

**DEVELOPMENT OF A COMPUTER BASED  
AIRSPACE SECTOR OCCUPANCY MODEL**

by  
**Shrinivas M. Sale**

Thesis submitted to the faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE  
IN  
CIVIL ENGINEERING**

**APPROVED:**

---

**Antonio A. Trani, Ph.D., Chairman    Hanif D. Sherali, Ph. D., Chairman**

---

**Wei H. Lin, Ph.D.**

**June, 1998  
Blacksburg, Virginia**

# **DEVELOPMENT OF A COMPUTER BASED AIRSPACE SECTOR OCCUPANCY MODEL**

by  
Shrinivas M. Sale

Dr. A. A. Trani, Chairman

Dr. Hanif D. Sherali, Chairman

Civil Engineering

Industrial and Systems Engineering

## **(ABSTRACT)**

This thesis deals with the development of an Airspace Sector Occupancy Model (ASOM). The model determines the occupancy of Air Traffic Control Center (ARTCC) sectors for a given geometry of sectors and flight schedules, and can be used to study the impact of alternative flight schedules on the workload imposed on the sectors. Along with complimentary airspace analysis models, this can serve as an advisory tool to approve flight plans in the Free Flight Scenario, or to reschedule flights around a Special Use Airspace (SUA).

ASOM is developed using Matlab 5.2, and can be run on an IBM compatible PC, Macintosh, or Unix Workstation. The computerized model incorporates the powerful features of graphics and hierarchical modeling inherent in Matlab, to design an effective tool for analyzing air traffic scenarios and their respective sector occupancies.

# TABLE OF CONTENTS

## 1. INTRODUCTION

1.1 Background.....	1
1.2 Air Traffic Operations.....	2
1.3 Free Flight.....	3
1.4 Impact of Special Use Airspace (SUA) on Air Traffic Management.....	3
1.5 Research Scope, Objective and Approach.....	4

## 2. LITERATURE REVIEW

2.1 Introduction .....	5
2.2 Definition of Workload.....	5
2.3 Workload and Air Traffic Control Performance .....	5
2.4 Modeling Workload.....	6
2.4.1 Time-Line Analysis .....	6
2.4.2 Extended Time-Line Analysis.....	6
2.4.3 Kip Smith Dynamic Density Metric Analysis .....	7
2.4.4 Laudeman’s Dynamic Density Metric Analysis.....	8
2.5 Occupancy and Workload.....	9

### **3. MODEL METHODOLOGY**

3.1 Introduction .....	11
3.2 Flight Plan Generation .....	12
3.3 Sector Description.....	12
3.4 Occupancy Determination.....	13

### **4. MODEL DESCRIPTION**

4.1 Occupancy Model Framework.....	16
4.2 Definition of Terms .....	17
4.3 Pre-processing Sector Data .....	19
4.3.1 Inward Gradient .....	20
4.3.2 Types of Vertices .....	21
4.3.3 Adjacency with Respect to Nodes.....	21
4.3.4 Adjacency Information with Respect to Sector Modules .....	22
4.3.5 Identifying Pseudo-Vertices.....	23
4.3.6 Adjacency Information with Respect to Vertical Faces.....	25
4.4 Pre-processing Airport Data.....	25
4.5 Pre-processing Flight Plans.....	26
4.5.1 Dummy Sectors.....	26
4.5.2 Vacuums .....	26
4.6 Occupancy Determination Algorithm .....	28

4.6.1	Algorithm for Two Dimensional Case .....	28
4.6.2	Extension of Algorithm to Handle Airspace Vacuums.....	31
4.6.3	Extension of the Algorithm for the Three Dimensional Case.....	31
4.6.4	Determination of Point of Exit .....	33
4.6.5	Determination of the Next Sector Module after Exiting.....	34
4.6.6	Determination of the Next Sector Module after Passing through a Vacuum. ...	36
4.7	Data Structures.....	37
4.7.1	Sector Module Information Structure (S).....	37
4.7.2	Node Information Structure (Node) .....	39
4.7.3	Height Information about the Sector Modules (h).....	39
4.7.4	Structure with Mathematical Representation of Sector Modules (Se). .....	39
4.7.5	Structure with Adjacency Information of Sector Modules with Respect to Faces (Adjsec).....	42
4.7.6	Structure with Adjacency Information of Sector Modules with Respect to Nodes (Adjsecnode).....	43
4.7.7	Sector Module Adjacency Information (Adj).....	43
4.7.8	Flight Plan Structure (Fp).....	44
4.7.9	Sector Information (main_S ).....	47
4.7.10	Sector Adjacency Information ( main_Adj) .....	48
4.7.11	Adjacency Information of Sectors with Respect to Nodes (main_Adjsecnode) .....	48
4.8	M-Files .....	49

4.8.1 M-file Description.....	52
<b>5. MODEL APPLICATION AND ANALYSIS</b>	
5.1 Introduction .....	66
5.2 Model Application .....	66
5.2.1 Current National Airspace (NAS) Operations (Rt).....	69
5.2.2 Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes (Cardinal_Asg).....	70
5.2.3 Wind-Optimized Routing with a Reduced Vertical Separation and Assigned Altitudes (RVSM_Asg).....	70
5.2.4 Wind-Optimized Routing with Hemispherical Rules (Cardinal).....	71
5.2.5 Wind-Optimized Profiles with a Reduced Vertical Separation Method (RVSM) .....	71
5.2.6 Wind-Optimized Profiles without Hemispherical Rules (Climb_Cruise).....	72
5.3 Model Assumptions .....	72
5.4 Model Output.....	72
<b>6. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH</b>	
6.1 Introduction .....	89
6.2 Conclusions .....	89
6.3 Recommendations for Future Research.....	90
<b>BIBLIOGRAPHY.....</b>	<b>91</b>

<b>APPENDIX A</b>	<b>List of Airports.....</b>	<b>92</b>
<b>APPENDIX B</b>	<b>Input File Format.....</b>	<b>93</b>
<b>APPENDIX C</b>	<b>Matlab Programs.....</b>	<b>101</b>
<b>VITA</b>	<b>.....</b>	<b>145</b>

## LIST OF FIGURES

Figure 1 Overall Process of the Airspace Sector Occupancy Model (ASOM). .....	11
Figure 2 Typical Sector Geometry. ....	13
Figure 3 Occupancy Determination Methodology. ....	15
Figure 4 Occupancy Model Framework. ....	16
Figure 5 Components of a Sector Module. ....	17
Figure 6 Pre-processing of Sector Data. ....	19
Figure 7 Representation of a Module. ....	20
Figure 8 Types of Vertices. ....	21
Figure 9 Adjacency with Respect to a Vertex. ....	22
Figure 10 Pseudo-Vertices on a Face. ....	24
Figure 11 Pseudo-Vertices on a Vertical Face. ....	25
Figure 12 Dummy Sectors and Vacuums. ....	27
Figure 13 Representation of a Flight Segment. ....	28
Figure 14 Flight Internally Glancing a Vertex Point. ....	29
Figure 15 Data Structure of <i>S</i> . ....	38
Figure 16 Data Structure of <i>Node</i> . ....	39
Figure 17 Data Structure of <i>Se</i> . ....	40
Figure 18 Data Structure of <i>Adjsec</i> . ....	42
Figure 19 Data Structure of <i>Adjsecnode</i> ....	43
Figure 20 Data Structure of <i>Adj</i> . ....	43
Figure 21 Data Structure of Flight Plans ( <i>Fp</i> ). ....	46
Figure 22 Data Structure of <i>main_S</i> . ....	47

Figure 23	Data Structure of <i>main_Adj</i> . .....	48
Figure 24	Data Structure of <i>main_Adjsecnode</i> . .....	48
Figure 25	M-files Called by the Main Program. ....	49
Figure 26	Organization of M-Files to Process Aircraft Flight Plans (Process_Fp). .....	49
Figure 27	Organization of M-Files to Determine the Sector Modules Crossed (Occup)	50
Figure 28	Organization of M-Files to Pre-processes the Sector Information (Prepro_sectors). .....	51
Figure 29	Sector Modules in ZJX and ZMA Centers along with Dummy Sectors. ....	66
Figure 30	Sector Modules in the ZJX and the ZMA Center. ....	67
Figure 31	Airports Considered for the Analysis. ....	68
Figure 32	Flight Trajectories over the Airspace under Consideration. ....	69
Figure 33	Occupancy of Sector 14 (CDK) under Current NAS Operations. ....	74
Figure 34	Occupancy of Sector 57 (SJS) under Current NAS Operations. ....	75
Figure 35	Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes. ....	76
Figure 36	Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes. ....	77
Figure 37	Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with RVSM and Assigned Altitudes. ....	78
Figure 38	Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with RVSM and Assigned Altitudes. ....	79
Figure 39	Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with Hemispherical Rules. ....	80
Figure 40	Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with	

	Hemispherical Rules. ....	81
Figure 41	Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with RVSM. ....	82
Figure 42	Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with RVS. ....	83
Figure 43	Occupancy of Sector 14 (CDK) under Wind-Optimized Routing without Hemispherical Rules. ....	84
Figure 44	Occupancy of Sector 57 (SJS) under Wind-Optimized Routing without Hemispherical Rules. ....	85

## LIST OF TABLES

Table 1 Possible Cases of Sector Piercing Patterns.....	36
Table 2 Occupancy of Sector 14 (CDK) and Sector 57 (SJS) under Different Route Structures.....	86
Table 3 Occupancy of Sector 67 (HUN) and Sector 17 (FPY) under Different Route Structures.....	87
Table 4 Occupancy of Sector 15 (OCF) and Sector 58 (SGJ) under Different Route Structures.....	87
Table 5 Occupancy of Sector 84 (SEM)and Sector 16 (MAY) under Different Route Structures.....	88
Table 6 Occupancy of Sector 67 (D67) and Sector 72 (CAE) under Different Route Structures.....	88

# 1. INTRODUCTION

## 1.1 Background

En-route Air Traffic in the United States is managed by designated Federal Aviation Administration (FAA) Air Route Traffic Control Centers (ARTCC). Each of these ARTCCs is sub-divided into sectors, and each sector is managed by Air Traffic Controllers, in charge of monitoring all flight operations. Information is transferred from one sector to a corresponding adjacent sector when a flight moves from this sector to the neighboring sector. In this way the activity of the controllers is coordinated and a flight is continuously monitored as it flies from its origin to its destination.

In order to determine the workload imposed on air traffic controllers, the number of potential conflicts to be resolved can be estimated. The number of flights passing through a sector along with the number of potentially conflicting flights will determine, to a first order, the workload of the sector.

The importance of workload analysis and determining sector occupancies takes more relevance in a Free Flight scenario, where a flight is free to choose optimal waypoints rather than relying on fixed ground navigational aids such as VOR (Very High Frequency, Omni-Directional Range). In such a system, the FAA needs to approve the flight plans to minimize any network congestion that might impose excessive workloads on Air Traffic Controllers (ATC).

The study of sector occupancy and accompanying workloads is also needed in a scenario where an airspace is blocked, and the flights need to be rerouted. This may cause excessive workloads on the sectors around Special Use Airspace (SUA) boundaries. In this case, it is imperative to reschedule or detour flights in an optimal fashion so as

to avoid overcrowding of sectors.

This thesis deals with the development of a computer model to study airspace sector occupancy. Given the sector geometry and flight schedules, the model developed determines the occupancy of sectors, a characteristic measure of workload.

## **1.2 Air Traffic Operations**

The entire airspace over the United States is divided into twenty-one centers, each regulated by an Air Route Traffic Control Center (ARTCC). Each of these centers is subdivided into sectors. Sectors are classified into three groups: low, high and super-high sectors depending upon the floor and ceiling boundaries. Low sectors lie below the FL 240 (i.e. flight level 24,000 ft). High sectors extend between FL 240 and FL 350. The super-high sectors lie above FL 350.

Air traffic operations are monitored by air traffic controllers, having assigned duties pertaining to a particular sector. Air traffic controllers keep an eye on the radar display and communicate with the pilots in order to resolve any potential conflicts. Controllers coordinate their activities with their counterparts in adjacent sectors so that the monitoring of flight operations is smooth and continuous. The workload imposed on the air traffic controllers will depend on the number of flights crossing the sector at any instant of time, the number of potentially conflicting flights, the level of ATC equipment automation, and the conflict geometry of each conflicting flight pair. Human factor parameters such as conflict detection time, conflict resolution strategy, and secondary task loading also play a role in workload assessment.

### **1.3 Free Flight**

Free Flight is the way in which future air traffic will operate. Free Flight operations will be mainly governed by communications, navigation, and surveillance information transmitted through satellites, using advanced on-board navigation equipment and transponders. The existing ATC system establishes aircraft positions (i.e. surveillance function) through ground based radar equipment. In the current system, navigation is also dependent upon ground navaids, and communications are based on a hybrid of Very High Frequency (VHF) line-of-sight and satellite based techniques. In Free Flight, the pilot will receive real-time information regarding nearby flights, and on-board traffic advisories will provide cues required for air traffic control separation. This way, a decentralized air traffic control service could be provided. In a critical situation, the air traffic controller may interfere to resolve the conflict. The main motivation behind Free Flight is that the airlines will have more flexibility in filing their flight plans using point-to-point routes without reliance on ground navaids. This will result in more efficient and cost effective flight trajectories. The FAA will still have some degree of oversight to approve these flight plans making sure that they will not impose excessive workload on any of the control centers.

### **1.4 Impact of Special Use Airspace (SUA) on Air Traffic Management**

Special Use Airspace (SUA) are presently operated independently from the Air Traffic operations. Some SUAs, for example the ones around military bases or strategic

areas, will always be blocked for commercial air traffic operations. Few other SUAs such as those around spaceports (i.e. spacecraft launch sites) will be blocked during certain periods of time as needed. During this time, the flights will have to be re-routed around the blocked region or delayed at departure. The rescheduled flights could impose additional workloads on the sectors neighboring the Special Use Airspace (SUA).

Currently, a satellite is launched about once every 15 days from the Kennedy Space Center (KSC). Launch operations are expected to increase and this will require a better integration of the air traffic and SUA operations. This thesis is part of an ambitious model to efficiently plan optimal air traffic management strategies that minimize sector workloads, and at the same time, provide optimal detours around SUA.

## **1.5 Research Scope, Objective and Approach**

In the future Air Traffic Management System, it is imperative to have a model to assess the workloads imposed on the different sectors by flights traversing the airspace. Such a model may serve as an advisory tool for FAA to approve flight plans in the Free Flight scenario or to reschedule flights around the SUA areas such as in the event of a launch of satellites.

The objective of this research is to develop a tool which will determine the occupancy of Air Traffic Control sectors for a given geometry of sectors and flight schedules. A computer model has been developed for this purpose using Matlab 5.2. The model named Airspace Sector Occupancy Model (ASOM) can be run on any IBM compatible PC, Macintosh, or Unix Workstation without modifications.

## **2. LITERATURE REVIEW**

### **2.1 Introduction**

This chapter deals with a review of various workload models as applied to air traffic control tasks. We begin by defining workload, and then discuss the different models used to measure workload and examine the relationship between sector occupancy and workload.

### **2.2 Definition of Workload**

Wiener and Nagel (1988) defines workload as the objective task demands imposed on the human operator, the mental effort exerted by the operator to meet these demands, the performance of the operator, the psychophysiological state of the operator, and the operator's subjective perception of the expended effort. Workload reflects the relationship between the environmental demands imposed on the human operator and the capabilities of the operator to meet those demands.

### **2.3 Workload and Air Traffic Control Performance**

An air traffic controller's workload is directly related to his/her ability to manage traffic efficiently. Wyndemere, Inc, (1996) has made a comprehensive study of the complexity of air traffic control tasks. The greater the number of aircraft to be handled, the greater is the workload. A high workload will negatively impact the controller's performance and also set an upper limit on traffic handling capacity. Decreasing the size

of the sectors or increasing the number of controllers do not necessarily solve the problem because of the consequent increase in inter-sector and inter-controller coordination and communication. Decreasing the sector size also reduces the amount of time spent on each aircraft and so, the controller will have less cognitive time to assess the traffic situation. While a greater number of aircraft to be handled increases workload, a low traffic load may result in boredom and reduce alertness, and this may in turn reduce the controller's ability to handle emergencies. In fact most of the operational errors in the National Airspace System in the US (NAS) are made under light traffic conditions following a large surge of traffic.

## **2.4 Modeling Workload**

Workload assessment models are necessary for predicting the effect of new systems and new hardware and software capabilities which are proposed to be introduced in NAS. Some of the models used for this purpose are briefly described below.

### ***2.4.1 Time-Line Analysis***

In the time-line analysis model, workload is modeled as a function of the proportion of the time spent in performing a task relative to the total time available. The workload level during a specific time interval can be determined by summing the time lines of each task performed during that period and dividing by the time interval (National Research Council, 1997).

### **2.4.2 Extended Time-Line Analysis**

The conventional time-line analysis was extended by Lauderman and Palmer (1995) by using a linear function of increasing workload during the time available for completing a task. In this model, the function begins at zero before the task is attempted and returns to zero after the task is completed. By summing the workload function for each task, a predicted workload profile over time is obtained. The area under this overall workload function is proposed as an index of workload.

### **2.4.3 Kip Smith Dynamic Density Metric Analysis**

Dynamic density is a function of the factors that impact the cognitive complexity of controlling an air traffic situation. It provides a measure of perceived complexity of the situation and the mental workload on the controller. The Kip Smith dynamic density metric is a product of an FAA sponsored dynamic density research effort conducted by Drs. K. Smith, S.F. Scallen, W. Knecht, and P.A. Hancock (1998). This metric focuses on separation as the single most important factor in estimating collision risk and is given as follows.

$$K_t = \sum_{i=1}^{N-1} \sum_{j=i+1}^N \frac{1}{\left(\frac{d_{ij}(t)}{c}\right)^a}$$

where

$N$  - number of aircraft

$d_{ij}$  - distance between two aircraft not separated by altitude

$c$  - normalization constant, equal to 5 nmi., the minimum allowable lateral separation by current FAA regulations

$a$  - weighting parameter, set to 3 for a reasonable balance between the relative contributions of close and distant aircraft

#### **2.4.4 Laudeman's Dynamic Density Metric Analysis**

Laudeman's dynamic density metric is the result of research activity conducted by Dr. Irene Laudeman at the NASA Ames Research Center. This metric incorporates nine traffic factors, uses two-minute time increments and a twenty-minute projection of future aircraft positions. Two flights are predicted to be in conflict if the separation between them is going to be less than 5 nmi in a twenty-minute projection into the future.

$$D_t = N_t + N_{\Delta h(t-2,t)} + N_{\Delta V(t-2,t)} + N_{\Delta \psi(t-2,t)} + N_{out(t-2,t)} + N_{\epsilon(t) < 5, d(t+20) > 5} + N_{\epsilon(t) \in (5,10), d(t+20) > 5} + N_{d(t) < 25, d(t+20) < 5} + N_{d(t) \in (25,40), d(t+20) < 5} + N_{d(t) \in (40,70), d(t+20) < 5}$$

where

- $N_t$  - count of all aircraft currently in the sector (cell).
- $N_{\Delta h(t-2,t)}$  - count of aircraft found to have changed altitude during the previous two minutes.
- $N_{\Delta V(t-2,t)}$  - count of aircraft found to have changed speed during the previous two minutes.
- $N_{\Delta \psi(t-2,t)}$  - count of aircraft found to have changed heading during the previous two minutes.
- $N_{out(t-2,t)}$  - count of aircraft whose mean aircraft speed in sector is 150 knots.
- $N_{\epsilon(t) < 5, d(t+20) > 5}$  - count of aircraft pairs (not predicted to be in conflict) whose Euclidean distance is less than 5 nm.

$N_{\epsilon(t) \in (5,10), d(t+20) > 5}$  - count of aircraft pairs (not predicted to be in conflict) whose Euclidean distance is between 5 and 10 nm.

$N_{d(t) < 25, d(t+20) < 5}$  - count of aircraft pairs predicted to be in conflict with current range is less than 25nm.

$N_{d(t) \in (25,40), d(t+20) < 5}$  - count of aircraft pairs predicted to be in conflict with current range between 25 and 40 nm.

$N_{d(t) \in (40,70), d(t+20) < 5}$  - count of aircraft predicted to be in conflict with current range between 40 and 70 nm.

Threshold for altitude change: 750 ft over a 2 minute interval.

Threshold for speed change: 10 knots over a 2 minute interval.

Threshold for heading change: 10 degrees over a 2 minute interval.

This metric is very sensitive to trajectory prediction accuracy and uses inputs from the CTAS system for aircraft position and conflict prediction information. As part of a recent validation effort, the dynamic density was computed for operational sectors at the Denver ARTCC. The validation data collected regarding controller activity will be used to determine what weights (if any) provide the best fit of the data. Presently, the metric is considered simply as a sum of the nine traffic factors. This metric was used by CSSI Inc., for their study of the impact of Free Flight on the ATC System (1997).

## 2.5 Occupancy and Workload

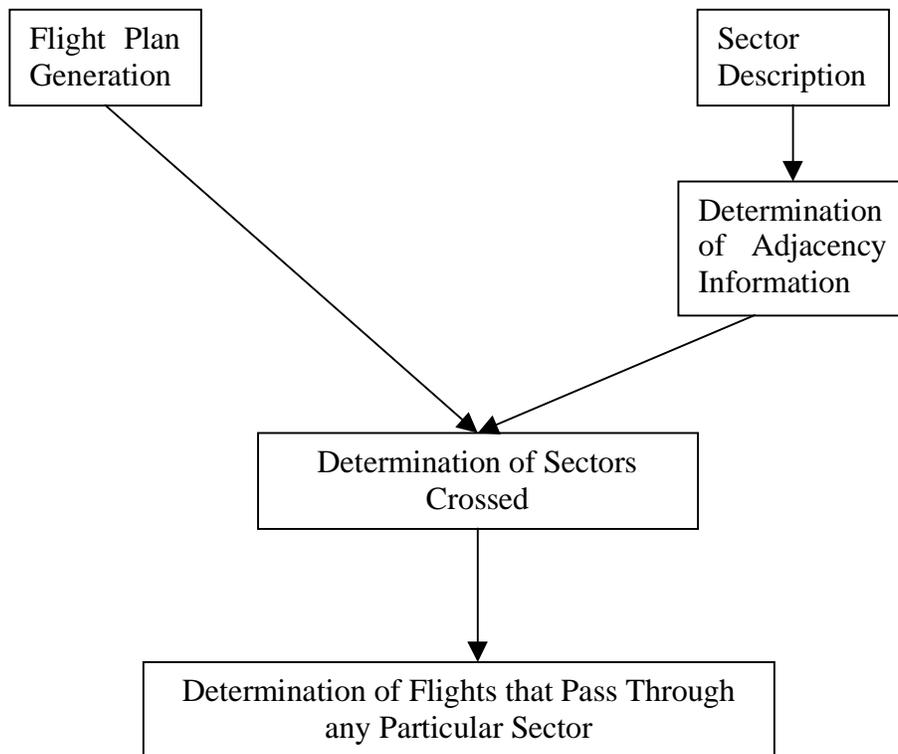
It may be inferred from the different workload models that the number of flights occupying a sector at a given instant of time will determine, to a first order, the workload

on the sector. The number of aircraft being handled by a controller serves as a prelude to assess the load on the controller. This variable, along with other factors such as traffic complexity (number of potentially conflicting flights) and weather, determine the demand imposed on the controller (FAA, 1997). Hence, a model that can determine the occupancy of the sectors and provide the number of flights in a particular sector at any instant of time will be useful to study the workload imposed on the controllers for the given flight schedule scenario. This work is being complemented with a more detailed analysis of aircraft trajectories to identify the geometry of each encounter.

### 3. MODEL METHODOLOGY

#### 3.1 Introduction

The Airspace Sector Occupancy Model (ASOM) requires a series of aircraft flight plans and the sector geometry as inputs. The model processes the information to determine the occupancy of each sector by different flights over time. The essence of the model lies in storing the adjacency information of sectors, and identifying the sectors crossed by a flight plan. The overall process of the computer model can be depicted as shown in Figure 1.



**Figure 1 Overall Process of the Airspace Sector Occupancy Model (ASOM).**

## **3.2 Flight Plan Generation**

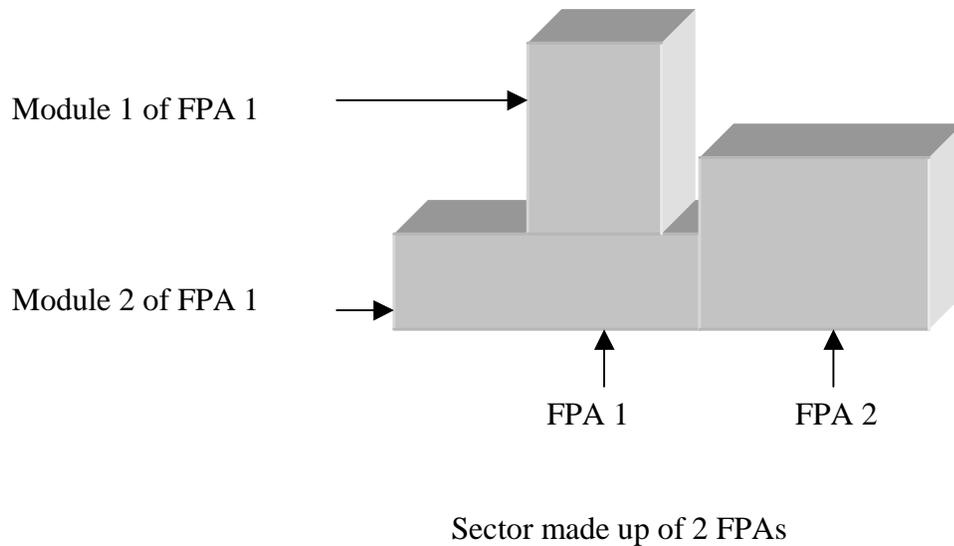
The flight plan for our analysis may either have been filed for a particular day in the past, or might be a hypothetical predicted flight plan for the future. The flight plan should carry the following information.

1. Way-points in latitude (degree), longitude (degree) and altitude (hundreds of feet).
2. Time tags corresponding to the crossing of each of the above way-points (in any time unit).
3. The originating airport (a three letter airport designator). (Optional)
4. The destination airport (a three letter airport designator). (Optional)

The flight plans for any particular day in the past can be obtained from the FAA En-route Traffic Management System (ETMS) database or Sector Design and Analysis Tool (SDAT) database. In order to use the model to analyze predicted air traffic, an independent flight generator that develops flight plans having the above mentioned five attributes, could be coupled with the Airspace Occupancy Determination Model.

## **3.3 Sector Description**

Sectors are well-defined airspace regions specified by the FAA for regulating air traffic. Each sector is comprised of Fixed Point Airspace units (FPA) and each of these FPAs is made up of modules. A module is an airspace, that is a convex or non-convex polytope in shape, defined by its vertices, its floor and ceiling altitudes. Modules are stacked one over another to form an FPA, and several such adjacent FPAs form a sector as shown in Figure 2.



