3-1997

An Object Oriented Simulation of the C-17 Wingtip Vortices in the Airdrop Environment

Hans J. Petry

AN OBJECT ORIENTED SIMULATION OF THE C-17 WINGTIP VORTICES IN THE AIRDROP ENVIRONMENT

THESIS

Hans J. Petry, Major, USAF

AFIT/GOA/ENS/97M-13

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOA/ENS/97M-13

AN OBJECT ORIENTED SIMULATION OF THE C-17 WINGTIP VORTICES IN THE AIRDROP ENVIRONMENT

THESIS

Hans J. Petry, Major, USAF

AFIT/GOA/ENS/97M-13

THESIS APPROVAL

Student: Hans J. Petry, Major, USAF Class: GOA-97M

Title: An Object-Oriented Simulation of the C-17 Wingtip Vortices in the Airdrop Environment

Defense Date: 4 March 1997

Committee:	Name/Title/Department	Signature
Advisor	T. Glenn Bailey, Lt Col, USAF Assistant Professor Department of Operational Sciences	
Reader	Phillip S. Beran, Associate Professor Department of Aeronautics and Astronautics	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government

AFIT/GOA/ENS/97M-13

AN OBJECT ORIENTED SIMULATION OF THE C-17 WINGTIP VORTICES IN THE AIRDROP ENVIRONMENT

THESIS

Presented to the Faculty of the Graduate School of Engineering
Air Education and Training Command
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Hans J. Petry, B.S., M.S.B.A.
Major, USAF

March 1997

Acknowledgments

I am deeply indebted to my advisor, Lt Col Glenn Bailey, for presenting such an interesting and worthwhile topic and keeping his enthusiasm throughout the whole effort. Even listening to my classmates grunt and groan about the simulation course didn't keep me from choosing this topic. His help in team selection, model verification, and input and output analysis were invaluable in the completion of the project

I would also like to thank William Blake of the Wright Laboratories, without the use of his model and his help in understanding the FORTRAN implementation, this project would not have gone nearly as far as it has. My reader, Dr. Phil Beran also helped decipher the aerodynamics behind the model and eased the transition from the FORTRAN model to my MODSIM interpretation.

Another key player in the model completion is my partner, Capt. Joey Belano. His help in creating the output files and vortex polling method enabled the model to be much easier to use.

Finally, I would like to thank my wife, Karla, and children; Emily; Gwendolyn; and Matthew. Your understanding of why daddy can't come out and play today helped immensely in giving me the time to complete this project and motivation to get it done quickly to move on to more enjoyable undertakings.

Hans J. Petry

Table of Contents

Page

List of Figures

List of Tables

List of Equations

AFIT/GOA/ENS/97M-13

Abstract

This thesis effort focuses on the development of an object-oriented simulation of C-17 personnel airdrop operations and provides a tool for risk assessment of jumper and wingtip vortex interaction. During operational flight testing of the C-17 some jumpers encountered a wingtip vortex from a preceding airplane jeopardizing safety. After stopping live testing, the Air Force and Army realized they had no tool for analyzing the risk of a vortex encounter. Using the initial modeling efforts of the Wright Laboratory, this model expands those efforts to include random aircraft, wind and jumper movement within the simulation using MODSIM III as its language.

Once the model was built, verified, and calibrated, it helped perform a preliminary analysis of jumper risk with varying element spacing and no crosswind. The results of the simulation provided 15 data points with which linear and logistic regression provided an estimation of the marginal rate of change of jumper/vortex encounter rate.

Results showed that the reduced third order linear regression model most closely approximated the data points but that the logistic regression model should fit this type of simulation better. The logistic model failed the lack of fit test but that may be correctable by using the model to collect more data points at smaller intervals.

Using the third order model shows that the encounter rate levels off around 24,000 feet spacing between element leaders at 12%, and stays as high as 11% at 32,000 feet before dropping to 0.4% at 34,000 feet. Further research and model improvements may bring the encounter rate down at the more distant spacing but that is left for post thesis analysis efforts.

AN OBJECT ORIENTED SIMULATION OF THE C-17 WINGTIP VORTICES IN THE AIRDROP ENVIRONMENT

I. INTRODUCTION

Background

One of the major factors used in the recent decision to buy 120 new C-17 aircraft was it's advertised capability for personnel airdrop. Analysis by Air Mobility Command showed that in order to meet the Army's requirements for Strategic Brigade Airdrop operations the Air Force would need 120 C-17's (White, 1996). During operational evaluation and testing of the C-17 for it's strategic brigade airdrop role, it was discovered that there was a problem with vortex and parachute interaction. The preceding planes in the formation generate a much stronger wingtip vortex than the current primary airdrop aircraft, the C-141. The stronger vortex causes parachutes to collapse, oscillate, and descend at faster than safe speeds. Flight testing at Edwards AFB shows statistically that jumpers interact with the vortices at almost twice the rate when jumping from C-141's versus C-17's ("Summary ...", 1996). Furthermore, this statistic is misleading in that it measures interaction only, and not the severity of interaction. One serious incident of interaction actually collapsed the parachute of a jumper and then interfered with the deployment of the reserve chute. (Only luck kept the jumper from serious injury as the main parachute reinflated shortly before impact with the ground.) After this incident, the Army and Air Force agreed to stop live testing and flight test with mannequins to assess the risk associated with jumping from the C-17 in a large formation. Realizing that flight testing all possible conditions would be cost prohibitive, they decided to look for a tool to predict the interaction before actually flight testing. This problem has not been answered and flight testing continues simultaneously with the development of this tool.

The purpose of this thesis is to develop an object oriented simulation of the C-17 wingtip vortices in the airdrop configuration. When used in conjunction with the paratroop objects being developed by Belano (1997), this simulation model will provide the C-17 System Program Office (SPO), and other Air Force and Army agencies, with the capability of assessing paratrooper performance during C-17 formation airdrops. Some specific uses of the model will be to test different formation geometries and their affects on airdrop operations, and jumper risk assessment given current formation and weather conditions. Specifically, the analyst will be able to do a risk assessment of jumper interaction with the vortices after exit from the airplane

Problem Statement

The C-17 test and evaluation community need a tool to better predict the behavior of the wingtip vortices over the drop zone. The current mode they are using, developed by Blake ("Prediction ...", 1996), is a FORTRAN-based, deterministic model. Flight tests have proven that this model is inadequate in predicting the dynamic behavior of the vortices (although it is the best currently available). This thesis proposes to develop an object-oriented simulation tool to model the wingtip vortices of the C-17. This simulation will be written in MODSIM III and incorporate objects representing the wingtip vortex trailing off of each wing. These objects will behave in a similar manner, but allow some random variation to attempt to better predict where the vortices will be as the jumpers descend to the ground. This variation will allow movement of the vortices outside of the straight deterministic equations in trying to account for some of the unknown forces acting on these vortices. Outputs from this simulation include the number of jumpers contacting a critical area of the vortex, which vortex they hit, and the altitude of encounter.

Blake's static prediction model will be converted and expanded to a MODSIM III object that represents dynamic movement of the vortex due to outside influences. The data collected from lidar measurement of the C-17 vortex strength and position was compared with the predicted strength and position using aerodynamic equations. This data then helped define the equations of the position and size of the vortex (Blake, 1996). The simulation will be a discrete-time simulation that constantly updates the position of the airplanes, vortices, and jumpers. Using the property of inheritance, the initial C-17 objects will move in formation and calculate their positions as the flight progresses. Furthermore, when over the dropzone, the paratroop objects will inherit the position data of the C-17 as they exit the aircraft.

Expected Contribution

This model should immediately help the airdrop test and evaluation community make better predictions about the behavior of C-17 vortices over the dropzone. This will allow the analyst to vary formation geometry to minimize the risk of jumper vortex interaction. If this formation is not acceptable for mission accomplishment due to tactical constraints, the user can then assess the risk to jumpers in a formation that is more tactically sound.

II. RELATED WORK

Previous Vortex Models

Hazards associated with wingtip vortices are not a new phenomenon. With the advent of larger aircraft, vortices have become stronger and stronger. The Federal Aviation Administration (FAA) has long recognized these hazards and mandated four mile separation between heavy aircraft and subsequent light aircraft on approach to landing. Recently, they have increased this separation between heavy aircraft (over 300,000 lbs. gross weight); mandated Boeing 757 and 767 aircraft and trailing small aircraft separation to 6 miles; and, redefined what constitutes a small aircraft (up to 41,000 lbs.) (Wald, 1996).

The hazards over dropzones have also been recognized. A 1988 study commissioned by the Military Airlift Command (MAC) concludes that the wingtip vortices of C-130, C-141, and C-5 aircraft significantly hindered personnel airdrop operations at altitudes below 600 feet above ground level (AGL) (Johnson, SEP and DEC 1988). These studies recommend that airdrop operations be conducted above that altitude to maintain safety of the jumpers during descent. Personnel drops from C-5 aircraft were deemed infeasible and have not been developed further (Johnson, DEC 1988).

MAC Project 15-105-86 (Johnson, SEP 1988) includes a computer model for vortex jumper interaction. A verbal description of the model in Appendix C of the MAC Project report describes a model similar to what is proposed in this paper, although this model is in a DOS-based, non-object-oriented, BASIC format. The characteristics of the vortices were measured by hot film anemometers at the National Oceanic and

Atmospheric Administration (NOAA) test facility at Idaho Falls, Idaho. This type of testing, while the best available at the time for measurement of vortices, is somewhat flawed by the proximity of the tower to the ground. More recent advances in the measurement of vortices (Hannon, 1995, Thomson, 1995) were used by Blake in the development of his current model (Blake, 1996).

Blake's model is the most recent work in this area (Blake, 1996). His model initially included interaction between vortices through time integration. It did not include interaction with the ground since the predicted descent rate put the vortex at 200 feet AGL at a point when its predicted strength no longer effects the jumpers. In the fall of 1996, Blake modified his model to no longer take into consideration the interaction between vortices of separate aircraft because the spread of the formation was so great that one vortex pair had little impact on the other airplanes' vortices (Blake, Personal Interview, 1996). At this time he also included ground effect in the model, enabling the vortices of a pair to separate quicker when approaching the ground while slowing their descent rate. The vortices now asymptotically approach the ground but never hit. The data used to measure the characteristics of the C-17 vortices was obtained by lidar measuring equipment at Edwards AFB, CA. A laser radar measuring the turbulence inside the vortex core, this method of measurement eliminates some of the problems with the hot film anemometer and ground effect (Hannon, 1995). It represents the vortex in free air better than the hot film anemometer. The lidar data forms the basis for Blake's model, which in turn is the basis for representing the vortices in our simulation.

1988 Military Airlift Command Project

In 1988 the Military Airlift Command, which has since become Air Mobility Command, commissioned a study of airdrop operations at very low altitude (Johnson, SEP and DEC, 1988). Supposedly the first ever to study the effects of aircraft vortices on parachute systems, its purpose was to gather information and measure what the effects of the vortices on airdrop operations from 300 to 600 feet above ground level (AGL). (Normal airdrop operations were conducted at 800 to 1000 feet AGL.) The test was conducted in three phases, with the first being the collection of vortex measurement information. The second phase determined what effect the vortex has on a parachute, while the last developed a computer model to predict the effects at low altitude. The first phase will be covered in more detail in the next section of this paper on hot film anemmometer measurement.

The second phase consisted of mounting self contained smoke generators on the lead aircraft, and flying the trailing plane over the vortices while releasing dummies to see what the effect of vortex encounter is. Encounters that occurred on the outer edge of the vortex the showed a descent rate slowed by 60%, while those that penetrated between the vortices saw increased descent rates up to 140%. Encounters that occurred before parachute stabilization had no effect on the parachute system. Those that occurred after parachute stabilization affected the parachutes in various ways, with the most severe effect being a chute collapse for 1 to 3 seconds. Other effects were partial deflation of the canopy, oscillation of the load, increased or decreased descent rate, collisions, entanglement, and dragging of the load on the surface. Another conclusion was that the effects of an encounter with a vortex were not dependent on the aircraft type in the study (C-130 and C-141). An expanded study including the C-5 arrived at a similar conclusion.

The last phase developed a computer model to predict the encounter rate of different aircraft, loads, altitudes, and formation geometries. One finding from this model was that interactions between the vortices and canopies ranged from 0% to 93%. It also concluded that the possibility of more than one vortex affecting any one parachute's descent was very small; however, this may result from the model coding and is suspect, especially if the vortex life is longer than the 90 seconds predicted for the C-130 or C-141.

This test had five major objectives encompassed in the three phases. The first objective was to determine the vital characteristics of the vortices such as size, strength and duration. The second was to determine the maximum change in descent rates of different parachutes after vortex encounter. The third objective was to assess the hazards associated with vortex encounter. The fourth objective was to determine a vertical safety correction to use in the very low environment. The fifth and last objective was to determine minimum and maximum in-trail distances for formations during airdrop operations in a very low environment. While this test looked at the dynamics of the very low airdrop environment, the results can apply to the work in Chapter III.

Hot Film Anemometer Measurements

After searching for ways to measure the size and strength of the vortex, the testers in the MAC Project contacted the Federal Aviation Administration about using a system in Idaho to measure vortices (Clawson, 1988). Previously used for tests on heavy transport category aircraft and B-52 vortex measurement, this hot film anemometer is a wire filament heated to 250°C. The measurement of the amount of electricity needed to maintain that temperature is used to calculate the wind speed at that point. The tower in Idaho stands 200 feet above the ground and is filled with filaments every 3 feet on the top 100 feet of the tower and every 6 feet on the bottom half. Additionally, there are 26 extra filaments mounted on 13 towers 11 feet tall to try to measure any vortices that descend to the ground. C-130, C-141, and C-5 aircraft were used for the fly-bys to characterize the vortices for each airplane (Table1).

Table 1. Vortex Characteristics from 1988 Test of C-130, C-141, and C-5 Aircraft

Aircraft Type	Maximum Core Velocity (fps)	Minimum Core Velocity (fps)	Maximum Age (seconds)	Minimum Age (seconds)	Maximum Descent Rate (fps)	Minimum Descent Rate (fps)
C-130	153	12	97	6	7.0	0.8
C-141	162	8	176	6	13.5	0.4
C-5/A	268	11	109	6	9.9	0.9

The testers warn that this data may not represent the true vortex characteristics of these airplanes because of varying weather conditions. They expect that some C-5 vortices could live longer, although a strong wind on all of the days of C-5 testing and calm winds on the days of C-141 testing prevented a more definitive answer. In general the observed characteristics were expected with the C-5, C-141 and C-130, in decreasing order of severity.

Blake's Model

The latest model to address the problem of paratroop and vortex interactions is one by Blake (Blake, 1996). He models the vortices as a record of six data points taken at evenly spaced intervals behind the vortex generating aircraft using Eulers explicit time integration. The six data points are the x, y , and z coordinate, the strength, the radius,

Equation 1. Basic Vortex Circulation Formula

$$\Gamma = \frac{4W}{\pi \rho v_{\infty} b}$$

Equation 2. Vertical Velocity Formula

$$V_D = \frac{-\Gamma}{2\pi b_{eff}}$$

and the age of the vortex. Blake used an interval (step size) of 200 feet, but suggested I use 100 feet for more accuracy. With this information he iteratively calculates the six values for each subsequent point. (The FORTRAN code for Blake's model is in Appendix 4). Each vortex is varying models used to calculate its movement as a single ship vortex field, or to account for the interactions between all of the vortices generated by a large formation. Subsequent testing has led Blake to conclude that given at least 3000 feet spacing between aircraft within an element, the single ship model is more useful in predicting vortex position (Blake, Personal Interview, 22 JAN 1997). This model has been invaluable in establishing an acceptable formation element geometry, but does not account for any variation in crosswind strength or aircraft position. This inability to account for the uncertainty of the vortex position at greater distances makes it difficult to predict and find an acceptable spacing between elements of the formation.

Blake used the elliptical load of a fully rolled up vortex to calculate the vortex strength and vertical velocity (Equations 1 and 2). The main formulae used in Blake's model calculate the motion of the vortex based on the information at the previous interval

(Figure 1, Table 2). The main calculations involve the horizontal and vertical velocities; the next y and z positions (using Eulers explicit time integration); vortex circulation

Figure 1. Blake's Formulae for Vortex Position Prediction.

$$vage = xv \,/\, vf$$

$$gam = wt(i) \,/\, (rho * vf * bp) * g\,\mathrm{mod}$$

$$gamd = gam \quad \text{if } vage \leq 60$$

$$gamd = gam * (60 \,/\, vage) \quad \text{if } vage > 60$$

$$vd = -1 * gamd(i) \,/\, (2 * pi) * 4 * hag ** 2 \,/\, (bp * (bp ** 2 + 4 * hag ** 2))$$

$$yd = \pm 1 * gamd(i) \,/\, (2 * pi) * bp ** 2 \,/\, (2 * hag * (bp ** 2 + 4 * hag ** 2))$$

$$y0(i) = y0(i) + vs * delt + yd * delt$$

$$z0(i) = z0(i) + vd * delt$$

$$radius = gamd \,/\, (2 * pi * threshold)$$

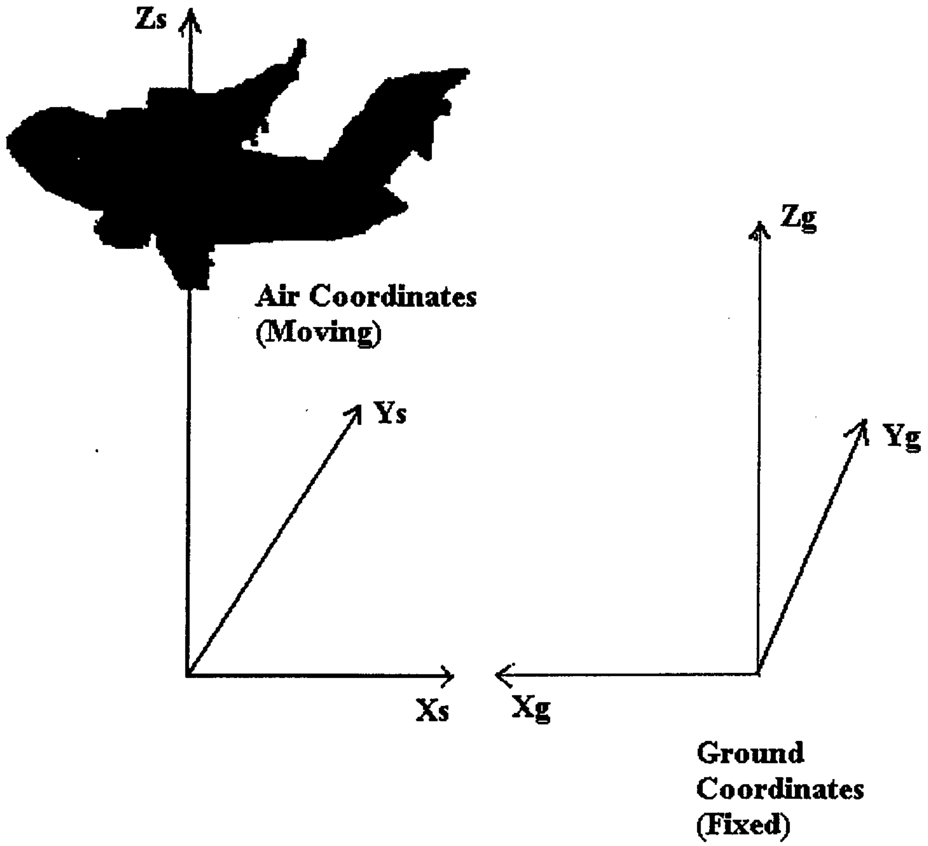
Where table 2 shows notational definitions

Table 2. Table of Blake's Notation.

Symbol	Meaning
vage	Vortex age (time since aircraft passed)
xv	Aircraft's formation position relative to lead. (In-trail)
vf	Velocity of Aircraft (ft/sec)
gam	Gamma, vortex circulation strength
$wt(i)$	Aircraft Weight
rho	Air density
gmod	Modification to Vortex strength from lidar data (0.8)
gamd	decayed vortex strength
vd	Vortex vertical velocity
hag	Height above ground
bp	Span of fully rolled vortex. (pi/4*beff)
beff	Effective wing span
yd	Horizontal velocity. Positive for left vortex, negative for right vortex
$y0(i)$	y Position at this instance. (Lateral distance from center of first aircraft)
vs	Velocity of crosswind
delt	Time since last point
$z0(i)$	z Position at this instance. (Altitude)
radius	Vortex radius to reach the threshold velocity
threshold	Threshold swirl velocity (Blake used 30 ft/sec)

strength; vortex decay; vortex age; and, radius for threshold swirl velocity. The x position is evenly spaced at 200 feet intervals. The coordinate system Blake uses is fixed to the first aircraft, we adopt the same convention and use a fixed ground coordinate system as well (Figure 2). By fixing the ground system we are able to collect the landing position of the jumpers on the ground since the airplane coordinate system is in constant motion.

Figure 2. Air and Ground Coordinate Systems



After presenting the preliminary progress on the model to the U.S. Army Operational Test and Evaluation Command, Babarsky pointed out some errors in Blake's modeling of vortex separation (Babarsky, 1997). The most notable error is that Blake uses a constant span *(bp)* to calculate the induced velocity of the vortices of a given pair on each other instead of the actual separation between the center of each vortex. Blake agrees that this is a correct assessment of his model and has since changed his code to reflect this correction (Blake, Personal Interview, 22 JAN 97). This model also incorporates this change by adding two fields to the vortex representation to track the actual separation of the vortices.

Lidar Measurement of the Vortex Core

Pulsed solid state coherent lidar equipment has measured aircraft wake vortices since a joint 1992 FAA and National Aeronautics and Space Administration (NASA) project (Hannon, 1995). Lidar measurement of wingtip vortices uses a ground based pulsed laser to measure the velocity of small particles (aerosols) in the air. Lidar is also used for windshear detection; optical air data sensing; clear air turbulence and gust front detection; and, wind profiling. The shift in frequency of the pulsed lidar can be manipulated through statistical analysis to arrive at a maximum likelihood estimator of vortex position, size, strength, and velocity. It is estimated that the position data is accurate on the order of 3 meters in height and range. Blake incorporates the lidar flight test data of the C-17 vortex as one option for vortex generation in his model. Since no detailed statistical analysis has been performed on this data, it remains one area available of improvement.

Proposed Methodology

The first step in model development is translating Blake's model into an object-oriented simulation using MODSIM III. Once running and verified by benchmark checking against the original results, modifications to the behavior of the vortices can begin. The two ways to add variation into the simulation of the vortex position are by varying the winds and varying the formation position of trailing aircraft. Therefore, the model accepts up to three different crosswind velocities, all based on a normal probability distribution with a user input standard deviation and altitude for change. It also accepts an integer value for the tolerance within which a trailing aircraft must remain from planned formation position. These changes refine Blakes's model while allowing for more analysis to find an acceptable formation spacing.

III. METHODOLOGY

Formal Formulation

During operational testing and evaluation of the capability of the C-17 in the airdrop mission, vortex encounters with the jumpers were discovered. This problem was addressed by varying the formation spacing both within and between elements through vortex prediction modeling and flight testing, with the goal to minimize jumper and vortex encounters. Specifically, within an element there can be no encounters, while between elements the encounter rate must be kept to a minimum.

Current modeling limitations do not allow for any way to assess the risk of encounter without extensive flight testing. Although many factors go into the encounter rate of jumpers with vortices, they can be placed into three distinct groups: airplane parameters, weather conditions, and formation parameters. The airplane parameters include such things as airspeed, weight, and wing characteristics, while weather conditions include headwind, crosswind, temperature, and pressure altitude. The final factor is the formation itself, and is composed of the in-trail and lateral spacing between aircraft within an element, and the spacing between element leaders. Of these factors, only the airplane and formation parameters are controllable.

The airplane parameters are limited by aircraft performance and are variable within a certain range. The normal airspeed for personnel airdrop operations is 130 to 135 knots indicated airspeed (KIAS). The actual speed that the airplane can safely fly is a factor of weight, altitude, and configuration, but for most conditions 130 knots approaches the lower limit for safe airdrop operations. Conversely, if the airplane speed is increased to more than 135 knots, then it is the jumpers themselves that become the limiting factor due to opening shock and parachute integrity. Furthermore, while the aircraft weight limits are bound by the limits of the airplane itself, the drop zone imposes even more restrictions. Very rarely would the plane ever be over the drop zone at maximum weight due to the distance of the drop zone from the point of departure. Similarly, it would be difficult for the planes to fly close to minimum weight due to fuel requirements for the return flight.

The most controllable parameter is the formation itself, although this is subject to some restrictions. The width of the formation is limited by the width of the drop zone itself. If the formation becomes too wide, then we limit the places we can use airborne assault to areas of that width or larger. This impacts where airborne operations can be conducted and takes away some flexibility in choosing a location for the assault. Even if the drop zone itself is wide enough to handle a wide formation, the dispersion of combat troops once they hit the ground affects the combat capabilities of the paratroopers. Limitations on the length of the formation are driven by the time requirements of the Army for the closure of the airdrop portion of the Strategic Brigade Airdrop mission. The challenge, then, is to find a formation that minimizes the risk to the jumper from vortex encounters while not exceeding the width, time, and aircraft limits.

Solution Techniques

One solution technique that is readily apparent is flight testing. However, experience with flight testing shows this method to be very time consuming and expensive. Any failure necessitates starting over from the beginning with very little idea of what will happen with each subsequent test. Also, there is little or no control over the weather conditions with this approach. Consequently, this approach was quickly abandoned by the test and evaluation community as not feasible.

Blake's model was the first modeling step in the airdrop problem solution. The model predicts where a vortex will be with a given set of weather and airplane conditions. The formation parameters are then superimposed on this information to give an analytical approach to predicting whether or not a vortex encounter is likely. The limitation to this approach is that there was no varying of wind conditions with time or altitude. Also, the position of the planes within the formation remain constant.

Because of the limitations of a static prediction model, this problem lends itself very well to a simulation that incorporates random behavior of the variable elements. With multiple runs one can test different formation parameters with controlled weather conditions and aircraft parameters to arrive at a valid prediction interval of jumper vortex encounter rate. Simulation gives the added benefit of variable behavior of the winds, airplane position within the formation, and jumper trajectory during the descent, thus examining a whole range of operating conditions with little or no user intervention in data collection. These characteristics will then give a broader perspective to the analysis of the risk faced by the jumpers as they make their descent to the ground.

A simulation also gives the added benefit of possible uses outside of the scope of the present problem. Once validated, a simulation could be used to predict the encounter rate of a specific planned mission with the prevailing weather and location conditions expected during that mission. This could aid battle planners in accurately assessing how many jumpers would be available for combat duties when they hit the ground.

Object-oriented simulation itself allows a building block approach. Once the objects are modeled and verified, they can be added into the simulation in as large a number as necessary for the analysis. (The model is capable of running as many as 120 C-17's with 102 jumpers per aircraft.) Another benefit of using a simulation language is it takes care of the 'bookkeeping' of all the data. For instance, a vortex of length 41,000 feet with six data points at each 100 foot increment: each aircraft having two vortices, and the position updated every ½ second, the amount of data processed is quite large. Thus, each airplane requires 4,920 data storage points, updated each ½ second, with over 7,380 calculations for vortex definition alone. By adding the position updates of the airplanes and jumpers, and the search of all preceding vortices, one begins to see the complexity of this simulation.

Implementation

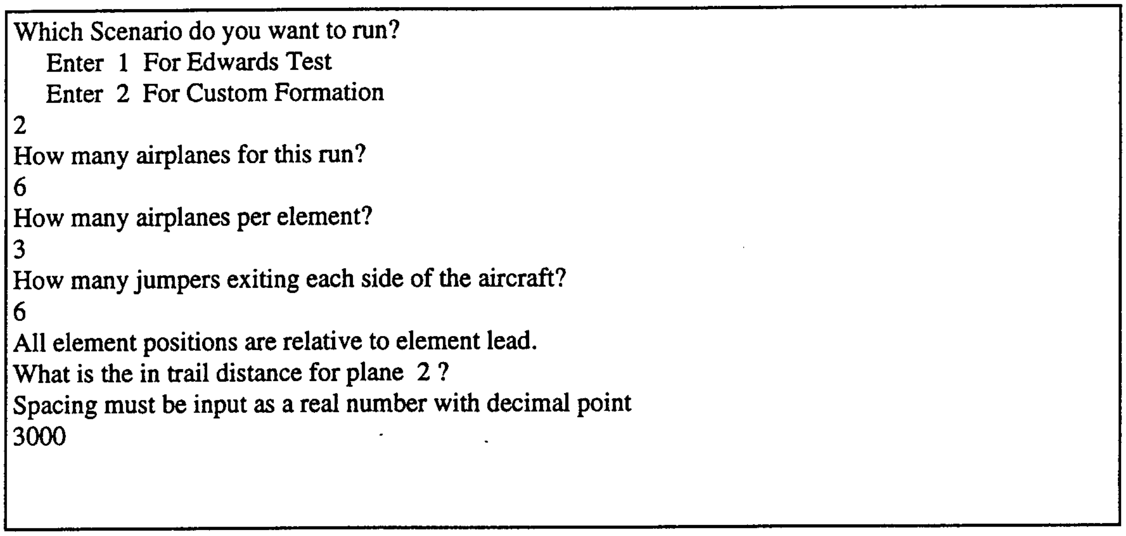
This section of the paper describes the airdrop model. The simulation includes four basic objects: a control object; C17 object; vortex object; and, a paratroop object. This thesis develops the first three; for more information on the paratroop object see Belano (1997). Although there was close collaboration with Belano, the following describes those portions of the program directly related to the first three objects.

The simulation uses MODSIM III as its language of implementation. The MODSIM III compiler uses C++ as the base language and compiles the modules into a C++ file, which in turn is linked to the main modules to form an executable file. Once compiled, the executable file prompts a few questions before starting the simulation. Output is displayed both on screen and saved to individual files, with input file names for post-simulation analysis. The inputs necessary to start the simulation can be inserted manually or with a data file. Once all questions are answered, the simulation starts by creating the control object, which in turn creates the C-17 objects and their associated left and right vortex objects. The control object keeps track of the formation position and other information about the C-17 object. The C-17 object passes its position and aircraft information to the vortex objects, who then update their position taking into account the current wind information. Once all these objects are created and in place, the formation begins to fly towards the drop zone. The lead aircraft is initially positioned 300 feet short of the drop zone so shortly after one second it enters above the drop zone and flies to its computed air release point. Once over this point, the control object tells the C-17 to create the jumpers and let them jump. The jumper objects exit the airplane while asking the C-17 for initial position data, and start calculating their descent towards the ground updating their position and outputting their exit information to the screen and file. On the way down they check their position every ½ second against the position of every vortex generated from every plane in front of the plane they jumped from. If they are inside the vortex, they output their number, airplane, which vortex from which airplane, altitude, distance to the center of the vortex, and the simulation time. Jumpers continue jumping and the planes continue flying until all aircraft in the formation have crossed the drop zone and all of the jumpers have landed. As the jumpers land they output their landing information to the screen and scatter file. Once all the jumpers objects land, the simulation disposes of all objects, then starts over until it completes the number of repetitions required. All repetitions occur without any required input and without resetting the random numbers used within the simulation.

Input

The model currently runs either the flight test scenario at Edwards AFB or a custom formation in an interactive mode through a series of questions. The program goes through a series of questions so the operator can input the desired parameters (Figure 3). The first question is the number of airplanes in the formation followed by the number of airplanes within each element (by assumption, all elements have the same spacing between aircraft). The program then asks for the in-trail and cross-track spacing for each aircraft in the element. Once the user enters the element spacing, the program asks for the desired spacing between element leaders. This is followed by the weight of all aircraft,

Figure 3. Sample Input Screen.



```
Which Scenario do you want to run?
   Enter  1  For Edwards Test
   Enter  2  For Custom Formation
2
How many airplanes for this run?
6
How many airplanes per element?
3
How many jumpers exiting each side of the aircraft?
6
All element positions are relative to element lead.
What is the in trail distance for plane  2 ?
Spacing must be input as a real number with decimal point
3000
```

Figure 3 (cont.). Sample Input Screen.

```
What is the weight of all aircraft?
385000
What is the airdrop altitude? (ft AGL) (Real number with decimal)
900
What is the airspeed of the formation in knots?
135
What is the Cross Wind Component? (knots)
Input up to three different winds with altitude where wind changes
and standard deviation of the normal distribution for wind variation
(All eight values must be entered. If constant wind, then zeros can be
entered for the last seven values.)
i.e. 5 .5 500 10 1 200 0 .25 would indicate 5 knots with a normal
distribution and a standard deviation of .5 at drop altitude till 500
feet AGL, then 10 knots with a standard deviation of 1 until 200 feet AGL,
 then calm to the ground with a standard deviation of .25
0 .1 0 0 0 0 0 0
What is the Head Wind Component? (knots)
0
What is the air density factor (rho)?
0.002309
ENTER OUTPUT FILENAME (WITH NO SUFFUX EXTENSION) FOR JUMPER INFORMATION
DATA :
Seed332J
ENTER OUTPUT FILENAME (WITH NO SUFFIX EXTENSION) FOR ENCOUNTER DATA :
Seed332E
ENTER OUTPUT FILENAME (WITH NO SUFFIX EXTENSION) FOR SCATTER   DATA :
Seed332S
ENTER AN INTEGER (BETWEEN 1 AND 10) FOR THE RANDOM SEED INPUT FOR JUMPERS
3
PLEASE CHOOSE ONE OF THE FOLLOWING OPTIONS:
   1 - WRITE INDIVIDUAL PARATROOP TRAJECTORY INFORMATION TO SEPARATE FILES
   0 - DO NOT WRITE TRAJECTORY INFORMATION
0
```

the airdrop altitude above ground, the formation speed, the cross wind and head wind components, and the air density factor.

The crosswind input allows for three different winds to be in the program changing with altitude, with eight inputs required for the crosswind. The first number is the magnitude of the crosswind component in knots at drop altitude. Next is the standard deviation used for the normal distribution the wind uses to update every ½ second. The third number is the altitude at which the wind changes. The next three numbers repeat the same questions for winds between the drop altitude winds and the ground winds, while the last two are the magnitude of the crosswind component and standard deviation for the ground winds. This information is then passed into the objects. All input coding is in the input module and was coded jointly with Belano.

CONTROL OBJECT

The control object, named *TotalVortexObj*, is the first object created and contains information about each of the other objects. The object itself is a large multidimensional database, since the main part is an array with a self defined record type that contains records with various field names. Some of the fields of the record are numeric while others are objects. The fields within the control object keep track of the position of the *C17Obj, LeftVortexObj*, and *RightVortexObj* objects during the simulation.

When the program creates the control object, it creates the *C17Obj* based on the input number of airplanes. Each C-17 object then creates two vortex objects; one for the left wing and one for the right wing. In affect, a field within the array contains an object, which itself contains two other objects that are also arrays.

Each airplane in the formation has a record associated with it in the control object. The fields of this record include the 3-axis air position; 3-axis ground position; aircraft speed; aircraft weight; the aircraft itself; the length of the vortex calculation; and, a true-or-false field indicating if the plane has a full load of jumpers. The number of jumpers is an input, and by assumption each aircraft object has the same weight, number of jumpers, and speed. If analysis requires different parameters between aircraft, the program can be modified to accommodate these requirements.

