Hot Air Balloon Navigation

A Senior Project

presented to

the Faculty of the Aerospace Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Dustin Blackwell

December, 2010

Hot Air Balloon Navigation

Dustin Blackwell
California Polytechnic State University, San Luis Obispo, Ca, 93401

This report describes a program used for navigating a hot air balloon. The program, *Balloon_Trip*, was written using MATLAB and gives a flight path to follow from a start position to an end position. Balloon_Trip calculates the flight path by taking in wind conditions and then flying through these different winds so as steer the hot air balloon. The program calculates the flight path by taking into consideration at all times how the wind will propel the balloon while it is rising or falling in elevation. It then takes the most direct and least complicated, if not fastest, route from the starting location to the ending location. All of the flight paths chosen are segmented into five parts, two segments that move the hot air balloon strictly horizontal and three in which the balloon ascends or descends to the specific elevations in which the horizontal movement segments occur.

I. Introduction

Hot air balloons were first used to take to the sky's in 1780 (Briggs, 1986). Since then, both a better understanding of how hot Air Balloons fly and newer technology have brought about advances in ballooning.

Hot Air balloons fly because of the heated air that is trapped in the Envelope, the "balloon" part of the Hot Air Balloon that allows the craft to take flight. (Owen, 1999) Air that is heated up expands, causing it's density to lower. When this happens, the hot air (low density) rises above that of the cold air (higher density). When greater thought and design was put into making the Envelope lighter, it allowed the balloon to fly higher on the same amount of fuel simply because now it didn't have to carry as much of a payload attached to it.

Hot air balloons were first flown using smoke to power their flight: or at least that was what those flying thought was powering the balloon (Briggs, 1986). It wasn't until the 1960s that the smoky fuels used to power the balloon were replaced with a propane burner. Propane was a far less expense way to power the balloon, but it wasn't until progress was made on the Envelope of the balloon as well that Ballooning became as popular as it did. Making the Envelope using better and lighter materials allowed easier and less expensive flights.

The Envelope of a Balloon is composed of a number of sections that are held together with a series of horizontal and vertical reinforcing tape (Owen, 1999). This helps greatly to alleviate the stress that the balloon holds and instead transfers it to the lines of tape running along the balloon, which allows the fabric to be designed under less intense conditions to make it lighter and more durable.

The standard shape of the balloon, that of an inverted teardrop, was found by studying the effects that a payload would have on a spherical gas balloon. By seeing how the balloon would distort by having a payload connected to the underside of, it was seen that the balloon would naturally want to take this inverted teardrop shape. This also helped the placement of the burners, as it gave them an adequate opening in which to heat up the air inside the envelope.

Hot air balloons are flown using only the burners to control the flight elevation. The balloon travels by being pushed around by the wind blowing. If the wind is blowing to the north, then the hot air balloon riding in that wind will be blown north. The only manner in which a hot air balloon controls its direction of travel is by rising or falling in elevation until it comes upon a different wind blowing. And it is this that forms the basis of my navigation code.

My navigation program is built on the basis that a hot-air balloon can only directly control its elevation. By changing its elevation, it will encounter different winds that blow in different directions and at different speeds. By flying through these winds, the hot air balloon can be ultimately steered to go in the desired direction. If knowledge of the weather is known beforehand, it is possible to completely map out a trajectory to follow to get from a starting location to an ending location. My MATLAB program, *Balloon_Trip*, does just that.

II. Program Procedure

My program has gone through three stages. The first stage was mainly for getting the program set-up correctly, but also to get the structure of the program built. The second stage added most of the complexity to the program, mainly in the form of accounting for the wind blowing the balloon around as it rose or fell in the air during the first and last elevation changes in the trip. The third stage added the finishing touches, as well as the movement from the wind during the 2nd elevation change.

There are 8 inputs for *Balloon_Trip*. The first 4 are control values that turn on or off different settings. The next 4 are the true function inputs that the code uses to calculate possible trip trajectories. Details on the program's 8 inputs can be found in Table 1. Details on the 6 outputs can be found in Table 2.

Table 1: List of the Input Variables used in *Balloon_Trip*.

Input	Matrix Size	Description
Graph	1x1	Turns graphing on or off
Fig_Num	1x5	Labels the 5 graphs according to the inputted values
Display	1x1	Turns various display settings on or off
Random	4x3	Matrix used to create random data within the program
Wind	Nx3	Variable Sized Matrix that contains Wind Speed, Wind Direction, and Wind Elevation
Start	1x3	Matrix containing the start location in 3-Dimensions
End	1x3	Matrix containing the end location in 3-Dimensions
a	1x2	Matrix containing the rising and falling acceleration of the hot-air balloon

Table 2: List of Variables outputted from *Balloon_Trip*.

Output	Description
Total_Distance	The total Distance traveled by the hot-air balloon during the trip
Coordinates	Matrix containing all the coordinates of the trip
Wind	Matrix containing the Wind Specifications used in the program; Identical to the Wind input, except when using the Random data generator
Start	Matrix containing the Start Specifications used in the program; Identical to the Start input, except when using the Random data generator
End	Matrix containing the Wind Specifications used in the program; Identical to the End input, except when using the Random data generator
a	Matrix containing the A Specifications used in the program; Identical to the A input, except when using the Random data generator

Balloon_Trip calculates a trip from the Start location to the End location by splitting the trajectory into 5 sections: the 1st Elevation Change, the 1st Horizontal Movement, the 2nd Elevation Change, the 2nd Horizontal Movement, and the 3rd Elevation Change. However, the trip is calculated in a different order: the 1st Elevation Change, the 2nd Elevation Change, the 3rd Elevation Change, the 1st Horizontal Movement, and then the 2nd Horizontal Movement.

The reason that the sections are computed in this order is because of how *Balloon_Trip* ultimately decides what trajectory to take. The decision making hinges on the two horizontal sections travelled. For the 1st Horizontal section the program will choose a wind to follow based on how closely it will lead the balloon straight to the Ending location from the Starting location, and it will then choose the 2nd Horizontal section based on how far it will need to travel after the 2nd Elevation change, which is based on how far the balloon travels during the 1st Horizontal section. But it cannot choose this without first finding where the balloon would be located when it first starts the 1st Horizontal section and where it would end after the 2nd Horizontal section.

This means that every option for the 1st Elevation Change needs to be calculated before anything else. With this completed, the most direct route from the end of the 1st Elevation change to the End location can be found. This is the trajectory that will be traveled along for the 1st Horizontal Movement.

Now, possible options for the 2nd and 3rd Elevation Change must be calculated. This means that the way in which the wind blows the balloon needs to be found as it travels from the elevation chosen for the 1st Horizontal Movement to any other elevation and for how the balloon will be blown around during its decent from any possible

elevation. Then the elevation that the balloon will travel along during the 2nd Horizontal movement can be chosen, which also chooses which trajectory to follow for the 3rd Elevation change.

Now the actual trajectory for the balloon to travel along has been completed. At this point, there are only a few details left to complete: correctly recording the coordinates traveled to and outputting the data in tables and graphs. There are two tables: one for outputting the distance covered in flight and the other for displaying the six main coordinates from the flight trajectory. The six coordinates are shown in Table 3. There are a total of 5 possible graphs that can be outputted from the program. Details on these graphs can be seen in Table 4.

Table 3: The Six Coordinates that define the Flight Path.

Coordinate	Description
Coordinate 1	Start location
Coordinate 2	Location after 1st Elevation Change
Coordinate 3	Location after 1st Horizontal Movement
Coordinate 4	Location after 2nd Elevation Change
Coordinate 5	Location after 2nd Horizontal Movement
Coordinate 6	End location, location after 3rd Elevation Change

Table 4: Descriptions of each of the 5 different graphs generated by *Balloon_Trip*.

Graph	Description
Graph 1	3-D Graph; displaying the entre flight path, with a Wind representation in the center of the graph
Graph 2	2-D Graph; displays the flight path in the X-Y coordinate plane, with a Wind representation in the center of the graph
Graph 3	2-D Graph; displays the flight path in 3 subplots: X-Y, X-Z, and Y-Z coordinate planes, with the prominent view being the X-Y coordinate plane
Graph 4	2-D Graph; displays the flight path in 3 subplots: X-Y, X-Z, and Y-Z coordinate planes, with the prominent view being the X-Z coordinate plane
Graph 5	2-D Graph; displays the flight path in 3 subplots: X-Y, X-Z, and Y-Z coordinate planes, with the prominent view being the Y-Z coordinate plane

III. Program Overview

This section of the report will describe each of the programs inputs, outputs, tables, and graphs in detail. Each will be described in the order as they appear in the program.

A. Inputs

The ***Graph*** input is used to control whether or not any of the graphs are displayed at the end of the program. It uses a value of "1" for on, and a value of "0" for off.

Fig_Num is a matrix that contains all the figure numbers used when creating the 5 graphs. Each graph can be set to any figure number, but if the value of "0" is used for a particular graph, then that graph only will not be shown. As an example, [1 0 2 0 3] will display the 1st, 3rd, and 5th graphs with figure numbers of 1, 2, and 3 respectively; the 2nd and 4th graphs will not be displayed because values of "0" where used for their figure numbers.

The ***Display*** input is another control variable. It controls whether or not the tables or messages will display in MATLAB's Command Window. It uses a value of "1" for on, and a value of "0" for off.

The ***Random*** variable is used to control the built-in random data generator. Using the values in this matrix, Balloon_Trip will create values for the ***Wind***, ***Start***, ***End*** and ***a*** inputs. This input has three settings, depending on what values it is set to and if the remaining four inputs are defined or not. If it is set to a value of "0" with the next four inputs defined, then it is set to off and will not generate any data. If it is set to a value of "0" with the next four inputs not defined, then it will create random data based on default parameters built into the program. If it is defined as a full 4x3 matrix with the next four inputs not defined then it will generate random data based on the defined parameters. There are a total of 12 parameters used to define the ***Random*** matrix. Table 5 shows these parameters describing what index they hold and what they represent.

Table 5: Parameters used when generating random data, units are in feet unless otherwise stated.

Index	Description
Random(1,1)	The total number of winds that will be in the ***Wind*** matrix, no units
Random(1,2)	Minimum speed that can be used in the ***Wind*** matrix, in feet per minute
Random(1,3)	Maximum acceleration that can be used in the ***Wind*** matrix, in feet per minute
Random(2,1)	Maximum elevation that can be used the ***Wind*** matrix
Random(2,2)	Minimum acceleration that can be used in the ***a*** matrix, in feet per square second
Random(2,3)	Maximum acceleration that can be used in the ***a*** matrix, in feet per square second
Random(3,1)	Minimum value that the Start and End location can be located in the X-direction
Random(3,2)	Minimum value that the Start and End location can be located in the Y-direction
Random(3,3)	Minimum value that the Start and End location can be located in the Z-direction
Random(4,1)	Maximum value that the Start and End location can be located in the X-direction
Random(4,2)	Maximum value that the Start and End location can be located in the Y-direction
Random(4,3)	Maximum value that the Start and End location can be located in the Z-direction

The Wind matrix contains all the weather data used. It is an Nx3 sized matrix, where N represents the total number of different winds. The 1st column represents the speeds of the different winds in feet per minute, the 2nd column represents the direction that the wind is blowing in degrees, and the 3rd column represents the elevation that the winds are defined at in feet. Each row of this matrix represents one specific wind. The speeds defined must all be values above Zero. The directions that the winds blow are defined as an angle with 0 degrees representing east, 90 degrees representing north, and so on. While values of equal and above 360 and below 0 can used when defining the wind directions, the program will automatically change these numbers to be within the values of 0 and 360. The elevations must be defined in increasing order, from lowest elevation to highest, with no value being below Zero. Each wind is defined in the matrix as how the wind is behaving at and below the defining elevation. For example, [30 45 1000; 90 73 2000] defines two winds, one blowing at 30 feet per minute and another at 90 feet per minute. The first wind is defined as blowing between the elevation of 0 and 1000 feet, with the second wind blowing from above 1000 feet up to and including an elevation of 2000 feet.

The Start and End matrices define where the Starting and Ending locations are. They are defined in the Cartesian coordinate system so each input has three values, one for each axis of X, Y, and Z. The X-axis travels in the positive direction straight east and the Y-axis travels in the positive direction straight north, with the Z-axis being the Elevation axis.

The last input variable, a, is the acceleration matrix. It contains the value for the acceleration that the balloon travels at while rising or falling. Both values are inputted as positive values.

B. Outputs

The ***Total_Distance*** output is defined as the total distance that the hot air balloon would travel over the course of the entire flight. This is not a ground path value which would be just distance travelled in 2-dimensions, but instead the distance travelled in all 3-dimension while the balloon travels horizontally and vertically.

The ***Coordinates*** matrix defines every coordinate that the balloon travels to. A coordinate is defined as a point in space where the balloon changes direction because of the wind or where the balloon stops rising or falling in elevation. This output is enables the user to be able to re-create any graphs that they wish without having to re-run the program.

The next four outputs, ***Wind***, ***Start***, ***End*** and ***a***, are the input values when running the code. The reason they can be outputted as separate variables is because this is the only way that the user can gather any data created by the random data generator. That way, if there is an interesting set of data that the user wants to look at later on in the future, they can use these output variables to save this generated data.

C. Tables

Balloon Trip Outputs is the name of the 1st of two tables outputted by the program. It shows the output variable ***Total_Distance*** in the table. ***Coordinates*** is the name of the 2nd table, and it shows the 6 main defining coordinate points. These 6 points are define the boundaries of the 5 mission segments: 1st Elevation Change (Take-off), the 1st Horizontal Movement, the 2nd Elevation Change, the 2nd Horizontal Movement, and the 3rd Elevation Change (Landing).

Each table is outputted in MATLAB's Command Window. Figure 1 shows an example of the ***Balloon Trip Outputs*** table, and Figure 2 shows an example of the ***Coordinates*** table.

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Balloon Trip Outputs
..........Output...........|...Value.......|...Units..
-------------------------------------------------------
Total Distance Travelled...|...7942.2699...|...feet...
-------------------------------------------------------

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 1: Balloon Trip Output Table Example

```
+++++++++++++++++++++++++++++++++++++++++++

Coordinates
.......X.......|.......Y.......|....Z.....
-------------------------------------------
...0...........|...0...........|...0......
...57.5604.....|...1.4437......|...3000...
...1836.6262...|...2121.6517...|...3000...
...1865.0801...|...2134.9199...|...1000...
...1987.7526...|...2012.2474...|...1000...
...2000........|...2000........|...0......

+++++++++++++++++++++++++++++++++++++++++++
```

Figure 2: Coordinates Table Example

D. Graphs

There are a total of 5 possible graphs that be outputted from *Balloon_Trip*. The first two graphs are unique, while the last three graphs are just different versions of the same graph. There are also another 2 graphs that will only show in the event that program determines that there is no path available to travel from Start to End.

