes)        //FindAxesWith2Pts
        Wave inX, inY, inCenter, inPoints                                    //Inputs
        Variable angle                                                       //Inputs
        Wave outAxes                                                         //Output

        variable h, k
        h = inCenter[0]
        k = inCenter[1]

        Make/O/D/N = (6,2) Point1, Point2
        //These two arrays create the combination of all four inPoints paired with each of the other
inPoints
        Point1[0][0] = inPoints[0][0]
        Point1[0][1] = inPoints[0][1]
        Point1[1][0] = inPoints[0][0]
        Point1[1][1] = inPoints[0][1]
        Point1[2][0] = inPoints[0][0]
        Point1[2][1] = inPoints[0][1]
        Point1[3][0] = inPoints[1][0]
        Point1[3][1] = inPoints[1][1]
        Point1[4][0] = inPoints[1][0]
        Point1[4][1] = inPoints[1][1]
        Point1[5][0] = inPoints[2][0]
        Point1[5][1] = inPoints[2][1]

        Point2[0][0] = inPoints[1][0]
        Point2[0][1] = inPoints[1][1]
        Point2[1][0] = inPoints[2][0]
        Point2[1][1] = inPoints[2][1]
        Point2[2][0] = inPoints[3][0]
        Point2[2][1] = inPoints[3][1]
        Point2[3][0] = inPoints[2][0]
        Point2[3][1] = inPoints[2][1]
        Point2[4][0] = inPoints[3][0]
        Point2[4][1] = inPoints[3][1]
        Point2[5][0] = inPoints[3][0]
        Point2[5][1] = inPoints[3][1]

        Make/O/D/N=(6,2) possibleAxes
        FindPossisbleAxes(Point1, Point2, h, k, angle, possibleAxes)

        variable length2 = dimSize(possibleAxes,0)
        variable n,rawr
        rawr = 0

        Variable tempRawr, row
        tempRawr = 0

        for(n = 0; n< length2; n+=1)
                rawr = OptimizePntsOut(inX, inY, h, k, possibleAxes[n][0], possibleAxes[n][1], angle)
                if(n==0)
```

```
                        tempRawr = rawr
                endIf

                if(rawr < tempRawr)
                        tempRawr = rawr
                        row = n
                endif
        endfor

        outAxes[0] = possibleAxes[row][0]
        outAxes[1] = possibleAxes[row][1]

        //Kill waves
        killWaves Point1, Point2, possibleAxes
End


//This function finds the coordinate location for each of the x and y mins and maxes
Function FindingFourCorners(inX, inY, outWave)                          //FindingFourCorners
        Wave inX, inY                                           //Inputs
        Wave outWave                                            //Outputs

        //Setting up variable names
        variable xMax, xMaxLoc, xMin, xMinLoc
        variable yMax, yMaxLoc, yMin, yMinLoc
        variable xMax_yMatch, xMin_yMatch, yMax_xMatch, yMin_xMatch

        // Xwave Stats
        WaveStats/Q/C=1/W inX
        Wave M_WaveStats        //output wave for WaveStats
        Wave W_IPIV                     //wave made by WaveStats, but is unused by us
        xMin = M_WaveStats[10]
        xMinLoc = M_WaveStats[9]
        xMax = M_WaveStats[12]
        xMaxLoc = M_WaveStats[11]
        //Finding the corresponding y values for the X min and max values
        xMin_yMatch = inY[xMinLoc]
        xMax_yMatch = inY[xMaxLoc]
        //Kill unnecessary waves
        KillWaves M_WaveStats, W_IPIV

        // Ywave Stats
        WaveStats/W/Q/C=1 inY
        Wave M_WaveStats        //output wave for WaveStats
        Wave W_IPIV                     //wave made by WaveStats, but is unused by us
        yMin = M_WaveStats[10]
        yMinLoc = M_WaveStats[9]
        yMax = M_WaveStats[12]
        yMaxLoc = M_WaveStats[11]
        //Finding the corresponding X values for the Y min and max values
        yMin_xMatch = inX[yMinLoc]
        yMax_xMatch = inX[yMaxLoc]
        //Kill unnecessary waves
        KillWaves M_WaveStats, W_IPIV
```

```
            //Put answers into the output wave: outWave
            // columns: [0] - x values;  [1] - y values
            // rows: [0] - x Min;          [1] - x Max;        [2] - y Min          [3] - y Max
            outWave[][0] = { xMin, xMax, yMin_xMatch, yMax_xMatch}
            outWave[][1] = { xMin_yMatch, xMax_yMatch, yMin, yMax}
End


//Solves the equation of an ellipse: (g/a)^2 + (h/b)^2 = 1, with center at (h,k) and angle wrt xaxis
//of phi where:
//lambda = (x-h)cos(phi) + (y-k)sin(phi)
//eta                   = (x-h)sin(phi)  - (y-k)cos(phi)
Function FindLambdaAndEta(inWave, Xmid, Ymid, phi, outWave)                 //FindLambdaAndEta
            Wave inWave                                                 //Input
            variable Xmid, Ymid, phi                                    //Inputs
            Wave outWave                                                //Output

            variable eta, lambda
            lambda  =           (inWave[0] - Xmid)*cos(phi)          +  (inWave[1] - Ymid)*sin(phi)
            eta         =           (inWave[0] - Xmid)*sin(phi)          -  (inWave[1] - Ymid)*cos(phi)
            outWave = {eta, lambda}
End


//This function finds the midpoint between the mins and maxes to be center point of the data
// the inMatrix is a 4x2 matrix that contains the ponts of xmin, xmax, ymin, ymax points respectively
// with rows - xvals & columns - yvals
// output: outWave[0] = midX, outWave[1] = midY
Function FindMidPoints(inMatrix, outWave)                                                   //FindMidPoints
            Wave inMatrix                                                           //Inputs
            Wave outWave                                                            //Outputs

            Variable MidX, MidY
            MidX = (inMatrix[0][0] + inMatrix[1][0])/2
            MidY = (inMatrix [2][1] + inMatrix[3][1])/2
            outWave = {MidX, MidY}
End


//This function recieves two points, the midpoint of the data, and the angle of the regression line.
//Using these parameters the function calculates the semiaxes a and b, using the equations:
// a^2 = lambda1^2/[1-{(lambda2/lambda1)^2eta1^2-eta1^2 }/{(lambda2/lambda1)^2eta1^2-eta2^2}]
// b^2 = [ (lambda2/lambda1)^2eta1^2 - eta2^2 ]/[ (lambda2/lambda1)^2 - 1 ]
//Please refer to Victoria Smith's MS Thesis for all equations
Function FindPossisbleAxes(inPoint1, inPoint2, midX, midY, inPhi, outAxes)    //FindPossisbleAxes
            Wave inPoint1, inPoint2                                     //Inputs
            Variable inPhi, midX, midY                                  //Inputs
            Wave outAxes                                                //Output

            Variable lambda1, eta1, lambda2, eta2
            variable i
            variable a,b

            variable j, length, numer, denom
            j = 0
```

```
	length = Dimsize(outAxes,0)

	for(i=0; i<length; i+=1)
		lambda1 = (inPoint1[i][0] - midX)*cos(inPhi) + (inPoint1[i][1] - midY)*sin(inPhi)
		eta1	= (inPoint1[i][0] - midX)*sin(inPhi)   - (inPoint1[i][1] - midY)*cos(inPhi)
		lambda2 = (inPoint2[i][0] - midX)*cos(inPhi) + (inPoint2[i][1] - midY)*sin(inPhi)
		eta2	= (inPoint2[i][0] - midX)*sin(inPhi)   - (inPoint2[i][1] - midY)*cos(inPhi)

		numer = (lambda1^2)/(1-((eta1^2*((lambda2/lambda1)^2-1))))
		denom = ((lambda2/lambda1)^2)*eta1^2-eta2^2
		a = sqrt(numer/denom)
		b = sqrt(((lambda2/lambda1)^2*eta1^2 - eta2^2) / ((lambda2/lambda1)^2 - 1))

		if(numtype(a) ==0 && numtype(b) ==0)
			outAxes[j][0] = a
			outAxes[j][1] = b
			j+=1
		endif
	endfor