The methods associated with the control object are *ObjInit*, *PositionInformation*, *Fly*, and *greenLight*. The *ObjInit* method calls the *PositionInformation* method and both are described at the same time. Since all arrays in MODSIM initialize with values of zero, the initialization method takes the input data and loads all of the information into the array before the simulation begins. Each aircraft is placed into its proper formation position with spacing behind the lead aircraft, and either left, right, or directly in trail based on the information provided during input. If the user enters a value for the in-trail or lateral tolerance box the aircraft position is moved to a position within that box using a uniform random number draw on both the in-trail tolerance and lateral tolerance. For example, if the box were 500 feet in-trail and 200 feet lateral, then each airplane draws a different random number from -500 feet to 500 feet and -200 feet to 200 feet. These numbers are then added to the planned position of the aircraft to put the plane in a random position within the tolerance box. All airplanes fly at the same airdrop altitude. The last step of this method asks the C-17 to update the vortex position based on its position in the formation. (This information is not available at the creation of the vortex; hence, they are not in the proper position.) After initialization each aircraft in the formation knows its air and ground position, and the position of the trailing vortices. The lead aircraft's position is 300 feet short of the computed air release point and aligned perfectly with the drop zone axis. All other aircraft are positioned relative to the lead aircraft according to the spacing input by the user.

The *Fly* method actually moves the formation by changing the position of each *C17Obj* each ½ second, then asking the *C-17Obj* to update its vortex position. All aircraft position information is contained within the Control Object and passed to the aircraft when necessary. The *Fly* method assumes that the lead aircraft remains on the desired track along the dropzone and only updates the *x* ground position for lead. The following aircraft movement is more complicated because they are allowed to move fore and aft and laterally in relationship to the lead aircraft. Getting the aircraft to move within the box required the addition of six extra fields within the control object (the *next* and *plan* position for each axis of the air coordinate system). The *plan* position holds the information input by the user on formation spacing. If the aircraft stayed in perfect formation position it would always be in the *plan* position. The *next* position holds the position that the airplane is flying towards. Once it reaches the *next* position the airplane check its position against the *plan* position. If its position does not equal the *plan* position the *plan* becomes the *next* position, while if it does equal the *plan* position, the program draws a new random position within the box using a uniform draw about the *plan* position and assigns that value to *next.* Once the formation starts flying, planes move back and forth into and out of *plan* position until the simulation stops. This movement occurs one foot per time increment simultaneously on the *x* and *y* axis. Provisions for movement in the *z* axis are in the program but not implemented. If the plane arrives at the *xsplan* position before reaching the *ysplan* position it stays in position on the *x* axis until it reaches position on both axes.

The *greenLight* method asks the *C17Obj* to turn on the green light when the aircraft hits the computed air release point. This is defined as the x coordinate of the airplane changing from negative to positive and the plane containing a full load. The C-17 Object contains the actual green light method.

C-17 OBJECT

The *C17Obj* itself is relatively simple when compared to the control object and the vortex object. The main purpose of this object is to generate a vortex characteristic of the wing represented in the object. It contains information about the C-17 wing that is necessary for the vortex calculations; i.e. wing area, aspect ratio, and wing span. The control object passes in aircraft weight, speed, and air density as necessary. (This object could easily be modified to represent the C-141, C-5, or C130 aircraft.) Its methods include *ObjInit*, *VortexPosition*, *FlyPosition*, and *greenLight.*

The *ObjInit* method creates both the left and the right vortex. Since the C-17 does not know its position information upon creation, the vortices do not know the right positions or possess the right length on initialization. For this reason, the control object calls the *VortexPosition* method during initialization. When the necessary information is in place, the control object tells the C-17 where it is, while the C-17 tells the vortices where to start.

This same process occurs within the *FlyPosition* method when the airplane moves from its original position since the vortex must move as well. However, once the aircraft passes a point in space, the vortex it creates is independent of subsequent aircraft movement and does not follow the aircraft movement.. For this reason the vortex update methods are different for the *VortexPosition* and the *FlightPosition* updates. These differences are explained in the vortex *Update* and *BackwardUpdate* methods

The last method in the *C17Obj* is the *greenLight* method that creates the jumpers. To cut down on the required number of calculations, the jumpers are not created until they exit the aircraft. The *greenLight* method triggers the creation of the jumper objects. At ½ second intervals jumpers exit the aircraft, while at the same time requesting the position, speed, and altitude of their aircraft. The exit side of the airplane is recorded when creating the jumpers, thus allowing the model to know which jumper encounters which vortex. This method was developed and coded by Belano(1997).

VORTEX OBJECT

Since a real vortex is a continuous turbulent area generated by a passing aircraft, it is very difficult to implement as an object; therefore, we converted this continuous vortex into discrete points behind the aircraft. Choosing a step size of 100 feet (on the recommendation of the Wright Laboratory) each vortex becomes an array of its characteristics spaced at 100 foot intervals. Each record initially contained an *x*, *y*, and *z* coordinate, with the vortex *strength* measured by lidar, *age*, and *radius* but was expanded to include an no wind *y* position and the aircraft *y* position at the time of vortex creation. These additions were added to correct the error pointed out by Babarsky. We define *x* as the distance behind the aircraft plus the aircraft's distance behind the lead aircraft, *y* the lateral distance from the center of the aircraft plus the aircraft's lateral distance from the lead aircraft, and *z* the altitude above the ground.

The vortex strength uses the theoretical vortex strength as calculated from the Wright Labs, and modified by a factor of 0.8 (Blake, 1996). This factor accounts for the difference between the theoretical vortex strength and the vortex strength as measured by lidar. Different aircraft have a different modification factor and this factor must be an aircraft attribute for the vortex to calculate correctly with different aircraft objects. The vortex age is purely a function of aircraft speed. To calculate the radius, we set a threshold swirl velocity and use the strength to find the distance from the actual point where the swirl velocity exceeds that threshold (the threshold is currently set at 20 ft/sec and is modifiable depending on the type of analysis desired). The strength remains constant for some time after the aircraft passes and then begins to decay; the current delay is 60 seconds based on the lidar data. Provisions are in the code for calculating interactions between vortices using horizontal and vertical velocity components of the vortices, but is not implemented due to the spacing between aircraft and the small likelihood that two vortices would actually significantly influence another vortex pair. This likelihood is small for two reasons. The first is that the influence of one vortex on another is inversely related to the distance between those vortices and the second is that one vortex will have its pair influencing the same vortex in the opposite direction at a similar distance and somewhat canceling the effect of its pair has on the motion..

The methods for the vortex objects, *LeftVortexObj* and *RightVortexObj,* are *ObjInit* and *Update*, *BackwardUpdate*, and *writeData.* The *ObjInit* of the vortex again occurs upon its creation. Since not all information is available at creation, the control object tells the *C17Obj* to *Update* as the last step of its *ObjInit* method, which in turn tells the vortices to *Update*. This *Update* calculates the position of each vortex until a point 42,000 feet behind the generating aircraft. (Non-implemented provisions in the code allow the vortex calculation to be extended until 20,000 feet behind the last aircraft in the formation. This distance allows time for all jumpers to exit the aircraft and descend to the ground before the vortex calculations end.) The vortex calculations increment the *x* coordinate by a step size of 100 ft for each record. The program calculates the first point, then uses vortex strength to calculate a vertical and horizontal speed (Figure 4). The right and left vortex objects differ only in *yCord* and *NDyCord.* The difference is the sign of *bp/2.0* is negative for the left vortex.

Figure 4. First Vortex Point Formulae for *rightVortex* Object

```
        gam  := weight/(rho*vf*bp)*gmod;
        CompletePosition[1].xCord  := 100.0+xs;           {100 ft behind aircraft}
        CompletePosition[1].yCord  := ys+bp/2.0;          {Off right wing of aircraft}
        CompletePosition[1].zCord  := altitude;           {Aircraft altitude}
        CompletePosition[1].Vage   := delx/vf;
        CompletePosition[1].Gamd   := gam;                {Use gam from above}
        CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);       {Threshold set at
20 ft/sec}
        CompletePosition[1].NDyCord := ys + bp/2.0;       {Keep no drift spacing}
        CompletePosition[1].Airy   := ys;                 {Aircraft position when vortex point
created}

Where
        gam = vortex strength
        weight = aircraft weight
        rho = air density
        vf = aircraft velocity (ft/sec)
        bp = span of rolled up vortex
        CompletePosition[1].xCord = position behind lead aircraft
        xs = aircraft air position behind lead aircraft
        CompletePosition[1].yCord = position left (positive) or right (negative) of lead aircraft
        ys = aircraft air position left or right of lead
        CompletePosition[1].zCord = altitude
        delx = step size
        CompletePosition[1].Vage = age of the vortex
        CompletePosition[1].Gamd = decayed vortex strength
        CompletePosition[1].radius = radius of vortex with swirl velocity greater than threshold
        CompletePosition[1].NDyCord = no drift position left or right of lead position
        CompletePosition[1].Airy = Aircraft position when vortex point created
```

Crosswind speed is updated every ½ second using a normal distribution and the input standard deviation. The *y* and *z* coordinates are then updated based on the position of the previous point and the calculated velocities (Figure 5).

Figure 5 . Subsequent Vortex Points and Velocity Formulae for *rightVortex* Object.

```
        CompletePosition[i].xCord := xs + (FLOAT(i)*delx);
        sD    := ABS(CompletePosition[i-1].NDyCord-CompletePosition[i-1].Airy);
        CompletePosition[i].Airy   := ys;
        vd := (-1.0*CompletePosition[i-1].Gamd)/(4.0*pi)*((CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord)/(sD*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord))));
        yd := ((CompletePosition[i-1].Gamd)/(4.0*pi))*(sD*sD)/(CompletePosition[i-
1].zCord*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord)));
        CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
        CompletePosition[i].NDyCord := CompletePosition[i-1].NDyCord+yd*delt;
        IF CompletePosition[i].zCord <= ShearAlt1;
                IF CompletePosition[i].zCord <= ShearAlt2;
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs3*delt+yd*delt;
                ELSE
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs2*delt+yd*delt;
                END IF;
        ELSE
                CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs1*delt+yd*delt;
        END IF;
        CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
        IF CompletePosition[i].Vage <= 60.0;
        CompletePosition[i].Gamd  := gam;
        ELSE
        CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
        END IF;
        CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);

        Where (all previous variables in Figure 4 the same)
                FLOAT(i) = real number conversion of step number
                sD = distance of vortex from center of airplane (updated bp)
                vd = vertical velocity
                yd = horizontal velocity
                delt = time change from previous point (function of speed)
                ShearAlt1 = highest altitude for wind shear
                ShearAlt2 = lowest altitude for wind shear
                vs3 = ground winds up to ShearAlt2
                vs2 = winds between ShearAlt2 and ShearAlt1
                vs1 = winds from ShearAlt1 to drop altitude
```

This iterative process continues until reaching the last step and the vortex extends either 42,000 feet (or past the end of the formation).

The *BackwardUpdate* is slightly different from the *Update*. The same formulae used for the calculations are applied with a new first point; and, then from the last point up to the second point since the aircraft can actually move laterally and vertically after it passes the point of generation (if the update were to happen from the airplane to the end the entire vortex would move with the aircraft). The only factor that affects the vortex over time is the crosswind (up and down drafts also affect the vortex but are not modeled) and not changes in the aircraft flight path; therefore, once the aircraft generates a point its path is set except for changes in the wind. However, there must be a continuous path from each point to the aircraft. For this reason, during a *BackwardUpdate*, the first point behind the aircraft is generated exactly the same as before, assuming that any change in aircraft position in each ½ second interval is small. Then, instead of sequentially updating from point number two to the end the program, it starts at the end and updates the last point with the information of the point prior to it. When point number 2 (200 feet behind the aircraft) updates, it uses the information from the newly created point 1 to update, thus using the information from the previous time step without saving it.

The last method, *writeData,* was used for verification and validation of the vortex object and remains in the code (but is not active). It includes a method to print the database representing the vortex at any point in time, and its output to the screen contains each vortex representation sequentially.

Output

There are many possible types of output available within this program. Not all of the output statements are active at this time, but remain in the program with comment statements. As mentioned previously, the entire vortex representation is a possible output for verification and validation of the vortex position and characteristics. The air and ground position of each aircraft can be output at each time step, or at certain intervals. Most of the remaining output options come from Belano's objects. If necessary for verification, more output can be specified at any point in the code.

Since there was such close collaboration between Belano and myself, I feel it necessary to describe the output of two files in more detail; the jumper file and the encounter file. Although coded by Belano and contained within the paratroop objects, we both decided what types of output would be useful for further analysis when the output files were coded. The program creates the jumper and encounter files with the names provided by the user. The first file contains all of the information about the jumper, while the second contains information about all encounters. Since the program loops through a certain number of repetitions, the repetition number is the first entry for each line of output. Table 3 shows the meaning of the lines of output, while Tables 4 and 5 show the output from the jumper and encounter files.

Table 3. Field Meaning of Output Files.

	1	2	3	4	5	6	7	8	9
Jumper	Repetition Number	Plane Number	Exit Time	Jumper Number	Weight	X Ground Coordinate	Y Ground Coordinate	Altitude	
Encounter	Repetition Number	Jumper Number	Jumper Plane	Vortex Type	Vortex Plane	Jumper Altitude	Distance	Step Number	Sim Time

Table 4. Sample Output from Jumper File.

1	2	13.000000	1R	253.743686	14.441750	743.25	900.00
1	2	13.000000	1L	242.434541	14.441750	724.75	900.00
1	2	13.500001	2R	256.747860	127.574125	742.25	900.00
1	2	13.500001	2L	219.935088	127.574125	723.75	900.00

Table 5. Sample Output from Encounter File.

1	1R	4	RV	1	435.493262	19.170270	218.	98.492000
1	1R	4	RV	1	426.637141	15.126430	219.	98.992000
1	1R	4	RV	1	417.780554	13.779084	220.	99.492000
1	6R	4	RV	1	446.278066	19.297184	215.	99.492005
1	1R	4	RV	1	408.923719	16.021869	221.	99.992000
1	3L	4	RV	1	431.680199	21.835205	219.	99.992002

Table 4 shows the information for the first four jumpers to exit during this simulation. The first field shows that this is the first repetition while the second shows airplane number two. The first plane does not drop jumpers since we know they won't encounter a vortex if there are no planes preceding it. If the program is used differently to test the distance between the equipment and personnel portions of the airdrop, then this needs to be changed. The next fields show the simulation time at jumper exit from the aircraft; jumper number; jumper weight; *x*; *y*; and, *z* coordinates. The R and L indicate the side of the airplane the jumper exited from. The different *y* coordinates show that the jumpers are 18.5 feet apart when they exit the aircraft.

Table 5 shows the information output for an encounter. Recalling that an encounter occurs when a jumper falls with the radius of a vortex, where that radius is defined as the distance from the center to where the swirl velocity is 20 ft/sec. The first fields in this output are identification of the jumper and vortex; it is from repetition number one and shows an encounter of 3 jumpers from airplane 4 with the right vortex from airplane 1. The multiple entries for jumper 1R, show this object first entered the right vortex of plane number 1 at an altitude of 435 feet and a distance from the center of 19 feet. At the next ½ second update, jumper number 1R is still in the vortex but is at a lower altitude and closer to the center. This continues at each half second interval -- as long as the jumper is within the radius of a vortex, it will generate a line of output.

The step value shows how far behind the generating aircraft the jumper encounters the vortex. Since the step size is 100 feet per step, a step of 218 on the first line indicates that jumper 1R hit the right vortex from plane number 1 at a distance of 21,800 feet. Note that simply counting the lines of output in the encounter file won't give an accurate count of the actual encounters; e.g., the sample shows six lines of output but only three encounters; 1R, 6R, and 3L.

Verification And Validation

Verification and validation should be accomplished in every phase of model development. The Wright Labs model used as a basis for the vortex representation was also used as a benchmark for validation of the vortex. A test run for a single ship and three ship formation provided the same vortex arrays as the Wright Labs model. With no movement of the airplane except along the axis of the drop zone, the vortex of this model matched the vortex calculated by the Wright Labs model. There are also numerous points within the code where key variables were checked for proper operation of the code. The formation position of each aircraft in both the ground and air coordinate systems were checked with a three ship formation at each time step. With a formation velocity of 135 knots (228 ft/sec), the aircraft moves about 114 feet every step, and maintains proper position. The correct number of jumpers were created and exited the aircraft when the ground coordinate of that aircraft was first greater than zero, and all jumpers had the same position upon creation as the airplane at the same time step.

One area that created a problem was the second jumper from the second airplane was inheriting the same position as the third jumper. This occurred because the jumpers were added to the pending list before the aircraft position update. Although both events occur at the same simulation time they need to be accomplished in sequential order. We solved this problem by making the exit time between jumpers 0.500001 seconds instead of .5 seconds, thus as long as the limit remains 102 jumpers per C-17, this extra time should have a negligent impact on simulation performance.

Two major areas of validation completed on the model were based on flight test data from Edwards AFB ("Summary ...", 1996) and Fort Bragg ("Flight ...", 1996) supplied by the C-17 SPO. The Edwards flight test was conducted with two airplanes flying directly in trail with 15,000 feet spacing while the Bragg test involved six aircraft in formation using the approved element geometry.

The Edwards flight test involving two aircraft directly in trail was flown to establish an upper bound on the risk of encounter. All flights were directed into the wind to minimize any crosswind affects. With 15,000 feet spacing between the two aircraft and the second following as close to directly in trail as possible, 12 mannequins were released from plane number two and scored visually to count the number of visible vortex

Table 6. Screening Test for Encounter Rate of Edwards Flight Test Data.

Lateral Box	Standard Deviation	Rate	Lateral Box	Standard Deviation	Rate
50	.5	41.7%	30	.5	0%
35	.5	0%	40	.5	33.3%
40	.1	25%			

encounters. This distance was chosen because it was the distance where the predicted vortex strength of the C-17 equals the vortex strength of the C-141 at 6,000 feet. The C-141 also flew this same test at 6,000 feet separation to establish an upper bound on its encounter rate.

We use the data collected from the Edwards test as a benchmark of our model performance. Flight testing showed an encounter rate of 16.25%±8.71 (95% Confidence Level) for the Edwards test. To calibrate our model we tried to get an encounter rate close to the actual by varying the standard deviation of the wind distribution and the formation tolerance position. Starting with one repetition screening experiments to see the effects of changing the tolerance box, the wind was varied to obtain an encounter rate similar to the test data (Table 6).

A standard deviation of .1 knot with a tolerance box of ±50 feet in-trail and ±40 feet laterally gives the desired results. These parameters were then entered into the program for fifty repetitions (Table 7) using random number seed 3 to see if the same percentage would result. The difference in aircraft altitude is due to the unavailability of the actual report until after simulation data was collected. A large variance in the encounter rate occurs, with some repetitions giving an encounter rate of 58% while others produce 0%. Results were promising because the model provided an estimated encounter rate of 13.5±5% (95%), which includes the rate for flight test results. A statistical test for

Table 7. Edwards Validation Test Input. (Jun 96)

	Actual	Simulation
Type of formation	Custom	2 (Custom)
Number of Planes	2	2
Planes Per Element	1	1
Jumpers Per Side	6	6
Element Leader Spacing	15,000	15,000
In-Trail Tolerance	?	50
Lateral Tolerance	?	40
Aircraft Weight	?	385000
Drop Altitude	1100	900
Airspeed	135	135
Crosswind at Altitude	0	0
Standard Deviation (1)	?	.1
Shear Altitude (1)	?	0
Crosswind (Middle)	?	0
Standard Deviation (2)	?	0
Shear Altitude (2)	?	0
Crosswind (Ground)	?	0
Standard Deviation (3)	?	0
Headwind	?	0
Air Density	0.00197	0.002000
Jumper Output File		EdwardJ
Encounter Output File		EdwardE
Scatter Output File		EdwardS
Random Seed		3
Individual Trajectories	0	0

equal means showed that there was no statistical difference between the two rates to the 95% confidence level.

The next area of validation was to simulate a series of flights from the flight testing at Fort Bragg. Flight tests on 16 August comprised of a 6 ship test with 12 jumpers each from airplanes 4, 5, and 6 were selected for this purpose. This test used the standard element spacing of 3,000 feet in-trail and 600 feet right for plane number two,

Table 8. Actual and Input Data for Bragg Test (16Aug 96)

Input Data	Actual (Average of two passes)	Ten Repetitions	50 Repetitions (X2)
Type of formation	2 (Custom)	2 (Custom)	2 (Custom)
Number of Planes	6	6	6
Planes Per Element	3	3	3
Jumpers Per Side	6 (2nd Element Only)	6	6
#2 In-Trail Spacing	3000	3000	3000
#2 Lateral Spacing	600	600	600
#3 In- Trail Spacing	6000	6000	6000
#3 Lateral Spacing	-900	-900	-900
Element Leader Spacing	27,000	27,000	27,000
In-Trail Tolerance	500	500	500
Lateral Tolerance	200	200	200
Aircraft Weight	382,396	383000	383000
Drop Altitude	900	900	900
Airspeed	134.33	134	134
Crosswind at Altitude	-3.1068	-3.1068	-3.1068
Standard Deviation (1)	?	.5	.5
Shear Altitude (1)	?	600	600
Crosswind (Middle)	?	-2.505	-2.505
Standard Deviation (2)	?	.5	.5
Shear Altitude (2)	?	300	300
Crosswind (Ground)	-2.369	-2.369	-2.369
Standard Deviation (3)	?	.5	.5
Headwind	3.4835	3.4835	3.4835
Air Density	0.002309	0.002309	0.002309
Jumper Output File		AUG16J	AUG16J#
Encounter Output File		AUG16E	AUG16E#
Scatter Output File		AUG16S	AUG16S#
Random Seeds		3	5, 9
Individual Trajectories		0	0

and 6,000 feet in-trail and 900 feet left for plane number three. Spacing between element leaders was 27,000 feet. The tolerance box was ±500 feet in-trail and ±200 feet laterally. Since two passes were conducted on this day, the average of the data for the two passes is taken as the input for the program (Table 8). Using 10 runs initially yields results that while close to the actual rate gives an interval judged too wide. In order to reduce this interval two additional runs were run using different random number seeds for 50 repetitions each, thus yielding a total sample size of 110 runs. When comparing the results of the simulation to the flight test they initially looked very good. Statistical tests for equal population means show that when the data is grouped by three airplanes, airplane 4 and 6, airplane 4 alone, and airplane five alone that the means are all statictically equal. There is a problem when airplane 6 is taken alone. Since the simulation showed no encounters for jumpers from airplane six its mean and variance is zero. When comparing it to the three encounters experienced by flight testing it fails the equal means test. I am encouraged by the fact that the rate for airplane 4 was statistically equal. Due to the large variance I think that to be absolutely sure of the accuracy of our simulation response we need a very large sample. This problem does not mean that the model is invalid but I do feel that it warrants further validation. One possible problem is the interpretation of wind within the model. The wind shift could have occured higher or lower than the 600 and 300 feet that I input. This data is not readily available to test where the wind shifts really occurred and what effects they will have on encounter rate.

VI. Results

ANALYSIS

My analysis focuses on varying the spacing between element leaders from 9,000 feet in trail to 40,000 feet, and measuring the encounter rate as the response. With the current element geometry, 9000 feet is the closest the second element should ever be to the preceding element (3000 ft between aircraft). It would be preferable for the experimental design to start at 8000 feet, but operational limits preclude using that distance. The current flight testing uses a spacing of 40,000 feet as the element spacing. With the lidar data showing the longest recorded vortex having a life of 3 minutes, this distance should result in zero encounters. (Interestingly, this spacing translates roughly into the 6 mile separation prescribed by the FAA for small aircraft trailing large aircraft.) Five design points and three random seeds give the necessary data to derive a formula for predicting an encounter rate (Table 9). This was later expanded to include three runs at 34,000 feet while eliminating the data collected at 40,000 feet. It was felt that the runs at 40,000 feet established an artificial zero because it was not possible for any jumpers to encounter a vortex at this distance, while at 34,000 feet it is still theoretically possible to hit a vortex.

Table 9. Distances and Random Seeds used for analysis

ELEMENT SPACING	9,000	16,000	24,000	32,000	34,000	40,000
Random Seeds (All Distances)	3	6	7			

Each design point and random seed combination consists of 50 repetitions, with 6 aircraft having 12 jumpers per aircraft. Blocking on the random seed provides three point estimates at each design point to derive a regression line predicting an encounter rate. Since preliminary results indicated a large variation on the encounter rate, it was necessary to use a large number of repetitions to get an accurate estimate of the response.

Using the current flight-tested element geometry as a basis, with three aircraft per element and 3000 feet in-trail spacing between aircraft, the second aircraft is 3,000 feet behind and 600 feet to the right of the lead, while the third is 6,000 feet behind and 900 feet to the left. This positioning is used for wind conditions up to 3 degrees of drift.

Table 10. Input Data for Simulation Runs with 50 Replications at Each Point.

Input Data	9000 Ft	16000 Ft	24000 Ft	32000 Ft	34000 Ft	40000 Ft
Type of formation	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)
Number of Planes	6	6	6	6	6	6
Planes Per Element	3	3	3	3	3	3
Jumpers Per Side	6	6	6	6	6	6
#2 In-Trail Spacing	3000	3000	3000	3000	3000	3000
#2 Lateral Spacing	600	600	600	600	600	600
#3 In- Trail Spacing	6000	6000	6000	6000	6000	6000
#3 Lateral Spacing	-900	-900	-900	-900	-900	-900
Element Leader Spacing	9000	16000	24000	32000	32000	40000
In-Trail Tolerance	500	500	500	500	500	500
Lateral Tolerance	200	200	200	200	200	200
Aircraft Weight	385000	385000	385000	385000	385000	385000
Drop Altitude	900	900	900	900	900	900
Airspeed	135	135	135	135	135	135
Crosswind at Altitude	0	0	0	0	0	0
Standard Deviation (1)	.1	.1	.1	.1	.0	.1
Shear Altitude (1)	0	0	0	0	0	0
Crosswind (Middle)	0	0	0	0	0	0
Standard Deviation (2)	0	0	0	0	0	0
Shear Altitude (2)	0	0	0	0	0	0
Crosswind (Ground)	0	0	0	0	0	0
Standard Deviation (3)	0	0	0	0	0	0
Headwind	0	0	0	0	0	0
Air Density	0.002309	0.002309	0.002309	0.002309	0.002309	0.002309
Jumper Output File (# = 3 ,6, 7)	Seed#9J	Seed#16J	Seed#24J	Seed#32J	Seed#34J	Seed#40J
Encounter Output File (#=3, 6, 7)	Seed#9E	Seed#16E	Seed#24E	Seed#32E	Seed#34E	Seed#40E
Scatter Output File (# = 3, 6, 7)	Seed#9S	Seed#16S	Seed#24S	Seed#32S	Seed#34S	Seed#40S
Random Seeds	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7
Individual Trajectories	0	0	0	0	0	0

With more than 3 degrees of drift the three airplanes fly with the same in-trail spacing, but aircraft two and three both fly upwind from the lead aircraft. This places number two 600 feet upwind, while number three flies 1500 feet upwind. This geometry was not tested in the simulation.

The simulation analysis uses a light and variable wind condition with the mean crosswind being zero, and the wind following a normal distribution with a standard deviation of 0.1 knot. Jumpers from aircraft two and three are analyzed separately from the jumpers from aircraft four, five, and six. Since no encounters between jumpers and vortices from the same element are acceptable, there should be no encounters from aircraft within the same element. I also expect the encounter rate to decrease as spacing increases, and am interested in the order of that decrease. Table 10 shows the input data used for the simulations to obtain the 15 point estimates (Table 11). Subsequent review

Table 11. Results of Simulation Runs, Blocked by Seed.

Seed Number	Distance	Coded Spacing	Mean	95 % Confidence Interval
3	9,000	-.9375	40.67 %	6.93 %
6	9,000	-.9375	42.94 %	6.97 %
7	9,000	-.9375	36.72 %	6.86 %
3	16,000	-.5	18.33%	4.53 %
6	16,000	-.5	19.56 %	4.63 %
7	16,000	-.5	18.44 %	4.93 %
3	24,000	0	13.72 %	3.33 %
6	24,000	0	10.89 %	3.00 %
7	24,000	0	10.83 %	2.98 %
3	32,000	.5	12.06 %	2.57 %
6	32,000	.5	10.56 %	2.68 %
7	32,000	.5	8.94 %	2.18 %
3	34,000	.625	0.33%	0.31%
6	34,000	.625	0.83%	0.49%
7	34,000	.625	0.00%	0.00%
3	40,000	1	0.00 %	0.00 %
6	40,000	1	0.00 %	0.00 %
7	40,000	1	0.00 %	0.00 %

of the analysis, suggests that the data collected at 40,000 feet artificially constrains the encounter rate to 0% and it should not be used in the regression calculations. The 34,000 feet spacing was added at this time and the regression reaccomplished with the 15 points including 34,000 feet and excluding 40,000 feet.

Since the encounter file has a one line entry for every ½ second that a jumper is in the vortex, much of the data is redundant for calculating the encounter rate but necessary for further analysis. I import each file into Microsoft Access and used the duplicate query wizard to reduce the data to number of encounters per airplane. (This analysis tool helps me identify which jumpers encountered a vortex and for how many time steps the encounter occurred). I then link that data into a Microsoft Excel spreadsheet and add the appropriate number of planes having zero encounters before finding the point estimates

Table 12. First Order Linear Regression Results, Coded Spacing vs. Encounter Rate.

Linear Fit

Encounter Rate = 15.0226 - 20.7943 Coded Spacing

Summary of Fit

RSquare	0.84997
RSquare Adj	0.838429
Root Mean Square Error	5.551883
Mean of Response	16.32222
Observations (or Sum Wgts)	15

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	1	2270.1180	2270.12	73.6492	<.0001
Error	13	400.7043	30.82		
C Total	14	2670.8223			

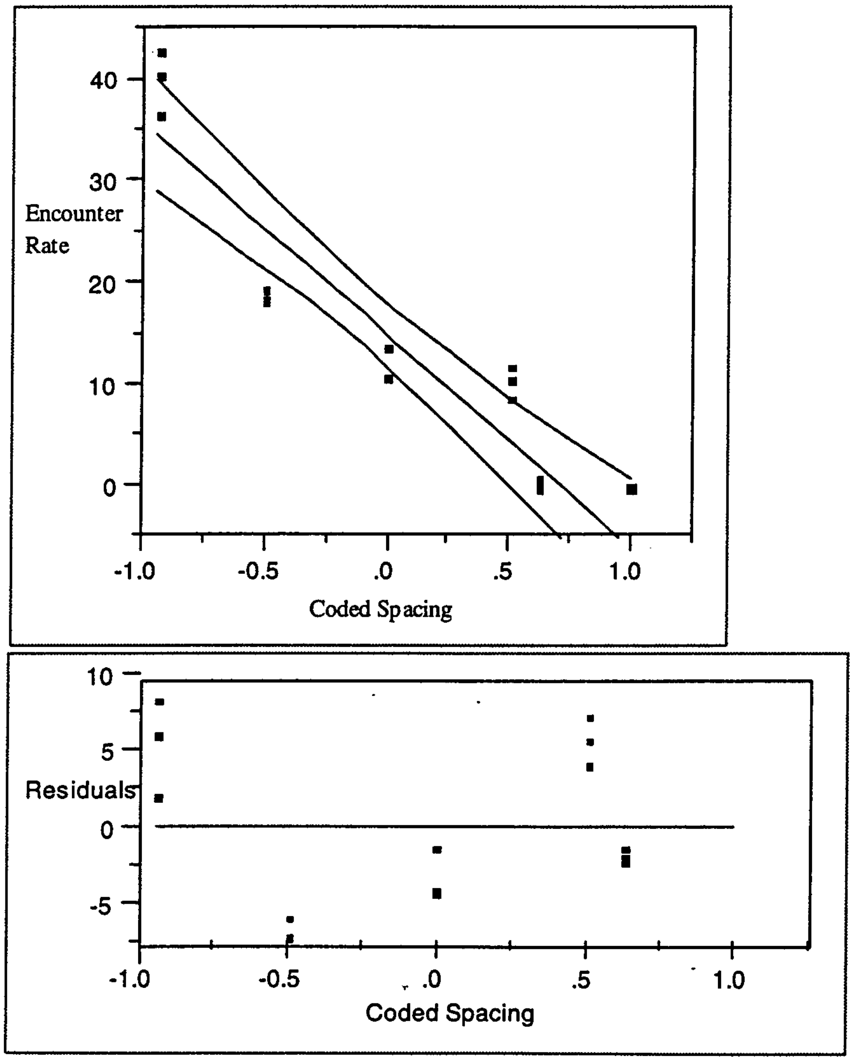
Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	15.022577	1.441467	10.42	<.0001
Coded Spacing	-20.79431	2.423041	-8.58	<.0001

for encounters rates (Table 11). The range of encounters for any airplane varied from 100% to 0% at both 9,000 and 16,000 feet spacing but dropped off at 32,000 feet to a maximum of 50% and ranging to 0%.

These 15 points (including 34,000, excluding 40,000 feet) provide the data set necessary to fit a line describing the encounter rate as a function of element spacing. The first try at regression was a simple first order least squares linear regression. I use the

Figure 6 . First Order Linear Regression with Confidence Interval and Residuals



JMP statistical software package and Microsoft Excel to assist the calculation of the regression parameters (Table 12). Since the actual numbers for the spacing are so large I coded the variables between -1 and 1 for 8,000 feet and 40,000 feet respectively. (I chose 8,000 feet as the lower end to have equally spaced intervals.)

Figure 6 shows the plot of the line with confidence interval and the residuals. Although the R Squared and Adjusted R Squared are fairly high (Table 12), the residuals indicate a clear non-linearity in the data. This indicates that a higher order function is necessary to better define the relationship between element spacing and encounter rate.

Next I use a second-order model to find a curve that would best fit the data points (Table 13 and Figure 7). R-Square and Adjusted R-Square both improve, but the

Table 13. Second Order Linear Regression Results, Coded Spacing vs. Encounter Rate.

Polynomial Fit degree=2

Encounter Rate = 10.3955 - 17.1144 Coded Spacing + 13.7241 Coded Spacing^2

Summary of Fit

RSquare	0.913015
RSquare Adj	0.898518
Root Mean Square Error	4.400014
Mean of Response	16.32222
Observations (or Sum Wgts)	15

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	2	2438.5008	1219.25	62.9774	<.0001
Error	12	232.3214	19.36		
C Total	14	2670.8223			

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	10.395537	1.940792	5.36	0.0002
Coded Spacing	-17.11439	2.290117	-7.47	<.0001
Coded Spacing^2	13.724073	4.653589	2.95	0.0122

residuals still suggest a higher order relationship between Coded spacing and Encounter rate.

Thus, the next model is a full, third-order model (Table 14 and Figures 8 and 9). Once again R-Square and Adjusted R-Square improve and the curve seems to fit the data points almost exactly. This time, however, the residuals appear to be randomly dispersed; the only noticeable problem being the significance of the first and second order terms. The

Figure 7 . Second Order Regression with Confidence Interval and Residuals.