The first graph is a 3-Dimensional graph that shows the entire flight path taken. It also has an overlay representation of the winds placed in the center of the plot in the form of blue arrows. While the sizes of the arrows have no relationship to the size of the flight path, they are scaled according to each other so that the largest arrow is the fastest wind and the smallest arrow is the slowest wind. They are placed in the center of the figure, and are shown oriented according to the direction that that wind is blowing. The Start and End locations are also labeled, along with their coordinates. Each of the five flight path segments also have their respective travelled distance labeled on the graph next to the corresponding segment. Figure 3 is an example of what this first graph looks like.

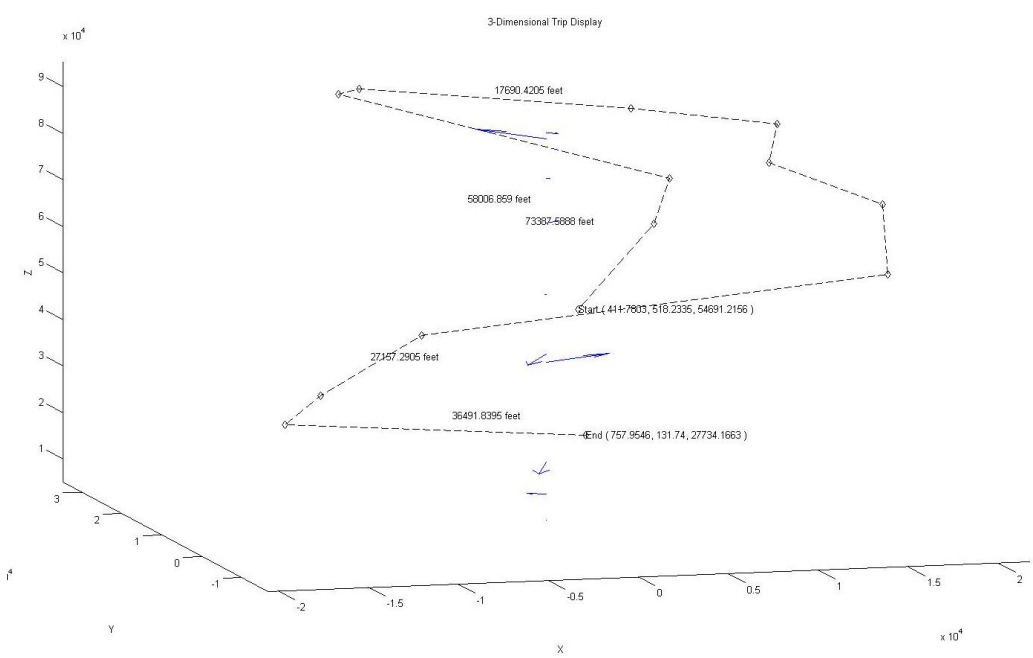

Figure 3: Graph 1 Example

The second graph is a 2-Dimensional view of the flight path, as seen from above and looking down the Z-axis (an X-Y view). This graph also has the wind representation overlay and labels for the Start and End locations. It does not, however, have the distance labels. Figure 4 shows an example of this graph.

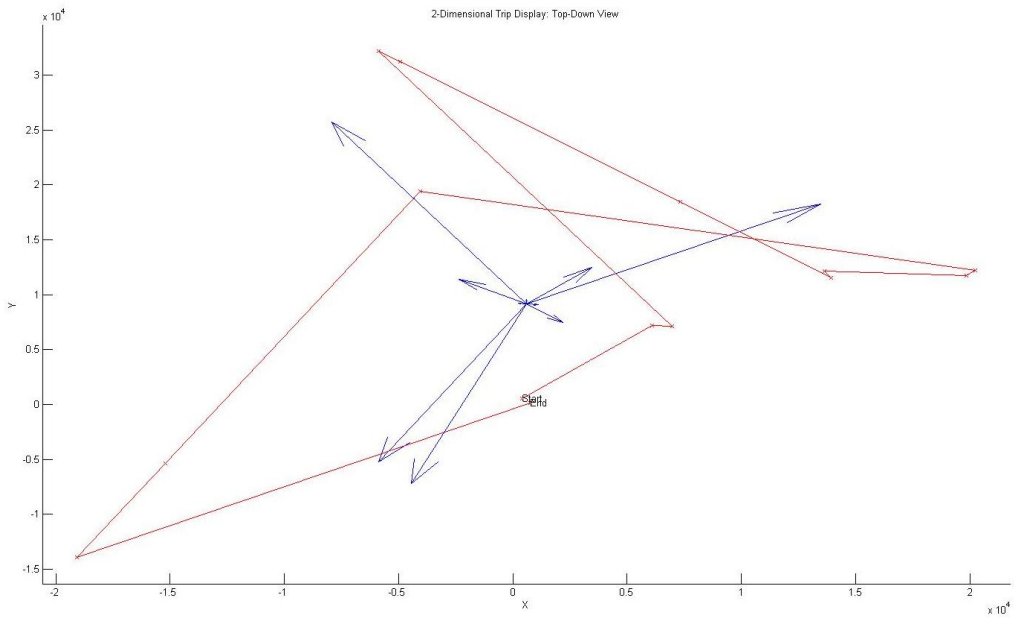

Figure 4: Graph 2 Example

The third, fourth, and fifth graphs all show 3 different 2-Dimensional views of the flight path, but each graph focuses on a different particular view: the third graph focuses on the top-down X-Y view, the fourth graph focuses on the side X-Z view, and the fifth focuses on the side Y-Z view. Each of these three graphs shows the other 2 views in half the space of the focus. Figures 5 – 7 show examples of these graphs.

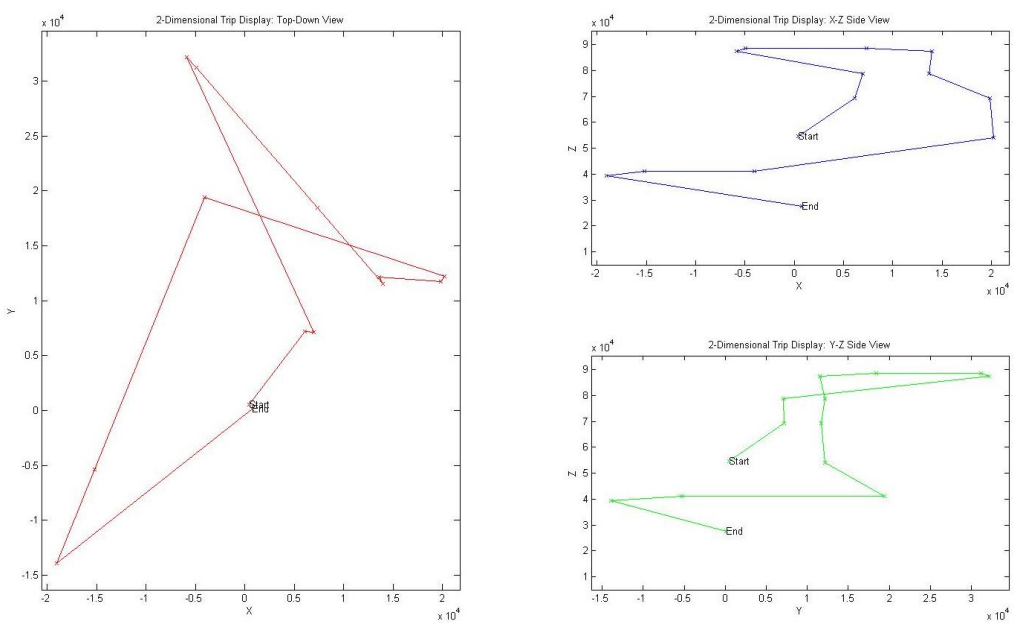

Figure 5: Graph 3 Example

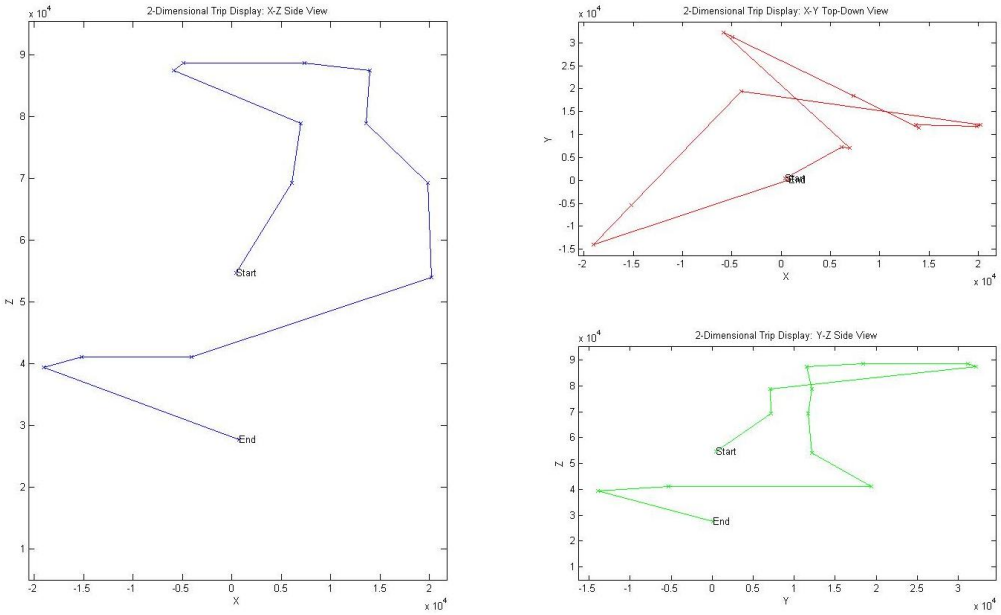

Figure 6: Graph 4 Example

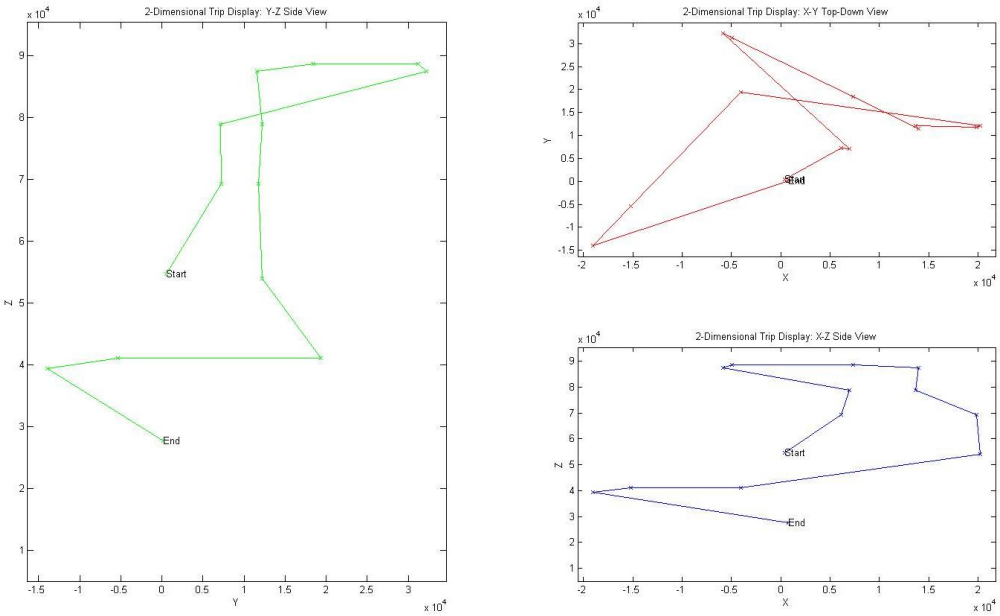

Figure 7: Graph 5 Example

The last two graphs are special graphs that will only show when there is no path to travel along from Start to End. They are essentially the first two normal graphs, the 3-D graph and the 2-D top-down view, except that they show no flight path as there is no path to travel along. Figures 8-9 show examples of these graphs.

