(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PROCEDURAL GUIDE FOR MODELLING AND ANALYZING THE FLIGHT CHARACTERISTICS OF A HELICOPTER DESIGN USING FLIGHTLAB

by

Gary P. McVaney

September, 1993

Thesis Advisor: E. Roberts Wood

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 23 September 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Procedural Guide for Modelling and Analyzing the Flight Characteristics of a Helicopter Design using Flightlab

5. FUNDING NUMBERS

6. AUTHOR(S) Gary P. McVaney

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

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

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13.
ABSTRACT *(maximum 200 words)*

This thesis presents one method for modelling and analyzing a helicopter design using Flightlab. Flightlab is a computer program that provides for engineering design, analysis and simulation of aircraft using non-linear dynamic modeling techniques. The procedure to model a single main rotor helicopter is outlined using the sample helicopter design in the book "Helicopter Performance, Stability, and Control" by Ray Prouty. The analysis procedure contains computer program scripts for determining the time response of the helicopter to standard control inputs such as a longitudinal impulse, a lateral step, and a pedal doublet. A linear model of the helicopter can be extracted from the non-linear model, and a comparison of the time response to the control inputs based on these two models is presented. The procedure for conducting frequency sweep testing for the linear model is also discussed. This guide to using Flightlab for aircraft modelling and analysis is designed to make it easier to use Flightlab for creating additional aircraft models for use in control system analysis and additional engineering design.

14. SUBJECT TERMS Helicopter Design, Analysis, Flight Characteristics, Flight Simulation, Computer Modelling			15. NUMBER OF PAGES 150 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

Procedural Guide for Modelling and Analyzing the Flight Characteristics of a Helicopter Design using Flightlab

by

Gary P. McVaney
Major, United States Army
B.S., United States Military Academy, 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: Gary P McVaney
Gary P. McVaney

Approved by: E. R. Wood
E. R. Wood, Thesis Advisor

I. I. Kaminer
I. I. Kaminer, Second Reader

D. J. Collins
D. J. Collins, Chairman
Department of Aeronautics and Astronautics

ABSTRACT

This thesis presents one method for modelling and analyzing a helicopter design using Flightlab. Flightlab is a computer program that provides for engineering design, analysis and simulation of aircraft using non-linear dynamic modeling techniques. The procedure to model a single main rotor helicopter is outlined using the sample helicopter design in the book "Helicopter Performance, Stability, and Control" by Ray Prouty. The analysis procedure contains computer program scripts for determining the time response of the helicopter to standard control inputs such as a longitudinal impulse, a lateral step, and a pedal doublet. A linear model of the helicopter can be extracted from the non-linear model, and a comparison of the time response to the control inputs based on these two models is presented. The procedure for conducting frequency sweep testing for the linear model is also discussed. This guide to using Flightlab for aircraft modelling and analysis is designed to make it easier to use Flightlab for creating additional aircraft models for use in control system analysis and additional engineering design.

ACKNOWLEDGEMENTS

I would like to thank several people who have provided me with assistance in completing this work. Herb Gillman, III, from Advance Rotorcraft Technologies, provided me with a lot of help with explanations about procedures and methods used by the Flightlab program. Antonio Cricelli provided me with a large amount of help to set up the computer system to run Flightlab and also aided me in my efforts to learn the Unix™ computer operating system. I would also like to thank two of the professors here at the Naval Postgraduate School, Dr. E.R. Wood and Dr. I.I. Kaminer for their review and critique of this work.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The most recent trend in the procurement of helicopters by the United States government is to have the helicopter design developed and flown using some type of flight simulation program prior to development of the first flyable prototype. The selection of the Boeing/Sikorsky team to build the new Light Helicopter (LH) for the U.S. Army, the Comanche, was the result of a competition based solely on the flight simulation of an engineering design. No prototypes have yet been built.

The development of a helicopter flight simulation of this type is usually done with a very complex set of computer programs, a very powerful and expensive computer system, and a full motion based simulator. This type of model also takes a great deal of manpower and expertise, is very expensive to operate, and it takes a very long period of time to develop. This type of flight simulation is not feasible for use within the learning environment and resources of a university.

Most university aircraft design courses are conducted without any type of flight simulation model to verify the acceptability of the aircraft design being developed by the students. Most often the students are required to make complex calculations by hand or with the help of many

different computer programs to resolve the aircraft's stability derivatives. Programs such as MATLAB™ are then used to evaluate the matrix representation of the aircraft design to determine the eigenvalues and mode shapes of the dynamic system response. This type of program is limited to analyzing the aircraft as only a six degree of freedom model and it can only produce numerical data output which can then be plotted graphically.

The advent of low-cost, powerful engineering workstations combined with multi-processing computer systems has led to the development of a new computer program called Flightlab. This program allows you to develop a helicopter design that includes the rotor system degrees of freedom in addition to the six degrees of freedom of the body. This allows for a more accurate representation of the aircraft and subsequent improvement in the analysis of the flight characteristics of the aircraft. This model can then be used to create a computer flight simulation of the aircraft design. The flight simulation model provides real time operation with pilot in the loop capability to analyze the model.

B. OVERVIEW OF FLIGHTLAB

The Flightlab program is written in a higher level language which uses a combination of the C and FORTRAN computer languages. Although many of the basic features of Flightlab are similar to MATLAB™, Flightlab offers the

advantage of being able to do a non-linear representation of the aircraft system. The program is written for use on computers utilizing a Unix™ operating system. The basic computer requirements to run Flightlab are a Silicon Graphics International (SGI) workstation with 36 megabytes of random access memory (RAM) and 100 megabytes of mass storage space for the program. Additionally the program requires that you use an Tektronix 4014 terminal emulation window, such as xterm, to display any of the plots generated by the simulation.

In order to use Flightlab the user must be familiar with the basic features of Unix and the use of Unix editors. There are two programs that make up the Flightlab computer flight simulation program package. These program differ in the method by which users interface with the program to develop the aircraft models and simulations. The first is called Scope and the second is called Gscope. The only difference between the two is the manner in which the user interacts with them. In Scope the user interfaces with the program directly at the programs command prompt. This requires the user to type commands and inputs to develop models and execute commands directly within the program. This method requires a very high degree of knowledge of command syntax and format. In Gscope the interface with the program is through the use of graphically displayed windows. The Gscope program window

based method is more familiar to most computer users and is generally less difficult to use. Gscope offers the additional advantage in that an aircraft model can be built by linking various graphically displayed model components in a building block approach. Data required for each component can be entered in the associated object fields and then the Gscope program can be used to generate the executable computer code that defines the model in the correct syntax and format.

Once the aircraft model is developed, a program must be written to run the flight simulation and develop the desired output. These programs are called script files and are written in the Scope language. Through the use of these scripts and the built in functions, the nonlinear model can be solved in real time to determine the forces and moments of the aircraft and its components. This solution is used to provide the aircraft dynamic response in both the time and frequency domains, and you can extract a linear model of the aircraft for comparison to other linear system models.

C. MOTIVATION

Although the Flightlab program allows for a dynamic real time simulation of an aircraft model with the modern engineering workstations and computers now found at most universities, it still requires a great deal of man hours and expertise to develop the model and associated programs. The developer of Flightlab provides a user's manual that consists

of four sub-manuals: Scope Users Tutorial [Ref. 1]; Component Reference Manual [Ref. 2]; and a Scope Theory Manual [Ref. 3]; and a Scope Command Reference Manual [Ref. 4]. These manuals are all based on using direct interface with Scope and do not discuss how to use Gscope. The Theory Manual discusses the aerodynamic or control theory used to develop each of the components in FLIGHTLAB and is useful for determining what sources are available for deriving data needed for each of these components. The Component Reference Manual describes the overall modelling and solution procedure and each component in detail, including what data is required by the program for each component. The Tutorial deals with much of the basics of modeling various items, but it does not show you how to model a complete aircraft.

This thesis is an attempt to provide a procedural guide for using Gscope to model a helicopter and analyze its flight characteristics. This guide can be used as a reference manual to reduce the amount of time required to learn the FLIGHTLAB program and should make it possible for FLIGHTLAB to be used during a one quarter design course to set up a model of a helicopter or other aircraft and analyze its flight characteristics. This guide can also improve the users understanding of the program and therefore allow better utilization of its capabilities for developing a model of more complicated thesis research topics, such as higher

harmonic control for helicopters or Unmanned Air Vehicle (UAV) control system analysis .

The following chapters outline a standardized method of modeling and analysis of a helicopter using the example helicopter design developed by Ray Prouty. The characteristics of this helicopter are presented his book, "Helicopter Stability and Control" [Ref. 5:pp.669-682].

II MODEL DEVELOPMENT PROCEDURE

A. GETTING STARTED WITH GSCOPE

Gscope is an X windows program that is run in the Unix™ operating system environment. This program is installed on the Naval Postgraduate School's parallel multi-processor computer in the Computer Center Visualization Laboratory (alioth.cc) and the Department of Aeronautics SGI Indigo computers. The program can be run remotely from any Unix™ workstation on campus capable of running X. Table 1 lists a

Table I UNIX ACCOUNT SETUP

Running on alioth.cc:
path must include /usr/local/flightlab and
/usr/local/flightlab/bin
setenv FL_DIR /usr/local/flightlab
setenv GVSDIR /usr/local/flightlab/gvs
setenv PS_PRINTER 'tiVis'
add xhost+alioth.cc to local machine for remote access

Running on Department of Aeronautics Machines:
path must include /usr/local/flightlab and
/usr/local/flightlab/bin
setenv XAPPLRESDIR $HOME/app-defaults
setenv FL_DIR /usr/local/flightlab
setenv PS_PRINTER 'hp2p_ps'
mkdir app-defaults
Copy /usr/local/flightlab/Scope.ad to
app-defaults/Scope

few necessary commands that must be used to specify path and environment variables for FLIGHTLAB.

The Gscope program has five main windows used to develop models and programs, the main, model, editor, plot and help windows. The program is started by typing the command "gscope" with or without an ampersand (&) sign behind it in an xterm window. The ampersand allows the program to run in the background keeping the xterm window active for other use. This is useful for editing script files while viewing the model window at the same time.

The program will start by opening a window that provides information about the program and then the main window will appear as shown in Figure 1.

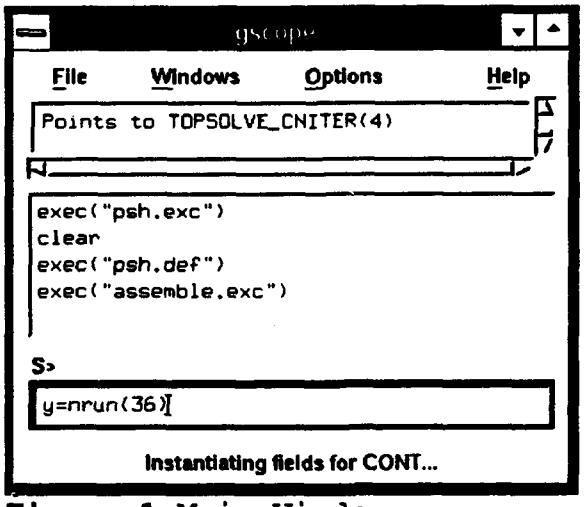

Figure 1 Main Window

The upper area of this window below the file menu is where the commands and output of the Gscope program are shown. The box at the bottom part of this window beneath the S> is where

Scope commands are entered. The area just above this box displays the most recently used commands. Selecting the windows menu button allows the user to open the other windows that are available for use.

The model window shown in Figure 2, is used to design the model.

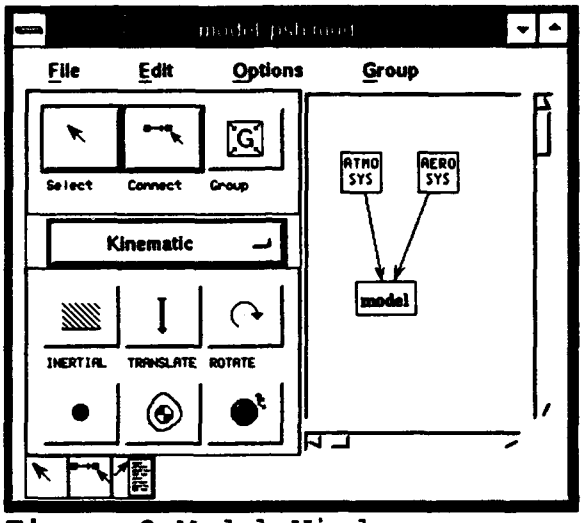

Figure 2 Model Window

This windows file menu bar allows you to open, edit, and save files. The model is built by selecting and placing the desired components in the area on the right side of this window. The components are then connected and the required data fields are entered for each component. The different components that can be utilized are shown on the left hand side of the window. The components shown in Figure 2 are the kinematic components, however, other components can be selected by clicking the mouse button over the box labeled Kinematic. Other groups include aerodynamic, control, non-

linear control, and transducer components, and are shown in Figures 22-26 in Appendix A.

B. MODELLING THE HELICOPTER

1. Model Hierarchy

A Flightlab model is usually built from the bottom up, i.e, you build subsystems, test them, and then combine them to form a complete system [Ref. 1:p. I-2]. This method allows you to rapidly reconfigure your model by replacing a subsystem within the model with a different variation of that subsystem, e.g. a main rotor system with four blades instead of three blades. All the components in a subsystem collectively form one group within the model hierarchy. This group is given the name of the sub-system. Many sub-system groups can exist in a higher order group within the model. All of the different subsystem groups connected together then form the group named "model".

The remainder of this section describes the model hierarchy used for Prouty's sample helicopter (PSH) and can be used with minor modifications for most single main rotor helicopters. The hierarchy of the model will be presented from the top level down to the bottom level. This is the way models are presented when first opened in Gscope. For ease of understanding, all component names will be *italicized*, all group names will be **bold**, and all program script file names will be <u>underlined</u>.

2. The World Group

The top level in the hierarchy of all models is called the world level, and this group for the model of psh is shown in Figure 3.

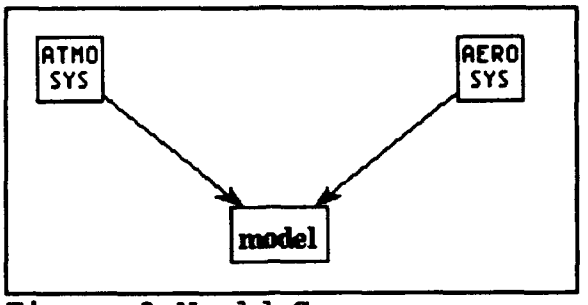

Figure 3 World Group

As shown in Figure 3, the world level consists of the **model** group, and the *atmo*system and *aerosystem* components. The **model** group contains all of the other groups (subsystems) that make up this helicopter. The *atmosystem* and *aerosys* components are system components used to collect information from all groups within the model and put it all together in one place [Ref. 2:p. 6]. The arrows indicate that the system components are connected somewhere in the model group, which indicates that data is shared between these groups.

Each component within a group has a set of mandatory data fields which must be assigned a value when building the model. These data fields are displayed by double clicking the left mouse button on a component symbol in what is called the object field window. You may insert the value of the data

fields in the space provided or you can use pointer variable names to direct Flightlab to look in a data file for the value of that variable name. The naming convention for all files created for the psh model is to use psh with an appropriate extension. Data files are given the .prolog extension, so the data file for psh is called psh.prolog. The value of all the variables in this file is loaded into the scope program prior to execution of the model script file. Using one data file for all components in the model allows rapid changes to be made, such as changing the location of the center of gravity (c.g.), without having to generate a new model script.

All the data fields required for each component used for psh, along with the variable name used to refer to the data in the psh.prolog file are listed in the psh.exc file provided in Appendix A. This appendix also includes a set of five figures that show the symbol for each type of component and also the component names used in the following sections. Additionally this appendix contains all the program scripts that include tables of data values used by aerodynamic components. Explanatory remarks about entries in a script file are entered by using either a comment marker, //, or by using the describe feature of Flightlab. The describe feature allows data fields in a program script to have a description string added after the input, e.g., stc=zeros(1,6); "stc is a 1x6 matrix of zeros". The description string will be displayed for all variables in a group whenever the describe

command is used during execution of scope. This feature is highly beneficial for keeping track of the variable names and the data to which they refer. Where appropriate, comments and descriptions of data fields in the script files for psh have been used to make it easier to understand what each line of program code means.

Selecting the connections box in the lower right corner of a components object field window will show the other components in the model which the selected component is connected. The *aerosys* component is connected to the rotor group and the *atmosys* component is connected to **model** group.

3. The Model Group

The **model** group is the first level in the model hierarchy below the world group. This is where all components and subsystem groups that define the helicopter are located. Double clicking the left mouse button will take you to this level and will display the subgroups within the **model** group as shown in Figure 4.

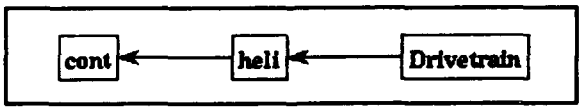

Figure 4 Model Group

4. The Heli Group

The **heli** group consists of three sub groups and an *atmosphere* component and *inertial* component which represent the helicopter model as shown in Figure 5.

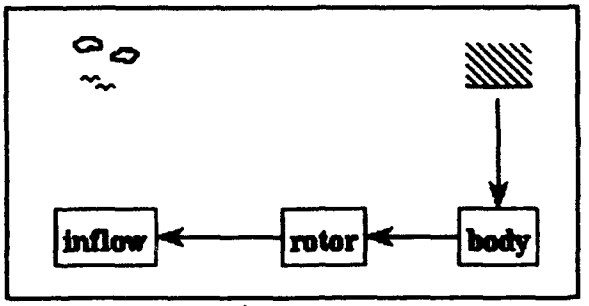

Figure 5 Heli Group

The *atmosphere* component models the atmospheric conditions and shares this data with other components within the group. This component uses the ARDC62 model of the standard atmosphere and is listed in atmo.tab. The *inertial* component represents the inertial reference coordinate axis system. All velocities and accelerations are measured in this frame and are transformed to each component's frame of reference using the appropriate transformation matrix. The inertial coordinate system is oriented positive x axis forward along the nose of the aircraft, positive y axis to the right side of the aircraft, and positive z axis down towards earth.

a. The Body Group

The **body** group is used to model the fuselage and tail section of the helicopter as shown in Figure 6.