**Figure 2 Typical Sector Geometry.**

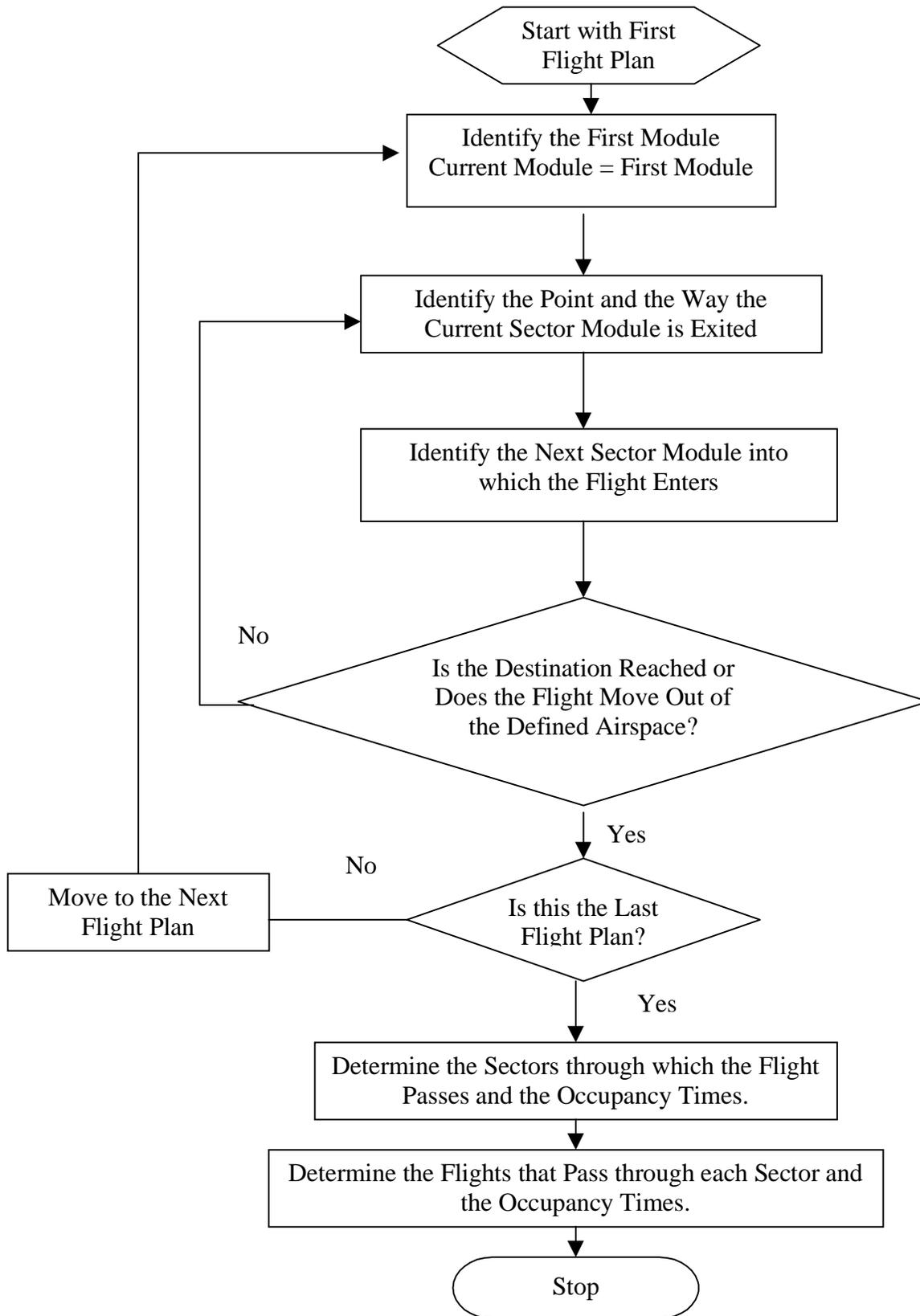
### 3.4 Occupancy Determination

A flight that crosses a sector will be detected by the model based on the adjacency information, that is generated and stored during the pre-processing step. Since each sector is complex in shape, the analysis is done at the module level and the result is translated to the sector level by considering the particular modules that make up the sector.

The model first identifies the initial module encountered by the flight. This may be the module that encompasses the originating airport. Sometimes, the originating airport may not lie within the defined modules. In such a case, the model identifies the module through which the flight enters the defined airspace.

Once the initial module through which the flight passes is detected, the point and time of exit is identified. This point is found by checking if the flight crosses any of the

faces or the floor or the ceiling defining the module, without merely glancing it and remaining within the same module. The program also identifies the way the flight exits the module, i.e., if the flight exits across a face, or the floor, or the ceiling, or at a vertex, or across an edge. With this knowledge, and since module adjacency information is known, the next module into which the flight enters is determined. This process of identifying each traversed module and the corresponding occupancy time is continued until the flight reaches its destination. Next, the sectors through which the flight passes is identified by examining the modules that comprise each sector. This provides information on all the flights that cross a particular sector along with related occupancy times. A flow chart illustrating the sector occupancy determination process is shown in Figure 3.

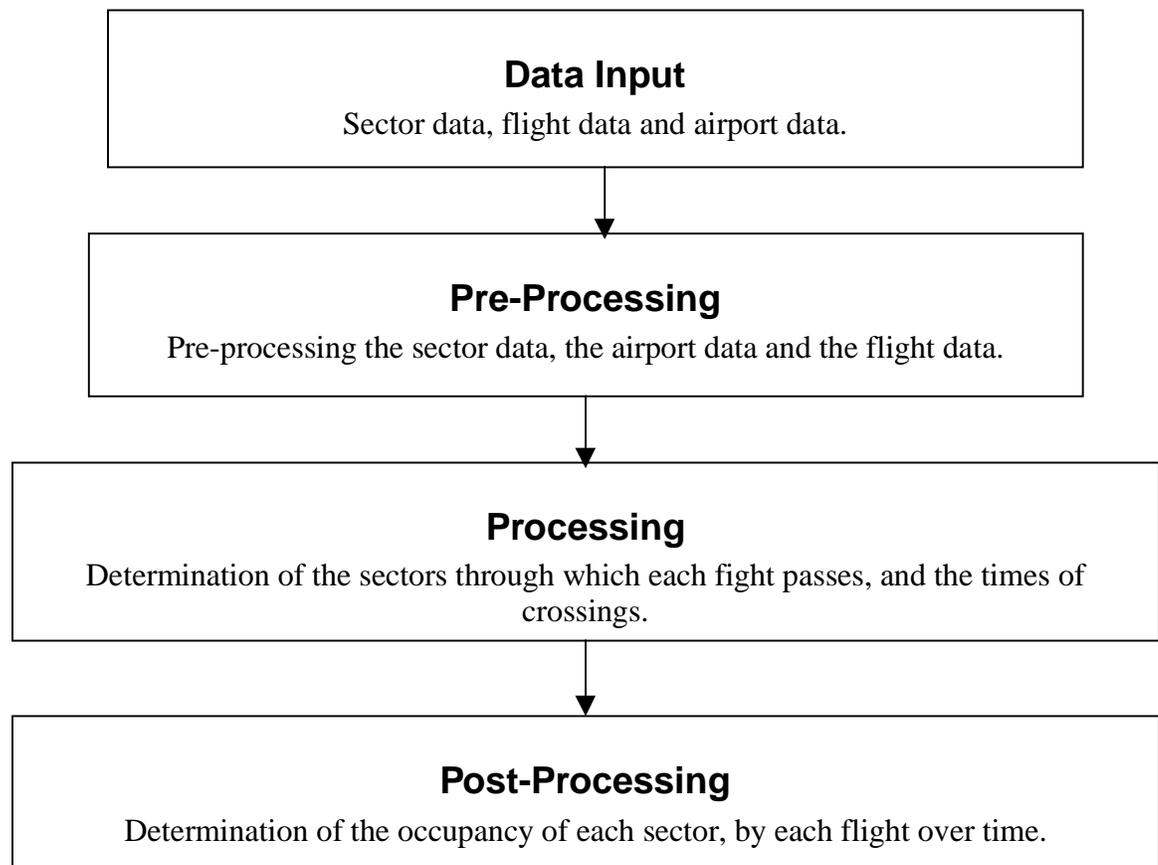


**Figure 3 Occupancy Determination Methodology.**

## 4. MODEL DESCRIPTION

### 4.1 Occupancy Model Framework

The working of the Occupancy Model can be categorized broadly into four steps namely data input, pre-processing, processing, and post-processing. These steps are shown graphically in Figure 4.



**Figure 4 Occupancy Model Framework.**

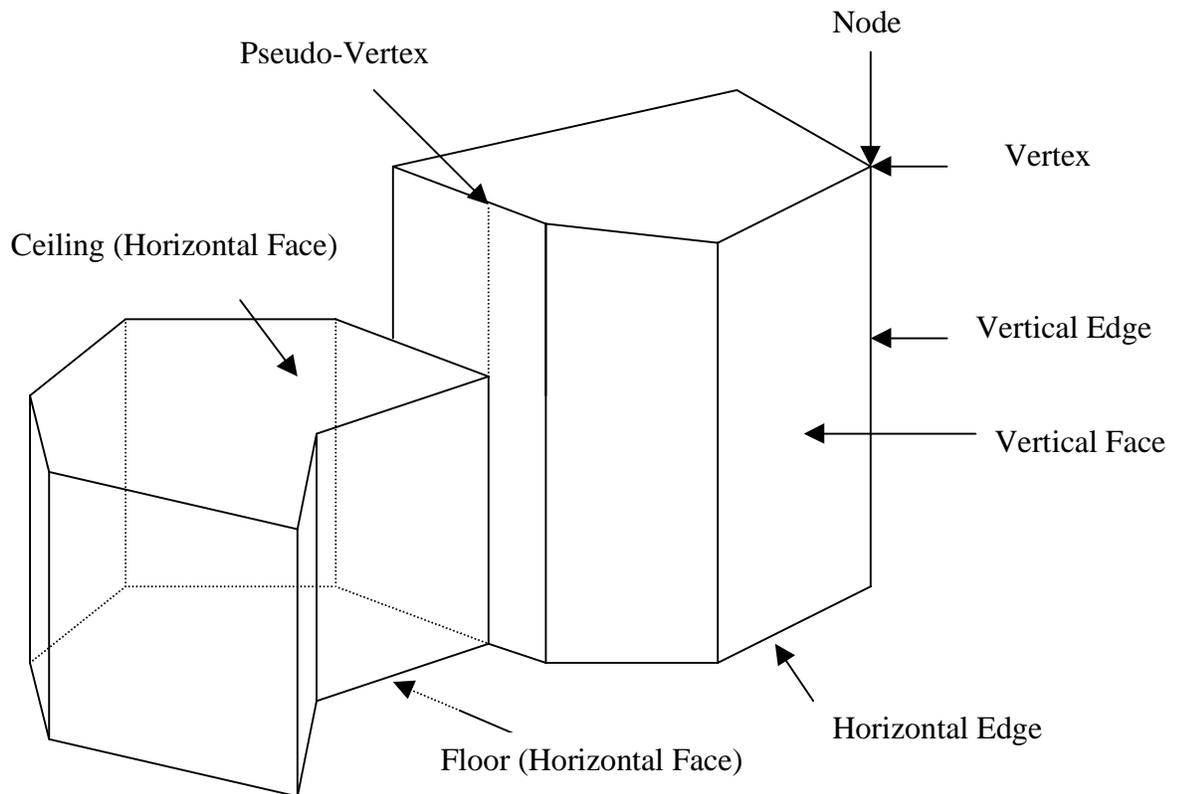
## Definition of Terms

### Sector Module.

A sector module is a fundamental unit in the definition of an airspace. One or more sector modules form a sector. A sector module is a three dimensional convex or non-convex polytope in shape.

### Vertical Faces

These are the rectangular, two dimensional, vertical bounding faces that define a sector module as shown in Figure 5.

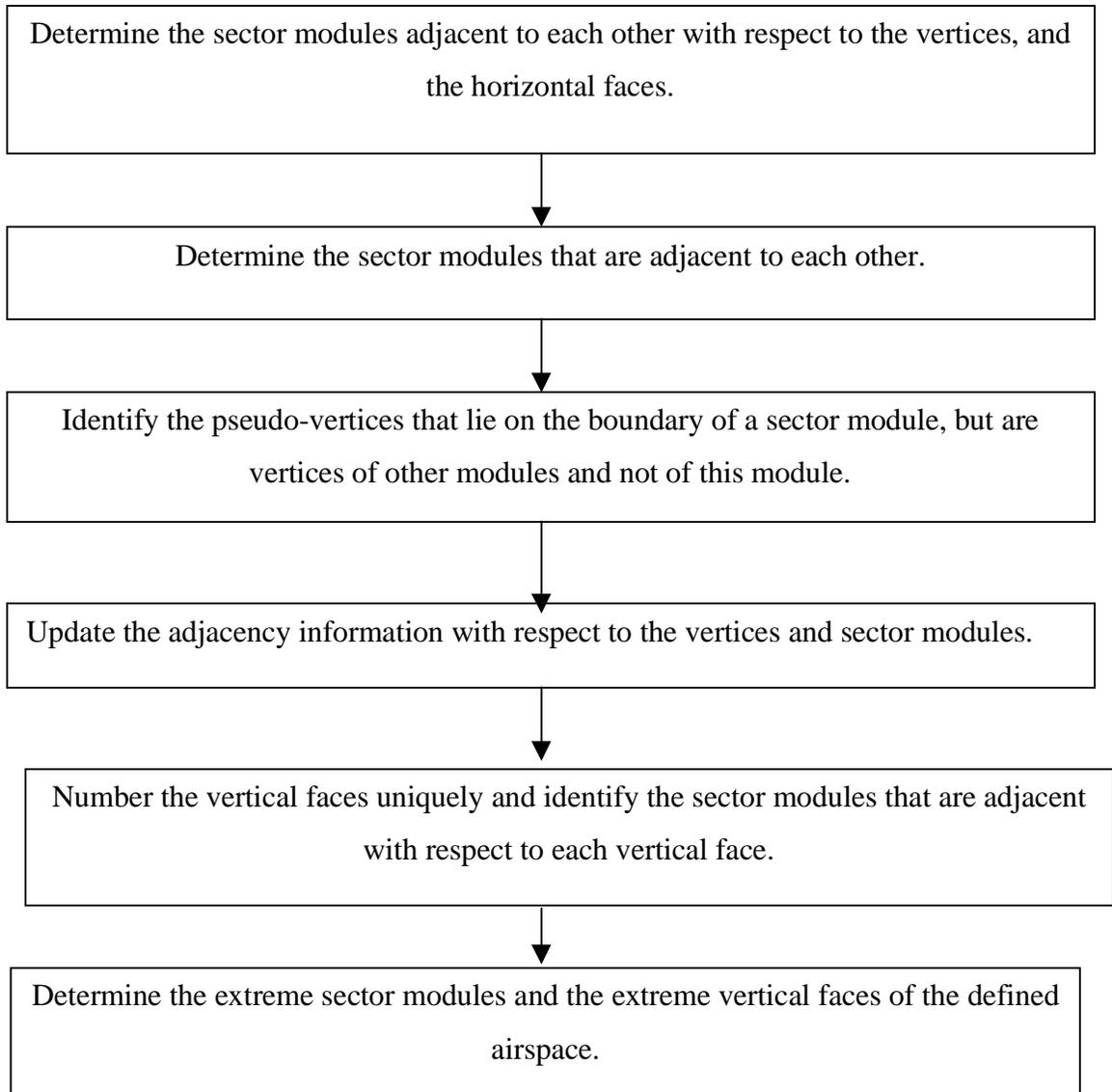


**Figure 5 Components of a Sector Module.**

<b>Floor</b>	This is the lower horizontal face of a sector module.
<b>Ceiling</b>	This is the top horizontal face of a sector module.
<b>Vertex</b>	A vertex is a corner point of a sector module.
<b>Pseudo-Vertex</b>	A pseudo-vertex for a sector module, is a vertex for some other sector module that is present on a vertical face of the given sector, but is not an original defining vertex of its floor and ceiling.
<b>Vertical Edge</b>	This is the line of intersection of two adjacent vertical faces of a sector module.
<b>Horizontal Edge</b>	This is the line of intersection of the floor or ceiling with a vertical face.
<b>Node</b>	A node corresponds to a corner point formed by the two dimensional projection of a module onto its floor or ceiling. It is used to define the floor and ceiling geometry of a sector module, and might correspond to the projection of one or more vertical edges along with associated vertices belonging to adjacent modules.
<b>Extreme Sector Module</b>	These are the sector modules that lie along the boundaries of the defined airspace.
<b>Extreme Vertical Faces</b>	These are the vertical faces of the extreme sector modules that form the boundary of the defined airspace.

### 4.3 Pre-processing Sector Data

The pre-processing of the sector data involves the steps shown in Figure 6. The analysis is done at the module level and later, the occupancy information is aggregated to the sector level.

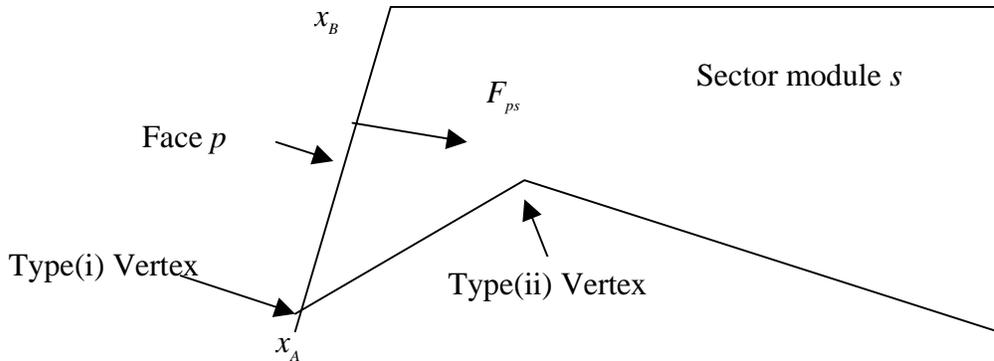


**Figure 6 Pre-processing of Sector Data.**

All modules are represented in terms of their vertices, and the equations of the vertical faces (determined by the pre-processing routine) represented in the form  $\alpha \bullet X - c = 0$ , where  $\alpha$  is a normalized vector and  $c$  is the distance of the face from the origin in the direction of  $\alpha$ . The adjacency information with respect to the faces and vertices is determined and stored during pre-processing.

#### 4.3.1 Inward Gradient

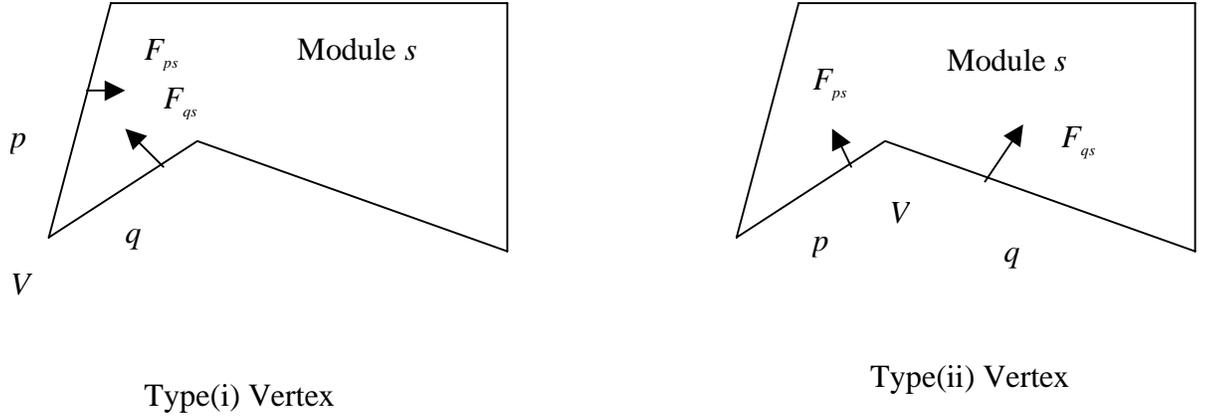
Consider a two dimensional projection of a module. (A projection will always refer to a collapsing of the module in the vertical direction into a 2-D polygon.) An inward gradient  $F_{ps}$  for a face  $p$  of a projected sector module  $s$  is that gradient vector orthogonal to the face such that a trajectory which starts at an interior point of this face  $p$  and moves in a direction  $d$ , will reside in module  $s$  for some positive distance if and only if  $F_{ps} \bullet d \geq 0$ . It was noticed that the FAA Sector Analysis and Design Tool (SDAT) data contains the coordinates of the vertices for all the modules in a clockwise sequence. Hence for any pair of vertices  $x_A$  and  $x_B$  defining the face  $p$  as shown in Figure 7, if the direction along the face is  $d_p = x_B - x_A = [d_{p1}, d_{p2}]$ , then the inward gradient  $F_{ps}$  is given by  $F_{ps} = [d_{p2}, -d_{p1}]$ .



**Figure 7 Representation of a Module.**

### 4.3.2 Types of Vertices

Each vertex is classified as type(i) or type(ii), based on its associated faces  $p$  and  $q$ , as described below and depicted in Figure 8.



**Figure 8 Types of Vertices.**

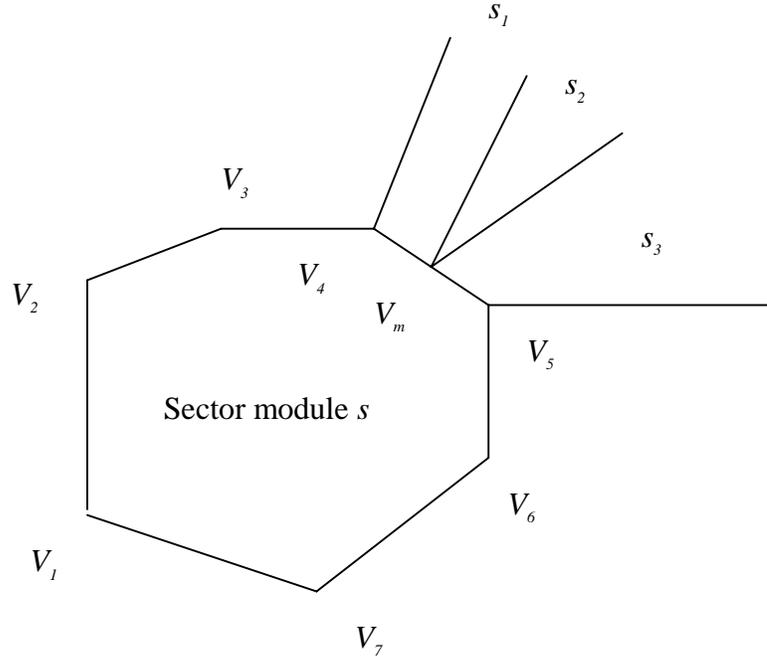
Type(i): Here the local neighborhood of the vertex is described by the conjunction of the faces  $p$  and  $q$ . Hence, if a trajectory starts at this vertex and moves in a direction  $d$ , then it would reside in module  $s$  for some positive step if and only if  $F_{ps} \bullet d \geq 0$  and  $F_{qs} \bullet d \geq 0$ .

Type(ii): Here, the local neighborhood of the vertex is described by the disjunction of the faces  $p$  and  $q$ . Hence, if a trajectory starts at this vertex and moves in a direction  $d$ , then it would reside in module  $s$  for some positive step if and only if  $F_{ps} \bullet d \geq 0$  or  $F_{qs} \bullet d \geq 0$ .

### 4.3.3 Adjacency with Respect to Nodes

Consider a node  $V_m$  as shown in Figure 9, which might correspond to a real or a pseudo-vertex. All the sector modules which have  $V_m$  on the boundary of their two dimensional projections are considered to be adjacent with respect to  $V_m$  and are stored in

the record  $Adjsecnode(m).sect$ . The pre-processing step will identify if there is any sector module  $s$  that contains the node  $V_m$  internally on a face of its two dimensional projection, and the program will then recognize  $s$  in terms of  $V_m$  and other defined nodes for  $s$ .



**Figure 9 Adjacency with Respect to a Vertex.**

In Figure 9, the originally defined nodes for  $s_m$  are  $[ V_1 V_2 V_3 V_4 V_5 V_6 V_7 ]$ . After preprocessing, the sector module  $s$  will be recognized in terms of the nodes  $[ V_1 V_2 V_3 V_4 V_m V_5 V_6 V_7 ]$ . The sector modules  $s, s_1, s_2$  and  $s_3$  will be considered to be adjacent with respect to  $V_m$  and this information will be stored in the record  $Adjsecnode(m).sect$ .

#### **4.3.4 Adjacency Information with Respect to Sector Modules**

Sector modules adjacent to other sector modules are identified and stored during the pre-processing step. The adjacency information with respect to the nodes is used to identify this adjacency information. For a sector module  $s$ , let  $V_s$  be the set of nodes defining its floor and ceiling. Then, all the sector modules that share any node in  $V_s$  will

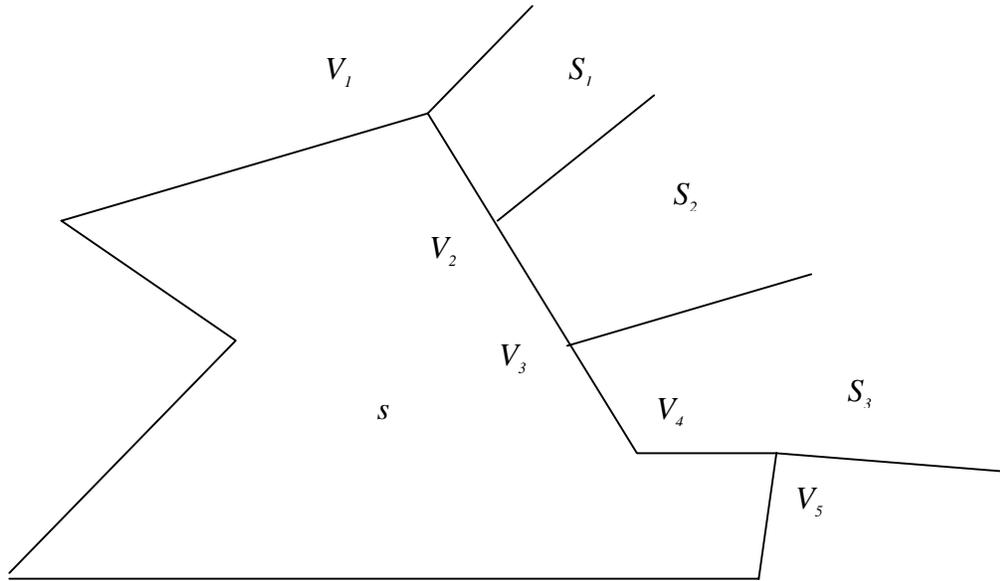
be adjacent to  $s$  if they extend in part or whole over an altitude between the floor and ceiling of sector module  $s$ . The main purpose of storing this adjacency information is to determine the nodes that lie around a projected sector module. Later, all nodes are checked to see if they lie on projected vertical faces of modules while not being defined as its original nodes.

#### **4.3.5 Identifying Pseudo-Vertices**

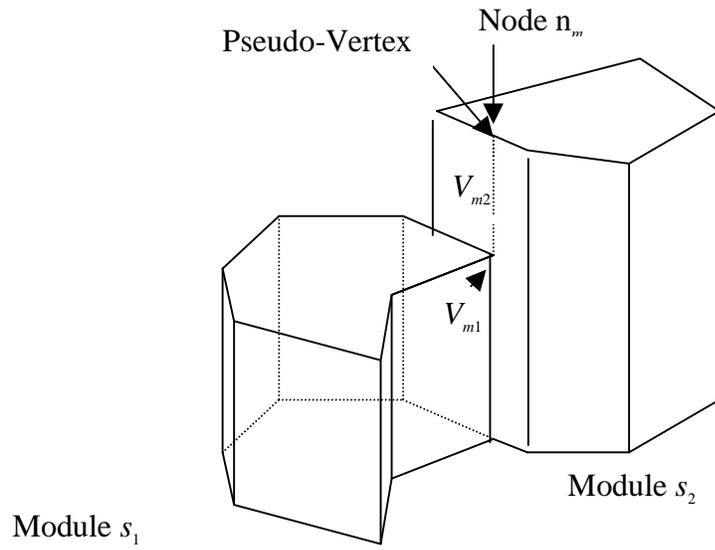
Consider the sector modules shown in Figure 10. Nodes  $V_2$  and  $V_3$  lie on the projected vertical face of module  $s$ , but are not defining nodes of the floor and ceiling of module  $s$ . Since the occupancy model makes use of the adjacency information in order to determine the next sector module into which a given flight enters after exiting a previous sector module, it becomes necessary to (a) identify nodes such as  $V_2$  and  $V_3$  as corresponding to pseudo-vertices of a sector module  $s$  and (b) to redefine its floor and ceiling faces in terms of all original as well as such pseudo-vertex induced nodes.

In order to identify such nodes, a check is made for all the nodes lying around a sector module  $s$  to see if any lies on a projected vertical face of  $s$ . The nodes which lie around a sector module  $s$  are determined from the sectors that are adjacent to it.

Figure 11 illustrates an example of a pseudo-vertex in a three dimensional depiction.  $V_{m1}$  is a real vertex defining the floor and ceiling of the sector module  $s_1$ . This induces a pseudo-vertex  $V_{m2}$  for Psector module  $s_2$ . Both  $V_{m1}$  and  $V_{m2}$  correspond to the same node  $n_m$  and so sectors  $s_1$  and  $s_2$  will be considered adjacent with respect to node  $n_m$ .