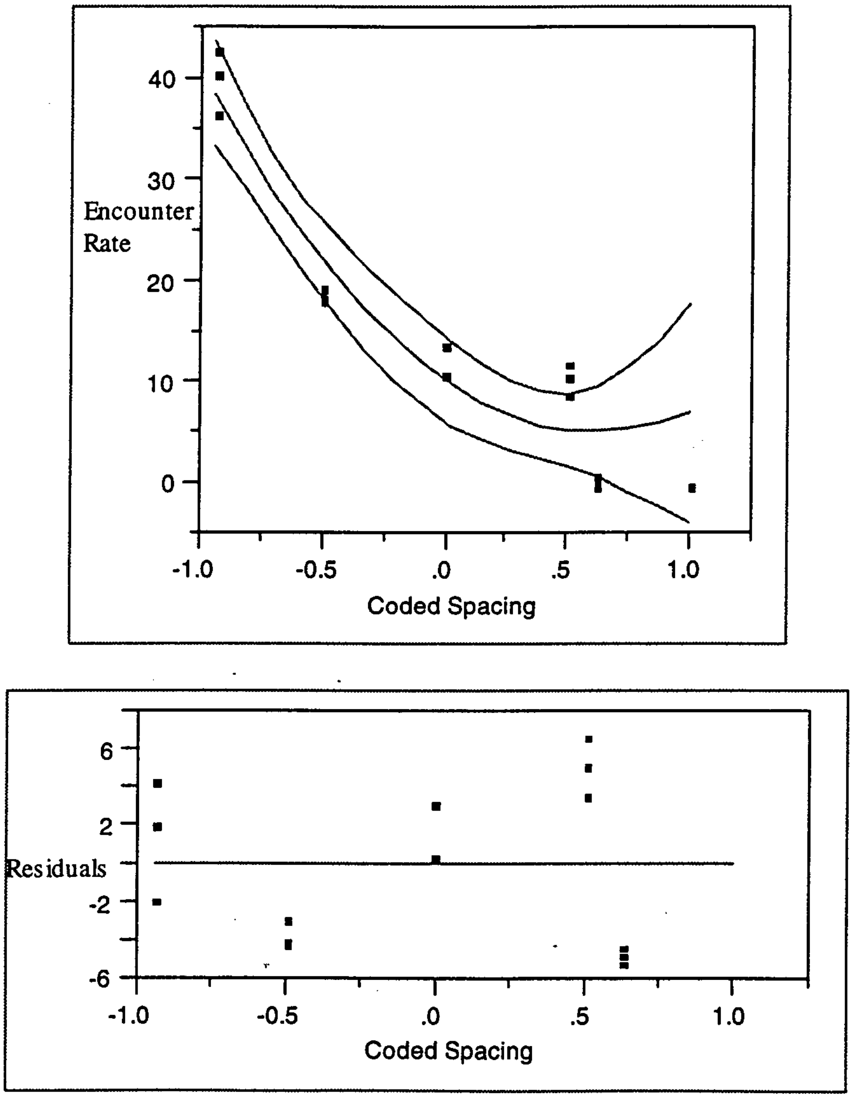


Table 14. Third Order Linear Regression Results, Coded Spacing vs. Encounter Rate.

Polynomial Fit degree=3

Encounter Rate = 13.6927 - 1.23977 Coded Spacing - 4.53751 Coded Spacing^2 - 35.854 Coded Spacing^3

Summary of Fit

RSquare	0.967746
RSquare Adj	0.95895
Root Mean Square Error	2.798434
Mean of Response	16.32222
Observations (or Sum Wgts)	15

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	3	2584.6787	861.560	110.0158	<.0001
Error	11	86.1436	7.831		
C Total	14	2670.8223			

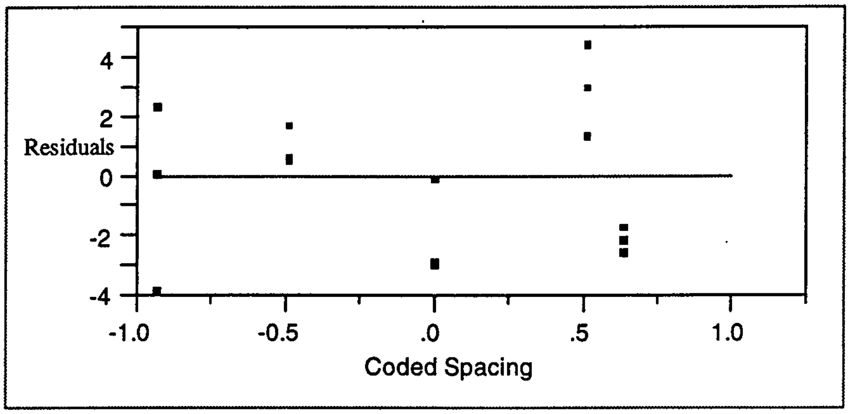
Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	13.692721	1.451224	9.44	<.0001
Coded Spacing	-1.239774	3.952485	-0.31	0.7596
Coded Spacing^2	-4.537511	5.160021	-0.88	0.3980
Coded Spacing^3	-35.85395	8.298726	-4.32	0.0012

regression model should be accomplished again with a reduced model excluding the first and second order terms (Coded Spacing and Coded Spacing^2).

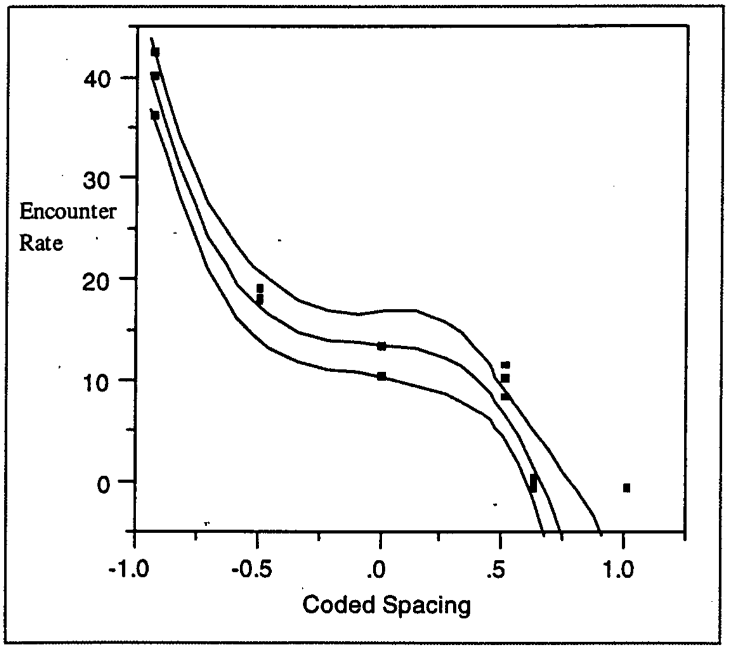
The results of the reduced third order model initially look very promising (Table 15 and Figure 10). Dropping the insignificant first and second order terms has very little effect on the total model, and explains over 96% of the variation in the data. Comparing the four regression models, the reduced third order model is the best. But, its curve suffers from lack of fit and a discernible pattern in the residuals, with all of the factors significant. Further experimentation is necessary to determine the true nature of the curve. I suggest this be done in conjunction with an experiment including the effects of crosswinds as well.

Figure 9. Full Third Order Residuals



With any regression model, prediction outside the range of the data can be dangerous - this caveat is particularly true in this instance. Obviously, any prediction outside the range of 0% to 100% is invalid since our response is the encounter rate. Since I used 9,000 and 40,000 feet (later modified to 34,000 feet) when setting up the experiment, prediction outside of this range should not be necessary. Only a fundamental

Figure 8. Full Third Order Regression Line with Confidence Interval.



change in element spacing, or new data on the life span of a vortex, will change the area of interest. Therefore, the reduced third order model should be sufficient in representing the risk function with the given input parameters but only within the 9,000 to 40,000 feet spacing.

Table 15. Reduced Third Order Regression Results, Coded Spacing vs. Encounter Rate.

Response: Encounter Rate

Summary of Fit

RSquare	0.961561
RSquare Adj	0.958605
Root Mean Square Error	2.810184
Mean of Response	16.32222
Observations (or Sum Wgts)	15

Lack of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Lack of Fit	3	71.27592	23.7586	7.5696	0.0062
Pure Error	10	31.38685	3.1387		
Total Error	13	102.66278			

Max RSq 0.9882

Parameter Estimates

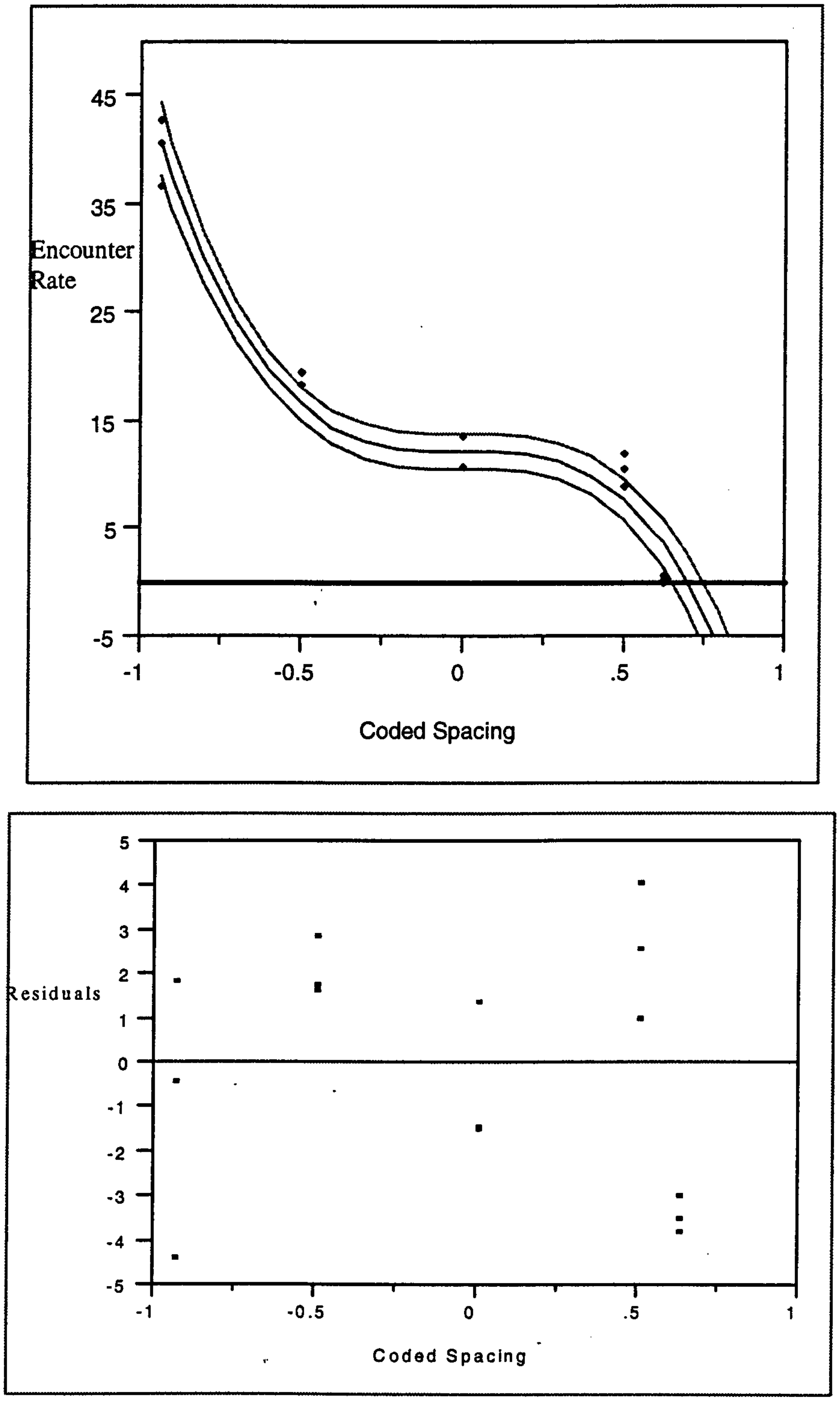
Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	12.274168	0.759516	16.16	<.0001
Coded Sp^3	-34.907	1.935693	-18.03	<.0001

Since the response is a rate defined by a binary variable (either the jumper encounters the vortex or not) logistic regression could be a better choice for predicting the response. Logistic regression limits the response between 0 and 1 using the function shown in equation 3. Using JMP to assist in the regression calculations yields the results in Table 16 and Figure 11.

Equation 3 . General Logistic Response Function.

$$E[Y] = [1 + \exp(-\beta_0 - \beta_1 X)]^{-1}$$

Figure 10. Reduced Third Order Curve with Confidence Interval and Residuals.



While the Chi Square test statistic looks acceptable for the model as a whole, there does seem to be a lack of fit. To accept the null hypothesis that the model fits the data would require a smaller Chi Square value to raise the probability above the .05 level. Adding higher order terms (up to fourth-order) into the model to get a better fit did not produce a better fit.

Table 16. Logistic Regression Results, Encounter Rate vs. Coded Spacing

Response: Response

Iteration History

Iter	LogLikelihood	Step	Delta-Criterion	Obj-Criterion
1	-12014.31529	Initial	1.52249241	1.4963e304
2	-10606.75979	Newton	0.28578082	0.1327035
3	-10475.98057	Newton	0.0370223	0.01248371
4	-10474.53191	Newton	0.00077349	0.0001383
5	-10474.53126	Newton	0.00000037	0.00000006

Converged by Gradient

Whole-Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	1539.784	1	3079.568	0.0000
Full	10474.531			
Reduced	12014.315			

RSquare (U)	0.1282
Observations (or Sum Wgts)	27000

Lack of Fit

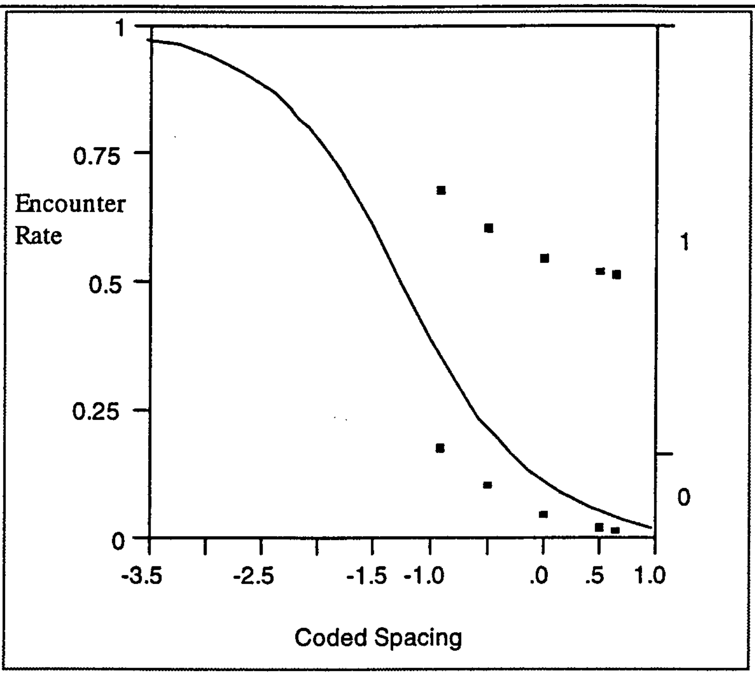
Source	DF	-LogLikelihood	ChiSquare	Prob>ChiSq
Lack of Fit	3	314.673	629.3451	<.0001
Pure Error	26995	10159.859		
Total Error	26998	10474.531		

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept	-2.0531783	0.0224769	8344.1	0.0000
Coded Spacing	-1.6609887	0.0330304	2528.8	0.0000

Even though the logistic regression model is not problem free, it may provide some useful information. Since there are only five design points in the model, the degrees of freedom for the Chi Square test for lack of fit require that the test statistic be

Figure 11. Logistic Regression Prediction Curve.



very small. To test to the 95% level with three degrees of freedom requires the test statistic to be below 7.81. Running the model at more points would increase the degrees of freedom, thus raising the critical value. This may allow future analysis to fit a better curve using logistic regression. Next, I compare the reduced third order and logistic regression models for their use as prediction models.

The main uses of the models are to determine the point where the encounter rate begins to rise sharply and to quantify the risk to the jumpers at a given element spacing. Either of the two models can be used for this purpose. By visual inspection the reduced third order model rises sharply from almost 0% at 34,000 to 10% at 28,000. It then stays fairly flat, rising slowly until 20,000 feet, then rising sharply to a high of 40% at 9,000 feet.

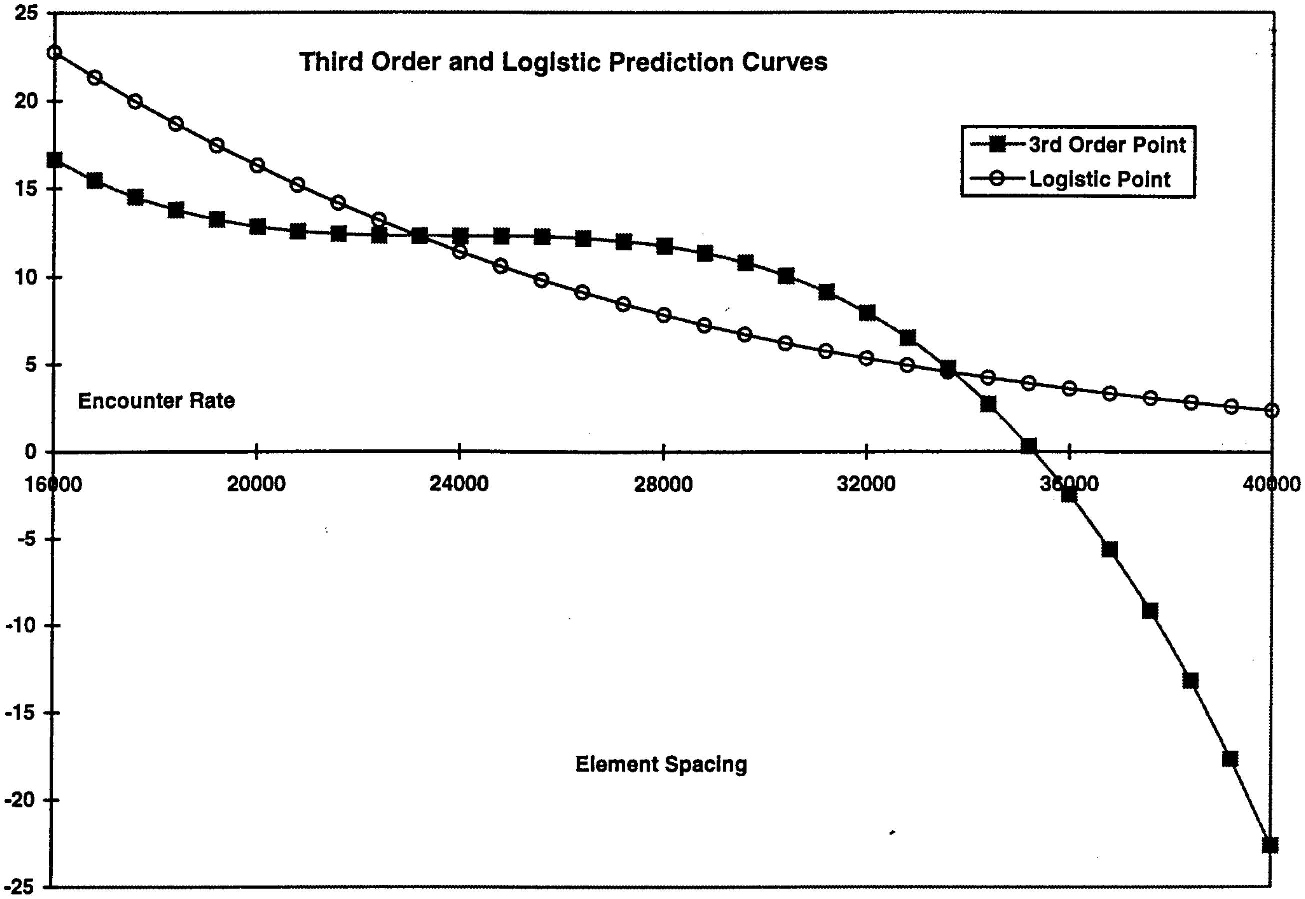
The logistic regression curve doesn't have the flat spot in the center like the third order model. It rises slowly from 2% at 40,000 feet to 10% at 24,000 feet, then increases up to 40% and 9,000 feet. Table 17 shows a comparison of predicted responses using the two models at various element intervals, while Figure 12 is a plot of Table 17. Note the larger interval on the third order encounter rate compared to the logistic prediction. The logistic regression curves actually cross and all three are equal at 24,000 feet.

Another interesting find was the discovery and correction of a bug during data collection at 32,000 feet element spacing. At this spacing some of the jumpers were encountering the vortex at the very end of its simulated life. It would be in the vortex at one time step and the next it would be within 100 feet of the end. Belano (1997) describes the search algorithm in more detail, but the way it was implemented compared the jumper *x* position with the vortex *x* position, and when they were within 100 feet the *y* and *z* position were checked. If all three axes were within 100 feet then the method asked for the next vortex point and drew a line between the two points to take the normal distance. If that distance was less then the largest value of the radius at the two points then an encounter occurred. When the jumper was within 100 feet of the end of the vortex, the program crashed when the array asked for a value of the vortex that was out of bounds. This problem was easily corrected, but did shed some insight into the distance at which the jumpers were encountering vortices. Since this was a no wind scenario, the distance between airplanes was essentially the element spacing. At 32,000 feet, encounters were occurring at the end of the vortex life span. Lidar data showed some vortices dissipated sooner than others, which is not modeled due to the data collection occurring simultaneously with model creation. If this data were incorporated in the next version of the model, I feel the encounter rate between 24,000 to 32,000 feet would decrease in the simulation. The model predicts the worse case scenario in this spacing range, since every vortex lasts as long as the longest recorded vortex. Analysis of the lidar data for some distribution of vortex life, and implementation of that distribution within the model, should improve the predictive capabilities in the 24,000 to 40,000 feet range. This may make the logistic regression curve fit better and solve some of the lack of fit problems.

Table 17. Predicted Encounter Rate for Third Order and Logistic Models

		Third Order Encounter Rate			Logistic Encounter Rate		
Spacing	Coded Spacing	Point	Lower	Upper	Point	Lower	Upper
16000	-0.5	16.64	15.52	17.76	22.75	23.41	22.27
16800	-0.45	15.46	14.20	16.71	21.32	21.89	20.91
17600	-0.4	14.51	13.14	15.88	19.96	20.44	19.61
18400	-0.35	13.77	12.31	15.23	18.67	19.07	18.38
19200	-0.3	13.22	11.69	14.74	17.44	17.76	17.20
20000	-0.25	12.82	11.24	14.40	16.27	16.53	16.09
20800	-0.2	12.55	10.95	14.16	15.17	15.37	15.03
21600	-0.15	12.39	10.77	14.02	14.14	14.27	14.04
22400	-0.1	12.31	10.67	13.95	13.16	13.24	13.10
23200	-0.05	12.28	10.64	13.92	12.24	12.28	12.21
24000	0	12.27	10.63	13.92	11.37	11.37	11.37
24800	0.05	12.27	10.63	13.91	10.56	10.53	10.59
25600	0.1	12.24	10.59	13.88	9.80	9.74	9.85
26400	0.15	12.16	10.50	13.81	9.09	9.00	9.16
27200	0.2	11.99	10.32	13.67	8.43	8.32	8.51
28000	0.25	11.73	10.02	13.43	7.81	7.68	7.91
28800	0.3	11.33	9.58	13.09	7.23	7.08	7.34
29600	0.35	10.78	8.96	12.60	6.69	6.53	6.82
30400	0.4	10.04	8.13	11.95	6.19	6.02	6.32
31200	0.45	9.09	7.07	11.12	5.73	5.55	5.86
32000	0.5	7.91	5.75	10.07	5.30	5.11	5.44
32800	0.55	6.47	4.13	8.80	4.90	4.71	5.04
33600	0.6	4.73	2.19	7.28	4.52	4.33	4.67
34400	0.65	2.69	-.10	5.48	4.18	3.99	4.32
35200	0.7	0.30	-2.77	3.38	3.86	3.67	4.00
36000	0.75	-2.45	-5.86	0.95	3.56	3.37	3.70
36800	0.8	-5.60	-9.38	-1.82	3.29	3.10	3.43
37600	0.85	-9.16	-13.37	-4.95	3.03	2.85	3.17
38400	0.9	-13.17	-17.86	-8.48	2.80	2.62	2.93
39200	0.95	-17.65	-22.88	-12.43	2.58	2.41	2.71
40000	1	-22.63	-28.46	-16.81	2.38	2.21	2.51

Figure 12 . Third Order and Logistic Prediction Curves.



V. Conclusions

Summary of Results

Of the regression models analyzed, the reduced third order seems the best model even with the problems noted (Table 18). However this model suffers from the fact that the data is better suited to a logistic regression since the response is of the form of a rate between 0% and 100%. Logistic regression limits the response function between these values. However the fit with the logistic regression does not seem to be as good as with least squares regression. Comparing both methods shows that the logistic model may under estimate encounter rate at some intervals and over estimate in others. We have no empirical data for validation of which regression model is the most correct. Both models have strengths and weaknesses associated with them. The third order model has a large prediction interval associated with any prediction of the encounter rate at any spacing while the logistic model tends to both over and under estimate the rate in certain spacing intervals.

Table 18. Comparison of Regression Models.

Model	Effects	R Square	Adjusted R Square	Reason for Rejection
First Order	Coded Spacing	.8500	.8384	Residuals
Second Order`	Coded Spacing Coded Spacing^2	.9130	.8985	Residuals
Full Third Order	Coded Spacing Coded Spacing^2 Coded Spacing^3	.9677	.9659	First Order Effect Second Order Effect Insignificant
Reduced Third Order	Coded Spacing^2 Coded Spacing^3	.9616	.9586	Lack of Fit
Logistic Regression	Coded Spacing	.1282		Lack of Fit

I recommend that both models be presented as possible representations of the encounter rate in this type of an experiment. Only continued collection of flight test data will give the information necessary to choose which model better represents reality. Although any model will have flaws, either of these models can be used. The important factor in choosing which model to use is how much risk are we willing to take with live jumpers. One possibility for better prediction would be another experiment collecting more data points at the same intervals with different random number seeds. This would increase the degrees of freedom and may drive down the lack of fit statistic into an acceptable range. Without further experimentation, I cannot confidently recommend either model; but, if I had to choose I would take the reduced third order model.

Model Improvements

While this model appears useful, it is by no means perfect; there are a few areas where the model can be improved upon with further research. Two major areas are the wind behavior and formation positioning within the tolerance box. It could also be easily expanded to include other aircraft and equipment drops. Further analysis of vortex characteristics could also improve our representation of a vortex, as well as a test of the assumption that aircraft spacing makes the influence by any vortex pair upon another insignificant compared to its own induced motion.

Since the decision to vary the winds was not implemented at the last minute, there was not a lot of time for research into the type of distribution that the winds follow. The model uses the normal distribution with a user input standard deviation. Coffin (1997) suggests some sources for further research into modeling changing winds, but also concludes that for the purposes of this model the normal distribution is sufficient. (Other possible distribution types include gamma, weibull, and Log-normal.) I feel this could be an area of model improvement if one could test the sensitivity of the encounter rate to the type of wind distribution used. I suspect the wind has a significant impact on vortex position, but do not know how a different representation of the winds will affect the overall model performance.

Another area of improvement is in the fly method and how the aircraft fly in relation to the lead ship. I know from personal experience that an airdrop run-in can be quite busy, and my experience is with a six man crew of the C-141 not the three man crew of the C-17. If there were some way to collect data on actual formation position of the aircraft, the fly method of always moving from in position to a random point out and back again could be improved upon. I realize that the method induces a lot of movement that is probably not evident during actual flight conditions. Once over the dropzone, baring unsafe flight conditions, the primary concern is maintaining a stable jump platform, not correcting formation position. After the jumpers are clear, the pilot would again worry about his position within the formation. This is all personal pilot technique, but with the proper data one could analyze the time spent in position, the magnitude of error before pilot recognition, the rate of correction after recognition, and the time spent out of position. Intuitively, jumper encounter rate seems to be highly sensitive to relative position of the vortex and airplane. This was also evident in the model. If the fly method induces more aircraft movement than is actually there, the response could have a built in bias.

Another area of research is the expansion of the model to include different airplanes and equipment drops. The *C-17Obj* is easily modifiable to take on the characteristics of another airplane. One could copy the definition and implementation of the *C-17Obj,* change the wing characteristics and the control object, and easily have C-141, C-130, and C-5 objects in the model. One definite change would be in the strength decay equations. The strength now stays constant for 60 seconds before starting to decay. The other airplanes have a different characteristic value before the strength begins to decay. If this value were made an attribute of the airplane, it could be passed into the vortex object upon update; thus the vortex objects would have the same code but generate different vortices for each aircraft type. If the rate at which they decay is different for each aircraft type, then the code could be modified with different decay formulae within the vortex implementation by using a case statement that chooses the correct formula within the module.

The creation of a heavy equipment object is also very simple compared to our initial effort. Belano's paratroop objects could be modified to include the characteristics of a heavy equipment pallet instead of a person, and have different parachute characteristics. This equipment object could be used to assess the risk to jumpers with different spacing between the heavy equipment and personnel segments of the strategic brigade airdrop. If the C-5 were used for the equipment portion of the drop, their vortices are stronger than the C-17's and could have an even greater impact on jumper safety.

The last area for further research I will address is that of vortex life. The model assumes that every vortex will last for as long as the longest recorded vortex in the lidar data. Obviously, this is not the case because most of the recorded vortices did not last that long. The lidar data was collected and analyzed as this project was on going, and was not completely available for input analysis. I feel that the data could be analyzed to find a distribution of life spans and somehow implemented within the code. By ignoring the fact that some vortices decay beyond recognition in a shorter distance than others, the model may be overestimating the encounter rate at the longer intervals. (This range is probably in the 24,000 to 40,000 feet range.)

We also found and corrected a bug in the program during simulation runs at 32,000 that led to an interesting discovery. Some jumpers were encountering vortices near the very end of the vortex life at this spacing. They would encounter the vortex during one time interval and then be within 100 feet of the end at the next. The bug was that our algorithm asked for the next point to draw a line between to take the normal distance for comparison to the radius. When the point was where the jumper was within 100 feet of the last point, it asked for a point that was not in the vortex array causing the program to crash. We fixed the problem, but this shed some insight into the distance at which the jumpers would encounter a vortex. If the change to the vortex representation of shortening its life or length based on empirical data were implemented I feel the predicted encounter rate at 32,000 feet would decrease dramatically.

Finally, I feel a test of the assumption that the vortex pairs are far enough apart that they do not have a significant impact on another pair when compared to the motion induced by the opposite vortex within each pair is required. Some sort of diagnostic should be added into the code to record the distance between any vortices throughout the simulation. It is possible within this simulation that the vortices are sufficiently close to one another to influence the movement of another vortex pair and we would not even know it. Since the influence of any vortex is inversely proportional to the distance between the vortices, a simple check of the actual distance between any vortex pairs should be sufficient. Visualization of the simulation could also aid in this function.

Contribution

The biggest contribution of this model is as a tool for engineering analysis. This model should be a significant addition to the tools currently being used to analyze airdrop operations. Even without the listed improvements, we now have a way to measure the effects of changes within a formation on jumper-vortex interaction. It provides a solid basis for follow-on research and may be the beginning of a useful combat model. If the results of this model could be incorporated into a higher level campaign model, then we would have some way of assessing the risks associated with a planned airborne assault, and could reduce the number of affected troops by some percentage in planning this assault.

Another contribution is the preliminary analysis provided by this model. Due to time and computer constraints the analysis doesn't provide us with a clear picture of the relationship between element spacing and encounter rate. Further experimentation should help in defining this relationship. (I suspect the final model will either be a third order or a logistic regression model.) Further testing should focus on expanding the current experiment to include the full range of crosswinds from -3 to 3 degrees drift; increase the number of element spacing points; and, the number of repetitions at each point. Since each data point of 50 repetitions took 9 hours to complete this will require a significant amount of computer time to complete. A simple design of experiments using the extremes of the two factors will not be sufficient because there are some critical angles between the elements where the crosswinds will dramatically increase the encounter rates. Blake's model (or simple geometry) can help find these angles, but the simulation should test around these critical points to see what effect approaching these angles has on the encounter rate. My suspicion is that the rate will be fairly flat throughout the surface until these critical angles are approached; then, the rate will increase sharply with the maximum being very close to what I discovered in this analysis.