	DeletePoints/M=0 j,length, outAxes
End


//This function will project the center point [aka inCenter] and 4 corners (x&y mins and maxes)
//[aka in4Corners] onto the trend line with the slope and y-intercept given [aka inCoef] and put all
//of the projected coordinates into the wave matrix outPoints. Note that the output wave has 5 points,
//the first point pair is the projected center, the x & y mins and maxes. The points will be projected by
//solving linear eqs in matrix form: A*X = B ; X = A^-1*B, where we know values for the A & B matrices
Function KeyPointProjections(inCenter, in4Corners, inCoef, outPoints)		//KeyPointProjections
	Wave inCenter, in4Corners, inCoef						//Inputs
	Wave outPoints									//Outputs

	//Creating local variables in order to make the equations easier to decifer
	variable slope, yIntercept
	slope = inCoef[0]
	yIntercept = inCoef[1]

	variable midX, midY
	midX	=	inCenter[0]
	midY	=	inCenter[1]

	variable xMin, xMax, yMin, yMax
	variable xMin_yMatch, xMax_yMatch, yMin_xMatch, yMax_xMatch
	xMin			=	in4Corners[0][0]
	xMin_yMatch		=	in4Corners[0][1]
	xMax			=	in4Corners[1][0]
	xMax_yMatch		=	in4Corners[1][1]
	yMin			=	in4Corners[2][1]
	yMin_xMatch		=	in4Corners[2][0]
	yMax			=	in4Corners[3][1]
	yMax_xMatch		=	in4Corners[3][0]

	// Naming convention for matrices are based on AX = B, X = A^-1B
	// A Matrix
```

```
          Make/O/D/N=(2,2) AMatrix
          AMatrix[][0] = { -slope, (1/slope)}  //note that Coef[0] = slope
          AMatrix[][1] = { 1, 1}

          //B Matrices
          Make/O/D/N=(2,1) BMidPt, BxMax, ByMax, BxMin, ByMin
          BMidPt          =         {yIntercept, (midY + midX*(1/slope))}
          BxMax           =         {yIntercept, (xMax_yMatch + xMax*(1/slope))}
          ByMax           =         {yIntercept, (yMax + yMax_xMatch*(1/slope))}
          BxMin           =         {yIntercept, (xMin_yMatch + xMin*(1/slope))}
          ByMin           =         {yIntercept, (yMin + yMin_xMatch*(1/slope))}

          Make/O/D/N=(2,1) X_MidPt, X_xMax, X_yMax, X_xMin, X_yMin

          //Answer matrix will be in the format of: [0] = x; [1] = y
          //Projection of Mid point onto trendline
          SolveLinearEq(AMatrix, BMidPt, X_MidPt)
          //Projection of XMin point onto trendline
          SolveLinearEq(AMatrix, BxMin, X_xMin)
          //Projection of XMax point onto trendline
          SolveLinearEq(AMatrix, BxMax, X_xMax)
          //Projection of YMin point onto trendline
          SolveLinearEq(AMatrix, ByMin, X_yMin)
          //Projection of YMax point onto trendline
          SolveLinearEq(AMatrix, ByMax, X_yMax)

          //Output wave contains the projected pts of the midpts, xmin, xmax, y min, and ymax onto the
          //trend line
          outPoints[][0] = {X_MidPt[0], X_xMin[0], X_xMax[0],   X_yMin[0], X_yMax[0]}
          outPoints[][1] = {X_MidPt[1], X_xMin[1], X_xMax[1], X_yMin[1], X_yMax[1]}

          Killwaves  BMidPt, BxMax, ByMax, BxMin, ByMin, AMatrix
          KillWaves X_MidPt, X_xMax, X_yMax, X_xMin, X_yMin
End


//This function takes in the wave containing the projected major points onto the trend line.
//The length of the majorAxis "a" is the greatest distance between the midpont and either the xmax or
//xmin coordinate. The output is return and thus the function should be set equal to something when called.
Function MajorAxisSelection(inWave)                                                    //MajorAxisSelection
          Wave inWave                                                         //Inputs
          Variable answer                                                     //Output

          //Pt1 - projMidPoint          Pt2 - projXmin     Pt3 - projXmax
          Variable x1, x2, x3, y1, y2, y3
          x1 = inWave[0][0]
          y1 = inWave[0][1]
          x2 = inWave[1][0]
          y2 = inWave[1][1]
          x3 = inWave[2][0]
          y3 = inWave[2][1]

          Variable dmax, dmin //Distances 3 & 2 are for xmax, xmin to midpt; respectively
          //could just find the distances using the x values
          dmax = sqrt((x3- x1)^2 + (y3 - y1)^2)
          dmin = sqrt((x2- x1)^2 + (y2 - y1)^2)

          //Selection of largest "a" for building an ellipse
          if (dmax > dmin)
                    answer = dmax
                    return answer
          elseif (dmin > dmax)
                    answer = dmin
                    return answer
          elseif (dmin == dmax)
                    answer = dmax
                    return answer
          endif
End


Function MakeCoefs(fourCorners, coefs)                                          //MakeCoefs
          Wave fourCorners                                              //Input
          Wave coefs                                                    //Output

          Variable phi_f, phi_b, phi_r, phi_l
          //Maximum lean angles
          phi_f     =         fourCorners[1][0]                              //xmax - x value
          phi_r     =         fourCorners[2][1]                              //ymin - y value
          phi_b     =         fourCorners[0][0]                              //xmin - x value
          phi_l     =         fourCorners[3][1]                             // ymax - y value

          //Variables used to solve the f_crit curve
          Variable A,B,C,D,E,G,H,J, I

          A         =         -(phi_l*phi_r)
          B         =         (phi_l*phi_r) * (phi_f + phi_b)
          C         =         -(phi_f*phi_b)
          D         =         (phi_f*phi_b)* (phi_l + phi_r)
          E         =         phi_l*phi_r*phi_f*phi_b
          G         =         phi_b + phi_f
          H         =         phi_l + phi_r
          J         =         -(phi_l + phi_r)*(phi_b + phi_f)
          I         =         0.3

          //Output:
          coefs = {A,B,C,D,E,G,H, I, J}
End


//Function that makes the points for graphing an ellipse. Uses A*R =A'
//ROTATION IN CCW
Function MakeEllipse2(midX, midY, a,b, phi, fitElliX, fitElliY)                  //MakeEllipse2
          Variable midX, midY, a, b, phi                                //Inputs
          Wave fitElliX, fitElliY                                       //Outputs

          Variable length = numPnts(fitElliY)
          //Creating the pnts for the ellipse
          variable theta, k
          theta = 0.0
```

```
	for(k=0; k<length; k+=1)
		fitElliX[k]		= midX  + a*cos(theta)*cos(phi)	-		b*sin(theta)*sin(phi)
		fitElliY[k]		= midY  + a*cos(theta)*sin(phi)	+		b*sin(theta)*cos(phi)
		theta += 2*Pi/(length-1)
	endfor

End


//This function takes the inXw (Xwave) and inYw (Ywave) and calculates the linear regression line of
//the data. The output of this function is trendOut matrix which contains 2 pts to create the trend line
//and a wave containing  the coefs of the linear equation of y = mx +b. [0] = m (slope);
//[1] = b (y-intercept). In outTrend, the point in row[0] is the min point and row[1] is the max point
Function MakeLinearTrend(inXw, inYw, outTrend, outCoef)				//MakeLinearTrend
	Wave inXw, inYw								//Inputs
	Wave outTrend, outCoef							//Outputs

	//Prep work for using Curvefit
	variable length = numpnts(inYw)
	Variable trendM, trendB // think y = mx + b
	Make/O/D/N=(length) fitting
	Make/O/D/N=2 tempCoef

	CurveFit/Q/NTHR=0 line  kwCWave=tempCoef inYw /X= inXw /D= fitting
	wave W_sigma //ignore wave, it is killed at the end of the program