**Figure 10 Pseudo-Vertices on a Face.**



**Figure 11 Pseudo-Vertices on a Vertical Face.**

### **4.3.6 Adjacency Information with Respect to Vertical Faces**

During the pre-processing step, the occupancy model stores sector modules that are adjacent to each other with respect to a given projected vertical face. This is done after identifying all the pseudo-vertices and revising the adjacency information of sector modules with respect to the nodes and modules. The projected vertical faces are distinguished from each other based on their defining end nodes. For any projected vertical face  $p$  having defining end nodes  $V_1$  and  $V_2$ , (including the pseudo-vertex induced nodes), all the sector modules that contain the nodes  $V_1$  and  $V_2$  are considered adjacent with respect to it. These sector modules can be determined from the adjacency information with respect to the nodes. The model also classifies the sector modules that are adjacent with respect to a particular vertical face into two categories based on whether the sector module lies on the side towards the origin (equator on Greenwich meridian) or on the side opposite to the origin. This additional information will be used to identify the extreme vertical faces. These extreme faces either define the external boundaries of the defined airspace, or the vacuums that may be present in the airspace.

## **4.4 Pre-processing Airport Data**

The pre-processing routine also associates each airport with the sector module that contains it. This is done by checking if the airport lies in any of the low lying sector modules. The built-in Matlab function *inpolygon* is used to check if a point lies within a polygon. If the airport lies outside the defined airspace, no sector module will be associated with it.

## **4.5 Pre-processing Flight Plans**

This pre-processing routine identifies the first sector module that a flight trajectory encounters. It also records the entry point and the time of entry. If the originating airport lies within the defined airspace, the identification process will be trivial. If the flight originates outside the defined airspace, the point of entry and the first module entered will be determined by checking each of the flight segments for a possible crossing of an extreme face of the defined airspace. Dummy sectors are defined in order to speed up the computation in this step. This is elaborated below.

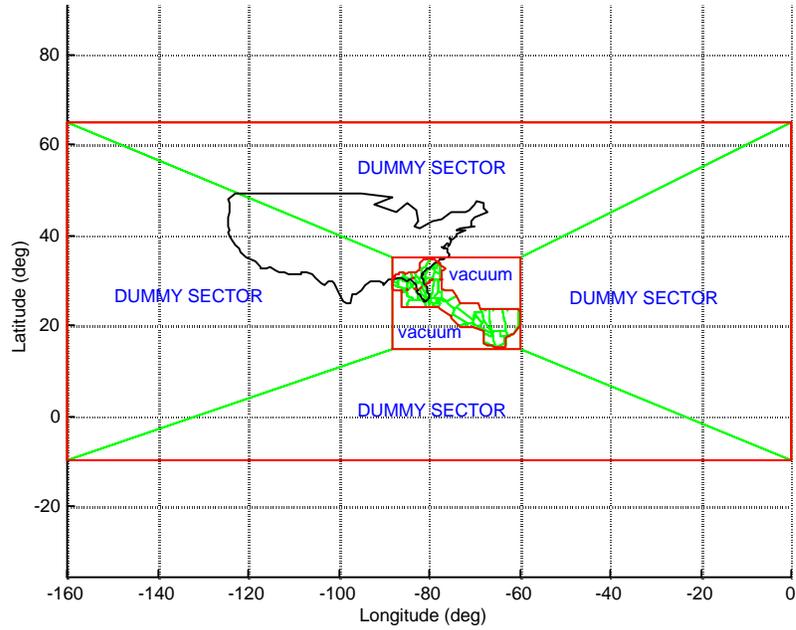
### **4.5.1 Dummy Sectors**

During the initialization step, the first module which each flight encounters is determined. If the origin airport does not lie in the defined airspace, then the program will move along the flight trajectory, segment by segment, to identify that flight segment which crosses any extreme face of the defined airspace. Since this is computationally intensive, dummy sectors are defined as shown in Figure 12 around the given airspace so that the airports of concern lie within this extended airspace. This cuts down the search during the initialization step drastically.

### **4.5.2 Vacuums**

The dummy sectors defined around the defined airspace under consideration are trapezoidal polytopes. Within the rectangular region formed by these sectors, that contains the airspace under consideration, there exists an undefined airspace. This space is termed as the vacuum airspace. The program will handle the case of a flight passing through this vacuum and identifies its entrance into the defined airspace, if at all or its re-

entrance into the dummy trapezoidal polytopes.



**Figure 12 Dummy Sectors and Vacuums.**

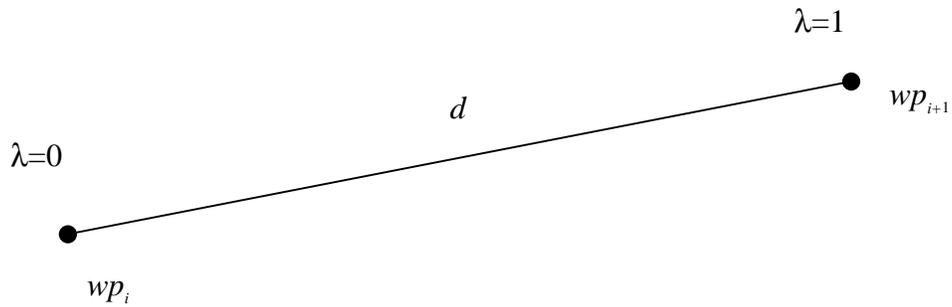
In addition to this deliberately created vacuum in between the dummy sectors and the actual sectors under consideration, there may be instances of vacuums being present between actual sectors because of inaccuracies in sector definitions. The program can also handle such instances efficiently as explained in Section 4.6.2.

## 4.6 Occupancy Determination Algorithm

The algorithm for determining sector module occupancies is first described for a projected two dimensional case, and is later extended to three dimensions as explained in Section 4.6.3.

### 4.6.1 Algorithm for a Two Dimensional Case

Consider a flight path that is comprised of linear discretized flight segments represented in terms of the coordinates of way-points. Such a flight path will be represented as  $[wp_1, wp_2, \dots, wp_p, \dots, wp_n]$ .



**Figure 13 Representation of a Flight Segment.**

Let any linear segment of the trajectory as shown in Figure 13 be defined as  $x = wp_i + \lambda d$  for  $0 \leq \lambda \leq 1$  where  $d = wp_{i+1} - wp_i$ .

Suppose that for  $wp_i$  we know the sector module  $s$  in which the current point lies, and its actual location in this sector module (interior point, interior of a face or at a vertex). This is initially determined during the pre-processing routine, and is sequentially deduced by the algorithm as explained below.

**Initialization:**

Set  $x_0 = wp_i$ ; current point  $x = x_0$ ;  $\lambda = 0$  and  $d = wp_{i+1} - wp_i$ . Let  $s$  be the sector module in which  $x$  lies.

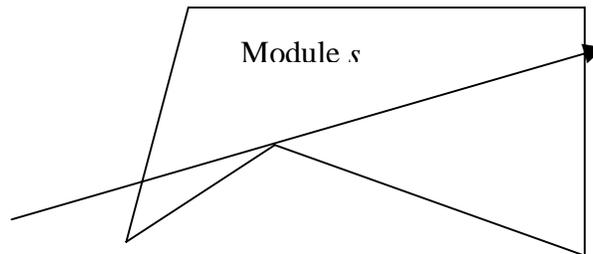
**Step 1: ( Determination of Exit Point)**

Examine the faces of the sector  $s$  and find a first face that the straight line trajectory  $x + \lambda d$  intersects (internally or at a vertex of a face) at  $\lambda = \lambda^*$ .

Let  $\lambda_{new} = \lambda + \lambda^*$  and  $x_{new} = x_0 + \lambda_{new} d$ .

Go to Step 2.

(Note that the occupancy of module  $s$  can continue in case we have just internally glanced a vertex at as shown in Figure 14, and this will be automatically determined in the next loop of this procedure.)



**Figure 14 Flight Internally Glancing a Vertex Point.**

### **Step 2: (Checking the Processing of Linear Segments)**

If  $\lambda_{new} < 1$ , record the occupancy in the interval  $[\lambda, \lambda_{new}]$ . Set  $x = x_{new}$  and  $\lambda = \lambda_{new}$ , and proceed to Step 3.

Else, if  $\lambda_{new} = 1$ , record the occupancy in the interval  $[\lambda, \lambda_{new}]$ . Stop if  $i + 1 = n$ . Else, proceed to the next linear segment of the trajectory by incrementing  $i$  by 1 and transferring to Step 3.

Else, if  $\lambda_{new} > 1$ , record the occupancy in the interval  $[\lambda, 1]$ . Stop if  $i + 1 = n$ . Else, proceed to the next linear segment of the trajectory by incrementing  $i$  by 1 and returning to Step 1.

### **Step 3: ( Determination of the Next Sector Module)**

Determine the next sector module into which the flight enters based on the adjacency information and replace  $s$  by this module. Return to Step 1.

In this way, all the sector modules that the flight passes through are sequentially determined. The above algorithm makes an assumption that the flight will enter another sector module after it exits one. However, in the definition of airspace, there may be a case where two neighboring sector modules may not be close enough to share any common vertices. This will result in an undefined airspace “vacuum” enclosed between such sector modules that the flight enters. To accommodate this case, we adopt the following strategy.

#### **4.6.2 Extension of Algorithm to Handle Airspace Vacuums**

The algorithm is extended to incorporate the scenario where a flight may encounter a vacuum in the airspace. During the pre-processing, the program identifies all the vacuums that are present in the airspace and stores the information regarding the vertical faces that surround such vacuums as explained in Section 4.2.6. In Step 3 of Section 4.6.1, if the program is not able to identify the sector module that the flight enters based on the adjacency information, it will realize that the flight has entered into a vacuum. The flight's segments are then checked to see when and if they cross any of the extreme faces. Based on the extreme face encountered, the program will identify the sector module entered and then proceed as usual.

As explained in Section 4.5.2, such instances of vacuums being present between sector modules occur mainly because of inaccuracies in sector definitions. In order to correct for inaccuracies, nodes having slightly perturbed locations are assumed to define the same point. This circumvents the creation of such vacuum airspaces.

#### **4.6.3 Extension of the Algorithm for the Three Dimensional Case**

The foregoing algorithm is extended to the prevalent three dimensional situation as explained in this section.

##### **Initialization:**

Set  $x_0 = wp_i$ ; current point  $x = x_0$ ;  $\lambda = 0$  and  $d = wp_{i+1} - w_i$ . Let  $s$  be the sector module in which  $x$  lies.

**Step 1: ( Determination of Exit Point)**

Examine the faces of the sector module  $s$  and find the first face (vertical or horizontal) that the trajectory  $x + \lambda d$  intersects (internally, or at an edge, or at a vertex) at  $\lambda = \lambda^*$ .

This procedure is explained in Section 4.6.4.

Let  $\lambda_{new} = \lambda + \lambda^*$ .

$x_{new} = x_0 + \lambda_{new} d$ .

Go to Step 2.

**Step 2: (Checking the Processing of Linear Segments)**

If  $\lambda_{new} < 1$ , record the occupancy in the interval  $[\lambda, \lambda_{new}]$ . Set  $x = x_{new}$  and  $\lambda = \lambda_{new}$ , and proceed to Step 3.

Else, if  $\lambda_{new} = 1$ , record the occupancy in the interval  $[\lambda, \lambda_{new}]$ . Stop if  $i + 1 = n$ . Else, proceed to the next linear segment of the trajectory by incrementing  $i$  by 1 and transferring to Step 3.

Else, if  $\lambda_{new} > 1$ , record the occupancy in the interval  $[\lambda, 1]$ . Stop if  $i + 1 = n$ . Else, proceed to the next linear segment of the trajectory by incrementing  $i$  by 1 and returning to Step 1.

**Step 3: ( Determination of the Next Sector Module)**

Determine the next sector module into which the flight enters as explained in Section 4.6.5 and replace  $s$  by this module. Return to Step 1. If no new sector is encountered, proceed to Step 4.

#### **Step 4: (Determination of the Next Sector Module after passing through a Vacuum)**

Determine the next sector module into which the flight enters by checking for the intersection of the flight segments starting, with the current segment, with all the extreme faces of the defined airspace. This procedure is explained in Section 4.6.6. Update  $x$ ,  $\lambda$  and  $i$  based on the entry point. Return to Step 1. If no sector module is encountered until the last segment, (i.e when  $i = n$ ), the flight terminates in a vacuum. Record this and stop.

#### **4.6.4 Determination of Point of Exit**

Given a sector module  $s$ , the point  $x$  that lies in it, the parameter  $\lambda$ , and the direction  $d$  of the flight path at that point, we need to identify whether the flight will terminate in this sector module, or else, determine the point at which the sector module  $s$  is exited. The following steps are followed for this purpose.

##### **Step 1:**

Identify the vertical faces  $p$  for which,  $F_{ps} \cdot d < 0$ . Among these vertical faces, the ones that are crossed internally or at the boundary by the flight segment are selected, and of these, the one that is closest to  $x$  is the face that may be crossed. Record  $\lambda_{new}$  and  $x_{new} = x_0 + \lambda_{new}d$ . Proceed to Step 2.

##### **Step 2:**

Check if  $x_{new}$  lies within the floor and ceiling of the sector module  $s$ . If not, identify the point on the floor or the ceiling that is crossed and update  $x_{new}$  and  $\lambda_{new}$  that correspond to this new point. Record the following:

1. If the sector module is crossed across the relative interior of a vertical face, record the

- vertical face that is crossed.
2. If the sector module is crossed across the relative interior of a vertical edge, record the vertical edge that is crossed.
  3. If the sector module is crossed across the relative interior of a horizontal face, record the horizontal face that is crossed.
  4. If the sector module is crossed across the relative interior of a horizontal edge, record the horizontal face and the vertical face that contains the edge.
  5. If the sector module is crossed across a vertex, record the horizontal face and the vertical edge that contains the vertex.

#### **4.6.5 Determination of the Next Sector Module after Exiting**

When a flight trajectory is on the boundary of a sector module, it will either be located on the interior of a vertical face, at the interior of a horizontal face, on a vertical edge, on a horizontal edge, or at a vertex. A flight which exits a sector module in one of the above ways, will enter another sector module in one of the same five ways. Table 1 shows the thirteen possible piercing patterns in which an exiting flight can enter a new sector.

Based on the adjacency information and the type of exit, the probable sector modules into which the flight may have entered are selected. From these, the sector module  $s$  that satisfies one of the requirements below will be the one entered.

Case (a) :  $x$  belongs to the interior of a vertical face  $p$  of  $s$  and, then  $F_{ps} \cdot d \geq 0$ .

Case (b):  $x$  belongs to the interior of a vertical edge, as determined by faces  $p$  and  $q$ , and if the node corresponding to this vertical edge is of type(i) for sector module  $s$ , then we have  $F_{ps} \cdot d \geq 0$  and  $F_{qs} \cdot d \geq 0$ , and if it is of type(ii), then we have  $F_{ps} \cdot d \geq 0$  or  $F_{qs} \cdot d \geq 0$ .

Case (c):  $x$  belongs to the interior of the ceiling and the  $z$  component of  $d$  is nonpositive. Alternatively if  $x$  belongs to the interior of the floor, and the  $z$  component of  $d$  is nonnegative.

Case (d):  $x$  belongs to the interior of a horizontal edge, and requirements (a) and (c) are satisfied, where (a) is applied to the corresponding vertical face containing the edge.

Case (e):  $x$  is a vertex, and the corresponding requirements (b) and (c) are satisfied.

If more than one sector module is entered, as when a flight moves along a vertical face or along a horizontal edge, only one of such modules will be considered, with a preference given to the currently occupied module.

**Table 1. Possible Cases of Sector Piercing Patterns.**

		<b>Point of Entry</b>				
		Vertical Face	Vertical Edge	Top or Bottom Face	Top or Bottom Edge	Vertex
<b>Point of Exit</b>	Vertical Face	X	-	-	X	-
	Vertical Edge	-	X	-	-	X
	Top or Bottom Face	-	-	X	X	X
	Top or Bottom Edge	X	-	X	X	-
	Vertex	-	X	X	-	X

**4.6.6 Determination of the Next Sector Module after Passing through a Vacuum**

The pre-processing routine identifies the extreme faces of the defined airspace (including the faces of the dummy sectors). If a flight enters a vacuum (i.e the volume of airspace is not defined by the sectors), it will either re-enter into the defined airspace through one of the extreme faces or will terminate in the vacuum. The sector module entered after passing through the vacuum will be determined by identifying the next

extreme face that is crossed by the flight trajectory. If no extreme face is encountered, the flight terminates in the vacuum. Here an assumption is made that a flight enters a sector only through a vertical face after passing through a vacuum. This is a valid assumption as it is observed that the vacuums, wherever present, are always bounded by vertical faces alone.

## 4.7 Data Structures

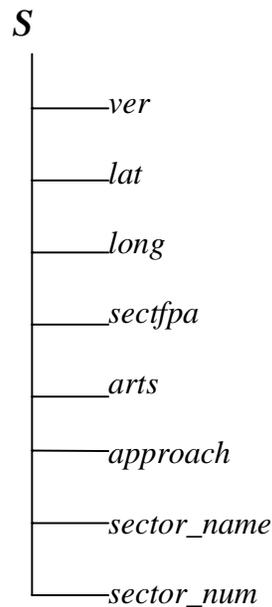
The data structures used in the program to determine the occupancies of the sectors are described in this section.

### 4.7.1 Sector Module Information Structure (S)

*S* stores the information about each sector module. The records of *S* are explained below.

1. *ver* This stores the vertices that define the floor and ceiling of the sector module. These vertices will be arranged in the order as they appear in the input file. The sector information extracted from Sector Design and Analysis (SDAT) contains vertices arranged in a clockwise order.
2. *lat* This contains the latitude of the vertices in the order as they appear in *S.ver*.
3. *long* This contains the longitude of the vertices in the order as they appear in *S.ver*. If a sector module has  $n$  vertices, both *S.lat* and *S.long* will be rows of size  $(n+1)$  where the last value corresponds to the first value. This way of storing the coordinates is useful while plotting the sector using the

Matlab built-in function *plot*. This arrangement is also helpful while using the Matlab function *inpolygon* to check if a point lies within a polygon.



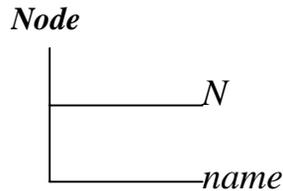
**Figure 15 Data Structure of S.**

4. *name* This is a three letter designation of the sector to which this sector module belongs.
5. *sectfpa* This is a four digit string where the first two correspond to the sector label and the last two correspond to the FPA label.
6. *arts* If the sector module is a part of a terminal approach sector, this field will have a value that corresponds to the type of ARTS equipment available in the sector module.
7. *approach* If the sector module is a part of a terminal approach sector, this field will have a value that corresponds to the approach control pertaining to this sector.

#### 4.7.2 Node Information Structure (Node)

*Node* is a data structure storing the information about the vertices that define the sector modules. Node has two records as explained below.

1. *Name* This is a two dimensional character array storing the name of all the vertices.



**Figure 16 Data Structure of *Node*.**

2. *N* This is a two dimensional array storing the latitudes and longitudes of all the vertices.

#### 4.7.3 Height Information about the Sector Modules (*h*)

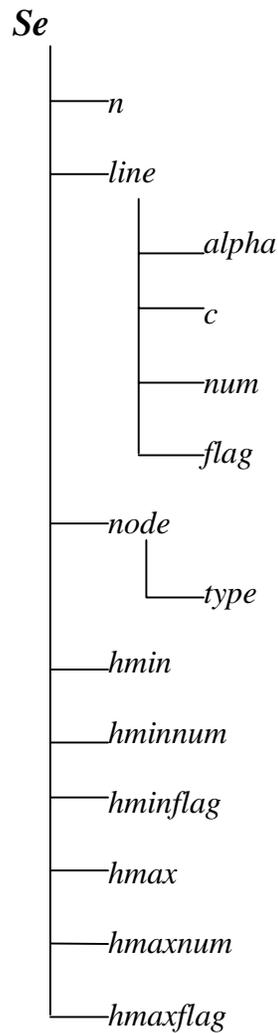
*h* stores the floor and ceiling altitudes of the sector modules. This is a two dimensional array where each row corresponds to the floor and ceiling altitude in hundreds of feet.  $h(i, 1)$  will be the floor altitude of the  $i^{\text{th}}$  sector module and  $h(i, 2)$  will be the ceiling altitude of the  $i^{\text{th}}$  sector module.

#### 4.7.4 Structure with Mathematical Representation of Sector Modules (*Se*).

This data structure stores the mathematical representation of the sector modules.

The records under  $Se$  are shown below.

1. *line* This field defines the equation of each of the vertical faces.  $Se(i).line(1,j)$  gives information about the  $j^{\text{th}}$  face of the  $i^{\text{th}}$  sector module.  $Se.line$  has four records under it.  
*num* This is the number associated with the vertical face.



**Figure 17 Data Structure of  $Se$ .**

- alpha* This is the inward gradient of the vertical face. Determination of the inward gradient is explained in Section 4.3.1.
- c* This corresponds to the normal distance from the origin to the face in the direction of the inward gradient. Hence *c* will be negative if the origin (intersection of equator with Greenwich meridian) lies in the half-space toward the direction of the inward gradient and positive otherwise.  $\alpha \bullet (x, y) = c$  will hold true for any point  $(x, y)$  lying on the face.
- flag* This is a digit that has a value 0 if the face is already numbered and a value 1 if the face is not numbered.
2. *node* The field node is an array structure having two fields *nodenum* and *type* , corresponding respectively to the node number and the vertex type for each of the vertices of a sector module. Determination of the vertex type is explained in Section 4.3.2. The record *Se(k).node(m)* will have information regarding the  $m^{\text{th}}$  vertex of the  $k^{\text{th}}$  sector.
  3. *hmin* This corresponds to the floor altitude in hundreds of feet of the sector module.
  4. *hmax* This corresponds to the ceiling altitude in hundreds of feet of the sector module.
  5. *hminnum*: This is a label corresponding to the floor altitude level.
  6. *hmaxnum*: This is a label corresponding to the ceiling altitude level.

#### 4.7.5 Structure with Adjacency Information of Sector Modules with Respect to Faces (*Adjsec*)

*Adjsec* is a data structure storing information about sectors that are adjacent with respect to a face. This has two records as described below.

1. *pos* This is an array that contains all the sector modules that lie on that side of the face which does not contain the origin. It has sub-fields under it, namely, *sect* and *loc*, corresponding respectively to the sector module number and the location of the vertical face in the sector module.
2. *neg* This is an array that contains all the sector modules that lie on that side of the face which contains the origin. Like *pos*, *neg* has two fields *sect* and *loc*.

Classifying the sector modules into those lying towards the origin and those lying away from the origin is helpful to identify the extreme faces of the defined airspace. A vertical face having sector modules lying toward only one side will be an extreme face.

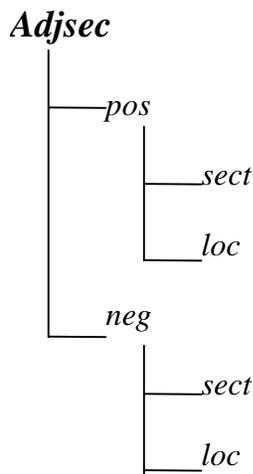


Figure 18 Data Structure of *Adjsec*.

#### 4.7.6 Structure with Adjacency Information of Sector Modules with Respect to Nodes (*Adjsecnode*)

*Adjsecnode* is the data structure that stores adjacency information with respect to the nodes. It has two records as explained below.

1. *sect* This is a row array of all the sector modules containing the vertex.
2. *loc* This is a row array of the location of the vertex in the sector module corresponding to the record *sect*.

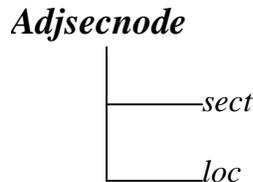


Figure 19 Data Structure of *Adjsecnode* .

$Adjsecnode(i).loc(1,j)$  gives the location of the  $i^{\text{th}}$  vertex on the sector  $Adjsecnode(i).sect(1,j)$ .

#### 4.7.7 Sector Module Adjacency Information (*Adj*)

*Adj* is a data structure storing the information regarding all the sector modules that are adjacent to a sector module in question. This has a record *sect* which is a row array storing the sector module numbers.

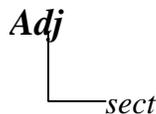


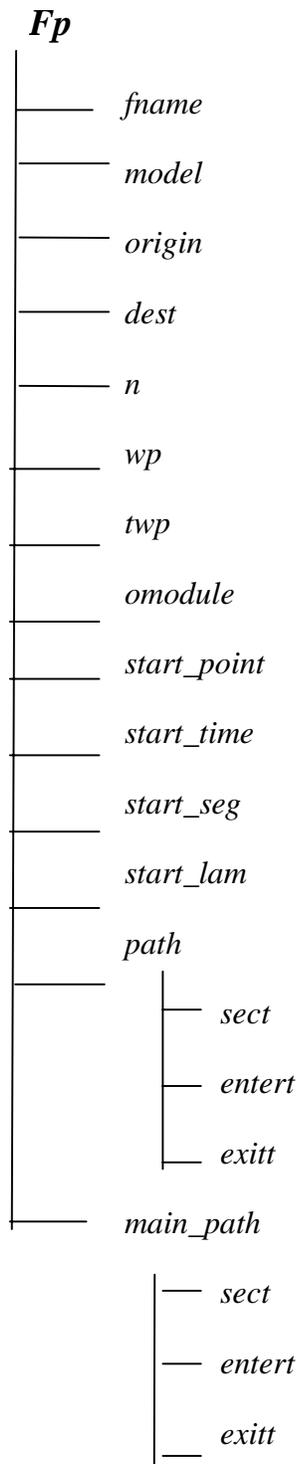
Figure 20 Data Structure of *Adj*.

#### 4.7.8 Flight Plan Structure (Fp)

*Fp* is the data structure which stores the information about the Flight Plans. *Fp* has the following fields.

1. *fname* Name designating the flight plan.
2. *model* Designator indicating the type of aircraft model.
3. *origin* Three letter designator of the origin airport.
4. *dest* Three letter designator of the destination airport.
5. *n* Number of way-points comprising the flight trajectory.
6. *wp* Array storing the latitude, longitude and altitude of each of the *n* way points
7. *twp* Array of size *n* by 1 storing the time corresponding to each way-point.
8. *omodule* The sector module that is first encountered by the flight.
9. *start\_point* The point where the flight first encounters a defined sector module.
10. *start\_time* The time when the flight first encounters a defined sector module.
11. *start\_seg* The flight segment which enters the defined airspace.
12. *start\_lam* The location of *start\_point* on the flight segment *start\_seg*.
13. *path* Structure storing all the sector modules encountered by the flight and the time of crossing. Path has the following three records under it.
  - a) *sect* Sector module encountered. This will be the number used by the program while storing the sector module information.
  - b) *entert* Time of entry into the sector module.
  - c) *exitt* Time of exit from the sector module.

14. *main\_path*      Structure storing all the sectors encountered by the flight and the time of crossing. Like *path*, *main\_path* has three records storing the sector number, entry time, and the exit time corresponding to the crossing.

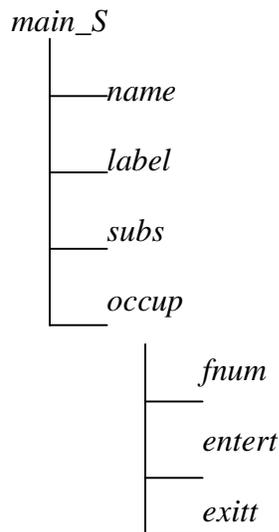


**Figure 21 Data Structure of Flight Plans (*Fp*).**

#### 4.7.9 Sector Information (*main\_S*)

This is a data structure corresponding to a sector having the following records.

1. *name*      Array of characters denoting the name of the sector. This is unique for a sector.
2. *label*      Row array of size 1 by 2 denoting the sector label.
3. *subs*      Row array which stores the sector module numbers that comprise the sector. The size of this array depends on the number of sector modules that make up the sector.
4. *occup*      Structure that stores the information about the flights that are crossed and the time of crossing. *Occup* has the following three records under it.
  1. *fnum*      :      Flight number that is crossed.
  2. *entert*    :      Time during which the *fnum* enters the sector.
  3. *exitt*     :      Time during which the *fnum* exits the sector.



**Figure 22 Data Structure of *main\_S*.**

#### 4.7.10 Sector Adjacency Information ( *main\_Adj* )

*main\_Adj* is a data structure storing the information regarding all the sectors that adjacent to each sector. This has a record *sect* which is a row array storing the sector numbers.