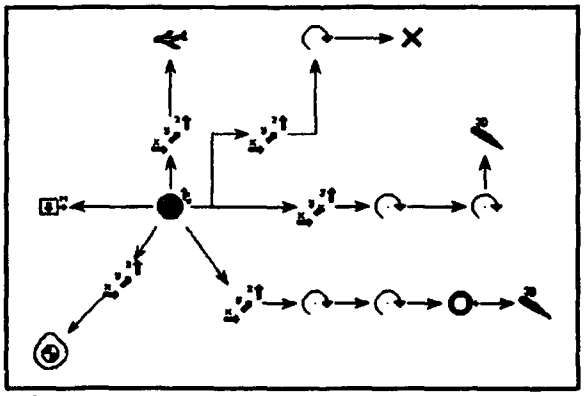
Figure 6 Body Group, PSH

The first component used is the *dof6* component which represents the six degrees of freedom of the fuselage. Connected to this is the *dmass* component which is used to model the distributed mass and the effective forces about the center of gravity of the helicopter. The data needed for this component includes the mass of the helicopter minus the mass of the main rotor system, the inertia matrix for the helicopter and the value of the gravity vector to be used with the model.

The *dmass* and *dof6* components are connected with a *translate3* component which is used to locate the *dmass* component at the center of gravity. The location is specified by a vector which consists of the fuselage station, buttline and waterline station locations of the center of gravity.

Another *translate3* component connects the *dof6* component with a *aero3ds* component which represents the three-dimensional aerodynamic characteristics of the fuselage. Data requirements for the fuselage include the lift, drag, and pitching moments as a function of angle of attack and sideslip. The values for psh were taken from the charts in Prouty's book [Ref. 5:pp. 679-682]. This component requires data from +90 to -90 degrees angle of attack and since the reference did not provide this data, a user-designed function (odli) was used to perform one-dimensional linear interpolation of the available data to meet this need. This function can be used whenever insufficient data is available,

however, you must verify the accuracy of this extrapolated data using acceptable aerodynamic theory.

Two tables of the aerodynamic characteristics of the fuselage are required for this component. One is the high resolution data for low angles of attack, in this case from -25 to 25 degrees using five degree increments. The low resolution data table provides the characteristics from -90 to 90 degrees in ten degreee increments. The aerodynamic characteristics must also include cross coupling effects due to sideslip from -180 to 180 degrees. The data fields and tables for the fuselage component for psh is loaded into the scope program by executing the c_wfaero.exc file. The method used by Prouty for determining the aerodynamic characteristics of his theoretical fuselage shape is presented in the USAF Stability and Control Datcom manual [Ref. 6].

Similar tables are needed for the *aero2d3d* components used to model the horizontal and vertical tail sections two-dimensional aerodynamic characteristics with three-dimensional flow. The tables for main rotor blade segments *aero2d* component, include the lift, drag and pitching moment characteristics based on angle of attack and mach number. The files c_htail1.tab and c_htail2.tab contain the high and low resolution lift and drag tables for the horizontal tail, and the vertical tail tables are in the files c_vtail1.tab and c_vtail2.tab. The commands to load the tables for the horizontal and vertical tail section are

horizontal and vertical tail section are contained within the appropriate section of the psh.prolog file.

Each of the *aero2d3d* components used to model the horizontal and vertical tail sections are connected to the *dof6* component with *translate3* components and *rotate* components. The *translate3* components provide the location of these sections in terms of their fuselage, buttline and waterline station. The orientation of the coordinate system for aerodynamic components including the horizontal and vertical tails and rotor blades is different than the inertial coordinate system. *Rotate* components are used to create the transformation from the inertial frame to the component frame of reference. Each *rotate* component has data fields to indicate the axis of rotation and the amount of rotation from the previous frame of reference to the next. Two *rotate* components are needed to change the orientation of the coordinate axis so that the x axis is to the right along the span of the tail section, y is forward along the chord and z is up. An extra *rotate* component is used for the horizontal tail to allow adjustment of its angle of incidence.

The tail rotor is modeled using the *Bailey* component. This component is a simplified model of a tail rotor based on the theory presented in NACA Report No. 716 [Ref. 7]. The orientation of this component's frame of reference requires one *rotate* component to rotate the y-z plane so that the z axis points in the direction of the tail

rotor thrust. One of the required data field parameters is for tail rotor blockage due to the vertical tail. These values represents the amount of tail rotor thrust loss due to the vertical tail at low speeds. Since there was no data available for psh concerning tail blockage, this parameter was set to be equal to one, which represents the assumption of no losses.

The final component in the body group is a *msensor* component. This component provides data about the motion of the fuselage rigid body to the control system including the inertial position, body axis velocities and accelerations, and body axis angular velocity and accelerations in the inertial frame of reference. The data fields for this component state the number of outputs desired and a gain matrix used to select which information is provided to the control system.

Each of the components of the **heli** group are connected to others as indicated by the arrows between them. There is a limit to the number of connections that some components can have and is listed in the appropriate section of the Component Reference Manual [Ref. 2]. Each connection between components is defined in terms of the parent node and child node for the connection. The correct connection between the parent and child frame is important for example in the case of a *sumj* component, which represents a summing junction. This component is limited to two connections which are summed

together based on an assigned gain value for each, which is either plus or minus one.

b. The Rotor Group

The **rotor** group represents the helicopter blade system, its rotor hub and its swashplate as shown in Figure 7.

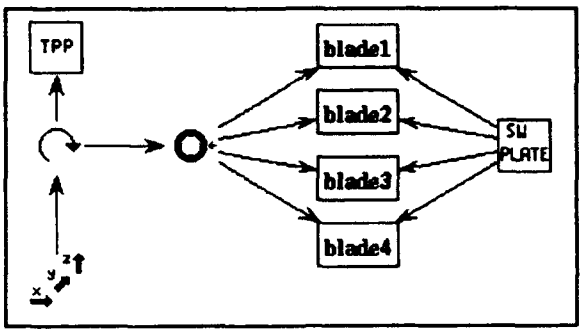

Figure 7 Rotor Group, PSH

The *translate3* component of the **rotor** group provides the same generalized location data in terms of stations for the rotor hub as was used for the aerodynamic components in the body group. The *rotate* component changes the orientation of the x-z plane such that x is pointed aft along blade one at the 0 degree azimuth position and z is pointing up. This rotation also includes the tilt of the main rotor shaft. Connected to this rotation component is the *tpp* component which is used to compute the tip path plane angles for the blades. Also connected to the *rotate* component is a *chinge* component (chinge) which is used to model a controlled hinge which provides the angular velocity input from the **drive train** group to the main rotor blades. This provides the

blades with the rotation force to keep them moving at the selected main rotor speed (rpm).

The control hinge is connected to four identical **blade** groups which are in turn connected to the *swashplate* component. The *swashplate* component is connected to each blade at the feathering hinge and is used to provide control input for the main rotor system. The data for the control inputs is provided to the swashplate by the control system without being connected to the swashplate through use of the data variable field input. A rigid blade element model was used for psh, and a representative *blade* group is shown in Figure 8.

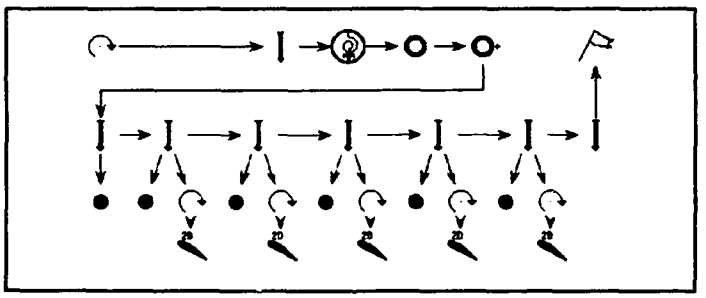

Figure 8 Rigid Blade Model,PSH

The first *rotate* component represents the transformation of the tip path plane coordinate system to have the x axis point along the span of the blades with the y axis forward along the chord and the z axis up. Obviously the rotation required for blade one pointing aft towards the tail of the aircraft is zero, and the rotation for the other blades

is 90 degrees from the previous blade since there are four blades in this rotor system.

The *translate* component connected to the *rotate* component is used to represent the distance from the center of the rotor hub to the blade hinges, i.e. the hinge offset. Connected to this is a *torspdm* component which represents a torsional spring damper that is used to model the lead-lag damping in this degree of freedom. The lead-lag damper was not originally in the design for psh, but it was added since the rotor system model is unstable without a lead-lag damper. Prouty does not include one because he assumes that for the liner model analysis used in his book there is no effect on the stability of his design [Ref. 5:p. 146], but there is an effect since we have included the blade degrees of freedom in this model. The next *hinge* component represents the blades flapping hinge and degree of freedom about the respective axes. The *chinge* component is where the feathering motion input for the blade is input via the swashplate.

The multiple sets of components following the control hinge represent sections of the blade itself. The first portion represents the blade spar and is modeled using a *translate* component with a *pmass* component which represents the mass of the blade at that point. Five identical blade segments follow, each with *translate*, *pmass*, *rotate*, and *aero2d* components. The end of the blade has a *translate* component which represents the distance from the center of the

last blade segment to the tip of the blade. A tip position marker, the *markpos* component, is used by the *tpp* component to compute flapping motion of the blades. The blade segments individual data fields are computed using a program file called the blade_segment.geom to divide the blade into a specified number of segments that sweep out equal areas during one revolution. Inputs required for this program include blade mass distribution, radius, the number of segments desired and the twist distribution of the blade. The model for psh uses five blade segments, with a constant mass distribution and the blade twist profile as shown in the mpl.exc and blade_twist.exc files listed in Appendix A.

The *aero2D* components represent the two-dimensional aerodynamic characteristics of the blade airfoil and require tables for the lift, drag and pitching moment coefficients as a function of angle of attack and mach number. No data was available for the 0012 airfoil for changing mach number, so the same values are used for all mach numbers. Like the horizontal and vertical tail section, the data for the main rotor blade segments is loaded by the appropriate section of the psh.prolog file. The data tables for the main rotor system are listed in the c_mrot1.tab and c_mrot2.tab files, and were determined using the characteristics for the NACA 0012 blade as listed in NACA technical note 3361 [Ref. 9].

c. The Inflow Group

The **inflow** group represents the induced velocity through the main rotor system and the interference effects between the rotor inflow and the body of the helicopter. The **inflow** *group consists of a uniformiv* component which models the inflow velocity based on momentum theory. This component includes the effect of close proximity to the ground. The ground effect parameters are determined analytically. The inflow time constant for the ground effect, tau, is determined by the equation tau=k_m/2*omega, where k_m=8/(3*pi) and omega is the main rotor speed [Ref 10: p. 100]. The ground effect parameter equation used for flightlab has the first ground effect parameter, gef1, always equal to 0.5 and gef2 = -0.6667 [Ref 11: p. 147]. The **inflow** group is shown in Figure 9.

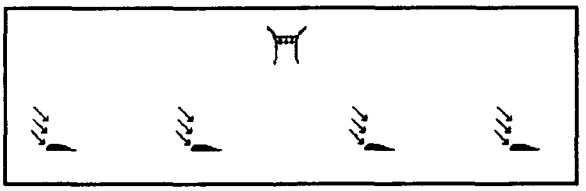

Figure 9 Inflow Group,PSH

The four *interfer* components in the **inflow** group represent the interference between the man rotor downwash and the fuselage, horizontal tail, vertical tail, and tail rotor. The data field requirements for these components are the dynamic pressure ratio, downwash and sidewash effects as a function of angle of attack and sideslip, and the incremental

velocities due to the rotor as a function of main rotor wake skew angle, Chi, and the longitudinal flapping angle of the tip path plane, A1f. The values used for these components assume no interference or dynamic pressure ratio loss and are presented in the files c_wfintf.exc, c_htintf.exc, c_vtintf.exc, and c_trintf.exc for the fuselage, horizontal and vertical tails, and tail rotor respectively.

5. The Cont Group

The **cont** group includes the pilot's flight control components, the mechanical flight controls, and any automatic flight control sub-systems for the helicopter. The **cont** group has five sub groups as shown in Figure 10.

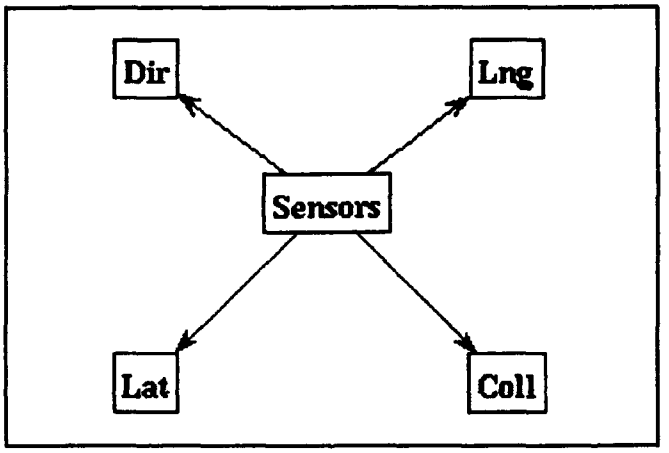

Figure 10 Control Group,PSH

a. Sensors Group

The **sensors** group is used to provide state variable input to the various control groups during execution of the model. The contents of the sensors group is shown in Figure 11.

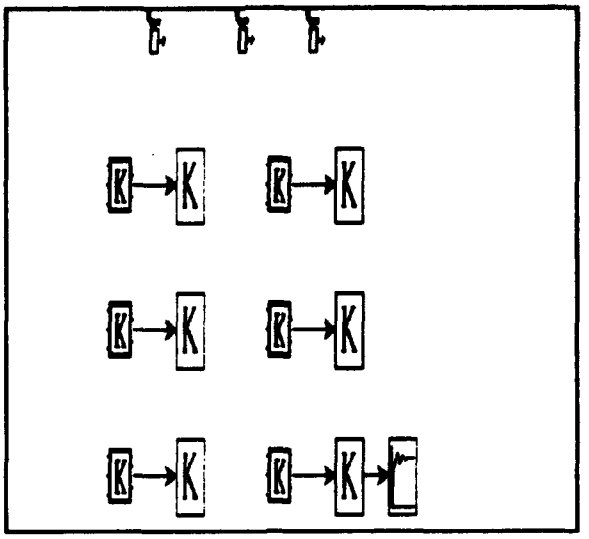

Figure 11 Sensor Group, PSH

The three *source* components at the top of this figure output the model's angle of attack, angle of sideslip, and the value of unity to the model control subgroups. The two euler angles are input to the solution group which is used to determine the values of other data fields that depend on these angles, e.g., the fuselage aerodynamics. The unity *source* provides a constant value of one as input to the *gain* components which represent the control axis bias in each control system subgroup, which is multiplied by a gain factor to select the reference position for that control axis.

The *ngain* components in the **sensor** group blocks below the *sources* are connected to the body degree of freedom motion sensor. Each *ngain* acts as a demultiplexer to select one of the degrees of freedom from the motion sensor as its output. The output of each *ngain* component is connected to a *gain* component which converts the units of the selected output from radians to degrees or radians/second to degrees/second as

appropriate. The state variables of the body degree of freedom which are "sensed" by these components are phi, theta, psi, p, q, and r.

The *solo* component connected to the gain component in the yaw axis rate sensor is a second order low pass filter. This filter is included to give the control group one degree of freedom which is required by the control solution method.

The outputs from these components are set to the desired input location in the configure.exc file, which is listed in Appendix B.

b. Lat, Dir, Long, Coll Control Groups

The **cont** group also includes the subgroups which model the control system for each of the four control axes. These subgroups are identical except for the values assigned to the component data fields. The lateral control system will be used to explain the way the control system groups are modeled and is shown in Figure 12.

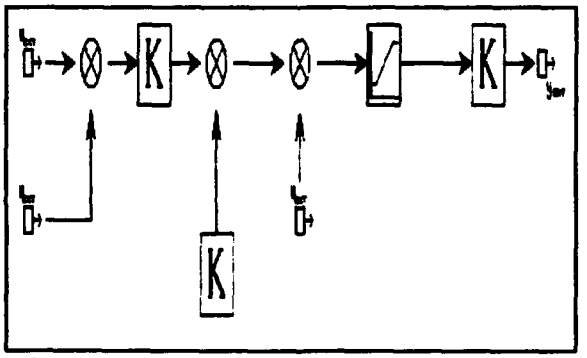

Figure 12 Lateral Control Group, PSH

The lateral control system model begins with two *source* components which are used to represent the input of the pilot and the trim system. The pilot input is set to zero except when performing analysis or flying the model, at which time it gets its input value from the pilot's workstation or the analysis program. The trim *source* component is set to the value determined by the trim control matrix, which is calculated during the trim routine for the model. The output from these are then summed then multiplied by a gain factor which converts the input units from inches of control movement to degrees of control system movement. This output is then summed again with the control axis bias. The bias for each control axis represents the reference control position. The reference positions are full left lateral cyclic, full aft longitudinal cyclic, full left pedal, and full down collective. The value of the bias and the gains for each control axis is set in the prolog file along with all other object data field variables. The sign of each *gain* component is based on the convention that positive control movement is forward longitudinal control, right lateral control, right pedal control and up collective control.

The next part of the lateral control system sums the previous output with a check position *source*. This component is used to check the control system model output during development of the control systems and may also be used by any augmentation control systems developed at a later time.

The outputs of the previously discussed components are then input to a *limiter* component, a *gain* component and finally to a *sink* component. The *limiter* is used to set the upper and lower limit for the control system output in terms of the swashplate's limits of travel. The *gain* component converts the units of the output from degrees to radians. The output of the *source* component is used by the swashplate and tail rotor components as the input for their respective control positions.

6. The Drivetrain Group

The **drivetrain** group is a generic group of components used to model the engine and drivetrain of the helicopter, as shown in Figure 13.

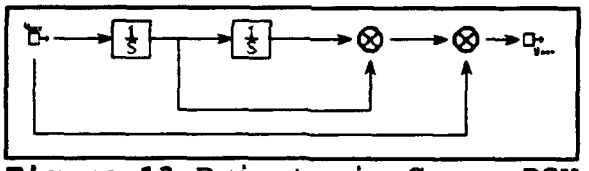

Figure 13 Drivetrain Group,PSH

The design of this group is very simplistic, but it functions well as an approximation for an actual system. There are no engine and transmission system components modeled yet in flightlab so all helicopter models use this drivetrain group to command the main rotor speed at the rotor hub. The *source* component models the accelerations of the drivetrain, but for this model these accelerations are constant and equal

to zero. The *inttf* components take the output of the previous components and integrate to determine the main rotor speed, omega, and azimuth position, psi. The two *sumj* components concatenate the acceleration, omega and psi values into a vector which is output by the final *sink* component to the main rotor control hinge. This component is connected to the main rotor hub *chinge* component as previously discussed.

C. MODEL SCRIPT

Upon completion of the model's development the contents of the model is saved to a file using the file menu's save command. All graphical model files are given the default extension mod, hence this file is called the psh.mod. The psh.mod file contains the graphical representation of the helicopter model as shown in the model window, however, this file cannot be executed by the Flightlab program. This file can be used to load the model back into the graphical user interface for modification or as the basis for creating a new model.

The file menu option "generate script" is used to create a file that is executable by Flightlab. This command generates the executable file from the model file in the correct syntax of the scope language. This is done automatically by the Gscope program and requires that the user input only the desired name of the file. By convention all program scripts generated in this manner are given the

filename extension of .exc, so this file is called psh.exc and is presented in Appendix A.