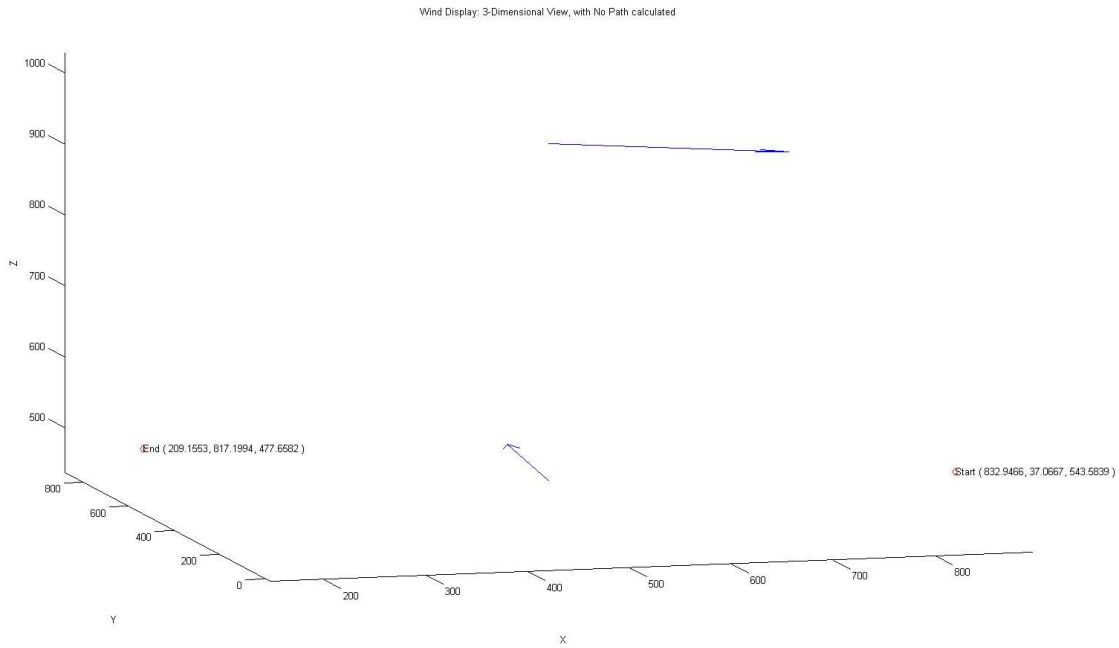

Figure 8: No Path Graph 1 Example

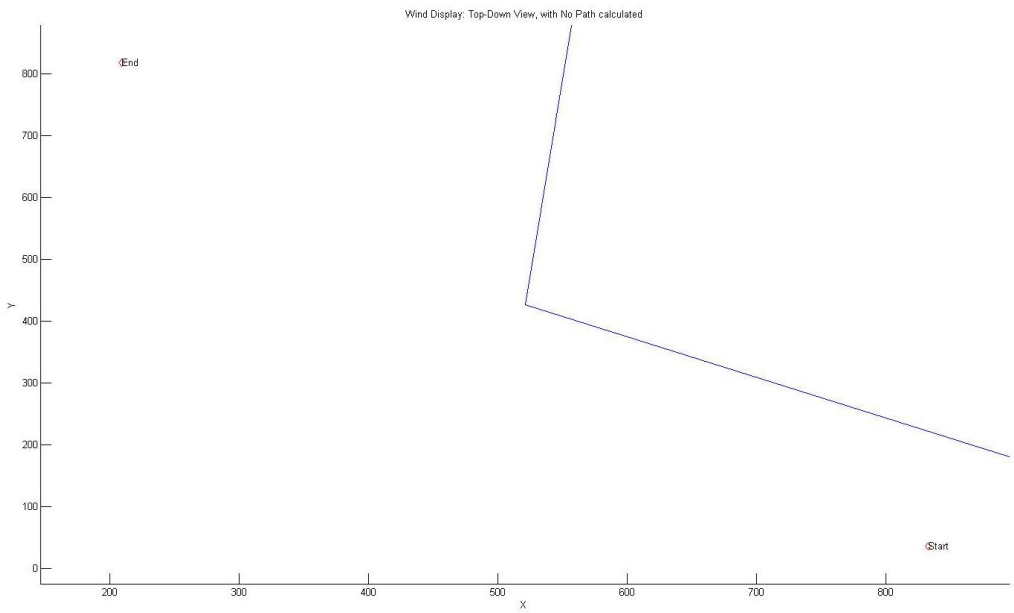

Figure 9: No Path Graph 2 Example

IV. Program Analysis

When first writing *Balloon_Trip*, I created a small set of data that could be easily followed by hand so as to test the logic of how the program chooses the final trajectory. By the time that I had completed the program, I could test it instead by continuously generating random data to input and then following the flight path to double check that the program was still working as intended.

Figure 10 shows the initial data being used and the resulting flight path chosen to follow. The data consisted of four very slow winds and simple Start and End locations. The slow winds are there so as to disturb the climb and descents in elevation as little as possible.

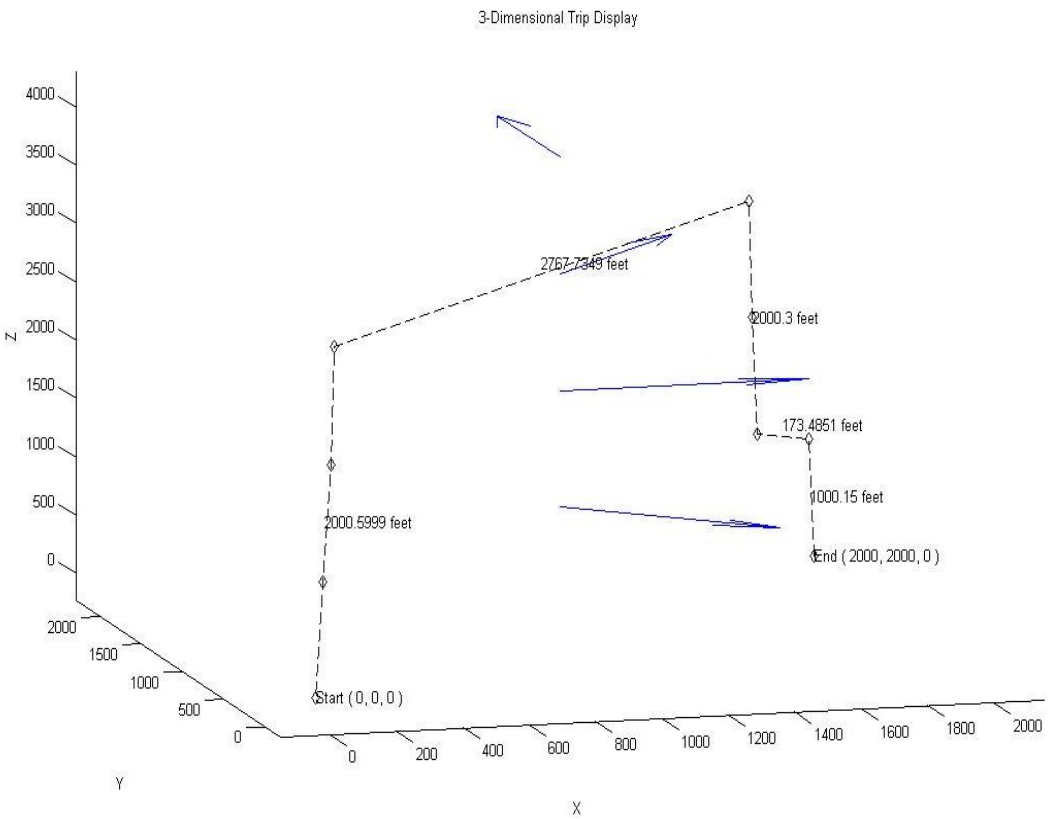

Figure 10: 3-D Graph showing data initially used to create the path finding code.

One of the more interesting sets of data that I came upon while testing was one where all the winds blew in nearly the same direction. But, since the start location was positioned upwind, there still existed a path to travel along to the end. Figure 11 shows the X-Y view of this path, so that the wind directions can be better seen.

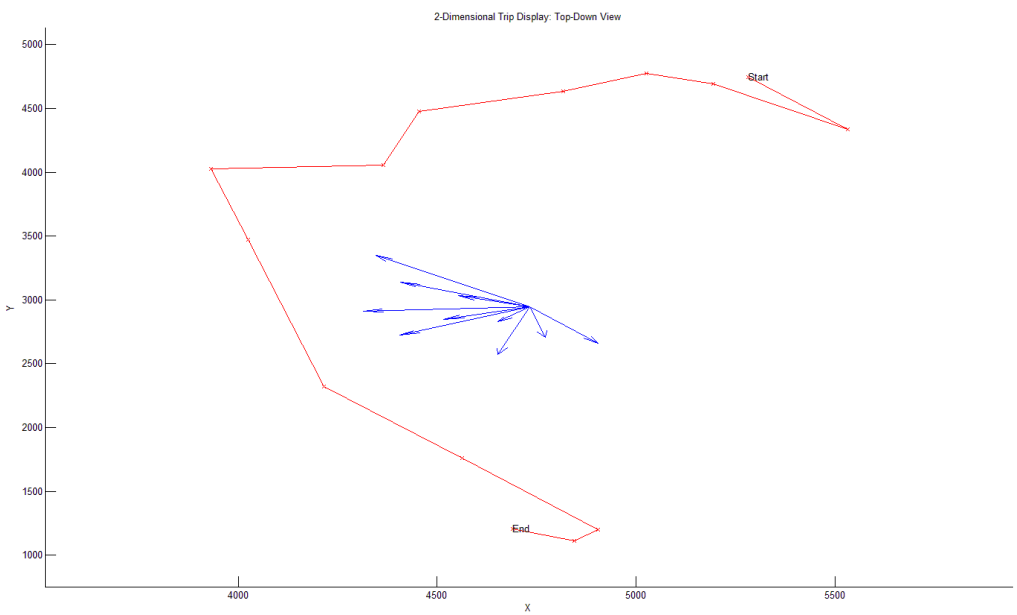

Figure 11: Data with no North-Eastern blowing wind.

This next set of data shows an interesting oddity: two very different rising and falling accelerations. In this case, the rising acceleration of the hot air balloon is very slow, just crawling upwards at a value of .0290 feet per square second. The falling acceleration is almost 142 times greater, at a value of 4.1174 feet per square second. This causes the wind to push the balloon around much more while moving upward, then it does when moving down. Figure 12 shows this in a 3-Dimensional graph of the flight path.

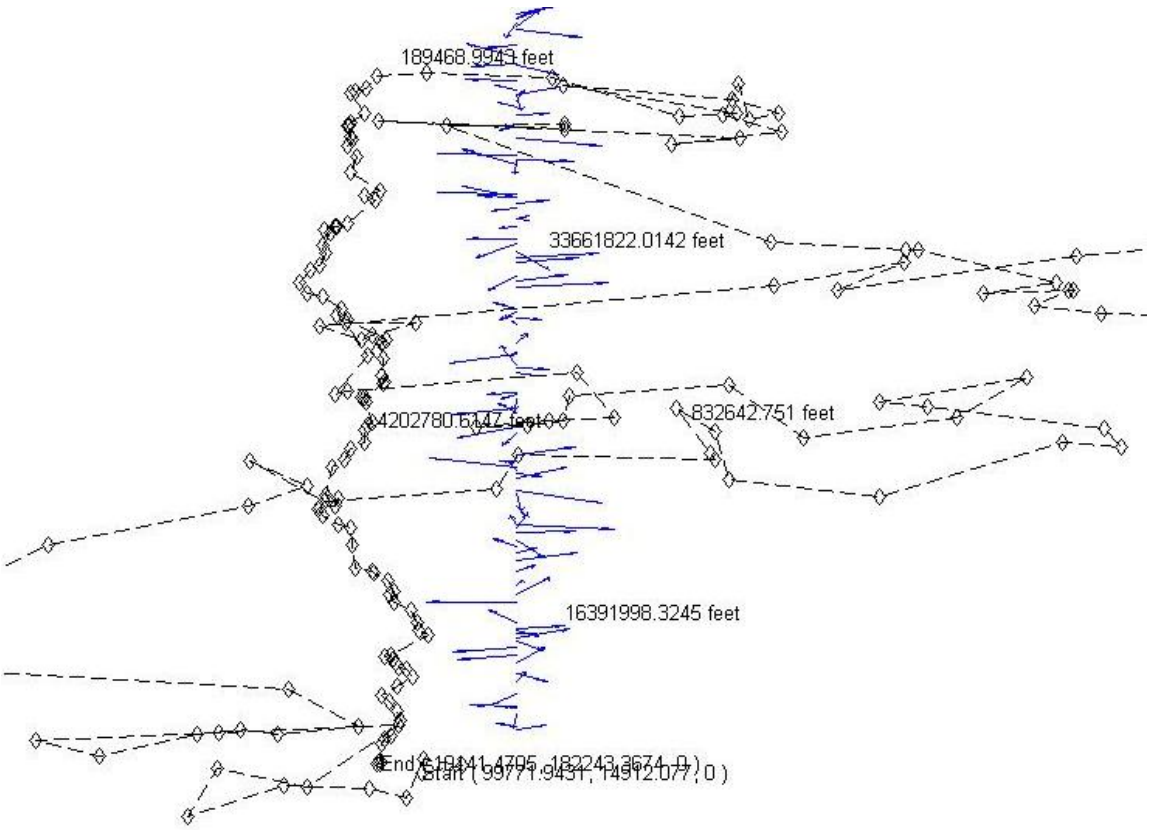

Figure 12: Flight Path of a slow rising and fast falling balloon.

Balloon_Trip can even choose a path to travel along regardless of the number of different winds. Figure 13 shows an example of this when there are 1000 winds to travel within. Perhaps unsurprising, this flight path is actually not otherwise interesting except for the large number of winds to travel through

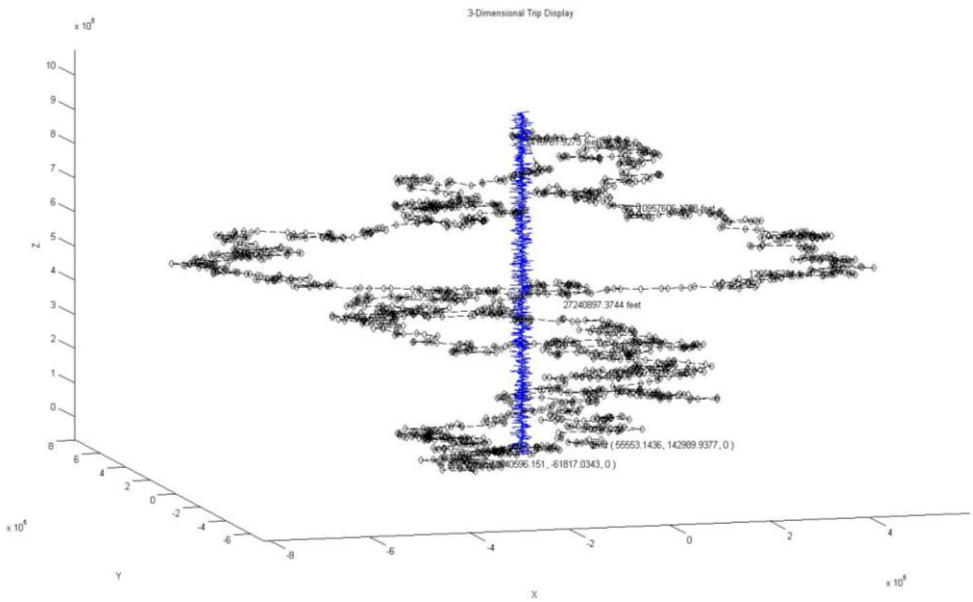

Figure 13: 1000 Winds with a flight path through almost all of them

My program *Balloon_Trip* is capable of handling most weather conditions to find a suitable flight path to travel along. It does, however, have some difficulty with some specific cases. While testing the program, I had come across one case where technically someone flying a hot air balloon could have found a path from the starting location to the end location, but the program didn't allow it to happen because of the manner in which the Horizontal Movement segments are calculated. They are always handled by having the balloon flying at the maximum height for that specific wind condition, even if the balloon was only going to descend at the end of it. Because of this, there may be cases where a balloon could be navigated to only barely within the wind it wants to travel along before then descending out of it and continuing along its path, but *Balloon_Trip* won't allow this because it wants to have the balloon ascend higher than is truly needed. Because of this, the balloon will be blown farther by the wind it is in, and by the time it reaches the maximum elevation for that wind it has been blown to far of course to be capable of finding a path to the end location. This was something that I noticed near to the end of writing the program. And while it could be "fixed" by decreasing the speeds of the winds or increasing the rising or falling acceleration which would then make the balloon be blown around less, it is not a true correction to the problem. The true correction would be to allow the program to choose if it needs to fly at the maximum or minimum elevation for a wind, not always at its maximum. But doing this would have required more major changes to the program than I deemed necessary, as I would have needed to make changes to core parts of the entirety of the program to change this one detail. So for now, *Balloon_Trip* will incorrectly state that there is no path when in some perfectly make cases, there actually is a path that can be followed. In almost every other set of weather conditions *Balloon_Trip* will work correctly. And while I have shown it can handle a very large set of weather conditions in most practical applications it would probably never need to use such a large number of data conditions.