Figure 23 Data Structure of *main\_Adj*.

#### 4.7.11 Adjacency Information of Sectors with Respect to Nodes (*main\_Adjsecnode*)

*main\_Adjsecnode* is the data structure that stores adjacency information with respect to the nodes. It has one record as explained below.

*sect* : This is a row array of all the sectors containing the vertex.

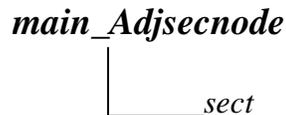
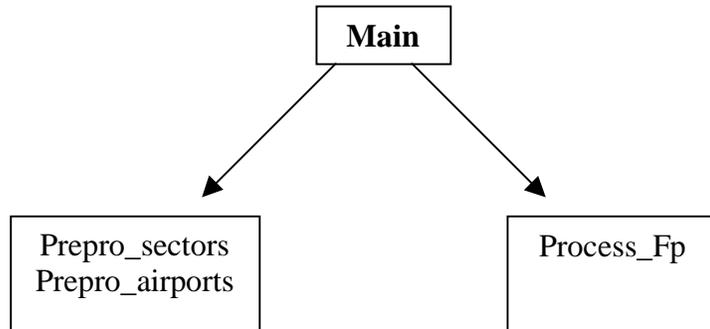


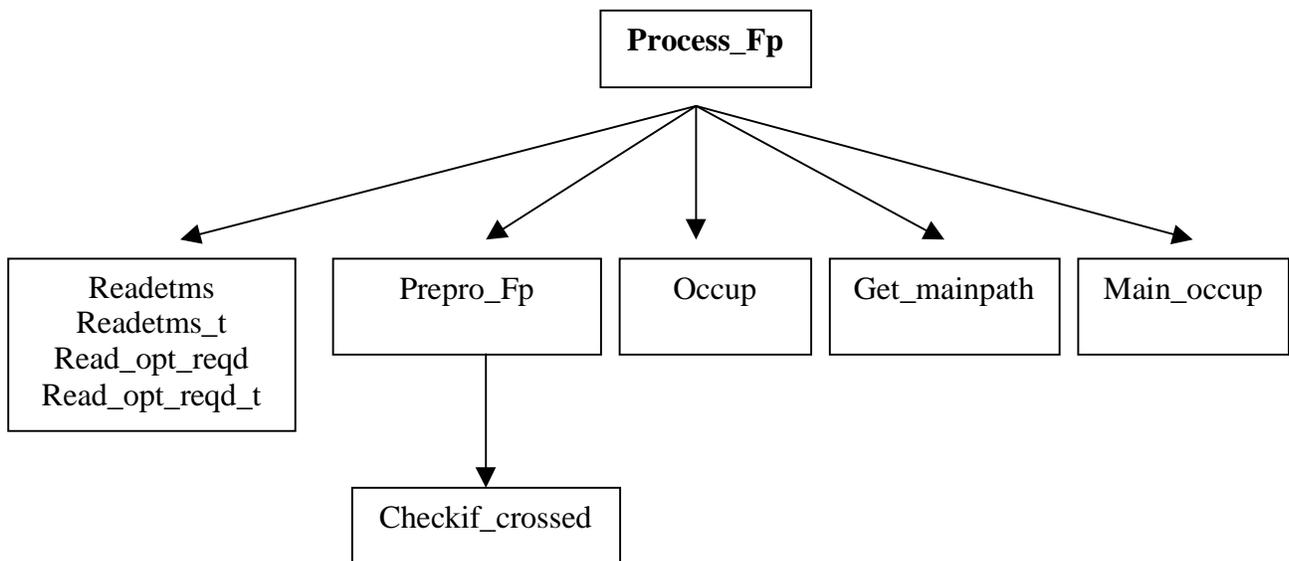
Figure 24 Data Structure of *main\_Adjsecnode*.

## 4.8 M-Files

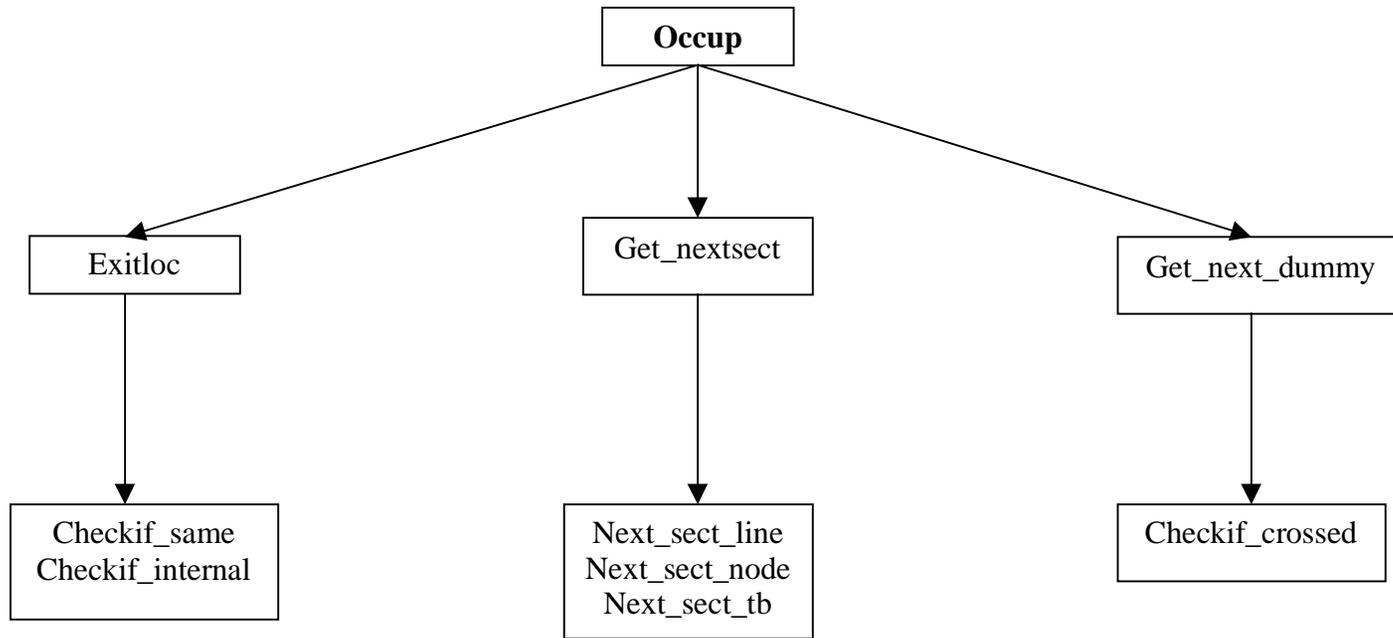
The m-files developed for the Airspace Sector Occupancy Model are described briefly in this section. The arrangement of these m-files and their hierarchy is depicted in Figures 25-28. Important m-files are discussed in Section 4.8.1.



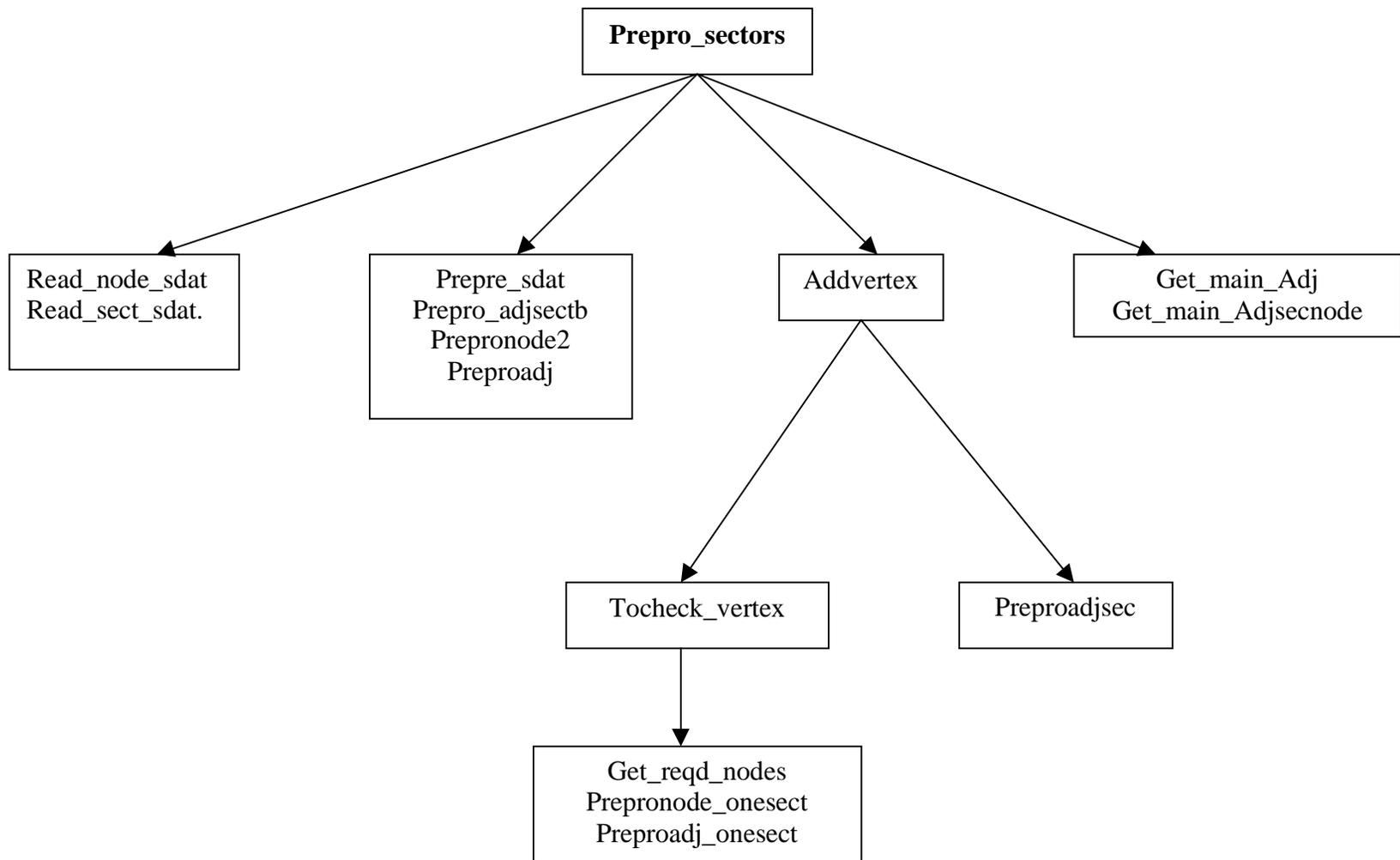
**Figure 25 M-files Called by the Main Program.**



**Figure 26 Organization of M-Files to Process Aircraft Flight Plans (Process\_Fp).**



**Figure 27 Organization of M-Files to Determine the Sector Modules Crossed (Occup).**



**Figure 28 Organization of M-Files to Pre-process the Sector Information (Prepro\_sectors).**

## 4.8.1 M-file Description

Important m-files used in the Airspace Sector Occupancy Model are described briefly below in alphabetical order. The Matlab programs are listed in the Appendix C.

### 1. **Addvertex**

**Purpose** This function determines all the nodes that are present on the faces of sector modules but are not originally defined for it. This m-file also updates all the adjacency information.

**Input** This contains information about the sector modules, the nodes, and the adjacency information with respect to nodes and sector modules.

**Output** This contains revised information about the sector modules and the adjacency relationships.

### 2. **Checkif\_crossed**

**Purpose** This determines if a flight segment crosses a particular face of a sector module.

**Input** This contains information about the flight segment the sector modules.

**Output** This contains a binary flag indicating if a crossing has taken place. If yes, the coordinate of the exit point and location of the exit point on the flight segment is determined.

### **3. Checkif\_internal**

**Purpose** This checks if a point lies on a line connecting two other given points.

**Input** This contains the coordinates of the three points.

**Output** This contains a binary flag with a value 1 if the given third point lies internally on the line connecting the other two points, and is 0 otherwise.

### **4. Checkif\_same**

**Purpose** This checks if two points are within an acceptable tolerance to be considered as the same point.

**Input** This contains the coordinates of the two points.

**Output** This contains a binary flag with a value 1 if the two points are close enough to be considered as the same, and 0 otherwise

### **5. Exitloc**

**Purpose** This determines the information about the point where the current sector module is exited by the flight segment in question.

**Input** This contains information about the flight segment, information about the current sector module, information regarding the point of entry and the previous sector module number.

**Output** This contains information about the point where the current sector module is exited by the flight segment under consideration.

## 6. Find\_ext\_sect

**Purpose** This identifies the vertical faces that are open to an undefined airspace on one of the sides.

**Input** This contains information about the sector modules and the adjacency information of the sector modules with respect to vertical faces.

**Output** This contains extreme faces identified based on their locations in the sector modules.

## 7. Get\_mainpath

**Purpose** This function identifies the sectors a flight will pass through knowing the sector modules it passes through.

**Input** This contains information about the flight trajectory and the sector modules.

**Output** This contains updated information about the flight trajectory with the information about the sectors that it passes through.

## 8. Getnextsect

**Purpose** This identifies the sector module the flight enters after exiting another module.

**Input** This contains information about the sectors, the information about the exit pattern from the previous sector module and the adjacency information.

**Output** This contains the sector module number that is entered. An indicator 0 is returned if the flight does not enter any of the sector modules.

## 9. **Getnext\_afterdummy**

**Purpose** This determines the sector module entered by the flight after passing through a vacuum.

**Input** This contains information about the flight trajectory under consideration, the extreme faces, and the information about the sector modules.

**Output** This contains the sector module number that is entered by the flight and the information about the point of entry. It includes the coordinates of the entry point and the flight segment number that enters the sector module.

## 10. **Get\_dummy**

**Purpose** This function extends the defined airspace by defining the dummy sectors surrounding the defined airspace.

**Input** This contains information about the sector modules, the nodes and the order in which the nodes are used to define a sector module (clockwise or anti-clockwise).

**Output** This contains modified information about the sector modules after the inclusion of the dummy sectors.

### **11. Get\_main\_Adj**

**Purpose** This determines the adjacency information of the sectors with respect to each other.

**Input** This contains adjacency information about the sector modules with respect to each other and the information about the sectors and sector modules.

**Output** This contains adjacency information of the sectors with respect to each other.

### **12. Get\_main\_Adjsecnode**

**Purpose** This determines the adjacency information of the sectors with respect to the nodes.

**Input** This contains adjacency information about the sector modules with respect to nodes and the information about the sectors and sector modules.

**Output** This contains adjacency information of the sector with respect to nodes.

### **13. Main**

**Purpose** This is the main function that calls all other functions and determines the occupancy of the sectors.

**Input** This contains the input file for the sector geometry and the flight plans.

**Output** This contains complete information about the sectors and the flight plans including the occupancy information.

#### **14. Main\_occup**

**Purpose** This function identifies the flights that pass through a sector, knowing the sector modules it encounters.

**Input** This contains information about the flight trajectories and the sectors.

**Output** This contains updated information about the sectors along with the flights passing through each sector.

#### **15. Next\_sect\_line**

**Purpose** This determines the sector module a flight enters after crossing one sector module across a vertical face.

**Input** This contains information about the sector modules, previous exit point and the adjacency information of the sector modules with respect to vertical faces.

**Output** This contains the sector module number that the flight enters.

#### **16. Next\_sect\_node**

**Purpose** This determines the sector module a flight enters after crossing one sector module across a vertical edge.

**Input** This contains information about the sector modules, the previous exit point, and the adjacency information of the sector modules with respect to nodes.

**Output** This contains the sector module number that the flight enters.

#### 17. **Next\_sect\_tb**

**Purpose** This determines the sector module a flight enters after crossing one sector module across its ceiling or floor.

**Input** This contains information about the sector modules, the previous exit point and the adjacency information of the sector modules with respect to floors and ceilings.

**Output** This contains the sector module number that the flight enters.

#### 18. **Occup**

**Purpose** This determines the sector modules that a flight passes through.

**Input** This contains information about the flight trajectory, sector modules, extreme faces, and the adjacency relationships.

**Output** This contains updated information about the flight trajectory, identifying all the sector modules that it passes through.

## **19. Plot\_hist\_view**

**Purpose** This plots the histogram corresponding to the occupancies of the sector and depicts its location on the US map.

**Input** This contains information about the occupancy of the sectors, the time interval of the histogram, and the sector number for which the plot is needed.

**Output** This contains the histogram plot showing the occupancies of the sector and the plot showing the location of the sector on the US map.

## **20. Preproadj**

**Purpose** This function identifies the sector modules adjacent to other sector modules

**Input** This contains information about the sector modules and the adjacency with respect to nodes.

**Output** This contains the information about the adjacency of sector modules with respect to one another.

## **21. Preproadjsec**

**Purpose** This function identifies the sector modules that are adjacent to one another with respect to vertical faces.

**Input** This contains information about the sector modules and the adjacency information with respect to nodes.

**Output** This contains information about the adjacency of sector modules with respect to vertical faces.

## **22. Preproadjsectb**

**Purpose** This function identifies the sector modules that are adjacent to one another with respect to horizontal faces.

**Input** This contains information about the sector modules.

**Output** This contains information about the adjacency of sector modules with respect to vertical faces.

## **23. Prepronode**

**Purpose** This function identifies the sector modules that are adjacent to one another with respect to nodes.

**Input** This contains information about the sector modules.

**Output** This contains information about the adjacency of sector modules with respect to nodes.

## **24. Prepro\_airports**

**Purpose** This function scans the input file regarding the airports and identifies the sector modules the airports lie in.

**Input** This contains information about the sector modules.

**Output** This contains information about the airports.

## 25. **Prepro**

**Purpose** This function obtains the mathematical representation of the sector modules when the vertices are defined in an anti-clockwise fashion.

**Input** This contains information about the sector modules.

**Output** This contains the mathematical representation of the sector modules.

## 26. **Prepro\_sdat**

**Purpose** This function obtains the mathematical representation of the sector modules when the vertices are defined in a clockwise fashion.

**Input** This contains information about the sector modules.

**Output** This contains the mathematical representation of the sector modules.

## 27. **Prepro\_sectors**

**Purpose** This function does the preprocessing of the sector information. It determines the mathematical representation of the sector modules, determines the adjacency information and identifies the extreme faces of the defined airspace.

**Input** This contains information about the sector modules.

**Output** This contains the mathematical representation of the sector modules, the adjacency information and the information about the extreme faces.

## **28. Process\_Fp**

**Purpose** This function scans the flight plan input file, does the pre-processing of the flight plan data and determines the occupancy information.

**Input** This contains information about the sector modules.

**Output** This contains the occupancy information.

## **29. Prepro\_Fp**

**Purpose** This function does the pre-processing of the flight plan information.

**Input** This contains information about the flight plans, airports and the sector modules.

**Output** This contains pre-processed flight plan information.

## **30. Readetms**

**Purpose** This function scans the input file for flight plans. The input file should be in FAA ETMS Format(Appendix B).

**Input** This contains the name of the input file.

**Output** This contains the flight plan information.

## **31. Read\_opt\_reqd**

**Purpose** This function scans the input file for flight plans. The input file should be in FAA ETMS Optimized Trajectory Format (Appendix B).

**Input** This contains the name of the input file.

**Output** This contains information about the flight plans.

### **32. Read\_opt\_reqd\_t**

**Purpose** This function scans the input file for flight plans corresponding to flights which are in the airspace during the time of interest. The input file should be in FAA ETMS Optimized Trajectory Format (Appendix B).

**Input** This contains the name of the input file and time of interest.

**Output** This contains the information about the flight plans corresponding to flights which are in the airspace during the time of interest.

### **33. Read\_sdat\_node**

**Purpose** This function scans the input file which contains information about the nodes that define the sector modules. The input file should be in the FAA SDAT Generic Format (Appendix B).

**Input** This contains the name of the input file.

**Output** This contains information about the nodes.

### **34. Read\_sdat\_sect**

**Purpose** This function scans the input file which contains information about the sector modules. The input file should be in the FAA SDAT Generic Format (Appendix B).

**Input** This contains the name of the input file and the information about the nodes.

**Output** This contains information about the sector modules.

### **35. Tocheck\_vertex**

**Purpose** This function determines all the nodes that are present on the faces of sector modules but are not originally defined for it. This m-file also updates the adjacency information on the sector modules with respect to nodes and with respect to each other.

**Input** This contains information about the sector modules, the nodes and the adjacency information with respect to nodes and sector modules.

**Output** This contains revised information about the sector modules and the adjacency relationships of the sector modules with respect to nodes and each other.

### **36. View\_main\_S**

**Purpose** This function plots the sector of interest in three dimensions.

**Input** This contains information about the sector and sector modules, and the number of the sector of interest.

**Output** This contains the plot of the sector of interest in three dimensions.

### **37. View\_sect\_Fp\_h**

**Purpose** This function plots the flight trajectories and the sector modules present at a particular altitude of interest.

**Input** This contains information about the sector modules and the flight trajectories, and the altitude of interest (hundreds of feet).

**Output** This contains the plot of the flight trajectories and the sector modules present at a particular altitude of interest.

### **38. View\_sect\_ht**

**Purpose** This function plots the sector modules present at a particular altitude of interest.

**Input** This contains information about the sector modules and the altitude of interest (hundreds of feet).

**Output** This contains the plot of the sector modules present at a particular altitude of interest.

## 5. MODEL APPLICATION AND ANALYSIS

### 5.1 Introduction

This Chapter deals with the application of the developed Airspace Sector Occupancy Model (ASOM) to several data sets. The FAA ETMS data and other optimized trajectory data sets are analyzed for the Jacksonville (ZJX) and Miami (ZMA) centers. The sector information is obtained from the FAA Sector Design and Analysis Tool (SDAT) database. The occupancies of sectors under various NAS scenarios are studied.

### 5.2 Model Application

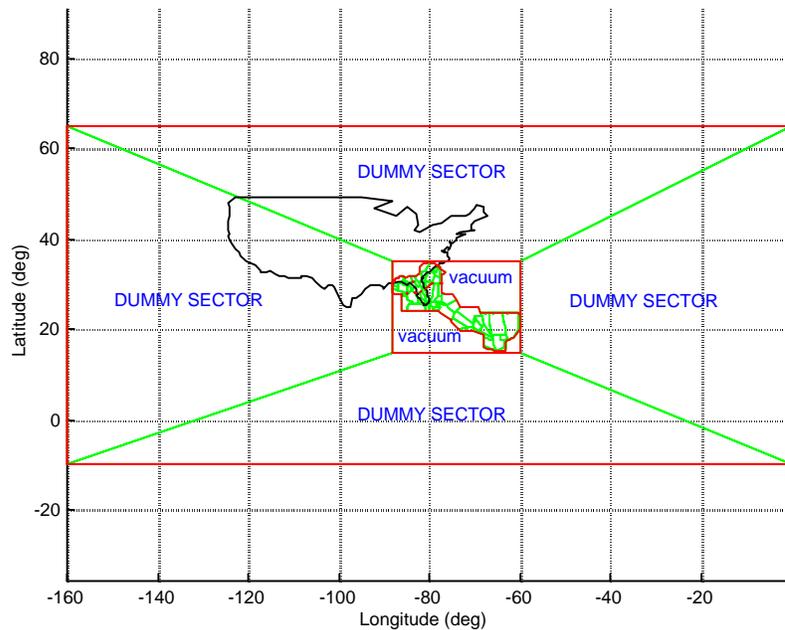
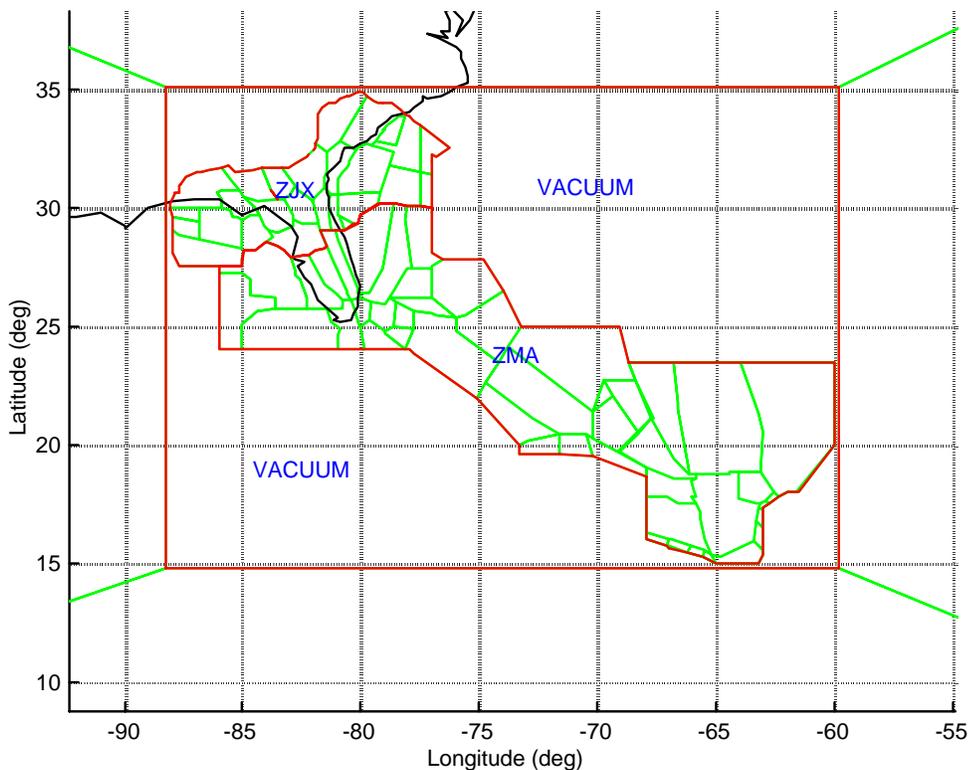


Figure 29 Sector Modules in ZJX and ZMA Centers along with Dummy Sectors.

The occupancy of sectors in the ZJX and ZMA Centers are studied using the Airspace Sector Occupancy Model (ASOM). The arrangement of the sector modules for these two centers at Flight Level 30,000 ft (FL300) is as shown in Figures 30 and 31.

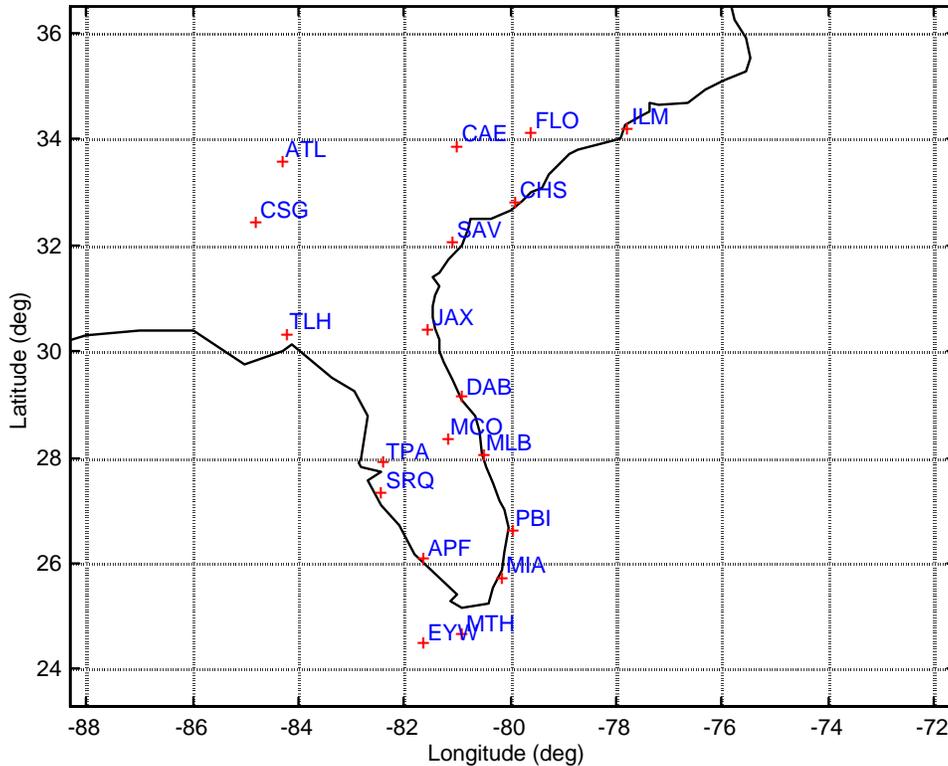
Overall, there are 88 sectors, comprised of 358 sector modules. Given a set of flight plans, the flights passing through any of the sectors can be obtained from the model.