III. MODEL ANALYSIS PROCEDURE

A. ANALYSIS PROCEDURE OVERVIEW

Once the model program script has been generated using Gscope you must then create and execute a program file which provides scope with instructions for solving the various states of the model based on given input and selected procedures. As a minimum this program script must load the model and associated data, instantiate the system components, create solution component structure, initialize the states of the model, invoke a solution method for determining the time history of the model states, select the desired outputs from the simulation, and finally execute the model simulation. The structure of the scope program requires a certain number of standard things be done in order to analyze and create a flight simulation for a helicopter. The following sections outline the method used for psh which can be modified as necessary for any other single main rotor helicopter model. Much of the information presented in the following sections is a summary of the detailed explanations found in the Flightlab/Scope Component Reference Guide [Ref. 2:p. 12-44].

The first section outlines the assembly procedure, i.e., how to load the model and component data, configure model parameters for analysis and reporting, create a solution

structure, initialize the model states, and assemble everything together. This procedure must be completed before anything else can be done with the model.

The second section describes the trim procedure which explains how to determine the trim conditions in terms of airspeed, flight path, and control positions for the helicopter model based on selected initial conditions. This requires conducting a trim sweep over a range of airspeeds and must be performed whenever any significant change is made to the operating conditions for the model, e.g., changes in gross weight, altitude, main rotor speed.

The third section outlines the analysis procedure, which demonstrates how to determine the response of the model to four basic tests: a longitudinal impulse, a lateral step, a lateral impulse, and a pedal doublet. Additionally this section describes the procedure necessary to obtain a reduced order linear state-space system matrix representation of the model and compares the output of the above tests for the linear and nonlinear simulations. This step is crucial since the linear system matrix is needed for control system design and analysis studies. The final part of this section also outlines how to determine the frequency response characteristics of the linear model.

B. ASSEMBLY PROCEDURE

The assembly procedure for psh model is listed in the file psh.def found in Appendix B. A file with the .def extension is by convention a file that defines a sequence of instructions and other script files to execute. The psh.def file contains all the instructions needed to set up the psh model for running and must be accomplished prior to the execution of any other simulation script file.

This file initially sets the path to all directories for the files used to assemble the model. This step is important since scope does not use the previously defined path for the unix system. This procedure also allows the use of files previously written for other models so they do not need to be copied into the current model directory. The psh sub-directory under flightlab contains all the script files used for the psh and can be used for new aircraft modelling and analysis.

The next step in assembling the model is to load the user defined functions (UDF) used during the assembly procedure. UDF's are used to define a specific function and are constructed from built-in functions which are part of the scope language and other UDF's which have been previously created [Ref. 1:p. II-22]. UDF's are similar to matlab.m files and add to the versatility of the scope program. UDF's are usually given the .fun filename extension. All UDF's must be executed prior to calling that function in the script

files. The assembly procedure uses the cycle and odli functions. The cycle UDF defines the method used to cycle through the iterative solution process and the odli UDF performs one-dimensional linear interpolation as previously discussed in the modelling section.

The third part of the assembly procedure is constructing the model in the correct hierarchy for the scope language. This is accomplished by executing files that contain these instructions. The psh.prolog (data) and psh.exc (model) files load the model and data parameters into the world level. The psh.epilog file then equivalences the model variable names to standard names for the solution and system components variables and also initialize the control connections and motion sensor gain matrix. The system.exc file creates and connects the system component to the model, and finally the solution.exc file which sets up the solution components and connections.

The solution file creates the solution configuration for psh which includes six solution components. Each solution component integrates the states and propagates in time its associated model group. The *helisolve*, *rotorsolve*, and *rhsolve* components use the numeric method, hsolve, to compute the states of the heli, rotor, and the combined rotor and heli groups. The *drivesolve* and *contsolve* components use the analytic method, csolve, to compute the states of the drivetrain and cont groups respectively. The final solution

component, *topsolve*, uses the fully coupled numeric and analytic method to solve for the states of the entire model group. Each solution component method also sets the value of required data fields, an explanation of which can be found in the component reference manual [Ref. 2: p.31-32].

The final part of the assembly procedure is to initialize, configure and setup the model for running the simulation. This involves using several built-in scope functions to initialize and invoke the model. The init command links, equivalences and analyzes the data flow for all the components in the model. The world::setup command initialize the states and methods for the model. The world::reset command sets the initial conditions for the states and their derivatives and invokes the model.

The mbc.exc file is executed in order to set up a multi-blade coordinate transformation of the rotor states. This improves the speed and accuracy of the solution of the rotor states during execution of the model. The configure.exc file is used to configure the model structure for the desired reporting and sharing of information between solution components by creating the **cpg** (compute parameter group) and **results** groups at the world level. These groups provide a central location for the output of the simulation to be stored.

The final part of the assembly process is the execution of the assemble.exc file. This file sets several of the compute flags (cf) to assemble and reset the various solution groups. The psh model is now ready for execution.

C. TRIM PROCEDURE

The trim procedure script, trimsweep.def, is used to conduct a trim sweep for the model over a user specified range of airspeeds. The trim sweep for psh was done from 0 to 140 knots at sea level standard day conditions. Executing the trimsweep.def file and specifying the desired range of airspeeds and the flight path angles is all that is necessary to complete the trim procedure. Upon completion of the trimsweep program the user is given the option of displaying and printing the results of the trim sweep. All the files used for the trim sweep procedure are listed in Appendix C.

The first file executed by the trimsweep program is the psh.def file. This is done to assemble the model for running as explained in the previous section. Once the model is assembled, the UDF, limitchange.fun, is loaded into the scope program. The limitchange.fun is used by the trim program to limit the amount of control input changes to a max of 2.5 percent of control travel during the trim sweep. This reduces the time needed for the trim program to converge to a solution.

The Trim_Sweep.exc file is executed next. This file sets the number of rotor revolutions used to determine the average value of the body accelerations and then executes the main trim script file, Trim.exc.

The Trim.exc file is based on a simple algorithm designed to reduce the steady state translational and angular body accelerations, (bacc), to zero. The trim process calculates the initial bacc and then runs a trim routine that iteratively calculates the trim control positions changes needed to reduce the accelerations and then re-evaluates the body accelerations. The iteration cycle continues until the largest singular difference from the previous to the current body acceleration is less than the convergence limit, 0.0001, or the maximum number of iterations is reached. Three trim loop iterations are used, with a maximum of 60 iterations.

After setting the number of rotor revolutions to use for determining steady state conditions, a trim matrix is computed at each airspeed in the trim sweep. The trim matrix is a diagonal matrix which is the partial of one acceleration which is coupled to one control as shown below:

```
th ~ ud (pitch attitude couples with longitudinal accel)
ph ~ vd (roll  attitude couples with lateral      accel)
xc ~ wd (collective     couples with vertical     accel)
xa ~ pd (lateral cyclic couples with roll         accel)
xb ~ qd (long.   cyclic couples with pitch        accel)
xp ~ rd (pedal          couples with yaw          accel)
```

The negative inverse of this matrix is used to determine the control change per unit acceleration used during the

iteration process. The diagonal matrix for each airspeed is converted into vector form and concatenated into a single trim matrix, trmd. This matrix is saved to Trim_Matrix.rbe at the end of the trim sweep for use during the analysis procedure.

Two other scripts files are executed from within the Trim.exc file, Update_CW.exc and Update.exc. These files are used to determine the amount of control change to apply for each iteration and then to determine the new steady state body accelerations and update the control positions. Upon completion of the iteration for each airspeed the trim control positions are formed into a vector and concatenated to a matrix called stc. This matrix is saved to a file called Trim_Controls.rbe at the end of the trimsweep procedure.

A graph of the bacc is presented after each iteration within the trim loop along with a listing of the average bacc and current control positions. At the end of the trimsweep procedure, the user is given the chance to view a plot of control positions versus airspeed as shown in Figure 14.

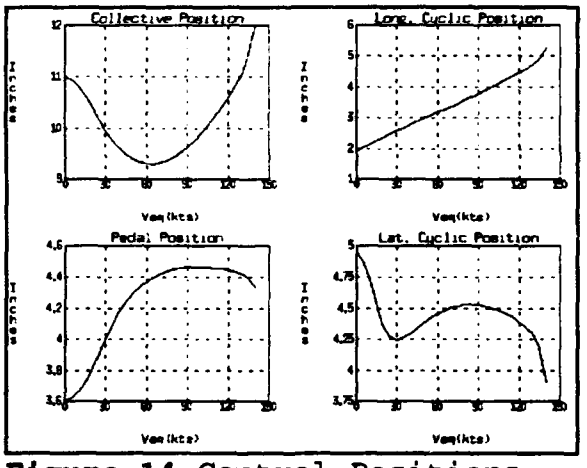

Figure 14 Control Positions

The values for the control positions obtained by this trim method for 115 knots are very close to the values obtained analytically by Prouty [Ref 5:pp. 527-529]. The plot also shows that the sample helicopter runs out of control power at approximately 135 knots, which would represent the maximum speed psh is capable of achieving with its control system design.

D. ANALYSIS PROCEDURE

The analysis.def file contains a sample method of analyzing the time response and frequency response of a helicopter at a given flight speed and condition. The analysis.def file sets up a method for analyzing the time response and frequency response of psh to control inputs. The time response procedure is set up to find an open loop nonlinear model solution based on the rhsolve solution group and compare that to the response of a reduced order linear model solution based on the topsolve solution group. The frequency response procedure analyzes the model based on the linear model solution only.

The analysis.def file executes the psh.def to set up the model for running and then it adds the test directory to the search path for the files used to conduct the various tests. The next part of the script executes three additional UDF's required for the analysis procedure. The qsreduce.fun is used in the 6dof_linearize.exc file which reduces the nonlinear

model with 37 states to a linear model with six degrees of freedom and eight state variables. The linear state variables include u, v, w, p. q. r. theta, and phi. The 6dof_linearize.exc file can be modified to return a linear model with more states and degrees of freedom, such as a 10 degree of freedom model which includes the blade motion degrees of freedom. The logspace.fun UDF and the freq.fun UDF are used to set up a logarithmically spaced input frequency vector and the frequency sweep test respectively.

The linearization procedure uses the convolution integral method to determine the linear characteristic and control matrices. The coefficients of the characteristic equation, and the eigenvalues and vectors for the characteristic matrix are listed at the end of Appendix D. Upon completion of the linearization program, a plot of the eigenvalues for the linear model is provided as shown in Figure 15.

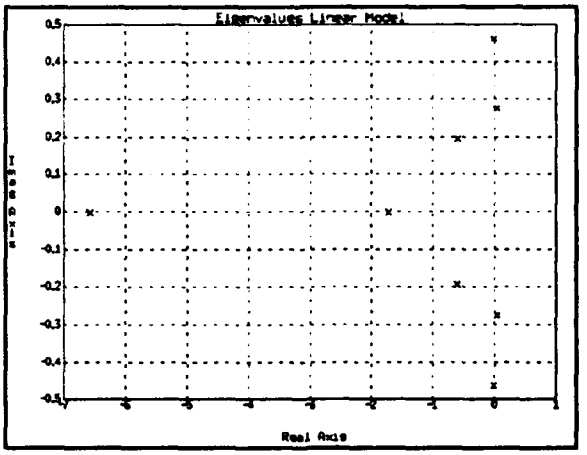

Figure 15 Linear Eigenvalues

As shown in the plot of eigenvalues, there is one pair of complex roots that are unstable roots. It will be shown later that this pair represents the longitudinal phugoid response mode. All of the other roots are stable, however, there is another complex pair that is just barely on the stable side of the real axis. The stable roots represent the longitudinal short period mode, the lateral-directional dutch roll mode, and the lateral spiral and roll modes. It is difficult to determine which root corresponds with which model just from the plot, so we can conduct several standard control input tests to determine this.

The first test conducted by the analysis.def file is a longitudinal impulse test. The Long_Impulse.exc file is used to set up a longitudinal cyclic control input with the user defined parameters for duration of the run, size of the input, and input delay time. Figure 16 shows a comparison of the nonlinear response and the linear response of psh in terms of pitch attitude and forward airspeed to a 1 inch forward cyclic impulse.

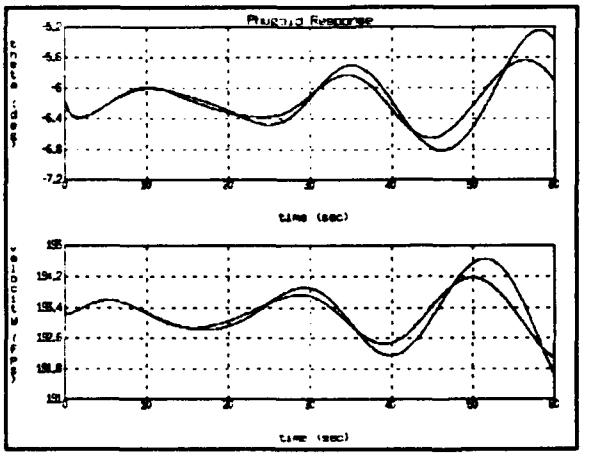

Figure 16 Phugoid Response

As shown in the figure above, the response of the nonlinear and linear models correspond well. The change in pitch attitude and velocity in response to the impulse input is oscillatory and divergent. The rate of divergence is slow, however, which corresponds to the location of the unstable roots on the eigenvalue plot. The results of this test show that psh is longitudinally unstable, which is the same conclusion reached by Ray Prouty using his analytic methods [Ref. 4: pp. 616-623].

The second test of the helicopters response is a lateral step input. The test script, Lat_Step.exc asks the user to input the duration of the test, the size of the input, the duration of the input, the time delay of the input, and the rise and fall time of the input. Figure 17 shows the response of psh in terms of roll attitude and roll rate to a 1 inch right cyclic step input lasting for 2.5 seconds with the input delay time, rise and fall time all set to 0.025 seconds.

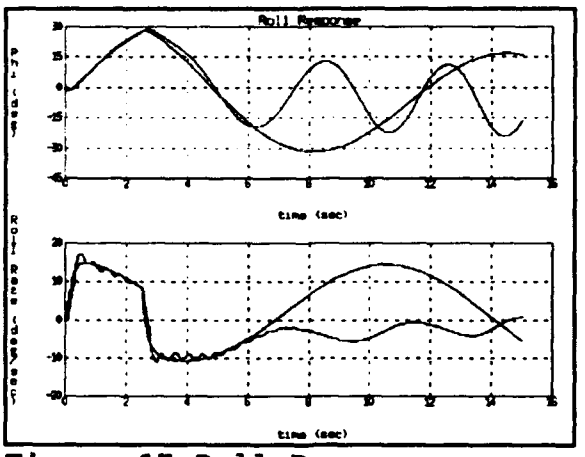

Figure 17 Roll Response

This plot shows a large difference between the nonlinear and linear response of the model after the first six seconds. The nonlinear model appears to have a much higher frequency at which it oscillates and may also be divergent. The linear response shows a much lower frequency of oscillation and it appears to slowly converge. This type of response is expected based on the location of the roots of the characteristic matrix as shown on the plot of eigenvalues, since we have already identified the phugoid mode as the unstable mode. The bank angle and roll rate response shown in the above plot for the nonlinear model includes the effects of control cross coupling.

The linear response does not correspond well with the nonlinear response because the method used to reduce the nonlinear model to a linear model is based on the assumption that the nonlinear model response remains in the linear range, which the data shows in not the case. This type of response would not be seen when applying this type of test to a purely linear model, because this type of model looks at the response of the system with no control cross coupling. The major cause of the roll response shown for psh appears to be a coupling of pitch to roll. The pure lateral cyclic input to the right causes a corresponding pitch up and when the cyclic is moved back to the left it causes a corresponding pitch down. The coupling effect of roll to pitch in terms of pitch attitude

and pitch rate due to the lateral step input is shown in Figure 18.

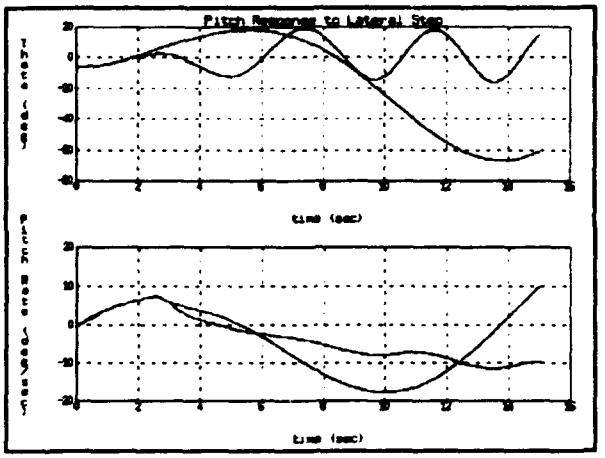

Figure 18 Pitch Coupling Due to Lateral Step Input

The third test of the helicopters response is a lateral impulse input. The test script, Lat_Impulse.exc, allows the user to input the duration of the test, the time delay of the input, and the size of the input. Figure 19 shows the response of psh in terms of roll attitude and roll rate to a 1 inch right cyclic impulse input with an input delay time of 0.025 seconds for a time interval of 30 seconds.

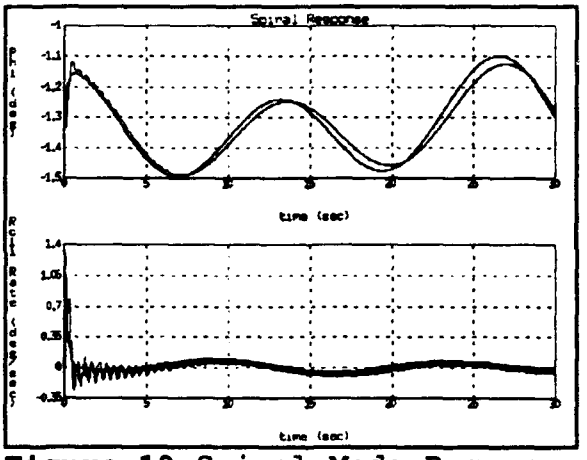

Figure 19 Spiral Mode Response

As shown in this plot, the response of the nonlinear and linear models correspond very well with each other. The spiral mode response of psh is considered to be stable because both the bank angle and roll rate are oscillatory and convergent.

The fourth test of the helicopters response is a pedal doublet input. The doublet input is created by using two step inputs in opposite directions. The test script, Ped_Doublet.exc, asks the user to input the duration of the test, the time delay of the input, the size of the step input, the duration time of the step inputs, and the rise, fall and delay times between each step. Figure 20 shows the response of psh for a 20 second interval in terms of roll attitude, yaw attitude, and roll rate to a 1 inch pedal doublet input. Each step input lasts 1.5 seconds and the delay, rise, and fall times are all equal to 0.025.