	//Prep work for putting a fitted line onto the graph, Column 0: X values, Column 1: Y values
	outTrend[0][1] = WaveMin(fitting)
	outTrend[1][1] = WaveMax(fitting)
	outTrend[0][0] = (outTrend[0][1] - tempCoef[0])/tempCoef[1]
	outTrend[1][0] = (outTrend[1][1] - tempCoef[0])/tempCoef[1]

	//Switching the position of the coefficients in the wave such that [0] - slope, [1]-y-intercept
	trendM = tempCoef[1]
	trendB = tempCoef[0]
	//printf "Slope = %.4f\r", trendM
	//printf "B = %.4f\r", trendB
	outCoef[0] = trendM
	outCoef[1] = trendB

	//Kill excess waves
	Killwaves fitting, tempCoef, W_sigma
End
```

```
//This function calculates the minor axis of the ellipse based on a single point. That single point is chosen
// from the inMatrix [aka orig4Corners]. The output is returned and thus the function should be set equal
//to something when called.
Function MinorAxisSelection( inMatrix, MidX, MidY, majorAxis, angle)          //MinorAxisSelection
          Wave inMatrix                                                   //Input
          Variable midX, midY, majorAxis, angle                           //Inputs
          variable answer                                                 //Output

          //Set up the possible points to be used to calculate minor axis length "b"
          Make/O/D/N=2 Ymin, Ymax
          //column[0] - xvals           column[1] - yvals
          Ymin     =          {inMatrix[2][0] , inMatrix[2][1]}
          Ymax     =          {inMatrix[3][0] , inMatrix[3][1]}

          //From the equation of an ellipse: (lamdba/a)^2 + (eta/b)^2 = 1, with center at (h,k) and angle
          // wrt xaxis of phi
          //lambda = (x-h)cos(phi) + (y-k)sin(phi)
          //eta        = (x-h)sin(phi)  - (y-k)cos(phi)
          Make/O/D/N=2  temp1, temp2

          //Function
          FindLambdaAndEta(Ymin, MidX, MidY, angle, temp1)
          variable lambda1, eta1, b1//,  b1a
          eta1 = temp1[0]
          lambda1 = temp1[1]
          b1= majorAxis*eta1*sqrt(-1/(lambda1^2-majorAxis^2))

          FindLambdaAndEta(Ymax, MidX, MidY, angle, temp2)
          variable lambda2, eta2, b2
          eta2 = temp2[0]
          lambda2 = temp2[1]
          b2 = majorAxis*eta2*sqrt(-1/(lambda2^2-majorAxis^2))

          if (numtype(b1)==2) //b1 is NaN
                    if (numtype(b2) == 0) //b2 is num
                              answer = b2
                    endif
          else //b1 is a number
                    if (numtype(b2) == 2) //b2 is NaN
                              answer = b1
                    else        // b1 &b2 are numbers
                              answer = max(b1,b2)
                    endif
          endif

          //Killwaves before returning answer
          Killwaves Ymin, Ymax, temp1, temp2
          return  answer
End
```

```
//This function is checking for how many points lay outside of the ellipse using the Analytic Eq for Ellipses
//Eq : AA*X^2 + BB*X*Y + CC*Y^2 + DD*X + EE*Y + FF = a^2*b^2
//Note that normally the equation is set to 0 and that the coef FF absorbs the a^2b^2 term
Function OptimizePntsOut(Xwave, Ywave, h, k, a, b, phi)                    //OptimizePntsOut
          Wave Xwave, Ywave                                          //Inputs
          Variable h, k, a, b, phi                                   //Inputs
          Variable count                                             //Output

          variable i, length
          length = numpnts(Xwave)
          count = 0

          Variable AA, BB, CC, DD, EE, FF
          AA = a^2*(sin(phi))^2 + b^2*(cos(phi))^2
          BB = 2*(b^2 - a^2)*sin(phi)*cos(phi)
          CC = a^2*(cos(phi))^2 + b^2*(sin(phi))^2
          DD = -2*AA*h - BB*k
          EE = -BB*h - 2*CC*k
          FF = AA*h^2 + BB*h*k + CC*k^2

          for (i = 0; i<length; i+=1)
                    variable temp
                    temp = AA*(Xwave[i])^2 + BB*Xwave[i]*Ywave[i] + CC*(Ywave[i])^2 +
DD*Xwave[i] + EE*Ywave[i] + FF

                    if (temp > a^2*b^2)
                              count +=1
                    endif
          endfor

          return count
End


//Function finds the perimeter of the convex hull
Function PerimeterCH(Xpts, Ypts)                                                 //PerimeterCH
          Wave Xpts, Ypts                                                  //Inputs
          Variable answer                                                  //Output
          Variable i, length, wrappt
          variable d = 0

          answer = 0
          length = numPnts(Xpts)
          for(i = 0; i<length - 1; i +=1)
                    d = sqrt((Xpts[i+1]-Xpts[i])^2 + (Ypts[i+1]-Ypts[i])^2)
                    answer = answer +d
          endfor

          wrappt = sqrt((Xpts[0]-Xpts[length-1])^2 + (Ypts[0]-Ypts[length-1])^2)
          answer = answer + wrappt
          return answer

End
```

```
//This function is checking for how many points lay outside of the ellipse using the Analytic Eq for Ellipses
//Eq : AA*X^2 + BB*X*Y + CC*Y^2 + DD*X + EE*Y + FF = a^2*b^2
//Note that normally the equation is set to 0 and that the coef FF absorbs the a^2b^2 term
Function PntCheck(Xwave, Ywave, h, k, a, b, phi, outsidePts)                    //PntCheck
        Wave Xwave, Ywave                                                //Inputs
        Variable h, k, a, b, phi                                         //Inputs
        Wave outsidePts                                                  //Output

        Variable count = 0
        variable i, length
        length = numpnts(Xwave)

        //Sets up variable's equation
        Variable AA, BB, CC, DD, EE, FF
        AA = a^2*(sin(phi))^2 + b^2*(cos(phi))^2
        BB = 2*(b^2 - a^2)*sin(phi)*cos(phi)
        CC = a^2*(cos(phi))^2 + b^2*(sin(phi))^2
        DD = -2*AA*h - BB*k
        EE = -BB*h - 2*CC*k
        FF = AA*h^2 + BB*h*k + CC*k^2

        //Loop is calculating if the point[i] is inside the ellipse or not. f point[i]>a^2*b^2 then the pt is
        // outside the ellipse and the point[i] is put into the wave to be returned
        for (i = 0; i<length; i+=1)
                variable temp
                temp = AA*(Xwave[i])^2 + BB*Xwave[i]*Ywave[i] + CC*(Ywave[i])^2 +
DD*Xwave[i] + EE*Ywave[i] + FF

                if (temp > a^2*b^2)
                        outsidePts[count][0] = Xwave[i]
                        outsidePts[count][1] = Ywave[i]
                        outsidePts[count][2] = temp

                        count +=1
                endif
        endfor

        //True  output is the number of points that fall outside of the ellipse
        return count
End


//This function is given a list of waves (using WaveList) in a string. Each wave in the string has a different
//number of pnts in the wave and the total sum of pnts from all waves in the list are counted. The sum
//(var: count) is returned. Should be used like: Variable pntCount = PntsInWaveList(string list)
Function PntsInWaveList(listOfWaves)                                            //PntsInWaveList
        String listOfWaves                                               //Inputs
        //Local variable set-up
        variable i, numOfwaves
        variable count = 0
        String tempName

        //Calculation of Pnts in all Waves in the passed listOfWaves
        numOfwaves = ItemsInList(listOfWaves)
        for (i=0; i<numOfwaves; i+=1)
```

```
                tempName = StringFromList(i, listOfWaves)
                count += numPnts($tempName)
        endfor
        return count                                                        //Ouput
End


// This functions separates the x values and their respective y coordinates by positive and negative values
// Once separated, the x and corresponding y coordinates are sorted in increasing order (pos x vals) and
// decreasing order (neg x vals). They are then  strung back together into the original wave format
Function ReorderWaves(x, y, j, k)                                                   //ReorderWaves
        Wave x, y                                                           //Output
        Variable j,k                                                        //input?