**Figure 30 Sector Modules in the ZJX and the ZMA Center.**

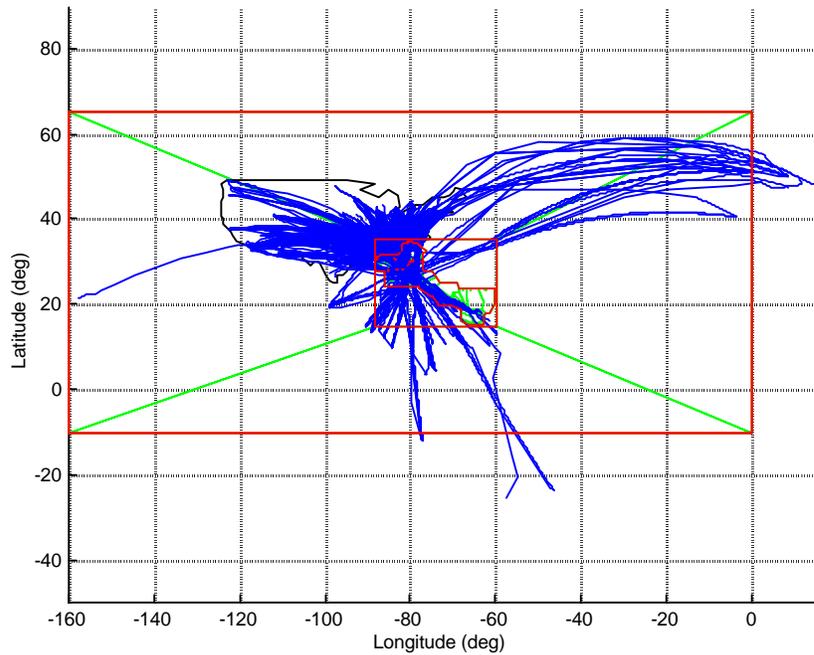
The flights that either originate or terminate in one of the 19 airports in the vicinity of these centers were considered. The airports are shown in Figure 31. Since this set encompasses airports in both the ZJX and ZMA centers and in the area to the eastern side of the airspace under consideration, it is assumed that all the flights passing over the

airspace under consideration will be captured. Close to 3000 flights crossed over the airspace under consideration in a single eighteen hour period according to ETMS data collected on Nov 12, 1996. The flight trajectories are shown in Figure 32.



**Figure 31 Airports Considered for the Analysis.**

The analysis is performed using six different route structures for the peak period of operations (1000-1350 minutes Reference Time). Reference Time in this case is the Greenwich Mean Time (GMT). These route structures consist of a baseline scenario, and five different variations representing the systematic reduction in airspace constraints as might occur in the future, as the NAS system evolves towards Free Flight.



**Figure 32 Flight Trajectories over the Airspace under Consideration.**

The flight plans for a particular day were used for the purpose of analyzing these scenarios. Flight plans obtained from the FAA ETMS database along with the corresponding air traffic situation on November 12, 1996, were used for this purpose. Each of the six scenarios is briefly described below. Whenever a flight is assumed not to rely on the ground-based NAVAIDs, a wind-optimized trajectory is adopted. Wind optimized routing is that trajectory which will have the least possible opposition from the prevailing winds.

### **5.2.1 Current National Airspace (NAS) Operations (Rt)**

This scenario represents the baseline case. The trajectories are based on the flight plans filed by the airlines. This scenario includes mostly fixed route flight plans using the high altitude airway system in the US that relies on ground based Navigational Aids

(NAVAIDS) such as Very High Frequency, Omni-Directional Range instruments (VORs).

**5.2.2 Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes (Cardinal\_Asg)**

This scenario reflects the removal of reliance on the ground-based NAVAIDs but retains the current directional flight levels. The altitudes for these routes are filed flight altitudes, and thus satisfy the following criteria required under the current concept of operations.

<b>Westbound Flights Levels</b>	<b>Eastbound Flights Levels</b>
FL180 to FL290 at intervals of 2000’ beginning at FL180.	FL180 to FL290 at intervals of 2000’ beginning at FL190.
Above FL290 at intervals of 4000’ beginning at FL310.	Above FL290 at intervals of 4000’ beginning at 290.

**5.2.3 Wind-Optimized Routing with a Reduced Vertical Separation and Assigned Altitudes (RVSM\_Asg)**

This is an extension of the previous case that considers reduction in the restriction on vertical flight separation levels. The minimum vertical separation between the flight plans is reduced to 1000 feet across the complete Class A Airspace. The altitudes are assigned based upon the following requirements and are closest to the filed levels that satisfy them.

<b>Westbound Flights Levels</b>	<b>Eastbound Flights Levels</b>
At intervals of 2000’ beginning at FL180.	At intervals of 2000’ beginning at FL190.

#### **5.2.4 Wind-Optimized Routing with Hemispherical Rules (Cardinal)**

This is similar to the scenario 5.2.3 except that the flight levels are based on the aircraft performance. Cardinal flight altitudes apply and the altitudes for this route structure satisfy the following criteria.

<b>Westbound Flights Levels</b>	<b>Eastbound Flights Levels</b>
FL180 to FL290 at intervals of 2000' beginning at FL180.	FL180 to FL290 at intervals of 2000' beginning at FL190.
Above FL290 at intervals of 4000' beginning at FL310.	Above FL290 at intervals of 4000' beginning at 290.

#### **5.2.5 Wind-Optimized Profiles with a Reduced Vertical Separation Method (RVSM)**

This is similar to the previous scenario except that the flight levels in this scenario adopt the reduced vertical separation rules that satisfy the following criteria

<b>Westbound Flights Levels</b>	<b>Eastbound Flights Levels</b>
At intervals of 2000' beginning at FL180.	At intervals of 2000' beginning at FL190.

### ***5.2.6 Wind-Optimized Profiles without Hemispherical Rules (Climb\_Cruise)***

This scenario reflects a complete relaxation of the stated restrictions. The trajectories are not constrained by the ground-based NAVAIDs or the flight levels, or the current cardinal altitude rules, or the vertical separation standards. The profiles represent complete cruise climb both in the terminal environment and en route.

## **5.3 Model Assumptions**

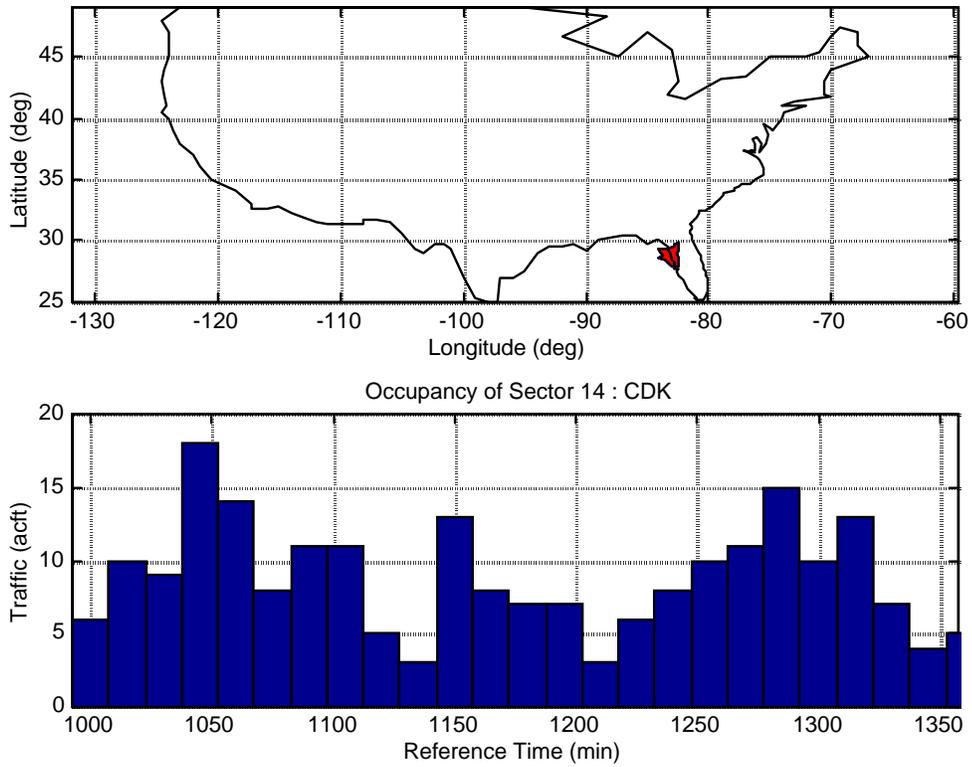
The assumptions made in the development of ASOM are as follows.

1. The flights are assumed to fly along straight lines between way-points. (dummy way-points could be specified to further discretize curvilinear flight trajectories)
2. Two nodes which are less than 0.35 nautical miles apart are assumed to define the same point in the airspace. (This assumption is made to correct for inaccuracies in data that sometimes assign different slightly perturbed locations to the same node, and hence create vacuums within the airspace.)
3. If a flight moves along a common boundary of some sector modules, it is assumed to pass through only one of them. The choice is made based on selecting the currently occupied sector, if applicable, or arbitrarily otherwise.

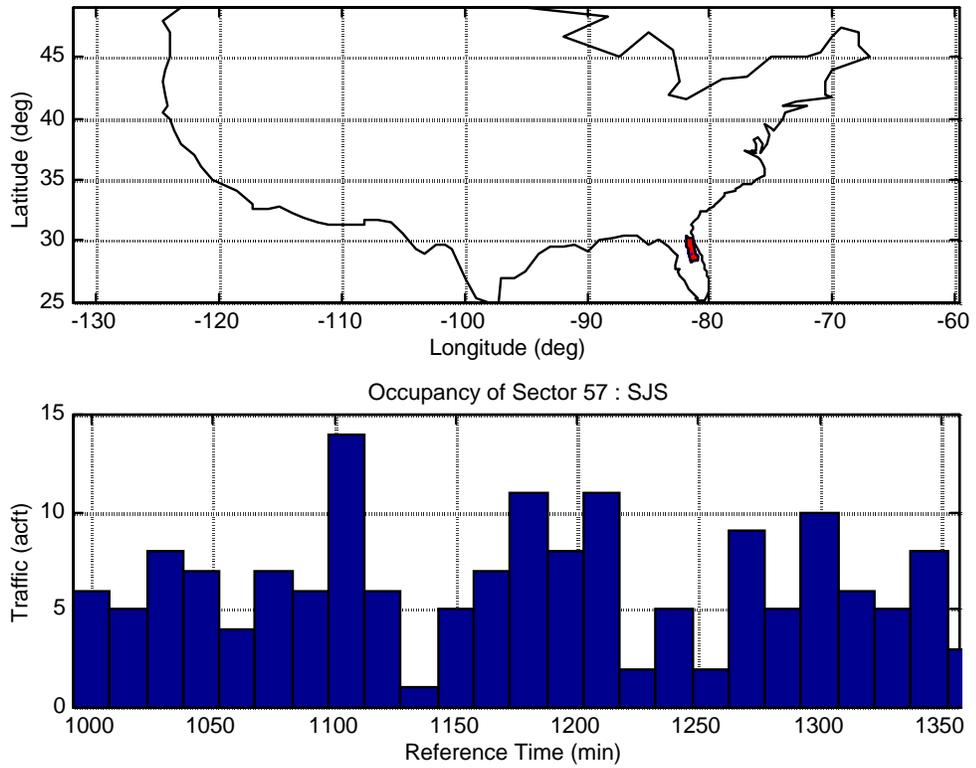
## **5.4 Model Output**

The occupancy of the sectors under the six different route structures as described in Section 5.2 were determined. The occupancies of ten sectors namely 14 (CDK), 57 (SJS), 67(HUN), 17 (FPY), 15 (OCF), 58 (SGJ), 84 (SEM), 16 (MAY), 67 (D67), and 72

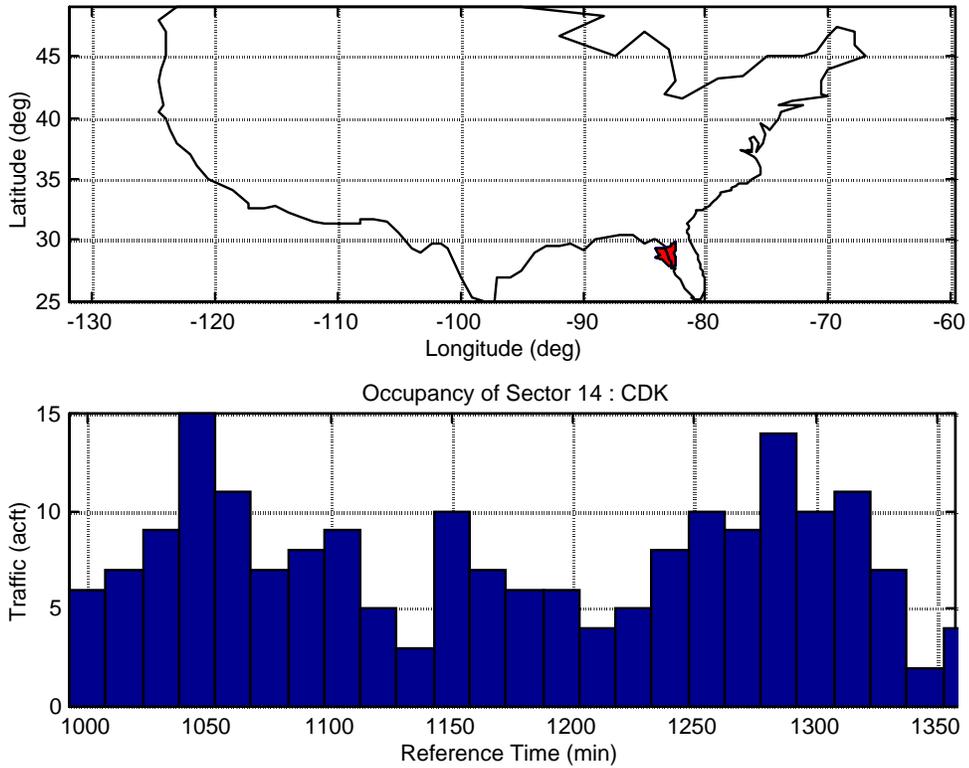
(CAE) are tabulated in the Tables 2-6. During the reference time period of 1000-1350 min, Sectors 57 (SJS), 67 (HUN), 15 (OCF), 58 (SGJ), and 72 (CAE) experienced more traffic whereas Sectors 14 (CDK), 17 (FPY), 84 (SEM), and 16 (MAY) experienced less traffic under the wind-optimized route structures. Traffic on Sector 67 (D67) remains more or less the same under all the six route structures. However, there was an increase in the maximum frequency (for a 15 minute time interval) in only three of the ten sectors studies, namely, Sectors 67(HUN), 17 (FPY) and 58 (SGJ). The location of the sectors 14 and 57, and the histogram showing the occupancies of the corresponding sectors over intervals of 15 minutes covering the duration 1000-1350 min Reference Time is shown in Figures 33 through 44.



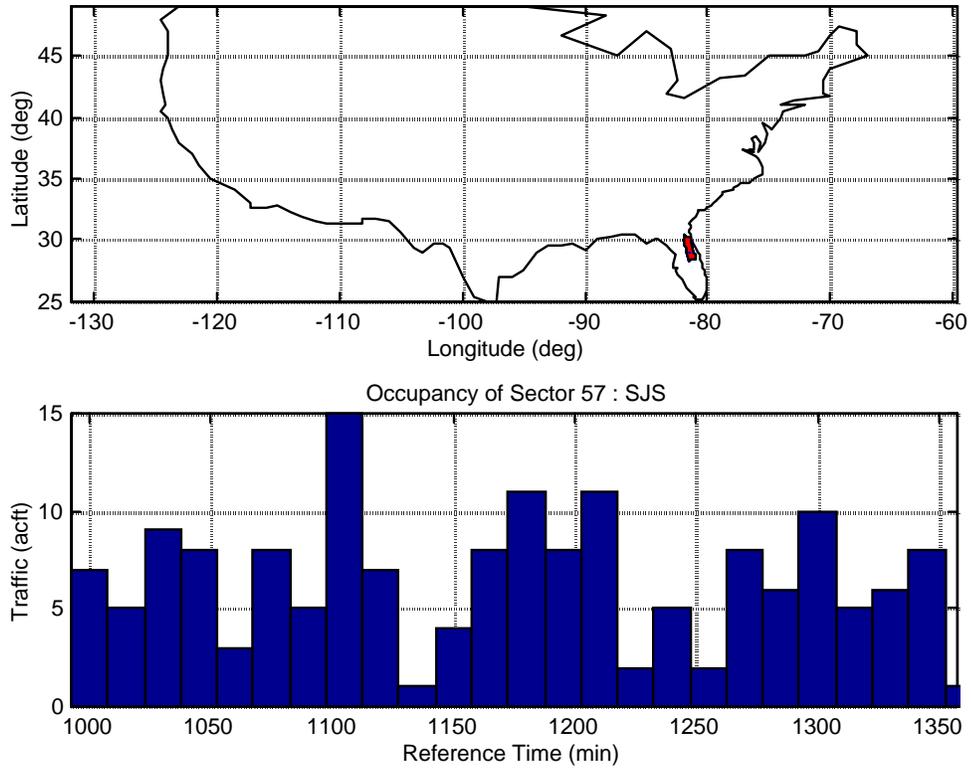
**Figure 33 Occupancy of Sector 14 (CDK) under Current NAS Operations.**



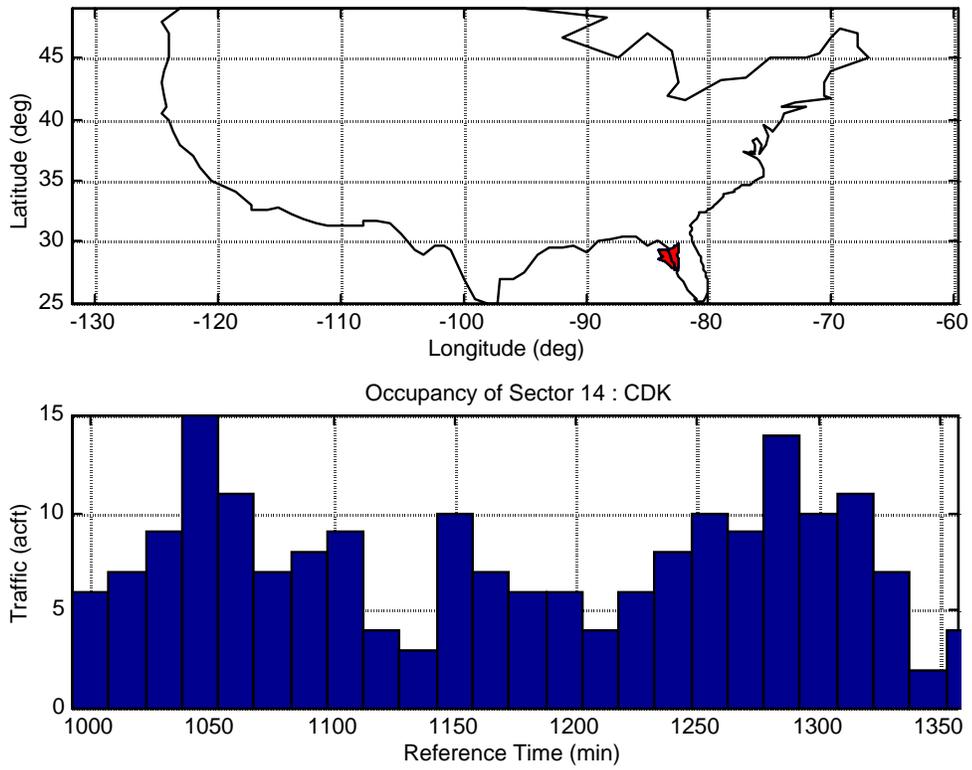
**Figure 34 Occupancy of Sector 57 (SJS) under Current NAS Operations.**



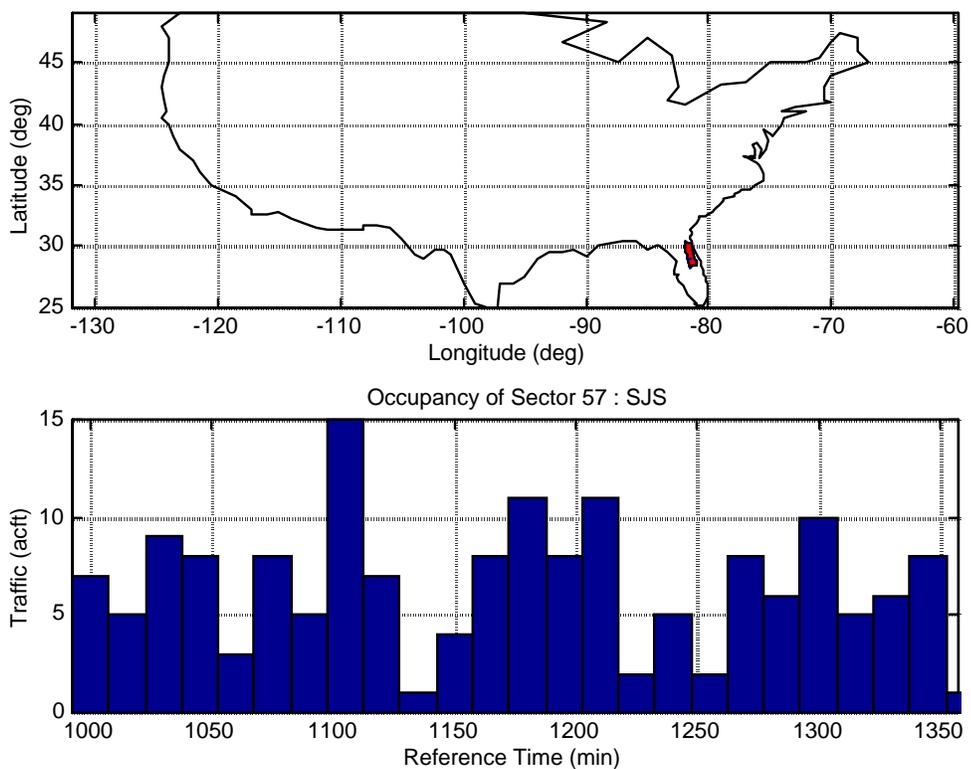
**Figure 35 Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes.**



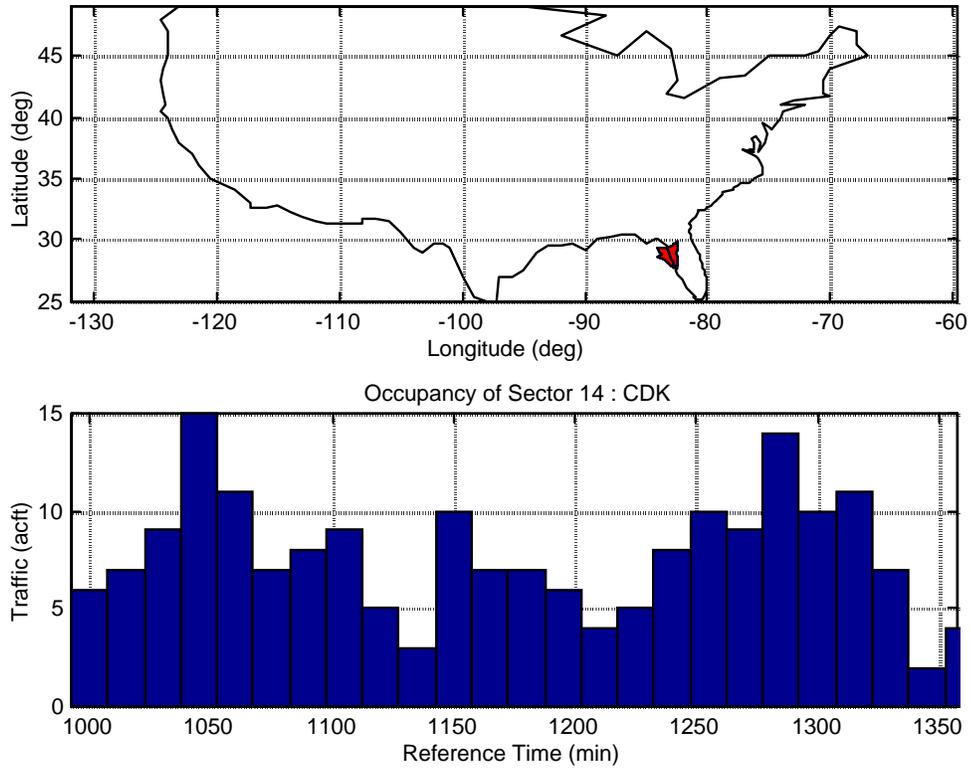
**Figure 36 Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with Hemispherical Rules and Assigned Altitudes.**



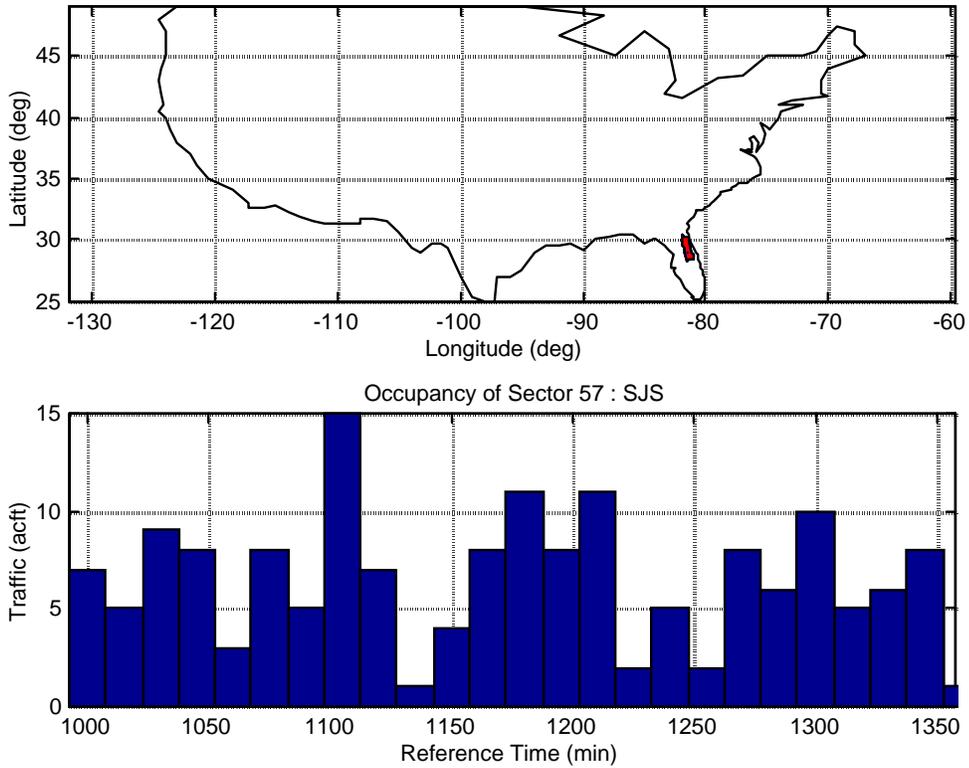
**Figure 37 Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with RVSM and Assigned Altitudes.**