Another drawback to my program is that it does require a rather detailed set of data, not only for the wind conditions, but also for how the balloon handles in rising and descending. The program always assumes that the balloon is either rising or falling at its maximum acceleration for that direction of travel. It does the same for the horizontal speed of the balloon, assuming that it is always travelling at the speed of the wind it is in, regardless of the nature of the wind that the balloon just exited.

V. Conclusion

In conclusion, my navigation program *Balloon_Trip* is not perfectly refined. It has a few built in assumptions that make it a rough navigation tool. But it could still easily be used as a guide for navigating a hot air balloon through the air from one location to another. By following its chosen flight path, you are guaranteed to come close to the correct landing site. How close would depend entirely on the specific conditions on hand. Very fast winds or a very large number of them would lead to being pushed more off course. But that means that the opposite is true as well: calm winds with a fewer number of them to navigate through would lessen any error to landing at the desired location.

References

[1]Briggs, C. S. (1986). *Ballooning.* Minneapolis, Minnesota: Lerner Publications Company.

[2]Owen, D. (1999). *Lighter Than Air.* Edison, New Jersey: Chartwell Books.

Appendix

Program MATLAB Code

```
function [Total_Distance, Coordinates, Wind, Start, End, a] =...
    Balloon_Trip(Graph, Fig_Num, Display, Random,...
                    Wind, Start, End, a)
%[Total_Distance, Coordinates, Wind, Start, End, a] =...
%                           Balloon_Trip(Graph, Fig_Num, Display, Random,...
%                                       Wind, Start, End, a)
%
%Balloon Trip Function Final Version
%Senior Project
%Navigation program used to Travel from some Start location to some End
%   location while in a Hot-Air Balloon
%By Dustin Blackwell
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%INPUTS
%Graph      =   on / off switch for graphing
%              = 1 for on
%              = 0 for off
%Fig_Num    =   Maxtrix of Figure Numbers for the 5 graphs
%              If one of the Values is set to "0" then that graph will not
%                 display, and will output a message saying so
%              EX: [5 9 0 14 99]
%Display    =   on / off switch for displaying outputs,tables,
%              and other various messages
%              = 1 for on
%              = 0 for off
%Random     =   Controls the built in Random Data Generator
%              can only be used if "Wind," "Start," "End," and "a"
%              variables are not defined
%              = 0 for off
%              = 0 for Default random paramaters with "Wind," "Start,"
%                "End," and "a" variables not defined
%              = [(Number of Elevations) (Min WindSpeed) (Max WindSpeed);
%                 (Max Elevation) (Min Accerleration) (Max Acceleration);
%                 (Min X) (Min Y) (Min Z); {for Start and End Locations}
%                 (Max X) (Max Y) (Max Z);] {for Start and End Locations}
%              with "Wind," "Start," "End," and "a" variables not defined
%              (Number of Elevations) must be geater than 0
%              (Min WindSpeed), (Min Z), (Min_a) can be defined as lower
%                 than Zero but will be reset as Zero within the program
%              Default: [50 0 30; 50000 0 10; 0 0 0; 7000 6000 9000;]
%Wind       =   Wind Matrix
%              Contains the neccessary Atmospheric Conditions
%              [Wind_Speed, Direction, Elevation]
%>Wind_Speed=   Matrix that contains all the didfferent Wind Speed Values
%                 in the atmosphere
%              All values should be positive
%              EX: [WS_1; WS_2; WS_3...etc]
%              Units: Feet per Minute
%>Direction =   Matrix that contains all the different Wind Direction
%                 Values in the atmosphere
%              EX: [WD_1; WD_2; WD_3...etc]
%              Units: Degrees, based off...
```

```
%                        East    = 0, 360 Degrees
%                        North   = 90 Degrees
%                        West    = 180 Degrees
%                        South   = 270 Degrees
%>Elevation =   Matrix that contains all the different Maximum Wind
%               Elevation Values in the atmosphere
%            All values should be positive
%            EX: [WE_1; WE_2; WE_3...etc]
%            Units: Feet
%Start      =   Start Position (on Ground)
%            Matrix that contains the Start Position for the Balloon
%            [X_S, Y_S, Z_S]
%            Units: Feet, from Origin (0,0,0)
%End        =   End Position (on Ground)
%            Matrix that contains the End Position for the Balloon
%            [X_E, Y_E, Z_E]
%            Units: Feet, from Origin (0,0,0)
%>>Orientation= In terms of the compass:
%               North = + Y Direction
%               South = - Y Direction
%               East  = + X Direction
%               West  = - X Direction
%a         =   Acceleration Matrix
%            Matrix that contains the Rising and Falling Acceleration
%               that the Balloon travels at
%            Both values should be inputted as positive numbers
%            [a_rise, a_fall]
%>a_rise    =   Acceleration for the Balloon when Rising in Elevation
%            Units: Feet per Second Squared
%>a_fall    =   Acceleration for the Balloon when Falling in Elevation
%            Units: Feet per Second Squared
%
%%%
```

```
%using default random parameters
if nargin < 5
    if Random == 0
        Random = [50 0 30;...
                  50000 0 10;...
                  0 0 0;...
                  7000 6000 9000;];
    end
    [R_Wind, R_Start, R_End, R_a] = SUB_Random_Balloon_Trip(Random);
    Wind = R_Wind;
    Start = R_Start;
    End = R_End;
    a = R_a;
end

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Errors
%Graph variable set to an incorrect value
if Graph ~= 1 && Graph ~= 0
    error('Choose either 1 or 0 for "Graph" Variable only');
end

%Fig_Num Matrix does not have exactly 5 Values
if length(Fig_Num) ~= 5
    error('"Fig_Num" Variable must ALWAYS contain exactly 4 Values');
end

%Display variable set to an incorrect value
if Display ~= 1 && Display ~= 0
    error('Choose either 1 or 0 for "Display" Variable only');
end

%Wind Matrix is not a N x 3 Matrix
if size(Wind) ~= 3
    E1 = '"Wind" Matrix must ALWAYS be a N x 3 Matrix';
    E2 = ', where N can be any non-negative integer';
    E = [E1 E2];
    error(E);
end

%Start variable does not have exactly 3 values
if length(Start) ~= 3
    error('"Start" Variable must ALWAYS contain exactly 3 Values');
end

%End variable does not have exactly 3 values
if length(End) ~= 3
    error('"End" Variable must ALWAYS contain exactly 3 Values');
end

%a Variable does not have 2 values
if length(a) ~= 2
    error('"a" Variable must ALWAYS contain exactly 2 Values');
end

%End of Section
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Inputs

Wind_Speed  =   Wind(:,1)*60/1;%Speed matrix for Wind, in seconds
Direction   =   Wind(:,2);%Directional Matrix for Wind
Elevation   =   Wind(:,3);%Elevation for Wind
[Wind_C]    =   size(Wind);%Number of Columns and Rows in Wind
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
X_S     =   Start(1,1);%Starting X Location
Y_S     =   Start(1,2);%Starting Y Location
Z_S     =   Start(1,3);%Starting Z Location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
X_E     =   End(1,1);%End X Location
Y_E     =   End(1,2);%End Y Location
Z_E     =   End(1,3);%End Z Location
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
a_rise   =   a(1,1);%Rising Acceleration
a_fall   =   a(1,2);%Falling Acceleration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Re-stating Direction so that each Direction value is between
% +360 and 0
while max(Direction) >= 360 || min(Direction) < 0
    for i = 1:Wind_C
        if Direction(i,1) >= 360
            Direction(i,1) = Direction(i,1) - 360;
        elseif Direction(i,1) < 0
            Direction(i,1) = Direction(i,1) + 360;
        end
    end
end
for i = 1:Wind_C
    if Direction(i,1) == 90 || Direction(i,1) == 270
        Direction(i,1) = Direction(i,1) + .0001;
    end
end

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Constants and Variables Directly based on Inputs
X = [X_S];%x component of the Location Variable (North / South)
X_1 = X(length(X));%X1
Y = [Y_S];%y component of the Location Variable (West / East)
Y_1 = Y(length(Y));%Y1
Z = [Z_S];%z component of the Location Variable (Elevation)
Z_1 = Z(length(Z));%Z1

X_bar = [X_E - X_S];%Distance between X Start and End locations
Y_bar = [Y_E - Y_S];%Distance between Y Start and End locations
```

```
Z_bar = [Z_E - Z_S];%Distance between Z Start and End locations
Q = atand(Y_bar / X_bar);%Angle to x/y End location from Start location

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Trip Calculator
%Step 1, Possible Horizontal movement during the 1st Elevation Change
%Step 2, Initial Direction Chooser
%Step 3, Calculate 2nd Elevation Change Movement
%Step 4, Possible Horizontal movement during the Final Elevation Change
%Step 5, Final Direction Chooser
%Step 6, Remaining Coordinate Variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Step 1

%Possible Horizontal movement during the 1st Elevation Change
for i = 1:Wind_C
    if i == 1
        if Elevation(i,1) < Z_S
            %set values to NaN if the elevation is lower that start
            %elevation
            X2(i,i) = NaN;
            Y2(i,i) = NaN;
            Z2(i,i) = NaN;
            Time2(i) = NaN;
            H2(i) = NaN;
            Q2(i) = NaN;
        else
            %Calculate values for i=1
            [D2, C2, S2] = SUB_Direction(Direction(i,1));
            a2 = a_rise;
            Time2(i) = ((Elevation(i,1) - Z_S) / a2)^.5;
            X2(i,i) = X_S + C2 * Wind_Speed(i,1) * Time2(i);
            Y2(i,i) = Y_S + S2 * Wind_Speed(i,1) * Time2(i);
            Z2(i,i) = Elevation(i,1);

            H2(i) = (( Wind_Speed(i,1) * Time2(i) )^2 +...
                     ( Elevation(i,1) - Z_S )^2)^.5;
            Q2(i) = [atand( ( abs(Y_E - Y2(i)) ) /...
                            ( abs(X_E - X2(i)) ) )];

            %make sure that Q2 is in the correct direction
            if X2(i,i) > X_E
                if Y2(i,i) > Y_E
                    Q2(i) = Q2(i) + 180;
                elseif Y2(i,i) <= Y_E
                    Q2(i) = 180 - Q2(i);
                end
            elseif X2(i,i) <= X_E
                if Y2(i,i) > Y_E
                    Q2(i) = 360 - Q2(i);
                elseif Y2(i,i) <= Y_E
                    Q2(i) = Q2(i);
                end
            end
```

```
    end%end i=1

    %set k for use in Elevation values above Z_E
    k = i;%[k=1]
elseif i ~=1
    if Elevation(i,1) < Z_S
        %set values to NaN if the elevation is lower that start
        %elevation
        X2(i,i) = NaN;
        Y2(i,i) = NaN;
        Z2(i,i) = NaN;
        Time2(i) = NaN;
        H2(i) = NaN;
        Q2(i) = NaN;

        %set k for use in Elevation values above Z_E
        k = i;
    else
        %Calculate values for i>1
        for j = k+1:i
            %calculate various values for each elevation change up to
            %to the "final elevation" of (i)

            if Elevation(j-1,1) < Z_S
                Elevationj(j-1) = Z_S;
            else
                Elevationj(j-1) = Elevation(j-1,1);
            end
            if j == k+1 && isnan(X2(1,1)) ~= 1 && isnan(Y2(1,1)) ~= 1
                Xj(j-1) = X2(1,1);
                Yj(j-1) = Y2(1,1);
            elseif j == k+1
                Xj(j-1) = X_S;
                Yj(j-1) = Y_S;
            else
                Xj(j-1) = Xj(j-1);
                Yj(j-1) = Yj(j-1);
            end
            [D2,C2,S2] = SUB_Direction(Direction(j,1));
            a2 = a_rise;
            Timej(j) = ((Elevation(j,1) - Elevationj(j-1)) / a2)^.5;