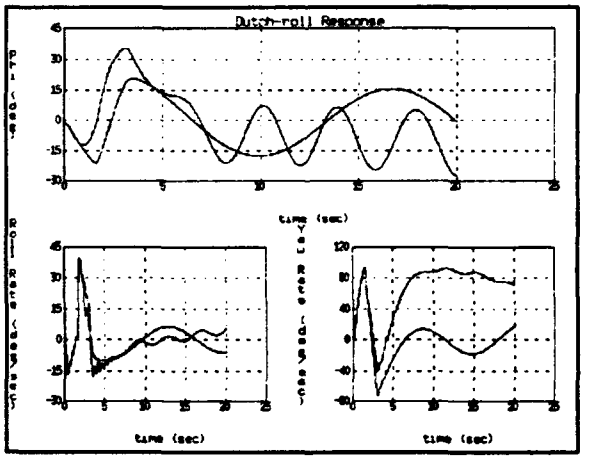

Figure 20 Dutch Roll Response

This plot also shows a large difference between the linear and nonlinear response of the model. The nonlinear model again appears to oscillate at a much higher frequency than the linear model and it also appears to be divergent. The linear response shows a much lower frequency of oscillation and it appears to be convergent. This type of response for the linear model is reasonable based on the location of the linear model eigenvalues. The response shown in the above plot for the nonlinear model also includes the effects of roll to pitch control cross coupling.

The final test conducted during the analysis procedure was to evaluate the frequency response to longitudinal sinusoidal inputs over a range of input frequencies. The frequency sweep test is conducted by using the freq.fun UDF. This function uses three inputs to determine the frequency response of the aircraft, the linear system matrix, the number of states in the linear model, and a vector of frequencies for the sinusoidal inputs. A vector of logarithmicly spaced frequencies from 0.1 to 100 radians/second was created using the logspace.fun UDF. This function was used to make creating a bode plot of the response possible, because one of the limitations of the scope program is its lack of a good log scale plotting feature. The system matrix was constructed by splitting the original linear matrix into its F, G, H, and D matrices, and then changing the input and output matrices to select only a longitudinal input and pitch attitude as the

output desired. The system matrix was then reconstructed using the original characteristic and control matrices and the modified input and output matrices.

The freq.fun UDF returns the amplitude and phase of the response for the given range of input frequencies. In order to create a bode plot the amplitude and the frequencies must be converted from natural logarithms to logarithms to the base 10. This is done by dividing the natural log of both the amplitude and frequency by the natural log of 10 and multiplying the amplitude by 20 to convert to gain in decibels. The frequency response of psh to a longitudinal frequency sweep is presented in Figure 21.

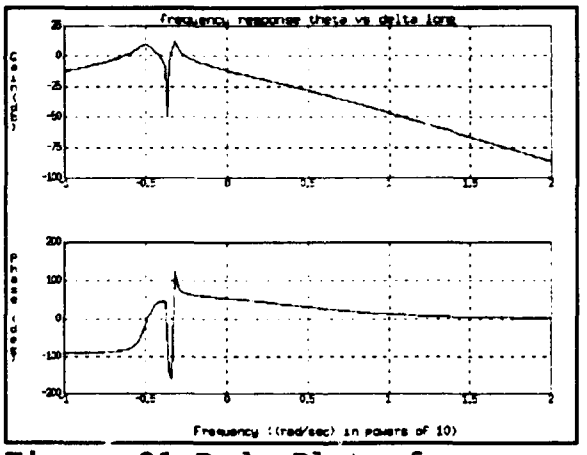

Figure 21 Bode Plot of Longitudinal Frequency Sweep

The data presented in the plot shows that the linear model response is typical of a second order system with a natural frequency of approximately 0.4 rad/sec and with a large amount

of damping as evidenced by the 40 decibels per decade decrease in gain at frequencies higher than the natural frequency.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis presents one method for modelling and analyzing a helicopter design using Flightlab. The example files and model can be modified and used as the basis for building models of different helicopters. The Flightlab program provides a very good tool for engineering design, analysis and simulation of helicopters using nonlinear dynamic modeling. The methodical procedure presented herein should supplement the user's manuals provided for Flightlab, and together they should make future modelling efforts for other helicopter designs and types a little easier. The analysis procedure shows that the time response of the helicopter to standard control inputs using the nonlinear modelling capabilities of Flightlab provides more information about the aircraft's flight characteristics in that it includes control cross coupling and is not limited to the assumption of linear modelling. The linear model of the helicopter which is extracted from the non-linear model can also be used to determine the frequency response to control inputs. This guide to using Flightlab for aircraft modelling and analysis provides the stepping stone for learning Flightlab and creating additional aircraft models for use in control system

analysis and additional engineering design at the Naval Postgraduate School.

B. RECOMMENDATIONS FOR FURTHER WORK

Although the procedures for modelling and analysis presented herein provide good results, there are several possible areas of improvements. This procedure uses several components based on basic theory, e.g., the rigid blade model of the rotor system uses blade element theory and the inflow model uses momentum theory. It is possible to develop a more advanced model of a rotor system using an elastic blade model and a better model of the inflow using the genwake theory. The capabilities are currently present within the Flightlab program.

The helicopter model used for this work did not include any flight control system augmentation so the procedure for developing a model of such a system was not discussed. A procedural guide for this type of model is also needed.

During the course of this effort, several problems were identified with the Flightlab program itself. Most of these were corrected by the developer of the program and implemented into this model. However, there was a recent problem discovered for which the solution was not included in this work. This problem concerns an omission of an angular velocity term in the calculation of the lateral acceleration rate for the *dof6* component. A corrected *dof6i* component was

developed but it uses several different object fields from the *dof6* component so some modification to the program files presented in this work would be necessary. A short comparison of the results between models with both types of components did not show much change in the aircraft response for the tests conducted in this work, but a large effect may occur for other tests and this correction should be implemented.

The final recommendation is to develop a procedural guide which discusses how to modify the rigid blade element model to enable real time engineering flight simulation of the helicopter for use with the pilot's workstation. This procedure is necessary because the current version of the Flightlab program version being used does not yet take advantage of the full capabilities of the parallel processing computer systems to run in real time. This procedure would involve creating a map of the rotor system states based on azimuth, collective position, inflow, and advance ratio. This "rotor map" along with replacing the rotor system in the model with a *rotor map* component would create a model capable of real time simulation of the helicopter with the pilot's workstation.

The current version of Flightlab includes the programs necessary to create a visual scene which uses a generic heads up display and a small portable box containing a three axis control stick and a collective and/or throttle. Once the Flightlab program is updated to run in parallel, this

procedure will no longer be necessary. Creating a flight simulation of the helicopter still requires the use of a multi-processor computer because the Flightlab process requires that three programs be run simultaneously, the visual scene program, the model program, and the program that is used to operate the pilot's workstation. This can only be done on a computer system that is capable of sharing memory and passing data between these programs in real time, otherwise, there would be an update problem between the visual system and the simulation input and output.

LIST OF REFERENCES

1. *Flightlab/Scope Users Tutorial Manual*, Advanced Rotorcraft Technologies, 22 July 1993.

2. *Flightlab/Scope Component Reference Manual*, Advanced Rotorcraft Technologies, 22 July 1993.

3. *Flightlab/Scope Theory Manual*, Advanced Rotorcraft Technologies, 22 July 1993.

4. *Flightlab/Scope Command Reference Manual*, Advanced Rotorcraft Technologies, 22 July 1993.

5. Prouty,R.W., *Helicopter Performance, Stability and Control*, Robert E. Krieger Publishing Company,Inc.,1990.

6. Hoak, *USAF Stability and Control Datcom*, USAF DATCOM, 1960.

7. Bailey, *A Simplified Theoretical Method of Determining the Characteristics of a Lifting Rotor in Forward Flight*, NACA Report No. 716, 1941.

8. Cheeseman and Bennet, *The Effect of the Ground on a Helicopter Rotor in Forward Flight*, British R&M 3021, 1957.

9. Critzos, Heyson, and Boswinkle, *Aerodynamic Characteristics of NACA 0012 Airfoil Section at Angles of Attack from 0° to 180°*, NACA TN 3361, 1955.