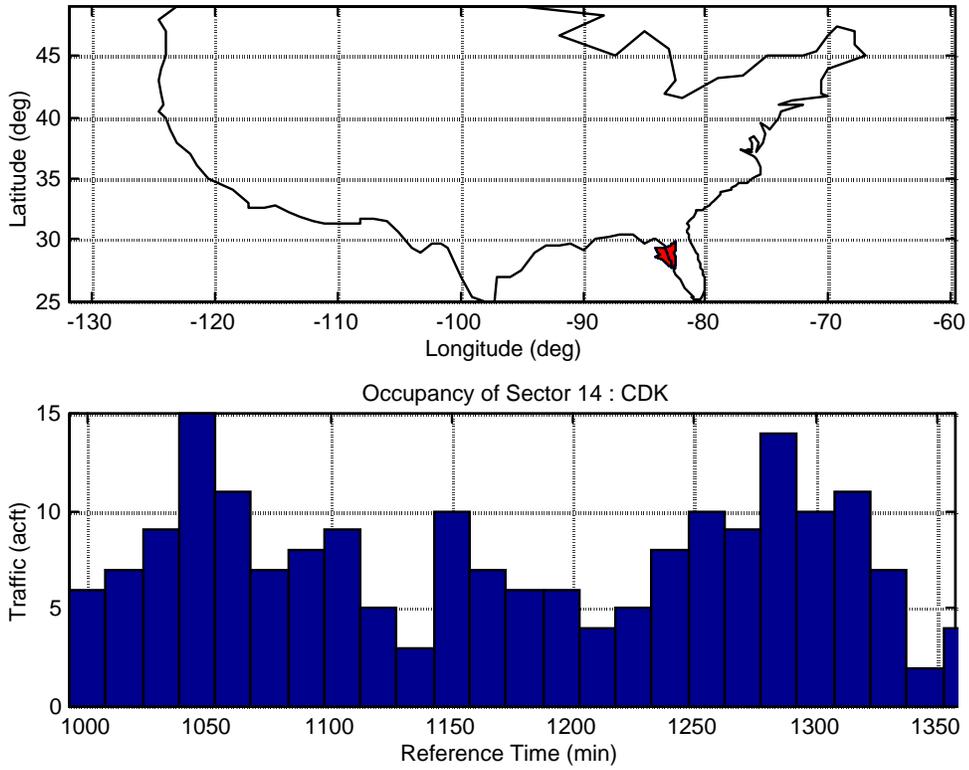
**Figure 38 Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with RVSM and Assigned Altitudes.**



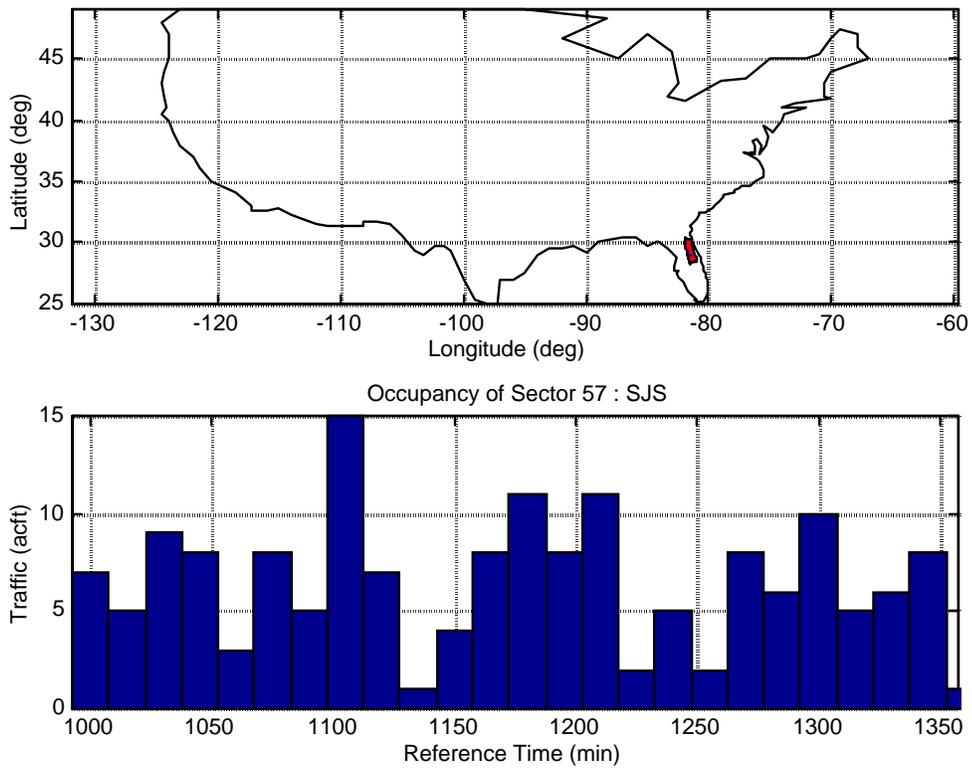
**Figure 39 Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with Hemispherical Rules.**



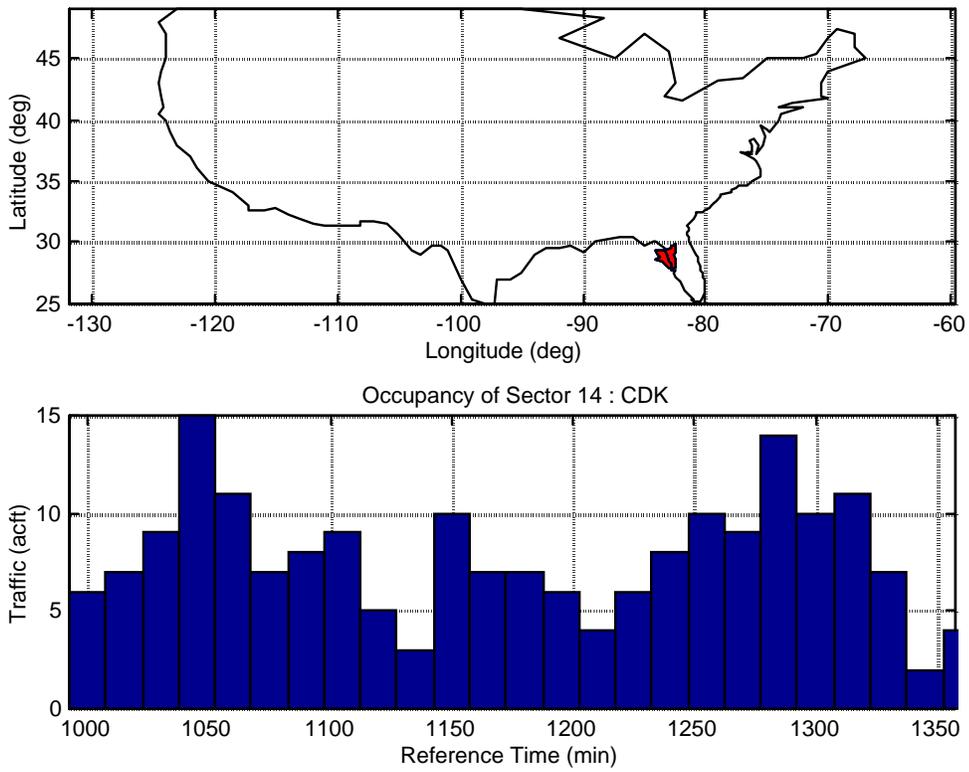
**Figure 40 Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with Hemispherical Rules.**



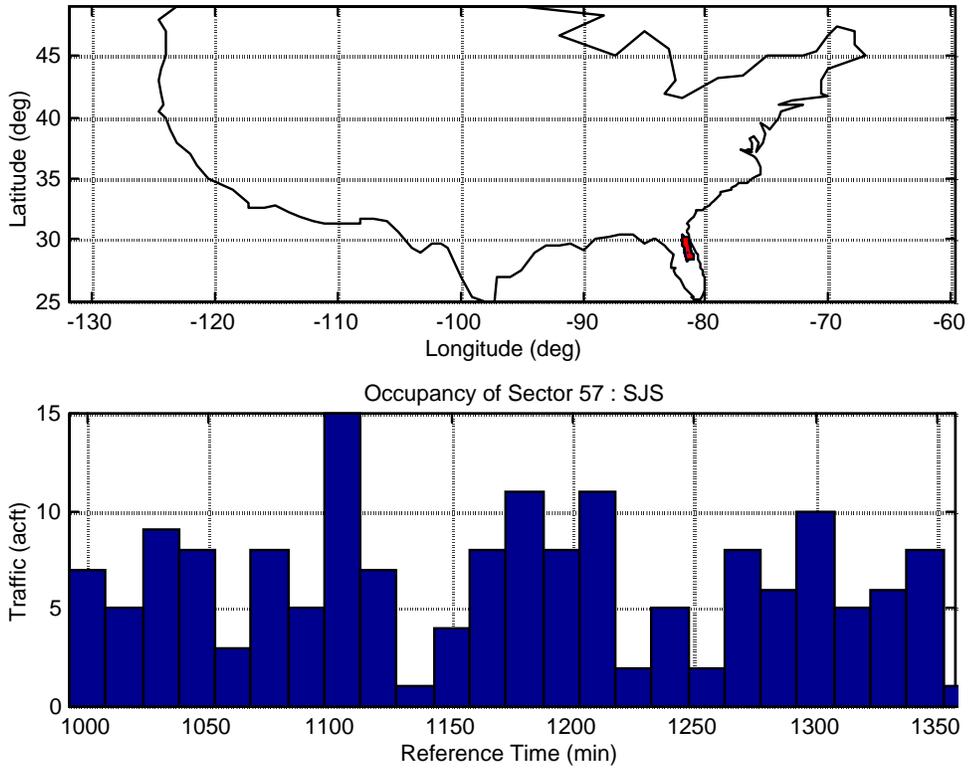
**Figure 41 Occupancy of Sector 14 (CDK) under Wind-Optimized Routing with RVSM.**



**Figure 42 Occupancy of Sector 57 (SJS) under Wind-Optimized Routing with RVSM.**



**Figure 43 Occupancy of Sector 14 (CDK) under Wind-Optimized Routing without Hemispherical Rules.**



**Figure 44 Occupancy of Sector 57 (SJS) under Wind-Optimized Routing without Hemispherical Rules.**

The results are summarized in Table 5.1. The total numbers of flights passing through Sector 14 (CDK) and Sector 57 (SJS) during the time duration 1000 to 1350 min GMT are tabulated. The maximum number of flights present in any 15 minute time duration is summarized (Maximum Frequency) in Table 2.

**Table 2 Occupancy of Sector 14 (CDK) and Sector 57 (SJS) under Different Route Structures.**

Case	Sector 14 (CDK)		Sector 57 (SJS)	
	Number of flights 1000-1350 min	Max. Frequency (15 min interval)	Number of flights 1000-1350 min	Max. Frequency (15 min interval)
Rt	209	18	154	19
Cardinal_Asg	178	15	157	15
RVSM_Asg	179	15	157	15
Cardinal	179	15	157	15
RVSM	178	15	157	15
Climb_Cruise	180	15	157	15

It may be noted that the maximum frequency in a 15 minute time interval for the optimized route structures is reduced from that for the base scenario that depicts current operations (Rt).

**Table 3 Occupancy of Sector 67 (HUN) and Sector 17 (FPY) under Different Route Structures.**

Case	Sector 67 (HUN)		Sector 17 (FPY)	
	Number of flights 1000-1350 min	Max. Frequency (15 min interval)	Number of flights 1000-1350 min	Max. Frequency (15 min interval)
Rt	101	13	148	12
Cardinal_Asg	174	19	127	13
RVSM_Asg	175	19	125	13
Cardinal	175	19	126	13
RVSM	176	19	127	13
Climb_Cruise	175	19	127	13

**Table 4 Occupancy of Sector 15 (OCF) and Sector 58 (SGJ) under Different Route Structures.**

Case	Sector 15 (OCF)		Sector 58 (SGJ)	
	Number of flights 1000-1350 min	Max. Frequency (15 min interval)	Number of flights 1000-1350 min	Max. Frequency (15 min interval)
Rt	138	10	143	11
Cardinal_Asg	147	11	159	14
RVSM_Asg	147	11	158	13
Cardinal	148	11	158	14
RVSM	147	11	156	13
Climb_Cruise	147	11	158	14

**Table 5 Occupancy of Sector 84 (SEM) and Sector 16 (MAY) under Different Route Structures.**

Case	Sector 84 (SEM)		Sector 16 (MAY)	
	Number of flights 1000-1350 min	Max. Frequency (15 min interval)	Number of flights 1000-1350 min	Max. Frequency (15 min interval)
Rt	122	11	116	9
Cardinal_Asg	81	8	107	9
RVSM_Asg	81	8	105	9
Cardinal	81	7	106	8
RVSM	82	7	106	9
Climb_Cruise	79	8	105	8

**Table 6 Occupancy of Sector 67 (D67) and Sector 72 (CAE) under Different Route Structures.**

Case	Sector 67 (D67)		Sector 72 (CAE)	
	Number of flights 1000-1350 min	Max. Frequency (15 min interval)	Number of flights 1000-1350 min	Max. Frequency (15 min interval)
Rt	46	5	42	5
Cardinal_Asg	46	5	46	8
RVSM_Asg	48	5	46	8
Cardinal	46	5	46	8
RVSM	47	5	46	8
Climb_Cruise	47	5	46	8

## **6. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH**

### **6.1 Introduction**

This chapter deals with the development and use of the Airspace Sector Occupancy Model (ASOM) to analyze the current Aviation System. Recommendations are made to enhance the user interface and to couple with other complimentary airspace models to be used for airspace planning.

### **6.2 Conclusions**

The Airspace Sector Occupancy Model (ASOM) can evaluate the occupancies of sectors for a given set of flight schedules and paths. It can therefore be used to analyze and compare the impacts on workload and collision risk of different sets of flight schedules traversing over the air traffic control sectors. In particular, the model can serve as a valuable tool to study the impact of Free Flight on the workload of air traffic controllers.

The processing time of the program depends on the input data, the model structure and the computer type. The size of the input data depends on the number of Air Route Traffic Control Centers (ARTCC) and the number of flight trajectories to be analyzed. To analyze all the flights traversing the sectors in the ZJX and ZMA Centers, it takes about

eight hours on an SGI-O2 machine. Altogether, there were close to 3,000 flights traversing these sectors on a typical day in 1996.

### **6.3 Recommendations for Future Research**

The Airspace Sector Occupancy Model (ASOM) may be integrated with other airspace analysis models that are used for airspace planning. The following extensions and upgrades to the model are suggested to move in this direction.

1. An independent flight trajectory generator can be coupled with ASOM, so that the flight trajectories for any hypothetical set of flight schedules can be generated and analyzed.
2. The model can be coupled with an independent conflict detection and analysis model to analyze the nature of conflicts between the flights. From the output of the ASOM (sector occupancy information), one can identify the flights that will be close to any particular flight at any instant of time and a conflict analysis may be conducted for these selected pairs of flights. This effort is being pursued as part of a collision risk analysis study being performed for the FAA at Virginia Tech.
3. A graphical user interface could be developed to make the model more user-friendly.

## BIBLIOGRAPHY

1. National Research Council, "Flight to the Future, Human Factors in Air Traffic Control", *National Academy Press*, 1997.
2. Federal Aviation Administration, "Air Traffic Control.", *FAA Order 7110.65J*, 1995.
3. Smith, K., Scallen, S. F., Knecht, W., Hancock, P., "An Index of Dynamic Density", *The Journal of the Human Factors Society*, 20:69-78, 1998.
4. Laudeman, I. V., and Palmer E. A., "Quantitative Measurement of Observed Workload in the Analysis of Aircrew Performance", *International Journal of Aviation Psychology*, 5:187-197, 1995.
5. Wiener E. L., and Nagel D. C., "Human Factors in Aviation.", *Academic Press Inc.*, 1988.
6. Suchkov, A., Embt, D., Colligan, W., "Dynamic Density Study", *CSSI, Inc.*, December 1997.
7. Wyndemere, Inc., "An Evaluation of Air Traffic Control Complexity", *Final Report for Contract Number NAS2-14284*, October, 1996.
8. Using Matlab®, Version 5, *The Mathworks, Inc.*, December, 1996.
9. Using Matlab® Graphics, Version 5, *The Mathworks, Inc.*, December, 1996.

## APPENDIX A

**Airports in the ZJX, the ZMA and the Nearby Centers.**

<b>Airport Designation</b>	<b>ARTCC</b>	<b>Longitude (deg)</b>	<b>Latitude (deg)</b>
TLH	ZJX	-84.3500	30.3830
JAX	ZJX	-81.6830	30.4830
SAV	ZJX	-81.2000	32.1330
MCO	ZMA	-81.3170	28.4170
DAB	ZMA	-81.0500	29.1830
TPA	ZJX	-82.5330	27.9670
SRQ	ZMA	-82.5500	27.3830
MLB	ZMA	-80.6330	28.1000
CSG	ZTL	-84.9330	32.5170
CHS	ZJX	-80.0330	32.8830
CAE	ZJX	-81.1170	33.9330
PBI	ZMA	-80.0830	26.6830
MIA	ZMA	-80.2830	25.7830
APF	ZMA	-81.7670	26.1500
MTH	ZMA	-81.0500	24.7170
EYW	ZMA	-81.7500	24.5500
FLO	ZJA	-79.7170	34.1830
ILM	ZDC	-77.9000	34.2670
ATL	ZTL	-84.4330	33.6500

## APPENDIX B

### Sector and Flight Plan Data Format

#### *FAA Sector Design and Analysis Tool (SDAT) , Sector Data Format*

The sector information is represented in two separate files. The first one contains information about the nodes and the other contains information about the sectors in terms of the nodes. The format of these two files are described below.

#### **Input File for the Nodes**

A sample of the input file for nodes is shown below.

```
7ABA 275602N 0805530W
7B81 274502N 0811200W
7BA9 274652N 0814820W
7CB1 275802N 0816001W
8B02 280002N 0811401W
8B13 280832N 0811750W
A640 293602N 0880101W
AC02 342803N 0810640W
```

Each line of the input file consists of the node-name followed by its latitude and longitude coordinates.

### **Input File for the Sectors**

A sample of the input file for sectors is shown below.

```
0001 A-SSS C-SAV M-000/020 SA08 SA09 NBC3 NBC2 H3A4 NBC1
J504 J501 J503 G44B SA26 QT02 SA25 QT01 SA15 G410 SA11 SA16
SA17 SA18 SA19 SA20 SA21 SA22 SA23 SA24 SA6A SA07 SA7A M-
021/030 SA08 SA09 NBC3 NBC4 H3A4 NBC1 J504 J501 J503 G44B
SA26 QT02 SA25 QT01 SA15 G410 SA11 SA16 SA17 SA18 SA19 SA20
SA21 SA22 SA23 SA24 SA6A SA07 SA7A M-031/100 SA08 SA09 NBC3
NBC4 H3A4 NBC1 J504 J501 J503 G44B SA26 QT02 SA25 QT01 SA15
G410 SA11 SA16 SA24 SA6A SA07 SA7A
```

```
9506 A-TTT C-PIE M-000/100 6CB7 7C28 7C33 7C43 7C10 7C01
9601 A-MMM C-MIA M-000/040 594A 5A31 5A15 5A48 5A9D 5A7A
5B90 5B92 6A1B 6A2A 6A47 6A56 6A52 6978 694A 6908 6909 6910
5964 5958 M-041/160 594A 5A31 5A15 5A48 5A9D 5A7A 5A9A 6A1B
6A2A 6A47 6A56 6A52 6978 694A 6908 6909 6910 5964 5958
```

The sectors are defined in terms of Fixed Point Approaches (FPAs). Each FPA record contains the FPA name, optional Automated Radar Terminal System (ARTS), Approach Control, sector name, and one or more module records. Each module record consists of the module floor and ceiling altitudes in hundreds of feet followed by the list of nodes which define its boundary. The ceiling of the module is given as 1 less (100 feet) than the floor of the module above it. Therefore the ceiling of a module is redefined to be 1 more than the value in the input file so that there is a continuity in the description of the airspace. The following representation is used to define the FPAs and the modules.

## **FPA Record**

FPA Label	This is a four digit integer. The first two correspond to the sector and the next two correspond to the FPA.
Optional ARTS	This consists of two alpha-numeral characters preceded by a label 'A-'.
Optional Approach	This consists of three alpha-numeral characters preceded by a label 'C-'.
Optional Sector Name	This consists of two to five alpha-numeral characters preceded by a label 'N-'.

## **Module Record**

Altitudes	This consists of three digit integers representing the floor and ceiling altitudes in hundreds of feet. This is preceded by a label 'M-'.
Node-names	This consists of the list of node-names that define the module boundary.

## **FAA En-route Traffic Management System (ETMS) , Flight Plan Data Format**

There are two main formats for trajectory description as used by FAA namely,

1. Current NAS Operations (rt) Format.
2. Free Flight Format.

### **Current NAS Operations (rt) Format**

A sample of current NAS operations (rt) format is shown below.

```

AAL1_____00_0  YYY B767  YYY  YYY      1
AAL1_____00_0   JFK    LAX      40.640    73.779    866
33.943  118.408 1152  14:26    286 0 C  50
IZYYYYYYY  40.633    73.783    0    866.000    123    123
YYYYYYYYY  40.417    74.143   112   871.885    330    304
YYYYYYYYY  40.200    74.500   208   875.749    386    345
YYYYYYYYY  40.285    74.988   282   879.526    441    386
YYYYYYYYY  40.369    75.477   330   882.941    477    413
YYYYYYYYY  40.450    75.967   347   886.272    477    411
YYYYYYYYY  40.500    76.283   348   888.425    477    412
YYYYYYYYY  40.550    76.600   349   890.573    477    412
YYYYYYYYY  40.592    77.016   350   893.360    477    413
YYYYYYYYY  40.633    77.433   350   896.142    477    414
YYYYYYYYY  40.650    77.800   350   898.568    477    414
YYYYYYYYY  40.687    78.282   350   901.762    477    415
YYYYYYYYY  40.723    78.765   350   904.951    477    416
YYYYYYYYY  40.756    79.249   350   908.132    477    417
YYYYYYYYY  40.787    79.732   350   911.305    477    418
YYYYYYYYY  40.817    80.217   350   914.472    477    419
YYYYYYYYY  40.900    81.683   350   923.999    477    424

```

YYYYYYYY	40.900	81.817	350	924.854	477	425
YYYYYYYY	40.967	84.200	350	940.081	477	430
YYYYYYYY	40.983	85.183	350	946.289	477	433
YYYYYYYY	41.167	86.583	350	955.179	477	437
YYYYYYYY	41.167	89.583	350	973.665	477	448
YYYYYYYY	40.550	91.400	350	985.723	477	455
YYYYYYYY	40.417	91.783	350	988.259	477	455
YYYYYYYY	40.283	92.183	350	990.891	477	457
YYYYYYYY	40.133	92.583	350	993.574	477	456
YYYYYYYY	39.783	94.183	350	1003.617	477	459
YYYYYYYY	39.633	94.800	350	1007.519	477	459
YYYYYYYY	38.933	97.617	350	1025.428	477	461
YYYYYYYY	37.917	100.733	350	1046.062	477	461
YYYYYYYY	36.500	104.867	350	1074.123	477	457
YYYYYYYY	35.483	108.867	350	1100.881	477	456
YYYYYYYY	34.933	111.383	350	1117.655	477	458
YYYYYYYY	34.700	112.483	350	1124.986	477	461
YYYYYYYY	34.567	113.633	350	1132.452	477	456
YYYYYYYY	34.550	113.683	350	1132.802	477	459
YYYYYYYY	34.465	114.101	350	1135.583	477	459
YYYYYYYY	34.378	114.518	350	1138.365	477	459
YYYYYYYY	34.290	114.935	350	1141.146	477	459
YYYYYYYY	34.200	115.350	350	1143.926	477	460
YYYYYYYY	34.117	115.767	350	1146.704	477	466
YYYYYYYY	34.109	116.250	348	1149.798	438	427
YYYYYYYY	34.100	116.733	275	1153.199	435	419
YYYYYYYY	34.075	116.983	234	1155.014	429	410
YYYYYYYY	34.050	117.233	191	1156.889	409	393
YYYYYYYY	34.033	117.400	162	1158.184	393	381
YYYYYYYY	34.017	117.633	129	1160.075	369	361
YYYYYYYY	34.000	117.767	113	1161.221	345	340
YYYYYYYY	33.975	118.083	66	1164.440	252	253
LZYYYYYY	33.950	118.400	0	1168.621	219	225

The above representation carries the following information in the order shown.

Aircraft Id    YYY   Aircraft Type   YYY   YYY   Number of Flight legs

Flight leg Id    Origin   Destination

Origin Latitude                      Origin Longitude                      Departure Time with Respect  
to a Reference Time in Minutes.

Departure Latitude                      Departure Longitude                      Arrival Time with Respect to  
a Reference Time in Minutes.

Departure Time in Hour and Minutes                      Time of Flight in Minutes                      Data  
Data    Number of points for this Flight Leg.

IZYYYYYY (initial point)    Latitude (floating point degrees), Longitude (floating point  
degrees), Flight Level (100's of feet), Time (minutes), Airspeed (knots), Ground Speed  
(knots).

YYYYYYYY (intermediate point)    Latitude    (floating    point    degrees), Longitude  
(floating point degrees), Flight Level (100's of feet), Time (minutes), Airspeed (knots),  
Ground speed (knots).

LZYYYYYY (final point)    Latitude (floating point degrees), Longitude (floating point  
degrees), Flight Level (100's of feet), Time (minutes), Airspeed (knots), Ground Speed  
(knots).

## Free Flight Format

A sample of free flight format is shown below.

```
AAL1_____00_0      1
AAL1_____00_0 25
IZYYYYYY 40.633000 73.783  0   866.013
YYYYYYYY 40.417000 74.143 112 871.896
YYYYYYYY 40.200000 74.5    208   875.763
YYYYYYYY 40.237008 74.7109 240 877.399
YYYYYYYY 40.333100 77.0462 360 894.086
YYYYYYYY 40.380737 79.3864 400 909.689
YYYYYYYY 40.300621 84.0661 400 940.669
YYYYYYYY 40.193320 86.3988 400 955.786
YYYYYYYY 39.837100 91.0339 400 985.195
YYYYYYYY 39.604959 93.3329 400 999.639
YYYYYYYY 39.296191 95.6086 400 1013.89
YYYYYYYY 38.958421 97.8664 400 1028.03
YYYYYYYY 38.152998 102.311 400 1055.94
YYYYYYYY 37.687488 104.494 400 1069.87
YYYYYYYY 37.181643 106.649 400 1083.87
YYYYYYYY 36.660788 108.782 400 1097.79
YYYYYYYY 35.388725 112.909 400 1125.89
YYYYYYYY 34.730278 114.934 360 1139.99
YYYYYYYY 34.078690 116.946 240 1154.75
YYYYYYYY 34.075000 116.983 234 1155.02
YYYYYYYY 34.050000 117.233 191 1156.88
YYYYYYYY 34.017000 117.633 129 1160.08
YYYYYYYY 34.000000 117.767 113 1161.22
YYYYYYYY 33.975000 118.083 66  1164.43
LZYYYYYY 33.950000   118.4   0  168.62
```

The above representation carries the following information in the order shown below.

Aircraft Id	Number of flight legs
Flight leg Id	Number of points for this flight leg

IZYYYYYY (initial point) Latitude (floating point degrees), Longitude (floating point degrees), Flight Level (100's of feet), Time (minutes).

YYYYYYYY (intermediate point) Latitude (floating point degrees), Longitude (floating point degrees), Flight Level (100's of feet), Time (minutes).

LZYYYYYY (final point) Latitude (floating point degrees), Longitude (floating point degrees), Flight Level (100's of feet), Time (minutes).

## APPENDIX C

Important Matlab Programs developed are listed in this Appendix.