        printf "j = %d\t k= %d\r", j, k
        Make/O/D/N=(j) temp1x, temp1y
        Make/O/D/N=(k) temp2x, temp2y

        Variable i, leng
        i = 0
        j= 0
        k=0
        leng = numpnts(x)

        for (i=0; i<leng; i+=1)
                if(x[i] > 0 || x == 0 )
                        temp1x[j] = x[i]
                        temp1y[j] = y[i]
                        j+=1
                elseif(x[i] < 0)
                        temp2x[k] = x[i]
                        temp2y[k] = y[i]
                        k+=1
                endif
        endfor

        Sort temp1y, temp1y, temp1x
        Sort/R temp2y, temp2y, temp2x

        Concatenate/O/NP/KILL {temp1x, temp2x}, rawr
        Concatenate/O/NP/KILL {temp1y, temp2y}, duh
        x = rawr
        y = duh
        Killwaves rawr, duh
End
```

```
//Takes in 3 matrices A, B, and X. Where A and B are known and solving for X  --> X = A^-1*B
Function SolveLinearEq(A, B, X)                                                //SolveLinearEq
            Wave A, B                                                  //Inputs
            Wave X                                                     //Output
            //Built In Igor Pro Matrix solver
            MatrixLinearSolve/M=1 A  B
            Wave M_B, M_A //waves for Matrix Linear Solve to put answers into
            Duplicate/O/D M_B, X
            //Kills waves that are not needed outside of this function
            Killwaves M_A, M_B
End


//Takes the inputted waves and finds the number of positive and neg. x values. It then sorts the waves
//by increasing pos. x values and then decreasing order of neg. x values.
Function sortFcritCurve(inX, inY, outX, outY)                                  //SortFcritCurve
            Wave inX, inY                                              //Input
            Wave outX, outY                                            //Output

            Duplicate/O inX, inX2
            Make/O/D/N=2 output1
            //Finds how many positive x values and negative x values there are.
            determineWavesSize(inX, output1)
            //output = {pos. Xs, neg. Xs}
            Variable size1, size2
            size1 = output1[0]
            size2 = output1[1]
            Killwaves output1

            // Reorders the waves
            reorderWaves(inX, inY, size1, size2)
            outX = inX
            outY = inY

            Killwaves inX2
End
```

# APPENDIX B

# MATLAB CODE: DATA PARSING

SCRIPT FILE

```
%%----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Created on: Mar 08 2016
% Updated on: Mar 16 2016
% ---------------------------------------------------------------------
% Script Description:
% Function for importing and exporting data in a specific file format of
% Vicon data for forceplate data and marker set. This function will import
% two separate files of data with the endings 1)_FP and 2)_Markers. Each
% individual file only contains one type of data.
%
% The force plate import function (ImportTxt_FP.m) should be able to import
% any force plate data in txt form. The data is collected at 1000 Hz and is
% unfiltered.
% The marker import file(ImportTxt_Markers.m) will only correctly import the
% marker position for the LowerLimb_HBMN2 marker set only. The data is
% collected at 100 Hz.
%
% Files to be imported must be in the DataToBeProcessed folder and in the
% respective FP and Markers folder.
% FP folder ending:     _FP.txt
% Marker folder ending: _Markers.txt
%% ---------------------------------------------------------------------

%This number reflects how many files you want to import and export from a
%given folder.
numFiles = 1;
%This folder must be with in the Matlab directory i.e.
%/Users/vasmith/Documents/MATLAB/~ the rest of the folder information can
%be put at the end and into the variable pathname.

% Creates the prompt box and then assigns the inputted string to the
% variables: date, ID. Asking for date of the data collection and the
% subject's ID number.
prompt = {'Date data collected on:','Patient ID'};
title =  'Data Information';
lines = 1;
def = {'mm/dd/yy', ''}; %default text in edit box
answer = inputdlg(prompt, title, lines, def);
assignin('base', 'date', answer{1});
assignin('base', 'ID', answer{2});

for i = 1:numFiles
```

```
	%---------------------- Importing All Data ----------------------------
	%The settings below are specific for the Vicon files that have
	%forceplate data
	startRow = 6;
	endRow = inf;

	%Double check that below is the correct pathway to the desired data file;
	%change as needed
	%Office computer:
	'/Users/vasmith/Documents/MATLAB/Lean_0316/DataToBeProcessed/FP/'
	pathname_FP =
'/Users/vasmith/Documents/MATLAB/Lean_0316/DataToBeProcessed/FP/';
	pathname_M  =
'/Users/vasmith/Documents/MATLAB/Lean_0316/DataToBeProcessed/Markers/';

	%Determines the final file name
	if i<10
		text = 'lean0'; %Change as necessary
		count = num2str(i);
		ending1 = '_FP.txt';
		ending2 = '_Markers.txt';
		%file1 = strcat(text,count, ending1)
		%file2 = strcat(text,count, ending2)
		file1 = strcat(text,ending1)
		file2 = strcat(text,ending2)
		filename1 = strcat(pathname_FP, file1);
		filename2 = strcat(pathname_M, file2);
	else
		text = 'lean'; %Change as necessary
		count = num2str(i);
		ending1 = '_FP.txt';
		ending2 = '_Markers.txt';
		file1 = strcat(text,count, ending1)
		file2 = strcat(text,count, ending2)
		%file1 = strcat(text,ending1)
		%file2 = strcat(text,ending2)
		filename1 = strcat(pathname_FP, file1);
		filename2 = strcat(pathname_M, file2);
	end

	%Function that imports the  FP data, ImportTxt_FP, is specifically
	%designed to create the following variable vectors: Frame_FP,
	%SubFrame_FP - time steps; Fx,Fy,Fz - Forces given by the left force
	%plate; Fx1,Fy1,Fz1 - forces given by the right force plate
	[Frame_FP,SubFrame_FP,Fx,Fy,Fz,Fx1,Fy1,Fz1] = ImportTxt_FP(filename1,
startRow, endRow);

	%Function that imports the marker data, ImportTxt_Markers is
	%specifically designed to create variable matracies for each of the
	%markers and containing the x, y,z coordinates for that marker. It also
	%creates Frame_M and SubFrame_M - which are time step vectors
	[Frame_M, SubFrame_M, LASIS, RASIS, LPSIS, RPSIS, SACR, T10, STRN,
XYPH, NAVE, LGTRO, FLTHI, LLEK, LATI, LLM, LHEE, LTOE, LMT5, RGTRO,
FRTHI, RLEK, RATI, RLM, RHEE, RTOE, RMT5] = ImportTxt_Markers(filename2,
startRow, endRow);

	%--------------------- Prep for file Processing -----------------------

	% Creates the prompt box and then assigns the inputted string to the
	% variable: fileString
	prompt = {'Enter File Name:', 'Abbreviation:'};
	title =  file1;
	%title =  'Filename'; %for if not running for loop
	lines = 1; %edit lines
	def = {'enter name (no spaces)', ''}; %default text in edit box
	answer = inputdlg(prompt, title, lines, def);
	assignin('base', 'fileString', answer{1});
	assignin('base', 'abbr', answer{2});

	%Creates two file names with user entry from fileString
	type1 = '_thetas.txt';
	type2 = '_markers.txt';
	%Office computer: '/Users/vasmith/Documents/MATLAB/Lean_0316/Processed'
	%Laptop:'C:\Users\Victoria\Documents\MATLAB\postProcessing_v1\LeanExper
iments\Data\Processed\'
	%Thetas calculated from FP output file
	newPath = '/Users/vasmith/Documents/MATLAB/Lean_0316/Processed/';
	newFile1 =  strcat(newPath, fileString, type1);
	name1 = strcat(fileString, type1);
	%Marker output file
	newFile2 =  strcat(newPath, fileString, type2);
	name2 = strcat(fileString, type2);
%% ------------------------ Force Plate Processing ------------------------

	%Rotates the coordinate systeme to match the desired Greek coordinate
	%system (Zakythinaki et al. Chaos 2004)
	[Fx,Fy,Fz,Fx1,Fy1,Fz1] = RotateCoordSys(Fx,Fy,Fz,Fx1,Fy1,Fz1);