10. Peters,D.A., *Hingeless Rotor Frequency Response with Unsteady Inflow*, NASA SP-362, February 1974.

11. Johnson, W., *Helicopter Theory*, Princeton University Press, 1980.

APPENDIX A

```
////////////////////////////////////////////////////////////////
// file: psh.prolog
// date: 16 Jul 1992
//
// This script loads the data needed for psh model
////////////////////////////////////////////////////////////////

group data

//  Trim parameters

trimg = 1     "Percentage of trim change to apply on a control update";
nr    = 3     "Number of rotor revolutions between control updates";

//  Useful Constants

pi       = acos(-1)     "Ratio of diameter to circumferance";
d2r      = pi/180.0     "Degrees to radians conversion factor";
r2d      = 180.0/pi     "Radians to degrees conversion factor";
k2f      = 6076.115/3600"Knots to feet per second conversion factor";
f2k      = 3600/6076.115"Feet per second to knots conversion factor";
g        = 32.2;//      "Acceleration due to gravity (fpss)";
gravity  = [0 0 g]      "Inertial gravity vector";
dt       = 0.001  "Integration step size (sec)";
eps      = 5      "Solution convergence criteria on the Q's";
imax      = 20          "Maximun number of convergence iterations";

// Inflow data

gef1  =  0.0625         "Cheeseman Bennett ground effect parameter";
gef2  =  1.0       "Cheeseman Bennett ground effect parameter";
dwtau =  0.01959   "Inflow time constant (sec)";
agl   = 90         "Alttitude above ground plane (ft)";
chimr = 0          "Wake skew angle (rad)";
lam   = 0          "Inflow velocity (nd)";
nblades = 4        "Number of rotor blades";
nseg    = 5        "Number of blade segments";

// C G data

Vweight = 20000         "Total vehicle weight (lbs)";
ixx = 5000.0       "Total moment of inertia about x (sl-ft2)";
iyy = 40000.0           "Total moment of inertia about y (sl-ft2)";
izz = 35000.0           "Total moment of inertia about z (sl-ft2)";
ixy = 0.0          "Total cross product xy (sl-ft2)";
ixz = 0.0          "Total cross product xz (sl-ft2)";
iyz = 0.0          "Total cross product yz (sl-ft2)";

// Rotor data
lagdamper = 4000 "Lag Damping Coefficient (lbs sec /rad)";
imr = 0 "Longitudinal Shaft Tilt + Forward (rad)";
mrloc = [-0.5 0 -7.5]   "Main Rotor Location (ft)";
rpmnom  = 21.6667 "Nominal main rotor speed (r/s)";
rmr  = 30.0        "Main rotor radius (ft)";
naz      = 24      "Number of azimuth steps/rev";
dpsi     = 360/naz "Change in azimuth angle/integration step (deg)";

dt       = d2r*dpsi/rpmnom; // Integration step size
```

```
tippos  = zeros(3,nblades)"Inertial positions of blade tips (ft)";
tippa   = zeros(3,1)    "Tip path plane angles a0 a1f b1f (rad)";



//   Load main rotor blade segment aero tables
//

//  Rotor
rtable1= read("c_mrot1.tab");

clmr1(1:25,1:1) = rtable1(:,1)..
"Main rotor blade segment cl(alpha,mach) table (nd): high res/low
angle";
cdmr1(1:25,1:1) = rtable1(:,2) ..
"Main rotor blade segment cd(alpha,mach) table (nd): high res/low
angle";
cmmr1(1:25,1:1) = rtable1(:,3) ..
"Main rotor blade segment cm(alpha,mach) table (nd): high res/low
angle";

clear(rtable1);

rtable2= read("c_mrot2.tab");
clmr2   = rtable2(:,1) ..
"Main rotor blade segment cl(alpha) table (nd): low res/high angle";
cdmr2   = rtable2(:,2) ..
"Main rotor blade segment cd(alpha) table (nd): low res/high angle";
cmmr2   = rtable2(:,3) ..
"Main rotor blade segment cm(alpha) table (nd): low res/high angle";
clear(rtable2);

mrbp = 30.0*d2r ..
"Main rotor blade segment angle of attack transition angle (rad)";
mraoat1 = [d2r*[-30   30 2.5]  25] ..
"Main rotor blade segment angle of attack breakpoint table for high res
tables";
mraoat2 = [d2r*[-180 180 5] 73] ..
"Main rotor blade segment angle of attack breakpoint table for low res
tables";
mrmacht = [0 0 1 1] ..
"Main rotor blade segment Mach number breakpoint table for high res
tables";
mrna1   = mraoat1(4)    " of rows in main rotor blade segment high res
tables";
mrna2   = mraoat2(4)    " of rows in main rotor blade segment low res
tables";
mrnm1   = mrmacht(4)    " of cols in main rotor blade segment high res
tables";

exec("blade_seg_geom",1);// Compute the blade segment geometry

// C G reference Data (DMASS)

fscg = 296.0       "Fuselage Station of cg (in)";
blcg = 0.0         "Buttline Station of cg (in)";
wlcg = 113.0       "Waterline Station of cg (in)";

xcgf = 0  "Rigid body fuselage  station offset (ft)";
```

```
ycgf = 0  "Rigid body buttline  station offset (ft)";
zcgf = 0  "Rigid body waterline station offset (ft)";

cgloc = [xcgf ycgf zcgf] "Inertial cg offset (ft)";


fmass     = (Vweight - nblades*bw)/g"Mass of the fuselage (sl)";
finertia  = [ixx ixy ixz
             ixy iyy iyz
             ixz iyz izz] "Inertia matrix of the fuselage (sl-ft2)";

// Aero Wing/Fuselage data  (AERO3DS and INTERFER)

fswf = 302.0 "Fuselage  station of wing/fuselage (in)";
blwf =   0.0 "Buttline  station of wing/fuselage (in)";
wlwf = 119.0 "Waterline station of wing/fuselage (in)";

fswfft = (fscg - fswf)/12.0;
blwfft = (blcg - blwf)/12.0;
wlwfft = (wlcg - wlwf)/12.0;

wfloc=[fswfft blwfft wlwfft] "Aero fuselage location (ft)";

exec("c_wfaero.exc",1);      // Fuselage Aerodynamics

wabp = 25.0*d2r   "Wing/fuselage angle of attack transition angle
(rad)";
wbbp = 25.0*d2r   "Wing/fuselage angle of sideslip transition angle
(rad)";

alfwf  = 0    "Wing/fuselage angle of attack (rad)";
betwf  = 0    "Wing/fuselage sideslip angle (rad)";
alfiv  = 0    "Angle of attack for the interference effects (rad)";
betiv  = 0    "Sideslip angle for the interference effects (rad)";

wfiv = zeros(3,1) "Interference velocities on the wing/fuselage (fps)";
exec("wf_intf.exc",1); // W/F interference

// Horizontal Stabilizer (AERO2D3D and INTERFER)

fsht = 692.0 "Fuselage  station of horizontal stabilizer (in)";
blht =   0.0 "Buttline  station of horizontal stabilizer (in)";
wlht = 95.0  "Waterline station of horizontal stabilizer (in)";

fshtft = (fscg - fsht)/12.0;
blhtft = (blcg - blht)/12.0;
wlhtft = (wlcg - wlht)/12.0;

htloc = [fshtft blhtft wlhtft] "Horizontal tail location (ft)";

htincang = -0.052 "horizontal incidence angle + up (rad)";
htable1 = read("c_htail1.tab");

clh1(1:13,1:2)  =  htable1(:,1) ..
"Horizontal stabilizer cl(alpha,mach) table (nd): high res/low angle";
cdh1(1:13,1:2)  =  htable1(:,2) ..
"Horizontal stabilizer cd(alpha,mach) table (nd): high res/low angle";
cmh1(1:13,1:2)  =  0.0*ones(26,1) ..
"Horizontal stabilizer cm(alpha,mach) table (nd): high res/low angle";

clear(htable1);
```

```
htable2 = read("c_htail2.tab");

clh2    =  htable2(:,1) ..
"Horizontal stabilizer cl(alpha) table (nd): low res/high angle";
cdh2    =  htable2(:,2) ..
"Horizontal stabilizer cd(alpha) table (nd): low res/high angle";
cmh2    =  0.0*ones(19,1) ..
"Horizontal stabilizer cm(alpha) table (nd): low res/high angle";

clear(htable2);

hbp = 30.0*d2r "Horizontal stabilizer angle of attack transition angle
(rad)";

haoat1 = [d2r*[-30 30  5]  13] ..
"Horizontal stabilizer angle of attack breakpoint table for high res
tables";

haoat2 = [d2r*[-90 90 10]  19] ..
"Horizontal angle of attack breakpoint table for low res tables";

hmacht = [0 1 1 2] ..
"Horizontal stabilizer Mach number breakpoint table for high res
tables";

hna1    = haoat1(4);     // # of rows in clh1,cdh1,cmh1
hna2    = haoat2(4);     // # of rows in clh2,cdh2,cmh2
hnm1    = hmacht(4);     // # of cols in clh1,cdh1,cmh1

hchord = 2    "Chord length of horizontal stabilizer (ft)";
hlen   = 9    "Span of horizontal stabilizer (ft)";
hdefic = 1    "Lift deficiency factor (nd)";
htiv = zeros(3,1) "Interference velocities on the horizontal stabilizer
(fps)";
exec("ht_intf.exc",1); // H tail interference

// Vertical Tail (AER2D3D and INTERFER)


fsvt = 716.0;           // Fuselage Station of Vertical Tail
blvt = 0.0;       // Buttline Station of Vertical Tail
wlvt = 149.0;           // Waterline Station of Vertical Tail

fsvtft = (fscg - fsvt)/12.0;
blvtft = (blcg -blvt)/12.0;
wlvtft = (wlcg -wlvt)/12.0;

vtloc = [fsvtft blvtft wlvtft] "Vertical Tail location (ft)";
vtable1= read("c_vtail1.tab");

clv1(1:13,1:2)    =  vtable1(:,1) ..
"Vertical tail cl(alpha,mach) table (nd): high res/low angle";
cdv1(1:13,1:2)    =  vtable1(:,2) ..
"Vertical tail cd(alpha,mach) table (nd): high res/low angle";
cmv1(1:13,1:2)    =  0.0*ones(26,1) ..
"Vertical tail cm(alpha,mach) table (nd): high res/low angle";

clear(vtable1);

vtable2= read("c_vtail2.tab");
```

```
clv2   =  vtable2(:,1) ..
"Vertical tail cl(alpha) table (nd): low res/high angle";
cdv2   =  vtable2(:,2) ..
"Vertical tail cd(alpha) table (nd): low res/high angle";
cmv2   =  0.0*ones(19,1) ..
"Vertical tail cm(alpha) table (nd): low res/high angle";

clear(vtable2);

vbp = 30.0*d2r;          // Transition AoA fro clv1 to clv2
vaoat1 = [d2r*[-30 30  5]  13];// AoA break pts for clv1,cdv1,cmv1
vaoat2 = [d2r*[-90 90 10]  19]; // AoA break pts for clv2,cdv2,cmv2
vmacht = [0 1 1 2];      // Mach break pts for clv1,cdv1,cmv1
vna1   = vaoat1(4);      // # of rows in clv1,cdv1,cmv1
vna2   = vaoat2(4);      // # of rows in clv2,cdv2,cmv2
vnm1   = vmacht(4);      // # of cols in clv1,cdv1,cmv1

vchord = 33/7.7;         // Chord length of v tail
vlen   = 7.7;            // Width of the v tail
vdefic = 1;              // No lift deficiency

vtiv   = zeros(3,1);
exec("vt_intf.exc",1); // No V tail interference

// Bailer Tail Trotor (BAILEY)

fstr = 740.0;            // Fuselage Station of tail rotor
bltr = 24.0;       // Buttline Station of tail rotor
wltr = 185.0;            // Waterline Station of tail rotor

fstrft = (fscg - fstr)/12.0;
bltrft = (blcg -bltr)/12.0;
wltrft = (wlcg -wltr)/12.0;

trloc = [fstrft bltrft wltrft] "tail rotor location (ft)";

atr    = 5.73;           // Lift curve slope
ttr    = 1.00;           // Tail rotor thrust
cdtr   = 0.0;            // Rotor head drag coefficient
d0tr   = 0.0087;   // Taylor series drag coeff.
d1tr   =-0.0216;   // Taylor series drag coeff.
d2tr   = 0.4000;   // Taylor series drag coeff.
biastr = 14.0;           // Blade pitch bias
trblades = 3;            // Number of blades
btltr  = 0.92;           // Blade tip loss
bvttr  = 1.0;            // Blockage effect parameter
bvt1tr = 1.0;            // Blockage effect parameter
chordtr= 1;        // Blade chord
delttr = 0.001455;       // Partial of coning wrt thrust
omegatr= 100;            // Tail rotor speed
rtr    = 6.5;            // Tail rotor radius
thettr = 0.0;            // Tail rotor collective pitch
twsttr = -5;             // Tail rotor blade twist
td3tr  = -0.5774; // Tan of delta 3 angle
vbvttr = 30*k2f;  // Blockage velocity parameter
xibtr  = 0.67;           // Mass moment of inertia
xlamtr = 1.0;            // Tail rotor inflow (inital value)

triv   = zeros(3,1);
exec("tr_intf.exc",1);// No Tail rotor interference
```

```
// Atmosphere data  (ATMOS)

atmtab   = read("atmo.tab");// Get ARDC62 atmosphere tables

densityt =  atmtab(:,1);// Air density as function of altitude
ssoundt  =  atmtab(:,2);// Speed of sound as f(altitude)
natmo    = prod(size(densityt));// Size of data tables
altt     = [0 240000 2000 121]';// Alitude break points
clear(atmtab);

wind = zeros(3,1);      // No wind
bodytr = eye(3);
pos = [0 0 -90];
bodyvel = [10 0 0]';




//  Data for the UH60 control system
//
// Control system data

     mtheta   = 17.25*d2r;// main rotor pitch (rad)
     mthetad  = 0;       // main rotor pitch (rad)
     mthetadd = 0;       // main rotor pitch (rad)
     a1s      = -1.1*d2r;// lateral cyclic   (rad)
     a1sd     = 0;       // lateral cyclic   (rad)
     a1sdd    = 0;       // lateral cyclic   (rad)
     b1s      = -0.78*d2r;// long cyclic      (rad)
     b1sd     = 0;       // long cyclic      (rad)
     b1sdd    = 0;       // long cyclic      (rad)
        phase    =-4.0*d2r;// Swash plate phase angle


//  Pilot Controls, Trim Signals, Fps and Sas

Xa    = 0.0;
Xb    = 0.0;
Xc    = 0.0;
Xp    = 0.0;

Xatrm = 4.95;
Xbtrm = 1.196;
Xctrm = 10.817;
Xptrm = 1.925;

// Control bias inputs

A1sbias = -10.625;
B1sbias = -10.0;
Th0bias = 1.25;
Thrbias = 31.875;

// Swashplate test inputs

A1schk = 0.0;
B1schk = 0.0;
Th0chk = 0.0;
Thrchk = 0.0;
```

```
// Swashplate limits
A1sll = -11.0;
A1sul = 7.0;
B1sll = -10.0;
B1sul = 20.0;
Th0ll = 1.25;
Th0ul = 19.0;
Thrll = -15.0;
Thrul = 36.875;

//  Stick to swashplate scale factor

kxaa1s = 17.5/9      "Lateral pitch degrees per inch control movement";
kxbb1s = 30/10       "Longitudinal pitch degrees per inch control
movement";
kxcth0 =17.75/12     "Collective pitch degrees per inch control
movement";
kxpthr =-46.875/5.5 "Tail rotor pitch degrees per inch control
movement";


// Pilot input stops

xcll = (Th0ll-Th0bias)/kxcth0    "Collective stick lower stop (in)";
xcul = (Th0ul-Th0bias)/kxcth0    "Collective stick upper stop (in)";
xall = (A1sll-A1sbias)/kxaa1s    "Lateral cyclic stick lower stop (in)";
xaul = (A1sul-A1sbias)/kxaa1s    "Lateral cyclic stick upper stop (in)";
xbll = (B1sll-B1sbias)/kxbb1s    "Long.   cyclic stick lower stop (in)";
xbul = (B1sul-B1sbias)/kxbb1s    "Long.   cyclic stick upper stop (in)";
xpll = (Thrul-Thrbias)/kxpthr    "Pedal lower stop (in)";
xpul = (Thrll-Thrbias)/kxpthr    "Pedal upper stop (in)";

parentg

//
// End of psh.prolog data script
```

```
//////////////////////////////////////////////////////////////
// This is the file c_htail1.tab which contains the high resolution
// data tables lift and drag coefficients for psh horizontal tail
//////'///////////////////////////////////////////////////////
26 2
    -0.9700     0.4300
    -1.0300     0.3700
    -1.0300     0.3600
    -0.9300     0.1900
    -0.7100     0.0400
    -0.3560     0.0220
     0.0000     0.0100
     0.3560     0.0220
     0.7100     0.0400
     0.9300     0.1900
     1.0300     0.3600
     1.0300     0.3700
     0.9700     0.4300
    -0.9700     0.4300
    -1.0300     0.3700
    -1.0300     0.3600
    -0.9300     0.1900
    -0.7100     0.0400
    -0.3560     0.0220
     0.0000     0.0100
     0.3560     0.0220
     0.7100     0.0400
     0.9300     0.1900
     1.0300     0.3600
     1.0300     0.3700
     0.9700     0.4300

//////////////////////////////////////////////////////////////
// This is the file c_htail2.tab which contains the low resolution
// data tables for the horizontal tail for psh
//////////////////////////////////////////////////////////////

19 2
     0.0000     1.2000
    -0.2940     1.1610
    -0.5580     1.0500
    -0.7450     0.8880
    -0.8470     0.7020
    -0.8470     0.5310
    -0.9700     0.4300
    -1.0300     0.3600
    -0.7100     0.0400
     0.0000     0.0100
     0.7100     0.0400
     1.0300     0.3600
     0.9700     0.4300
     0.8470     0.5310
     0.8470     0.7020
     0.7450     0.8880
     0.5580     1.0500
     0.2940     1.1610
     0.0000     1.2000
```

```
//////////////////////////////////////////////////////////////
// This is the file c_vtail1.tab which contains the high resolution
// data tables for the vertical tail for psh
//////////////////////////////////////////////////////////////
   26 2
   -1.0000     0.3600
   -1.0000     0.2650
   -0.9300     0.1740
   -0.7300     0.1180
   -0.5000     0.0660
   -0.2800     0.0330
   -0.0600     0.0180
    0.1600     0.0210
    0.3800     0.0440
    0.6100     0.0920
    0.8200     0.1620
    0.8900     0.2480
    0.8900     0.3550
   -1.0000     0.3600
   -1.0000     0.2650
   -0.9300     0.1740
   -0.7300     0.1180
   -0.5000     0.0660
   -0.2800     0.0330
   -0.0600     0.0180
    0.1600     0.0210
    0.3800     0.0440
    0.6100     0.0920
    0.8200     0.1620