### **Addvertex**

%THIS FUNCTION DETERMINES IF ANY VERTEX NOT DEFINED FOR A %SECTOR LIES ON ANY OF ITS FACES. IT ALSO UPDATES ALL THE %ADJACENCY MATRICES.

```
function out=addvertex(S,Se,N,Adj,Adjsecnode);
```

```
% modifying S,Se,Adjsec & Adjsecnode
```

```
ans1=tocheck_vertex(S,Se,N,Adj,Adjsecnode);  
S=ans1.S;  
Se=ans1.Se;  
Adjsecnode=ans1.Adjsecnode;  
clear ans1;
```

```
for(i=1:length(S))  
    len=length(S(i).ver(1,:));  
    for(j=1:len)  
        ver=S(i).ver(1,j);  
        S(i).long(1,j)=-N(ver,1);  
        S(i).lat(1,j)=N(ver,2);  
    end  
    S(i).long(1,len+1)=S(i).long(1,1);  
    S(i).lat(1,len+1)=S(i).lat(1,1);  
end
```

```
for(i=1:length(Se))  
    for( j=1:Se(i).n)  
        Se(i).line(1,j).flag=1;  
    end  
end
```

```
Adj=preproadj(S,Se,Adjsecnode);
```

```
ans1=preproadjsec_mod_node(Se,S,Adjsecnode);  
Adjsec=ans1.Adjsec;  
Se=ans1.Se;  
clear ans1;
```

```
out.S=S;  
out.Se=Se;  
out.Adj=Adj;  
out.Adjsecnode=Adjsecnode;  
out.Adjsec=Adjsec;
```

### ***Checkif\_crossed***

```
function out=checkif_crossed(P,d,sect_num,loc,Se,S,N,lam)

% DETERMINES IF THE FLIGHT SEGMENT FROM P IN THE DIRECTION d % WILL CROSS THE MODULE
SECT_NUM AT THE FACE WHICH IS DEFINED %AT LOC TIMES.

% Assumption that a flight will always enter the defined airspace through a vertical face of a module.

% returns lambda which is the ratio of distance of P - enterp to det_d

% Pis the cordinales row of size 3.
% d is the direction. row of size 3.
out.check=0;
out.enterp=[0 0 0];
out.lambda=0;

check1=0;

dis_tolerance=0.004;
bignum=1000000;
smallnum=0.001;

det_d=sqrt(d(1,1).^2+d(1,2).^2);

if(det_d>dis_tolerance)

norm_d=d(1,1:2)/sqrt(d(1,1).^2+d(1,2).^2);

i=loc;

line=Se(sect_num).line(1,i);
    if( (line.alpha*d(1,1:2)) > smallnum)                % if along the inward gradient

        leng=( P(1,1:2)*line.alpha'- line.c);
        leng=abs(leng/(line.alpha*norm_d(1,1:2)));
        len=leng;
        clear leng;

        lambda=len/det_d;
        Ex_P=P(1,1:2)+lambda*d(1,1:2);

        First_P=N( S(sect_num).ver(1,i,:));
        First_P(1,1)=-First_P(1,1);

        if(i==Se(sect_num).n)

            Sec_P=N( S(sect_num).ver(1,1,:));
        else

            Sec_P=N( S(sect_num).ver(1,i+1,:));
        end
        Sec_P(1,1)=-Sec_P(1,1);
```

```

                if( checkifinternal(First_P, Sec_P, Ex_P)==1) % checking if the intersection is internal
                    lmin=len;
                    check1=1;
                end
            end
        if(check1==1)
            if((lam+lmin/det_d)<1)
                Exit.coord=P+(lmin/det_d)*d;
                if(check_h(sect_num,Se,Exit.coord)==1)
                    out.check=1;
                    out.lambda=lam+lmin/det_d;
                    out.enterp=Exit.coord;
                end
            end
        end
    end
end
end
end

```

---

### ***Exitloc***

% THIS FUNCTION DETERMINES THE EXIT POINT, EXIT LINENUMBER OR % VERTEX NUMBER AND THE LAMBDA( which measures the fraction of distance % moved along direction d).

```
function Exit=exitloc(P,d,sect_num,Se,S,N,enterp,prev_Exit,prev_cuse)
```

```
d=d(1,1:3);
% P is the coordinates row of size 3.
% d is the direction. row of size 3.
```

```
bignum=1000000;
smallnum=0.001;
```

```
Exit.linenum=0;
Exit.nodenum=0;
Exit.tbnum=0;
```

```
det_d=sqrt(d(1,1).^2+d(1,2).^2);
```

```
if( det_d<smallnum)
```

```
    if(abs(d(1,3))<smallnum)
```

```
        Exit.lambda=5;    % any # greater than 1 is fine
        Exit.coord=P;
    elseif( d(1,3)>smallnum)
```

```

        Exit.coord=[ P(1,1) P(1,2) Se(sect_num).hmax];
        Exit.lambda= ( Se(sect_num).hmax-P(1,3) )/d(1,3) ;
        Exit.tbnum=Se(sect_num).hmaxnum;
elseif( d(1,3)<-smallnum)

        Exit.coord=[ P(1,1) P(1,2) Se(sect_num).hmin];
        Exit.lambda=( Se(sect_num).hmin-P(1,3) )/d(1,3) ;
        Exit.tbnum=Se(sect_num).hminnum;
end

else

norm_d=d(1,1:2)/sqrt(d(1,1).^2+d(1,2).^2);

lmin=bignum;
for(i=1:Se(sect_num).n) % for all faces

        line=Se(sect_num).line(1,i);
        if( (line.alpha*d(1,1:2))/det_d < -smallnum) % if opposing the inward gradient

                leng=( P(1,1:2)*line.alpha'- line.c);
                leng=abs(leng/(line.alpha*norm_d(1,1:2)));
                if(leng>smallnum)
                        len=leng;
                        clear leng;

                        lambda=len/det_d;
                        Ex_P=P(1,1:2)+lambda*d(1,1:2);

                        First_P=N( S(sect_num).ver(1,i,:));
                        First_P(1,1)=-First_P(1,1);

                        if(i==Se(sect_num).n)

                                Sec_P=N( S(sect_num).ver(1,1,:));
                        else

                                Sec_P=N( S(sect_num).ver(1,i+1,:));
                        end
                        Sec_P(1,1)=-Sec_P(1,1);

                if( checkifinternal(First_P, Sec_P, Ex_P)==1) % checking if the intersection is internal
                        if(len<lmin & (globedis([Ex_P;enterp(1,1:2)]) >smallnum) )
                                if(len<lmin)

                                        if((prev_cuse==sect_num)& prev_Exit.nodenum>0)
                                                glance=1;
                                        else
                                                glance=0;
                                        end

                                        if(glance==0|(globedis([Ex_P;enterp(1,1:2)]) >smallnum))

                                                minnum=i;
                                                lmin=len;
                                        end
                                        % the crossed face i
                                end
                        end
                end
end

```

```

end
end
end

Exit.lambda=lmin/det_d;

Exit.coord=P+Exit.lambda*d;

Exit.linenum=Se(sect_num).line(1,minnum).num;

Exit.nodenum=0;

% checking if the exit point is a node.

i=minnum;

N1=N(S(sect_num).ver(1,i,:));
N1(1,1)=-N1(1,1);

if(i==Se(sect_num).n)
    n2num=S(sect_num).ver(1,1);
    N2=N(S(sect_num).ver(1,1,:));
else
    n2num=S(sect_num).ver(1,i+1);
    N2=N(S(sect_num).ver(1,i+1,:));
end
N2(1,1)=-N2(1,1);

if(checkifsame( Exit.coord,N1 )==1) % checking if the exit point is a node.
    Exit.nodenum=S(sect_num).ver(1,i);
elseif(checkifsame( Exit.coord,N2 )==1)
    Exit.nodenum=n2num;
end

% checking if the exited point lies within the floor and ceilings
if(abs(Exit.coord(1,3)-Se(sect_num).hmax)<smallnum)
    Exit.tbnum=Se(sect_num).hmaxnum;
elseif(( Exit.coord(1,3)>Se(sect_num).hmax ))
    Exit.tbnum=Se(sect_num).hmaxnum;
    Exit.linenum=0;
    Exit.nodenum=0;
    Exit.coord=P+(Exit.coord-P)*(Se(sect_num).hmax - P(1,3)) / ( Exit.coord(1,3) - P(1,3) );
    Exit.lambda=sqrt( (Exit.coord(1,1)-P(1,1))^2 + (Exit.coord(1,2)-P(1,2))^2 ) / det_d;

elseif(abs(Exit.coord(1,3)-Se(sect_num).hmin)<smallnum)
    Exit.tbnum=Se(sect_num).hminnum;

```

```

elseif( Exit.coord(1,3)<Se(sect_num).hmin )

    Exit.tbnum=Se(sect_num).hminnum;
    Exit.linenum=0;
    Exit.nodenum=0;
    Exit.coord= P + (Exit.coord-P)*( P(1,3)-Se(sect_num).hmin ) / (P(1,3) - Exit.coord(1,3));
    Exit.lambda= sqrt( (Exit.coord(1,1)-P(1,1))^2 + (Exit.coord(1,2)-P(1,2))^2 ) / det_d;

end
end

```

---

### ***Find\_ext\_sect***

% THIS FUNCTION WILL DETERMINE THE EXTREME FACES OF THE  
% DEFINED AIRSPACE.

```

function ext_sect=find_ext_sect(Adjsec,S)

ext_sect=struct('sect',[],'loc',[]);

n_s=length(S);
ext_sect.sect(1)=n_s;
ext_sect.sect(2)=n_s-1;
ext_sect.sect(3)=n_s-2;
ext_sect.sect(4)=n_s-3;

ext_sect.loc(1:4)=1;

num_ext_sect=4;

for(i=1:length(Adjsec))

    check1=length(Adjsec(i).pos);
    check2=length(Adjsec(i).neg);

    if(check1==0 | check2==0)

        if(check1==0)

            for(j=1:check2)

                if(ispresent_adj(Adjsec(i).neg(j).sect,ext_sect.sect)==0)
                    num_ext_sect=num_ext_sect+1;
                    ext_sect.sect(num_ext_sect)=Adjsec(i).neg(j).sect;
                    ext_sect.loc(num_ext_sect)=Adjsec(i).neg(j).loc;
                end
            end
        end
    else

        for(j=1:check1)

            if(ispresent_adj(Adjsec(i).pos(j).sect,ext_sect.sect)==0)
                num_ext_sect=num_ext_sect+1;
                ext_sect.sect(num_ext_sect)=Adjsec(i).pos(j).sect;
            end
        end
    end
end

```





```

                                next=next_sect_tb(Exit.tbnum,Adjsectb,S,Se,Exit.coord,d);
                                end
                                end
                                end
                                end
                                end

```

---

### ***Getnext\_afterdummy***

% THIS FUNCTIO WILL DETERMINE THE SECTOR MODULE THE FLIGHT  
 % ENTERS AFTER EXITING A DUMMY SECTOR OR A VACUUM

function out=getnext\_afterdummy(P,d,lam,Fp,f,i,ext\_sect,Se,S,N)

% returns the entered sector module, entry segment, entry lambda, entry point,entry time & entry direction.

```

    first_i=i;

```

```

    ans1.check=0;
    while( ans1.check==0) & (i<Fp(f).n)

```

```

        for(j=1:length(ext_sect.sect(1,:)))

```

```

            ans1=checkif_crossed(P,d,ext_sect.sect(1,j),ext_sect.loc(1,j),Se,S,N,lam);

```

```

            if(ans1.check==1)

```

```

                next=ext_sect.sect(1,j);
                lambda=ans1.lambda;
                enterp=Fp(f).wp(i,1:3)+lambda*d(1,1:3);
                entert=Fp(f).twp(i,1) + lambda*( Fp(f).twp(i+1,1)-Fp(f).twp(i,1) );
                last_i=i;
                last_d=d;
                break

```

```

            end

```

```

        end

```

```

    i=i+1;

```

```

    lam=0;

```

```

    if(i<Fp(f).n)

```

```

        P=Fp(f).wp(i,1:3);

```

```

        d=Fp(f).wp(i+1,1:3)-Fp(f).wp(i,1:3);

```

```

    end

```

```

    if(i==Fp(f).n & ans1.check==0)

```

```

                                next=0;
                                last_i=i;
                                lambda=[];
                                enterp=[];
                                entert=[];
                                last_d=[];

                                end

                                end

                                out.next=next;
                                out.last_i=last_i;
                                out.lambda=lambda;
                                out.enterp=enterp;
                                out.entert=entert;
                                out.last_d=last_d;

```

---

### ***get\_dummy***

%THIS FUNCTION DEFINES THE DUMMY SECTOR AROUND A DEFINED  
% AIRSPACE

```
function ans1=get_dummy(N,S,h)
```

```
n_n=length(N(:,1));
n_s=length(S);
```

%DETERMINING THE BOUNDARIES OF AIRSPACE.

```
max_long=max( N(:,1));
min_long=min( N(:,1));
```

```
max_lat=max( N(:,2));
min_lat=min( N(:,2));
```

```
N(n_n+1,:)= [ 160 65];
N(n_n+2,:)= [ 0 65];
N(n_n+3,:)= [0 -10];
N(n_n+4,:)= [160 -10];
N(n_n+5,:)= [max_long+0.2 max_lat+0.2];
N(n_n+6,:)= [min_long-0.2 max_lat+0.2];
N(n_n+7,:)= [min_long-0.2 min_lat-0.2];
N(n_n+8,:)= [max_long+0.2 min_lat-0.2];
```

% DEFINING THE SECTOR MODULES IN TERMS OF NODES

```
S(n_s+1).ver=[ n_n+1 n_n+2 n_n+6 n_n+5];
S(n_s+2).ver=[ n_n+2 n_n+3 n_n+7 n_n+6];
S(n_s+3).ver=[ n_n+3 n_n+4 n_n+8 n_n+7];
S(n_s+4).ver=[ n_n+4 n_n+1 n_n+5 n_n+8];
```

```

% NAMING
S(n_s+1).sector_name=['DUM'];
S(n_s+2).sector_name=['DUM'];
S(n_s+3).sector_name=['DUM'];
S(n_s+4).sector_name=['DUM'];

```

```

% SECTOR FPA NAMING
S(n_s+1).sectfpa=['DUDU'];
S(n_s+2).sectfpa=['DUDU'];
S(n_s+3).sectfpa=['DUDU'];
S(n_s+4).sectfpa=['DUDU'];

```

```

h(n_s+1:n_s+4,1)=0;
h(n_s+1:n_s+4,2)=1000;

```

```

ans1.S=S;
ans1.N=N;
ans1.h=h;

```

### ***Get\_main\_Adj***

```

function main_Adj=get_main_Adj(Adj,main_S,S)

```

```

% THIS FUNCTION WILL DETERMINE THE SECTORS ADJACENT TO EACH OF
% THE SECTOR.

```

```

main_Adj(1:length(main_S))=struct('sect',[]);
num=0;

```

```

for(i=1:length(main_S))

```

```

    for(j=1:length(main_S(i).subs))

```

```

        module_j=main_S(i).subs(1,j);

```

```

        for(k=1:length(Adj(module_j).sect))

```

```

            adj_module=Adj(module_j).sect(1,k);

```

```

            adj_sect=S(adj_module).sector_num;

```

```

            if( ispresent_adj(adj_sect,main_Adj(i).sect)==0 & (adj_sect~=i))

```

```

                len=length(main_Adj(i).sect);

```

```

                main_Adj(i).sect(1,len+1)=adj_sect;

```

```

            end

```

```

        end

```

```

    end

```

```

end

```

### ***Get\_main\_Adjsecnode***

```
function main_Adjsecnode=get_main_Adjsecnode(Adjsecnode,main_S,S)

% THIS FUNCTION WILL DETERMINE THE SECTORS ADJACENT TO EACH OF
% THE VERTICES.

main_Adjsecnode(1:length(Adjsecnode))=struct('sect',[]);
num=0;

for(i=1:length(Adjsecnode))

    for(j=1:length(Adjsecnode(i).sect))

        adj_modulej=Adjsecnode(i).sect(1,j);
        adj_sect=S(adj_modulej).sector_num;

        if(ispresent_adj(adj_sect,main_Adjsecnode(i).sect)==0)

            len=length(main_Adjsecnode(i).sect);

            main_Adjsecnode(i).sect(1,len+1)=adj_sect;

        end

    end

end
```

---

---

### ***Main***

```
% THIS IS THE MAIN FILE THAT DOES ALL THE COMPUTATIONS.

function out=main

sect_filename='zjxzma9705';
Fp_filename='rtout.nov_12';

% =====
% READING SECTOR DATA.
% =====

% SDAT DATA.
caseof='clockwise';

% Node=read_sdat_node(sect_filename);
% out=read_sdat_sect(sect_filename,Node);
% S=out.S;
% h=out.h;
% N=Node.N;
% clear out

% -----
```

```

% EXAMPLE DATA
    caseof='anticlockwise';
    N=node_scan;
    S=sect_scan(N);
    h=height_scan;

%-----
%=====

% READING FLIGHT DATA (either regular trajectory or optimized trajectory)

if(strcmp('rtout',Fp_filename(1:5))==1)
    Fp=readetms(Fp_filename);
elseif(strcmp('opout',Fp_filename(1:5))==1)
    Fp=read_opt_reqd(Fp_filename);
end

%=====

% This will read the sector input file and does pre-processing fpr sectors.
ans1=prepro_sectors(S,N,h,caseof);
N=ans1.N;
S=ans1.S;
Se=ans1.Se;
ext_sect=ans1.ext_sect;
Node=ans1.Node;
main_S=ans1.main_S;
Adj=ans1.Adj;
Adjsec=ans1.Adjsec;
Adjsecnode=ans1.Adjsecnode;
Adjsectb=ans1.Adjsectb;
main_Adj=ans1.main_Adj;
main_Adjsecnode=ans1.main_Adjsecnode;
clear ans1;

% This will read the input file for airports and identifies the module that encompasses the
% airport.
airport=prepro_airports(S,Se)

%=====

% This will identify the first point in the trajectory that lies in the defined airspace.

Fp=prepro_Fp(Fp,airport,ext_sect,Se,S,N)

% This will do the processing and the post-processing.
ans2=pro_postpro(Fp,Se,S,N,Adjsec,Adjsecnode,Adjsectb,Adj,ext_sect,main_S);

Fp=ans2.Fp;
main_S=ans2.main_S;
clear ans2;

%=====
out.N=N;
out.S=S;
out.main_S=main_S;
out.Se=Se;
out.Node=Node;

```

```

out.Adj=Adj;
out.Adjsec=Adjsec;
out.Adjsecnode=Adjsecnode;
out.Adjsectb=Adjsectb;
out.airport=airport;
out.Fp=Fp;
out.ext_sect=ext_sect;
out.main_Adj=main_Adj;
out.main_Adjsecnode=main_Adjsecnode;

```

---



---

### ***Main\_occup***

```

% THIS FUNCTION WILL ASSOCIATE FLIGHTS THAT PASS THROUGH A
%SECTOR

```

```

function out_main_S=main_occup(Fp,main_S)

```

```

for(f=1:length(Fp))

```

```

    for(i=1:length(Fp(f).main_path))

```

```

        main_secti=Fp(f).main_path(1,i).sect;

```

```

        if(strcmp(main_S(main_secti).name,'DUM')==0)

```

```

            len=length(main_S(main_secti).occup);

```

```

            main_S(main_secti).occup(1,len+1).fnum=f;

```

```

            main_S(main_secti).occup(1,len+1).entert=Fp(f).main_path(1,i).entert;

```

```

            main_S(main_secti).occup(1,len+1).exitt=Fp(f).main_path(1,i).exitt;

```

```

        end

```

```

    end

```

```

end

```

```

out_main_S=main_S;

```

---



---

### ***Next\_sect\_line***

```

function next=next_sect_line(Exit,cuse,Adjsec,Se,S,N,d)

```

```

% THIS FUNCTION DETERMINES THE NEXT SECTOR THE TRAJECTORY
% PASSES THROUGH AFTER EXITING ONE SECTOR ACROSS A FACE.

```

```

linenum=Exit.linenum;

```

```

Exitp=Exit.coord;

```

```

tbnun=Exit.tbnun;

```

```

if(ispresent(cuse,Adjsec(linenum).pos)==1)
    nextset=Adjsec(linenum).neg;
    otherset=Adjsec(linenum).pos;
elseif(ispresent(cuse,Adjsec(linenum).neg)==1)
    nextset=Adjsec(linenum).pos;
    otherset=Adjsec(linenum).neg;
end
next=0;
next=next_sect_line_oneside(nextset,S,N,Se,Exit,d);

if(next==0)
    nextset=otherset;
    next=next_sect_line_oneside(nextset,S,N,Se,Exit,d);
end

```

---

### ***Next\_sect\_node***

```

function next=next_sect_node(Exit,cuse,Adjsecnode,Se,d)
% THIS FUNCTION DETERMINES THE NEXT SECTOR THE TRAJECTORY
% PASSES THROUGH AFTER EXITING ONE SECTOR VIA A VERTEX.

nodenum=Exit.nodenum;
Exitp=Exit.coord;
tbnnum=Exit.tbnnum;
smallnum=0.001;

next=0;

for(i=1:length(Adjsecnode(nodenum).sect))
%     if(Adjsecnode(nodenum).sect(1,i)~=cuse)

        sect=Adjsecnode(nodenum).sect(1,i);
        loc=Adjsecnode(nodenum).loc(1,i);
        type=Se(sect).node(loc).type;

        check=1;

        if(check_h(sect,Se,Exitp)==1)

            alpha1=Se(sect).line(1,loc).alpha;

```



```

    if( abs(d(1,3))<smallnum)
        if( abs( Se(sect).hmin-Exitp(1,3))<smallnum | abs( Se(sect).hmax-Exitp(1,3))<smallnum )
            if(inpolygon(Exitp(1,1),Exitp(1,2),S(sect).long,S(sect).lat)>0 &
inpolygon(inc_Exitp(1,1),inc_Exitp(1,2),S(sect).long,S(sect).lat)>0 )
                next=sect;
                break
            end
        end
    end
    % if moving up
    elseif(d(1,3)>0)
        if( abs( Se(sect).hmin-Exitp(1,3))<smallnum)
            if(inpolygon(Exitp(1,1),Exitp(1,2),S(sect).long,S(sect).lat)>0 &
inpolygon(inc_Exitp(1,1),inc_Exitp(1,2),S(sect).long,S(sect).lat)>0 )
                next=sect;
                break
            end
        end
    end
    % if moving down
    elseif(d(1,3)<0)
        if( abs( Se(sect).hmax-Exitp(1,3))<smallnum)
            if(inpolygon(Exitp(1,1),Exitp(1,2),S(sect).long,S(sect).lat)>0 &
inpolygon(inc_Exitp(1,1),inc_Exitp(1,2),S(sect).long,S(sect).lat)>0 )
                next=sect;
                break
            end
        end
    end
end
end
end

```

---

### ***Occup***

```

function out_Fp=occup(Fp,Se,S,N,Adjsec,Adjsecnode,Adjsectb,Adj_ext_sect)

% THIS FUNCTION DETERMINES THE SECTORS THROUGH WHICH THE
% FLIGHT PASSES.
tic

smallnum=0.001;
% The number of flight segments that need to be checked before concluding that
% the flight moves out of the defines airspace.
max_segment=1;

for(f=1:length(Fp))

```

```

if(rem(f,10)==0)
    f
end

if(Fp(f).n>0)
i=Fp(f).start_seg;

if(length(Fp(f).start_point)>0)
    X=Fp(f).start_point(1,1:3);

else
    X=[0 0 0];
end
% entry point into cuse
enterp=X;

if(i<Fp(f).n)
    d=Fp(f).wp(i+1,1:3)-Fp(f).wp(i,1:3);
end
lam=Fp(f).start_lam;
entert=Fp(f).start_time;
sofar=0;
cuse=Fp(f).omodule;

prev_cuse=cuse;
Exit.nodenum=0;
start=1;

while(i<Fp(f).n)

    if(i==30)
        i
    end

    prev_Exit=Exit;
    Exit=exitloc(X,d,cuse,Se,S,N,enterp,prev_Exit,prev_cuse);

    lam=lam+Exit.lambda;

    if(lam<(1-smallnum))
        start=0;
        X=Exit.coord;
        exitt=Fp(f).twp(i,1)+lam*(Fp(f).twp(i+1,1)-Fp(f).twp(i,1));

        if( abs(exitt-entert)>smallnum )
            sofar=sofar+1;
            Fp(f).path(sofar).sect=cuse;
            Fp(f).path(sofar).entert=entert;
            Fp(f).path(sofar).exitt=exitt;
            entert=exitt;
            enterp=Exit.coord;
        end

        prev_cuse=cuse;
        cuse=getnextsect(Exit,cuse,Adjsec,Adjsecnode,Adjsectb,Se,S,N,d);
        if(cuse==0)
            if(prev_cuse>length(S)-4)

```

```

%
%d',f,prev_cuse)

sprintf('Vacuum encountered by Flight %d after passing Sector
n_s=length(S);
check_out=0;
for(z=1:4)
    if(Exit.linenum==Se(n_s+1-z).line(1,1).num)
        check_out=1;
    end
end
if(check_out==0)
    ans1=getnext_afterdummy(X,d,lam,Fp,f,i,ext_sect,Se,S,N);
    cuse=ans1.next;
    i=ans1.last_i;
    lam=ans1.lambda;
    d=ans1.last_d;
    X=ans1.enterp;
    entert=ans1.entert;
end

% if the trajectory goes outside the defined sectors end
if(cuse==0)
    i=Fp(f).n;
end

end

elseif(lam>(1+smallnum))
    i=i+1;
    if(i==Fp(f).n)
        exitt=Fp(f).twp(i,1);
        sofar=sofar+1;
        Fp(f).path(sofar).sect=cuse;
        Fp(f).path(sofar).entert=entert;
        Fp(f).path(sofar).exitt=exitt;
    else
        lam=0;
        X=Fp(f).wp(i,1:3);
        d=Fp(f).wp(i+1,1:3)-Fp(f).wp(i,1:3);
    end

elseif(abs(lam-1)<smallnum)
    exitt=Fp(f).twp(i+1,1);
    sofar=sofar+1;
    Fp(f).path(sofar).sect=cuse;

```



toc

---

---

%THIS FUNCTION PLOTS THE HISTOGRAM SHOWING OCCUPANCY AND  
%THE LOCATION OF SECTOR i.

function plot\_hist\_view(main\_S,S,Se,i)

subplot(2,1,1)  
usmap;  
axis equal  
hold;

interval=15;

for(j=1:length( main\_S(i).subs )

    secti=main\_S(i).subs(1,j);

        plot(S(secti).long,S(secti).lat,'b');  
        fill(S(secti).long,S(secti).lat,'r')

end

subplot(2,1,2)

hist\_array=[];  
if(length(main\_S(i).occup)>0)  
    for(j=1:length(main\_S(i).occup))  
        entert=main\_S(i).occup(j).entert;  
        exitt=main\_S(i).occup(j).exitt;  
        ai=entert:interval:exitt;  
        l\_h=length(hist\_array);  
        l\_a=length(ai);  
  
        hist\_array(l\_h+1:l\_h+l\_a)=ai;  
        clear ai;  
    end

min\_t=min(hist\_array);  
max\_t=max(hist\_array);

num=round((max\_t-min\_t)/interval);

hist(hist\_array,num)

sectnum=int2str(i);

```

title_h=[' Occupancy of Sector ',main_S(i).label, ': ', main_S(i).name];

% title(title_h)

subplot(2,1,1)
xlabel('Longitude (deg) ')
ylabel('Latitude (deg)')

subplot(2,1,2)
xlabel('GMT (min) ')
ylabel('Traffic (acft)')

grid
else

    printf(' No flights cross this sector')
end

```

---



---

### ***Preproadj***

% THIS FUNCTION IDENTIFIES THE SECTORS LYING AROUND EACH  
% SECTOR AND STORES IN THE DATA STRUCTURE ADJ.SECT

```
function Adj=preproadj(S,Se,Adjsecnode)
```

```
% Adj=struct('sect',[]);
```

```
    for(i=1:length(S)) %for all sectors
```

```
        Adj(i).sect=[];
```

```
        adjnum=0;
```

```
        hi_min=Se(i).hmin;hi_max=Se(i).hmax;
```

```
        for(j=1:length(S(i).ver)) %for all vertices of sector i
```

```
            v=S(i).ver(1,j);
```

```
            for(k=1:length(Adjsecnode(v).sect)) % for all sectors adjacent to ith sector w.r.t vth
```

```
vertex.
```

```
                s=Adjsecnode(v).sect(k);
```

```
                if(s~=i)
```

```
                    hs_min=Se(s).hmin;    hs_max=Se(s).hmax;
```

```
%
(hs_max>=hi_min) & (hs_max<=hi_max) ))    if( ( (hs_min>=hi_min) & (hs_min<=hi_max) ) | (
```

```
if( ( hs_min<=hi_max) & ( hs_max>=hs_min) )
```

```
                    if( ispresent_adj(s,Adj(i).sect)==0)
```

```
                        adjnum=adjnum+1;
```

```
                        Adj(i).sect(adjnum)=s;
```

```
                    end
```

```
                end
```

```
            end
```

```

end
end
end