            Xj(j)= Xj(j-1) + C2 * Wind_Speed(j,1) * Timej(j);
            Yj(j)= Yj(j-1) + S2 * Wind_Speed(j,1) * Timej(j);

            Hj(j) = (( Wind_Speed(j,1) * Timej(j) )^2 +...
                    ( Elevation(j,1) - Elevationj(j-1) )^2)^.5;
            X2(i,j) = [Xj(j)];%final value in Xj
            Y2(i,j) = [Yj(j)];%final value in Yj
            Z2(i,j) = Elevation(j,1);
        end
        %end calculated values

        Time2(i) = [sum(Timej)];%sum of time it took to get from
                                %start to Elevation(i)
        H2(i) = [sum(Hj)];%sume of distance traveled

        Q2(i) = [atand( ( abs(Y_E - Y2(i,i)) ) /...
                    ( abs(X_E - X2(i,i)) ) )];
        %direction from (X2(i),Y2(i)) to (X_E,Y_E)

        %make sure that Q2 is in the correct direction
        if X2(i,i) > X_E
```

```
                if Y2(i,i) > Y_E
                    Q2(i) = Q2(i) + 180;
                elseif Y2(i,i) <= Y_E
                    Q2(i) = 180 - Q2(i);
                end
            elseif X2(i,i) <= X_E
                if Y2(i,i) > Y_E
                    Q2(i) = 360 - Q2(i);
                elseif Y2(i,i) <= Y_E
                    Q2(i) = Q2(i);
                end
            end

            %Clear "j" variables for use in next "i" Value
            clear Timej Xj Yj Hj
        end%end i~=1
    end%end if i
end%end for i

%Step 1 "Outputs"
X2;%Starting location after 1st Elevation change in X
Y2;%"New" Starting location after 1st Elevation change in Y
Z2;%"New" Starting location after 1st Elevation change in Z
H2;%Distances from (X_S,Y_S,Z_S) to (X2,Y2,Z2)
Q2;%Directions from (X2,Y2) to (X_E,Y_E)

clear D2;%clears D2 Variable for so that it can be used in the next step

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Step 2

%Initial Direction Chooser
for i = 1:Wind_C

    for j = 1:Wind_C

        D2(i) = Direction(i,1);

        %Find the difference between the Wind directions and the direction
            %between the start and end location
        ID(i,j) = abs( D2(i) - Q2(j) );

        %makes sure that the Initial_Elevation is above the start location
        if Elevation(i) < Z_S
            ID(i,j) = NaN;
        end

    end%end for j

end%end for i

%Find the angle direction closest to that of the angle
%between Start and End Locations
[ID_V1, ID_I1] = min(ID);
[ID_V2, ID_I2] = min(ID_V1);
ID_I = ID_I2;
ID_V = ID_V2;

Initial_Direction = Direction(ID_I1(ID_I2),1);
%Initial Horizontal Direction chosen to travel
Initial_Wind_Speed = Wind_Speed(ID_I1(ID_I2),1);
```

```
%Initial Horizontal Wind Speed
Initial_Elevation = Elevation(ID_I1(ID_I2),1);
%Initial Horizontal Wind Speed Elevation

if isnan(X2(1,1)) == 1
   I=0;
else
   if isnan(Y2(1,1)) == 1
      I=0;
   else
      I = 1;
      X(2) = X2(1,1);
      Y(2) = Y2(1,1);
      Z(2) = Z2(1,1);
   end
end
for i = k+1:ID_I1(ID_I2)
   if ID_I1(ID_I2) == 1
   else
     X(i+I-k+1) = [X2(ID_I1(ID_I2),i)];
     %Initial Horizontal Movement start in X
     Y(i+I-k+1) = [Y2(ID_I1(ID_I2),i)];
     %Initial Horizontal Movement start in Y
     Z(i+I-k+1) = [Z2(ID_I1(ID_I2),i)];
     %Initial Horizontal Movement start in Z
   end
end
%get rid of incorrectly added NaNs and Zeros
for i = 1:length(X)
   if X(i) == 0 && Y(i) == 0 && Z(i) == 0 && i ~= 1
      ii(i) = i;
   else
      ii(i) = 0;
   end
end
XYZ = 0;
for i = 1:length(X)
   if ii(i) == i
      XYZ = XYZ + 1;
      XX(i) = X(i+XYZ);
      YY(i) = Y(i+XYZ);
      ZZ(i) = Z(i+XYZ);
   elseif i+XYZ <= length(X)
      XX(i) = X(i+XYZ);
      YY(i) = Y(i+XYZ);
      ZZ(i) = Z(i+XYZ);
   end
end
clear X Y Z
X = XX;
Y = YY;
Z = ZZ;
clear XX YY ZZ i j

X_2 = X(length(X));%X2
Y_2 = Y(length(Y));%Y2
Z_2 = Z(length(Z));%Z2

Distance(1) = [H2(ID_I1(ID_I2))];
%Distance from Start to Initial Horizontal start
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Step 3

%Calculate 2nd Elevation Change Movement
for iii = 1:Wind_C
    if Initial_Elevation >= Elevation(iii,1)
        %higher elevation to lower
        a2P = a_fall;
        count = 0;
        for i = 1:Wind_C
            if Elevation(i,1) == Initial_Elevation
                j = i;
                jj = i;
            end
        end
        for i = 1:Wind_C
            if Elevation(i,1) < Elevation(iii,1)
                %do nothing
                count = count;
            elseif Elevation(i,1) > Initial_Elevation
                %do nothing
                count = count;
            else
                %Stuff happens here
                count = count +1;
                if count == 1
                    [D2P, C2P, S2P] = SUB_Direction(Direction(j,1));
                    Time2P(count,iii) = ((Elevation(j,1) - Z(length(Z)))...
                                         / a2P )^.5;
                    WS2P(count,iii) = Wind_Speed(j,1);
                    X2_PRIME(count,iii) = X(length(X)) +...
                                   C2P * WS2P(count,iii) *...
                                   Time2P(count,iii);
                    Y2_PRIME(count,iii) = Y(length(Y)) +...
                                   S2P * WS2P(count,iii) *...
                                   Time2P(count,iii);
                    Z2_PRIME(count,iii) = Z(length(Z));

                    DeltaX(count,iii) = X2_PRIME(count,iii) - X(length(X));
                    DeltaY(count,iii) = Y2_PRIME(count,iii) - Y(length(Y));
                else
                    j = j - 1;
                    [D2P, C2P, S2P] = SUB_Direction(Direction(j+1,1));
                    Time2P(count,iii) = ( (Elevation(j+1,1) -...
                                          Elevation(j,1))...
                                         / a2P )^.5;
                    WS2P(count,iii) = Wind_Speed(j,1);
                    X2_PRIME(count,iii) = X2_PRIME(count-1,iii) +...
                                   C2P * WS2P(count,iii) *...
                                   Time2P(count,iii);
                    Y2_PRIME(count,iii) = Y2_PRIME(count-1,iii) +...
                                   S2P * WS2P(count,iii) *...
                                   Time2P(count,iii);
                    Z2_PRIME(count,iii) = Elevation(j,1);

                    DeltaX(count,iii) = X2_PRIME(count,iii) - X(length(X));
                    DeltaY(count,iii) = Y2_PRIME(count,iii) - Y(length(Y));
                end%End if
            end%End if
        end%End for i
    elseif  Initial_Elevation < Elevation(iii,1)
```

```
        %lower elevation to hogher
        a2P = a_rise;
        count = 0;
        for i = 1:Wind_C
            if Elevation(i,1) < Initial_Elevation
                %do nothing
                count = count;
            elseif Elevation(i,1) > Elevation(iii,1)
                %do nothing
                count = count;
            elseif Elevation (i,1) >= Initial_Elevation
                %Stuff happens here
                count = count +1;
                if count == 1
                    [D2P, C2P, S2P] = SUB_Direction(Direction(i,1));
                    Time2P(count,iii) = ((Elevation(i,1) - Z(length(Z)))...
                                        / a2P)^.5;
                    WS2P(count,iii) = Wind_Speed(i,1);
                    X2_PRIME(count,iii) = X(length(X)) +...
                                      C2P * WS2P(count,iii) *...
                                      Time2P(count,iii);
                    Y2_PRIME(count,iii) = Y(length(Y)) +...
                                      S2P * WS2P(count,iii) *...
                                      Time2P(count,iii);
                    Z2_PRIME(count,iii) = Z(length(Z));

                    DeltaX(count,iii) = X2_PRIME(count,iii) - X(length(X));
                    DeltaY(count,iii) = Y2_PRIME(count,iii) - Y(length(Y));
                else
                    [D2P, C2P, S2P] = SUB_Direction(Direction(i,1));
                    Time2P(count,iii) = ( (Elevation(i,1) -...
                                          Elevation(i-1,1))...
                                        / a2P )^.5;
                    WS2P(count,iii) = Wind_Speed(i,1);
                    X2_PRIME(count,iii) = X2_PRIME(count-1,iii) +...
                                      C2P * WS2P(count,iii) *...
                                      Time2P(count,iii);
                    Y2_PRIME(count,iii) = Y2_PRIME(count-1,iii) +...
                                      S2P * WS2P(count,iii) *...
                                      Time2P(count,iii);
                    Z2_PRIME(count,iii) = Elevation(i,1);

                    DeltaX(count,iii) = X2_PRIME(count,iii) - X(length(X));
                    DeltaY(count,iii) = Y2_PRIME(count,iii) - Y(length(Y));
                end%End if
            end%End if
        end%End for i
    end%End if higher or lower
end%End for iii

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
%Step 4

%Possible Horizontal movement during the Final Elevation Change
for i = 1:Wind_C
    if i == 1
        if Elevation(i,1) < Z_E
            %set values to NaN if the elevation is lower that end
            %elevation
            X5(i,i) = NaN;
            Y5(i,i) = NaN;
```

```
        Z5(i,i) = NaN;
        Time5(i) = NaN;
        H5(i) = NaN;
    else
        %Calculate values for i=1
        [D3, C3, S3] = SUB_Direction(Direction(i,1)+180);

        a5 = a_fall;
        Time5(i) = ((Elevation(i,1) - Z_E) / a5)^.5;
        X5(i,i) = X_E + C3 * Wind_Speed(i,1) * Time5(i);
        Y5(i,i) = Y_E + S3 * Wind_Speed(i,1) * Time5(i);

        H5(i) = (( Wind_Speed(i,1) * Time5(i) )^2 +...
                ( Elevation(i,1) - Z_E )^2)^.5;
        Z5(i,i) = Elevation(i,1);
    end%end i=1

    %set k for use in Elevation values above Z_E
    k = i;%[k=1]
elseif i ~=1
    if Elevation(i,1) < Z_E
        %set values to NaN if the elevation is lower that end
        %elevation
        X5(i,i) = NaN;
        Y5(i,i) = NaN;
        Z5(i,i) = NaN;
        Time5(i) = NaN;
        H5(i) = NaN;

        %set k for use in Elevation values above Z_E
        k = i;
    else
        %Calculate values for i>1
        for f = k+1:i
            %calculate various values for each elevation change up to
            %to the "final elevation" of (i)
            if Elevation(f-1,1) < Z_E%if Elevation(k,1)<Z_E
                Elevationf(f-1) = Z_E;
            else
                Elevationf(f-1) = Elevation(f-1,1);
            end
            if f == k+1 && isnan(X5(1,1)) ~= 1 && isnan(Y5(1,1)) ~= 1
                Xf(f-1) = X5(1,1);
                Yf(f-1) = Y5(1,1);
            elseif f == k+1
                Xf(f-1) = X_E;
                Yf(f-1) = Y_E;
            else
                Xf(f-1) = Xf(f-1);
                Yf(f-1) = Yf(f-1);
            end
            [D3,C3,S3] = SUB_Direction(Direction(f,1)+180);
            a5 = a_fall;
            Timef(f) = ((Elevation(f,1) - Elevationf(f-1)) / a5)^.5;
            Xf(f)= Xf(f-1) + C3 * Wind_Speed(f,1) * Timef(f);
            Yf(f)= Yf(f-1) + S3 * Wind_Speed(f,1) * Timef(f);

            Hf(f) = (( Wind_Speed(f,1) * Timef(f) )^2 +...
                    ( Elevation(f,1) - Elevationf(f-1) )^2)^.5;
            X5(i,f) = [Xf(f)];%final value in Xf
            Y5(i,f) = [Yf(f)];%final value in Yf
            Z5(i,f) = Elevation(f,1);
```

```
            end
            %end calculated values
            Time5(i) = [sum(Timef)];%sum of time it took to get from
                                    %end to Elevation(i)
            H5(i) = [sum(Hf)];%sum of the distance traveled

            %Clear "f" variables for use in next "i" Value
            clear Timef Xf Yf Hf
        end%end i~=1
    end%end if i
end%end for i

%re-set Complex values to NaNs
for i = 1:length(X5)
    for j = 1:length(X5)
        REAL_X(i,j) = isreal(X5(i,j));
        REAL_Y(i,j) = isreal(Y5(i,j));
        REAL_Z(i,j) = isreal(Z5(i,j));
        if REAL_X(i,j) == 0
            X5(i,j) = NaN;
        end
        if REAL_Y(i,j) == 0
            Y5(i,j) = NaN;
        end
        if REAL_Z(i,j) == 0
            Z5(i,j) = NaN;
        end
    end
    REAL_T(i) = isreal(Time5(1,i));
    REAL_H(i) = isreal(H5(1,i));