Appendices

Appendix A

Petry's MODSIM Code

Implementation Airplane Module

```
IMPLEMENTATION MODULE AirplaneMod;

FROM MathMod          IMPORT pi;
FROM VortexMod        IMPORT RightVortexObj, LeftVortexObj;
FROM globalMod        IMPORT rho, CrossWind1, CrossWind2, CrossWind3, ShearAlt1, ShearAlt2,
                             weight, seed4, windseed1, windseed2, windseed3, StandDev1,
                             StandDev2, StandDev3, vs1, vs2, vs3;
FROM VortexControlMod IMPORT TotalVortexObj, VortexControl;
FROM inputMod         IMPORT stickSize;

OBJECT C17Obj;

   ASK METHOD ObjInit;

{Initalize the C-17 Object and update the winds}
{To create other types of airplanes the time before vortex decay starts needs to be moved into the airplane
objects}
{It is currently set to 60 seconds in to vortex object for the C-17}

      BEGIN

        beff :=  165.0;
        ar   :=  7.16 ;
        s    := 3800.0 ;
        bp   := pi*beff/4.0;
        gmod := 0.8;
        vs1  := CrossWind1 ;
                vs2  := CrossWind2 ;
                vs3  := CrossWind3 ;
        NEW (RightVortex)  ;
        NEW (LeftVortex)   ;

      END {ASK} METHOD {ObjInit};

{This method updates the vortex after the control object passes in the aircraft position using a steady wind}

   ASK METHOD VortexPosition (IN xs, ys, altitude, vfk, vf, xg, yg, weight : REAL    ;
                    IN NumberOfSteps                  : INTEGER);

      BEGIN
        cl      := 2.0*weight/(rho*vf*vf*s);
                vs1     := CrossWind1;
                vs2     := CrossWind2;
                vs3     := CrossWind3;
        ASK RightVortex TO Update(beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1, vs2,
vs3, NumberOfSteps);
        ASK LeftVortex TO Update(beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1, vs2,
                                 vs3, NumberOfSteps);
        END METHOD {VortexPosition};

    {This method updates the vortex position while the airplane flys with a variable wind}

      ASK METHOD FlyPosition(IN xs, ys, altitude, vfk, vf, xg, yg, weight : REAL    ;
                     IN NumberOfSteps                    : INTEGER);

        BEGIN
          cl := 2.0*weight/(rho*vf*vf*s);
                      IF StandDev1 > 0.0;

                        vs1     := windseed1.Normal (CrossWind1, StandDev1);
                      ELSE
                           vs1     := CrossWind1;
                      END IF;

                      IF StandDev2 > 0.0;

                        vs2     := windseed2.Normal (CrossWind2, StandDev2);
                      ELSE
                        vs2     := CrossWind2;
                      END IF;

                      IF StandDev3 > 0.0;

                        vs3     := windseed3.Normal (CrossWind3, StandDev3);
                      ELSE
                           vs3     := CrossWind3;
                      END IF;

          ASK RightVortex TO BackwardUpdate(beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod,
    vs1, vs2, vs3, NumberOfSteps);
          ASK LeftVortex TO BackwardUpdate(beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod,
    vs1, vs2, vs3, NumberOfSteps);
        END METHOD {VortexPosition};

      TELL METHOD greenLight (IN myPlane : INTEGER);

        VAR

          stick : INTEGER;

        BEGIN

          stick := 1;

          {SCHEDULE FIRST JUMPERS TO EXIT AT 0 SECONDS GREEN LIGHT}

          NEW  (rightJumper);
          ASK  rightJumper TO initialize (stick, myPlane);
          TELL rightJumper TO jump;

          NEW  (leftJumper);
      ASK  leftJumper TO initialize (stick, myPlane);
      TELL leftJumper TO jump;

      WHILE stick < stickSize
        stick := stick + 1;
        WAIT DURATION 0.500001;

          NEW  (rightJumper);
          ASK  rightJumper TO initialize (stick, myPlane);
          TELL rightJumper TO jump;

          NEW  (leftJumper);
          ASK  leftJumper TO initialize (stick, myPlane);
          TELL leftJumper TO jump;

        END WAIT {DURATION};

      END WHILE;

    END {TELL} METHOD {greenLight};

END OBJECT {C17Obj};

END {IMPLEMENTATION} MODULE {AirplaneMod}.
```

Definition Vortex Control Module

```
DEFINITION MODULE VortexControlMod;

FROM globalMod   IMPORT NumberofPlanes, PlanesPerElement, ElementGeometry,
                 FormationGeometry, ElementSpacing, NumberOfElements,
                 altitude, vfk, ElementGeometryType, weight;
FROM VortexMod   IMPORT RightVortexObj, LeftVortexObj;
FROM AirplaneMod IMPORT C17Obj;

TYPE

{Record for Vortex Control Object}

    VortexControlType = RECORD
        PlaneNumber  : INTEGER;       {Aircraft position in formation}
        xs           : REAL;          {Actual X Air Coordinate relative to lead airplane}
        ys           : REAL;          {Actual Y Air Coordinate relative to lead airplane}
        altitude     : REAL;          {Actual Z Air Coordinate (Altitude (AGL))}
        xsplan       : REAL;          {Planned X position relative to lead aircraft}
        ysplan       : REAL;          {Planned Y position relative to lead aircraft}
        altitudeplan : REAL;          {Planned Z position relative to lead aircraft (not
used)}
        xsnext       : REAL;          {Next X Position, from uniform draw if in position or
planned position if out of position}
        ysnext       : REAL;          {Next Y Position, from uniform draw if in position or
planned position if out of position}
        altitudenext : REAL;          {Next Altitude (not used)}
        xg           : REAL;          {Airplane's X Ground Coordinate}
        yg           : REAL;          {Airplane's Y Ground Coordinate}
        vf           : REAL;          {Velocity of aircraft (ft/sec)}
        vfk          : REAL;          {Velocity of aircraft (knots)}
        weight       : REAL;          {Weight of aircraft}
        C17          : C17Obj;     {C-17 Object and all it methods and records}
        NumberOfSteps : INTEGER;      {How many steps to calculate the vortex}
   fullLoad     : BOOLEAN;  {Initialized to true before the CARP then false}
    END RECORD;

    VortexControl  =  ARRAY INTEGER OF VortexControlType;

    TotalVortexObj  =  OBJECT;

        Information     : VortexControl;
        i,j             : INTEGER;
        ElementPosition : INTEGER;
        VortexLength    : INTEGER;

        ASK METHOD ObjInit;
        ASK METHOD PositionInformation;
        TELL METHOD Fly;
```

```
           TELL METHOD greenLight;

          END OBJECT {TotalVortexObject};

VAR

   Airdrop     : TotalVortexObj;
   trailshift  : INTEGER;
   lateralshift : INTEGER;

END {DEFINITION} MODULE {VortexControlMod}.
```

Implementation Vortex Control Module

```
IMPLEMENTATION MODULE VortexControlMod;

FROM globalMod IMPORT NumberofPlanes, PlanesPerElement, ElementGeometry,
          FormationGeometry, NumberOfElements, ElementSpacing,
          altitude, vfk, ElementGeometryType, weight, RunLength, streamE, trailseed, lateralseed,
HeadWind,
          lateralBox, trailBox;
FROM VortexMod IMPORT RightVortexObj, LeftVortexObj;
FROM AirplaneMod IMPORT C17Obj;
FROM SimMod IMPORT SimTime;
FROM UtilMod IMPORT DateTime;

OBJECT TotalVortexObj;

{Initialization of Vortex Control Array}

  ASK METHOD ObjInit;

    BEGIN

{Assume that first airplane is origin of air coordinate system with a positive altitude}

      NEW (Information, 1..NumberofPlanes);
      NEW (Information[1]);
      Information[1].PlaneNumber  := 1;
      Information[1].xs           := 0.0;
      Information[1].ys           := 0.0;
      Information[1].altitude     := altitude;
      Information[1].xsplan       := 0.0;
      Information[1].ysplan       := 0.0;
      Information[1].altitudeplan := altitude;
      Information[1].xsnext       := 0.0;
      Information[1].ysnext       := 0.0;
      Information[1].altitudenext := altitude;
      Information[1].xg           := -300.0;
      Information[1].yg           := 0.0;
      Information[1].vfk          := vfk;
      Information[1].vf           := vfk*1.69085;
      Information[1].weight       := weight;
      Information[1].fullLoad     := TRUE;
      NEW (Information[1].C17);

{Load formation position information from input files}

      FOR i  := 2 TO NumberofPlanes;

{Calculate if all elements are full, if not then add an element for remaining airplanes}
```

```
        ElementPosition   := i MOD PlanesPerElement;

        IF ElementPosition = 0;
           ElementPosition := PlanesPerElement;
        END IF;

        IF ElementPosition = 1;

           NEW (Information[i]);
           Information[i].PlaneNumber  := i;
           Information[i].altitude     := altitude;
           Information[i].altitudeplan := altitude;
           Information[i].altitudenext := altitude;
           Information[i].vfk          := vfk;
           Information[i].vf           := vfk*1.69085;
           Information[i].weight       := weight;
           Information[i].xsplan       := Information[i-PlanesPerElement].xs+ElementSpacing;
           Information[i].ysplan       := Information[i-PlanesPerElement].ys;
           Information[i].xs           := Information[i-PlanesPerElement].xs+ElementSpacing;
           Information[i].ys           := Information[i-PlanesPerElement].ys;
                                Information[i].xsnext                    :=     Information[i-
PlanesPerElement].xs+ElementSpacing;
           Information[i].ysnext       := Information[i-PlanesPerElement].ys;
           Information[i].xg           := Information[i-PlanesPerElement].xg-ElementSpacing;
           Information[i].yg           := Information[i-PlanesPerElement].yg;


           Information[i].fullLoad     := TRUE;
           NEW (Information[i].C17);

        ELSE

           NEW (Information[i]);
           Information[i].PlaneNumber  := i;
           Information[i].altitude     := altitude;
           Information[i].altitudeplan := altitude;
           Information[i].altitudenext := altitude;
           Information[i].vfk          := vfk;
           Information[i].vf           := vfk*1.69085;
           Information[i].weight       := weight;
           Information[i].xsplan                                          :=     Information[i-
ElementPosition+1].xs+ElementGeometry[ElementPosition].Intrail;
           Information[i].ysplan                                          :=     Information[i-
ElementPosition+1].ys+ElementGeometry[ElementPosition].CrossTrack;
           Information[i].xs                                               :=    Information[i-
ElementPosition+1].xs+ElementGeometry[ElementPosition].Intrail;
           Information[i].ys                                               :=    Information[i-
ElementPosition+1].ys+ElementGeometry[ElementPosition].CrossTrack;
           Information[i].xsnext                                          :=     Information[i-
ElementPosition+1].xs+ElementGeometry[ElementPosition].Intrail;
           Information[i].ysnext                                          :=     Information[i-
ElementPosition+1].ys+ElementGeometry[ElementPosition].CrossTrack;
```

```
                    Information[i].xg                    :=  Information[i-ElementPosition+1].xg-
ElementGeometry[ElementPosition].Intrail;
                    Information[i].yg                                        :=     Information[i-
ElementPosition+1].yg+ElementGeometry[ElementPosition].CrossTrack;
                    Information[i].fullLoad     := TRUE;
                    NEW (Information[i].C17);

                END IF;
            END FOR;

{Calculate Vortex position until 20,000 feet behind last airplane in formation}

            VortexLength  := TRUNC(Information[NumberofPlanes].xs)+20000;

{Calculate end condition for the flight}

            RunLength     := FLOAT(VortexLength)/Information[1].vf;


{Calculate the number of steps behind each airplane for vortex calculations}
{With 420 selected the vortex is terminated at 42,000 ft behind the generating aircraft.
This is from LIDAR data on the life of the vortex.  The commented statement will let the
vortex live until 20,000 ft behind the last aircraft in the formation.  This gives enough
time for all 102 jumpers to exit and descend to the ground}

            FOR i  := 1 TO NumberofPlanes;
                Information[i].NumberOfSteps    :=  420    {(VortexLength-TRUNC(Information[i].xs))  DIV
100};
                ASK SELF TO PositionInformation;
            END FOR;

        END METHOD {ObjInit};

    ASK METHOD PositionInformation;

        BEGIN

{Pass aircraft information into the C17 Object to pass onto the vortices}
{Position the aircraft in the tolerance box at some random point}

                IF i <> 1;
                    lateralshift  := lateralseed.UniformInt (-lateralBox, lateralBox);
                    trailshift    := trailseed.UniformInt (-trailBox, trailBox);

                    Information[i].xs := Information[i].xs + FLOAT(trailshift);
                    Information[i].xg := Information[i].xg + FLOAT(trailshift);

                    Information[i].ys := Information[i].ys + FLOAT(lateralshift);
                    Information[i].yg := Information[i].yg + FLOAT(lateralshift);

                END IF;

{Update the vortex position based on airplane position}
```

```
          ASK    Information[i].C17    TO    VortexPosition(Information[i].xs,    Information[i].ys,
Information[i].altitude,
                                      Information[i].vfk,       Information[i].vf,        Information[i].xg,
Information[i].yg,
                                      Information[i].weight, Information[i].NumberOfSteps);

      END METHOD {PostionInformation};

{Begin movement of the formation, continues every 1/2 second}

   TELL METHOD Fly;

      BEGIN
         WHILE SimTime < RunLength;
            WAIT DURATION 0.5;
{
OUTPUT("Flying : ", SimTime);

ASK streamE TO WriteString ("Flying: ", REALTOSTR (SimTime),"  ", DateTime);
}
{Update ground and air positions while flying}
{If the aircraft is away from its planned position then it flys back to that position}
{After it reaches position it then draws a new position to fly to.  When it gets out}
{of position again it flys back to the planned position}

                                      FOR i := 1 TO NumberofPlanes;
                  IF i = 1;
                                               Information[i].xg   :=   Information[i].vf*.5+Information[i].xg-
.5*HeadWind;
                                               ELSE
                                               IF Information[i].xs = Information[i].xsplan;
                                                        IF Information[i].ys = Information[i].ysplan;
                                                                 Information[i].ysnext                    :=
Information[i].ysplan+FLOAT(lateralseed.UniformInt (-lateralBox, lateralBox));
                                                                 Information[i].xsnext                    :=
Information[i].xsplan+FLOAT(trailseed.UniformInt (-trailBox, trailBox));
                                                        END IF;
                                               END IF;

                                               IF Information[i].xs = Information[i].xsnext;
                                                        IF Information[i].ys = Information[i].ysnext;
                                                                 Information[i].ysnext                    :=
Information[i].ysplan;

                                                                 Information[i].xsnext                    :=
Information[i].xsplan;

                                                        END IF;
                                               END IF;

                                               IF Information[i].xs <> Information[i].xsnext;
                                                        IF Information[i].xs < Information[i].xsnext
                                                        Information[i].xs := Information[i].xs + 1.0;
```

```
                                    Information[i].xg                          :=
Information[i].vf*.5+Information[i].xg-.5*HeadWind+1.0;

                                    ELSE
                                    Information[i].xs := Information[i].xs - 1.0;
                                    Information[i].xg                          :=
Information[i].vf*.5+Information[i].xg-.5*HeadWind-1.0;
                                    END IF;
                              ELSE
                                    Information[i].xg                          :=
Information[i].vf*.5+Information[i].xg-.5*HeadWind;
                              END IF;

                              IF Information[i].ys <> Information[i].ysnext;
                                    IF Information[i].ys < Information[i].ysnext;
                                    Information[i].ys := Information[i].ys + 1.0;
                                    Information[i].yg := Information[i].yg + 1.0;

                                    ELSE
                                    Information[i].ys := Information[i].ys - 1.0;
                                    Information[i].yg := Information[i].ys - 1.0;

                                    END IF;
                              END IF;
                        END IF;
{
OUTPUT(i," ",SimTime, " ",Information[i].xsnext," ",Information[i].xs, " ",Information[i].ysnext,"
",Information[i].ys)
}
                              ASK  Information[i].C17  TO  FlyPosition(Information[i].xs,
Information[i].ys, Information[i].altitude,
                            Information[i].vfk, Information[i].vf, Information[i].xg,
                            Information[i].yg, Information[i].weight,
                            Information[i].NumberOfSteps);
{
OUTPUT(Information[i].PlaneNumber,"   ",SimTime, "   ",Information[i].xs,"   ",Information[i].xg,"
",Information[i].yg);
}
          END FOR;
          TELL SELF greenLight;
        END WAIT;
      END WHILE;
    END METHOD {Fly};

  {*****METHOD NEEDED FOR AIRDROP*****}

  TELL METHOD greenLight;

    BEGIN

      FOR i := 2 TO NumberofPlanes;
        IF Information[i].xg >= 0.0
          IF Information[i].fullLoad
```

```
{
OUTPUT("Plane ", i, " GREEN LIGHT");
}
                    Information[i].fullLoad := FALSE;
                    TELL Information[i].C17 greenLight (Information[i].PlaneNumber);
                 END IF;
              END IF;
           END FOR;

        END {TELL} METHOD {greenLight};

END OBJECT {TotalVortexObj};

END {IMPLEMENTATION} MODULE {VortexControlMod}.
```

Definition Vortex Module

```
DEFINITION MODULE VortexMod;

FROM MathMod IMPORT pi;
FROM globalMod IMPORT rho, CrossWind, HeadWind;


TYPE

{Record Definition for Vortex Position Type }

                VortexPositionType  = RECORD
                        xCord   : REAL;           {X Air Coordinate for Vortex Position}
                        yCord   : REAL;           {Y Air Coordinate for Vortex Position}
                        zCord   : REAL;           {Z Air Coordinate (Altitude) for Vortex Position}
                        Gamd    : REAL;           {Decayed Vortex Strength}
                        Vage    : REAL;           {Vortex Age}
                        v       : REAL;           {not used}
                        w       : REAL;           {not used}
                        radius  : REAL;           {Radius from center with threshold swirl velocity}
                        NDyCord : REAL;                   {No drift Y Air Coordinate for calculating
actual vortex separation}
                        Airy    : REAL;           {Y Coordinate of the Airplane when vortex was
generated}
                END RECORD;

                CompleteType = ARRAY INTEGER OF VortexPositionType;

                RightVortexObj = OBJECT;

                        CompletePosition : CompleteType; {Vortex Position Array}
                        beff     :  REAL;           {Effective Wing Span}
                        ar       :  REAL;           {Aspect Ratio}
                        s        :  REAL;           {Wing Area}
                        bp, vs1, vs2, vs3, ys :  REAL;{Vortex span, wind speeds, Y Air Coordinate}
                        weight, vf, vfk, cl, altitude, gmod, gam : REAL;
                        {Weight, velocity (ft/sec), velocity (knots), coefficient of lift, altitude, Gamma
modification factor from lidar, Gamma (Vortex Strength)}
                        delx, delt, xs, vd, yd    : REAL;     {Change in x, change in time, X Air
Coordinate, Vertical velocity, Horizontal velocity}
                        i        : INTEGER;

                ASK METHOD ObjInit;
                ASK METHOD writeData(IN NumberOfSteps  :  INTEGER);
                ASK METHOD Update(IN beff, ar, s, bp, weight, vfk,vf, cl, altitude, xs, ys, gmod, vs1,
vs2, vs3 : REAL; IN NumberOfSteps  :  INTEGER);
                ASK METHOD BackwardUpdate(IN beff, ar, s, bp, weight, vfk,vf, cl, altitude, xs, ys,
gmod, vs1, vs2, vs3 : REAL; IN NumberOfSteps  :  INTEGER);
```

```
                END OBJECT {RightVortexObj};

                LeftVortexObj = OBJECT;

                        CompletePosition : CompleteType; {Vortex Position Array}
                        beff    : REAL;             {Effective Wing Span}
                        ar      : REAL;             {Aspect Ratio}
                        s       : REAL;             {Wing Area}
                        bp, vs, ys : REAL;                  {Vortex   span,   wind   speed,   Y   Air
Coordinate}
                        weight, vf, vfk, cl, altitude, gmod, gam : REAL;
                        {Weight, velocity (ft/sec), velocity (knots), coefficient of lift, altitude, Gamma
modification factor from lidar, Gamma (Vortex Strength)}
                        delx, delt, xs, vd, yd  : REAL;     {Change  in  x,  change  in  time,  X  Air
Coordinate, Vertical velocity, Horizontal velocity}
                        i       : INTEGER;

                ASK METHOD ObjInit;
                ASK METHOD writeData(IN NumberOfSteps  :  INTEGER);
                ASK METHOD Update(IN beff, ar, s, bp, weight, vfk,vf, cl, altitude, xs, ys, gmod, vs1,
vs2, vs3 : REAL; IN NumberOfSteps  :  INTEGER);
                ASK METHOD BackwardUpdate(IN beff, ar, s, bp, weight, vfk,vf, cl, altitude, xs, ys,
gmod, vs1, vs2, vs3 : REAL; IN NumberOfSteps  :  INTEGER);

                END OBJECT {LeftVortexObj};

VAR
        sD  :  REAL;

END {DEFINITION} MODULE {VortexMod}.
```

Implementation Vortex Module

```
IMPLEMENTATION MODULE VortexMod;

FROM MathMod IMPORT pi;
FROM globalMod IMPORT rho, CrossWind1, CrossWind2, CrossWind3, ShearAlt1, ShearAlt2,
HeadWind, windseed1, windseed2, windseed3;

OBJECT RightVortexObj;

   ASK METHOD ObjInit;

   BEGIN
      delx  := 100.0;
      delt  := delx/vf;

{OLD CODE FROM FIRST TRY}

         {NEW (CompletePosition, 1..800);
         NEW (CompletePosition[1]);

         CompletePosition[1].xCord := 100.0+xs;
         CompletePosition[1].yCord := ys+bp/2.0;
         CompletePosition[1].zCord := altitude;
         CompletePosition[1].Vage  := delx/vf;
         CompletePosition[1].Gamd  := gam;
         CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);
         CompletePosition[1].v     := 0.0;
         CompletePosition[1].w     := 0.0;


         FOR i := 2 TO 800;
         NEW (CompletePosition[i]);

         CompletePosition[i].xCord := xs + (FLOAT(i)*delx);

         vd          :=          -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));
         yd          :=          CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));
         CompletePosition[i].yCord := ys+CompletePosition[i-1].yCord+vs1*delt+yd*delt;
         CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
         CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
         IF CompletePosition[i].Vage <= 60.0;
                  CompletePosition[i].Gamd  := gam;
         ELSE
                  CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
         END IF;
         CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
```

```
        CompletePosition[i].v    := 0.0;
        CompletePosition[i].w    := 0.0;

        END FOR;}


        END METHOD {ObjInit};

        ASK METHOD Update(IN beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1, vs2, vs3 :
REAL; IN NumberOfSteps  :  INTEGER);
                BEGIN
        delx  :=  100.0;
        delt  :=  delx/vf;
        rho   :=  rho;
        gam   :=  weight/(rho*vf*bp)*gmod;

{Verification Output}
{OUTPUT(vs1, " ", vs2, " ", vs3);}

{Create the array}

        NEW (CompletePosition, 1..NumberOfSteps);
        NEW (CompletePosition[1]);

{Calculate the first point}

        CompletePosition[1].xCord  := 100.0+xs;              {100 ft behind aircraft}
        CompletePosition[1].yCord  := ys+bp/2.0;             {Off right wing of aircraft}
        CompletePosition[1].zCord  := altitude;              {Aircraft altitude}
        CompletePosition[1].Vage   := delx/vf;
        CompletePosition[1].Gamd   := gam;                           {Use gam from above}
        CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);        {Threshold set at
20 ft/sec}
        CompletePosition[1].v      := 0.0;                   {Not Used}
        CompletePosition[1].w      := 0.0;                   {Not Used}
        CompletePosition[1].NDyCord := ys + bp/2.0;          {Keep no drift spacing}
        CompletePosition[1].Airy   := ys;                    {Aircraft  position  when  vortex  point
created}

        FOR i := 2 TO NumberOfSteps;

        NEW (CompletePosition[i]);

{Calculate the remaining points}

        CompletePosition[i].xCord := xs + (FLOAT(i)*delx);

{This is the actual separation between the vortices from Babarsky's FAX}

        sD   := ABS(CompletePosition[i-1].NDyCord-CompletePosition[i-1].Airy);
        CompletePosition[i].Airy   := ys;

{Vertical and Horizontal Velocity Calculations}
```

```
        vd              :=              (-1.0*CompletePosition[i-1].Gamd)/(4.0*pi)*((CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord)/(sD*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord))));
        yd              :=              ((CompletePosition[i-1].Gamd)/(4.0*pi))*(sD*sD)/(CompletePosition[i-
1].zCord*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord)));

{This code will go back to the original calculation

        vd              :=              -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));
        yd              :=              CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));
        }

        CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;

{Update the separation between vortices}

        CompletePosition[i].NDyCord := CompletePosition[i-1].NDyCord+yd*delt;

{Use the correct wind value based on vortex altitude}
        IF CompletePosition[i].zCord <= ShearAlt1;
                IF CompletePosition[i].zCord <= ShearAlt2;
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs3*delt+yd*delt;
                ELSE
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs2*delt+yd*delt;
                END IF;
        ELSE
                CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs1*delt+yd*delt;
        END IF;
        CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
        IF CompletePosition[i].Vage <= 60.0;
        CompletePosition[i].Gamd  := gam;
        ELSE
        CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
        END IF;
        CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
        CompletePosition[i].v    := 0.0;
        CompletePosition[i].w    := 0.0;

        END FOR;

        END METHOD {Update};

{This method updates the vortex position when the aircraft fly}

        ASK METHOD BackwardUpdate(IN beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1,
vs2, vs3 : REAL; IN NumberOfSteps  :  INTEGER);
                BEGIN
        delx  := 100.0;
        delt  := delx/vf;
        rho   := rho;
        gam   := weight/(rho*vf*bp)*gmod;

{OUTPUT(vs1, " ", vs2, " ", vs3);}

{Update the first point}

        CompletePosition[1].xCord    := 100.0+xs;
        CompletePosition[1].yCord    := ys+bp/2.0;
        CompletePosition[1].zCord    := altitude;
        CompletePosition[1].Vage     := delx/vf;
        CompletePosition[1].Gamd     := gam;
        CompletePosition[1].radius   := CompletePosition[1].Gamd/(2.0*pi*20.0);
        CompletePosition[1].v        := 0.0;
        CompletePosition[1].w        := 0.0;
        CompletePosition[1].NDyCord  := ys +bp/2.0;
        CompletePosition[1].Airy     := ys;

{Update the remaining points from the end of the vortex to the front}

        FOR i := NumberOfSteps DOWNTO 2;



        CompletePosition[i].xCord := xs + (FLOAT(i)*delx);

        sD := ABS(CompletePosition[i-1].NDyCord-CompletePosition[i-1].Airy);
        CompletePosition[i].Airy   := CompletePosition[i-1].Airy;

        vd              :=              (-1.0*CompletePosition[i-1].Gamd/(4.0*pi))*((CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord)/(sD*(sD*sD+(CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord))));
        yd           :=              (CompletePosition[i-1].Gamd/(4.0*pi))*((sD*sD)/(CompletePosition[i-
1].zCord*(sD*sD+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord))));

{This Code will go back to the original vortex Calculation
        vd           :=              -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));
        yd           :=              CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));

        }



        CompletePosition[i].NDyCord := CompletePosition[i-1].NDyCord+yd*delt;

        IF CompletePosition[i].zCord <= ShearAlt1;
                IF CompletePosition[i].zCord <= ShearAlt2;
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs3*delt+yd*delt;
{Verification OUTPUT}
{OUTPUT ("THREE");}
```

```
                ELSE
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs2*delt+yd*delt;
{OUTPUT ("TWO");}
                END IF;
        ELSE
                CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs1*delt+yd*delt;
{OUTPUT ("ONE");}

        END IF;
        CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
        CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
        IF CompletePosition[i].Vage <= 60.0;
        CompletePosition[i].Gamd  := gam;
        ELSE
        CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
        END IF;
        CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
        CompletePosition[i].v     := 0.0;
        CompletePosition[i].w     := 0.0;

{Vortex Verification OUTPUT}
{OUTPUT (i,CompletePosition[i].xCord,CompletePosition[i].yCord,CompletePosition[i].zCord);}

        END FOR;


        END METHOD {BackwardUpdate};

        ASK METHOD writeData(IN NumberOfSteps  :  INTEGER);
                BEGIN;

                OUTPUT;
                OUTPUT ("Step  x Cord   yCord  zCord  Gamd  Vage  Radius   ");

                OUTPUT;
                FOR i :=1 TO NumberOfSteps;
                OUTPUT  (i,"   ",  CompletePosition[i].xCord,  "   ",  CompletePosition[i].yCord,  "
",CompletePosition[i].zCord,"       ",CompletePosition[i].Vage,"        ",CompletePosition[i].Gamd,        "
",CompletePosition[i].radius," ", CompletePosition[i].NDyCord, " ", CompletePosition[i].Airy);
                END FOR;

        END METHOD {writeData};


END OBJECT {RightVortexObj};

OBJECT LeftVortexObj;


        ASK METHOD ObjInit;

        BEGIN
```

```
                {gmod  := 0.8;
                gam   := weight/(rho*vf*bp)*gmod;}
                delx  := 100.0;
                delt  := delx/vf;

 {OLD CODE FROM FIRST TRY}

          {NEW (CompletePosition, 1..800);
          NEW (CompletePosition[1]);

          CompletePosition[1].xCord := 100.0+xs;
          CompletePosition[1].yCord := ys-bp/2.0;
          CompletePosition[1].zCord := altitude;
          CompletePosition[1].Vage  := delx/vf;
          CompletePosition[1].Gamd  := gam;
          CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);
          CompletePosition[1].v     := 0.0;
          CompletePosition[1].w     := 0.0;


          FOR i := 2 TO 800;
          NEW (CompletePosition[i]);

          CompletePosition[i].xCord := xs + (FLOAT(i)*delx);

          vd        :=        -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));
          yd        :=        -CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));
          CompletePosition[i].yCord := ys+CompletePosition[i-1].yCord+vs1*delt+yd*delt;
          CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
          CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
          IF CompletePosition[i].Vage <= 60.0;
          CompletePosition[i].Gamd  := gam;
          ELSE
          CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
          END IF;
          CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
          CompletePosition[i].v     := 0.0;
          CompletePosition[i].w     := 0.0;

          END FOR;}


          END METHOD {ObjInit};


          ASK METHOD Update(IN beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1, vs2, vs3 :
REAL;  IN NumberOfSteps  :  INTEGER);
                BEGIN

          delx  :=  100.0;
```

```
        delt  := delx/vf;
        gam   := weight/(rho*vf*bp)*gmod;

{OUTPUT(vs1, " ", vs2, " ", vs3);
OUTPUT("UPDATE");}

        NEW (CompletePosition, 1..NumberOfSteps);
        NEW (CompletePosition[1]);


        CompletePosition[1].xCord := 100.0+xs;
        CompletePosition[1].yCord := ys-bp/2.0;
        CompletePosition[1].zCord := altitude;
        CompletePosition[1].Vage  := delx/vf;
        CompletePosition[1].Gamd  := gam;
        CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);
        CompletePosition[1].v     := 0.0;
        CompletePosition[1].w     := 0.0;
        CompletePosition[1].NDyCord := ys-bp/2.0;
        CompletePosition[1].Airy    := ys;

        FOR i := 2 TO NumberOfSteps;
{OUTPUT(i);}
        NEW (CompletePosition[i]);

        CompletePosition[i].xCord := xs + (FLOAT(i)*delx);
        sD  := ABS(CompletePosition[i-1].NDyCord-CompletePosition[i-1].Airy);
        CompletePosition[i].Airy  := ys;

        vd              :=              (-1.0*CompletePosition[i-1].Gamd)/(4.0*pi)*((CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord)/(sD*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord))));
        yd          :=          -1.0*((CompletePosition[i-1].Gamd)/(4.0*pi))*(sD*sD)/(CompletePosition[i-
1].zCord*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord)));

        CompletePosition[i].NDyCord := CompletePosition[i-1].NDyCord+yd*delt;




{typos somewhere  Line 302
        vd              :=              (-1.0*CompletePosition[i-1].Gamd)/(4.0*pi)*((CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord)/(sD*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord))));
        yd          :=          -((CompletePosition[i-1].Gamd)/(4.0*pi))*(sD*sD)/(CompletePosition[i-
1].zCord*((sD*sD)+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord)));
}
{These are the old formulas using effective wing span
        vd              :=              -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));
```

```
        yd              :=              -CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));
}
        CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
        IF CompletePosition[i].zCord <= ShearAlt1;
                IF CompletePosition[i].zCord <= ShearAlt2;
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs3*delt+yd*delt;
                ELSE
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs2*delt+yd*delt;
                END IF;
        ELSE
                CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs1*delt+yd*delt;
        END IF;

        CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
        IF CompletePosition[i].Vage <= 60.0;
        CompletePosition[i].Gamd  := gam;
        ELSE
        CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
        END IF;
        CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
        CompletePosition[i].v     := 0.0;
        CompletePosition[i].w     := 0.0;

        END FOR;


        END METHOD {Update};

        ASK METHOD BackwardUpdate(IN beff, ar, s, bp, weight, vfk, vf, cl, altitude, xs, ys, gmod, vs1,
vs2, vs3 : REAL;  IN NumberOfSteps  :  INTEGER);
                BEGIN

        delx  :=  100.0;
        delt  :=  delx/vf;
        gam   :=  weight/(rho*vf*bp)*gmod;

{OUTPUT(vs1, " ", vs2, " ", vs3);
OUTPUT("BACKWARD");
OUTPUT(ShearAlt1, ShearAlt2);}

        CompletePosition[1].xCord := 100.0+xs;
        CompletePosition[1].yCord := ys-bp/2.0;
        CompletePosition[1].zCord := altitude;
        CompletePosition[1].Vage  := delx/vf;
        CompletePosition[1].Gamd  := gam;
        CompletePosition[1].radius := CompletePosition[1].Gamd/(2.0*pi*20.0);
        CompletePosition[1].v     := 0.0;
        CompletePosition[1].w     := 0.0;
        CompletePosition[1].NDyCord := ys-bp/2.0;
        CompletePosition[1].Airy   := ys;

        FOR i := NumberOfSteps DOWNTO 2;
```

```
{OUTPUT (i);}
        CompletePosition[i].xCord := xs + (FLOAT(i)*delx);
        sD  := ABS(CompletePosition[i-1].NDyCord-CompletePosition[i-1].Airy);
        CompletePosition[i].Airy  := CompletePosition[i-1].Airy;

        vd              :=              (-1.0*CompletePosition[i-1].Gamd/(4.0*pi))*((CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord)/(sD*(sD*sD+(CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord))));
        yd          :=          -1.0*(CompletePosition[i-1].Gamd/(4.0*pi))*((sD*sD)/(CompletePosition[i-
1].zCord*(sD*sD+(CompletePosition[i-1].zCord*CompletePosition[i-1].zCord))));

        CompletePosition[i].NDyCord := CompletePosition[i-1].NDyCord+yd*delt;



{These are the old formulas using effective wing span
        vd              :=              -1.0*CompletePosition[i-1].Gamd/(2.0*pi)*4.0*CompletePosition[i-
1].zCord*CompletePosition[i-1].zCord/(bp*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-
1].zCord));

        yd              :=              -CompletePosition[i-1].Gamd/(2.0*pi)*bp*bp/(2.0*CompletePosition[i-
1].zCord*(bp*bp+4.0*CompletePosition[i-1].zCord*CompletePosition[i-1].zCord));
}
        IF CompletePosition[i].zCord <= ShearAlt1;
                IF CompletePosition[i].zCord <= ShearAlt2;
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs3*delt+yd*delt;
                ELSE
                        CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs2*delt+yd*delt;
                END IF;
        ELSE
                CompletePosition[i].yCord := CompletePosition[i-1].yCord+vs1*delt+yd*delt;
        END IF;
        CompletePosition[i].zCord := CompletePosition[i-1].zCord+vd*delt;
        CompletePosition[i].Vage  := delx*FLOAT(i)/vf;
        IF CompletePosition[i].Vage <= 60.0;
        CompletePosition[i].Gamd  := gam;
        ELSE
        CompletePosition[i].Gamd  := gam*(60.0/CompletePosition[i].Vage);
        END IF;
        CompletePosition[i].radius := CompletePosition[i].Gamd/(2.0*pi*20.0);
        CompletePosition[i].v     := 0.0;
        CompletePosition[i].w     := 0.0;

        END FOR;

        END METHOD {BackwardUpdate};


        ASK METHOD writeData(IN NumberOfSteps  :  INTEGER);
                BEGIN;

                OUTPUT;
                OUTPUT ("Step  xCord   yCord  zCord  Vage    Gamma   Radius    NoDrifty Ystart");
                OUTPUT;
                FOR i :=1 TO NumberOfSteps;
                OUTPUT  (i,"  ",  CompletePosition[i].xCord,  "  ",  CompletePosition[i].yCord,  "
",CompletePosition[i].zCord,"    ",CompletePosition[i].Vage,"    ",CompletePosition[i].Gamd,    "    ",
CompletePosition[i].radius," ", CompletePosition[i].NDyCord, " ", CompletePosition[i].Airy);
                END FOR;

        END METHOD {writeData};


END OBJECT {LeftVortexObj};


END {IMPLEMENTATION} MODULE {VortexMod}.
```

Appendix B

Shared MODSIM Code

Main Vortex Module

```
MAIN MODULE vortex;

FROM inputMod        IMPORT readData, disposeStreams;
FROM globalMod       IMPORT i, NumberofPlanes, nu, knotconv, initializeData, repeat;
FROM AirplaneMod     IMPORT C17Obj;
FROM VortexMod       IMPORT RightVortexObj, LeftVortexObj;
FROM MathMod         IMPORT pi;
FROM VortexControlMod IMPORT Airdrop;
FROM SimMod          IMPORT ResetSimTime, StartSimulation;
FROM UtilMod         IMPORT DateTime;

VAR

BEGIN
{   OUTPUT ("Start time:  ", DateTime);}

{ ----- Start the input questions and set up the random seeds ----- }

   readData;
   initializeData;

   FOR repeat := 1 TO 50;

      ResetSimTime(0.0);

      { ----- Create the Vortex Control Object named Airdrop ----- }

      NEW (Airdrop);

{ ----- The following loops output the vortex information ----- }

{
   OUTPUT;
   OUTPUT("Airplane 1 Data");
   OUTPUT;
   FOR i  := 1 TO NumberofPlanes;
      OUTPUT;
      OUTPUT(Airdrop.Information[i].PlaneNumber,"          ",   Airdrop.Information[i].xs,"          ",
Airdrop.Information[i].ys, "     ", Airdrop.Information[i].altitude, "     ", Airdrop.Information[i].xg, "     ",
Airdrop.Information[i].yg);
   END FOR;
   FOR  i:= 1 TO NumberofPlanes;
      OUTPUT("Airplane  ", i, " Data");
      ASK                Airdrop.Information[i].C17.RightVortex                TO
writeData(Airdrop.Information[i].NumberOfSteps);
      ASK Airdrop.Information[i].C17.LeftVortex TO writeData(Airdrop.Information[i].NumberOfSteps);
   END FOR;
}

      { ----- Schedule the first event to initiate the simulation ----- }

      TELL Airdrop TO Fly;

      StartSimulation;

{ ----- More vortex information.  This will write the position of the vortex at the end of the simulation ----- }

{
   FOR i:= 1 TO NumberofPlanes;
      OUTPUT("Airplane  ", i, " Data");
      ASK                         Airdrop.Information[i].C17.RightVortex                         TO
writeData(Airdrop.Information[i].NumberOfSteps);
      ASK Airdrop.Information[i].C17.LeftVortex TO writeData(Airdrop.Information[i].NumberOfSteps);
   END FOR;
}

      DISPOSE (Airdrop);

{   OUTPUT("End time:  ", DateTime);}


   END FOR;

   disposeStreams;

END {MAIN} MODULE {Vortex}.
```