	%Filtering raw data using 4th order Butterworth lowpass filter.
	%Function filterData must have Fs, cutFreq, and 1 vector to filter and
```

```
        %upto 6 different vectors to pass through a 4th order Butterworth low
        %pass filter called butter4th: function [ filteredData ] =
        %butter4th(FreqS, FreqC, dataSet)
        Fs = 1000; %Hz - Sampling frequency
        cutFreq = 7; %Hz - cut off frequency
        [f_Fx, f_Fy, f_Fz, f_Fx1, f_Fy1, f_Fz1] = FilterFPData(Fs, cutFreq, Fx, Fy, Fz,
Fx1, Fy1, Fz1);

        %This function calculates the phase space from the given Vicon forces.
        %In addition this function changes the coordinate systems from the
        %Vicon coord system to the desired coordinate system (i.e. the Greek
        %Coordinate System)
        output = CalcFinalThetas(f_Fx, f_Fy, f_Fz, f_Fx1, f_Fy1, f_Fz1);

        %If you want all thetas (left, right, both feet) format = 1
        %If you only want final thetas format = 2
        format = 2;

        %output must be transposed to have a 6xN matrix for the makeThetaFiles
        %to properly put all the data correctly into a .txt file. The function
        %read down a coulmn and writes it in a row to create columns of data in
        %a .txt file.
        MakeThetaFiles_v3(format, newFile1, name1, abbr, date, ID, output');

%%--------------------- Marker Data Processing ----------------------------
        %Rotates the coordinate system to match the desired Greek coordinate
        %system (Zakythinaki et al. Chaos 2004) for the selected markers
        T10   = RotateCoordSys3D(T10);
        SACR  = RotateCoordSys3D(SACR);
        LGTRO = RotateCoordSys3D(LGTRO);
        LLEK  = RotateCoordSys3D(LLEK);
        LLM   = RotateCoordSys3D(LLM);
        LHEE  = RotateCoordSys3D(LHEE);
        LTOE  = RotateCoordSys3D(LTOE);
        RGTRO = RotateCoordSys3D(RGTRO);
        RLEK  = RotateCoordSys3D(RLEK);
        RLM   = RotateCoordSys3D(RLM);
        RHEE  = RotateCoordSys3D(RHEE);
        RTOE  = RotateCoordSys3D(RTOE);

        %No filteringis done

        %Calculating the angles in the XZ plane and YZ plane.
        [leftAngles]  = JointAngle2D(T10, SACR, LGTRO, LLEK, LLM, LHEE,
LTOE);
```

```
    [rightAngles] = JointAngle2D(T10, SACR, RGTRO, RLEK, RLM, RHEE,
RTOE);

    %outputs a .txt based on the path given in newFile2
    MakeMarkerFile_v2(newFile2, name2, abbr, date, ID, leftAngles, rightAngles)

end
```

FUNCTION FILES

(In alphabetical order)

```
function [ filteredData ] = Butter4th(FreqS, FreqC, dataSet)
%4th Order Butterworth filter with a cut off freqency of Wn. Where Wn =
%cutoff frequency desired divided by the half the sampling frequency. (This is done
%because Wn must be 0.0<Wn<1.0, with 1.0 corresponding to half the sample rate

        N = 4;
        Wn = FreqC/(FreqS/2);
        [b,a] = butter(N, Wn);
        filteredData = filter(b, a, dataSet);

end


function [ angle ] = Calc2DAngle( v1, v2 )
%DotProduct Caculates the angle between the 2 inputted vectors
%   INPUT:
%         v1, v2 : two vector matrix
%
%   OUTPUT:
%         angle
%
% Created by: Victoria A. Smith
% Created on: March 3rd, 2016
%Preassigning size of angle for faster computation:
angle = zeros(size(v1,1),1);

for i=1:size(v1,1)
        vector1 = [v1(i,:)]';
        vector2 = [v2(i,:)]';

        %in case there is no data
        if norm(vector1) ~= 0 && norm(vector2) ~= 0
                rawr =
180/pi*(acos(sum(vector1.*vector2)/(norm(vector1)*norm(vector2))));
        else
                rawr = 0;
        end

        angle(i) = rawr;
end
end
```

```
function  [output ] = CalcFinalThetas( l_Fx, l_Fy, l_Fz, r_Fx, r_Fy, r_Fz )
%%calcThetas %This function calculates the phase space from the given
%forces. The forces inputted into this function must be in the final
%desired global coordinate system
%   INPUTS:
% Forces in the x,y,z direction from left (l_F) and right (r_F) force
% plates
%
%   EQUATIONS:
% thetaX = atan(x/-z)
% thetaY = atan(y/-z)
% Note: all of the data in the z-axis is negative in these equations because we
% were given force applied to the forceplate and we need ground reaction
% force, which is the negative of the force applied to the force plate.
%
%   OUTPUTS:
% output - is a matrix of the thetas calculated for the left, right force
% plate as well as the thetas calculated to mimic a single force plate
% NOTE: The output of this file must be all the data extending down rows
% with only 6 columns
%----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Created on: Jan 12 2016
% Updated on: Jan 18 2016
% ---------------------------------------------------------------------

% left foot:
l_thetaX = atan2(l_Fx, -l_Fz);
l_thetaY = atan2(l_Fy, -l_Fz);
% right foot:
r_thetaX = atan2(r_Fx, -r_Fz);
r_thetaY = atan2(r_Fy, -r_Fz);

% together - this is the REAL PHASE SPACE we need for further calculations
% First sum all of the forces together
fx = l_Fx + r_Fx;
fy = l_Fy + r_Fy;
fz = l_Fz + r_Fz;
% Then calculate the theta's
thetaX = atan2(fx, -fz); %Nx1,
thetaY = atan2(fy, -fz);

output = [ l_thetaX, l_thetaY, r_thetaX, r_thetaY, thetaX, thetaY ];

end
```

```
function [ out1, out2, out3, out4, out5, out6 ] = FilterFPData( Fs, Fc, d1, d2, d3, d4, d5,
d6)
%filterData
%   Faster way to filter many sets of data at once
% Fs - sampling frequency
% Fc - cutoff Frequency
% d1 - d6 - data sets, dim =1
% butter4th - 4th order Butterworth lowpass filter

% Fs, Fc, d1 are mandatory
% d2 - d6 are optional
if nargin < 4
        out1 = Butter4th(Fs, Fc, d1);

elseif nargin < 5
        out1 = Butter4th(Fs, Fc, d1);
        out2 = Butter4th(Fs, Fc, d2);

elseif nargin < 6
        out1 = Butter4th(Fs, Fc, d1);
        out2 = Butter4th(Fs, Fc, d2);
        out3 = Butter4th(Fs, Fc, d3);

elseif nargin < 7
        out1 = Butter4th(Fs, Fc, d1);
        out2 = Butter4th(Fs, Fc, d2);
        out3 = Butter4th(Fs, Fc, d3);
        out4 = Butter4th(Fs, Fc, d4);

elseif nargin < 8
        out1 = Butter4th(Fs, Fc, d1);
        out2 = Butter4th(Fs, Fc, d2);
        out3 = Butter4th(Fs, Fc, d3);
        out4 = Butter4th(Fs, Fc, d4);
        out5 = Butter4th(Fs, Fc, d5);

elseif nargin < 9
        out1 = Butter4th(Fs, Fc, d1);
        out2 = Butter4th(Fs, Fc, d2);
        out3 = Butter4th(Fs, Fc, d3);
        out4 = Butter4th(Fs, Fc, d4);
        out5 = Butter4th(Fs, Fc, d5);
        out6 = Butter4th(Fs, Fc, d6);
end
end
```

```
 function [Frame_FP,SubFrame_FP,Fx,Fy,Fz,Fx1,Fy1,Fz1] = ImportTxt_FP(filename,
startRow, endRow)
%ImportTxt_FP Import numeric data from a text file as column vectors.
%This version of importfile is explicitly for extracting the forces and
%frame rates from the inputted Vicon txt file. That Vicon file can ONLY
%contain forceplate data. No marker data can be in the file. A different
%importfile file must be used (ImportTxt_Markers.m)
%
%   [FRAME,SUBFRAME,FX,FY,FZ,FX1,FY1,FZ1] = IMPORTFILE1(FILENAME,
STARTROW,
%   ENDROW) Reads data from rows STARTROW through ENDROW of text file
%   FILENAME.
%