    0.8900     0.2480
    0.8900     0.3550

//////////////////////////////////////////////////////////////
// This is the file c_vtail2.tab which contains the low resolution
// data tables for the vertical tail for psh
//////////////////////////////////////////////////////////////

 19 2
    0.0000     1.1000
   -0.1200     1.0250
   -0.2800     0.9650
   -0.4600     0.8750
   -0.6600     0.7450
   -0.8800     0.5750
   -1.0000     0.3600
   -0.9300     0.1740
   -0.5000     0.0660
   -0.0600     0.0180
    0.3800     0.0440
    0.8200     0.1620
    0.8900     0.3550
    0.8000     0.5800
    0.6300     0.7500
    0.4800     0.8750
    0.3200     0.9650
    0.1700     1.0200
    0.0000     1.0800
```

```
//////////////////////////////////////////////////////////////
// This is the file c_mrot1.tab which contains the high resolution
// data tables for the main rotor for psh
//////////////////////////////////////////////////////////////
 25 3
   -0.9        0.6 0.6
   -0.75       0.5 0.56
   -0.7        0.41      0.48
   -0.675      0.35      0.4
   -0.674      0.25      0.34
   -0.8        0.155     0.28
   -0.9        0.1 0.12
   -1.225      0.05      0.0000
   -1.12       0.045     0.0000
   -0.85       0.03      0.0000
   -0.5        0.0200     0.0000
   -0.3        0.02      0.0000
    0.0        0.0200     0.0000
    0.3        0.02      0.0000
    0.5        0.02      0.0000
    0.8        0.02      0.0000
    1.0        0.02      0.0000
    1.25       0.035     0.0000
    0.985      0.13      0.0000
    0.9        0.2 -.2
    0.7        0.275     -.4
    0.745      0.375     -.48
    0.8        0.4025    -.6
    0.85       0.52      -.68
    0.96       0.602     -.72

//////////////////////////////////////////////////////////////
// This is the file c_mrot2.tab which contains the low resolution
// data tables for the main rotor for psh
//////////////////////////////////////////////////////////////

  73 3
     0.0  0.02     0.0
     0.6  0.05     -1.0
     0.72     0.125      1.4
     0.7  0.25     1.2
     0.67     0.3  1.28
     0.8  0.475    1.5
     0.94     0.65       1.8
     1.025    0.8  2.0
     1.06     1.05       2.16
     1.03     1.2  2.24
     1.0  1.395    2.32
     0.925    1.525      2.36
     0.82     1.65       2.376
     0.725    1.8  2.4
     0.575    1.875      2.376
     0.425    1.95       2.36
     0.275    2.02       2.34
     0.1  2.05     2.32
     -0.75    2.075      2.3
     -0.23    2.07       2.2
     -0.4     2.025      2.12
     -0.55    2.0  2.0
     -0.7     1.93       1.92
     -0.804   1.83       1.8
```

```
-0.975   1.7  1.7
-1.06    1.575      1.6
-1.14    1.4  1.4
-1.16    1.25       1.2
-1.15    1.04       1.04
-1.06    0.875      0.8
-0.9     0.6  0.6
-0.7     0.41       0.48
-0.674   0.25       0.34
-0.9     0.1  0.12
-1.12    0.045      0.0
-0.5     0.02       0.0
0.0  0.02     0.0
0.5  0.02     0.0
1.0  0.02     0.0
0.985    0.13       0.0
0.7  0.275    -0.4
0.8  0.4025   -0.6
0.96     0.602      -0.72
1.1  0.8      -0.8
1.125    1.0  -1.0
1.125    1.2  -1.2
1.11     1.36       -1.3
1.025    1.5  -1.48
0.975    1.675      -1.6
0.85     1.8  -1.7
0.71     1.9  -1.8
0.575    1.98       -1.9
0.4  2.03     -2.0
0.23     2.07       -2.096
0.75     2.075      -2.2
-0.75    2.07       -2.28
-0.225   2.03       -2.3
-0.4     1.98       -2.32
-0.55    1.9  -2.36
-0.68    1.8  -2.376
-0.79    1.675      -2.376
-0.88    1.55       -2.36
-0.98    1.4  -2.32
-1.03    1.21       -2.28
-1.06    1.075      -2.2
-1.05    0.9  -2.12
-0.975   0.7  -1.88
-0.775   0.5  -1.56
-0.7     0.35       -1.3
-0.75    0.275      -1.3
-0.775   0.175      -1.44
-0.6     0.05       -1.2
0.0  0.02     0.0//
```

```
////////////////////////////////////////////////////////////
// This is the file c_wfaero.exc which contains the aerodynamic
// data tables for the fuselage for psh
////////////////////////////////////////////////////////////

// Wing Fuselage Aero Tables  Low Angle High Resolution
//
waoat1=[d2r*[-25 25 5] 11];  // Low  angle, high resolution alpha brkpts
waost1=[d2r*[-25 25 5] 11];  // Low  angle, high resolution beta  brkpts
waoat2=[d2r*[-90 90 10] 19];  // High angle, low  resolution alpha
brkpts
waost2=[d2r*[-90 90 10] 19];  // High angle, low  resolution beta
brkpts
waoat3=[d2r*[-90 90 180] 2];// Cross coupling angle of attack brkpts

wna1 = waoat1(4);       // Number of points in low  angle alpha tables
wna2 = waoat2(4);       // Number of points in high angle alpha tables
wna3 = waoat3(4);       // Number of points in cross couple alpha tables
wnb1 = waost1(4);       // Number of points in low  angle beta  tables
wnb2 = waost2(4);       // Number of points in high angle beta tables

// Low angle, high resolution Lift, Drag, and Pitch Moment tables as a
// function of alpha

clw1=[-35.0; -29.0; -22.0; -15.0; -8.0;
       -1.50;   5.0;  12.0;  18.5;  25.0;
       31.5];

cdw1=[30.0;  25.0;  22.5;  20.0;  18.0;
      17.5;  18.5;  21.0;  23.5;  27.0;
      32.5];

cmw1=[-900.0; -770.0; -615.0; -460.0; -305.0;
      -155.0;  0.0;   145.0;  300.0;  450.0;
       590.0];

// Low angle, high resolution Sideforce, Rolling and Yawing moment
tables as a
// function of beta and alpha

cyw1=[87.5; 62.5; 50.0; 31.5; 14.0;
      0.0; -14.0; -31.5; -50.0; -62.5;
      -87.5];
     cyw1 = [cyw1 cyw1];

crw1=[ -95.0;  -75.0;  -60.0;  -37.5;  -18.75;
         4.0;   25.0;   43.75;   62.5;  81.25;
       100.0];
     crw1 = [crw1 crw1];

cnw1=[325.0; 275.0; 210.0; 150.0; 80.0;
      0.0;    -80.0;  -150.0; -210.0; -275.0;
      -325.0];
     cnw1 = [cnw1 cnw1];
```

```
// High angle, low resolution Lift, Drag, and Pitch Moment tables as a
// function of alpha

     xi=(-25:5:25)';    // Independent vars for cmba1
     x=(-90:10:90)'; // New independent var for cmba2
     clw2 = odli(x,xi,clw1);// 1-D Linear Interpolation
     cdw2 = odli(x,xi,cdw1);// 1-D Linear Interpolation
     cmw2 = odli(x,xi,cmw1);// 1-D Linear Interpolation

// High angle, low resolution Sideforce, Rolling and Yawing moment
tables
// as a function of beta and alpha

        cyw2 = odli(x,xi,cyw1(:,1));// 1-D Linear Interpolation
        crw2 = odli(x,xi,crw1(:,1));// 1-D Linear Interpolation
        cnw2 = odli(x,xi,cnw1(:,1));// 1-D Linear Interpolation
     cyw2 = [cyw2 cyw2];
     crw2 = [crw2 crw2];
     cnw2 = [cnw2 cnw2];

//  Increments to lift, drag and pitch moment due to sideslip and angle
of
//  attack

clba1=zeros(11,2);

cmba1=zeros(11,1);

cdb1=zeros(11,1);

cdb2=zeros(19,1);

// Create a high angle low resolution table from
// the low angle high resolution table

     xi=(-25:5:25)';    // Independent vars for cmba1,cdb1
     x=(-90:10:90)'; // New independent var for cmba2,cdb2
     cmba2 = odli(x,xi,cmba1);// 1-D Linear Interpolation
     cmba1 = [cmba1 cmba1];
     cmba2 = [cmba2 cmba2];

// End of c_wfaero.exc
```

```
////////////////////////////////////////////////////////////
// This is the file mpl.exc which contains the
// data for the mass distribution along a rotor blade for psh
//  Data from a uniform blade weight distribution analysis
////////////////////////////////////////////////////////////

mpl=[0.000  0.372
     0.050  0.372
     0.100  0.372
     0.150  0.372
     0.200  0.372
     0.250  0.372
     0.300  0.372
     0.350  0.372
     0.400  0.372
     0.450  0.372
     0.500  0.372
     0.550  0.372
     0.600  0.372
     0.650  0.372
     0.700  0.372
     0.750  0.372
     0.800  0.372
     0.850  0.372
     0.900  0.372
     0.950  0.372
     1.000  0.372];

////////////////////////////////////////////////////////////
// This is the file blade_twist.exc which contains the
// data for the twist distribution along a rotor blade for psh
////////////////////////////////////////////////////////////

//  Prouty's Sample Helicopter Main Rotor Preset Blade Twist
//  From  p648 Helicopter Performance, Stability and Control
//  Figure 10.6

btw=[0.000   0.000
     0.050  -0.500
     0.100  -1.000
     0.150  -1.500
     0.200  -2.000
     0.250  -2.500
     0.300  -3.000
     0.350  -3.500
     0.400  -4.000
     0.450  -4.500
     0.500  -5.000
     0.550  -5.500
     0.600  -6.000
     0.650  -6.500
     0.700  -7.000
     0.750  -7.500
     0.800  -8.000
     0.850  -8.500
     0.900  -9.000
     0.950  -9.500
     1.000  -10.00];
```

```
////////////////////////////////////////////////////////////
// This is the file wf_intf.exc which contains the
// data for the wing fuselage interference effects for psh
////////////////////////////////////////////////////////////

//
// Dynamic Presure Loss at Wing / Fuselage
//
wfaost1 = [d2r*[-90 90 180] 2];
wfaoat1 = [d2r*[-90 90 180] 2];

wfqdyn=[ 1 1; 1 1];     // represents no dynamic pressure loss

wfns1 = wfaost1(4);
wfna1 = wfaoat1(4);

// Fuselage downwash at Fuselage

wfaost2 = [d2r*[-90 90 180] 2];
wfaoat2 = [d2r*[-90 90 180] 2];

fvzwf=[ 0 0; 0 0];      // no downwash at fuselage caused by fuselage

wfns2 = wfaost2(4);
wfna2 = wfaoat2(4);

// Fuselage sidewash at Fuselage

wfaost3 = [d2r*[-90 90 180] 2];
wfaoat3 = [d2r*[-90 90 180] 2];

fvywf=[ 0 0; 0 0];      // no sidewash at fuselage caused by fuselage

wfns3 = wfaost3(4);
wfna3 = wfaoat3(4);
//
//  Rotor Wash on Fuselage
//
wfchit = [d2r*[0 100 10] 11];
wfa1ft = [d2r*[-6 6 6] 3];
wfnchi = wfchit(4);
wfna1f = wfa1ft(4);

rvxwfi= zeros(11,3);    // interference velocity,Vx  due to rotor
rvywfi = zeros(11,3);   // interference velocity,Vy  due to rotor

rvzwfi=-1*ones(11,3);   // interference velocity,Vz  due to rotor

// End of wf_intf.exc
```

```
////////////////////////////////////////////////////////////
// This is the file ht_intf.exc which contains the
// data for the horizontal tail interference effects for psh
////////////////////////////////////////////////////////////

//
// Dynamic Presure Loss at Horizontal Tail
//
htaost1 = [d2r*[-90 90 180] 2];
htaoat1 = [d2r*[-90 90 180] 2];

htqdyn=[ 1 1; 1 1];
htns1 = htaost1(4);
htna1 = htaoat1(4);

// Fuselage downwash at Horizontal Tail

htaost2 = [d2r*[-90 90 180] 2];
htaoat2 = [d2r*[-90 90 180] 2];

fvzht=[ 0 0; 0 0];

htns2 = htaost2(4);
htna2 = htaoat2(4);

// Fuselage sidewash at Horizontal Tail

htaost3 = [d2r*[-90 90 180] 2];
htaoat3 = [d2r*[-90 90 180] 2];

fvyht=[ 0 0; 0 0];

htns3 = htaost3(4);
htna3 = htaoat3(4);
//
//  Rotor Wash on Horizontal Tail
//
htchit = [d2r*[0 100 10] 11];
htalft = [d2r*[-6 6 6] 3];
htnchi = htchit(4);
htnalf = htalft(4);

rvxhti= zeros(11,3);

rvyhti = zeros(11,3);

rvzhti= zeros(11,3);

// End of ht_intf.exc
```

```
////////////////////////////////////////////////////////////
// This is the file tr_intf.exc which contains the
// data for the tail rotor interference effects for psh
////////////////////////////////////////////////////////////
// Dynamic Presure Loss at Tail Rotor is same as vertical tail
//
//  Downwash Component of Fuselage Wash on Tail Rotor is same as
//  horizontal tail
//
//  Sidewash Component of Rotor Wash on Tail Rotor is same as
//  vertical tail
//
// Rotor wash on Tail Rotor is same as
// horizontal tail
// End of tr_intf

////////////////////////////////////////////////////////////
// This is the file vt_intf.exc which contains the
// data for the vertical tail interference effects for psh
////////////////////////////////////////////////////////////
//
// Dynamic Presure Loss at Vertical Tail
vtaost1 = [d2r*[-90 90 180] 2];
vtaoat1 = [d2r*[-90 90 180] 2];

vtqdyn=[ 1 1; 1 1];
vtns1 = vtaost1(4);
vtna1 = vtaoat1(4);

// Fuselage downwash at Vertical Tail

vtaost2 = [d2r*[-90 90 180] 2];
vtaoat2 = [d2r*[-90 90 180] 2];

fvzvt=[ 0 0; 0 0];

vtns2 = vtaost2(4);
vtna2 = vtaoat2(4);

// Fuselage sidewash at Vertical Tail

vtaost3 = [d2r*[-90 90 180] 2];
vtaoat3 = [d2r*[-90 90 180] 2];

fvyvt=[ 0 0; 0 0];

vtns3 = vtaost3(4);
vtna3 = vtaoat3(4);
//
//  Rotor Wash on Vertical Tail
vtchit = [d2r*[0 100 10] 11];
vta1ft = [d2r*[-6 6 6] 3];
vtnchi = vtchit(4);
vtna1f = vta1ft(4);

rvxvti= zeros(11,3);

rvyvti = zeros(11,3);

rvzvti= zeros(11,3);
// End of vt_intf.exc
```

```
////////////////////////////////////////////////////////////
// This is the psh.exc file which contains the script
// file generated from the psh.mod
////////////////////////////////////////////////////////////

GOTO WORLD
GROUP model
GROUP Drivetrain
SINK mrsout;
NSUMJ mux2;
SOURCE rra;
INTTF rai;
INTTF rsi;
NSUMJ mux1;
CONNECT mux2(1) TO mrsout(1);
CONNECT rra(1) TO rai(1);
CONNECT rai(1) TO rsi(1);
CONNECT rsi(1) TO mux1(1);
CONNECT rra(1) TO mux2(2);
CONNECT rai(1) TO mux1(2);
CONNECT mux1(1) TO mux2(1);


PARENTG
GROUP cont
GROUP Dir
SINK Thettr;
GAIN k1;
GAIN Thettrrad;
LIMITER Thettrlim;
SUMJ sj3;
SOURCE Thrchk;
SOURCE Xp;
SOURCE Xptrm;
SUMJ sj1;
SUMJ sj2;
GAIN Bias;
CONNECT Thettrrad(1) TO Thettr(1);
CONNECT Thrchk(1) TO sj3(2);
CONNECT sj2(1) TO sj3(1);
CONNECT Bias(1) TO sj2(2);
CONNECT k1(1) TO sj2(1);
CONNECT sj1(1) TO k1(1);
CONNECT Thettrlim(1) TO Thettrrad(1);
CONNECT sj3(1) TO Thettrlim(1);
CONNECT Xp(1) TO sj1(1);
CONNECT Xptrm(1) TO sj1(2);


PARENTG
GROUP Sensors
SOURCE Beta;
SOURCE Alpha;
NGAIN r;
NGAIN q;
NGAIN p;
NGAIN psi;
NGAIN theta;
NGAIN Phi;
GAIN Phideg;
GAIN Thetadeg;
```

```
GAIN Rdeg;
GAIN Qdeg;
GAIN Pdeg;
GAIN Psideg;
SOURCE Unity;
SOLO SOLO283;
CONNECT r(1) TO Rdeg(1);
CONNECT q(1) TO Qdeg(1);
CONNECT p(1) TO Pdeg(1);
CONNECT psi(1) TO Psideg(1);
CONNECT theta(1) TO Thetadeg(1);
CONNECT Phi(1) TO Phideg(1);
CONNECT Rdeg(1) TO SOLO283(1);


PARENTG
GROUP Coll
SINK mTheta;
GAIN k1;
GAIN Theta0rad;
LIMITER Theta0Lim;
SUMJ sj3;
SOURCE Th0chk;
SOURCE Xc;
SOURCE Xctrm;
SUMJ sj1;
SUMJ sj2;
GAIN Bias;
CONNECT Theta0rad(1) TO mTheta(1);
CONNECT Bias(1) TO sj2(2);
CONNECT sj1(1) TO k1(1);
CONNECT sj2(1) TO sj3(1);
CONNECT Th0chk(1) TO sj3(2);
CONNECT Theta0Lim(1) TO Theta0rad(1);
CONNECT sj3(1) TO Theta0Lim(1);
CONNECT k1(1) TO sj2(1);
CONNECT Xctrm(1) TO sj1(2);
CONNECT Xc(1) TO sj1(1);


PARENTG
GROUP Lat
SINK A1s;
GAIN A1srad;
LIMITER A1slim;
SOURCE A1schk;
SUMJ Sj3;
SUMJ sj2;
GAIN k1;
SUMJ sj1;
SOURCE Xatrm;
SOURCE Xa;
GAIN Bias;
CONNECT A1srad(1) TO A1s(1);
CONNECT sj2(1) TO Sj3(1);
CONNECT A1schk(1) TO Sj3(2);
CONNECT Bias(1) TO sj2(2);
CONNECT k1(1) TO sj2(1);
CONNECT sj1(1) TO k1(1);
CONNECT A1slim(1) TO A1srad(1);
CONNECT Sj3(1) TO A1slim(1);
```

```
CONNECT Xatrm(1) TO sj1(2);
CONNECT Xa(1) TO sj1(1);


PARENTG
GROUP Lng
SINK Bls;
GAIN Blsrad;
LIMITER Blslim;
SUMJ sj3;
SOURCE Blschk;
SUMJ sj2;
SOURCE Xb;
SOURCE Xbtrm;
SUMJ sj1;
GAIN k3;
GAIN Bias;
CONNECT Blsrad(1) TO Bls(1);
CONNECT Blslim(1) TO Blsrad(1);
CONNECT Blschk(1) TO sj3(2);
CONNECT sj2(1) TO sj3(1);
CONNECT k3(1) TO sj2(1);
CONNECT Bias(1) TO sj2(2);
CONNECT sj1(1) TO k3(1);
CONNECT sj3(1) TO Blslim(1);
CONNECT Xb(1) TO sj1(1);
CONNECT Xbtrm(1) TO sj1(2);


PARENTG
CONNECT Sensors_Unity(1) TO Lat_Bias(1);
CONNECT Sensors_Unity(1) TO Dir_Bias(1);
CONNECT Sensors_Unity(1) TO Lng_Bias(1);
CONNECT Sensors_Unity(1) TO Coll_Bias(1);


PARENTG
GROUP heli
ATMOSPHERE atmos;
GROUP body
ROTATE htxr2;
MSENSOR dofsensor;
TRANSLATE3 wfloc;
TRANSLATE3 cgloc;
AERO3DS wf;
DMASS body;
DOF6 bodydof;
AERO2D3D Hstab;
AERO2D3D Vstab;
TRANSLATE3 trboom;
TRANSLATE3 vtboom;
ROTATE htzr;
ROTATE htxr;
TRANSLATE3 htboom;
ROTATE vtyr;
ROTATE vtxr;
ROTATE trxr;
BAILEY Trotor;
CONNECT htxr2(1) TO Hstab(1);
CONNECT htzr(1) TO htxr2(1);
CONNECT bodydof(1) TO wfloc(1);
```

```
CONNECT wfloc(1) TO wf(1);
CONNECT cgloc(1) TO body(1);
CONNECT bodydof(1) TO cgloc(1);
CONNECT bodydof(1) TO dofsensor(1);
CONNECT trxr(1) TO Trotor(1);
CONNECT trboom(1) TO trxr(1);
CONNECT vtxr(1) TO Vstab(1);
CONNECT vtyr(1) TO vtxr(1);
CONNECT vtboom(1) TO vtyr(1);
CONNECT htboom(1) TO htxr(1);