Definition Global Module

```
DEFINITION MODULE globalMod;

FROM RandMod        IMPORT RandomObj;
FROM VortexMod      IMPORT RightVortexObj, LeftVortexObj;
FROM rightJumperMod IMPORT rightJumperObj;
FROM leftJumperMod  IMPORT leftJumperObj;

CONST

   re = 20855531.5;
   nu  = 0.0001654;
   knotconv = 1.69085;  {Converts knots to ft/sec}

TYPE

   eType      = ARRAY INTEGER OF INTEGER;
   delType    = ARRAY INTEGER, INTEGER OF REAL;
   matrixType = ARRAY INTEGER, INTEGER OF REAL;
   vectorType = ARRAY INTEGER OF REAL;

   encounterType = RECORD
      airplane : INTEGER;
      side     : STRING;
      position : INTEGER;
   END RECORD {encounterType};

{Set up Element Position Type for building of elements}

   ElementPositionType = RECORD
      ElementPosNum : INTEGER;
      Intrail       : REAL;
      CrossTrack    : REAL;
   END RECORD;

{Collection of Element Position Types for Formation Definition}

   ElementGeometryType    =  ARRAY INTEGER OF ElementPositionType;

   FormationPositionType  = RECORD;
      PositonNumber   : INTEGER;
      Intrail         : REAL;
      CrossTrack      : REAL;
   END RECORD;

   FormationGeometryType  = ARRAY INTEGER OF FormationPositionType;

VAR
```

```
   NumberofPlanes    : INTEGER;          {Number of Planes in Formation}
   PlanesPerElement  : INTEGER;          {Number of Planes per Element}
   NumberOfElements  : INTEGER;                  {Number of Elements in Formation}
   i, j, repeat      : INTEGER;          {Loop control variables}

   ElementSpacing    : REAL;
   ElementGeometry   : ElementGeometryType;
   FormationGeometry : FormationGeometryType;

   CrossWind1      : REAL;               {Mean Crosswind at drop altitude to ShearAlt1}
   CrossWind2      : REAL;               {Mean Crosswind from ShearAlt1 to ShearAlt2}
   CrossWind3      : REAL;               {Mean Crosswind from ShearAlt2 to ground}
   ShearAlt1     : REAL;                 {Altitude where wind first shifts}
   ShearAlt2     : REAL;                 {Altitude where wind shifts second time}
   StandDev1       : REAL;               {Standard deviation for Crosswind1 normal distribution for
wind velocity}
   StandDev2       : REAL;               {Standard deviation for Crosswind2 normal distribution for
wind velocity}
   StandDev3       : REAL;               {Standard deviation for Crosswind3 normal distribution for
wind velocity}
   trailBox       : INTEGER;             {Size of tolerance for in-trail spacing}
   lateralBox      : INTEGER;            {Size of tolerance for lateral spacing}
   altitudeBox      : INTEGER;           {Size of tolerance for altitude deviation (not used)}
   HeadWind        : REAL;               {Headwind at drop altitude}
   rho          : REAL;          {Air density}
   altitude      : REAL;         {Altitude}
   vfk          : REAL;          {Aircraft velocity (knots)}
   weight         : REAL;        {Aircraft weight}
   RunLength       : REAL;               {Length of simulation run}
   vs1          : REAL;          {Crosswind1 speed (ft/sec) after random draw}
   vs2          : REAL;          {Crosswind2 speed (ft/sec) after random draw}
   vs3          : REAL;          {Crosswind3 speed (ft/sec) after random draw}

   rightJumper     : rightJumperObj;
   leftJumper      : leftJumperObj;
   e            : eType;
   del          : delType;
   seed1          : RandomObj;
   seed2          : RandomObj;
   seed3          : RandomObj;
   seed4          : RandomObj;
   windseed1        : RandomObj;          {Seed for Crosswind1 normal distribution}
   windseed2        : RandomObj;          {Seed for Crosswind2 normal distribution}
   windseed3        : RandomObj;          {Seed for Crosswind3 normal distribution}
   trailseed      : RandomObj;            {Seed for in-trail uniform distribution}
   lateralseed     : RandomObj;           {Seed for lateral uniform distribution}
   timeseed        : RandomObj;           {Seed for time in position (not used)}

PROCEDURE initializeData;

END {DEFINITION} MODULE {globalMod}.
```

Implementation Global Module

```
IMPLEMENTATION MODULE globalMod;

FROM RandMod  IMPORT FetchSeed;
FROM inputMod IMPORT jumperseed;

PROCEDURE initializeData;

   BEGIN

{Create random streams and get seed numbers}

      NEW (e, 1..5);
      NEW (del, 1..3, 1..3);
      NEW (seed1);
      NEW (seed2);
      NEW (seed3);
      NEW (seed4);
      NEW (windseed1);
      NEW (windseed2);
      NEW (windseed3);
      NEW (trailseed);
      NEW (lateralseed);
      NEW (timeseed);
      ASK seed1 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK seed2 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK seed3 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK seed4 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK windseed1 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK windseed2 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK windseed3 TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK trailseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK lateralseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));
      ASK timeseed TO SetSeed (FetchSeed ( jumperseed.UniformInt (1,10) ));


      e[1] := 1;
      e[2] := 2;
      e[3] := 3;
      e[4] := 1;
      e[5] := 2;

      FOR i := 1 TO 3
         FOR j := 1 TO 3
            IF i = j
               del[i,j] := 1.0;
            ELSE
               del[i,j] := 0.0;
            END IF;
      END FOR;
    END FOR;

  END PROCEDURE {initializeData};

END {IMPLEMENTATION} MODULE {globalMod}.
```

Definition Input Module

```
DEFINITION MODULE inputMod;

FROM IOMod   IMPORT StreamObj, FileUseType(Output);
FROM RandMod IMPORT RandomObj;

  VAR
        i             : INTEGER;
        Scenario      : INTEGER;
    dummy         : INTEGER;
    stickSize     : INTEGER;
    printTrajectory : BOOLEAN;

    streamI       : StreamObj; {TO WRITE JUMPER INFORMATION TO AN OUTPUT FILE}
    informationfile : STRING;
    streamE       : StreamObj; {TO WRITE ENCOUNTER DATA TO AN OUTPUT FILE}
    encounterfile  : STRING;
    streamS       : StreamObj; {TO WRITE SCATTER   DATA TO AN OUTPUT FILE}
    scatterfile   : STRING;

    extension     : STRING;

    jumperseed    : RandomObj;

    {TROUBLE SHOOTING INPUT}
{
    dtdrift       : REAL;
}

{Get input data }

  PROCEDURE readData;

  PROCEDURE disposeStreams;

END {DEFINITION} MODULE {inputMod}.
```

Implementation Input Module

```
IMPLEMENTATION MODULE inputMod;

FROM globalMod IMPORT NumberofPlanes, PlanesPerElement, NumberOfElements,
            ElementGeometry, ElementSpacing, FormationGeometry,
            altitude, vfk, CrossWind1, CrossWind2, CrossWind3,
                      ShearAlt1, ShearAlt2, HeadWind, rho, weight, knotconv,
                 trailBox, lateralBox, StandDev1, StandDev2, StandDev3;
FROM RandMod  IMPORT FetchSeed;
FROM globalMod IMPORT loop;

 PROCEDURE readData;
   BEGIN
     OUTPUT("Which Scenario do you want to run?");
     OUTPUT("    Enter  1  For Edwards Test");
     OUTPUT("    Enter  2  For Custom Formation");
     INPUT(Scenario);

     CASE Scenario

     WHEN 1:
       NumberofPlanes := 2;
       PlanesPerElement := 2;

       NEW (ElementGeometry, 1..PlanesPerElement);
       NEW (ElementGeometry[1]);
       ElementGeometry[1].ElementPosNum := 1;
       ElementGeometry[1].Intrail      := 0.0;
       ElementGeometry[1].CrossTrack   := 0.0;
       NEW (ElementGeometry[2]);
       ElementGeometry[2].Intrail := 15000.0;
       ElementGeometry[2].CrossTrack := 0.0;
       stickSize := 6;
       weight := 385000.0;
       altitude := 1000.0;
       vfk := 135.0;
       CrossWind1 := 0.0;
       HeadWind := 0.0;
       rho := 0.002000;

     WHEN 2 :

       OUTPUT("How many airplanes for this run?");
       INPUT(NumberofPlanes);
       OUTPUT("How many airplanes per element?");
       INPUT(PlanesPerElement);
       OUTPUT("How many jumpers exiting each side of the aircraft?");
       INPUT(stickSize);

       NEW (ElementGeometry, 1..PlanesPerElement);
```

```
        NEW (ElementGeometry[1]);
        ElementGeometry[1].ElementPosNum := 1;
        ElementGeometry[1].Intrail      := 0.0;
        ElementGeometry[1].CrossTrack   := 0.0;

        IF PlanesPerElement > 1;
          FOR i := 2 TO PlanesPerElement;
            NEW (ElementGeometry[i]);
            ElementGeometry[i].ElementPosNum := i;
            OUTPUT("All element positions are relative to element lead.");
            OUTPUT("What is the in trail distance for plane  ", i," ?");
            OUTPUT("Spacing must be input as a real number with decimal point");
            INPUT(ElementGeometry[i].Intrail);
            OUTPUT("What is the lateral spacing for plane  ", i," ?");
            OUTPUT("Positive to the Right, Negative to the left.");
            INPUT(ElementGeometry[i].CrossTrack);
          END FOR;
        END IF;

        IF (NumberofPlanes MOD PlanesPerElement) = 0;
          NumberOfElements := (NumberofPlanes DIV PlanesPerElement);
        ELSE
          NumberOfElements := (NumberofPlanes DIV PlanesPerElement)+1;
        END IF;

        IF NumberOfElements > 1;
          OUTPUT("What is the spacing between element leaders?");
          INPUT(ElementSpacing);
        ELSE
        END IF;

        OUTPUT("What tolerance are the trailing aircraft following?");
        OUTPUT("Enter the tolerance for fore and aft spacing (Must be an integer)");
        INPUT (trailBox);
        OUTPUT("Enter the tolerance for lateral spacing (Must be an integer)");
        INPUT (lateralBox);

        OUTPUT("What is the weight of all aircraft?");

        INPUT(weight);
        OUTPUT("What is the airdrop altitude? (ft AGL) (Real number with decimal)");
        INPUT(altitude);
        OUTPUT("What is the airspeed of the formation in knots?");
        INPUT(vfk);
        OUTPUT("What is the Cross Wind Component? (knots)");
                    OUTPUT("Input up to three different winds with altitude where wind changes");
                    OUTPUT("and  standard  deviation  of  the  normal  distribution  for  wind
variation");
                    OUTPUT("(All eight values must be entered.  If constant wind, then zeros can
be ");
                    OUTPUT("entered for the last seven values.)");
                    OUTPUT("i.e.  5 .5 500 10 1 200 0 .25 would indicate 5 knots with a normal");
                    OUTPUT("distribution and a standard deviation of .5 at drop altitude till 500");
```

```
                        OUTPUT("feet AGL, then 10 knots with a standard deviation of 1 until 200 feet
AGL,");
                        OUTPUT(" then calm to the ground with a standard deviation of .25");
          INPUT(CrossWind1, StandDev1, ShearAlt1, CrossWind2, StandDev2, ShearAlt2, CrossWind3,
StandDev3);
                        CrossWind1 := CrossWind1*knotconv;   {Convert knots to ft/s}
                        CrossWind2 := CrossWind2*knotconv;   {Convert knots to ft/s}
                        CrossWind3 := CrossWind3*knotconv;   {Convert knots to ft/s}
                        StandDev1  := StandDev1*knotconv;
                        StandDev2  := StandDev2*knotconv;
                        StandDev3  := StandDev3*knotconv;
          OUTPUT("What is the Head Wind Component? (knots)");
          INPUT(HeadWind);
                        HeadWind := HeadWind*knotconv; {Convert knots to ft/s}
          OUTPUT("What is the air density factor (rho)?");
          INPUT(rho);

        END CASE;

         OUTPUT ("ENTER OUTPUT FILENAME (WITH NO SUFFUX EXTENSION) FOR JUMPER
INFORMATION DATA :");
         INPUT  (informationfile);
    OUTPUT ("ENTER OUTPUT FILENAME (WITH NO SUFFIX EXTENSION) FOR ENCOUNTER
DATA : ");
    INPUT  (encounterfile);
    OUTPUT ("ENTER OUTPUT FILENAME (WITH NO SUFFIX EXTENSION) FOR SCATTER
DATA : ");
    INPUT  (scatterfile);
    OUTPUT ("ENTER AN INTEGER (BETWEEN 1 AND 10) FOR THE RANDOM SEED INPUT
FOR JUMPERS");
    INPUT  (dummy);

    NEW (jumperseed);
    ASK jumperseed TO SetSeed (FetchSeed (dummy));

    NEW (streamI);
    NEW (streamE);
    NEW (streamS);

    informationfile := informationfile + ".txt";
    encounterfile   := encounterfile   + ".txt";
    scatterfile     := scatterfile     + ".mat";

    ASK streamI TO Open (informationfile, Output);
    ASK streamE TO Open (encounterfile,   Output);
    ASK streamS TO Open (scatterfile,     Output);

    OUTPUT ("PLEASE CHOOSE ONE OF THE FOLLOWING OPTIONS:");
    OUTPUT ("      1 - WRITE INDIVIDUAL PARATROOP TRAJECTORY INFORMATION TO
SEPARATE FILES");
    OUTPUT ("    0 - DO NOT WRITE TRAJECTORY INFORMATION");
    INPUT  (dummy);
```

```
     IF dummy = 1
        OUTPUT ("ENTER UP TO FOUR (4) CHARACTERS TO DESIGNATE JUMPERS :");
        INPUT  (extension);
        printTrajectory := TRUE;
     ELSE
        printTrajectory := FALSE;
     END IF;

     {TROUBLE SHOOTING INPUTS}
{
     OUTPUT ("ENTER DRIFT UPDATE DT:");
     INPUT (dtdrift);
}
   END PROCEDURE {readData};

PROCEDURE disposeStreams;
   BEGIN

      DISPOSE(streamE);
      DISPOSE(streamS);

   END PROCEDURE {disposeStreams};

END {IMPLEMENTATION} MODULE {inputMod}.
```

Appendix C

Belano's MODSIM Code

Definition Calc Module

```
DEFINITION MODULE calcMod;


PROCEDURE gravCalc (IN a : REAL) : REAL;
PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);

END {DEFINITION} MODULE {calcMod}.
```

Implementation Calc Module

```
IMPLEMENTATION MODULE calcMod;


FROM MathMod IMPORT POWER, SIN, COS, SQRT, EXP;
FROM globalMod IMPORT re;

PROCEDURE gravCalc (IN a: REAL) : REAL;
    BEGIN
        RETURN 32.1741*POWER(re/(a+re), 2.0);
    END PROCEDURE {gravCalc};

PROCEDURE densityCalc (IN h, rhoz : REAL; OUT rho, sound : REAL);
    VAR

        t : REAL;

    BEGIN
        rho := rhoz * EXP(-1.0*h/23111.0 + 0.294 * SIN(h/28860.0) + 0.213 * SIN(h/86580.0));

        IF h > 0.0
            t := 518.688 - (3.56616E-03)*h;
            sound :=  49.02118 * SQRT(t);
            IF h > 36152.0
                sound := 968.08;
                IF h > 82345.0
                    t := 254.988 + (1.64592E-03)*h;
                    sound := 49.02118 * SQRT(t);
                    IF h > 155348.0
                        sound := 1105.0;
                        IF h > 262448.0
                            sound := 846.9;
                            IF h > 299516.0
                                t := -349.812 + 2.19456E-03*h
                            END IF;
                        END IF;
                    END IF;
                END IF;
            END IF;
        ELSE
            sound := 1116.44;
        END IF;

    END PROCEDURE {densityCalc};

END {IMPLEMENTATION} MODULE {calcMod}.
```

Definition Left Jumper Module

```
DEFINITION MODULE leftJumperMod;

FROM IOMod           IMPORT StreamObj, FileUSeType(Output);
FROM globalMod        IMPORT eType, matrixType, vectorType, encounterType;
FROM VortexMod        IMPORT RightVortexObj, LeftVortexObj;
FROM VortexControlMod IMPORT VortexControl;

TYPE

  leftJumperObj = OBJECT

    i1,
    i2,
    iend,
    ipts,
    ip,
    j1,
    j2,
    k,
    loop,
    mpts,
    myNumber,
    myPlane,
    bigloop : INTEGER;

    alt,
    alpha,
    alphad,
    beta,
    bxy,
    ca,
    cao,
    caa2,
    calpha,
    cbar,
    cbeta,
    cby2,
    cds,
    clo,
    clp,
    cm,
    cma,
    cma2,
    cmo,
    cmq,
    cn,
    cna,
    cna2,
cnb,
cnb2,
cnq,
cnr,
coswt,
coswtm,
cp,
cphi,
csl,
csn,
ct,
cy,
cyb,
cyb2,
cyr,
dens,
deptime,
dt,
dtpr,
dtpoll,
dtdrift,
fpc,
g,
gammad,
gees,
gmax,
h,
hmin,
mach,
mass,
myTime,
myDrift,
myDriftDirection,
pb,
phi,
psi,
q,
qb,
qdyn,
qs,
qsd,
rad,
rb,
rho,
rhoz,
sac,
sas,
sarea,
salpha,
sbc,
sbeta,
sbs,
sinwt,
sound,
sp,
sphi,
st,
t,
theta,
tpr,
tpoll,
tdrift,
ub1,
ub2,
ub3,
ue1,
ue2,
ue3,
uxy,
vp,
vp13,
vpe,
vpo,
w,
w2,
weight,
xbod,
xcg,
Xdrift,
xlast,
Ydrift    : REAL;

slength   : REAL; {length of suspension lines}
angle     : REAL; {the angle (in radians) which defines the "cone" of the suspension lines}
dcglength : REAL; {distance from end of suspension lines to paratrooper c.g.}
cweight   : REAL; {weight of canopy}
sweight   : REAL; {weight of suspension lines}

radius    : REAL;
addedmass : REAL;
distcm    : REAL;
sysmass   : REAL;
paymom    : REAL;
distcan   : REAL;
distline  : REAL;
distpay   : REAL;

addDrift  : BOOLEAN;

pcds,
pt,
pcdf,
pm    : vectorType; {1X2}

xe,
xs,
        xg,
        ue,
        wb,
        vwind,
        temp,
        fb,
        m,
        mb,
        fe,
        uedot,
        wbdot,
        hb    : vectorType; {1X3}

        in,
        jn,
        b,
        bn,
        bdot : matrixType; {3X3}

        lastRightLocation,
        lastLeftLocation : eType; {Dynamic Array}

        outfile : STRING;

        stream : StreamObj;

        encounter : encounterType;

        ASK METHOD ObjInit;
        TELL METHOD jump;
        ASK METHOD initialize (IN stick   : INTEGER;
                        IN myPlane : INTEGER);
        ASK METHOD pollVortices (IN vortexPlane : INTEGER);
        ASK METHOD changeDrift;
        ASK METHOD findDrift;

    END OBJECT {leftJumperObj};

END {DEFINITION} MODULE {leftJumperMod}.
```

Implementation Left Jumper Module

```
IMPLEMENTATION MODULE leftJumperMod;

FROM MathMod        IMPORT EXP, SIN, COS, POWER, SQRT, ATAN2, TAN, pi;
FROM VortexMod      IMPORT RightVortexObj, LeftVortexObj;
FROM globalMod      IMPORT re, e, del, i, j, NumberofPlanes, seed1, seed2, seed3, repeat;
FROM inputMod       IMPORT streamI, streamE, streamS, extension, printTrajectory;
FROM  globalMod          .IMPORT  CrossWind1,  CrossWind2,  CrossWind3,  ShearAlt1,  ShearAlt2,
HeadWind, vs1, vs2, vs3;
FROM calcMod        IMPORT gravCalc, densityCalc;
FROM SimMod         IMPORT SimTime;
FROM VortexControlMod IMPORT VortexControl, Airdrop;

{TROUBLE SHOOTING INPUT}
{
FROM inputMod IMPORT dtdrift;
}
OBJECT leftJumperObj;

  ASK METHOD ObjInit;

    BEGIN
      NEW(pcdf, 1..2);
      NEW(pm  , 1..2);

      NEW(fb  , 1..3);
      NEW(fe  , 1..3);
      NEW(hb  , 1..3);
      NEW(mb  , 1..3);
      NEW(temp , 1..3);
      NEW(ue  , 1..3);
      NEW(uedot, 1..3);
      NEW(vwind, 1..3);
      NEW(wb  , 1..3);
      NEW(wbdot, 1..3);
      NEW(xe  , 1..3);
      NEW(xs  , 1..3);
      NEW(xg  , 1..3);

      NEW(pcds, 1..4);
      NEW(pt  , 1..4);

      NEW(in  , 1..3, 1..3);
      NEW(jn  , 1..3, 1..3);
      NEW(b   , 1..3, 1..3);
      NEW(bn  , 1..3, 1..3);
      NEW(bdot, 1..3, 1..3);

    { system inertial properties }
```

```
        { parachute-payload system weight (lbs) = weight of jumper/gear under T-10C}

        weight := seed3.Normal (247.0, 25.0);

        mass  := weight/32.17;
        xcg   := 0.5;        { forebody c.g. (ft) in the horizontal }
        xbod  := 6.0;        { forebody length (ft) in the vertical }

        FOR i := 1 TO 3
          FOR j := 1 TO 3
            jn[i,j] := 0.0;
            in[i,j] := 0.0;
          END {j} FOR;
        END {i} FOR;

      { initial conditions }

        FOR i := 1 TO 3
          xe[i]   := 0.0; { (ft) 1: down range, 2: off range, 3: altitude loss                  }
          ue[i]   := 0.0; { (fps) 1: horizontal velocity, 2: lateral velocity, 3: ejection velocity positive
down }
          wb[i]   := 0.0; { ???                                                   }
          vwind[i] := 0.0; { (fps) 1: head (+) or tail (-) wind, 2: crosswind, 3: ????                  }
        END FOR;

        alt      := 0.0;    { altitude (ft)                 }
        hmin     := 0.0;    { ground level (ft)              }
        ue[1]    := 0.0;    { horizontal velocity (fps)         }
        ue[3]    := 0.0;    { ejection velocity, positive down      }
        theta    := 0.0;    { pitch angle (deg) nose up positive     }
        vwind[1] := 0.0;    { head (+) or tail (-) wind (fps)       }
        vwind[2] := 0.0;    { crosswind (fps)                 }
        dens     := 0.0;    { density (0 for standard atms) in slug/ft^3 }
        rhoz     := 0.002378; { ????                        }

        IF dens <> 0.0
          rhoz := dens * EXP (alt/23111.0 - 0.295 * SIN(alt/28860.0) - 0.213 * SIN(alt/86580.0))
        END IF;

      { program constants }

        dtpr   := 0.1;  { print interval (sec)                          }
        dtpoll := 0.5;  { poll vortex positions every 0.5 seconds                }
        dtdrift := 5.0; { change drift angle +/- 45.0 from current drift angle every 5 seconds }

      { forebody aerodynamic coefficients }

        cbar  := 6.0;          { reference length (ft)         }
        sarea := POWER(0.5,2.0)*pi; { reference area (ft^2)          }
        cna   := 0.0;          { normal force cn-alpha (/rad)     }
        cyb   := 0.0;          { side force cy-beta (/rad)      }
        caa2  := 0.0;          { axial force ca-alpha^2 (/rad^2)    }
```

```
   clo  := 0.0;            { roll torque coefficient (dimensionless) }
   clp  := 0.0;            { roll damping coefficient (/rad)         }
   cma  := -2.0;            { pitch moment cm-alpha (/rad)            }
   cmq  := -200.0;            { pitch damping (/rad)                 }
   cnb  := 0.0;            { yaw moment cn-beta (/rad)               }
   cnr  := 0.0;            { yaw damping (/rad)                    }

{ forebody drag versus mach number table }

   mpts   := 2;
   pm[1]  := 0.00; { mach number }
   pm[2]  := 2.00;
   pcdf[1] := 0.73+0.06*(360.0 - weight)/180.0; { drag coefficient }
   pcdf[2] := 0.73+0.06*(360.0 - weight)/180.0;

{ parachute drag-area versus time table }

   deptime := 0.25; { deployment time }
   ipts   := 2;

   IF deptime > 0.0
      ipts := ipts + 2; { ipts = 4 }
      pt[1]  := 0.0;
      pt[2]  := deptime;
      pt[3]  := 0.00 + deptime;
      pt[4]  := 2.80 + deptime;
      pcds[1] := 0.0;
      pcds[2] := 0.0;
      pcds[3] := 0.20;
      pcds[4] := 690.0;
   END IF;

{ convert EULER ANGLES to direction cosines }

   psi   := 0.0;
   phi   := 0.0;
   rad   := pi/180.0;
   st    := SIN (theta*rad);
   ct    := COS (theta*rad);
   sp    := SIN (psi*rad);
   cp    := COS (psi*rad);
   sphi  := SIN (phi*rad);
   cphi  := COS (phi*rad);
   xe[3]  := -1.0 * alt;
   b[1,1] := cp * ct;
   b[1,2] := sp * ct;
   b[1,3] := -1.0 * st;
   b[2,1] := -1.0 * sp * cphi + cp * st * sphi;
   b[2,2] := cp * cphi + sp * st * sphi;
   b[2,3] := ct * sphi;
   b[3,1] := sp * sphi + cp * st * cphi;
   b[3,2] := -1.0 * cp * sphi + sp * st * cphi;
   b[3,3] := ct * cphi;
```

```
      END {ASK} METHOD {ObjInit};

   TELL METHOD jump;

      BEGIN

         WHILE bigloop = 0

         WAIT DURATION dt;

         IF (-1.0*xe[3]) > hmin

            g := gravCalc(h);
            densityCalc (h, rhoz, rho, sound);

            IF cds = 690.0
               dt := 0.01;      { ----- ONCE CANOPY INFLATES, DECREASE TIME STEP SIZE TO
0.01 ----- }
            ELSE
               dt := 0.001;  { ----- OTHERWISE, START WITH A SMALLER STEP SIZE OF 0.001    -
---- }
            END IF;

            radius    := SQRT(cds/pi);
            addedmass := rho*(4.0/3.0)*pi*POWER(radius,3.0);
            distcm    := (32.17*addedmass*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                  + cweight*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                  +                    sweight*(0.5*slength*COS(angle)+dcglength))               /
(32.17*addedmass+weight+sweight+cweight);
            sysmass   := (weight+cweight+sweight)/32.17 + addedmass;
            paymom    := (1.0/12.0)*mass*(3.0*POWER((0.5*xcg),2.0) + POWER(xbod,2.0));
            distcan   := slength*COS(angle) + (4.0/3.0)*(radius/pi) - distcm;
            distline  := distcm - 0.5*slength*COS(angle);
            distpay   := distcm;

            in[1,1]                                                                          :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,2.0)
                  + (sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
            in[2,2]                                                                          :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,2.0)
                  + (sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
            in[3,3]              :=        ((2.0/5.0)*(cweight/32.17)*POWER(radius,2.0)        +
(2.0/3.0)*(rho*(4.0/3.0)*pi*POWER(radius,3.0)
                  * POWER(radius,2.0)) + (0.5*mass))/14.59;

            FOR i := 1 TO 3
               jn[i,i] := 1.0 / in [i,i];
            END FOR;

            IF (-1.0*xe[3]) <= ShearAlt1
               IF (-1.0*xe[3]) <= ShearAlt2
                  vwind[2] := vs3;
```

```
    ELSE
      vwind[2] := vs2;
    END IF;
  ELSE
    vwind[2] := vs1;
  END IF;

  pb   := wb[1];
  qb   := wb[2];
  rb   := wb[3];
  ue1  := ue[1] - vwind[1];
  ue2  := ue[2] - vwind[2];
  ue3  := ue[3] - vwind[3];
  vp   := SQRT(POWER(ue1,2.0) + POWER(ue2,2.0) + POWER(ue3,2.0));
  vpo  := vp;
  mach := vp/sound;
  ub1  := b[1,1]*ue1 + b[1,2]*ue2 + b[1,3]*ue3;
  ub2  := b[2,1]*ue1 + b[2,2]*ue2 + b[2,3]*ue3;
  ub3  := b[3,1]*ue1 + b[3,2]*ue2 + b[3,3]*ue3;
  vp13 := SQRT(POWER(ub1,2.0) + POWER(ub3,2.0));

{ USE SIN(ALPHA) for ALPHA and COS(BETA) for BETA }

  IF vpo < 1.0E-06
    vpo := 1.0E-06;
  END IF;

  sbeta := ub2 / vpo;
  cbeta := vp13 / vpo;
  beta  := sbeta;

  IF vp13 < 1.0E-06
    vp13 := 1.0E-06;
  END IF;

  salpha := ub3 / vp13;
  calpha := ub1 / vp13;
  alpha  := salpha;

{ AERODYNAMIC and BODY FORCES AND MOMENTS }

{ ISOLATED BODY AERODYNAMICS }

{ ----- BEGIN AERO ROUTINE ----- }
  sac  := salpha * calpha;
  sas  := salpha * ABS (salpha);
  sbc  := sbeta * cbeta;
  sbs  := sbeta * ABS(sbeta);
  rad  := cbar / (2.0*vpo);
  cna2 := 0.0;
  cnq  := 0.0;
  cyb2 := 0.0;
  cyr  := 0.0;
```

```
cmo  := 0.0;
cma2 := 0.0;
cnb2 := 0.0;

{ ----- FOREBODY AERO-LIFT DRAG COEFFIECIENT ----- }

i    := 0;
loop := 0;
WHILE loop = 0 { WILL LOOP WHEN mach > pm[ip] UNTIL i = mpts }
    i  := i + 1;
    ip := i + 1;
    IF i = mpts
        cao  := pcdf[2]; { when i = ipts }
        loop := 1;
    ELSE
        IF mach <= pm[ip]
            cao  := pcdf[i]+(pcdf[ip]-pcdf[i])*(mach-pm[i])/(pm[ip]-pm[i]);
            loop := 1;
        END IF;
    END IF;
    END WHILE;

cn  := cna * sac + cna2 * sas + cnq * qb * rad;
cy  := cyb + sbc + cyb2 * sbs + cyr * rb * rad;
ca  := cao + caa2 * (1.0 - POWER(calpha,2.0) * POWER(cbeta,2.0));
csl := clo + clp * pb * rad;
cm  := cmo + cma * sac + cma2 * sas + cmq * qb * rad;
csn := cnb * sbc + cnb2 * sbs + cnr * rb * rad;

{ ----- PARACHUTE DRAG-AREA ----- }

cds  := 0.0;
i    := 0;
loop := 0;

WHILE loop = 0
    IF t < pt[1]
        loop := 1;
    ELSE
        i  := i + 1;
        ip := i + 1;
        IF i = ipts
            cds  := pcds[ipts];
            loop := 1;
        ELSE
            IF t <= pt [ip]
                cds  := pcds[i] + (pcds[ip]-pcds[i]) * (t-pt[i]) / (pt[ip]-pt[i]);
                loop := 1;
            END IF;
        END IF;
    END IF;
END WHILE;
```

```
        { ----- END AERO ROUTINE ----- }

          q    := 0.5 * rho * vp;
          fpc  := -1.0 * q * cds;
          qs   := 0.5 * rho * POWER (vp, 2.0) * sarea;
          qsd  := qs * cbar;
          fb[1] := -1.0 * qs * ca + mass * g * b[1,3] + fpc * ub1;
          fb[2] := qs * cy + mass * g * b[2,3] + fpc * ub2;
          fb[3] := -1.0 * qs * cn + mass * g * b[3,3] + fpc * ub3;
          mb[1] := qsd * csl;
          mb[2] := qsd * cm + fpc * ub3 * (xbod - xcg);
          mb[3] := qsd * csn - fpc * ub2 * (xbod - xcg);
          gees  := -1.0 * fb[1] / (mass * g);

          IF ABS (gees) > ABS (gmax)
            gmax := gees;
          END IF;

          IF printTrajectory
             IF t >= tpr { THEN PRINT DATA }
               tpr := tpr + dtpr;
               h := -1.0 * xe[3];
               vpe := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0) );
               qdyn := 0.5 * rho * POWER (vp,2.0);
               bxy := SQRT ( POWER (b[1,1], 2.0) + POWER (b[1,2],2.0) );
               theta := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
               alphad := 57.295 * ATAN2 ( salpha, calpha );
               uxy := SQRT ( POWER (ue[1],2.0) + POWER (ue[2],2.0) );
               gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );
{
               OUTPUT ({myNumber, "L ", }SimTime, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ", vp, "
", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);
}
               ASK stream TO WriteString (INTTOSTR(myNumber) + "L ");
               ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
               ASK stream TO WriteString (REALTOSTR(h) + " ");
               ASK stream TO WriteString (REALTOSTR(xe[1]) + " ");
               ASK stream TO WriteString (REALTOSTR(xe[2]) + " ");
               ASK stream TO WriteString (REALTOSTR(vpe) + " ");
               ASK stream TO WriteString (REALTOSTR(vp) + " ");
               ASK stream TO WriteString (REALTOSTR(mach) + " ");
               ASK stream TO WriteString (REALTOSTR(qdyn) + " ");
               ASK stream TO WriteString (REALTOSTR(gees) + " ");
               ASK stream TO WriteString (REALTOSTR(gammad) + " ");
               ASK stream TO WriteString (REALTOSTR(theta) + " ");
               ASK stream TO WriteString (REALTOSTR(alphad) + " ");
               ASK stream TO WriteString (REALTOSTR(cds));
               ASK stream TO WriteLn;
            END IF;
          END IF;

          { EULER ROTATION FUNCTION FOR DIRECTION COSINE PROPOAGATION }
          w2     := POWER(wb[1],2.0) + POWER(wb[2],2.0) + POWER(wb[3],2.0);
          w      := SQRT (w2);
          coswt  := COS (w*dt);
          sinwt  := SIN (w*dt);
          coswtm := 1.0 - coswt;

          IF w2 < 1.0E-12
             w2 := 1.0E-12;
             w  := 1.0E-06;
          END IF;

          { ANGULAR MOMENTUM CROSS PRODUCT TERMS }

          FOR k := 1 TO 3
             hb [k] := in[k,1] * wb[1] + in[k,2] * wb[2] + in[k,3] * wb[3];
          END FOR;

          FOR i := 1 TO 3
              i1 := e[i+1];
              i2 := e[i+2];
              temp[i] := wb[i1] * hb[i2] - wb[i2] * hb[i1];
          END FOR;

          { ----- FORCE RESOLUTION TO EULER SYSTEM                    ----- }
          { ----- TRANSLATIONAL ACCELERATION AND DIRECTION COSINE ROTATION -----
}

          FOR i := 1 TO 3
             fe[i] := fb[1] * b[1,i] + fb[2] * b[2,i] + fb[3] * b[3,i];
             uedot[i] := fe[i] / mass;

             FOR j := 1 TO 3
                bn[i,j] := b[i,j];
                j1 := e[j+1];
                j2 := e[j+2];
                bdot[i,j]   :=   del[i,j]*coswt   +   wb[i]*wb[j]*coswtm/w2   +   (   wb[j1]*del[i,j2]   -
wb[j2]*del[i,j1] )* sinwt/w;
             END FOR;

             { ----- ANGULAR ACCELERATION IN BODY AXES ----- }

             wbdot[i] := jn[i,1]*(mb[1]-temp[1]) + jn[i,2]*(mb[2]-temp[2]) + jn[i,3]*(mb[3]-temp[3]);

          END FOR;

          { ----- INTEGRALS ----- }

          t := t + dt;
{         t := SimTime - myTime;}

          xlast := xe[1];

          FOR i := 1 TO 3
```

```
           xe[i] := xe[i] + dt*(ue[i]+0.5*dt*uedot[i]);
           ue[i] := ue[i] + dt*uedot[i];
           wb[i] := wb[i] + dt*wbdot[i];
           FOR j := 1 TO 3
              b[i,j] := bdot[i,1]*bn[1,j] + bdot[i,2]*bn[2,j] + bdot[i,3]*bn[3,j];
           END FOR;
        END FOR;

        { ----- INDUCE THE RANDOM BEHAVIOR OF THE T-10C TO TAKE INTO ACCOUNT
DRIFT ----- }
        { ----- DRIFT WILL ONLY BE INDUCED IF ALL OF THE FOLLOWING ARE MET        -
---- }
        { ----- 1) NO WIND                                  ----- }
        { ----- 2) PARACHUTE IS FULLY INFLATED                          ----- }

        IF MAXOF (vwind[1], vwind[2]) > 0.0
           addDrift := FALSE;
        ELSE
           addDrift := TRUE;
        END IF

        IF addDrift
           IF t >= tdrift
              tdrift := t + dtdrift;
              ASK SELF TO changeDrift;
           END IF;
        END IF;

        IF addDrift
           IF t >= 6.5
              xe[1] := xe[1] + Xdrift*dt;
              xe[2] := xe[2] + Ydrift*dt;
           END IF;
        END IF;

        IF t < 4.1
{                IF t > 1.4}
{                   xe[2] := xe[2] + ((xe[1]-xlast)*TAN(20.0*pi/180.0));}
                  xe[2] := xe[2] + 9.25/4100.0;
{                END IF;}
        END IF;

        { ----- UPDATING MOVING AND GROUND COORDINATE SYSTEMS ----- }

        xs[1] := Airdrop.Information[1].xg - xe[1];
        xs[2] := xe[2];
        xs[3] := xe[3];
        xg[1] := Airdrop.Information[1].xg - xs[1];
        xg[2] := xe[2];
        xg[3] := xe[3];
{
OUTPUT ("Lead Aircraft xg : ", Airdrop.Information[1].xg, " jumper ", myNumber, "L xs : ", xs[1]," xg :
", xg[1]);
```

```
}

                {POLL ALL VORTICES FOR MISSED DISTANCE}

                IF t >= tpoll
                    IF cds >= pcds[4]
                        tpoll := tpoll + dtpoll;
                        FOR i := 1 TO myPlane-1;
                            ASK SELF TO pollVortices (i);
                        END FOR;
                    END IF;
                END IF;

            ELSE { when -xe[3] <= hmin, THEN PRINT DATA FOR LAST TIME }

                h      := -1.0 * xe[3];
                vpe    := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0) );
                qdyn   := 0.5 * rho * POWER (vp,2.0);
                bxy    := SQRT ( POWER (b[1,1],2.0) + POWER (b[1,2],2.0) );
                theta  := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                alphad := 57.295 * ATAN2 ( salpha, calpha );
                uxy    := SQRT ( POWER(ue[1],2.0) + POWER(ue[2],2.0) );
                gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );

                OUTPUT (myNumber, "L ", myPlane, " ", SimTime, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ",
vp, " ", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);

                    ASK streamS TO WriteString (INTTOSTR(repeat) + " ");
                    ASK streamS TO WriteString (INTTOSTR(myNumber) + "L ");
                    ASK streamS TO WriteString (INTTOSTR(myPlane) + " ");
                    ASK streamS TO WriteString (REALTOSTR(SimTime) + " ");
                    ASK streamS TO WriteString (REALTOSTR(h) + " ");
                    ASK streamS TO WriteString (REALTOSTR(xe[1]) + " ");
                    ASK streamS TO WriteString (REALTOSTR(xe[2]) + " ");
                    ASK streamS TO WriteString (REALTOSTR(vpe) + " ");
                    ASK streamS TO WriteString (REALTOSTR(vp) + " ");
                    ASK streamS TO WriteString (REALTOSTR(mach) + " ");
                    ASK streamS TO WriteString (REALTOSTR(qdyn) + " ");
                    ASK streamS TO WriteString (REALTOSTR(gees) + " ");
                    ASK streamS TO WriteString (REALTOSTR(gammad) + " ");
                    ASK streamS TO WriteString (REALTOSTR(theta) + " ");
                    ASK streamS TO WriteString (REALTOSTR(alphad) + " ");
                    ASK streamS TO WriteString (REALTOSTR(cds));
                    ASK streamS TO WriteLn;

                IF printTrajectory
                        ASK stream TO WriteString (INTTOSTR(myNumber) + "L ");
                        ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
                        ASK stream TO WriteString (REALTOSTR(h) + " ");
                        ASK stream TO WriteString (REALTOSTR(xe[1]) + " ");
                        ASK stream TO WriteString (REALTOSTR(xe[2]) + " ");
                        ASK stream TO WriteString (REALTOSTR(vpe) + " ");
                        ASK stream TO WriteString (REALTOSTR(vp) + " ");
            ASK stream TO WriteString (REALTOSTR(mach) + " ");
            ASK stream TO WriteString (REALTOSTR(qdyn) + " ");
            ASK stream TO WriteString (REALTOSTR(gees) + " ");
            ASK stream TO WriteString (REALTOSTR(gammad) + " ");
            ASK stream TO WriteString (REALTOSTR(theta) + " ");
            ASK stream TO WriteString (REALTOSTR(alphad) + " ");
            ASK stream TO WriteString (REALTOSTR(cds));
            ASK stream TO WriteLn;
        END IF;

        bigloop := 1;

      END IF {hmin};

      END WAIT;

      END WHILE;

      IF printTrajectory
        ASK stream TO Close;
        DISPOSE (stream);
      END IF;

      DISPOSE (SELF);

    END {ASK} METHOD {jump};

  ASK METHOD initialize (IN stick   : INTEGER;
           IN Counter : INTEGER);

    BEGIN

      myPlane          := Counter;
      myNumber          := stick;
      gees            := 0.0;
      cds             := 0.0;
      myTime           := SimTime;
      t               := SimTime-myTime;
      gmax             := 0.0;
      tpr              := t;
      tpoll            := t;
      tdrift           := t + dtdrift;
      alt              := -1.0 * xe[3];
      h                := alt;
      bigloop           := 0;
      myDrift           := seed1.UniformReal (0.0, 4.0);
{        myDrift           := seed1.Normal (2.0, 0.5);}
      myDriftDirection  := seed2.UniformReal (0.0, 360.0);
{        myDriftDirection  := seed2.Normal (0.0, 2.8125);}
      xe[1]            := Airdrop.Information[myPlane].xg;
      xlast            := xe[1];
      xe[2]            := Airdrop.Information[myPlane].yg - 9.25;
      alt              := Airdrop.Information[myPlane].altitude;
```

```
         xe[3]          := -1.0 * alt;
         ue[1]          := Airdrop.Information[myPlane].vf;
         vwind[1]          := HeadWind;

         NEW (lastRightLocation, 1..myPlane-1);
         NEW (lastLeftLocation,  1..myPlane-1);

         FOR i := 1 TO myPlane-1
            lastRightLocation[i] := 1;
            lastLeftLocation[i]  := 1;
         END FOR;

         ASK SELF TO findDrift;

         IF printTrajectory
            NEW (stream);
            outfile := "LJ" + INTTOSTR(myPlane) + INTTOSTR(myNumber) + extension + ".mat";
            ASK stream TO Open (outfile, Output);
         END IF;
{
OUTPUT ();
OUTPUT ("My airplane    : ", myPlane);
OUTPUT ("Exit time      : ", SimTime);
OUTPUT ("My number      : ", stick, "L");
OUTPUT ("My weight      : ", weight);
OUTPUT ("Xs             : ", xe[1]);
OUTPUT ("Ys             : ", xe[2]);
OUTPUT ("altitude       : ", alt);
}
ASK streamI TO WriteString (INTTOSTR(repeat) + " ");
ASK streamI TO WriteString (INTTOSTR(myPlane) + " ");
ASK streamI TO WriteString (REALTOSTR(SimTime) + " ");
ASK streamI TO WriteString (INTTOSTR(stick) + "L ");
ASK streamI TO WriteString (REALTOSTR(weight) + " ");
ASK streamI TO WriteString (REALTOSTR(xe[1]) + " ");
ASK streamI TO WriteString (REALTOSTR(xe[2]) + " ");
ASK streamI TO WriteString (REALTOSTR(alt) + " ");
ASK streamI TO WriteLn;

      END METHOD {initialize};

   ASK METHOD pollVortices (IN vortexPlane : INTEGER);

      VAR

         x, xcord1, xcord2, vvx, vjx      : REAL;
         y, ycord1, ycord2, vvy, vjy      : REAL;
         z, zcord1, zcord2, vvz, vjz      : REAL;
         vjdistance, vvdistance, distance : REAL;
         projection                       : REAL;
         i, location                      : INTEGER;
         check                            : BOOLEAN;
         startRightSearch                 : INTEGER;
```

```
        startLeftSearch                 : INTEGER;

      BEGIN

        check := FALSE;
        location := 0;

        startRightSearch := lastRightLocation[vortexPlane];
{
OUTPUT ("Current search is of plane    ", currentRightVortex);
OUTPUT ("Start searching from position ", startRightSearch);
}
        {POLL RIGHT VORTEX}

        FOR i := startRightSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
          IF location = 0
            IF    (ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].xCord-
xs[1])) <= 50.0
              lastRightLocation[vortexPlane] := i;
              IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].zCord+xs[3])) <= 50.0
                lastRightLocation[vortexPlane] := i;
                IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].yCord-xs[2])) <= 50.0
                  check      := TRUE;
                  location    := i;
                  IF location = Airdrop.Information[vortexPlane].NumberOfSteps
                    check   := FALSE;
                  END IF;
                  lastRightLocation[vortexPlane] := i;
                ELSE
                  location := i;
                END IF;
              ELSE
                location := i;
              END IF;
            END IF;
          ELSE
            EXIT;
          END IF;
        END FOR;

        IF check
          xcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].xCord;
          ycord1                                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].yCord;
          zcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].zCord;
          xcord2                                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].xCord;
          ycord2                                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].yCord;
```

```
            zcord2                                                              :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].zCord;

            vvx := xcord2-xcord1;
            vvy := ycord2-ycord1;
            vvz := zcord2-zcord1;

            vjx := xs[1]-xcord1;
            vjy := xs[2]-ycord1;
            vjz := -1.0*xs[3]-zcord1;

            vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
            vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
            projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
            distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
            IF                               distance                            <=
MAXOF(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].radius)

              OUTPUT (myNumber, "L ", myPlane, " RV ", vortexPlane, " ", -1.0*xe[3], " ", distance, "
", location, " ", SimTime);

              ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
              ASK streamE TO WriteString (INTTOSTR(myNumber) + "L ");
              ASK streamE TO WriteString (INTTOSTR(myPlane) + " RV ");
              ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
              ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
              ASK streamE TO WriteString (REALTOSTR(distance) + " ");
              ASK streamE TO WriteString (REALTOSTR(location) + " ");
              ASK streamE TO WriteString (REALTOSTR(SimTime));
              ASK streamE TO WriteLn;

            ELSE
{
              OUTPUT (myNumber, "L no encounter with right vortex of C-17 number", vortexPlane, "
at distance ", distance);
}
            END IF;
          ELSE
{
            OUTPUT (myNumber, "L no encounter with right vortex of C-17 number", vortexPlane);
}
          END IF;

          check := FALSE;
          location := 0;

          startLeftSearch := lastLeftLocation[vortexPlane];
{
OUTPUT ("Current search is of plane   ", currentLeftVortex);
OUTPUT ("Start searching from position ", startLeftSearch);
}
          {POLL LEFT VORTEX}
```

```
         FOR i := startLeftSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
            IF location = 0
               IF     (ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].xCord-
xs[1])) <= 50.0
                  lastLeftLocation[vortexPlane] := i;
                  IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].zCord+xs[3])) <= 50.0
                     lastRightLocation[vortexPlane] := i;
                     IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].yCord-xs[2])) <= 50.0
                        check      := TRUE;
                        location   := i;
                        IF location = Airdrop.Information[vortexPlane].NumberOfSteps
                           check  := FALSE;
                        END IF;
                        lastRightLocation[vortexPlane] := i;
                     ELSE
                        location := i;
                     END IF;
                  ELSE
                     location := i;
                  END IF;
               END IF;
            ELSE
               EXIT;
            END IF;
         END FOR;

         IF check
            xcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].xCord;
            ycord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].yCord;
            zcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].zCord;
            xcord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].xCord;
            ycord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].yCord;
            zcord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].zCord;

            vvx := xcord2-xcord1;
            vvy := ycord2-ycord1;
            vvz := zcord2-zcord1;

            vjx := xs[1]-xcord1;
            vjy := xs[2]-ycord1;
            vjz := -1.0*xs[3]-zcord1;

            vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
            vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
```

```
        projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
        distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
        IF                              distance                                    <=
MAXOF(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].radius)

          OUTPUT (myNumber, "L ", myPlane, " LV ", vortexPlane, " ", -1.0*xe[3], " ", distance, " ",
location, " ", SimTime);

          ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
          ASK streamE TO WriteString (INTTOSTR(myNumber) + "L ");
          ASK streamE TO WriteString (INTTOSTR(myPlane) + " LV ");
          ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
          ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
          ASK streamE TO WriteString (REALTOSTR(distance) + " ");
          ASK streamE TO WriteString (REALTOSTR(location) + " ");
          ASK streamE TO WriteString (REALTOSTR(SimTime));
          ASK streamE TO WriteLn;

        ELSE
{
          OUTPUT (myNumber, "L no encounter with left vortex of C-17 number", vortexPlane, " at
distance ", distance);
}
        END IF;
      ELSE
{
        OUTPUT (myNumber, "L no encounter with left vortex of C-17 number", vortexPlane);
}
      END IF;

    END METHOD {pollVortices};

  ASK METHOD changeDrift;
    BEGIN
      myDrift        := seed1.UniformReal (0.0, 4.0);
{       myDrift        := seed1.Normal (2.0, 0.5);}
      myDriftDirection := seed2.Normal (myDriftDirection, 2.8125);
      ASK SELF TO findDrift;
    END {ASK} METHOD {changeDrift};

  ASK METHOD findDrift;
    BEGIN
      Xdrift := myDrift * COS (myDriftDirection*pi/180.0);
      Ydrift := myDrift * SIN (myDriftDirection*pi/180.0);
    END {ASK} METHOD {findDrift};

END OBJECT {leftJumperObj};

END {IMPLEMENTATION} MODULE {leftJumperMod}.
```