% Auto-generated by MATLAB on 2016/01/12 10:55:53
% Revised by Victoria Smith on 2016/03/08
%% Initialize variables.
delimiter = '\t';
if nargin<=2
        startRow = 6;
        endRow = inf;
end

%% Format string for each line of text:
%   column1: double (%f)
%   column2: double (%f)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column12: double (%f)
%   column13: double (%f)
%   column14: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec =
'%f%f%f%f%f%*s%*s%*s%*s%*s%*s%f%f%f%*s%*s%*s%*s%*s%*s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
textscan(fileID, '%[^\n\r]', startRow(1)-1, 'ReturnOnError', false);
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter',
delimiter, 'EmptyValue' ,NaN,'ReturnOnError', false);
for block=2:length(startRow)
        frewind(fileID);
        textscan(fileID, '%[^\n\r]', startRow(block)-1, 'ReturnOnError', false);
        dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-
startRow(block)+1, 'Delimiter', delimiter, 'EmptyValue' ,NaN,'ReturnOnError', false);
        for col=1:length(dataArray)
                dataArray{col} = [dataArray{col};dataArrayBlock{col}];
        end
end

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
Frame = dataArray{:, 1};
SubFrame = dataArray{:, 2};
Fx = dataArray{:, 3};
Fy = dataArray{:, 4};
Fz = dataArray{:, 5};
Fx1 = dataArray{:, 6};
Fy1 = dataArray{:, 7};
Fz1 = dataArray{:, 8};

Frame_FP = Frame;
SubFrame_FP = SubFrame;

end
```

```
function [Frame_M, SubFrame_M, LASIS, RASIS, LPSIS, RPSIS, SACR, T10, STRN,
XYPH, NAVE, LGTRO, FLTHI, LLEK, LATI, LLM, LHEE, LTOE, LMT5, RGTRO,
FRTHI, RLEK, RATI, RLM, RHEE, RTOE, RMT5] = ImportTxt_Markers(filename,
startRow, endRow)
%ImportTxt_Markers Import numeric data from a text file as column vectors.
%  This file takes the marker data from the LowerLimb_HBMN2 marker set in
%  from Vicon Nexus 2.2. Only this specific marker set can be imported
%  correctly, other import files would need to be created for a different
%  marker set.
%
%   This marker set contains 25 markers, concentrating on the Lower Limb
%   capture. The data is collected at 100Hz.
%
%   [Frame_M, SubFrame_M, LASIS, RASIS, LPSIS, RPSIS, SACR, T10, STRN,
XYPH, NAVE, LGTRO, FLTHI, LLEK, LATI, LLM, LHEE, LTOE, LMT5, RGTRO,
FRTHI, RLEK, RATI, RLM, RHEE, RTOE, RMT5]
%   = ImportTxt_Markers(FILENAME, STARTROW, ENDROW) Reads data from rows
STARTROW
%   through ENDROW of text file FILENAME.
%

% Auto-generated by MATLAB on 2016/03/08 10:36:51
% Adapted by Victoria Smith on 2016/03/08

%% Initialize variables.
delimiter = '\t';
if nargin<=2
        startRow = 6;
        endRow = inf;
end

%% Format string for each line of text:
%   column1 - 26: double (%f)
%   column27 - 29: text (%s)
%   column30 - 77: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec =
'%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%s%s%s%f
%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%
f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
```

```
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter',
delimiter, 'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
        frewind(fileID);
        dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-
startRow(block)+1, 'Delimiter', delimiter, 'HeaderLines', startRow(block)-1,
'ReturnOnError', false);
        for col=1:length(dataArray)
                dataArray{col} = [dataArray{col};dataArrayBlock{col}];
        end
end

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
Frame = dataArray{:, 1};
SubFrame = dataArray{:, 2};
X = dataArray{:, 3};
Y = dataArray{:, 4};
Z = dataArray{:, 5};
X1 = dataArray{:, 6};
Y1 = dataArray{:, 7};
Z1 = dataArray{:, 8};
X2 = dataArray{:, 9};
Y2 = dataArray{:, 10};
Z2 = dataArray{:, 11};
X3 = dataArray{:, 12};
Y3 = dataArray{:, 13};
Z3 = dataArray{:, 14};
X4 = dataArray{:, 15};
Y4 = dataArray{:, 16};
Z4 = dataArray{:, 17};
X5 = dataArray{:, 18};
Y5 = dataArray{:, 19};
Z5 = dataArray{:, 20};
```

```
X6 = dataArray{:, 21};
Y6 = dataArray{:, 22};
Z6 = dataArray{:, 23};
X7 = dataArray{:, 24};
Y7 = dataArray{:, 25};
Z7 = dataArray{:, 26};
X8 = dataArray{:, 27};
Y8 = dataArray{:, 28};
Z8 = dataArray{:, 29};
X9 = dataArray{:, 30};
Y9 = dataArray{:, 31};
Z9 = dataArray{:, 32};
X10 = dataArray{:, 33};
Y10 = dataArray{:, 34};
Z10 = dataArray{:, 35};
X11 = dataArray{:, 36};
Y11 = dataArray{:, 37};
Z11 = dataArray{:, 38};
X12 = dataArray{:, 39};
Y12 = dataArray{:, 40};
Z12 = dataArray{:, 41};
X13 = dataArray{:, 42};
Y13 = dataArray{:, 43};
Z13 = dataArray{:, 44};
X14 = dataArray{:, 45};
Y14 = dataArray{:, 46};
Z14 = dataArray{:, 47};
X15 = dataArray{:, 48};
Y15 = dataArray{:, 49};
Z15 = dataArray{:, 50};
X16 = dataArray{:, 51};
Y16 = dataArray{:, 52};
Z16 = dataArray{:, 53};
X17 = dataArray{:, 54};
Y17 = dataArray{:, 55};
Z17 = dataArray{:, 56};
X18 = dataArray{:, 57};
Y18 = dataArray{:, 58};
Z18 = dataArray{:, 59};
X19 = dataArray{:, 60};
Y19 = dataArray{:, 61};
Z19 = dataArray{:, 62};
X20 = dataArray{:, 63};
Y20 = dataArray{:, 64};
Z20 = dataArray{:, 65};
```

```
X21 = dataArray{:, 66};
Y21 = dataArray{:, 67};
Z21 = dataArray{:, 68};
X22 = dataArray{:, 69};
Y22 = dataArray{:, 70};
Z22 = dataArray{:, 71};
X23 = dataArray{:, 72};
Y23 = dataArray{:, 73};
Z23 = dataArray{:, 74};
X24 = dataArray{:, 75};
Y24 = dataArray{:, 76};
Z24 = dataArray{:, 77};

 %% Allocate each marker variable into a matrix for each marker containing
%those variables
 % marker = [x_position, y_position, z_position];
LASIS   = [X, Y, Z];
RASIS   = [X1, Y1, Z1];
LPSIS   = [X2, Y2, Z2];
RPSIS   = [X3, Y3, Z3];
SACR  = [X4, Y4, Z4];
T10      = [X5, Y5, Z5];
STRN  = [X6, Y6, Z6];
XYPH = [X7, Y7, Z7];
NAVE = [X8, Y8, Z8];
LGTRO   = [X9, Y9, Z9];
FLTHI   = [X10, Y10, Z10];
LLEK  = [X11, Y11, Z11];
LATI   = [X12, Y12, Z12];
LLM    = [X13, Y13, Z13];
LHEE  = [X14, Y14, Z14];
LTOE  = [X15, Y15, Z15];
LMT5  = [X16, Y16, Z16];
RGTRO   = [X17, Y17, Z17];
FRTHI   = [X18, Y18, Z18];
RLEK  = [X19, Y19, Z19];
RATI   = [X20, Y20, Z20];
RLM    = [X21, Y21, Z21];
RHEE  = [X22, Y22, Z22];
RTOE  = [X23, Y23, Z23];
RMT5 = [X24, Y24, Z24];
 %Renaming Frame and Subframe
Frame_M = Frame;
SubFrame_M = SubFrame;
 end
```