    if REAL_T(i) == 0
        Time5(1,i) = NaN;
    end
    if REAL_H(i) == 0
        H5(1,i) = NaN;
    end
end

%Step 3 "Outputs"
X5;%Starting location before Final Elevation change in X
Y5;%Starting location before Final Elevation change in Y
Z5;%Starting location before Final Elevation change in Z
H5;%Distances from (X5,Y5,Z5) to (X_E,Y_E,Z_E)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
%Step 5

%Final Direction Chooser
for i = 1:Wind_C
    if i == ID_I1(ID_I2)
        %Set A_Bar so that the inital direction is not chossen again.
        LDX(i) = NaN;
        LDY(i) = NaN;
        X_2E(i) = NaN;
        Y_2E(i) = NaN;
        X_F(i) = NaN;
        Y_F(i) = NaN;
        A_Bar(i) = NaN;
        DF(i) = NaN;
        CF(i) = NaN;
        SF(i) = NaN;
```

```
        TF(i) = NaN;
        Q5(i) = NaN;
    else
        %Find the Distance from the Initial direction line and the Final
            %direction line
        LDX(i) = length(DeltaX(:,i));
        LDY(i) = length(DeltaY(:,i));
        XY0 = 0;
        j = 0;
        while XY0 == 0
            LDX0 = DeltaX(LDX(i)-j,i) - 0;
            LDY0 = DeltaY(LDY(i)-j,i) - 0;
            if LDX0 ~= 0 && LDY0 ~= 0
                XY0 = 1;
%           elseif j == LDX(i) && LDX0 ~= 0 && LDY0 ~= 0
%
            else
                j = j + 1;
            end
        end
        LDX(i) = LDX(i) - j;
        LDY(i) = LDY(i) - j;

        X_2E(i) = [X(length(X)) + DeltaX(LDX(i),i)];
        Y_2E(i) = [Y(length(Y)) + DeltaY(LDY(i),i)];

        [DF(i), CF(i), SF(i), TF(i)] = SUB_Direction(Direction(i,1) + 180);
        X_F(i) = ( ( Y5(i,i) - Y_2E(i) ) +...
                ( X_2E(i) * tand(Initial_Direction) ) -...
                ( X5(i,i) * TF(i) ) ) /...
                ( tand(Initial_Direction) - TF(i) );
        Y_F(i) = Y5(i,i) + TF(i) * (X_F(i) - X5(i,i));
        A_Bar(i) = ( ( X5(i,i) - X_F(i) )^2 + ( Y5(i,i) - Y_F(i) )^2 )^.5;

        %make sure that Q5 is the correct direction
        if X_F(i)>X5(i,i) && isnan(X_F(i))~=1 && isnan(X5(i,i))~=1....
                    && isinf(X_F(i))~=1 && isinf(X5(i,i))~=1
            if Y_F(i)>Y5(i,i) && isnan(Y_F(i))~=1 && isnan(Y5(i,i))~=1....
                        && isinf(Y_F(i))~=1 && isinf(Y5(i,i))~=1
                Q5(i) = atand( ( abs( Y_F(i)-Y5(i,i) ) ) /...
                            ( abs( X_F(i)-X5(i,i) ) ) ) + 180;
            elseif Y_F(i)<=Y5(i,i)&&isnan(Y_F(i))~=1&&isnan(Y5(i,i))~=1....
                            && isinf(Y_F(i))~=1 && isinf(Y5(i,i))~=1
                Q5(i) = 180 - atand( ( abs( Y_F(i)-Y5(i,i) ) ) /...
                                ( abs( X_F(i)-X5(i,i) ) ) );
            end
        elseif X_F(i)<=X5(i,i) && isnan(X_F(i))~=1 && isnan(X5(i,i))~=1....
                        && isinf(X_F(i))~=1 && isinf(X5(i,i))~=1
            if Y_F(i)>Y5(i,i) && isnan(Y_F(i))~=1 && isnan(Y5(i,i))~=1....
                        && isinf(Y_F(i))~=1 && isinf(Y5(i,i))~=1
                Q5(i) = 360 - atand( ( abs( Y_F(i)-Y5(i,i) ) ) /...
                                ( abs( X_F(i)-X5(i,i) ) ) );
            elseif Y_F(i)<=Y5(i,i)&&isnan(Y_F(i))~=1&&isnan(Y5(i,i))~=1....
                            && isinf(Y_F(i))~=1 && isinf(Y5(i,i))~=1
                Q5(i) = atand( ( abs( Y_F(i)-Y5(i,i) ) ) /...
                            ( abs( X_F(i)-X5(i,i) ) ) );
            end
        else
            Q5(i) = NaN;
        end

        %check if Q5(i) = Direction(i,1)
```

```
        if  abs(Q5(i) - Direction(i,1)) <= .000001
            A_Bar(i) = A_Bar(i);
        else
            A_Bar(i) = NaN;
        end

        %make sure that the direction between X_F etc and X_2 etc are
        %valid with the Wind Direction
        if X_F(i) > X_2E(i) && isnan(X_F(i)) ~= 1 && isinf(X_F(i)) ~= 1
            if Y_F(i) > Y_2E(i) && isnan(Y_F(i)) ~= 1 && isinf(Y_F(i)) ~= 1
                Q4(i) = atand( ( abs( Y_F(i)-Y_2E(i) ) ) /...
                            ( abs( X_F(i)-X_2E(i) ) ) );
            elseif Y_F(i)<=Y_2E(i) && isnan(Y_F(i))~=1 && isinf(Y_F(i))~=1
                Q4(i) = 360 - atand( ( abs( Y_F(i)-Y_2E(i) ) ) /...
                                ( abs( X_F(i)-X_2E(i) ) ) );
            end
        elseif X_F(i) <= X_2E(i) && isnan(X_F(i))~=1 && isinf(X_F(i))~=1
            if Y_F(i) > Y_2E(i) && isnan(Y_F(i)) ~= 1 && isinf(Y_F(i)) ~= 1
                Q4(i) = 180 - atand( ( abs( Y_F(i)-Y_2E(i) ) ) /...
                                ( abs( X_F(i)-X_2E(i) ) ) );
            elseif Y_F(i)<=Y_2E(i) && isnan(Y_F(i))~=1 && isinf(Y_F(i))~=1
                Q4(i) = atand( ( abs( Y_F(i)-Y_2E(i) ) ) /...
                            ( abs( X_F(i)-X_2E(i) ) ) ) + 180;
            end
        else
            Q4(i) = NaN;
        end

        %check if Q4(i) = Initial_Direction
        if  abs(Q4(i) - Initial_Direction) <= .000001
            A_Bar(i) = A_Bar(i);
        else
            A_Bar(i) = NaN;
        end

    end%end if
end%END for

[FD_V, FD_I] = min(A_Bar);%Find the minimum distance that can be travelled
%from the first wind direction to the end location by chosening the second
%direction

Distance(4) = [FD_V];
%Second Horizontal Location Change

%if there is no answer, set NO_ANSWER to 1 to act as a trigger later
%Value used when no answer is capable
NA = isnan(FD_V);
if NA == 1
    NO_ANSWER = 1;
elseif NA ~= 1
    NA = isinf(FD_V);
    if NA == 1
        NO_ANSWER = 1;
    elseif NA ~= 1
        NO_ANSWER = 0;
    end
end

Final_Direction = Direction(FD_I,1);%Final Direction chosen to travel
Final_Wind_Speed = Wind_Speed(FD_I,1);%Final Wind Speed
Final_Elevation = Elevation(FD_I,1);%Final Wind Speed Elevation
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Step 6

%Remaining Coordinate Variables
if NO_ANSWER == 0

    X( length(X) + (LDX(FD_I)) ) = [X_F(FD_I)];%X4
    Y( length(Y) + (LDX(FD_I)) ) = [Y_F(FD_I)];%Y4
    Z( length(Z) + (LDX(FD_I)) ) = [Final_Elevation];%Z4

    X_4 = X(length(X));%X4
    L4X = length(X);
    Y_4 = Y(length(Y));%Y4
    L4Y = length(Y);
    Z_4 = Z(length(Z));%Z4
    L4Z = length(Z);

    X( length(X) - (LDX(FD_I)) + 1 ) =...
                        [ X(length(X)) - DeltaX(LDX(FD_I),FD_I) ];
    Y( length(Y) - (LDX(FD_I)) + 1 ) =...
                        [ Y(length(Y)) - DeltaY(LDX(FD_I),FD_I) ];
    Z( length(Z) - (LDX(FD_I)) + 1 ) =...
                        [ Z2_PRIME(1,FD_I) ];
    XX = X( length(X) - (LDX(FD_I)) + 1 );
    YY = Y( length(Y) - (LDX(FD_I)) + 1 );
    j = 1;
    for i = 1:LDX(FD_I)-1
        X( length(X) - (LDX(FD_I)) + 1 + j ) = [XX + DeltaX(i+1,FD_I)];
        Y( length(Y) - (LDX(FD_I)) + 1 + j ) = [YY + DeltaY(i+1,FD_I)];
        Z( length(Z) - (LDX(FD_I)) + 1 + j ) = [Z2_PRIME(i+1,FD_I)];
        H3(i) = [ ( ( WS2P(i+1,FD_I) * Time2P(i+1,FD_I) )^2 +...
                  ( Z(length(Z) - (LDX(FD_I)) + 1 + j) -...
                    Z(length(Z) - (LDX(FD_I)) + 1 + j -1) )^2 )^.5 ];
        j = j + 1;
    end
    clear XX YY;

    X_3 = X( length(X) - (LDX(FD_I)) + 1 );%X3
    Y_3 = Y( length(Y) - (LDX(FD_I)) + 1 );%Y3
    Z_3 = Z( length(Z) - (LDX(FD_I)) + 1 );%Z3

    Distance(2) = [( ( X_3 - X_2 )^2 +...
                ( Y_3 - Y_2 )^2 )^.5];
    %First Horizontal Location Change

    Distance(3) = [sum(H3)];
    %Second Elevation Change

    j = FD_I;
    for i = length(X)+1:length(X)+FD_I
        if j == 1
            %Initial Horizontal Movement start in X
            X(i) = [X5(j,j)];
            %Initial Horizontal Movement start in Y
            Y(i) = [Y5(j,j)];
            %Initial Horizontal Movement start in Z
            Z(i) = [Z5(j,j)];
        else%if j ~=1
            %Initial Horizontal Movement start in X
            X(i) = [X5(FD_I,j)];
```

```
         %Initial Horizontal Movement start in Y
         Y(i) = [Y5(FD_I,j)];
         %Initial Horizontal Movement start in Z
         Z(i) = [Z5(FD_I,j)];

         j = j-1;
      end
   end

   %get rid of incorrectly added NaNs and Zeros
   while isnan(X(length(X))) == 1
      for i = 1:length(X)-1
        XX(i) = X(i);
        YY(i) = Y(i);
        ZZ(i) = Z(i);
      end
      clear X Y Z
      X = XX;
      Y = YY;
      Z = ZZ;
      clear XX YY ZZ
   end
   while X(length(X)) == 0 && Y(length(Y)) == 0 && Z(length(Z)) == 0
      for i = 1:length(X)-1
        XX(i) = X(i);
        YY(i) = Y(i);
        ZZ(i) = Z(i);
      end
      clear X Y Z
      X = XX;
      Y = YY;
      Z = ZZ;
      clear XX YY ZZ
   end

   X_5 = X(L4X+1);%X5
   Y_5 = Y(L4Y+1);%Y5
   Z_5 = Z(L4Z+1);%Z5

   Distance(5) = [H5(FD_I)];
   %Third Elevation Change

   X(length(X)+1) = [X_E];%X6
   Y(length(Y)+1) = [Y_E];%Y6
   Z(length(Z)+1) = [Z_E];%Z6
   X_6 = X(length(X));%X6
   Y_6 = Y(length(Y));%Y6
   Z_6 = Z(length(Z));%Z6
end

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%Outputs
%Commented out Values are computed earlier, but shown here for reference

%Distance Calculator
%Distance(1) = [H2(ID_I1(ID_I2))];
```

```
%First Elevation Change

%Distance(2) = [( ( X_3 - X_2 )^2 +...
%               ( Y_3 - Y_2 )^2 )^.5];
%First Horizontal Location Change

%Distance(3) = [sum(H3)];
%Second Elevation Change

%Distance(4) = [FD_V];
%Second Horizontal Location Change

%Distance(5) = [H5(FD_I)];
%Third Elevation Change

[Final_Distance] = sum(Distance);%Sum of the Distances Travelled

%if there is no answer, set the Final_Distance to NaN
if NO_ANSWER == 1
    Final_Distance = NaN;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Final Outputs

Total_Distance = [Final_Distance];
%Total Distance Travelled

%if there is no answer, set the Coordinates to NaN
if NO_ANSWER == 1
    X = [X_S X_E];
    Y = [Y_S Y_E];
    Z = [Z_S Z_E];
end

if nargin < 5
    Wind = R_Wind;
    Start = R_Start;
    End = R_End;
    a = R_a;
else
    Wind = Wind;
    Start = Start;
    End = End;
    a = a;
end

%Set of Coordinates for travel
Coordinates = [X; Y; Z]';

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Tables
if Display == 1%Displaying the Outputs in 2 Tables
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %creating set pieces for creating the displayed tables
    if NO_ANSWER == 0
        %create basic table pieces
        dot1 = '.';
        dot2 = '..';
        dot3 = '...';
        dot4 = '....';
        dot5 = '.....';
        dot6 = '......';
        space = ' ';
        dash = '-';
        plus = '+';
        line = '|';

        TIME_STEP = 'seconds';
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %create Table1: Distance
    if NO_ANSWER == 0

        %Set up the 2nd Column of Table1
        DOTS = length('Value');
        if length(num2str(Total_Distance)) > DOTS
            DOTS = length(num2str(Total_Distance));
        end

        if DOTS > length('Value')
            dot_Value = DOTS - length('Value');
        else
            dot_Value = 0;
        end
        if DOTS > length(num2str(Total_Distance))
            dot_TD = DOTS - length(num2str(Total_Distance));
        else
            dot_TD = 0;
        end

        Ldot_Value = [dot_Value] + 3;
        Ldot_TD = [dot_TD] + 3;
        clear DOTS dot_Value dot_TD dot_TTT

        for i = 1:Ldot_Value
            dot_Value(i) = dot1;
        end
        for i = 1:Ldot_TD
            dot_TD(i) = dot1;
        end
        clear Ldot_Value Ldot_TD Ldot_TTT


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
```

```
    %Set up the 3rd column of Table1
    DOTS = length('Units');
    if length('feet') > DOTS
      DOTS = length('feet');
    end
    if length(TIME_STEP) > DOTS
      DOTS = length(TIME_STEP);
    end

    if DOTS > length('Units')
      dot_Units = DOTS - length('Units');
    else
      dot_Units = 0;
    end
    if DOTS > length('feet')
      dot_feet = DOTS - length('feet');
    else
      dot_feet = 0;
    end
    if DOTS > length(TIME_STEP)
      dot_TS = DOTS - length(TIME_STEP);
    else
      dot_TS = 0;
    end

    Ldot_Units = [dot_Units] + 3;
    Ldot_feet = [dot_feet] + 3;
    Ldot_TS = [dot_TS] + 3;
    clear DOTS dot_Units dot_feet dot_TS

    for i = 1:Ldot_Units
      dot_Units(i) = dot1;