Definition Right Jumper Module

```
DEFINITION MODULE rightJumperMod;

FROM IOMod          IMPORT StreamObj, FileUSeType(Output);
FROM globalMod       IMPORT eType, matrixType, vectorType, encounterType;
FROM VortexMod       IMPORT RightVortexObj, LeftVortexObj;
FROM VortexControlMod IMPORT VortexControl;

TYPE

   rightJumperObj = OBJECT

      i1,
      i2,
      iend,
      ipts,
      ip,
      j1,
      j2,
      k,
      loop,
      mpts,
      myNumber,
      myPlane,
      bigloop : INTEGER;

      alt,
      alpha,
      alphad,
      beta,
      bxy,
      ca,
      cao,
      caa2,
      calpha,
      cbar,
      cbeta,
      cby2,
      cds,
      clo,
      clp,
      cm,
      cma,
      cma2,
      cmo,
      cmq,
      cn,
      cna,
      cna2,
cnb,
cnb2,
cnq,
cnr,
coswt,
coswtm,
cp,
cphi,
csl,
csn,
ct,
cy,
cyb,
cyb2,
cyr,
dens,
deptime,
dt,
dtpr,
dtpoll,
dtdrift,
fpc,
g,
gammad,
gees,
gmax,
h,
hmin,
mach,
mass,
myTime,
myDrift,
myDriftDirection,
pb,
phi,
psi,
q,
qb,
qdyn,
qs,
qsd,
rad,
rb,
rho,
rhoz,
sac,
sas,
sarea,
salpha,
sbc,
sbeta,
sbs,
sinwt,
sound,
sp,
sphi,
st,
t,
theta,
tpr,
tpoll,
tdrift,
ub1,
ub2,
ub3,
ue1,
ue2,
ue3,
uxy,
vp,
vp13,
vpe,
vpo,
w,
w2,
weight,
xbod,
xcg,
Xdrift,
xlast,
Ydrift    : REAL;

slength   : REAL; {length of suspension lines}
angle     : REAL; {the angle (in radians) which defines the "cone" of the suspension lines}
dcglength : REAL; {distance from end of suspension lines to paratrooper c.g.}
cweight   : REAL; {weight of canopy}
sweight   : REAL; {weight of suspension lines}

radius    : REAL;
addedmass : REAL;
distcm    : REAL;
sysmass   : REAL;
paymom    : REAL;
distcan   : REAL;
distline  : REAL;
distpay   : REAL;

addDrift  : BOOLEAN;

pcds,
pt,
pcdf,
pm    : vectorType; {1X2}

xe,
xs,
    xg,
    ue,
    wb,
    vwind,
    temp,
    fb,
    m,
    mb,
    fe,
    uedot,
    wbdot,
    hb    : vectorType; {1X3}

    in,
    jn,
    b,
    bn,
    bdot : matrixType; {3X3}

    lastRightLocation,
    lastLeftLocation : eType; {Dynamic Array}

    outfile : STRING;

    stream : StreamObj;

    encounter   : encounterType;

    ASK METHOD ObjInit;
    TELL METHOD jump;
    ASK METHOD initialize (IN stick   : INTEGER;
                  IN myPlane : INTEGER);
    ASK METHOD pollVortices (IN vortexPlane : INTEGER);
    ASK METHOD changeDrift;
    ASK METHOD findDrift;

  END OBJECT {rightJumperObj};

END {DEFINITION} MODULE {rightJumperMod}.
```

<u>Implementation Right Jumper Module</u>

```
IMPLEMENTATION MODULE rightJumperMod;

FROM MathMod        IMPORT EXP, SIN, COS, POWER, SQRT, ATAN2, TAN, pi;
FROM VortexMod      IMPORT RightVortexObj, LeftVortexObj;
FROM globalMod      IMPORT re, e, del, i, j, NumberofPlanes, seed1, seed2, seed3, repeat;
FROM inputMod       IMPORT streamI, streamE, streamS, extension, printTrajectory;
FROM globalMod           IMPORT CrossWind1, CrossWind2, CrossWind3, ShearAlt1, ShearAlt2,
HeadWind, vs1, vs2, vs3;
FROM calcMod        IMPORT gravCalc, densityCalc;
FROM SimMod         IMPORT SimTime;
FROM VortexControlMod IMPORT VortexControl, Airdrop;

{TROUBLE SHOOTING INPUT}
{
FROM inputMod IMPORT dtdrift;
}
OBJECT rightJumperObj;

  ASK METHOD ObjInit;

    BEGIN
      NEW(pcdf, 1..2);
      NEW(pm  , 1..2);

      NEW(fb  , 1..3);
      NEW(fe  , 1..3);
      NEW(hb  , 1..3);
      NEW(mb  , 1..3);
      NEW(temp , 1..3);
      NEW(ue  , 1..3);
      NEW(uedot, 1..3);
      NEW(vwind, 1..3);
      NEW(wb  , 1..3);
      NEW(wbdot, 1..3);
      NEW(xe  , 1..3);
      NEW(xs  , 1..3);
      NEW(xg  , 1..3);

      NEW(pcds, 1..4);
      NEW(pt  , 1..4);

      NEW(in  , 1..3, 1..3);
      NEW(jn  , 1..3, 1..3);
      NEW(b   , 1..3, 1..3);
      NEW(bn  , 1..3, 1..3);
      NEW(bdot, 1..3, 1..3);

    { system inertial properties }
    { parachute-payload system weight (lbs) = wight of jumper/gear + weight of T-10C }
    weight := seed3.Normal (250.0, 25.0);

    mass   := weight/32.17;
    xcg    := 0.5;        { forebody c.g. (ft) in the horizontal }
    xbod   := 6.0;        { forebody length (ft) in the vertical }

    FOR i := 1 TO 3
      FOR j := 1 TO 3
        jn[i,j] := 0.0;
        in[i,j] := 0.0;
      END {j} FOR;
    END {i} FOR;

  { initial conditions }

    FOR i := 1 TO 3
      xe[i]    := 0.0; { (ft)  1: down range, 2: off range, 3: altitude loss                    }
      ue[i]     := 0.0; { (fps) 1: horizontal velocity, 2: lateral velocity, 3: ejection velocity positive
down }
      wb[i]    := 0.0; { ???                                                   }
      vwind[i] := 0.0; { (fps) 1: head (+) or tail (-) wind, 2: crosswind, 3: ????                  }
    END FOR;

    alt       := 0.0;     { altitude (ft)                    }
    hmin      := 0.0;     { ground level (ft)                  }
    ue[1]     := 0.0;     { horizontal velocity (fps)              }
    ue[3]     := 0.0;     { ejection velocity, positive down         }
    theta     := 0.0;     { pitch angle (deg) nose up positive       }
    vwind[1]  := 0.0;     { head (+) or tail (-) wind (fps)          }
    vwind[2]  := 0.0;     { crosswind (fps)                      }
    dens      := 0.0;     { density (0 for standard atms) in slug/ft^3 }
    rhoz      := 0.002378; { ????                           }

    IF dens <> 0.0
      rhoz := dens * EXP (alt/23111.0 - 0.295 * SIN(alt/28860.0) - 0.213 * SIN(alt/86580.0))
    END IF;

  { program constants }

    dtpr    := 0.1;   { print interval (sec)                              }
    dtpoll  := 0.5;   { poll vortex positions every 0.5 seconds                    }
    dtdrift := 5.0;   { change drift angle +/- 45.0 from current drift angle every 10 seconds }

  { forebody aerodynamic coefficients }

    cbar  := 6.0;             { reference length (ft)              }
    sarea := POWER(0.5,2.0)*pi; { reference area (ft^2)                 }
    cna   := 0.0;             { normal force cn-alpha (/rad)          }
    cyb   := 0.0;             { side force cy-beta (/rad)           }
    caa2  := 0.0;             { axial force ca-alpha^2 (/rad^2)       }
    clo   := 0.0;             { roll torque coefficient (dimensionless) }
    clp  := 0.0;          { roll damping coefficient (/rad)      }
    cma  := -2.0;          { pitch moment cm-alpha (/rad)          }
    cmq  := -200.0;         { pitch damping (/rad)                }
    cnb  := 0.0;          { yaw moment cn-beta (/rad)             }
    cnr  := 0.0;          { yaw damping (/rad)                 }

{ forebody drag versus mach number table }

    mpts   := 2;
    pm[1]  := 0.00; { mach number }
    pm[2]  := 2.00;
    pcdf[1] := 0.73+0.06*(360.0 - weight)/180.0; { drag coefficient }
    pcdf[2] := 0.73+0.06*(360.0 - weight)/180.0;

{ parachute drag-area versus time table }

    deptime := 0.25; { deployment time }
    ipts   := 2;

    IF deptime > 0.0
       ipts := ipts + 2; { ipts = 4 }
       pt[1]  := 0.0;
       pt[2]  := deptime;
       pt[3]  := 0.00 + deptime;
       pt[4]  := 2.80 + deptime;
       pcds[1] := 0.0;
       pcds[2] := 0.0;
       pcds[3] := 0.20;
       pcds[4] := 690.0;
    END IF;

{ convert EULER ANGLES to direction cosines }

    psi   := 0.0;
    phi   := 0.0;
    rad   := pi/180.0;
    st    := SIN (theta*rad);
    ct    := COS (theta*rad);
    sp    := SIN (psi*rad);
    cp    := COS (psi*rad);
    sphi  := SIN (phi*rad);
    cphi  := COS (phi*rad);
    xe[3]  := -1.0 * alt;
    b[1,1] := cp * ct;
    b[1,2] := sp * ct;
    b[1,3] := -1.0 * st;
    b[2,1] := -1.0 * sp * cphi + cp * st * sphi;
    b[2,2] := cp * cphi + sp * st * sphi;
    b[2,3] := ct * sphi;
    b[3,1] := sp * sphi + cp * st * cphi;
    b[3,2] := -1.0 * cp * sphi + sp * st * cphi;
    b[3,3] := ct * cphi;
```

```
      END {ASK} METHOD {ObjInit};

   TELL METHOD jump;

      BEGIN

         WHILE bigloop = 0

         WAIT DURATION dt;

         IF (-1.0*xe[3]) > hmin

            g := gravCalc(h);
            densityCalc (h, rhoz, rho, sound);

            IF cds = 690.0
               dt := 0.01;     { ----- ONCE CANOPY INFLATES, DECREASE TIME STEP SIZE TO
0.01 ----- }
            ELSE
               dt := 0.001;   { ----- OTHERWISE, START WITH A SMALLER STEP SIZE          -----
}
            END IF;

            radius    := SQRT(cds/pi);
            addedmass := rho*(4.0/3.0)*pi*POWER(radius,3.0);
            distcm    := (32.17*addedmass*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                   + cweight*(slength*COS(angle)+(4.0/3.0)*(radius/pi)+dcglength)
                   +                    sweight*(0.5*slength*COS(angle)+dcglength))              /
(32.17*addedmass+weight+sweight+cweight);
            sysmass   := (weight+cweight+sweight)/32.17 + addedmass;
            paymom    := (1.0/12.0)*mass*(3.0*POWER((0.5*xcg),2.0) + POWER(xbod,2.0));
            distcan   := slength*COS(angle) + (4.0/3.0)*(radius/pi) - distcm;
            distline  := distcm - 0.5*slength*COS(angle);
            distpay   := distcm;

            in[1,1]                                                                          :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,2.0)
                   + (sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
            in[2,2]                                                                          :=
(addedmass*((2.0/5.0)*POWER(radius,2.0)+POWER(distcan,2.0))+(cweight/32.17)*POWER(distcan,2.0)
                   + (sweight/32.17)*POWER(distline,2.0)+paymom+mass*POWER(distpay,2.0))/14.59;
            in[3,3]                 :=        ((2.0/5.0)*(cweight/32.17)*POWER(radius,2.0)         +
(2.0/3.0)*(rho*(4.0/3.0)*pi*POWER(radius,3.0)
                   * POWER(radius,2.0)) + (0.5*mass))/14.59;

            FOR i := 1 TO 3
               jn[i,i] := 1.0 / in [i,i];
            END FOR;

            IF (-1.0*xe[3]) <= ShearAlt1
               IF (-1.0*xe[3]) <= ShearAlt2
                  vwind[2] := vs3;
               ELSE
         vwind[2] := vs2;
      END IF;
   ELSE
      vwind[2] := vs1;
   END IF;

   pb   := wb[1];
   qb   := wb[2];
   rb   := wb[3];
   ue1  := ue[1] - vwind[1];
   ue2  := ue[2] - vwind[2];
   ue3  := ue[3] - vwind[3];
   vp   := SQRT(POWER(ue1,2.0) + POWER(ue2,2.0) + POWER(ue3,2.0));
   vpo  := vp;
   mach := vp/sound;
   ub1  := b[1,1]*ue1 + b[1,2]*ue2 + b[1,3]*ue3;
   ub2  := b[2,1]*ue1 + b[2,2]*ue2 + b[2,3]*ue3;
   ub3  := b[3,1]*ue1 + b[3,2]*ue2 + b[3,3]*ue3;
   vp13 := SQRT(POWER(ub1,2.0) + POWER(ub3,2.0));

{ ----- USE SIN(ALPHA) for ALPHA and COS(BETA) for BETA ----- }

   IF vpo < 1.0E-06
      vpo := 1.0E-06;
   END IF;

   sbeta := ub2 / vpo;
   cbeta := vp13 / vpo;
   beta  := sbeta;

   IF vp13 < 1.0E-06
     vp13 := 1.0E-06;
   END IF;

   salpha := ub3 / vp13;
   calpha := ub1 / vp13;
   alpha  := salpha;

{ ----- AERODYNAMIC and BODY FORCES AND MOMENTS ----- }

{ ----- ISOLATED BODY AERODYNAMICS ----- }

{ ----- BEGIN AERO ROUTINE ----- }

   sac  := salpha * calpha;
   sas  := salpha * ABS (salpha);
   sbc  := sbeta * cbeta;
   sbs  := sbeta * ABS(sbeta);
   rad  := cbar / (2.0*vpo);
   cna2 := 0.0;
   cnq  := 0.0;
   cyb2 := 0.0;
   cyr  := 0.0;
```

```
cmo  := 0.0;
cma2 := 0.0;
cnb2 := 0.0;

{ ----- FOREBODY AERO-LIFT DRAG COEFFIECIENT ----- }

i   := 0;
loop := 0;
WHILE loop = 0 {WILL LOOP WHEN mach > pm[ip] UNTIL i = mpts}
   i  := i + 1;
   ip := i + 1;
   IF i = mpts
      cao  := pcdf[2]; { when i = ipts }
      loop := 1;
   ELSE
      IF mach <= pm[ip]
         cao  := pcdf[i]+(pcdf[ip]-pcdf[i])*(mach-pm[i])/(pm[ip]-pm[i]);
         loop := 1;
      END IF;
   END IF;
   END WHILE;

cn  := cna * sac + cna2 * sas + cnq * qb * rad;
cy  := cyb + sbc + cyb2 * sbs + cyr * rb * rad;
ca  := cao + caa2 * (1.0 - POWER(calpha,2.0) * POWER(cbeta,2.0));
csl := clo + clp * pb * rad;
cm  := cmo + cma * sac + cma2 * sas + cmq * qb * rad;
csn := cnb * sbc + cnb2 * sbs + cnr * rb * rad;

{ ----- PARACHUTE DRAG-AREA ----- }

cds  := 0.0;
i    := 0;
loop := 0;

{ ----- THIS LOOP INFLATES THE PARACHUTE ----- }

WHILE loop = 0
   IF t < pt[1]
      loop := 1;
   ELSE
      i  := i + 1;
      ip := i + 1;
      IF i = ipts
         cds  := pcds[ipts];
         loop := 1;
      ELSE
         IF t <= pt [ip]
            cds  := pcds[i] + (pcds[ip]-pcds[i]) * (t-pt[i]) / (pt[ip]-pt[i]);
            loop := 1;
         END IF;
      END IF;
   END IF;
```

```
                  END WHILE;

              { ----- END AERO ROUTINE ----- }

                  q    := 0.5 * rho * vp;
                  fpc  := -1.0 * q * cds;
                  qs   := 0.5 * rho * POWER (vp, 2.0) * sarea;
                  qsd  := qs * cbar;
                  fb[1] := -1.0 * qs * ca + mass * g * b[1,3] + fpc * ub1;
                  fb[2] := qs * cy + mass * g * b[2,3] + fpc * ub2;
                  fb[3] := -1.0 * qs * cn + mass * g * b[3,3] + fpc * ub3;
                  mb[1] := qsd * csl;
                  mb[2] := qsd * cm + fpc * ub3 * (xbod - xcg);
                  mb[3] := qsd * csn - fpc * ub2 * (xbod - xcg);
                  gees  := -1.0 * fb[1] / (mass * g);

                  IF ABS (gees) > ABS (gmax)
                     gmax := gees;
                  END IF;

                  IF printTrajectory
                      IF t >= tpr { THEN PRINT DATA }
                          tpr := tpr + dtpr;
                          h := -1.0 * xe[3];
                          vpe := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0) );
                          qdyn := 0.5 * rho * POWER (vp,2.0);
                          bxy := SQRT ( POWER (b[1,1], 2.0) + POWER (b[1,2],2.0) );
                          theta := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                          alphad := 57.295 * ATAN2 ( salpha, calpha );
                          uxy := SQRT ( POWER (ue[1],2.0) + POWER (ue[2],2.0) );
                          gammad := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );
{
                          OUTPUT (myNumber, "R ", SimTime, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ", vp, " ",
mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);
}
                          ASK stream TO WriteString (INTTOSTR(myNumber) + "R ");
                          ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
                          ASK stream TO WriteString (REALTOSTR(h) + " ");
                          ASK stream TO WriteString (REALTOSTR(xe[1]) + " ");
                          ASK stream TO WriteString (REALTOSTR(xe[2]) + " ");
                          ASK stream TO WriteString (REALTOSTR(vpe) + " ");
                          ASK stream TO WriteString (REALTOSTR(vp) + " ");
                          ASK stream TO WriteString (REALTOSTR(mach) + " ");
                          ASK stream TO WriteString (REALTOSTR(qdyn) + " ");
                          ASK stream TO WriteString (REALTOSTR(gees) + " ");
                          ASK stream TO WriteString (REALTOSTR(gammad) + " ");
                          ASK stream TO WriteString (REALTOSTR(theta) + " ");
                          ASK stream TO WriteString (REALTOSTR(alphad) + " ");
                          ASK stream TO WriteString (REALTOSTR(cds));
                          ASK stream TO WriteLn;

                    END IF;
                END IF;
```

```
            { EULER ROTATION FUNCTION FOR DIRECTION COSINE PROPOAGATION }

            w2 := POWER(wb[1],2.0) + POWER(wb[2],2.0) + POWER(wb[3],2.0);
            w := SQRT (w2);
            coswt := COS (w*dt);
            sinwt := SIN (w*dt);
            coswtm := 1.0 - coswt;

            IF w2 < 1.0E-12
              w2 := 1.0E-12;
              w := 1.0E-06;
            END IF;

            { ANGULAR MOMENTUM CROSS PRODUCT TERMS }

            FOR k := 1 TO 3
              hb [k] := in[k,1] * wb[1] + in[k,2] * wb[2] + in[k,3] * wb[3];
            END FOR;

            FOR i := 1 TO 3
              i1 := e[i+1];
              i2 := e[i+2];
              temp[i] := wb[i1] * hb[i2] - wb[i2] * hb[i1];
            END FOR;

            { FORCE RESOLUTION TO EULER SYSTEM                    }
            { TRANSLATIONAL ACCELERATION AND DIRECTION COSINE ROTATION }

            FOR i := 1 TO 3
              fe[i] := fb[1] * b[1,i] + fb[2] * b[2,i] + fb[3] * b[3,i];
              uedot[i] := fe[i] / mass;

              FOR j := 1 TO 3
                bn[i,j] := b[i,j];
                j1 := e[j+1];
                j2 := e[j+2];
                bdot[i,j]  :=  del[i,j]*coswt  +  wb[i]*wb[j]*coswtm/w2  +  (  wb[j1]*del[i,j2]  -
wb[j2]*del[i,j1] )* sinwt/w;
              END FOR;

              { ANGULAR ACCELERATION IN BODY AXES }

              wbdot[i] := jn[i,1]*(mb[1]-temp[1]) + jn[i,2]*(mb[2]-temp[2]) + jn[i,3]*(mb[3]-temp[3]);

            END FOR;

            { ----- INTEGRALS ----- }

            t := t + dt;
{           t := SimTime - myTime;}

            xlast := xe[1];
```

```
          FOR i := 1 TO 3
             xe[i] := xe[i] + dt*(ue[i]+0.5*dt*uedot[i]);
             ue[i] := ue[i] + dt*uedot[i];
             wb[i] := wb[i] + dt*wbdot[i];
             FOR j := 1 TO 3
                b[i,j] := bdot[i,1]*bn[1,j] + bdot[i,2]*bn[2,j] + bdot[i,3]*bn[3,j];
             END FOR;
          END FOR;

          { ----- INDUCE A DRIFT DIRECTION AND VELOCITY ON THE PARATROOP     ----- }
          { ----- DRIFT WILL ONLY BE INDUCED IF ALL OF THE FOLLOWING ARE MET ----- }
          { ----- 1) NO WIND                                         ----- }
          { ----- 2) PARACHUTE IS FULLY INFLATED AND AFTER SECOND VERTICLE  ----- }

          IF ABS(MAXOF (vwind[1], vwind[2])) > 0.0
             addDrift := FALSE;
          ELSE
             addDrift := TRUE;
          END IF

          IF addDrift
             IF t >= tdrift
                tdrift := t + dtdrift;
                ASK SELF TO changeDrift;
             END IF;
          END IF;

          IF addDrift
             IF t >= 6.5
                xe[1] := xe[1] + Xdrift*dt;
                xe[2] := xe[2] + Ydrift*dt;
             END IF;
          END IF;

          IF t < 4.1
                xe[2] := xe[2] - 9.25/4100.0;
          END IF;

          { ----- UPDATING MOVING AND GROUND COORDINATE SYSTEMS ----- }

          xs[1] := Airdrop.Information[1].xg - xe[1];
          xs[2] := xe[2];
          xs[3] := xe[3];
          xg[1] := Airdrop.Information[1].xg - xs[1];
          xg[2] := xe[2];
          xg[3] := xe[3];
{
OUTPUT ("Lead Aircraft xg : ", Airdrop.Information[1].xg, " jumper ", myNumber, "R xs : ", xs[1]," xg :
", xg[1]);
}

          { ----- POLL ALL VORTICES FOR MISSED DISTANCE ----- }
```

```
                IF t >= tpoll
                    IF cds >= pcds[4]
                        tpoll := tpoll + dtpoll;
                        FOR i := 1 TO myPlane-1;
                            ASK SELF TO pollVortices (i);
                        END FOR;
                    END IF;
                END IF;

            ELSE { ----- when -xe[3] <= hmin, THEN PRINT DATA FOR LAST TIME ----- }

                h       := -1.0 * xe[3];
                vpe     := SQRT ( POWER (ue1,2.0) + POWER (ue2,2.0) + POWER (ue3,2.0) );
                qdyn    := 0.5 * rho * POWER (vp,2.0);
                bxy     := SQRT ( POWER (b[1,1],2.0) + POWER (b[1,2],2.0) );
                theta   := 57.295 * ATAN2 ( (-1.0 * b[1,3]), bxy );
                alphad  := 57.295 * ATAN2 ( salpha, calpha );
                uxy     := SQRT ( POWER(ue[1],2.0) + POWER(ue[2],2.0) );
                gammad  := 57.295 * ATAN2 ( (-1.0 * ue[3]), uxy );

                OUTPUT (myNumber, "R ", myPlane, " ", SimTime, " ", h, " ", xe[1], " ", xe[2], " ", vpe, " ",
vp, " ", mach, " ", qdyn, " ", gees, " ", gammad, " ", theta, " ", alphad, " ", cds);

                   ASK streamS TO WriteString (INTTOSTR(repeat)   + " ");
                   ASK streamS TO WriteString (INTTOSTR(myNumber) + "R ");
                   ASK streamS TO WriteString (INTTOSTR(myPlane)  + " ");
                   ASK streamS TO WriteString (REALTOSTR(SimTime) + " ");
                   ASK streamS TO WriteString (REALTOSTR(h)       + " ");
                   ASK streamS TO WriteString (REALTOSTR(xe[1])   + " ");
                   ASK streamS TO WriteString (REALTOSTR(xe[2])   + " ");
                   ASK streamS TO WriteString (REALTOSTR(vpe)     + " ");
                   ASK streamS TO WriteString (REALTOSTR(vp)      + " ");
                   ASK streamS TO WriteString (REALTOSTR(mach)    + " ");
                   ASK streamS TO WriteString (REALTOSTR(qdyn)    + " ");
                   ASK streamS TO WriteString (REALTOSTR(gees)    + " ");
                   ASK streamS TO WriteString (REALTOSTR(gammad)  + " ");
                   ASK streamS TO WriteString (REALTOSTR(theta)   + " ");
                   ASK streamS TO WriteString (REALTOSTR(alphad)  + " ");
                   ASK streamS TO WriteString (REALTOSTR(cds));
                   ASK streamS TO WriteLn;

               IF printTrajectory

                      ASK stream TO WriteString (INTTOSTR(myNumber) + "R ");
                      ASK stream TO WriteString (REALTOSTR(SimTime) + " ");
                      ASK stream TO WriteString (REALTOSTR(h)       + " ");
                      ASK stream TO WriteString (REALTOSTR(xe[1])   + " ");
                      ASK stream TO WriteString (REALTOSTR(xe[2])   + " ");
                      ASK stream TO WriteString (REALTOSTR(vpe)     + " ");
                      ASK stream TO WriteString (REALTOSTR(vp)      + " ");
                      ASK stream TO WriteString (REALTOSTR(mach)    + " ");
                      ASK stream TO WriteString (REALTOSTR(qdyn)    + " ");
            ASK stream TO WriteString (REALTOSTR(gees)   + " ");
            ASK stream TO WriteString (REALTOSTR(gammad)  + " ");
            ASK stream TO WriteString (REALTOSTR(theta)  + " ");
            ASK stream TO WriteString (REALTOSTR(alphad)  + " ");
            ASK stream TO WriteString (REALTOSTR(cds));
            ASK stream TO WriteLn;

        END IF;

        bigloop := 1;

      END IF {hmin};

      END WAIT;

      END WHILE;

      IF printTrajectory
         ASK stream TO Close;
         DISPOSE (stream);
      END IF;

      DISPOSE (SELF);

    END {ASK} METHOD {jump};

  ASK METHOD initialize (IN stick   : INTEGER;
           IN Counter : INTEGER);

    BEGIN

      myPlane          := Counter;
      myNumber          := stick;
      gees            := 0.0;
      cds             := 0.0;
      myTime           := SimTime;
      t             := SimTime-myTime;
      gmax             := 0.0;
      tpr             := t;
      tpoll            := t;
      tdrift           := t + dtdrift;
      alt             := -1.0 * xe[3];
      h              := alt;
      bigloop           := 0;
      myDrift          := seed1.UniformReal (0.0, 4.0);
{      myDrift          := seed1.Normal (2.0, 0.5);}
      myDriftDirection   := seed2.UniformReal (0.0, 360.0);
{      myDriftDirection   := seed2.Normal (0.0, 2.8125);}
      xe[1]            := Airdrop.Information[myPlane].xg;
      xlast            := xe[1];
      xe[2]            := Airdrop.Information[myPlane].yg + 9.25;
      alt             := Airdrop.Information[myPlane].altitude;
      xe[3]            := -1.0 * alt;
```

```
          ue[1]          := Airdrop.Information[myPlane].vf;

          vwind[1]         := HeadWind;  {FROM globalMod}

          NEW (lastRightLocation, 1..myPlane-1);
          NEW (lastLeftLocation,  1..myPlane-1);

          FOR i := 1 TO myPlane-1
             lastRightLocation[i] := 1;
             lastLeftLocation[i] := 1;
          END FOR;

          ASK SELF TO findDrift;

          IF printTrajectory
             NEW (stream);
             outfile := "RJ" + INTTOSTR(myPlane) + INTTOSTR(myNumber) + extension + ".mat";
             ASK stream TO Open (outfile, Output);
          END IF;
{
OUTPUT ();
OUTPUT ("My airplane    : ", myPlane);
OUTPUT ("Exit time      : ", SimTime);
OUTPUT ("My number      : ", stick, "R");
OUTPUT ("My weight      : ", weight);
OUTPUT ("Xs             : ", xe[1]);
OUTPUT ("Ys             : ", xe[2]);
OUTPUT ("altitude       : ", alt);
}
ASK streamI TO WriteString (INTTOSTR(repeat) + " ");
ASK streamI TO WriteString (INTTOSTR(myPlane) + " ");
ASK streamI TO WriteString (REALTOSTR(SimTime) + " ");
ASK streamI TO WriteString (INTTOSTR(stick) + "R ");
ASK streamI TO WriteString (REALTOSTR(weight) + " ");
ASK streamI TO WriteString (REALTOSTR(xe[1]) + " ");
ASK streamI TO WriteString (REALTOSTR(xe[2]) + " ");
ASK streamI TO WriteString (REALTOSTR(alt) + " ");
ASK streamI TO WriteLn;

      END METHOD {initialize};

   ASK METHOD pollVortices (IN vortexPlane : INTEGER);

      VAR

         x, xcord1, xcord2, vvx, vjx      : REAL;
         y, ycord1, ycord2, vvy, vjy      : REAL;
         z, zcord1, zcord2, vvz, vjz      : REAL;
         vjdistance, vvdistance, distance : REAL;
         projection                 : REAL;
         i, location                : INTEGER;
         check                      : BOOLEAN;
         startRightSearch               : INTEGER;
```

```
          startLeftSearch               : INTEGER;

       BEGIN

          check := FALSE;
          location := 0;

          startRightSearch := lastRightLocation[vortexPlane];

          { ----- POLL RIGHT VORTEX ----- }

          FOR i := startRightSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
             IF location = 0
                IF    (ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].xCord-
xs[1])) <= 50.0
                   lastRightLocation[vortexPlane] := i;
                   IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].zCord+xs[3])) <= 50.0
                      lastRightLocation[vortexPlane] := i;
                      IF
(ABS(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[i].yCord-xs[2])) <= 50.0
                         check      := TRUE;
                         location   := i;
                         IF location = Airdrop.Information[vortexPlane].NumberOfSteps
                            check  := FALSE;
                         END IF;
                         lastRightLocation[vortexPlane] := i;
                      ELSE
                         location := i;
                      END IF;
                   ELSE
                      location := i;
                   END IF;
                END IF;
             ELSE
                EXIT;
             END IF;
          END FOR;

          IF check
             xcord1                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].xCord;
             ycord1                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].yCord;
             zcord1                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].zCord;
             xcord2                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].xCord;
             ycord2                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].yCord;
             zcord2                                                                          :=
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].zCord;
```

```
                vvx := xcord2-xcord1;
                vvy := ycord2-ycord1;
                vvz := zcord2-zcord1;

                vjx := xs[1]-xcord1;
                vjy := xs[2]-ycord1;
                vjz := -1.0*xs[3]-zcord1;

                vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
                vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
                projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
                distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
                IF                           distance                                  <=
MAXOF(Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.RightVortex.CompletePosition[location+1].radius)

                  OUTPUT (myNumber, "R ", myPlane, " RV ", vortexPlane, " ", -1.0*xe[3], " ", distance, "
", location, " ", SimTime);

                  ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
                  ASK streamE TO WriteString (INTTOSTR(myNumber) + "R ");
                  ASK streamE TO WriteString (INTTOSTR(myPlane) + " RV ");
                  ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
                  ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
                  ASK streamE TO WriteString (REALTOSTR(distance) + " ");
                  ASK streamE TO WriteString (REALTOSTR(location) + " ");
                  ASK streamE TO WriteString (REALTOSTR(SimTime));
                  ASK streamE TO WriteLn;

               ELSE
{
                  OUTPUT (myNumber, "R no encounter with right vortex of C-17 number", vortexPlane, "
at distance ", distance);
}
               END IF;
             ELSE
{
               OUTPUT (myNumber, "R no encounter with right vortex of C-17 number", vortexPlane);
}
             END IF;

             check := FALSE;
             location := 0;

             startLeftSearch := lastLeftLocation[vortexPlane];

             { ----- POLL LEFT VORTEX ----- }

             FOR i := startLeftSearch TO Airdrop.Information[vortexPlane].NumberOfSteps
               IF location = 0
                 IF      (ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].xCord-
xs[1])) <= 50.0
                    lastLeftLocation[vortexPlane] := i;
                  IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].zCord+xs[3])) <= 50.0
                     lastRightLocation[vortexPlane] := i;
                     IF
(ABS(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[i].yCord-xs[2])) <= 50.0
                        check      := TRUE;
                        location   := i;
                        IF location = Airdrop.Information[vortexPlane].NumberOfSteps
                           check  := FALSE;
                        END IF;
                        lastRightLocation[vortexPlane] := i;
                     ELSE
                        location := i;
                     END IF;
                  ELSE
                     location := i;
                  END IF;
               END IF;
            ELSE
               EXIT;
            END IF;
         END FOR;

         IF check
            xcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].xCord;
            ycord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].yCord;
            zcord1                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].zCord;
            xcord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].xCord;
            ycord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].yCord;
            zcord2                                                                              :=
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].zCord;

            vvx := xcord2-xcord1;
            vvy := ycord2-ycord1;
            vvz := zcord2-zcord1;

            vjx := xs[1]-xcord1;
            vjy := xs[2]-ycord1;
            vjz := -1.0*xs[3]-zcord1;

            vjdistance := SQRT(POWER(vjx,2.0)+POWER(vjy,2.0)+POWER(vjz,2.0));
            vvdistance := SQRT(POWER(vvx,2.0)+POWER(vvy,2.0)+POWER(vvz,2.0));
            projection := (vjx*vvx + vjy*vvy + vjz*vvz)/vvdistance;
            distance   := SQRT(POWER(vjdistance,2.0)-POWER(projection,2.0));
            IF                                   distance                                       <=
MAXOF(Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location].radius,
Airdrop.Information[vortexPlane].C17.LeftVortex.CompletePosition[location+1].radius)
```

```
                OUTPUT (myNumber, "R ", myPlane, " LV ", vortexPlane, " ", -1.0*xe[3], " ", distance, "
", location, " ", SimTime);

                ASK streamE TO WriteString (INTTOSTR(repeat) + " ");
                ASK streamE TO WriteString (INTTOSTR(myNumber) + "R ");
                ASK streamE TO WriteString (INTTOSTR(myPlane) + " LV ");
                ASK streamE TO WriteString (INTTOSTR(vortexPlane) + " ");
                ASK streamE TO WriteString (REALTOSTR(-1.0*xe[3]) + " ");
                ASK streamE TO WriteString (REALTOSTR(distance) + " ");
                ASK streamE TO WriteString (REALTOSTR(location) + " ");
                ASK streamE TO WriteString (REALTOSTR(SimTime));
                ASK streamE TO WriteLn;

             ELSE
{
                OUTPUT (myNumber, "R no encounter with left vortex of C-17 number", vortexPlane, " at
distance ", distance);
}
             END IF;
          ELSE
{
             OUTPUT (myNumber, "R no encounter with left vortex of C-17 number", vortexPlane);
}
          END IF;

       END METHOD {pollVortices};

    ASK METHOD changeDrift;

       BEGIN

          myDrift        := seed1.UniformReal (0.0, 4.0);
{            myDrift        := seed1.Normal (2.0, 0.5);}
          myDriftDirection := seed2.Normal (myDriftDirection, 2.8125);
          ASK SELF TO findDrift;

       END {ASK} METHOD {changeDrift};

    ASK METHOD findDrift;

       BEGIN

          Xdrift := myDrift * COS (myDriftDirection*pi/180.0);
          Ydrift := myDrift * SIN (myDriftDirection*pi/180.0);

       END {ASK} METHOD {findDrift};

END OBJECT {rightJumperObj};

END {IMPLEMENTATION} MODULE {rightJumperMod}.
```