```

---



---

### ***Preproadjsec***

% PREPROCESSING ROUTINE WHICH ESTIMTES THE ADJACENCY OF SUB-  
 %SECTORS IN RELATION TO THE VERTICAL FACES & NUMBERS THE  
 %VERTICAL FACES UNIQUELY

```
function ans2=preproadjsec_mod_node(Se,S,Adjsecnode)
```

```
Adjsec=struct('pos',[],'neg',[]);
```

```
linenum=0;
smallnum=0.001;
% for checking equality of float variables.
```

```
for(i=1:length(Se))
```

```
% for all sub-sectors
```

```
    sectsize=Se(i).n;
    for(j=1:sectsize)
```

```
% for all vertical faces
```

```
        check=Se(i).line(1,j).flag;
```

```
% check if already numbered
        if(check==1)
```

```
            count_pos=0;
            count_neg=0;
```

```
            alpha1=Se(i).line(1,j).alpha;
            c1=Se(i).line(1,j).c;
            linenum = linenum+1;
```

```
            v1=S(i).ver(1,j);
```

```
            if(j==sectsize)
                v2=S(i).ver(1,1);
            else
                v2=S(i).ver(1,j+1);
            end
```

```
            if(c1>=0)
```

```

        count_pos=count_pos+1;
        Adjsec(linenum).pos(count_pos).sect=i;
        Adjsec(linenum).pos(count_pos).loc=j;

    else
        count_neg=count_neg+1;
        Adjsec(linenum).neg(count_neg).sect=i;
        Adjsec(linenum).neg(count_neg).loc=j;
    end

    Se(i).line(1,j).num =linenum;

% number the vertical face
    Se(i).line(1,j).flag =0;

    near_sect=getsectors_adj(v1,v2,Adjsecnode);

    if(length(near_sect)>0)
        for(n=1:length(near_sect) )

% for all sectors after ith sector

            k=near_sect(n);
            for(l=1:Se(k).n

% for all vertical faces
                checklm=(Se(k).line(1,l).flag);

                    if( checklm ==1)

                        alpha2=Se(k).line(1,l).alpha;
                        c2=Se(k).line(1,l).c;
                        v3=S(k).ver(1,l);

                        if(l==Se(k).n)
                            v4=S(k).ver(1,l);
                        else
                            v4=S(k).ver(1,l+1);
                        end

                        check1=0;
                        if( (abs(alpha1(1,1)-alpha2(1,1))<smallnum)      &      (abs(alpha1(1,2)-
alpha2(1,2))<smallnum) & (abs( c1-c2)<smallnum) )
                            check1=1;
                        elseif( (abs(alpha1(1,1)+alpha2(1,1))<smallnum)      &
( abs(alpha1(1,2)+alpha2(1,2))<smallnum) & (abs( c1+c2)<smallnum) )
                            check1=1;
                        end

                    if(check1==1 )

%
checking if adjacent

                        if( ((v3==v1) | (v3==v2)) & ((v4==v1) | (v4==v2)) )
                            Se(k).line(1,l).num =linenum;
                            Se(k).line(1,l).flag =0;

                            if(c2>=0)

```



```

Adjsectb(num).sect(nsec)=i;
Se(i).hminnum=num;

    for(j=i+1:length(S) )

        if(Se(j).hminflag==0 & abs(Se(j).hmin-h)<smallnum )
            nsec=nsec+1;
            Adjsectb(num).sect(nsec)=j;
            Se(j).hminflag=1;
            Se(j).hminnum=num;

            elseif(Se(j).hmaxflag==0 & abs(Se(j).hmax-h) < smallnum )
                nsec=nsec+1;
                Adjsectb(num).sect(nsec)=j;
                Se(j).hmaxflag=1;
                Se(j).hmaxnum=num;
            end

        end

    end

if(Se(i).hmaxflag==0)
num=num+1;
nsec=1;
Adjsectb(num).sect(nsec)=i;
h=Se(i).hmax;
Se(i).hmaxflag=1;
Se(i).hmaxnum=num;

    for(j=i+1:length(S) )

        if(Se(j).hminflag==0 & abs(Se(j).hmin-h)<smallnum )
            nsec=nsec+1;
            Adjsectb(num).sect(nsec)=j;
            Se(j).hminflag=1;
            Se(j).hminnum=num;

            elseif(Se(j).hmaxflag==0 & abs(Se(j).hmax-h) < smallnum )
                nsec=nsec+1;
                Adjsectb(num).sect(nsec)=j;
                Se(j).hmaxflag=1;
                Se(j).hmaxnum=num;
            end

        end

    end

end

ansadjtb.Se=Se;
ansadjtb.Adjsectb=Adjsectb;

```

---

### ***Prepronode***

% PREPROCESSING ROUTINE WHICH ESTIMATES THE ADJACENCY OF  
%SECTORS IN RELATION TO THE NODES.

```
function Adjsecnode=prepronode2(N,S)

Adjsecnode=struct('sect',[],'loc',[]);
Adjsecnode(length(N)).sect=[];
Adjsecnode(length(N)).loc=[];

for(i=1:length(S))

    for(j=1:length(S(i).ver(1,:)))

        nodenum=S(i).ver(1,j);
        cur_size=length(Adjsecnode(nodenum).sect) + 1;
        Adjsecnode(nodenum).sect(cur_size)=i;
        Adjsecnode(nodenum).loc(cur_size)=j;
    end
end
```

---

### ***Prepro\_airports***

% THIS FUNCTION READS & STORES THE LAT-LONG OF THE AIRPORTS.

```
function airport=prepro_airports(S,Se)

fp1=fopen('airports3.in');

airport=struct('latlong',[],'name',[],'modulenum',[]);

i=0;
while (feof(fp1)==0)

    i=i+1;

    airport(i).latlong=fscanf(fp1,'%g %g',[1 2]);
    airport(i).name=fscanf(fp1,'%s',1);
    %airport(i).latlong(1,1)=-airport(i).latlong(1,1);
end

fclose(fp1);

for(i=1:length(airport))

    airport(i).modulenum=0;

    for(j=1:length(S))
```

```

                if(Se(j).hmin==0)
                    if(inpolygon(airport(i).latlong(1,1),airport(i).latlong(1,2),S(j).long,S(j).lat)>0)
                        airport(i).modulenum=j;
                        break
                    end
                end
            end
        end
    end
end

```

---

### ***Prepro\_sdat***

% PREPROCESSING ROUTINE WHICH DOES THE MATHEMATICAL  
% REPRESENTATION OF SECTOR EDGES AND VERTICES

function Se=prepro\_sdat(N,S,sub\_sect\_ht)

bignum=1000000;  
smallnum=0.001;

% for slope of angle 90

Se.n=0;  
Se.line.alpha=[0 0];

% number of nodes in the sector  
% line is a array of structures which store info

% binding edges.: normalized outward normal,

normalized constant,

% linenumber & a boolean which will be 1 if the  
% line is numbered.

Se.line.c=0;

Se.line.flag=1;  
Se.line.num=0;

% estimates the # of nodes in a sector

for(i=1:length(S))

    Se(i).n = length(S(i).ver);

end

% estimates the slope, intercept of the binding edges of the sectors  
for(i=1:length(S))                      % for all sectors

    sectsize=Se(i).n;

    for(j=1:sectsize)                      % for all vertical faces

        if(S(i).ver(1,j)==0 )  
            j=j-1;

            disp('Prog. warning: unnecessary alpha & c calculations done')

```

        break
    end ;

    x1=-N(S(i).ver(1,j),1);
    y1=N(S(i).ver(1,j),2);
    coord1=[x1 y1]';

    if(j==sectsize )

        x2=-N(S(i).ver(1,1),1);
        y2=N(S(i).ver(1,1),2);

    else

        x2=-N(S(i).ver(1,j+1),1);
        y2=N(S(i).ver(1,j+1),2);

    end

    coord2=[x2 y2]';

    if(j==1 )

        coord0=[-N(S(i).ver(1,sectsize),1) N(S(i).ver(1,sectsize),2) ]';

    else

        coord0=[-N(S(i).ver(1,j-1),1) N(S(i).ver(1,j-1),2) ]';

    end

    alpha=-[(y2-y1) (x2-x1)];

    c=alpha*coord1;
    Se(i).line(1,j).c=c;
    Se(i).line(1,j).flag=1;
    Se(i).line(1,j).num=0;

    det_alpha=sqrt(alpha(1,1)^2+alpha(1,2)^2);
    Se(i).line(1,j).alpha=alpha/det_alpha;
    Se(i).line(1,j).c=Se(i).line(1,j).c/det_alpha;

    ref= coord0-coord1;

    if(alpha*ref>=0)

        Se(i).node(j).type=1;           % means convex type of vertex

    else

        Se(i).node(j).type=2;           % means non-convex type of vertex

    end

```

```

        end

    end
    clear i;

    for(i=1:length(S) )

        Se(i).hmin=sub_sect_ht(i,1);
        Se(i).hmax=sub_sect_ht(i,2);
        Se(i).hminflag=0;
        Se(i).hminnum=0;
        Se(i).hmaxflag=0;
        Se(i).hmaxnum=0;

    end
    %     Se(i).line(1,1:Se(i).n).flag=1;
    %     Se(i).line(1,1:Se(i).n).num=0;

```

---

### ***Prepro\_Sectors***

% THIS FUNCTION DOES THE PRE-PROCESSING OF THE SECTOR DATA.

function out=prepro\_sectors(S,N,h)

```

% Node=read_sdat_node(sect_filename);
% out=read_sdat_sect(sect_filename,Node);
% S=out.S;
% h=out.h;
% N=Node.N;
% clear out

```

```

    ans1=get_dummy(N,S,h);

```

```

    S=ans1.S;

```

```

    N=ans1.N;

```

```

    h=ans1.h;

```

```

    clear ans1;

```

```

    ans1=get_main_S(S);

```

```

    main_S=ans1.main_S;

```

```

    S=ans1.S;

```

```

    clear ans1;

```

%This determines the mathematical representation of sectors

```

    Se=prepro_sdat(N,S,h)

```

```

%Se=prepro(N,S,h)

```

% Preprocessing routine which estimates the adjacency of sectors in relation to the floor & ceiling.

% & numbers the horizontal faces uniquely

```
ansadjtb=preproadjsectb(Se,S);
Se=ansadjtb.Se;
Adjsectb=ansadjtb.Adjsectb
clear ansadjtb;
```

---

---

### ***Process\_Fp***

```
% THIS FUNCTION PROCESSES THE FLIGHT INFORMATION TO DETERMINE
% THE OCCUPANCY.
```

```
function out=process_Fp(a,start_time,end_time)
```

```
N=a.N;
S=a.S;
main_S=a.main_S;
Se=a.Se;
Node=a.Node;
Adj=a.Adj;
Adjsec=a.Adjsec;
Adjsecnode=a.Adjsecnode;
Adjsectb=a.Adjsectb;
airport=a.airport;
ext_sect=a.ext_sect;
main_Adj=a.main_Adj;
main_Adjsecnode=a.main_Adjsecnode;
```

```
tic
```

```
%Fp=read_sdat('sample_sdat.in');
% Fp=readetms('rtout.nov_12')
```

```
Fp_filename='opout.nov_12.cardinal'
```

```
Fp=read_opt_reqd_t(Fp_filename,airport,start_time,end_time)
```

```
% Plotting the sectors and flight trajectory
```

```
% view_sect_Fp(N,S,Se,Fp,210)
```

```
% This will identify the first point in the trajectory that lies in the defined airspace.
Fp=prepro_Fp(Fp,airport,ext_sect,Se,S,N);
```

```
% Determination of occupancy
```

```
Fp=occup(Fp,Se,S,N,Adjsec,Adjsecnode,Adjsectb,Adj,ext_sect);
```

```
Fp=get_mainpath(Fp,S);
```

```
main_S=main_occup(Fp,main_S);
```

```
toc
```

```
out.N=N;  
out.S=S;  
out.main_S=main_S;  
out.Se=Se;  
out.Node=Node;  
out.Adj=Adj;  
out.Adjsec=Adjsec;  
out.Adjsecnode=Adjsecnode;  
out.Adjsectb=Adjsectb;  
out.airport=airport;  
out.Fp=Fp;  
out.ext_sect=ext_sect;  
out.main_Adj=main_Adj;  
out.main_Adjsecnode=main_Adjsecnode;
```

---

### ***Prepro\_Fp***

```
% THIS FUNCTION DOES THE PRE-PROCESSING OF THE FLIGHT  
%INFORMATION
```

```
function out_Fp=prepro_Fp(Fp,airport,ext_sect,Se,S,N)
```

```
% if(strcmp('rtout',Fp_filename(1:5))==1)  
%     Fp=readetms(Fp_filename);  
% elseif(strcmp('opout',Fp_filename(1:5))==1)  
% Fp=read_opt_reqd(Fp_filename);  
% end
```

```
for(f=1:length(Fp))
```

```
    Fp(f).omodule=0;
```

```
    if(length(Fp(f).wp)~=0)
```

```
        Fp(f).n=length(Fp(f).wp(:,1));
```

```
    else
```

```
        Fp(f).n=0;
```

```
    end
```

```
    for(i=1:length(airport))
```

```
        if( strcmp(Fp(f).origin,airport(i).name)==1)
```

```

        Fp(f).omodule=airport(i).modulenum;
        if(Fp(f).omodule~=0)
            Fp(f).start_point=Fp(f).wp(1,:);
            Fp(f).start_time=Fp(f).twp(1,1);
            Fp(f).start_seg=1;
            Fp(f).start_lam=0;
        end
    end
    break
end
end

end

if(Fp(f).omodule==0)

    for(j=1:length(S))

        if(Se(j).hmin==0)
            if(inpolygon(Fp(f).wp(1,1),Fp(f).wp(1,2),S(j).long,S(j).lat)>0)
                Fp(f).omodule=j;
                Fp(f).start_point=Fp(f).wp(1,:);
                Fp(f).start_time=Fp(f).twp(1,1);
                Fp(f).start_seg=1;
                Fp(f).start_lam=0;
            end
            break
        end
    end
end

end

if(Fp(f).omodule==0)
    i=1;lam=0;
    ans1.check=0;
    while( (ans1.check==0) & (i<Fp(f).n) )

        P=Fp(f).wp(i,:);
        d=Fp(f).wp(i+1,:)-Fp(f).wp(i,:);

        for(j=1:length(ext_sect.sect(1,:)))

            ans1=checkif_crossed(P,d,ext_sect.sect(1,j),ext_sect.loc(1,j),Se,S,N,lam);

            if(ans1.check==1)

                Fp(f).omodule=ext_sect.sect(1,j);
                Fp(f).start_point=Fp(f).wp(i,:)+ans1.lambda*d;
                Fp(f).start_time=Fp(f).twp(i,1) + ans1.lambda*( Fp(f).twp(i+1,1)-
Fp(f).twp(i,1) );

                Fp(f).start_seg=i;
                Fp(f).start_lam=ans1.lambda;
                break
            end
        end
    end
end

```

```

        i=i+1;
        lam=0;

        if(i==Fp(f).n & ans1.check==0)

            Fp(f).omodule=0;
            Fp(f).start_point=Fp(f).wp(i,:);
            Fp(f).start_time=Fp(f).twp(i,1);
            Fp(f).start_seg=i;
            Fp(f).start_lam=0;
        end
    end
end
end
end
out_Fp=Fp;

```

---

### ***Readetms***

% THIS FUNCTION READS THE ETMS DATA FROM THE INPUT FILE.

```
function Fp=readetms(filename)
```

```
%RLV_airports=[ ' ATL BHM BWI CHS CLT DCA DFW DTW EWR IAD ILM JAX JFK LGA MDW MIA ORD
PHL RDU RIC SAV SRQ TLH TPA PBI'];
RLV_airports=[ 'TLH JAX SAV MCO DAB TPA SRQ MLB CSG CHS CAE PBI MIA APF MTH EYW FLO ILM
ATL'];
```

```
%RLV_airports=['JAX MCO FLO'];
```

```
tic
```

```
f1=fopen(filename,'r')
```

```
f=0;
```

```
fname=fscanf(f1,'%s',1);
```

```
whilefeof(f1)==0 )
```

```
    fname=fname;
```

```
    move=fscanf(f1,'%s',1);
    clear move;
```

```
    fmodel=fscanf(f1,'%s',1);
```

```
    move=fscanf(f1,'%s',[1 4]);
    clear move;
```

```

origin=fscanf(f1,'%s',1);
dest=fscanf(f1,'%s',1);

check1=length( findstr(origin,RLV_airports));

check2=length( findstr(dest,RLV_airports));

check=check1+check2;
% check=check1;
if(check>0)

    f=f+1;
    Fp(f).fname=fname;
    Fp(f).fmodel=fmodel;
    Fp(f).origin=origin;
    Fp(f).dest=dest;

    move=fscanf(f1,'%s',[1 10]);
    clear move;

    num=fscanf(f1,'%d',1);

    Fp(f).n=num;

    for(i=1:num)

        move=fscanf(f1,'%s',1);
        clear move;
        temp_wp(i, 1:6)=fscanf(f1,'%f',[1 6]);
        Fp(f).wp(i, 1:6)=fscanf(f1,'%f',[1 6]);
%     end
        Fp(f).wp(:,1:3)=temp_wp(:, 1:3);
        Fp(f).twp(:,1)=temp_wp(:, 4);
        clear temp_wp;

        temp=Fp(f).wp(:, 1);
        Fp(f).wp(:, 1)=-Fp(f).wp(:, 2);
        Fp(f).wp(:, 2)=temp;
        clear temp;

    else

        move=fscanf(f1,'%s',[1 10]);
        clear move;

        num=fscanf(f1,'%d',1);

        move=fscanf(f1,'%s',[1 num*7]);
        clear move;

```

```

        end

        fname=fscanf(f1,'%s',1);

end

fclose(f1);
toc

=====

read_opt_reqd

% THIS FUNCTION READS THE DATA PERTAINING TO REQUIRED
% AIRPORTS FROM THE OPT TRAJECTORY FILE.

function Fp=read_opt_reqd(filename)

tic;
airport=prepro_airports_opout

reqd_airports=[ 'TLH'; 'JAX'; 'SAV' ;'MCO' ;'DAB' ;'TPA'; 'SRQ'; 'MLB'; 'CSG'; 'CHS'; 'CAE'; 'PBI'; 'MIA' ;'APF'
;'MTH' ;'EYW' ;'FLO'; 'ILM'; 'ATL'];

for(i=1:length( reqd_airports(:,1) ) )

    apt=reqd_airports(i,:);
    reqd_airports_latlong(i,:)=get_airport_latlong (apt, airport);

end

f1=fopen(filename,'r')

f=0;

fname=fscanf(f1,'%s',1);

while(feof(f1)==0 )

    move=fscanf(f1,'%s',[1 2]);
    clear move;

    num=fscanf(f1,'%d',1);

    for(i=1:num)

        move=fscanf(f1,'%s',1);
        clear move;

        tempwp(i, 1:4)=fscanf(f1,'%f',[1 4]);

    end

```

```

check1=check_latlong([-tempwp(1,2) tempwp(1,1)], reqd_airports_latlong);
check2=check_latlong([-tempwp(num,2) tempwp(num,1)], reqd_airports_latlong);

if( (check1+check2)>0)

    f=f+1;
    Fp(f).fname=fname;
    Fp(f).n=num;
    Fp(f).wp=tempwp(:,1:3);
    Fp(f).twp=tempwp(:,4);
    clear tempwp;

    temp=Fp(f).wp(:, 1);
    Fp(f).wp(:, 1)=-Fp(f).wp(:, 2);
    Fp(f).wp(:, 2)=temp;
    clear temp;
    Fp(f).origin=['DONT KNOW'];

end

fname=fscanf(f1,'%s',1);

end

fclose(f1);

toc

```

---



---

### ***Read\_sdat\_node***

% THIS FUNCTION READS THE NODE INFORMATION .

```

function Node=read_sdat_node(filename)

lf=length(filename);

filename(lf+1:lf+13)='export.nodes';

f1=fopen(filename,'r');
Node=struct('N',[],'name',[]);
i=0;
move=fgets(f1)

N_name=fscanf(f1,'%s',1);

whilefeof(f1)==0)

    i=i+1;
    lat=fscanf(f1,'%s',1);
    lat=getdeg( str2num(lat(1:6)) );

    long=fscanf(f1,'%s',1);
    long=getdeg( str2num(long(1:7)) );

    Node.N(i,:)=[ long lat];
    Node.name(i,:)=N_name;

```

```

        N_name=fscanf(f1,'%s',1);

end

fclose(f1);

```

---

### **Read\_sdat\_sect**

```

% THIS FUNCTION READS THE SECTOR INFORMATION.

function out=read_sdat_sect(filename,Node)

lf=length(filename);

filename(lf+1:lf+15)='.export.sectors';

f1=fopen(filename,'r')

sect_num=0;
newsect=0;

move=fgets(f1)

S=struct('ver',[],'lat',[],'long',[]);
M_len=0;
arts=[];
approach=[];
sector_name=[];

value=fscanf(f1,'%s',1);
sectfpa=value;
whilefeof(f1)==0)

    if( M_len >0 )

        newsect=1;
        sect_num=sect_num+1;
        num=0;
        h(sect_num,:)= [ str2num(value(3:5)) str2num(value(7:9))+1 ];
        S(sect_num).sectfpa=sectfpa;
        S(sect_num).arts=arts;
        S(sect_num).approach=approach;
        S(sect_num).sector_name=sector_name;

        value=fscanf(f1,'%s',1);

    end

while( newsect==1 & (feof(f1)==0))

    next_value=fscanf(f1,'%s',1);
    M_len=length( findstr('M-',next_value));

```

```

        ACN_len= ( length( findstr('A-',next_value)) + length( findstr('C-',next_value)) + length(
findstr('N-',next_value)));

        if ( (M_len>0) )

            newsect=0;

            num=num+1;
            nodenum=get_sdat_nodenum(Node.name,value);
            S(sect_num).ver(1,num)=nodenum;
            S(sect_num).long(1,num)=-Node.N(nodenum,1);
            S(sect_num).lat(1,num)=Node.N(nodenum,2);

            S(sect_num).long(1,num+1)=S(sect_num).long(1,1);
            S(sect_num).lat(1,num+1)=S(sect_num).lat(1,1);

        elseif(ACN_len>0 )
            newsect=0;
            arts=[];
            approach=[];
            sector_name=[];
            sectfpa=value;
            S(sect_num).long(1,num+1)=S(sect_num).long(1,1);
            S(sect_num).lat(1,num+1)=S(sect_num).lat(1,1);

        else
            num=num+1;
            nodenum=get_sdat_nodenum(Node.name,value);
            S(sect_num).ver(1,num)=nodenum;
            S(sect_num).long(1,num)=-Node.N(nodenum,1);
            S(sect_num).lat(1,num)=Node.N(nodenum,2);
        end

        value=next_value;

    end

while(M_len==0 & (feof(f1)==0) )

    len=length(value);

    if( length( findstr('A-',value)) >0 )
        arts=value(3:len);

    elseif( length( findstr('C-',value)) >0 )
        approach=value(3:len);

    elseif( length( findstr('N-',value)) >0 )
        sector_name=value(3:len);

    end

    value=fscanf(f1,'%s',1);
    M_len=length(findstr('M-',value));

end

end
end

```

```

out.S=S;
out.h=h;

fclose(f1);

```

---

### ***Tocheck\_vertex***

% THIS FUNCTION DETERMINES IF ANY VERTEX NOT DEFINED FOR A  
%SECTOR LIES ON ANY OF ITS FACE.

```

function answer=tocheck_vertex(S,Se,N,Adj,Adjsecnode)

smallnum=0.001;

out.Se=Se;
out.sub_sect=S

% for all sectors
for(i=1:length(Se))

    j=1; over=0; insertion=0;
    % Adj_twice=Adj2Adj(i,Adj);
    Adj_twice=Adj(i).sect;
    reqd_nodes=get_reqd_node(Adj_twice,N,S);

    % for all faces of ith sector
    while(over==0)

        found=0;
        alpha=Se(i).line(1,j).alpha;
        c=Se(i).line(1,j).c;
        P1=N( S(i).ver(j),:);
        P1(1,1)=-P1(1,1);

        if(j==Se(i).n)
            P2=N( S(i).ver(1),:);
        else
            P2=N( S(i).ver(j+1),:);
        end
        P2(1,1)=-P2(1,1);

        % get all scetors around ith sector which are adjacen to it and also adjacen to the adjacent sectors
        for(k=1:length(reqd_nodes) )

            cur_vertex=reqd_nodes(k);
            P=N( cur_vertex,:);
            P(1,1)=-P(1,1);

            % if Pth point is different from P1 and P2
            if(abs(dis([P1;P]))>smallnum& abs(dis([P2;P]))>smallnum )

```

```

% if Pth point lie on the face
if ( (abs(alpha*P'-c) < smallnum) )

    % if Pth point lie in between P1 and P2
    if(abs(dis([P1;P])+dis([P2;P])-dis([P1;P2])<smallnum )

        insertion=1;
        sprintf('In sector %d for face %d, Vertex %d is
found',i,j,cur_vertex)

        if(i==334)
            sprintf('stop');
        end

        out=insert_vertex(S,Se,N,i,j,cur_vertex);
        S=out.sub_sect;
        Se=out.Se;

% decrement j so that the modified jth face will be checked again for vertices lying on it.
        j=j-1;
        found=1;

    end

    end

    end
    if(found==1)
        break;
    end

end

j=j+1;
if(j>Se(i).n)
    over=1;

    if(insertion==1)

        Adjsecnode=prepronode_onesect(i,S,Adjsecnode);

    end

end

end

end

answer.S=S;
answer.Se=Se;
answer.Adjsecnode=Adjsecnode;

```

---

### ***View\_main\_S***

```
function view_main_S(main_S,S,Se,sect_num)

% THIS FUNCTION PLOTS THE MAIN SECTOR sect_num.

hold

for(i=1:length( main_S(sect_num).subs) )

    secti=main_S(sect_num).subs(1,i);

    Z=ones(1,Se(secti).n+1);
    Zmin=Se(secti).hmin*Z;
    Zmax=Se(secti).hmax*Z;
    clear Z;

    plot3(S(secti).long,S(secti).lat,Zmin/100,'g')

    plot3(S(secti).long,S(secti).lat,Zmax/100,'g')

    clear Zmin Zmax;

    for(j=1:Se(secti).n)
        X(1,1:2)=S(secti).long(1,j);
        Y(1,1:2)=S(secti).lat(1,j);
        Z(1,1)=Se(secti).hmin/100;
        Z(1,2)=Se(secti).hmax/100;

        plot3(X,Y,Z,'m')
    end

end

end
usmap
axis equal
view(3)
```

---

### ***View\_sect\_Fp\_ht***

```
function view_sect_Fp_ht(S,Se,N,Fp,h)

% THIS FUNCTION PLOTS SECTOR MODULES WHICH ARE LYING AT HEIGHT
% h AND ALL FLIGHT TRAJECTORIES.

FP=Fp;
hold
for(i=1:length(S))

    if(h>=Se(i).hmin & h<=Se(i).hmax)

        plot( S(i).long, S(i).lat,'g')
```

```

                xloc=sum(S(i).long(1:Se(i).n))/(Se(i).n);
                yloc=sum(S(i).lat(1:Se(i).n))/(Se(i).n);
                snum=int2str(i);
                text(xloc,yloc,snum)
            end
        end

    end

    for(i=1:length(N))
        nnum=int2str(i);
        % text(-N(i,1),N(i,2),nnum)
    end

    usmap

    axis equal

    for(i=1:length(FP))
        n=FP(i).n;
        if(n>0)

            X=zeros((n),1);
            Y=zeros((n),1);
            Z=zeros((n),1);
            % X(:)=FP(i).wp(:,1);
            Y(:)=FP(i).wp(:,2);
            plot(X,Y,'b')
            name=['FP' int2str(i)];
            text(FP(i).wp(1,1),FP(i).wp(1,2),name)
            for(j=1:n)
                text(FP(i).wp(j,1),FP(i).wp(j,2),'*')
            end
        end
    end
end

```

---

### ***View\_sect\_ht***

```

function view_sect_ht(S,Se,N,h)

% THIS FUNCTION PLOTS SECTOR MODULES WHICH ARE LYING AT HEIGHT
% h.

hold
for(i=1:length(S))

    if(h>=Se(i).hmin & h<=Se(i).hmax)

        plot( S(i).long, S(i).lat,'g')

        xloc=sum(S(i).long(1:Se(i).n))/(Se(i).n);
        yloc=sum(S(i).lat(1:Se(i).n))/(Se(i).n);
        snum=int2str(i);
        % text(xloc,yloc,snum)
    end
end

```

```
        end
    end

    for(i=1:length(N))
        nnum=int2str(i);
        % text(-N(i,1),N(i,2),nnum)
    end

    usmap

    axis equal
```

---

## **VITA**

The author, Shrinivas M. Sale was born in Mangalore, India, on November 3, 1973. He graduated from the Indian Institute of Technology at Madras in India, with a BTech in Civil Engineering, in 1996. He then came to Virginia Polytechnic Institute and State University in August 1996 for his Masters in Civil Engineering with specialization in Transportation Engineering. Upon acquiring his M.S. degree, he will pursue his career as a Logistics Engineer at Schneider Logistics, in Green Bay, Wisconsin.