    end
    for i = 1:Ldot_feet
      dot_feet(i) = dot1;
    end
    for i = 1:Ldot_TS
      dot_TS(i) = dot1;
    end
    clear Ldot_Units Ldot_feet Ldot_TS


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
    %Creation of Table1
    TITLE1  ='Balloon Trip Outputs';

    Title1  = strcat('..........Output...........', [line dot3],...
         'Value', [dot_Value], [line dot3],...
         'Units', [dot_Units]);

    TD      = strcat('Total Distance Travelled', '...', [line dot3],...
         num2str(Total_Distance), [dot_TD], [line dot3],...
         'feet', [dot_feet]);

    ALMOST1 = char(TITLE1, Title1, TD);
    [trash S1] = size(ALMOST1);
    for i = 1:S1
      DASH1(i) = dash;
      PLUS1(i) = plus;
    end
```

```
        Table1 = char(space, PLUS1, space,...
                      TITLE1, Title1, DASH1,...
                      TD, DASH1,...
                      space, PLUS1, space);
        disp(Table1);

    elseif NO_ANSWER == 1
        Table1 = strcat(...
            'There are no Outputs since there is no Path calculated');
        Table1 = char(Table1);
        disp(Table1);
    end
    %End of Table1


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %create Table2: Coordinates
    if NO_ANSWER == 0

        %Set up of 1st Column
        DOTS = length('X');
        if length(num2str(X_1)) > DOTS
            DOTS = length(num2str(X_1));
        end
        if length(num2str(X_2)) > DOTS
            DOTS = length(num2str(X_2));
        end
        if length(num2str(X_3)) > DOTS
            DOTS = length(num2str(X_3));
        end
        if length(num2str(X_4)) > DOTS
            DOTS = length(num2str(X_4));
        end
        if length(num2str(X_5)) > DOTS
            DOTS = length(num2str(X_5));
        end
        if length(num2str(X_6)) > DOTS
            DOTS = length(num2str(X_6));
        end

        if DOTS > length('X')
            dot_X = DOTS - length('X');
        else
            dot_X = 0;
        end
        if DOTS > length(num2str(X_1))
            dot_X1 = DOTS - length(num2str(X_1));
        else
            dot_X1 = 0;
        end
        if DOTS > length(num2str(X_2))
            dot_X2 = DOTS - length(num2str(X_2));
        else
            dot_X2 = 0;
        end
        if DOTS > length(num2str(X_3))
            dot_X3 = DOTS - length(num2str(X_3));
        else
```

```
    dot_X3 = 0;
end
if DOTS > length(num2str(X_4))
    dot_X4 = DOTS - length(num2str(X_4));
else
    dot_X4 = 0;
end
if DOTS > length(num2str(X_5))
    dot_X5 = DOTS - length(num2str(X_5));
else
    dot_X5 = 0;
end
if DOTS > length(num2str(X_6))
    dot_X6 = DOTS - length(num2str(X_6));
else
    dot_X6 = 0;
end

if rem(dot_X,2) == 1
    %odd
    Ldot_bX = [(dot_X - 1) / 2];
    Ldot_aX = [(dot_X - 1) / 2 + 1]+3;
elseif rem(dot_X,2) == 0
    %even
    Ldot_bX = [dot_X / 2];
    Ldot_aX = [dot_X / 2] + 3;
end
Ldot_X1 = [dot_X1] + 3;
Ldot_X2 = [dot_X2] + 3;
Ldot_X3 = [dot_X3] + 3;
Ldot_X4 = [dot_X4] + 3;
Ldot_X5 = [dot_X5] + 3;
Ldot_X6 = [dot_X6] + 3;
clear DOTS dot_X dot_X1 dot_X2 dot_X3 dot_X4 dot_X5 dot_X6

for i = 1:Ldot_bX
    dot_bX(i) = dot1;
end
for i = 1:Ldot_aX
    dot_aX(i) = dot1;
end
for i = 1:Ldot_X1
    dot_X1(i) = dot1;
end
for i = 1:Ldot_X2
    dot_X2(i) = dot1;
end
for i = 1:Ldot_X3
    dot_X3(i) = dot1;
end
for i = 1:Ldot_X4
    dot_X4(i) = dot1;
end
for i = 1:Ldot_X5
    dot_X5(i) = dot1;
end
for i = 1:Ldot_X6
    dot_X6(i) = dot1;
end
clear Ldot_bX Ldot_aX Ldot_X1 Ldot_X2 Ldot_X3 Ldot_X4 Ldot_X5 Ldot_X6
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
    %Set up of 2nd Column

    DOTS = length('Y');
    if length(num2str(Y_1)) > DOTS
        DOTS = length(num2str(Y_1));
    end
    if length(num2str(Y_2)) > DOTS
        DOTS = length(num2str(Y_2));
    end
    if length(num2str(Y_3)) > DOTS
        DOTS = length(num2str(Y_3));
    end
    if length(num2str(Y_4)) > DOTS
        DOTS = length(num2str(Y_4));
    end
    if length(num2str(Y_5)) > DOTS
        DOTS = length(num2str(Y_5));
    end
    if length(num2str(Y_6)) > DOTS
        DOTS = length(num2str(Y_6));
    end

    if DOTS > length('Y')
        dot_Y = DOTS - length('Y');
    else
        dot_Y = 0;
    end
    if DOTS > length(num2str(Y_1))
        dot_Y1 = DOTS - length(num2str(Y_1));
    else
        dot_Y1 = 0;
    end
    if DOTS > length(num2str(Y_2))
        dot_Y2 = DOTS - length(num2str(Y_2));
    else
        dot_Y2 = 0;
    end
    if DOTS > length(num2str(Y_3))
        dot_Y3 = DOTS - length(num2str(Y_3));
    else
        dot_Y3 = 0;
    end
    if DOTS > length(num2str(Y_4))
        dot_Y4 = DOTS - length(num2str(Y_4));
    else
        dot_Y4 = 0;
    end
    if DOTS > length(num2str(Y_5))
        dot_Y5 = DOTS - length(num2str(Y_5));
    else
        dot_Y5 = 0;
    end
    if DOTS > length(num2str(Y_6))
        dot_Y6 = DOTS - length(num2str(Y_6));
    else
        dot_Y6 = 0;
    end

    if rem(dot_Y,2) == 1
```

```
        %odd
        Ldot_bY = [(dot_Y - 1) / 2];
        Ldot_aY = [(dot_Y - 1) / 2 + 1]+3;
    elseif rem(dot_Y,2) == 0
        %even
        Ldot_bY = [dot_Y / 2];
        Ldot_aY = [dot_Y / 2] + 3;
    end
    Ldot_Y1 = [dot_Y1] + 3;
    Ldot_Y2 = [dot_Y2] + 3;
    Ldot_Y3 = [dot_Y3] + 3;
    Ldot_Y4 = [dot_Y4] + 3;
    Ldot_Y5 = [dot_Y5] + 3;
    Ldot_Y6 = [dot_Y6] + 3;
    clear DOTS dot_Y dot_Y1 dot_Y2 dot_Y3 dot_Y4 dot_Y5 dot_Y6

    for i = 1:Ldot_bY
        dot_bY(i) = dot1;
    end
    for i = 1:Ldot_aY
        dot_aY(i) = dot1;
    end
    for i = 1:Ldot_Y1
        dot_Y1(i) = dot1;
    end
    for i = 1:Ldot_Y2
        dot_Y2(i) = dot1;
    end
    for i = 1:Ldot_Y3
        dot_Y3(i) = dot1;
    end
    for i = 1:Ldot_Y4
        dot_Y4(i) = dot1;
    end
    for i = 1:Ldot_Y5
        dot_Y5(i) = dot1;
    end
    for i = 1:Ldot_Y6
        dot_Y6(i) = dot1;
    end
   clear Ldot_bY Ldot_aY Ldot_Y1 Ldot_Y2 Ldot_Y3 Ldot_Y4 Ldot_Y5 Ldot_Y6


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
    %Set up of 3rd Column

    DOTS = length('Z');
    if length(num2str(Z_1)) > DOTS
        DOTS = length(num2str(Z_1));
    end
    if length(num2str(Z_2)) > DOTS
        DOTS = length(num2str(Z_2));
    end
    if length(num2str(Z_3)) > DOTS
        DOTS = length(num2str(Z_3));
    end
    if length(num2str(Z_4)) > DOTS
        DOTS = length(num2str(Z_4));
    end
    if length(num2str(Z_5)) > DOTS
        DOTS = length(num2str(Z_5));
```

```
end
if length(num2str(Z_6)) > DOTS
   DOTS = length(num2str(Z_6));
end

if DOTS > length('Z')
   dot_Z = DOTS - length('Z');
else
   dot_Z = 0;
end
if DOTS > length(num2str(Z_1))
   dot_Z1 = DOTS - length(num2str(Z_1));
else
   dot_Z1 = 0;
end
if DOTS > length(num2str(Z_2))
   dot_Z2 = DOTS - length(num2str(Z_2));
else
   dot_Z2 = 0;
end
if DOTS > length(num2str(Z_3))
   dot_Z3 = DOTS - length(num2str(Z_3));
else
   dot_Z3 = 0;
end
if DOTS > length(num2str(Z_4))
   dot_Z4 = DOTS - length(num2str(Z_4));
else
   dot_Z4 = 0;
end
if DOTS > length(num2str(Z_5))
   dot_Z5 = DOTS - length(num2str(Z_5));
else
   dot_Z5 = 0;
end
if DOTS > length(num2str(Z_6))
   dot_Z6 = DOTS - length(num2str(Z_6));
else
   dot_Z6 = 0;
end

if rem(dot_Z,2) == 1
   %odd
   Ldot_bZ = [(dot_Z - 1) / 2];
   Ldot_aZ = [(dot_Z - 1) / 2 + 1] + 3;
elseif rem(dot_Z,2) == 0
   %even
   Ldot_bZ = [dot_Z / 2];
   Ldot_aZ = [dot_Z / 2] + 3;
end
Ldot_Z1 = [dot_Z1] + 3;
Ldot_Z2 = [dot_Z2] + 3;
Ldot_Z3 = [dot_Z3] + 3;
Ldot_Z4 = [dot_Z4] + 3;
Ldot_Z5 = [dot_Z5] + 3;
Ldot_Z6 = [dot_Z6] + 3;
clear DOTS dot_Z dot_Z1 dot_Z2 dot_Z3 dot_Z4 dot_Z5 dot_Z6

for i = 1:Ldot_bZ
   dot_bZ(i) = dot1;
end
for i = 1:Ldot_aZ
```

```
      dot_aZ(i) = dot1;
    end
    for i = 1:Ldot_Z1
      dot_Z1(i) = dot1;
    end
    for i = 1:Ldot_Z2
      dot_Z2(i) = dot1;
    end
    for i = 1:Ldot_Z3
      dot_Z3(i) = dot1;
    end
    for i = 1:Ldot_Z4
      dot_Z4(i) = dot1;
    end
    for i = 1:Ldot_Z5
      dot_Z5(i) = dot1;
    end
    for i = 1:Ldot_Z6
      dot_Z6(i) = dot1;
    end
   clear Ldot_bZ Ldot_aZ Ldot_Z1 Ldot_Z2 Ldot_Z3 Ldot_Z4 Ldot_Z5 Ldot_Z6


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
    %creation of Table2
    TITLE2  = 'Coordinates';

    Title2  = strcat(dot3, dot_bX, 'X', dot_aX, [line dot3],...
             dot_bY, 'Y', dot_aY, [line dot3],...
             dot_bZ, 'Z', dot_aZ);

    C1     = strcat(dot3, num2str(X_1), dot_X1, [line dot3],...
             num2str(Y_1), dot_Y1,[line dot3],...
             num2str(Z_1), dot_Z1);

    C2     = strcat(dot3, num2str(X_2), dot_X2, [line dot3],...
             num2str(Y_2), dot_Y2, [line dot3],...
             num2str(Z_2), dot_Z2);

    C3     = strcat(dot3, num2str(X_3), dot_X3, [line dot3],...
             num2str(Y_3), dot_Y3, [line dot3],...
             num2str(Z_3), dot_Z3);

    C4     = strcat(dot3, num2str(X_4), dot_X4, [line dot3],...
             num2str(Y_4), dot_Y4, [line dot3],...
             num2str(Z_4), dot_Z4);

    C5     = strcat(dot3, num2str(X_5), dot_X5, [line dot3],...
             num2str(Y_5), dot_Y5, [line dot3],...
             num2str(Z_5), dot_Z5);

    C6     = strcat(dot3, num2str(X_6), dot_X6, [line dot3],...
             num2str(Y_6), dot_Y6, [line dot3],...
             num2str(Z_6), dot_Z6);

    ALMOST2 = char(TITLE2, Title2, C1, C2, C3 ,C4, C5, C6);
    [trash S2] = size(ALMOST2);
    clear trash
    for i = 1:S2
      DASH2(i) = dash;
      PLUS2(i) = plus;
```

```
        end

        Table2=char(space, PLUS2, space,...
                TITLE2, Title2, DASH2,...
                C1, C2, C3, C4, C5, C6,...
                space, PLUS2, space);
        disp(Table2);

    elseif NO_ANSWER == 1
        Table2 = strcat('There is no Path calculated to travel along');
        Table2 = char(Table2);
        disp(Table2);
    end%End of Table2


%%%
```

```
        Uq3(i) = Wind_Speed(i,1) * Cq;
        Vq3(i) = Wind_Speed(i,1) * Sq;
        Wq3(i) = 0;
        Xq3(i) = (X_MAX + X_MIN) / 2;
        Yq3(i) = (Y_MAX + Y_MIN) / 2;
        Zq3(i) = Elevation(i);
    end

    if abs(X_MAX) >= abs(X_MIN)
        SCALE = abs(X_MAX) / Z_MAX;
    elseif abs(X_MAX) < abs(X_MIN)
        SCALE = abs(X_MIN) / Z_MAX;
    end

    if abs(X_MAX) >= abs(X_MIN)
        SCALE2(1) = abs(X_MAX);
    elseif abs(X_MAX) < abs(X_MIN)
        SCALE2(1) = abs(X_MIN);
    end
    if abs(Y_MAX) >= abs(Y_MIN)
        SCALE2(2) = abs(Y_MAX);
    elseif abs(Y_MAX) < abs(Y_MIN)
        SCALE2(2) = abs(Y_MIN);
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Graph 1
    %3-D Graph of the entire trip
    if Fig_Num(1) == 0 && Display == 1
        %Graph 1 set to not display
        G_REPORT = strcat('Graph 1 set to not Display');
        G_REPORT = char(G_REPORT);
        disp(G_REPORT);
    else
        %Plot Graph 1
        figure(Fig_Num(1));
        clf(Fig_Num(1));
        figure(Fig_Num(1));

        hold on;

        quiver3(Xq3, Yq3, Zq3, Uq3, Vq3, Wq3, SCALE);

        plot3(X, Y, Z, 'kd--');

        view(-15, 15);
        title('3-Dimensional Trip Display');
        xlabel('X');
        ylabel('Y');
        zlabel('Z');
        axis( AXIS3 );

        %Start Location Label
        START=strcat('Start (',[' ' num2str(X_S)],',',...
                     [' ' num2str(Y_S)],',',...
                     [' ' num2str(Z_S)],' )');
        text(X_S,Y_S,Z_S,START);
        %End Location Label
        END=strcat('End (',[' ' num2str(X_E)],',',...
                   [' ' num2str(Y_E)],',',...
```

```
                    [' ' num2str(Z_E)],' )');
        text(X_E, Y_E, Z_E, END);

        %1st Distance
        text([(X_2 + X_1) / 2],...
            [(Y_2 + Y_1) / 2],...
            [(Z_2 + Z_1) / 2],...
            strcat([num2str(Distance(1))],...
            ' feet'));
        %2nd Distance
        text([(X_3 + X_2) / 2],...
            [(Y_3 + Y_2) / 2],...
            [Z_3 + .02 * (Z_MAX - Z_MIN)],...
            strcat([num2str(Distance(2))],...
            ' feet'));
        %3rd Distance
        text([(X_4 + X_3) / 2],...
            [(Y_4 + Y_3) / 2],...
            [(Z_4 + Z_3) / 2],...
            strcat([num2str(Distance(3))],...