CONNECT htxr(1) TO htzr(1);
CONNECT bodydof(1) TO trboom(1);
CONNECT bodydof(1) TO htboom(1);
CONNECT bodydof(1) TO vtboom(1);


PARENTG
GROUP rotor
GROUP blade1
TRANSLATE blseg6;
TRANSLATE spar;
PMASS sparmass;
TRANSLATE blseg1;
ROTATE twst1;
PMASS mass1;
AERO2D aero1;
AERO2D aero2;
PMASS mass2;
ROTATE twst2;
TRANSLATE blseg2;
AERO2D aero3;
PMASS mass3;
ROTATE twst3;
TRANSLATE blseg3;
AERO2D aero4;
PMASS mass4;
ROTATE twst4;
TRANSLATE blseg4;
AERO2D aero5;
PMASS mass5;
ROTATE twst5;
TRANSLATE blseg5;
MARKPOS tip;
CHINGE feat;
TRANSLATE hof1;
ROTATE rotr1;
HINGE flap;
TORSPDM Lead;
CONNECT Lead(1) TO flap(1);
CONNECT feat(1) TO spar(1);
CONNECT spar(1) TO sparmass(1);
CONNECT blseg1(1) TO mass1(1);
CONNECT blseg1(1) TO twst1(1);
CONNECT twst1(1) TO aero1(1);
CONNECT twst2(1) TO aero2(1);
CONNECT blseg2(1) TO twst2(1);
CONNECT blseg2(1) TO mass2(1);
CONNECT twst3(1) TO aero3(1);
CONNECT blseg3(1) TO twst3(1);
CONNECT blseg3(1) TO mass3(1);
CONNECT twst4(1) TO aero4(1);
```

```
CONNECT blseg4(1) TO twst4(1);
CONNECT blseg4(1) TO mass4(1);
CONNECT twst5(1) TO aero5(1);
CONNECT blseg5(1) TO twst5(1);
CONNECT blseg5(1) TO mass5(1);
CONNECT spar(1) TO blseg1(1);
CONNECT blseg1(1) TO blseg2(1);
CONNECT blseg2(1) TO blseg3(1);
CONNECT blseg3(1) TO blseg4(1);
CONNECT blseg4(1) TO blseg5(1);
CONNECT blseg5(1) TO blseg6(1);
CONNECT blseg6(1) TO tip(1);
CONNECT flap(1) TO feat(1);
CONNECT rotr1(1) TO hof1(1);
CONNECT hof1(1) TO Lead(1);


PARENTG
TPP tppc;
ROTATE hub;
TRANSLATE3 mrshaft;
GROUP blade2
TORSPDM Lead;
HINGE flap;
ROTATE rotr1;
TRANSLATE hof1;
CHINGE feat;
MARKPOS tip;
TRANSLATE blseg5;
ROTATE twst5;
PMASS mass5;
AERO2D aero5;
TRANSLATE blseg4;
ROTATE twst4;
PMASS mass4;
AERO2D aero4;
TRANSLATE blseg3;
ROTATE twst3;
PMASS mass3;
AERO2D aero3;
TRANSLATE blseg2;
ROTATE twst2;
PMASS mass2;
AERO2D aero2;
AERO2D aero1;
PMASS mass1;
ROTATE twst1;
TRANSLATE blseg1;
PMASS sparmass;
TRANSLATE spar;
TRANSLATE blseg6;
CONNECT Lead(1) TO flap(1);
CONNECT hof1(1) TO Lead(1);
CONNECT rotr1(1) TO hof1(1);
CONNECT flap(1) TO feat(1);
CONNECT blseg6(1) TO tip(1);
CONNECT blseg5(1) TO blseg6(1);
CONNECT blseg4(1) TO blseg5(1);
CONNECT blseg3(1) TO blseg4(1);
CONNECT blseg2(1) TO blseg3(1);
CONNECT blseg1(1) TO blseg2(1);
```

```
CONNECT spar(1) TO blseg1(1);
CONNECT blseg5(1) TO mass5(1);
CONNECT blseg5(1) TO twst5(1);
CONNECT twst5(1) TO aero5(1);
CONNECT blseg4(1) TO mass4(1);
CONNECT blseg4(1) TO twst4(1);
CONNECT twst4(1) TO aero4(1);
CONNECT blseg3(1) TO mass3(1);
CONNECT blseg3(1) TO twst3(1);
CONNECT twst3(1) TO aero3(1);
CONNECT blseg2(1) TO mass2(1);
CONNECT blseg2(1) TO twst2(1);
CONNECT twst2(1) TO aero2(1);
CONNECT twst1(1) TO aero1(1);
CONNECT blseg1(1) TO twst1(1);
CONNECT blseg1(1) TO mass1(1);
CONNECT spar(1) TO sparmass(1);
CONNECT feat(1) TO spar(1);


PARENTG
GROUP blade3
TRANSLATE blseg6;
TRANSLATE spar;
PMASS sparmass;
TRANSLATE blseg1;
ROTATE twst1;
PMASS mass1;
AERO2D aero1;
AERO2D aero2;
PMASS mass2;
ROTATE twst2;
TRANSLATE blseg2;
AERO2D aero3;
PMASS mass3;
ROTATE twst3;
TRANSLATE blseg3;
AERO2D aero4;
PMASS mass4;
ROTATE twst4;
TRANSLATE blseg4;
AERO2D aero5;
PMASS mass5;
ROTATE twst5;
TRANSLATE blseg5;
MARKPOS tip;
CHINGE feat;
TRANSLATE hof1;
ROTATE rotr1;
HINGE flap;
TORSPDM Lead;
CONNECT Lead(1) TO flap(1);
CONNECT feat(1) TO spar(1);
CONNECT spar(1) TO sparmass(1);
CONNECT blseg1(1) TO mass1(1);
CONNECT blseg1(1) TO twst1(1);
CONNECT twst1(1) TO aero1(1);
CONNECT twst2(1) TO aero2(1);
CONNECT blseg2(1) TO twst2(1);
CONNECT blseg2(1) TO mass2(1);
CONNECT twst3(1) TO aero3(1);
```

```
CONNECT blseg3(1) TO twst3(1);
CONNECT blseg3(1) TO mass3(1);
CONNECT twst4(1) TO aero4(1);
CONNECT blseg4(1) TO twst4(1);
CONNECT blseg4(1) TO mass4(1);
CONNECT twst5(1) TO aero5(1);
CONNECT blseg5(1) TO twst5(1);
CONNECT blseg5(1) TO mass5(1);
CONNECT spar(1) TO blseg1(1);
CONNECT blseg1(1) TO blseg2(1);
CONNECT blseg2(1) TO blseg3(1);
CONNECT blseg3(1) TO blseg4(1);
CONNECT blseg4(1) TO blseg5(1);
CONNECT blseg5(1) TO blseg6(1);
CONNECT blseg6(1) TO tip(1);
CONNECT flap(1) TO feat(1);
CONNECT rotr1(1) TO hof1(1);
CONNECT hof1(1) TO Lead(1);


PARENTG
GROUP blade4
TRANSLATE blseg6;
TRANSLATE spar;
PMASS sparmass;
TRANSLATE blseg1;
ROTATE twst1;
PMASS mass1;
AERO2D aero1;
AERO2D aero2;
PMASS mass2;
ROTATE twst2;
TRANSLATE blseg2;
AERO2D aero3;
PMASS mass3;
ROTATE twst3;
TRANSLATE blseg3;
AERO2D aero4;
PMASS mass4;
ROTATE twst4;
TRANSLATE blseg4;
AERO2D aero5;
PMASS mass5;
ROTATE twst5;
TRANSLATE blseg5;
MARKPOS tip;
CHINGE feat;
TRANSLATE hof1;
ROTATE rotr1;
HINGE flap;
TORSPDM Lead;
CONNECT Lead(1) TO flap(1);
CONNECT feat(1) TO spar(1);
CONNECT spar(1) TO sparmass(1);
CONNECT blseg1(1) TO mass1(1);
CONNECT blseg1(1) TO twst1(1);
CONNECT twst1(1) TO aero1(1);
CONNECT twst2(1) TO aero2(1);
CONNECT blseg2(1) TO twst2(1);
CONNECT blseg2(1) TO mass2(1);
CONNECT twst3(1) TO aero3(1);
```

```
CONNECT blseg3(1) TO twst3(1);
CONNECT blseg3(1) TO mass3(1);
CONNECT twst4(1) TO aero4(1);
CONNECT blseg4(1) TO twst4(1);
CONNECT blseg4(1) TO mass4(1);
CONNECT twst5(1) TO aero5(1);
CONNECT blseg5(1) TO twst5(1);
CONNECT blseg5(1) TO mass5(1);
CONNECT spar(1) TO blseg1(1);
CONNECT blseg1(1) TO blseg2(1);
CONNECT blseg2(1) TO blseg3(1);
CONNECT blseg3(1) TO blseg4(1);
CONNECT blseg4(1) TO blseg5(1);
CONNECT blseg5(1) TO blseg6(1);
CONNECT blseg6(1) TO tip(1);
CONNECT flap(1) TO feat(1);
CONNECT rotr1(1) TO hof1(1);
CONNECT hof1(1) TO Lead(1);


PARENTG
CHINGE mrspeed;
SWASHPLATE sp 1,world_data_nblades;
CONNECT mrspeed(1) TO blade1_rotr1(1);
CONNECT hub(1) TO tppc(1);
CONNECT mrshaft(1) TO hub(1);
CONNECT hub(1) TO mrspeed(1);
CONNECT mrspeed(1) TO blade2_rotr1(1);
CONNECT mrspeed(1) TO blade3_rotr1(1);
CONNECT mrspeed(1) TO blade4_rotr1(1);
CONNECT sp(3) TO blade3_feat(2);
CONNECT sp(4) TO blade4_feat(2);
CONNECT sp(2) TO blade2_feat(2);
CONNECT sp(1) TO blade1_feat(2);


PARENTG
INERTIAL earth;
GROUP inflow
INTERFER Wfi;
INTERFER Hti;
INTERFER Vti;
INTERFER Tri;
UNIFORMIV u1;


PARENTG
CONNECT earth(1) TO body_bodydof(1);
CONNECT body_bodydof(1) TO rotor_mrshaft(1);
CONNECT rotor_hub(1) TO inflow_u1(1);


PARENTG
CONNECT Drivetrain_mrsout(1) TO heli_rotor_mrspeed(2);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_Phi(1);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_theta(1);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_psi(1);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_p(1);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_q(1);
CONNECT heli_body_dofsensor(2) TO cont_Sensors_r(1);
```

```
PARENTG
ATMOSYSTEM Atmosys;
AEROSYSTEM Aerosys;
CONNECT Aerosys(1) TO model_heli_inflow(1);
CONNECT Aerosys(1) TO model_heli_rotor(1);
CONNECT Atmosys(1) TO model(1);


PARENTG
GOTO model
GOTO Drivetrain
     mrsout_N      =3;

     mux2_NI1      =2;
     mux2_NI2      =1;
     mux2_NO  =3;
     mux2_SM  =[1 0 0;0 1 0;0 0 1];

     rra_N    =1;
     rra_U    =0;

     rai_N    =1;

     rsi_N    =1;

     mux1_NI1      =1;
     mux1_NI2      =1;
     mux1_NO  =2;
     mux1_SM  =[1 0;0 1];


PARENTG
GOTO cont
GOTO Dir
     Thettr_N      =1;

     k1_N     =1;
     k1_K     =&world_data_kxpthr;

     Thettrrad_N   =1;
     Thettrrad_K   =&world_data_d2r;

     Thettrlim_N   =1;
     Thettrlim_LL  =&world_data_Thrll;
     Thettrlim_UL  =&world_data_Thrul;

     sj3_N    =1;
     sj3_SA1  =1;
     sj3_SA2  =1;

     Thrchk_N      =1;
     Thrchk_U      =&world_data_thrchk;

     Xp_N     =1;
     Xp_U     =&world_data_xp;

     Xptrm_N  =1;
     Xptrm_U  =&world_data_xptrm;

     sj1_N    =1;
```

```
      sj1_SA1  =1;
      sj1_SA2  =1;

      sj2_N    =1;
      sj2_SA1  =1;
      sj2_SA2  =1;

      Bias_N   =1;
      Bias_K   =&world_data_Thrbias;


PARENTG
GOTO Sensors
      Beta_N   =1;
      Beta_U   =&world_data_betwf;

      Alpha_N  =1;
      Alpha_U  =&world_data_alfwf;

      r_NO     =1;
      r_NI     =6;
      r_K  =[0 0 0 0 0 1];

      q_NO     =1;
      q_NI     =6;
      q_K  =[0 0 0 0 1 0];

      p_NO     =1;
      p_NI     =6;
      p_K  =[0 0 0 1 0 0];

      psi_NO   =1;
      psi_NI   =6;
      psi_K    =[0 0 1 0 0 0];

      theta_NO      =1;
      theta_NI      =6;
      theta_K =[0 1 0 0 0 0];

      Phi_NO   =1;
      Phi_NI   =6;
      Phi_K    =[1 0 0 0 0 0];

      Phideg_N      =1;
      Phideg_K      =&world_data_r2d;

      Thetadeg_N    =1;
      Thetadeg_K    =&world_data_r2d;

      Rdeg_N   =1;
      Rdeg_K   =&world_data_r2d;

      Qdeg_N   =1;
      Qdeg_K   =&world_data_r2d;

      Pdeg_N   =1;
      Pdeg_K   =&world_data_r2d;

      Psideg_N      =1;
      Psideg_K      =&world_data_r2d;
```

```
     Unity_N =1;
     Unity_U =1;

     SOLO283_N     =1;
     SOLO283_ZETA =0.707;
     SOLO283_OMEGA      =10;


PARENTG
GOTO Coll
     mTheta_N      =1;

     k1_N     =1;
     k1_K     =&world_data_kxcth0;

     Theta0rad_N  =1;
     Theta0rad_K  =&world_data_d2r;

     Theta0Lim_N  =1;
     Theta0Lim_LL =&world_data_Th0ll;
     Theta0Lim_UL =&world_data_Th0ul;

     sj3_N    =1;
     sj3_SA1  =1;
     sj3_SA2  =1;

     Th0chk_N      =1;
     Th0chk_U      =&world_data_th0chk;

     Xc_N     =1;
     Xc_U     =&world_data_xc;

     Xctrm_N  =1;
     Xctrm_U  =&world_data_xctrm;

     sj1_N    =1;
     sj1_SA1  =1;
     sj1_SA2  =1;

     sj2_N    =1;
     sj2_SA1  =1;
     sj2_SA2  =1;

     Bias_N   =1;
     Bias_K   =&world_data_Th0bias;


PARENTG
GOTO Lat
     Als_N    =1;

     Alsrad_N      =1;
     Alsrad_K      =&world_data_d2r;

     Alslim_N      =1;
     Alslim_LL     =&world_data_Alsll;
     Alslim_UL     =&world_data_Alsul;

     Alschk_N      =1;
```

```
     A1schk_U      =&world_data_a1schk;

     Sj3_N    =1;
     Sj3_SA1  =1;
     Sj3_SA2  =1;

     sj2_N    =1;
     sj2_SA1  =1;
     sj2_SA2  =1;

     k1_N     =1;
     k1_K     =&world_data_kxaa1s;

     sj1_N    =1;
     sj1_SA1  =1;
     sj1_SA2  =1;

     Xatrm_N  =1;
     Xatrm_U  =&world_data_xatrm;

     Xa_N     =1;
     Xa_U     =&world_data_xa;

     Bias_N   =1;
     Bias_K   =&world_data_A1sbias;


PARENTG
GOTO Lng
     B1s_N    =1;

     B1srad_N      =1;
     B1srad_K      =&world_data_d2r;

     B1slim_N      =1;
     B1slim_LL     =&world_data_B1sll;
     B1slim_UL     =&world_data_B1sul;

     sj3_N    =1;
     sj3_SA1  =1;
     sj3_SA2  =1;

     B1schk_N      =1;
     B1schk_U      =&world_data_b1schk;

     sj2_N    =1;
     sj2_SA1  =1;
     sj2_SA2  =1;

     Xb_N     =1;
     Xb_U     =&world_data_xb;

     Xbtrm_N  =1;
     Xbtrm_U  =&world_data_xbtrm;

     sj1_N    =1;
     sj1_SA1  =1;
     sj1_SA2  =1;

     k3_N     =1;
```

```
     k3_K     =&world_data_kxbb1s;

     Bias_N   =1;
     Bias_K   =&world_data_B1sbias;


PARENTG


PARENTG
GOTO heli
     atmos_XATMO   =&world_data_pos;
     atmos_ALTT    =&world_data_altt;
     atmos_NPOINTS      =&world_data_natmo;
     atmos_RHOTABLE     =&world_data_densityt;
     atmos_SOSTABLE     =&world_data_ssoundt;
     atmos_GASCN   =8.95e+4/28.93;
     atmos_GAMMA   =1.4;

GOTO body
     htxr2_ANGLE   =&world_data_htincang;
     htxr2_AXIS    =1;

     dofsensor_NO =6;
     dofsensor_K   =zeros(6,18);

     wfloc_LEN     =&world_data_wfloc;

     cgloc_LEN     =&world_data_cgloc;

     wf_ALPBRK     =&world_data_wabp;
     wf_BETBRK     =&world_data_wbbp;
     wf_ARGA1      =&world_data_waoat1;
     wf_ARGA2      =&world_data_waoat2;
     wf_ARGA3      =&world_data_waoat3;
     wf_ARGB1      =&world_data_waost1;
     wf_ARGB2      =&world_data_waost2;
     wf_NALFA1     =&world_data_wna1;
     wf_NALFA2     =&world_data_wna2;
     wf_NALFA3     =&world_data_wna3;
     wf_NBETA1     =&world_data_wnb1;
     wf_NBETA2     =&world_data_wnb2;
     wf_CLAL  =&world_data_clw1;
     wf_CDAL  =&world_data_cdw1;
     wf_CMAL  =&world_data_cmw1;
     wf_CLAH  =&world_data_clw2;
     wf_CDAH  =&world_data_cdw2;
     wf_CMAH  =&world_data_cmw2;
     wf_CLBA  =&world_data_clba1;
     wf_CDBL  =&world_data_cdb1;
     wf_CMBAL      =&world_data_cmba1;
     wf_CNBAL      =&world_data_cnw1;
     wf_CRBAL      =&world_data_crw1;
     wf_CYBAL      =&world_data_cyw1;
     wf_CDBH  =&world_data_cdb2;
     wf_CMBAH      =&world_data_cmba2;
     wf_CNBAH      =&world_data_cnw2;
     wf_CRBAH      =&world_data_crw2;
     wf_CYBAH      =&world_data_cyw2;
```

```
body_MASS     =&world_data_fmass;
body_INERTIA  =&world_data_finertia;
body_GRAVITY  =&world_data_gravity;


Hstab_BRKPTS  =&world_data_hbp;
Hstab_CHORD   =&world_data_hchord;
Hstab_DEFIC   =&world_data_hdefic;
Hstab_LEN     =&world_data_hlen;
Hstab_NALFA1  =&world_data_hna1;
Hstab_NALFA2  =&world_data_hna2;
Hstab_NMACH   =&world_data_hnm1;
Hstab_AOAT1   =&world_data_haoat1;
Hstab_MACHT1  =&world_data_hmacht;
Hstab_AOAT2   =&world_data_haoat2;
Hstab_CL1     =&world_data_clh1;
Hstab_CL2     =&world_data_clh2;
Hstab_CD1     =&world_data_cdh1;
Hstab_CD2     =&world_data_cdh2;
Hstab_CM1     =&world_data_cmh1;
Hstab_CM2     =&world_data_cmh2;

Vstab_BRKPTS  =&world_data_vbp;
Vstab_CHORD   =&world_data_vchord;
Vstab_DEFIC   =&world_data_vdefic;
Vstab_LEN     =&world_data_vlen;
Vstab_NALFA1  =&world_data_vna1;
Vstab_NALFA2  =&world_data_vna2;
Vstab_NMACH   =&world_data_vnm1;
Vstab_AOAT1   =&world_data_vaoat1;
Vstab_MACHT1  =&world_data_vmacht;
Vstab_AOAT2   =&world_data_vaoat2;
Vstab_CL1     =&world_data_clv1;
Vstab_CL2     =&world_data_clv2;
Vstab_CD1     =&world_data_cdv1;
Vstab_CD2     =&world_data_cdv2;
Vstab_CM1     =&world_data_cmv1;
Vstab_CM2     =&world_data_cmv2;

trboom_LEN    =&world_data_trloc;

vtboom_LEN    =&world_data_vtloc;

htzr_ANGLE    =-world_data_pi/2;
htzr_AXIS     =3;

htxr_ANGLE    =world_data_pi;
htxr_AXIS     =1;

htboom_LEN    =&world_data_htloc;

vtyr_ANGLE    =world_data_pi/2;
vtyr_AXIS     =2;

vtxr_ANGLE    =-world_data_pi/2;
vtxr_AXIS     =1;

trxr_ANGLE    =-(90+20)*world_data_d2r;
trxr_AXIS     =1;

Trotor_BIAS   =&world_data_biastr;
```

```
     Trotor_BLDS   =&world_data_trblades;
     Trotor_BTL    =&world_data_btltr;
     Trotor_BVT    =&world_data_bvttr;
     Trotor_BVT1   =&world_data_bvt1tr;
     Trotor_CHRD   =&world_data_chordtr;
     Trotor_DELT   =&world_data_delttr;
     Trotor_OMEG   =&world_data_omegatr;
     Trotor_R      =&world_data_rtr;
     Trotor_THETC  =&world_data_thettr;
     Trotor_TWST   =&world_data_twsttr;
     Trotor_TD3    =&world_data_td3tr;
     Trotor_VBVT   =&world_data_vbvttr;
     Trotor_XIB    =&world_data_xibtr;
     Trotor_XLAM   =&world_data_xlamtr;
     Trotor_A      =&world_data_atr;
     Trotor_CD     =&world_data_cdtr;
     Trotor_D0     =&world_data_d0tr;
     Trotor_D1     =&world_data_d1tr;
     Trotor_D2     =&world_data_d2tr;
     Trotor_DROT   =1;
     Trotor_T      =&world_data_ttr;


PARENTG
GOTO rotor
GOTO blade1
     blseg6_LEN    =&world_data_tipdist;

     spar_LEN      =world_data_sl/2;

     sparmass_MASS      =&world_data_sparmass;
     sparmass_GRAVITY   =&world_data_gravity;

     blseg1_LEN    =&world_data_seglen(1);

     twst1_ANGLE   =&world_data_segtwst(1);
     twst1_AXIS    =1;

     mass1_MASS    =&world_data_segmass(1);
     mass1_GRAVITY      =&world_data_gravity;

     aero1_BRKPTS  =&world_data_mrbp;
     aero1_CHORD   =&world_data_segcor(1);
     aero1_DEFIC   =&world_data_segdef(1);
     aero1_LEN     =&world_data_segwid(1);
     aero1_NALFA1  =&world_data_mrna1;
     aero1_NALFA2  =&world_data_mrna2;
     aero1_NMACH   =&world_data_mrnm1;
     aero1_AOAT1   =&world_data_mraoat1;
     aero1_MACHT1  =&world_data_mrmacht;
     aero1_AOAT2   =&world_data_mraoat2;
     aero1_CL1     =&world_data_clmr1;
     aero1_CL2     =&world_data_clmr2;
     aero1_CD1     =&world_data_cdmr1;
     aero1_CD2     =&world_data_cdmr2;