```
function [angleStruct] = JointAngle2D( T10, SACR, Hip, Knee, Ankle, Heel, Toe)
%JointAngle2D Calculating the Joint Angle in XZ and YZ planes
%   Inputs: are matrices of marker locations for an entire movement. Each
%   matrix is Nx3 with columns of x, y, z coordinates
%
%   Outputs: Two structs that contain the joint angles at the Hip, Knee,
%   and Ankle in the XZ and YZ plane, respectively
%----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Updated on: Mar 17 2016
% ---------------------------------------------------------------------

%Calculating the hip, knee, and ankle angles in the XZ (frontal) plane
hip_AP          = Calc2DAngle(T10(:,1:2:3)  - SACR(:,1:2:3),  Knee(:,1:2:3) -
Hip(:,1:2:3));
knee_AP   = Calc2DAngle(Hip(:,1:2:3)  - Knee(:,1:2:3), Ankle(:,1:2:3) - Knee(:,1:2:3));
ankle_AP  = Calc2DAngle(Knee(:,1:2:3) - Ankle(:,1:2:3), Heel(:,1:2:3) - Toe(:,1:2:3));

%Calculating the hip, knee, and ankle angles in the XZ (sagittal) plane
hip_ML          = Calc2DAngle(T10(:,2:3)  - SACR(:,2:3),  Knee(:,2:3)  - Hip(:,2:3));
knee_ML   = Calc2DAngle(Hip(:,2:3)  - Knee(:,2:3),  Ankle(:,2:3) - Knee(:,2:3));
ankle_ML  = Calc2DAngle(Knee(:,2:3) - Ankle(:,2:3), Heel(:,2:3)  - Toe(:,2:3));

angleStruct = struct('hipAP', hip_AP, 'kneeAP', knee_AP, 'ankleAP', ankle_AP, 'hipML',
hip_ML, 'kneeML', knee_ML, 'ankleML', ankle_ML);

end
```

```
function MakeMarkerFile_v2( filepath, title, nickName, date, ID, dataInL, dataInR)
%makeThetaFiles - This function creates .txt files from the input The
%function reads down a coulmn in the data matrix and writes it in a row of
%the .txt file to create columns of data in a .txt file.
%
%Inputs:
%filepath - contains the path that the file should be saved in and includes
%                    the desired name of the file
%title   - is the name of the file to be put in the header of the .txt file
%nickname - is the abbrevation of the title that is used to label each row
%date    - is the date the data was taken on and put into the header of the
%                    .txt file
%ID       - is the Patient ID number
%dataInL  - is a struct with 6 joint angles: hip, knee, ankle in the XZ and
%                    YZ plane, respectively. These are the angles from the left leg
%dataInR  - is a struct with 6 joint angles: hip, knee, ankle in the XZ and
%                    YZ plane, respectively. These are the angles from the right leg
%Output:
%the output of this function is the creation of a .txt file in a folder
%given by filepath, with the name of title and header (contains title and
%date) and the desired data in 12 columns
%-----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Updated on: Mar 17 2016
% ---------------------------------------------------------------------
% This is setting up the inputs into the header file
date = strcat('Data taken on: ', date);
patient = strcat('Patient ID: ', ID);

a = strcat(nickName,'_lhAP');
b = strcat(nickName,'_lkAP');
c = strcat(nickName,'_laAP');
d = strcat(nickName,'_lhML');
e = strcat(nickName,'_lkML');
f = strcat(nickName,'_laML');
g = strcat(nickName,'_rhAP');
h = strcat(nickName,'_rkAP');
i = strcat(nickName,'_raAP');
j = strcat(nickName,'_rhML');
k = strcat(nickName,'_rkML');
l = strcat(nickName,'_raML');

%Break apart the inputted structure
%They are all transposed from Nx1 vectors to 1xN vectors
L_hip_AP   = dataInL.hipAP';
```

```
L_knee_AP  = dataInL.kneeAP';
L_ankle_AP = dataInL.ankleAP';
L_hip_ML   = dataInL.hipML';
L_knee_ML  = dataInL.kneeML';
L_ankle_ML = dataInL.ankleML';

R_hip_AP   = dataInR.hipAP';
R_knee_AP  = dataInR.kneeAP';
R_ankle_AP = dataInR.ankleAP';
R_hip_ML   = dataInR.hipML';
R_knee_ML  = dataInR.kneeML';
R_ankle_ML = dataInR.ankleML';

%Create a matrix that contains the angle data.
%data1 should be a 12xN matrix
data1 = [L_hip_AP;
L_knee_AP;L_ankle_AP;L_hip_ML;L_knee_ML;L_ankle_ML;R_hip_AP; R_knee_AP;
R_ankle_AP;R_hip_ML; R_knee_ML; R_ankle_ML];

%Create an output file
fileID = fopen(filepath, 'w+');
fprintf(fileID, '%s\n', title);
fprintf(fileID, '%s\t %s\n', date, patient);
fprintf(fileID, '%s %12s %12s %12s %12s %12s %12s %12s %12s %12s %12s %12s\n',
a, b, c, d, e, f, g, h, i, j, k, l);
fprintf(fileID,
'%12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %
12.6f\n', data1);


end
```

```
function MakeThetaFiles_v3( format, filepath, title, nickName, date, ID, dataIn )
%makeThetaFiles - This function creates .txt files from the input The
%function reads down a coulmn in the data matrix and writes it in a row of
%the .txt file to create columns of data in a .txt file.
%
%Inputs:
%format   - int = 1 or 2.
%                    1: writes all thetas to .txt
%                    2: writes only real phase space thetas to .txt
%filepath - contains the path that the file should be saved in and includes
%                    the desired name of the file
%title  - is the name of the file to be put in the header of the .txt file
%nickname - is the abbrevation of the title that is used to label each row
%date   - is the date the data was taken on and put into the header of the
%                    .txt file
%ID       - is the Patient ID number
%data   - is a matrix of Nx6 dimensions
%Output:
%the output of this function is the creation of a .txt file in a folder
%given by filepath, with the name of title and header (contains title and
%date) and the desired data in 6 columns
%-----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Updated on: Mar 17 2016
% ---------------------------------------------------------------------

%The following strings will be the column names in the .txt
%Ex: if nickName = f1 --> e = f1_X
%It is critical for all of the columns to have unique names for igor pro to
%read and create the corresponding waves for further data analysis
a = strcat(nickName,'_lX');
b = strcat(nickName,'_lY');
c = strcat(nickName,'_rX');
d = strcat(nickName,'_rY');
e = strcat(nickName,'_X');
f = strcat(nickName,'_Y');


if format == 1
        patient = strcat('Patient ID: ', ID);
        date = strcat('Data taken on: ', date);

        fileID = fopen(filepath, 'w+');
        fprintf(fileID, '%s\n', title);
        fprintf(fileID, '%s\t %s\n', patient, date);
        fprintf(fileID, '%s %12s %12s %12s %12s %12s\n', a, b, c, d, e, f);
        fprintf(fileID, '%.8f %12.8f %12.8f %12.8f %12.8f %12.8f\n', dataIn);

else %format == 2
        patient = strcat('Patient ID: ', ID);
        date = strcat('Data taken on: ', date);

        myX = dataIn(5,:);
        myY = dataIn(6,:);
        myThetas = [myX; myY];
        fileID = fopen(filepath, 'w+');
        fprintf(fileID, '%s\n', title);
        fprintf(fileID, '%s\t %s\n', patient, date);
        fprintf(fileID, '%s %12s\n', e, f);
        fprintf(fileID, '%.8f %12.8f\n', myThetas);

end

end
```

```
function [ outFx,outFy,Fz,outFx1,outFy1,Fz1] = RotateCoordSys(inFx,inFy,
Fz,inFx1,inFy1,Fz1)
%rotateCoordSys - this function transforms the inputted data from the Vicon
%Nexus system into the desired (Greek) global coordinate system. Only the x
%an y axes are being rotated by 90 degrees CCW (right hand rule).
%   INPUTS:
% Forces - Left:          inFx,   inFy,   Fz
%                 - Right:   inFx1,  inFy1,  Fz1
%   *all of these are single vectors (nx1) and not matrices
%   ROTATION:
% We need to rotate the x and y axis 90 degress about the z axis, i.e.:
%          VICON                  =                        GREEK
%                Y                          =                                   -X
%                X                          =                                    Y
%                Z                          =                                    Z
% calculations for thetas in new coordinate system done on 011516
%   OUTPUTS:
%  Forces - Left:         outFx,   outFy,   Fz
%                  - Right:   outFx1,  outFy1,  Fz1
%----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Updated on: Jan 18 2016
% ---------------------------------------------------------------------