            ' feet'));
        %4th Distance
        text([(X_5 + X_4) / 2],...
            [(Y_5 + Y_4) / 2],...
            [Z_5 + .02 * (Z_MAX - Z_MIN)],...
            strcat([num2str(Distance(4))],...
            ' feet'));
        %5th Distance
        text([(X_6 + X_5) / 2],...
            [(Y_6 + Y_5) / 2],...
            [(Z_6 + Z_5) / 2],...
            strcat([num2str(Distance(5))],...
            ' feet'));

        hold off;
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Graph 2
    %Top-Down view of the Trip
    if Fig_Num(2) == 0 && Display == 1
        %Graph 2 set to not display
        G_REPORT = strcat('Graph 2 set to not Display');
        G_REPORT = char(G_REPORT);
        disp(G_REPORT);
    elseif Fig_Num(2) ~= 0
        %Plot Graph 2
        figure(Fig_Num(2));
        clf(Fig_Num(2));
        figure(Fig_Num(2));

        hold on;

        quiver(Xq3, Yq3, Uq3, Vq3, SCALE2(1)/SCALE2(2) *40);

        plot(X, Y, 'rx-');

        title('2-Dimensional Trip Display: Top-Down View');
        xlabel('X');
        ylabel('Y');
```

```
        axis( AXIS2_XY );

        hold off;
        text(X_S, Y_S, 'Start');
        text(X_E, Y_E, 'End');
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Graph 3
    %Collection of 3-graphs from all useful view points
    if Fig_Num(3) == 0 && Display == 1
        %Graph 3 set to not display
        G_REPORT = strcat('Graph 3 set to not Display');
        G_REPORT = char(G_REPORT);
        disp(G_REPORT);
    elseif Fig_Num(3) ~= 0
        %Plot Graph 3
        figure(Fig_Num(3));
        clf(Fig_Num(3));
        figure(Fig_Num(3));

        %Subplot 1 :X-Y
        subplot(2, 2, [1 3]), plot(X, Y, 'rx-');
        title('2-Dimensional Trip Display: Top-Down View');
        xlabel('X');
        ylabel('Y');
        axis( AXIS2_XY );
        text(X_S, Y_S, 'Start');
        text(X_E, Y_E, 'End');

        %Subplot 2: X-Z
        subplot(2, 2, 2), plot(X, Z, 'bx-');
        title('2-Dimensional Trip Display: X-Z Side View');
        xlabel('X');
        ylabel('Z');
        axis( AXIS2_XZ );
        text(X_S, Z_S, 'Start');
        text(X_E, Z_E, 'End');

        %Subplot 3: Y-Z
        subplot(2, 2, 4), plot(Y, Z, 'gx-');
        title('2-Dimensional Trip Display: Y-Z Side View');
        xlabel('Y');
        ylabel('Z');
        axis( AXIS2_YZ );
        text(Y_S, Z_S, 'Start');
        text(Y_E, Z_E, 'End');
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Graph 4
    %Collection of 3-graphs from all useful view points
    %Arranged differently than Graph 3
    if Fig_Num(4) == 0 && Display == 1
        %Graph 4 set to not display
        G_REPORT = strcat('Graph 4 set to not Display');
        G_REPORT = char(G_REPORT);
        disp(G_REPORT);
```

```
    elseif Fig_Num(4) ~= 0
        %Plot Graph 4
        figure(Fig_Num(4));
        clf(Fig_Num(4));
        figure(Fig_Num(4));

        %Subplot 1: X-Y
        subplot(2, 2, 2), plot(X, Y, 'rx-');
        title('2-Dimensional Trip Display: X-Y Top-Down View');
        xlabel('X');
        ylabel('Y');
        axis( AXIS2_XY );
        text(X_S, Y_S, 'Start');
        text(X_E, Y_E, 'End');

        %Subplot 2: X-Z
        subplot(2, 2, [1 3]), plot(X, Z, 'bx-');
        title('2-Dimensional Trip Display: X-Z Side View');
        xlabel('X');
        ylabel('Z');
        axis( AXIS2_XZ );
        text(X_S, Z_S, 'Start');
        text(X_E, Z_E, 'End');

        %Subplot 3: Y-Z
        subplot(2, 2, 4), plot(Y, Z, 'gx-');
        title('2-Dimensional Trip Display: Y-Z Side View');
        xlabel('Y');
        ylabel('Z');
        axis( AXIS2_YZ );
        text(Y_S, Z_S, 'Start');
        text(Y_E, Z_E, 'End');
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
    %Graph 5
    %Collection of 3-graphs from all useful view points
    %Arranged differently than Graph 3 & 4
    if Fig_Num(5) == 0 && Display == 1
        %Graph 5 set to not display
        G_REPORT = strcat('Graph 5 set to not Display');
        G_REPORT = char(G_REPORT);
        disp(G_REPORT);
    elseif Fig_Num(5) ~= 0
        %Plot Graph 5
        figure(Fig_Num(5));
        clf(Fig_Num(5));
        figure(Fig_Num(5));

        %Subplot 1: X-Y
        subplot(2, 2, 2), plot(X, Y, 'rx-');
        title('2-Dimensional Trip Display: X-Y Top-Down View');
        xlabel('X');
        ylabel('Y');
        axis( AXIS2_XY );
        text(X_S, Y_S, 'Start');
        text(X_E, Y_E, 'End');

        %Subplot 2: X-Z
        subplot(2, 2, 4), plot(X, Z, 'bx-');
```

```
        title('2-Dimensional Trip Display: X-Z Side View');
        xlabel('X');
        ylabel('Z');
        axis( AXIS2_XZ );
        text(X_S, Z_S, 'Start');
        text(X_E, Z_E, 'End');

        %Subplot 3: Y-Z
        subplot(2, 2, [1 3]), plot(Y, Z, 'gx-');
        title('2-Dimensional Trip Display: Y-Z Side View');
        xlabel('Y');
        ylabel('Z');
        axis( AXIS2_YZ );
        text(Y_S, Z_S, 'Start');
        text(Y_E, Z_E, 'End');
    end

elseif Graph == 0 && Display == 1 && NO_ANSWER == 0%Graph switch off

    G_REPORT = strcat('Graph set to not Display');
    G_REPORT = char(G_REPORT);
    disp(G_REPORT);

elseif Graph == 1 && NO_ANSWER == 1%No Answer Graphs

    %creating variables to set the axis limits in the following graphs
    Percent=.075;
    X_MIN = min(X) - Percent * max(abs(X));
    X_MAX = max(X) + Percent * max(abs(X));
    Y_MIN = min(Y) - Percent * max(abs(Y));
    Y_MAX = max(Y) + Percent * max(abs(Y));
    Z_MIN = min(Z) - Percent * max(Z);
    if Z_MIN > Elevation(1,1)
      Z_MIN = Elevation(1,1) - Percent * Elevation(1,1);
    end
    Z_MAX = max(Z) + Percent * max(Z);
    if Z_MAX < Elevation(length(Elevation),1)
      Z_MAX = Elevation(length(Elevation),1) +...
            Percent * Elevation(length(Elevation),1);
    end
    AXIS2_XY = [X_MIN X_MAX Y_MIN Y_MAX];%axis for 2-D graph of X-Y
    AXIS3 = [X_MIN X_MAX Y_MIN Y_MAX Z_MIN Z_MAX];%axis for 3-D graph


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Wind Plot Variables
    for i = 1:Wind_C
        [Dq,Cq,Sq,Tq] = SUB_Direction(Direction(i,1));
        Uq3(i) = Wind_Speed(i,1) * Cq;
        Vq3(i) = Wind_Speed(i,1) * Sq;
        Wq3(i) = 0;
        Xq3(i) = (X_MAX + X_MIN) / 2;
        Yq3(i) = (Y_MAX + Y_MIN) / 2;
        Zq3(i) = Elevation(i);
    end

    if abs(X_MAX) >= abs(X_MIN)
        SCALE = abs(X_MAX) / Z_MAX;
    elseif abs(X_MAX) < abs(X_MIN)
        SCALE = abs(X_MIN) / Z_MAX;
    end
```

```
    if abs(X_MAX) >= abs(X_MIN)
        SCALE2(1) = abs(X_MAX);
    elseif abs(X_MAX) < abs(X_MIN)
        SCALE2(1) = abs(X_MIN);
    end
    if abs(Y_MAX) >= abs(Y_MIN)
        SCALE2(2) = abs(Y_MAX);
    elseif abs(Y_MAX) < abs(Y_MIN)
        SCALE2(2) = abs(Y_MIN);
    end

    %finds current figure handle
    FIG = gcf;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Plot Graph 1
    figure(FIG + 1);

    hold on;

    quiver3(Xq3, Yq3, Zq3, Uq3, Vq3, Wq3, SCALE);

    plot3(X, Y, Z, 'rd');

    view(-15, 15);
    title('Wind Display: 3-Dimensional View, with No Path calculated');
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    axis( AXIS3 );

    %Start Location Label
    START=strcat('Start (',[' ' num2str(X_S)],',',...
                 [' ' num2str(Y_S)],',',...
                 [' ' num2str(Z_S)],' )');
    text(X_S,Y_S,Z_S,START);
    %End Location Label
    END=strcat('End (',[' ' num2str(X_E)],',',...
               [' ' num2str(Y_E)],',',...
               [' ' num2str(Z_E)],' )');
    text(X_E, Y_E, Z_E, END);

    hold off;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
    %Plot Graph 2
    figure(FIG + 2);

    hold on;

    quiver(Xq3, Yq3, Uq3, Vq3, SCALE2(1)/SCALE2(2) *30);

    plot(X,Y,'rd');

    title('Wind Display: Top-Down View, with No Path calculated');
    xlabel('X');
    ylabel('Y');
```

```
    axis( AXIS2_XY );

    %Start Location Label
    text(X_S, Y_S, 'Start');
    %End Location Label
    text(X_E, Y_E, 'End');

    hold off;

end%End Graph switch on

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
warning('on', 'MATLAB:divideByZero');%Turns ON divide by Zero Warnings
end%End of Function


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Sub Functions

function [D, C, S, T] = SUB_Direction(Direction)
%Sub Function that changes the Direction choosen to be inbetween the values
%of +90 and -90
while max(Direction) >= 360 || min(Direction) < 0
    if Direction >= 360
        Direction = Direction - 360;
    elseif Direction < 0
        Direction = Direction + 360;
    end
end
if Direction >= 0 && Direction < 90
    %Quadrant 1 Positive
    D = Direction;
    C = [1] * cosd(D);
    S = [1] * sind(D);
elseif Direction >= 90 && Direction < 180
    %Quadrant 2 Positive
    D = 180 - Direction;
    C = [-1] * cosd(D);
    S = [1] * sind(D);
elseif Direction >= 180 && Direction < 270
    %Quadrant 3 Positive
    D = Direction - 180;
    C = [-1] * cosd(D);
    S = [-1] * sind(D);
elseif Direction >= 270 && Direction < 360
    %Quadrant 4 Positive
    D = 360 - Direction;
    C = [1] * cosd(D);
    S = [-1] * sind(D);
else
    error('Direction value is greater than 360 or less than 0 Degrees');
```

```
end

D;
C;
S;
T = S / C;

end%End of Sub Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

function [R_Wind, R_Start, R_End, R_a] =...
    SUB_Random_Balloon_Trip(Random)
%SubFunction that is used to create Random Data

%Inputs
Number_E = Random(1,1);
Min_Speed = Random(1,2);
if Min_Speed < 0
    Min_Speed = 0;
end
Max_Speed = Random(1,3);

Max_Elevation = Random(2,1);
Min_a = Random(2,2);
if Min_a < 0
    Min_a = 0;
end
Max_a = Random(2,3);

Min_X = Random(3,1);
Min_Y = Random(3,2);
Min_Z = Random(3,3);
if Min_Z < 0
    Min_Z = 0;
end

Max_X = Random(4,1);
Max_Y = Random(4,2);
Max_Z = Random(4,3);

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Randomized Variables
%Random Wind
%Randomized Wind Speed
for i = 1:Number_E
    RWS = 1;
    while RWS == 1
        R_Wind_Speed(i,1) = [Max_Speed * rand];
        if R_Wind_Speed(i,1) <= Min_Speed%check if min is below Min_Speed
            RWS = 1;
        else
            RWS = 0;
```

```
        end
    end
end

%Randomized Wind Direction
R_Direction     =   [360 * rand(Number_E,1)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Randomized Constantly Increasing Elevation
for i = 1:Number_E
    RE = 1;
    while RE == 1
        R_Elevation(i,1) = [Max_Elevation * rand];
        if R_Elevation(i,1) > Max_Elevation * (i / Number_E)
            RE = 1;
        elseif i == 1
            RE = 0;
        elseif R_Elevation(i,1) <= R_Elevation(i-1,1)
            RE = 1;
        else
            RE = 0;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Random Locations
%Random Start Location
%Randomized X Start Location
R_X_S = [Min_X + ( Max_X - Min_X ) * rand];

%Randomized Y Start Location
R_Y_S = [Min_Y + ( Max_Y - Min_Y ) * rand];

%Randomized Z Start Location
R_Z_S = [Min_Z + ( Max_Z - Min_Z ) * rand];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Random End Location
%Randomized X End Location
R_X_E = [Min_X + ( Max_X - Min_X ) * rand];

%Randomized Y End Location
R_Y_E = [Min_Y + ( Max_Y - Min_Y ) * rand];

%Randomized Z End Location
R_Z_E = [Min_Z + ( Max_Z - Min_Z ) * rand];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Random Acceleration
%Randomized Rising Acceleration
RAR = 1;
```

```
while RAR == 1
    R_a_rise    =   [Max_a * rand];
    if R_a_rise <= Min_a
        RAR = 1;
    else
        RAR = 0;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Randomized Falling Acceleration
RAF = 1;
while RAF == 1
    R_a_fall    =   [Max_a * rand];
    if R_a_fall <= Min_a
        RAF = 1;
    else
        RAF = 0;
    end
end

%End of Section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Outputs
R_Wind  =   [R_Wind_Speed, R_Direction, R_Elevation];

R_Start =   [R_X_S, R_Y_S, R_Z_S];

R_End   =   [R_X_E, R_Y_E, R_Z_E];

R_a     =   [R_a_rise, R_a_fall];

end%End of SubFunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%End of SubFunctions
```