Appendix D

Blake's FORTRAN Code

```
program blake
c
c C-17 analysis program for up to 18 ship formation
c
c this is a time stepping code that calculates the mutual
c interference between each vortex at each axial station
c
c ref for radius term: affdl-tr-79-3060 (kurylowich)
c ref for vortex strength term: affdl-tr-79-3060 (kurylowich)
c
c Sept 95 LIDAR data used for updated vortex model
c LIDAR updates only valid for C-17 calculations
c final LIDAR data not received, interim results used
c awaiting C-141, C-5 LIDAR data
c
      dimension y(36),z(36),rc(36),v(36),w(36),xv(36),xp(18),zs(18)
      dimension y0(36),y1(36),z0(36),z1(36),gam(36),gamd(36),gamc(36)
      dimension zu1(36),zl1(36),yl1(36),yr1(36),wt(18)
      real nu
      pi=acos(-1.)
      open(8,file='time',status='unknown')
      open(9,file='amc',status='unknown')
      open(10,file='pvic',status='unknown')
c
c variables
c
c ar   = wing aspect ratio
c         C-17:  7.16
c         C-5:   7.75
c         C-141: 7.93
c
c beff = wing span
c         C-17:  165 ft (changed from 169.8 during 1996 AIAA AFM work)
c         C-5:   222.7 ft
c         C-141: 159.9 ft
c
c bp   = span of fully rolled up vortices (pi/4*beff)
c cl   = lift coefficient
c delx = distance between computation planes (currently 200 ft)
c delt = time increment between compiutation planes (delx/vf)
c gam  = initial vortex circulation strength (elliptic load)
c gamd = circulation strength of decayed vortex
c gmod = LIDAR based modification to vortex strength
c gpr  = critical value of vortex strength below which no data printed
c hag  = height above ground for ground effect on vortex calculation
c hz118= height of zone 1 for 180 lb jumper -1 std deviation (127-14=113ft)
c hz136= height of zone 1 for 180 lb jumper +1 std deviation (127+14=141ft)
c ns   = current number of ships
c n    = current number of vortices (2*ns)
c nslim= total number of ships
c nt   = number of time steps
c         400 for time stepping (=80000 ft downstream of lead ship)
c         100 for drop path (=20000 ft downstream of last ship)
```

```
c
c nu   = air viscosity (absolute)
c nump = number of ship for personnel/equipment airdrop
c pp   = vertical position of personnel/equipment out of last ship
c px   = initial axial location of personnel/equipment, xp(nump)
c py   = initial lateral position of personnel/equipment
c pz   = initial vertical position of personnel/equipment, z(nump)
c pside= downstream distance before drop path follows crosswind (400 ft)
c r    = distance from vortex center to point of interest
c rc   = vortex core radius (includes growth with time)
c rho  = air density
c rhof = flag for density value
c s    = wing area
c          C-17:  3800 sq ft
c          C-5:   6200 sq ft
c          C-141: 3228 sq ft
c
c sepy = lateral separation between aircraft (centerline to centerline)
c tchar= characteristic time, time it takes for vortices to descend a distance
c        equal to their initial spacing
c tz1  = time jumper is in zone 1 (4.1 sec)
c uvm  = flag for use of LIDAR updated vortex model
c vage = age of a vortex
c vfk  = aircraft forward velocity in knots
c vf   = aircraft forward velocity in ft/sec
c vmod = vortex induced velocity model:
c        (1) 6/95 or George Kurylowich model
c        (2) NASA Langley or Burnham model
c vsk  = crosswind velocity in knots
c vs   = crosswind velocity in ft/sec
c vth  = radial velocity due to vortex
c vpe  = descent rate of equipment (28 ft/sec)
c vp18 = descent velocity of 360 lb jumper (14 ft/sec)
c vp36 = descent velocity of 360 lb jumper (21 ft/sec)
c v    = v-component of induced velocity at j due to all vortices
c w    = w-component of induced velocity at j due to all vortices
c vjj  = v-component of induced velocity at j due to vortex at jj
c wjj  = w-component of induced velocity at j due to vortex at jj
c wt   = array of aircraft weights
c xp   = array of x coordinates of ships in formation
c xs   = x coordinate of current computation plane
c y    = array of initial y-positions of vortices
c y0   = current y coordinate of vortex position (no interactions)
c y1   = current y coordinate of vortex position with interactions
c z    = array of initial z-positions of vortices
c z0   = current z coordinate of vortex position (no interactions)
c z1   = current z coordinate of vortex position with interactions
c zz   = dummy entry for output file (-999)
c
      uvm=0.
      print*,' '
      print*,'Wright Laboratory wake vortex trajectory program'
      print*,'Written by W. Blake, WL/FIGC, (513) 255-6764'
      print*,'revised 7/96'
      print*,' '
      print*,'enter 1 for Ft Bragg, 2 for EAFB, 3 for Wallops'
      read(*,*)  rhof
      print*,' '
      print*,'enter vehicle type'
      read(*,*) veh
      print*,' '
      print*,'enter (1) for LIDAR updated vortex model'
      read(*,*) uvm
      print*,' '
      print*,'enter (1) for Kurylowich model, (2) for NASA model'
      read(*,*) vmod
      print*,' '
      print*,'(1) echelon,(2) c-r,(3) inc.,(4) slice demo,(5) custom'
      read(*,*) ftype
      print*,' '
      if(ftype.ne.1) print*,'enter ship 2 lateral offset'
      if(ftype.ne.1) read(*,*) sepy
      if(ftype.ne.1) print*,' '
      if(ftype.eq.5) print*,'enter ship 3 lateral offset'
      if(ftype.eq.5) read(*,*) sepy3
      if(ftype.eq.5) print*,' '
c
      if(ftype.eq.3.) nslim=2
      if(ftype.eq.3.) xp(2)=2350.
      if(ftype.eq.3.) nump=2
c
      if(ftype.eq.4.) nslim=6
      if(ftype.eq.4.) xp(2)=2000.
      if(ftype.eq.4.) xp(3)=6000.
      if(ftype.eq.4.) xp(4)=8000.
      if(ftype.eq.4.) xp(5)=34377.
      if(ftype.eq.4.) xp(6)=36377.
      if(ftype.eq.4.) nump=6
c
      if(ftype.lt.2.5.or.ftype.gt.4.5) then
        print*,'enter maximum number of ships'
        read(*,*) nslim
        print*,' '
        print*,'enter gap for ships 2-nslim in K ft'
        read(*,*) (xp(i),i=2,nslim)
        do 25 i=1,nslim
          xp(i)=xp(i)*1000.
 25     continue
        print*,' '
        print*,'enter vertical location for ships 1-nslim'
        read(*,*) (zs(i),i=1,nslim)
        print*,' '
        print*,'enter ship number for airdrop'
        read(*,*) nump
      endif
      print*,'enter forward speed, crosswind in knots'
      read(*,*) vfk,vsk
      print*,' '
      print*,'enter (1) for 385K lb analysis'
      read(*,*) wfl
      print*,' '
      if(wfl.eq.1.) then
        do 30 i=1,nslim
          wt(i)=385.
   30   continue
      endif
      if(wfl.ne.1.) then
        print*,'enter vehicle weights in K lbs'
        read(*,*) (wt(i),i=1,nslim)
      print*,' '
      endif
      print*,'enter minimum circulation strength for plot output'
      read(*,*) gpr
      print*,' '
      print*,'computing.  please wait.'
      print*,' '
      do 40 i=1,nslim
        wt(i)=1000.*wt(i)
   40 continue
c
c  define initial constants
c
      vf=vfk*1.69085
      vs=-1.*vsk*1.69085
      if(veh.eq.5.) then
        beff=222.7
        ar=7.75
        s=6200.
      elseif(veh.eq.141) then
        beff=159.9
        ar=7.93
        s=3228.
      endif
      if(veh.ne.141.and.veh.ne.5) then
        beff=165.0
        ar=7.16
        s=3800.
      endif
      bp=pi*beff/4.
c
c  air density and kinematic viscosity
c
      if(rhof.eq.1.) rho=0.002309
      if(rhof.eq.2.) rho=0.002000
      if(rhof.eq.3.) rho=0.002378
c
      if(rhof.eq.1.) nu=0.0001654
      if(rhof.eq.2.) nu=0.0001776
      if(rhof.eq.3.) nu=0.0001572
```

```
c
      cswp2=(cos(25.*pi/180.))**2
      do 50 i=1,nslim
        cl=2.*wt(i)/(rho*vf*vf*s)
   50 continue
c
c descent velocity of personnel is 20 ft/sec, equipment is 28 ft/sec
c for single ship, undecayed vortex descent velocity
c
      vp=0.
c      if(vfk.lt.140.) vp=-20.0
c      if(vfk.gt.140.) vp=-28.0
      vp36=-21.0
      vp18=-14.0
      tz1=4.1
      hz136=141.
      hz118=113.
c      if(nslim.eq.1)  vp=-wt(1)/(rho*vf*bp*(2.*pi*bp))
c
c calculate initial positions of vortices
c
      xp(1)=0.
      if(ftype.eq.1.) then
        y(1)=-bp/2.
        y(2)=bp/2.
        y(3)=650.-bp/2.
        y(4)=650.+bp/2.
        y(5)=1500.-bp/2.
        y(6)=1500.+bp/2.
        do 100 i=1,36
          y(i+6)=y(i)
  100     continue
      endif
c
      if(ftype.eq.2..or.ftype.eq.3.) then
        y(1)=-bp/2.
        y(2)=bp/2.
        y(3)=sepy-bp/2.
        y(4)=sepy+bp/2.
        do 110 i=1,36
          y(i+4)=y(i)
  110     continue
        if(ftype.eq.3.) zs(1)=800.
        if(ftype.eq.3.) zs(2)=900.
      endif
c
      if(ftype.eq.4.) then
        y(1)=-bp/2.
        y(2)=bp/2.
        y(3)=180.-bp/2.
        y(4)=180.+bp/2.
        y(5)=-bp/2.
        y(6)=bp/2.
```

```
          y(7)=180.-bp/2.
          y(8)=180.+bp/2.
          y(9)=-bp/2.
          y(10)=bp/2.
          y(11)=sepy-bp/2.
          y(12)=sepy+bp/2.
          zs(1)=1500.
          zs(2)=1500.
          zs(3)=1500.
          zs(4)=1500.
          zs(5)=800.
          zs(6)=800.
        endif
c
        if(ftype.eq.5.) then
          y(1)=-bp/2.
          y(2)=bp/2.
          y(3)=sepy-bp/2.
          y(4)=sepy+bp/2.
          y(5)=sepy3-bp/2.
          y(6)=sepy3+bp/2.
          do 130 i=1,36
            y(i+6)=y(i)
  130     continue
        endif
c
        do 140 i=1,18
          z0(2*i-1)=zs(i)
          z0(2*i)=zs(i)
          z1(2*i-1)=zs(i)
          z1(2*i)=zs(i)
  140 continue
c
c  initial position of equipment/personnel
c
        pzi=zs(nump)
        pxi=xp(nump)
        pyi=(y(2*nump)+y(2*nump-1))/2.
c
c  modified strength from LIDAR data
c  for deck angle = 3 deg, gmod=0.55* elliptic load value
c  for deck angle = 4 deg, gmod=0.65* elliptic load value
c  for deck angle = 6 deg, gmod=0.73* elliptic load value
c  for deck angle = 8 deg, gmod=0.71* elliptic load value
c
c  current deck angle is 6 deg so deck angle not active variable
c  gmod=0.8 based on WL avg (most conservative) of LIDAR 6 deg deck data
c
        do 60 i=1,18
          gmod=1.0
          if(uvm.eq.1.) gmod=0.80
          if(veh.eq.141.) gmod=1.0
          gam(2*i-1)=wt(i)/(rho*vf*bp)*gmod
        gam(2*i)=wt(i)/(rho*vf*bp)*gmod
 60  continue
c
      do 150 i=1,36
        v(i)=0.
        w(i)=0.
        y0(i)=y(i)
        y1(i)=y(i)
        xv(i)=1.
        gamd(i)=gam(i)
 150 continue
c
c  customized formation information
c
      if(ftype.eq.3.) then
        do 160 i=1,2
          z(i)=800.
          z0(i)=800.
          z1(i)=800.
          z(i+2)=900.
          z0(i+2)=900.
          z1(i+2)=900.
 160    continue
      endif
c
      if(ftype.eq.4.) then
        do 170 i=1,8
          z(i)=1500.
          z0(i)=1500.
          z1(i)=1500.
          z(i+8)=800.
          z0(i+8)=800.
          z1(i+8)=800.
 170    continue
      endif
c
c  time stepping loop (axial spacing)
c  time step size is axial step/velocity
c  solution starts 10ft aft of 1st ship
c
c     nt=(nslim-1)*20+10+1
      nt=800
      ns=0
      delx=100.
      delt=delx/vf
      xs=0.
c     write(8,*) 'ZONE T="   " i=1000'
      write(9,399)
 399 format('VARIABLES="t","x","y0","z0","y1","z1","gam",'
   c        '"zu","zl","yl","yr"')
      write(9,*) 'ZONE T="   "'
      tv=0.
c
```

```
c  master loop for time stepping analysis
c  printc controls printed output to every 10th time step
c
      printc=9.
      do 5000 it=1,nt
c
c  count number of ships forward of current x-station
c
      ns=1
      do 200 i=1,nslim
        if(xs.ge.xp(i)) ns=i
  200 continue
      n=2*ns
c
c  calculate vortex core radii including decay term for "old" vortices
c
      j=0
      do 300 i=1,ns
        j=j+1
        xv(j)=xs-xp(i)
        xv(j+1)=xv(j)
        rc(j)=36.2*sqrt(nu*xv(j)/vf*cswp2)
        rc(j+1)=rc(j)
        j=j+1
  300 continue
c
c  loop on each vortex for induced velocities at current core position
c
      do 3000 j=1,n
      yp=y1(j)
      zp=z1(j)
c
c  calculate induced velocites at vortex j due to vortices jj
c  circulation includes effects of vortex decay
c  cjj is counter for left/right vortices to ensure correct rotation
c
      vjj=0.
      wjj=0.
      cjj=-1.
      do 1000 jj=1,n
        cjj=cjj*-1.
        dfact=xv(jj)*cl/(beff*ar)
        gamd(jj)=gam(jj)
        if(dfact.gt.9.58) gamd(jj)=gam(jj)*9.58/dfact
c
c  modified vortex decay model from LIDAR data
c
c  waiting for final representation from LIDAR data
c  60/t based on peer review results
c  -.06 decay based on all runs from 9/11 and most from 9/7 and 9/12
c
        if(uvm.eq.1.) then
          vage=xv(jj)/vf
```

```
         tchar=2*pi*bp**2/gam(jj)
         if(vage.le.60.) gamd(jj)=gam(jj)
         if(vage.gt.60.) gamd(jj)=gam(jj)*(60./vage)
c         if(vage.le.2.*tchar) gamd(jj)=gam(jj)
c         if(vage.gt.2.*tchar) gamd(jj)=gam(jj)-0.06*xv(jj)
        endif
c
c  vortex interaction calculation
c
        if(j.eq.jj) go to 990
        if(xv(j).ne.xv(jj).and.xv(jj).lt.500.) go to 990
          dely=yp-y1(jj)
          delz=zp-z1(jj)
          r=sqrt(dely**2+delz**2)
          rcrit=bp
          if(xv(j).ne.xv(jj).and.r.lt.rcrit) r=rcrit
c
c  select vortex model
c
          if(vmod.eq.1) then
            vth=gamd(jj)*(1-exp(-1.26*(r/rc(jj))**2))/(2*pi*r)
            vm=gamd(jj)*(1-exp(-1.26))/(2*pi*rc(jj))
          else
            rcn=rc(jj)
            if(rcn.lt.2) rcn=2.
            vth=gamd(jj)*r/(2*pi*(rcn**2+r**2))
            vm=gamd(jj)/(4*pi*rcn)
          endif
          vth=vth*cjj
          r=sqrt(dely**2+delz**2)
          vjj=vjj+vth*delz/r
          wjj=wjj-vth*dely/r
  990 continue
 1000 continue
      v(j)=vjj
      w(j)=wjj
c
c
 3000 continue
c
c  write output if print control is at tenth time step
c
      printc=printc+1.
      if(printc.eq.10.) then
      printc=0.
c
c
c  write vortex positions in x-z plane as function of time
c  upper and lower vortex boundaries for radial velocity of 30ft/sec
c
        zz=-999.
        do 4500 i=1,n
        zu1(i)=z1(1)+gamd(1)/(2.*pi*30.0)
      zl1(i)=z1(1)-gamd(1)/(2.*pi*30.0)
      yl1(i)=y1(2)+gamd(1)/(2.*pi*30.0)
      yr1(i)=y1(2)-gamd(1)/(2.*pi*30.0)
      ncnt=(i+1)/2
      xsg=xv(i)+200.*(ncnt-1)
c
c  do not write vortex to output file if strength is less than gpr
c  unit 8 is LIDAR data comparison output file
c  unit 9 is main tecplot output file
c  unit 10 is pvi test output file, compatible with John Watkins data format
c
      if(gamd(i).gt.gpr) then
        write(9,4513) tv,xs,y0(i),z0(i),y1(i),z1(i),gamd(i),
   c              zu1(i),zl1(i),yl1(i),yr1(i)
4513       format(f5.0,f8.0,9f7.0)
        cjj=cjj*-1.
        if(cjj.eq.1.) write(8,4888) tv,xs,zs(1)-z0(i)
4888       format(3f9.1)
          if(xs.le.25000.) then
            wz=1.0
            write(10,4514) wz,wz,wz,wz,wz,wz,wz,wz,wz,zs(1)-z0(i),xs
4514         format(9f4.1,2f9.1)
          endif
        endif
4500    continue
    endif
c
    xs=xs+delx
    tv=tv+delt
c
c  shift vortex position for next iteration
c
c  here for all vortex-vortex interactions included
c
    do 400 i=1,n
      y1(i)=y1(i)+v(i)*delt+vs*delt
      z1(i)=z1(i)+w(i)*delt
 400 continue
c
c  here for single ship vortex interactions only
c
c   do 500 i=1,n
c     vd=-1.*gamd(i)/(2.*pi*bp)
c     y0(i)=y0(i)+vs*delt
c     z0(i)=z0(i)+vd*delt
c 500 continue
c
c  ground effects coding for single ship interactions
c  replaces above loop
c
       djj=1.
    do 600 i=1,n
       djj=djj*-1.
```

```
          hag=z0(i)
          vd=-1.*gamd(i)/(2.*pi)*4*hag**2/(bp*(bp**2+4.*hag**2))
          yd=djj*gamd(i)/(2.*pi)*bp**2/(2*hag*(bp**2+4.*hag**2))
          y0(i)=y0(i)+vs*delt+yd*delt
          z0(i)=z0(i)+vd*delt
  600 continue
c
 5000 continue
c
c  end of time stepping loop
c
c  write positions of personnel/equipment (360 lb heavy jumper)
c
      pz=pzi
      py=pyi
      px=pxi
      write(9,*) 'ZONE T="  " i=100'
      nt=100.
      delx=200.
      pside=0.
      do 5100 it=1,nt
        write(9,4550) zz,px,py,pz,py,pz,zz,zz,zz,zz,zz
 4550   format(f5.0,f8.0,9f7.0)
        delt=delx/vf
c
c  ballistic portion
c
        itcrit=tz1*vf/200.
        if(it.ge.1.and.it.lt.itcrit) then
          px=px+vf*(it*delt)**2/8.
          pz=pz-(hz136/tz1)*delt
          py=py
        endif
c
c  steady descent portion
c
        if(it.ge.itcrit) then
          pz=pz+delt*vp36
          py=py+delt*vs
          px=px+delx
          pside=pside+delx
        endif
c
 5100 continue
c
c  write positions of personnel/equipment (180 lb light jumper)
c
      pz=pzi
      py=pyi
      px=pxi
      write(9,*) 'ZONE T="  " i=100'
      nt=100.
      delx=200.
      pside=0.
      do 5102 it=1,nt
        write(9,4550) zz,px,py,pz,py,pz,zz,zz,zz,zz,zz
        delt=delx/vf
c
c  ballistic portion
c
        itcrit=tz1*vf/200.
        if(it.ge.1.and.it.lt.itcrit) then
          px=px+vf*(it*delt)**2/8.
          pz=pz-(hz118/tz1)*delt
          py=py
        endif
c
c  steady descent portion
c
        if(it.ge.itcrit) then
          pz=pz+delt*vp18
          py=py+delt*vs
          px=px+delx
          pside=pside+delx
        endif
c
 5102 continue
c
c  write positions of personnel in thermal from ship 3
c  light jumper zone I with 5 ft/sec thereafter
c
      vpther=-5.0
      pz=zs(3)
      py=(y(5)+y(6))/2.
      px=xp(3)
      write(9,*) 'ZONE T="  " i=295'
      nt=295.
      delx=200.
      pside=0.
      do 5104 it=1,nt
        write(9,4550) zz,px,py,pz,py,pz,zz,zz,zz,zz,zz
        delt=delx/vf
c
c  ballistic portion
c
        itcrit=tz1*vf/200.
        if(it.ge.1.and.it.lt.itcrit) then
          px=px+vf*(it*delt)**2/8.
          pz=pz-(hz118/tz1)*delt
          py=py
        endif
c
c  steady descent portion
c
        if(it.ge.itcrit) then
          pz=pz+delt*vpther
          py=py+delt*vs
          px=px+delx
          pside=pside+delx
        endif
c
 5104 continue
c
c  end of calculations
c
c  calculate velocity components for velocity contours
c  at final solution plane
c  "i" is J in tecplot header, ii is I in tecplot header
c
c     ys=-1010.
c     delyc=10.
c     delzc=10.
c     do 5200 i=1,200
c     ys=ys+delyc
c     zs=160.
c     do 5400 ii=1,120
c     zs=zs-delzc
c
c  calculate induced velocity at coordinate ys,zs
c
c     vjj=0.
c     wjj=0.
c     cjj=-1.
c     do 6000 jj=1,n
c        cjj=cjj*-1.
c           gamc(jj)=sqrt(gamd(jj)*gamd(jj))
c           dely=ys-y1(jj)
c           delz=zs-z1(jj)
c           r=sqrt(dely**2+delz**2)
c           dfact=xv(jj)*cl/(beff*ar)
c           vth=gamc(jj)*(1-exp(-1.26*(r/rc(jj))**2))/(2*pi*r)
c           vth=vth*cjj
c           vjj=vjj+vth*delz/r
c           wjj=wjj-vth*dely/r
c6000 continue
c
c  write velocity components at ys,zs to unit 9
c
c     write(10,5500) ys,zs,vjj,wjj
c5500 format(2x,4(f9.1,1x))
c
c5400 continue
c5200 continue
c
 9999 continue
      close(8,status='keep')
      close(9,status='keep')
      close(10,status='keep')
      stop
```

```
end
```

Appendix E

Alternate Results

VI. Results

ANALYSIS

My analysis initially focused on varying the spacing between element leaders from 9,000 feet in trail to 40,000 feet, and measuring the encounter rate as the response. This approach was flawed because it was impossible within the simulation for a jumper to encounter a vortex at this distance because it was truncated at 42,000 feet. The following tables and figures show the initial regression results using the 40,000 feet data point.