%rotation matrix set up
ang = 90;
rotMatrix = [ cosd(ang) -sind(ang); sind(ang) cosd(ang)];

%Rotate data
dataRot  = [inFx,  inFy] *rotMatrix;
dataRot1 = [inFx1, inFy1]*rotMatrix;

outFx   = dataRot(:,1);
outFy   = dataRot(:,2);
outFx1  = dataRot1(:,1);
outFy1  = dataRot1(:,2);

end
```

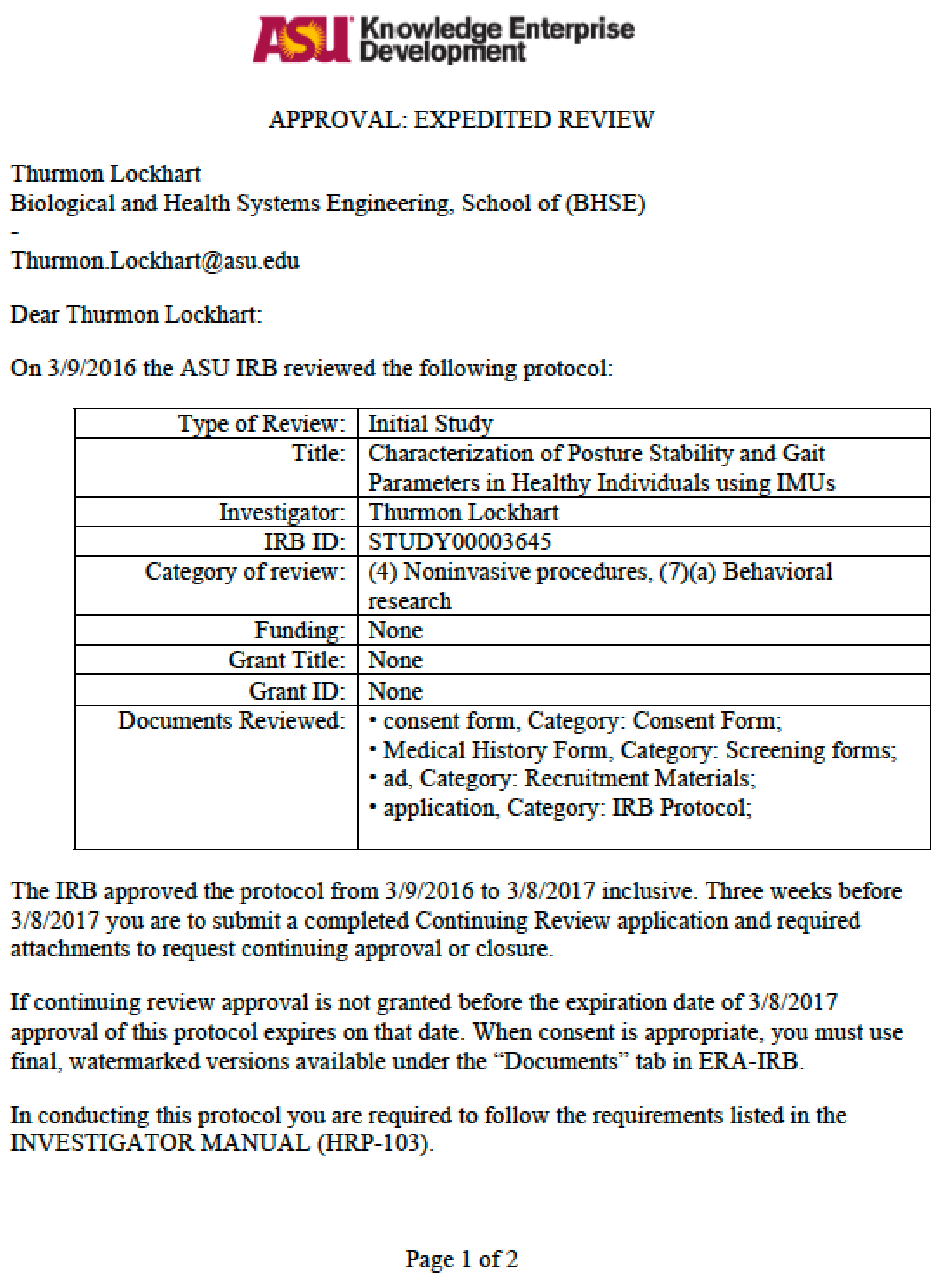
```
function [ output ] = RotateCoordSys3D( inMatrix )
%rotateCoordSys3D - this function transforms the inputted data from the Vicon
%Nexus system into the desired (Greek) global coordinate system. Only the x
%an y axes are being rotated by 90 degrees CCW (right hand rule).
%   INPUTS:
%           inMatrix - matrix data (Nx3), columns: x, y, z
%   ROTATION:
% We need to rotate the x and y axis 90 degress about the z axis, i.e.:
%           VICON                  =                        GREEK
%                 Y                        =                                   -X
%                 X                        =                                    Y
%                 Z                        =                                    Z
% calculations for thetas in new coordinate system done on 011516
%   OUTPUTS:
%           output - rotated data
%----------------------------------------------------------------------
% Created by: Victoria Smith, vasmith5@asu.edu
% Updated on: Jan 18 2016
% ----------------------------------------------------------------------
%Preassigning size of output for faster computation:
L = length(inMatrix);
output = zeros(L,3);

%rotation matrix set up
%This rotates the coord system 90 degrees about the z-axis
ang = 90;
rotMatrix = [ cosd(ang) -sind(ang) 0; sind(ang) cosd(ang) 0; 0 0 1];

%Rotating each point array in the inMatrix (by rows)
for i = 1:L
        temp = rotMatrix * inMatrix(i,:)'; %each row has to be transposed b/c: 3x3 x 3x1
        output(i,:) = temp';
end
```

## APPENDIX C

## IRB APPROVAL



APPROVAL: EXPEDITED REVIEW

Thurmon Lockhart
Biological and Health Systems Engineering, School of (BHSE)
-
Thurmon.Lockhart@asu.edu

Dear Thurmon Lockhart:

On 3/9/2016 the ASU IRB reviewed the following protocol:

| | |
|---|---|
| Type of Review: | Initial Study |
| Title: | Characterization of Posture Stability and Gait Parameters in Healthy Individuals using IMUs |
| Investigator: | Thurmon Lockhart |
| IRB ID: | STUDY00003645 |
| Category of review: | (4) Noninvasive procedures, (7)(a) Behavioral research |
| Funding: | None |
| Grant Title: | None |
| Grant ID: | None |
| Documents Reviewed: | • consent form, Category: Consent Form;<br>• Medical History Form, Category: Screening forms;<br>• ad, Category: Recruitment Materials;<br>• application, Category: IRB Protocol; |

The IRB approved the protocol from 3/9/2016 to 3/8/2017 inclusive. Three weeks before 3/8/2017 you are to submit a completed Continuing Review application and required attachments to request continuing approval or closure.

If continuing review approval is not granted before the expiration date of 3/8/2017 approval of this protocol expires on that date. When consent is appropriate, you must use final, watermarked versions available under the "Documents" tab in ERA-IRB.

In conducting this protocol you are required to follow the requirements listed in the INVESTIGATOR MANUAL (HRP-103).

Page 1 of 2

Sincerely,

IRB Administrator

cc:
Rahul Soangra
Saba Rezvanian
Victoria Smith