Table 19. Distances and Random Seeds used for analysis

ELEMENT SPACING	9,000	16,000	24,000	32,000	40,000
Random Seeds	3	6	7		

Table 20. Input Data for Simulation Runs with 50 Replications at Each Point.

Input Data	9000 Ft	16000 Ft	24000 Ft	32000 Ft	34000 Ft	40000 Ft
Type of formation	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)	2 (Custom)
Number of Planes	6	6	6	6	6	6
Planes Per Element	3	3	3	3	3	3
Jumpers Per Side	6	6	6	6	6	6
#2 In-Trail Spacing	3000	3000	3000	3000	3000	3000
#2 Lateral Spacing	600	600	600	600	600	600
#3 In- Trail Spacing	6000	6000	6000	6000	6000	6000
#3 Lateral Spacing	-900	-900	-900	-900	-900	-900
Element Leader Spacing	9000	16000	24000	32000	32000	40000
In-Trail Tolerance	500	500	500	500	500	500
Lateral Tolerance	200	200	200	200	200	200
Aircraft Weight	385000	385000	385000	385000	385000	385000
Drop Altitude	900	900	900	900	900	900
Airspeed	135	135	135	135	135	135
Crosswind at Altitude	0	0	0	0	0	0
Standard Deviation (1)	.1	.1	.1	.1	.0	.1
Shear Altitude (1)	0	0	0	0	0	0
Crosswind (Middle)	0	0	0	0	0	0
Standard Deviation (2)	0	0	0	0	0	0
Shear Altitude (2)	0	0	0	0	0	0
Crosswind (Ground)	0	0	0	0	0	0
Standard Deviation (3)	0	0	0	0	0	0
Headwind	0	0	0	0	0	0
Air Density	0.002309	0.002309	0.002309	0.002309	0.002309	0.002309
Jumper Output File (# = 3 ,6, 7)	Seed#9J	Seed#16J	Seed#24J	Seed#32J	Seed#34J	Seed#40J
Encounter Output File (# = 3, 6, 7)	Seed#9E	Seed#16E	Seed#24E	Seed#32E	Seed#34E	Seed#40E
Scatter Output File (# = 3, 6, 7)	Seed#9S	Seed#16S	Seed#24S	Seed#32S	Seed#34S	Seed#40S
Random Seeds	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7	3, 6, 7
Individual Trajectories	0	0	0	0	0	0

Table 21. Results of Simulation Runs, Blocked by Seed.

Seed Number	Distance	Coded Spacing	Mean	95 % Confidence Interval
3	9,000	-.9375	40.67 %	6.93 %
6	9,000	-.9375	42.94 %	6.97 %
7	9,000	-.9375	36.72 %	6.86 %
3	16,000	-.5	18.33%	4.53 %
6	16,000	-.5	19.56 %	4.63 %
7	16,000	-.5	18.44 %	4.93 %
3	24,000	0	13.72 %	3.33 %
6	24,000	0	10.89 %	3.00 %
7	24,000	0	10.83 %	2.98 %
3	32,000	.5	12.06 %	2.57 %
6	32,000	.5	10.56 %	2.68 %
7	32,000	.5	8.94 %	2.18 %
3	40,000	1	0.00 %	0.00 %
6	40,000	1	0.00 %	0.00 %
7	40,000	1	0.00 %	0.00 %

Table 22. First Order Linear Regression Results, Coded Spacing vs Encounter Rate.

Linear Fit

Encounter Rate = 16.4691-17.976 Coded Spacing

Summary of Fit

RSquare	0.851314	RSquare Adj	0.839876
Root Mean Square Error	5.565309	Mean of Response	16.24444
Observations (or Sum Wgts)	15		

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	1	2305.3666	2305.37	74.4323	<.0001
Error	13	402.6446	30.97		
C Total	14	2708.0112			

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	16.469143	1.437193	11.46	<.0001
Coded Spacing	-17.97596	2.083585	-8.63	<.0001

Figure 13 . First Order Linear Regression with Confidence Interval and Residuals

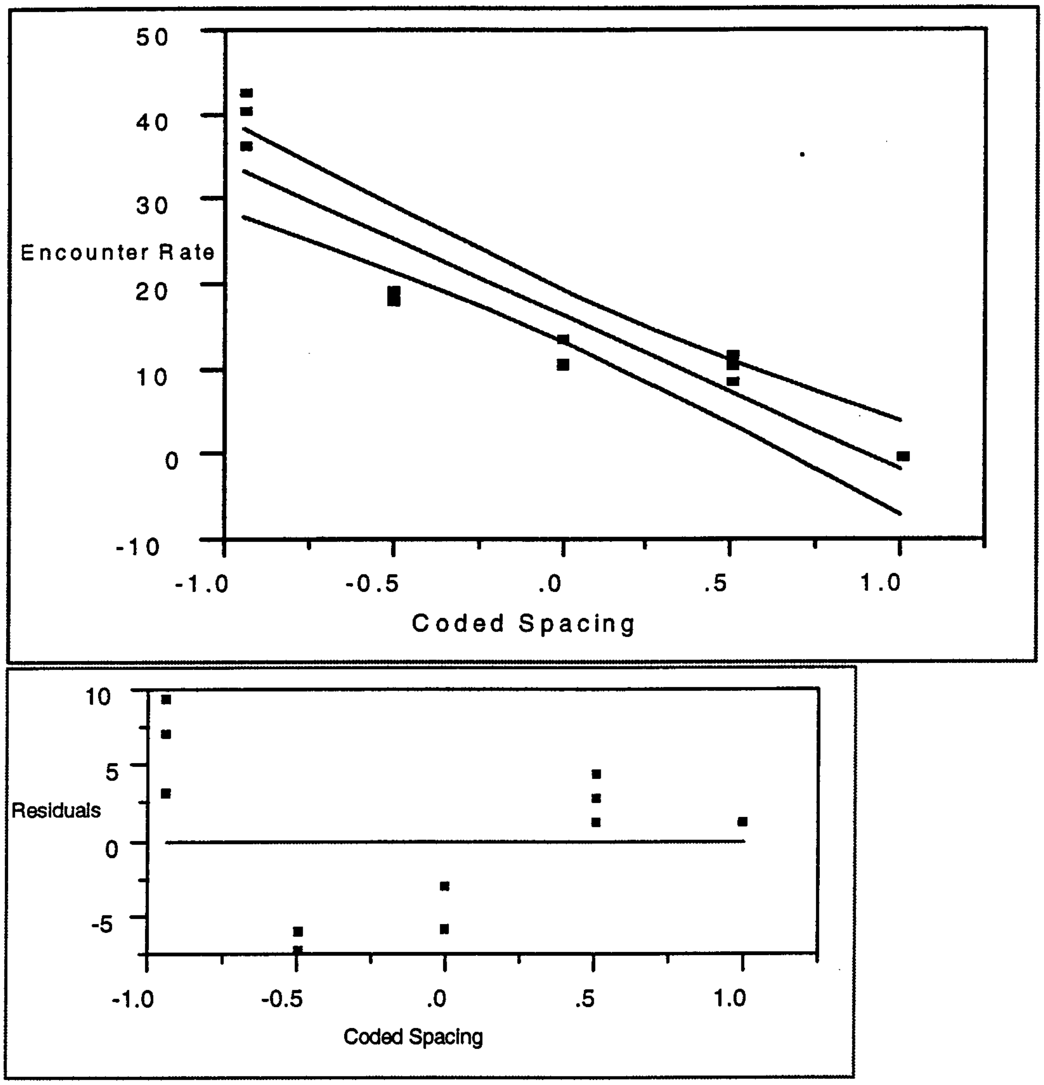


Table 23. Second Order Linear Regression Results, Coded Spacing vs. Encounter Rate.

Polynomial Fit degree=2

Encounter Rate = 12.3485-18.5096 Coded Spacing + 8.67476 Coded Spacing^2

Summary of Fit

RSquare	0.914391
RSquare Adj	0.900123
Root Mean Square Error	4.395358
Mean of Response	16.24444
Observations (or Sum Wgts)	15

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	2	2476.1811	1238.09	64.0861	<.0001
Error	12	231.8301	19.32		
C Total	14	2708.0112			

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
Intercept	12.348524	1.7913	6.89	<.0001	8.4456118	16.251437
Coded Spacing	-18.50958	1.655326	-11.18	<.0001	-22.11623	-14.90293
Coded Spacing^2	8.6747616	2.917356	2.97	0.0116	2.3183787	15.031145

Figure 14 . Second Order Regression with Confidence Interval and Residuals.

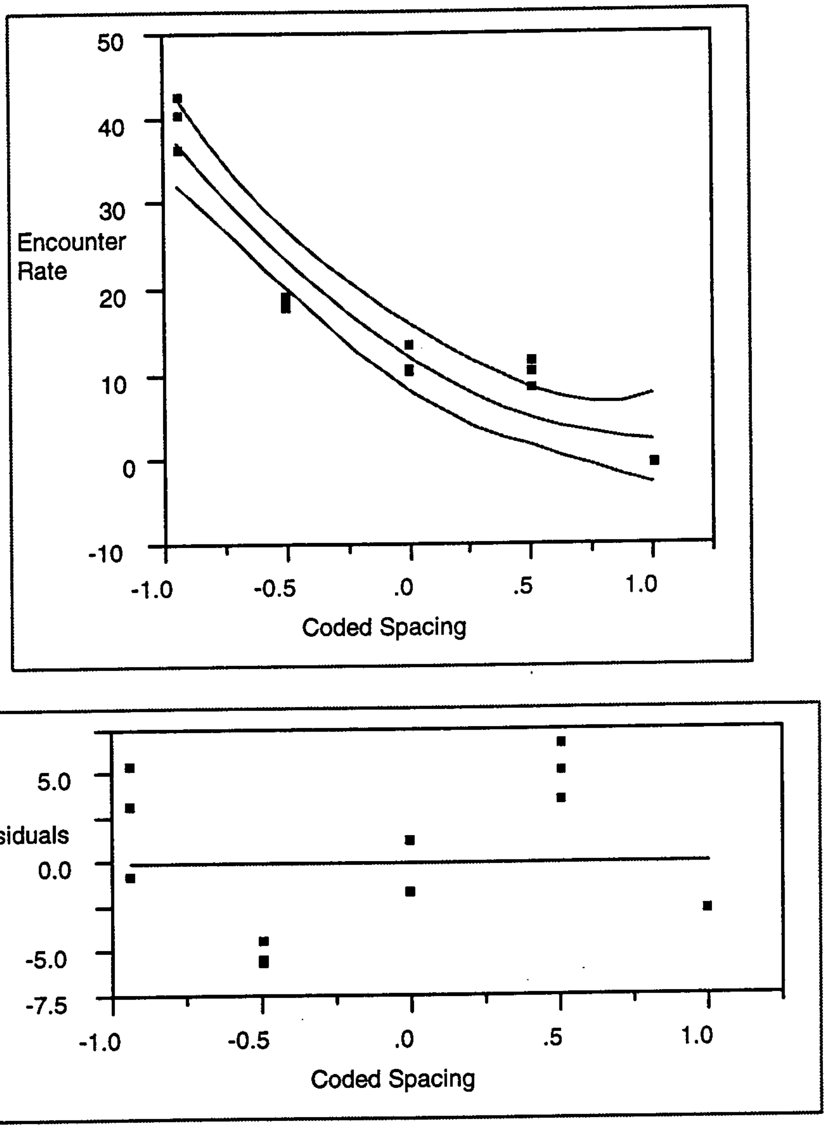


Table 24. Third Order Linear Regression Results, Coded Spacing vs. Encounter Rate.

Polynomial Fit degree=3

Encounter Rate = 11.9287-3.51065 Coded Spacing + 10.5664 Coded Spacing^2-18.9661 Coded Spacing^3

Summary of Fit

RSquare	0.988511
RSquare Adj	0.985377
Root Mean Square Error	1.681801
Mean of Response	16.24444
Observations (or Sum Wgts)	15

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Model	3	2676.8982	892.299	315.4726	<.0001
Error	11	31.1130	2.828		
C Total	14	2708.0112			

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
Intercept	11.928715	0.687216	17.36	<.0001	10.416156	13.441273
Coded Spacing	-3.51065	1.889804	-1.86	0.0902	-7.670096	0.6487963
Coded Spacing^2	10.566445	1.138634	9.28	<.0001	8.0603179	13.072572
Coded Spacing^3	-18.96614	2.251444	-8.42	<.0001	-23.92155	-14.01072

Figure 15. Full Third Order Residuals

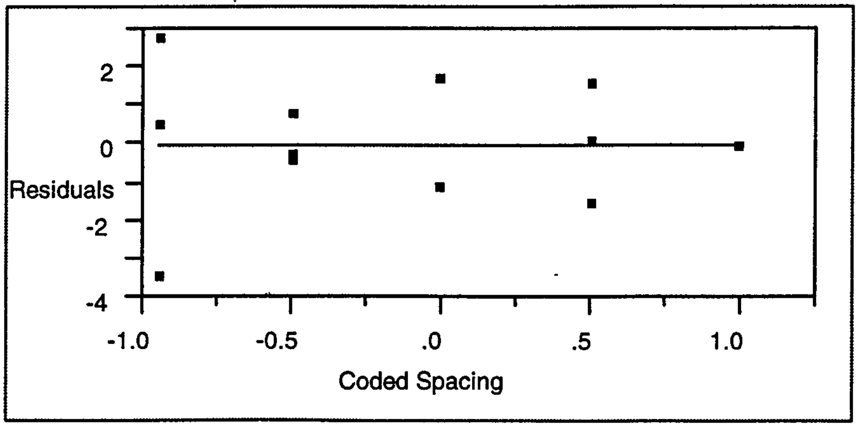


Figure 16. Full Third Order Regression Line with Confidence Interval.

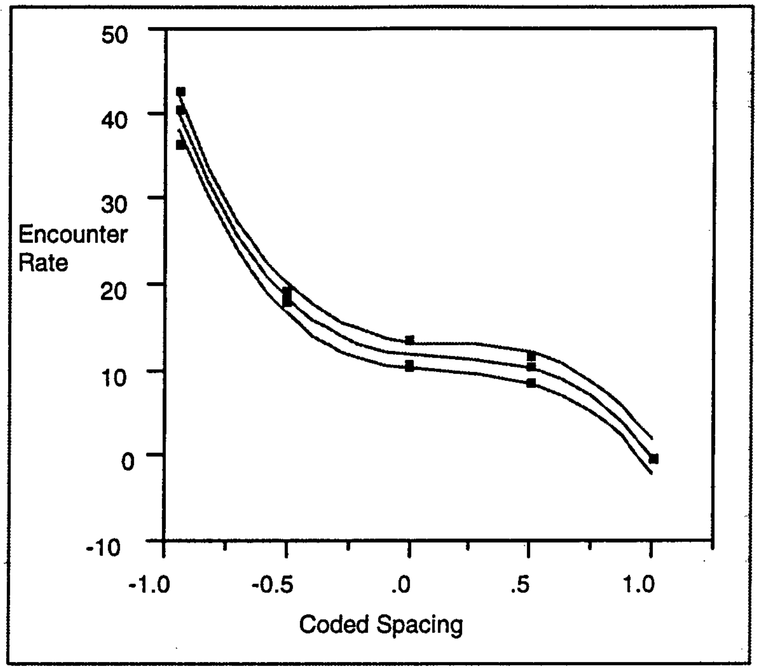


Table 25. Reduced Third Order Regression Results, Coded Spacing vs. Encounter Rate.

Response: Encounter Rate

Encounter Rate = 11.872+10.884 Coded Spacing^2-22.907 Coded Spacing^3

Summary of Fit

RSquare	0.984906
RSquare Adj	0.982391
Root Mean Square Error	1.845578
Mean of Response	16.24444
Observations (or Sum Wgts)	15

Anova

	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
Model	2	2667.137062	1333.568531	391.5169728	1.1824E-11
Residual	12	40.87389177	3.406157647		
Total	14	2708.010954			

Lack of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob>F
Lack of Fit	2	9.838898	4.91945	1.5851	0.2524
Pure Error	10	31.035002	3.10350		
Total Error	12	40.873901			

Max RSq
0.9885

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
Intercept	11.872412	0.753405	15.76	<.0001	10.23088	13.513943
Coded Spacing^2	10.884128	1.235344	8.81	<.0001	8.1925409	13.575716
Coded Spacing^3	-22.9067	0.828069	-27.66	<.0001	-24.71091	-21.10249

Figure 17. Reduced Third Order Curve with Confidence Interval and Residuals.

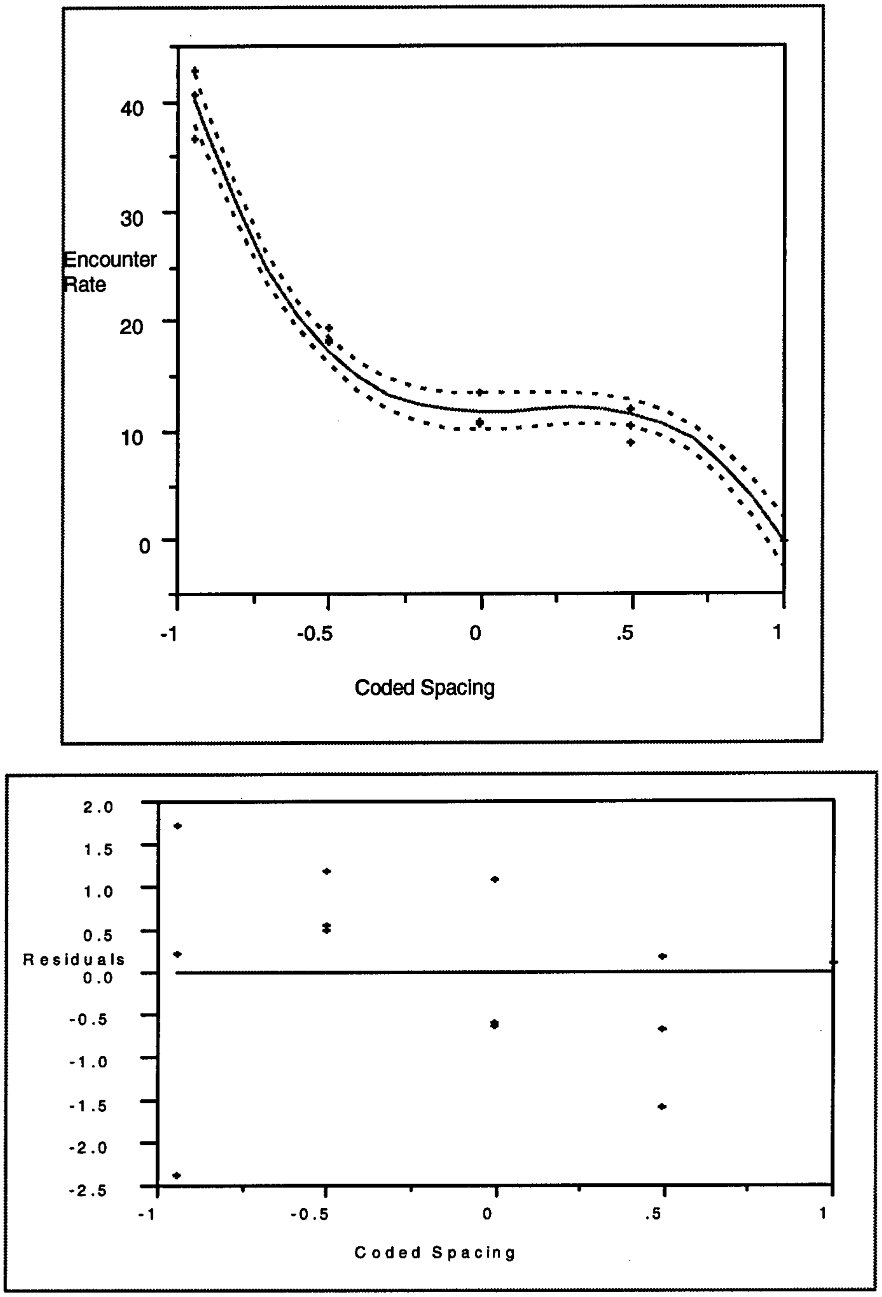


Table 26. Logistic Regression Results, Encounter Rate vs. Coded Spacing

Response: Response

Iteration History

Iter	LogLikelihood	Step	Delta-Criterion	Obj-Criterion
1	-11979.93205	Initial	1.32121516	1.5006e304
2	-10465.18195	Newton	0.32017887	0.14474174
3	-10302.02762	Newton	0.06034548	0.01583709
4	-10297.5812	Newton	0.00259701	0.00043179
5	-10297.57338	Newton	0.00000483	0.00000076

Converged by Gradient

Whole-Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	1682.359	1	3364.717	0.0000
Full	10297.573			
Reduced	11979.932			

RSquare (U)	0.1404
Observations (or Sum Wgts)	27000

Lack of Fit

Source	DF	-LogLikelihood	ChiSquare	Prob>ChiSq
Lack of Fit	3	275.216	550.4321	<.0001
Pure Error	26995	10022.357		
Total Error	26998	10297.573		

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq	Lower 95%	Upper 95%
Intercept	-2.0067577	0.0224443	7994.3	0.0000	?	?
C. S	-1.6160245	0.0321849	2521.1	0.0000	-1.6791123	-1.5529498

Effect Test

Source	Nparm	DF	Wald ChiSquare	Prob>ChiSq
Coded Spacing	1	1	2521.1051	0.0000

Effect Likelihood-Ratio Tests

Source	Nparm	DF	L-R ChiSquare	Prob>ChiSq
Coded Spacing	1	1	3364.7173	0.0000

Figure 18. Logistic Regression Prediction Curve.

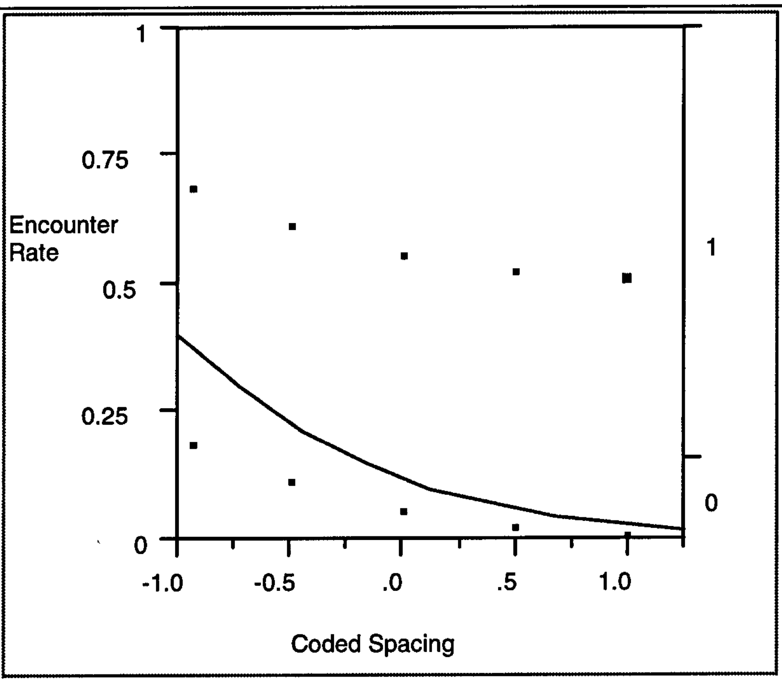
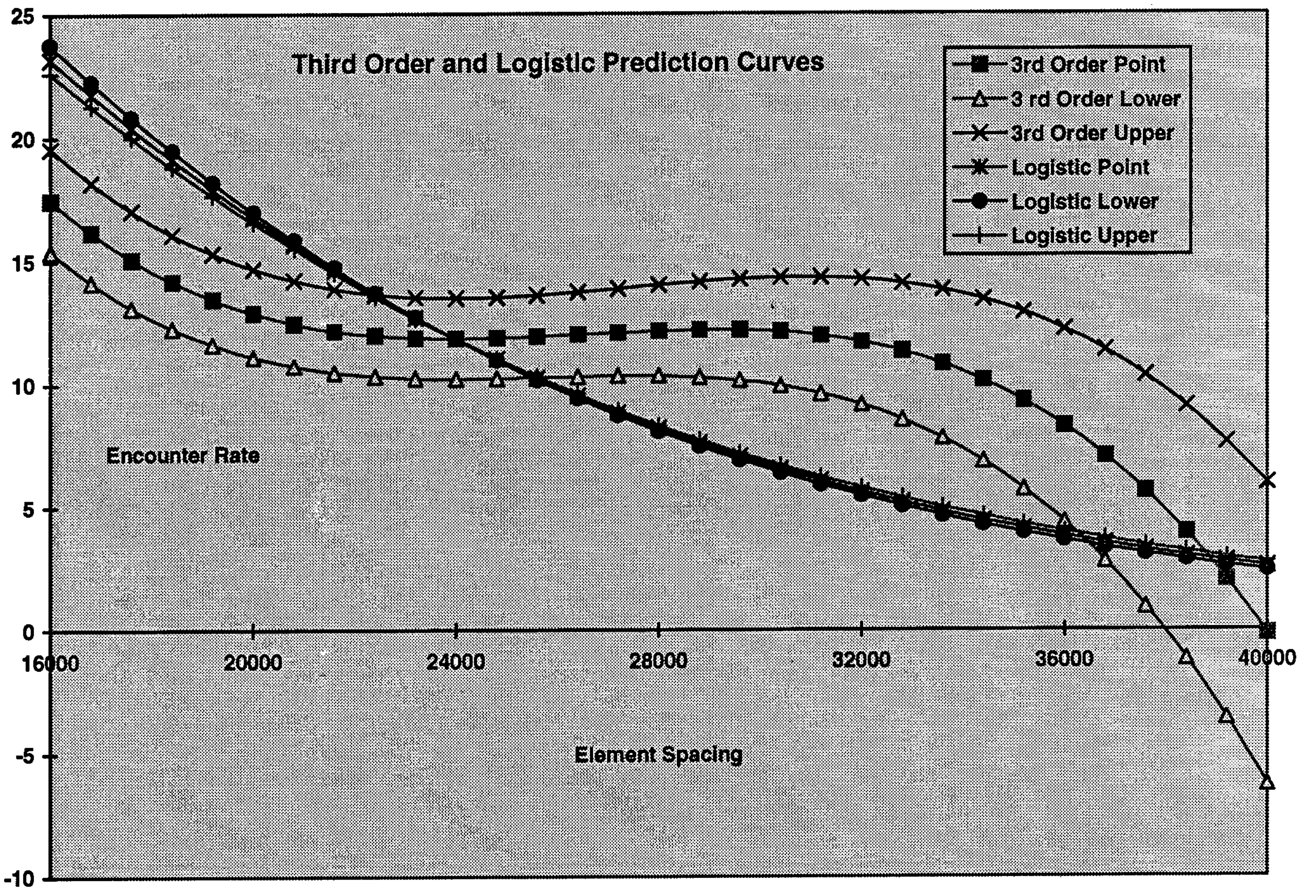


Table 27. Predicted Encounter Rate for Third Order and Logistic Models

		Third Order Encounter Rate			Logistic Encounter Rate		
Spacing	Coded Spacing	Point	Lower	Upper	Point	Lower	Upper
16000	-0.5	17.46	15.37	19.55	23.17	23.74	22.61
16800	-0.45	16.16	14.14	18.19	21.76	22.25	21.28
17600	-0.4	15.08	13.12	17.04	20.42	20.83	20.01
18400	-0.35	14.19	12.29	16.08	19.14	19.48	18.80
19200	-0.3	13.47	11.64	15.31	17.92	18.20	17.64
20000	-0.25	12.91	11.13	14.69	16.76	16.98	16.54
20800	-0.2	12.49	10.76	14.23	15.66	15.83	15.50
21600	-0.15	12.19	10.50	13.89	14.62	14.74	14.51
22400	-0.1	12.00	10.34	13.67	13.64	13.72	13.57
23200	-0.05	11.90	10.25	13.55	12.72	12.75	12.68
24000	0	11.87	10.23	13.51	11.85	11.85	11.85
24800	0.05	11.90	10.25	13.55	11.03	11.00	11.06
25600	0.1	11.96	10.29	13.63	10.26	10.20	10.32
26400	0.15	12.04	10.33	13.75	9.54	9.46	9.62
27200	0.2	12.12	10.36	13.89	8.87	8.77	8.97
28000	0.25	12.19	10.36	14.03	8.24	8.12	8.36
28800	0.3	12.23	10.30	14.17	7.65	7.51	7.78
29600	0.35	12.22	10.17	14.27	7.09	6.95	7.24
30400	0.4	12.15	9.96	14.34	6.58	6.43	6.74
31200	0.45	11.99	9.64	14.34	6.10	5.94	6.26
32000	0.5	11.73	9.19	14.27	5.65	5.49	5.82
32800	0.55	11.35	8.60	14.11	5.24	5.07	5.41
33600	0.6	10.84	7.84	13.84	4.85	4.68	5.03
34400	0.65	10.18	6.91	13.45	4.49	4.32	4.67
35200	0.7	9.35	5.77	12.93	4.16	3.98	4.34
36000	0.75	8.33	4.41	12.25	3.85	3.68	4.03
36800	0.8	7.11	2.82	11.40	3.56	3.39	3.74
37600	0.85	5.67	0.97	10.36	3.29	3.12	3.47
38400	0.9	3.99	-1.15	9.13	3.04	2.88	3.22
39200	0.95	2.06	-3.56	7.67	2.81	2.65	2.98
40000	1	-0.15	-6.29	5.99	2.60	2.45	2.77

Figure 19 . Third Order and Logistic Prediction Curves.



Bibliography

Babarsky, Richard, Aerodynamic Engineer, AEA Technology Engineering Software, Inc. Bethel Park, PA, Facsimile, 16 JAN 97, Personal Interview, 9 JAN 97.

Belano, Jose C., "C-17/Paratroop Risk Assessment Analysis", MS Thesis, AFIT/GOR/ENS/97M-1. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, March 1997.

Blake, William, Aerodynamic Engineer, Wright Laboratory, Wright Patterson AFB OH, Personal Interviews, 3 JUL 1996-22 JAN 97.

Blake, William, "Prediction of Paratroop/Wake Vortex Encounters During Formation Airdrop", AIAA paper 96-3387, July 96 Atmospheric Flight Mechanics Conference, San Diego, CA.

Clawson, Kirk L., Randy C. Johnson, G. R. Ackermann, Jerry F. Sagendorf, Neil F. Hukari, "Measurement of Wingtip Vortex Characteristics From C-130, C-141, and C-5A/B Aircraft", U.S. Department of Commerce, National Oceanic and Atmospheric Administration, Environmental Research Laboratory Field Research Division, Idaho Falls, ID, May 31, 1988.

Coffin, Charles, Statistician, Air Force Combat Climatology Center, Scott AFB, IL "Wind Distributions", Electronic Message, 21 FEB 97.

"Flight Test Data of 16 AUG 96 Six Ship Airdrop Test", C-17 System Program Office, Wright Patterson AFB, OH, 16 AUG 96.

Hannon, Stephen J., J. Alex Thomson, Sammy W. Henderson, R. Milton Huffaker,; "Windshear, Turbulence, and Wake Vortex Characterization Using Pulsed Solid State Coherent Lidar", Proceedings of SPIE - The International Society for Optical Engineering , Volume 2464, 1995, Society of Photo-Optical Instrumentation Engineers, pp. 94-102.

Johnson, David J., "Operational Test and Evaluation of the Effects of C-130/C-141B Wake Vortices on the Drop Zone Environment. Final Report.", USAF Airlift Center, Military Airlift Command, Pope Air Force Base, NC, September 1988.

Johnson, David J., Jon K. Reynolds, "Operational Test and Evaluation of the Effects of C-5 Wake Vortices on the Drop Zone Environment. Final Report", USAF Airlift Center, Military Airlift Command, Pope Air Force Base, NC, December 1988.

"Summary Report of Parachute Vortex Interaction Testing with C-141B and C-17 Aircraft (Preliminary)", C-17 System Program Office, Wright Patterson AFB OH, 20 JUN 1996.

Thomson, J. Alex, Stephen M. Hannon,; "Wake Vortex Modeling for Airborne and Ground-based Measurements Using a Coherent Lidar", Proceedings of SPIE - The International Society for Optical Engineering , Volume 2464, 1995, Society of Photo-Optical Instrumentation Engineers, pp. 63-78.

Wald, Matthew L., Commuter Planes Must Now Increase Distance From Jets, New York Times, 17 Aug 96, Section 1, page 6, Column 1.

White, Major Thomas, Mobility Analyst, Headquarters Air Mobility Command, Scott AFB IL, Personal Interview, 4-6 SEP 1996.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 97	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE AN OBJECT-ORIENTED SIMULATION OF THE C-17 WINGTIP VORTICES IN THE AIRDROP ENVIRONMENT	5. FUNDING NUMBERS
6. AUTHOR(S) Hans J. Petry, Major, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology/ENS 2950 P Street Wright-Patterson AFB, Ohio 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOA/ENS/97M-13
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ASC/YC (MS) IPT C-17 System Program Office Building 558 2590 Loop Road West Wright-Patterson AFB, OH 45433-7105	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release; Distribution is Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis effort focuses on the development of an object-oriented simulation of C-17 personnel airdrop operations and provides a tool for risk assessment of jumper and wingtip vortex interaction. Using the initial modeling efforts of the Wright Laboratory, this model expands those efforts to include random aircraft, wind and jumper movement within the simulation using MODSIM III as its language.

Once the model was built, verified, and calibrated, it helped perform a preliminary analysis of jumper risk with varying element spacing and no crosswind. The results of the simulation provided 15 data points with which linear and logistic regression provided an estimation of the marginal rate of change of jumper/vortex encounter rate.

Using the third order model shows that the encounter rate levels off around 24,000 feet spacing between element leaders at 12%, and stays as high as 11% at 32,000 feet before dropping to 0.4% at 34,000 feet. Further research and model improvements may bring the encounter rate down at the more distant spacing but that is left for post thesis analysis efforts.

14. SUBJECT TERMS Vortex modeling, object-oriented simulation, airdrop simulation, paratrooper/wake vortex encounter modeling, MODSIM	15. NUMBER OF PAGES 177
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to *stay within the lines* to meet *optical scanning requirements.*

Block 1. Agency Use Only *(Leave blank).*

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	Contract	PR	Project
G	Grant	TA	Task
PE	Program Element	WU	Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only).*

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

★U.S.GPO:1993-0-336-043