     aero1_CM1     =&world_data_cmmr1;
     aero1_CM2     =&world_data_cmmr2;

     aero2_BRKPTS  =&world_data_mrbp;
     aero2_CHORD   =&world_data_segcor(2);
```

```
aero2_DEFIC   =&world_data_segdef(2);
aero2_LEN     =&world_data_segwid(2);
aero2_NALFA1  =&world_data_mrna1;
aero2_NALFA2  =&world_data_mrna2;
aero2_NMACH   =&world_data_mrnm1;
aero2_AOAT1   =&world_data_mraoat1;
aero2_MACHT1  =&world_data_mrmacht;
aero2_AOAT2   =&world_data_mraoat2;
aero2_CL1     =&world_data_clmr1;
aero2_CL2     =&world_data_clmr2;
aero2_CD1     =&world_data_cdmr1;
aero2_CD2     =&world_data_cdmr2;
aero2_CM1     =&world_data_cmmr1;
aero2_CM2     =&world_data_cmmr2;

mass2_MASS    =&world_data_segmass(2);
mass2_GRAVITY      =&world_data_gravity;

twst2_ANGLE   =&world_data_segtwst(2);
twst2_AXIS    =1 ;

blseg2_LEN    =&world_data_seglen(2);

aero3_BRKPTS  =&world_data_mrbp;
aero3_CHORD   =&world_data_segcor(3);
aero3_DEFIC   =&world_data_segdef(3);
aero3_LEN     =&world_data_segwid(3);
aero3_NALFA1  =&world_data_mrna1;
aero3_NALFA2  =&world_data_mrna2;
aero3_NMACH   =&world_data_mrnm1;
aero3_AOAT1   =&world_data_mraoat1;
aero3_MACHT1  =&world_data_mrmacht;
aero3_AOAT2   =&world_data_mraoat2;
aero3_CL1     =&world_data_clmr1;
aero3_CL2     =&world_data_clmr2;
aero3_CD1     =&world_data_cdmr1;
aero3_CD2     =&world_data_cdmr2;
aero3_CM1     =&world_data_cmmr1;
aero3_CM2     =&world_data_cmmr2;

mass3_MASS    =&world_data_segmass(3);
mass3_GRAVITY      =&world_data_gravity;

twst3_ANGLE   =&world_data_segtwst(3);
twst3_AXIS    =1;

blseg3_LEN    =&world_data_seglen(3);

aero4_BRKPTS  =&world_data_mrbp;
aero4_CHORD   =&world_data_segcor(4);
aero4_DEFIC   =&world_data_segdef(4);
aero4_LEN     =&world_data_segwid(4);
aero4_NALFA1  =&world_data_mrna1;
aero4_NALFA2  =&world_data_mrna2;
aero4_NMACH   =&world_data_mrnm1;
aero4_AOAT1   =&world_data_mraoat1;
aero4_MACHT1  =&world_data_mrmacht;
aero4_AOAT2   =&world_data_mraoat2;
aero4_CL1     =&world_data_clmr1;
aero4_CL2     =&world_data_clmr2;
aero4_CD1     =&world_data_cdmr1;
```

```
    aero4_CD2     =&world_data_cdmr2;
    aero4_CM1     =&world_data_cmmr1;
    aero4_CM2     =&world_data_cmmr2;

    mass4_MASS    =&world_data_segmass(4);
    mass4_GRAVITY      =&world_data_gravity;

    twst4_ANGLE   =&world_data_segtwst(4);
    twst4_AXIS    =1;

    blseg4_LEN    =&world_data_seglen(4);

    aero5_BRKPTS  =&world_data_mrbp;
    aero5_CHORD   =&world_data_segcor(5);
    aero5_DEFIC   =&world_data_segdef(5);
    aero5_LEN     =&world_data_segwid(5);
    aero5_NALFA1  =&world_data_mrna1;
    aero5_NALFA2  =&world_data_mrna2;
    aero5_NMACH   =&world_data_mrnm1;
    aero5_AOAT1   =&world_data_mraoat1;
    aero5_MACHT1  =&world_data_mrmacht;
    aero5_AOAT2   =&world_data_mraoat2;
    aero5_CL1     =&world_data_clmr1;
    aero5_CL2     =&world_data_clmr2;
    aero5_CD1     =&world_data_cdmr1;
    aero5_CD2     =&world_data_cdmr2;
    aero5_CM1     =&world_data_cmmr1;
    aero5_CM2     =&world_data_cmmr2;

    mass5_MASS    =&world_data_segmass(5);
    mass5_GRAVITY      =&world_data_gravity;

    twst5_ANGLE   =&world_data_segtwst(5);
    twst5_AXIS    =1;

    blseg5_LEN    =&world_data_seglen(5);

    tip_POS  =&world_data_tippos(1:3,1);

    feat_AXIS     =1;

    hof1_LEN      =&world_data_ho;

    rotr1_ANGLE   =0;
    rotr1_AXIS    =3;

    flap_AXIS     =2;

    Lead_AXIS     =3;
    Lead_KK  =0;
    Lead_CC  =&world_data_lagdamper;
    Lead_ANG0     =0;

PARENTG
    tppc_NBLADES  =&world_data_nblades;
    tppc_TIPPOS   =&world_data_tippos;

    hub_ANGLE     =world_data_pi-world_data_imr;
    hub_AXIS      =2;

    mrshaft_LEN   =&world_data_mrloc;
```

```
GOTO blade2
     Lead_AXIS     =3;
     Lead_KK =0;
     Lead_CC =&world_data_lagdamper;
     Lead_ANG0     =0;

     flap_AXIS     =2;

     rotr1_ANGLE   =world_data_pi/2;
     rotr1_AXIS    =3;

     hof1_LEN      =&world_data_ho;

     feat_AXIS     =1;

     tip_POS =&world_data_tippos(1:3,1);

     blseg5_LEN    =&world_data_seglen(5);

     twst5_ANGLE   =&world_data_segtwst(5);
     twst5_AXIS    =1;

     mass5_MASS    =&world_data_segmass(5);
     mass5_GRAVITY      =&world_data_gravity;

     aero5_BRKPTS  =&world_data_mrbp;
     aero5_CHORD   =&world_data_segcor(5);
     aero5_DEFIC   =&world_data_segdef(5);
     aero5_LEN     =&world_data_segwid(5);
     aero5_NALFA1  =&world_data_mrna1;
     aero5_NALFA2  =&world_data_mrna2;
     aero5_NMACH   =&world_data_mrnm1;
     aero5_AOAT1   =&world_data_mraoat1;
     aero5_MACHT1  =&world_data_mrmacht;
     aero5_AOAT2   =&world_data_mraoat2;
     aero5_CL1     =&world_data_clmr1;
     aero5_CL2     =&world_data_clmr2;
     aero5_CD1     =&world_data_cdmr1;
     aero5_CD2     =&world_data_cdmr2;
     aero5_CM1     =&world_data_cmmr1;
     aero5_CM2     =&world_data_cmmr2;

     blseg4_LEN    =&world_data_seglen(4);

     twst4_ANGLE   =&world_data_segtwst(4);
     twst4_AXIS    =1;

     mass4_MASS    =&world_data_segmass(4);
     mass4_GRAVITY      =&world_data_gravity;

     aero4_BRKPTS  =&world_data_mrbp;
     aero4_CHORD   =&world_data_segcor(4);
     aero4_DEFIC   =&world_data_segdef(4);
     aero4_LEN     =&world_data_segwid(4);
     aero4_NALFA1  =&world_data_mrna1;
     aero4_NALFA2  =&world_data_mrna2;
     aero4_NMACH   =&world_data_mrnm1;
     aero4_AOAT1   =&world_data_mraoat1;
     aero4_MACHT1  =&world_data_mrmacht;
     aero4_AOAT2   =&world_data_mraoat2;
     aero4_CL1     =&world_data_clmr1;
```

```
aero4_CL2     =&world_data_clmr2;
aero4_CD1     =&world_data_cdmr1;
aero4_CD2     =&world_data_cdmr2;
aero4_CM1     =&world_data_cmmr1;
aero4_CM2     =&world_data_cmmr2;

blseg3_LEN    =&world_data_seglen(3);

twst3_ANGLE   =&world_data_segtwst(3);
twst3_AXIS    =1;

mass3_MASS    =&world_data_segmass(3);
mass3_GRAVITY      =&world_data_gravity;

aero3_BRKPTS  =&world_data_mrbp;
aero3_CHORD   =&world_data_segcor(3);
aero3_DEFIC   =&world_data_segdef(3);
aero3_LEN     =&world_data_segwid(3);
aero3_NALFA1  =&world_data_mrna1;
aero3_NALFA2  =&world_data_mrna2;
aero3_NMACH   =&world_data_mrnm1;
aero3_AOAT1   =&world_data_mraoat1;
aero3_MACHT1  =&world_data_mrmacht;
aero3_AOAT2   =&world_data_mraoat2;
aero3_CL1     =&world_data_clmr1;
aero3_CL2     =&world_data_clmr2;
aero3_CD1     =&world_data_cdmr1;
aero3_CD2     =&world_data_cdmr2;
aero3_CM1     =&world_data_cmmr1;
aero3_CM2     =&world_data_cmmr2;

blseg2_LEN    =&world_data_seglen(2);

twst2_ANGLE   =&world_data_segtwst(2);
twst2_AXIS    =1 ;

mass2_MASS    =&world_data_segmass(2);
mass2_GRAVITY      =&world_data_gravity;

aero2_BRKPTS  =&world_data_mrbp;
aero2_CHORD   =&world_data_segcor(2);
aero2_DEFIC   =&world_data_segdef(2);
aero2_LEN     =&world_data_segwid(2);
aero2_NALFA1  =&world_data_mrna1;
aero2_NALFA2  =&world_data_mrna2;
aero2_NMACH   =&world_data_mrnm1;
aero2_AOAT1   =&world_data_mraoat1;
aero2_MACHT1  =&world_data_mrmacht;
aero2_AOAT2   =&world_data_mraoat2;
aero2_CL1     =&world_data_clmr1;
aero2_CL2     =&world_data_clmr2;
aero2_CD1     =&world_data_cdmr1;
aero2_CD2     =&world_data_cdmr2;
aero2_CM1     =&world_data_cmmr1;
aero2_CM2     =&world_data_cmmr2;

aero1_BRKPTS  =&world_data_mrbp;
aero1_CHORD   =&world_data_segcor(1);
aero1_DEFIC   =&world_data_segdef(1);
aero1_LEN     =&world_data_segwid(1);
aero1_NALFA1  =&world_data_mrna1;
```

```
      aero1_NALFA2  =&world_data_mrna2;
      aero1_NMACH   =&world_data_mrnm1;
      aero1_AOAT1   =&world_data_mraoat1;
      aero1_MACHT1  =&world_data_mrmacht;
      aero1_AOAT2   =&world_data_mraoat2;
      aero1_CL1     =&world_data_clmr1;
      aero1_CL2     =&world_data_clmr2;
      aero1_CD1     =&world_data_cdmr1;
      aero1_CD2     =&world_data_cdmr2;
      aero1_CM1     =&world_data_cmmr1;
      aero1_CM2     =&world_data_cmmr2;

      mass1_MASS    =&world_data_segmass(1);
      mass1_GRAVITY       =&world_data_gravity;

      twst1_ANGLE   =&world_data_segtwst(1);
      twst1_AXIS    =1;

      blseg1_LEN    =&world_data_seglen(1);

      sparmass_MASS       =&world_data_sparmass;
      sparmass_GRAVITY    =&world_data_gravity;

      spar_LEN      =world_data_sl/2;

      blseg6_LEN    =&world_data_tipdist;

PARENTG
GOTO blade3
      blseg6_LEN    =&world_data_tipdist;

      spar_LEN      =world_data_sl/2;

      sparmass_MASS       =&world_data_sparmass;
      sparmass_GRAVITY    =&world_data_gravity;

      blseg1_LEN    =&world_data_seglen(1);

      twst1_ANGLE   =&world_data_segtwst(1);
      twst1_AXIS    =1;

      mass1_MASS    =&world_data_segmass(1);
      mass1_GRAVITY       =&world_data_gravity;

      aero1_BRKPTS  =&world_data_mrbp;
      aero1_CHORD   =&world_data_segcor(1);
      aero1_DEFIC   =&world_data_segdef(1);
      aero1_LEN     =&world_data_segwid(1);
      aero1_NALFA1  =&world_data_mrna1;
      aero1_NALFA2  =&world_data_mrna2;
      aero1_NMACH   =&world_data_mrnm1;
      aero1_AOAT1   =&world_data_mraoat1;
      aero1_MACHT1  =&world_data_mrmacht;
      aero1_AOAT2   =&world_data_mraoat2;
      aero1_CL1     =&world_data_clmr1;
      aero1_CL2     =&world_data_clmr2;
      aero1_CD1     =&world_data_cdmr1;
      aero1_CD2     =&world_data_cdmr2;
      aero1_CM1     =&world_data_cmmr1;
      aero1_CM2     =&world_data_cmmr2;
```

```
aero2_BRKPTS =&world_data_mrbp;
aero2_CHORD  =&world_data_segcor(2);
aero2_DEFIC  =&world_data_segdef(2);
aero2_LEN    =&world_data_segwid(2);
aero2_NALFA1 =&world_data_mrna1;
aero2_NALFA2 =&world_data_mrna2;
aero2_NMACH  =&world_data_mrnm1;
aero2_AOAT1  =&world_data_mraoat1;
aero2_MACHT1 =&world_data_mrmacht;
aero2_AOAT2  =&world_data_mraoat2;
aero2_CL1    =&world_data_clmr1;
aero2_CL2    =&world_data_clmr2;
aero2_CD1    =&world_data_cdmr1;
aero2_CD2    =&world_data_cdmr2;
aero2_CM1    =&world_data_cmmr1;
aero2_CM2    =&world_data_cmmr2;

mass2_MASS   =&world_data_segmass(2);
mass2_GRAVITY      =&world_data_gravity;

twst2_ANGLE  =&world_data_segtwst(2);
twst2_AXIS   =1 ;

blseg2_LEN   =&world_data_seglen(2);

aero3_BRKPTS =&world_data_mrbp;
aero3_CHORD  =&world_data_segcor(3);
aero3_DEFIC  =&world_data_segdef(3);
aero3_LEN    =&world_data_segwid(3);
aero3_NALFA1 =&world_data_mrna1;
aero3_NALFA2 =&world_data_mrna2;
aero3_NMACH  =&world_data_mrnm1;
aero3_AOAT1  =&world_data_mraoat1;
aero3_MACHT1 =&world_data_mrmacht;
aero3_AOAT2  =&world_data_mraoat2;
aero3_CL1    =&world_data_clmr1;
aero3_CL2    =&world_data_clmr2;
aero3_CD1    =&world_data_cdmr1;
aero3_CD2    =&world_data_cdmr2;
aero3_CM1    =&world_data_cmmr1;
aero3_CM2    =&world_data_cmmr2;

mass3_MASS   =&world_data_segmass(3);
mass3_GRAVITY      =&world_data_gravity;

twst3_ANGLE  =&world_data_segtwst(3);
twst3_AXIS   =1;

blseg3_LEN   =&world_data_seglen(3);

aero4_BRKPTS =&world_data_mrbp;
aero4_CHORD  =&world_data_segcor(4);
aero4_DEFIC  =&world_data_segdef(4);
aero4_LEN    =&world_data_segwid(4);
aero4_NALFA1 =&world_data_mrna1;
aero4_NALFA2 =&world_data_mrna2;
aero4_NMACH  =&world_data_mrnm1;
aero4_AOAT1  =&world_data_mraoat1;
aero4_MACHT1 =&world_data_mrmacht;
aero4_AOAT2  =&world_data_mraoat2;
aero4_CL1    =&world_data_clmr1;
```

```
     aero4_CL2      =&world_data_clmr2;
     aero4_CD1      =&world_data_cdmr1;
     aero4_CD2      =&world_data_cdmr2;
     aero4_CM1      =&world_data_cmmr1;
     aero4_CM2      =&world_data_cmmr2;

     mass4_MASS     =&world_data_segmass(4);
     mass4_GRAVITY       =&world_data_gravity;

     twst4_ANGLE    =&world_data_segtwst(4);
     twst4_AXIS     =1;

     blseg4_LEN     =&world_data_seglen(4);

     aero5_BRKPTS   =&world_data_mrbp;
     aero5_CHORD    =&world_data_segcor(5);
     aero5_DEFIC    =&world_data_segdef(5);
     aero5_LEN      =&world_data_segwid(5);
     aero5_NALFA1   =&world_data_mrna1;
     aero5_NALFA2   =&world_data_mrna2;
     aero5_NMACH    =&world_data_mrnm1;
     aero5_AOAT1    =&world_data_mraoat1;
     aero5_MACHT1   =&world_data_mrmacht;
     aero5_AOAT2    =&world_data_mraoat2;
     aero5_CL1      =&world_data_clmr1;
     aero5_CL2      =&world_data_clmr2;
     aero5_CD1      =&world_data_cdmr1;
     aero5_CD2      =&world_data_cdmr2;
     aero5_CM1      =&world_data_cmmr1;
     aero5_CM2      =&world_data_cmmr2;

     mass5_MASS     =&world_data_segmass(5);
     mass5_GRAVITY       =&world_data_gravity;

     twst5_ANGLE    =&world_data_segtwst(5);
     twst5_AXIS     =1;

     blseg5_LEN     =&world_data_seglen(5);

     tip_POS  =&world_data_tippos(1:3,1);

     feat_AXIS      =1;

     hof1_LEN       =&world_data_ho;

     rotr1_ANGLE    =world_data_pi;
     rotr1_AXIS     =3;

     flap_AXIS      =2;

     Lead_AXIS      =3;
     Lead_KK  =0;
     Lead_CC  =&world_data_lagdamper;
     Lead_ANG0      =0;


PARENTG
GOTO blade4
     blseg6_LEN     =&world_data_tipdist;
```

```
spar_LEN      =world_data_sl/2;

sparmass_MASS      =&world_data_sparmass;
sparmass_GRAVITY   =&world_data_gravity;

blseg1_LEN    =&world_data_seglen(1);

twst1_ANGLE   =&world_data_segtwst(1);
twst1_AXIS    =1;

mass1_MASS    =&world_data_segmass(1);
mass1_GRAVITY      =&world_data_gravity;

aero1_BRKPTS  =&world_data_mrbp;
aero1_CHORD   =&world_data_segcor(1);
aero1_DEFIC   =&world_data_segdef(1);
aero1_LEN     =&world_data_segwid(1);
aero1_NALFA1  =&world_data_mrna1;
aero1_NALFA2  =&world_data_mrna2;
aero1_NMACH   =&world_data_mrnm1;
aero1_AOAT1   =&world_data_mraoat1;
aero1_MACHT1  =&world_data_mrmacht;
aero1_AOAT2   =&world_data_mraoat2;
aero1_CL1     =&world_data_clmr1;
aero1_CL2     =&world_data_clmr2;
aero1_CD1     =&world_data_cdmr1;
aero1_CD2     =&world_data_cdmr2;
aero1_CM1     =&world_data_cmmr1;
aero1_CM2     =&world_data_cmmr2;

aero2_BRKPTS  =&world_data_mrbp;
aero2_CHORD   =&world_data_segcor(2);
aero2_DEFIC   =&world_data_segdef(2);
aero2_LEN     =&world_data_segwid(2);
aero2_NALFA1  =&world_data_mrna1;
aero2_NALFA2  =&world_data_mrna2;
aero2_NMACH   =&world_data_mrnm1;
aero2_AOAT1   =&world_data_mraoat1;
aero2_MACHT1  =&world_data_mrmacht;
aero2_AOAT2   =&world_data_mraoat2;
aero2_CL1     =&world_data_clmr1;
aero2_CL2     =&world_data_clmr2;
aero2_CD1     =&world_data_cdmr1;
aero2_CD2     =&world_data_cdmr2;
aero2_CM1     =&world_data_cmmr1;
aero2_CM2     =&world_data_cmmr2;

mass2_MASS    =&world_data_segmass(2);
mass2_GRAVITY      =&world_data_gravity;

twst2_ANGLE   =&world_data_segtwst(2);
twst2_AXIS    =1 ;

blseg2_LEN    =&world_data_seglen(2);

aero3_BRKPTS  =&world_data_mrbp;
aero3_CHORD   =&world_data_segcor(3);
aero3_DEFIC   =&world_data_segdef(3);
aero3_LEN     =&world_data_segwid(3);
aero3_NALFA1  =&world_data_mrna1;
aero3_NALFA2  =&world_data_mrna2;
```

```
aero3_NMACH   =&world_data_mrnm1;
aero3_AOAT1   =&world_data_mraoat1;
aero3_MACHT1  =&world_data_mrmacht;
aero3_AOAT2   =&world_data_mraoat2;
aero3_CL1     =&world_data_clmr1;
aero3_CL2     =&world_data_clmr2;
aero3_CD1     =&world_data_cdmr1;
aero3_CD2     =&world_data_cdmr2;
aero3_CM1     =&world_data_cmmr1;
aero3_CM2     =&world_data_cmmr2;

mass3_MASS    =&world_data_segmass(3);
mass3_GRAVITY      =&world_data_gravity;

twst3_ANGLE   =&world_data_segtwst(3);
twst3_AXIS    =1;

blseg3_LEN    =&world_data_seglen(3);

aero4_BRKPTS  =&world_data_mrbp;
aero4_CHORD   =&world_data_segcor(4);
aero4_DEFIC   =&world_data_segdef(4);
aero4_LEN     =&world_data_segwid(4);
aero4_NALFA1  =&world_data_mrna1;
aero4_NALFA2  =&world_data_mrna2;
aero4_NMACH   =&world_data_mrnm1;
aero4_AOAT1   =&world_data_mraoat1;
aero4_MACHT1  =&world_data_mrmacht;
aero4_AOAT2   =&world_data_mraoat2;
aero4_CL1     =&world_data_clmr1;
aero4_CL2     =&world_data_clmr2;
aero4_CD1     =&world_data_cdmr1;
aero4_CD2     =&world_data_cdmr2;
aero4_CM1     =&world_data_cmmr1;
aero4_CM2     =&world_data_cmmr2;

mass4_MASS    =&world_data_segmass(4);
mass4_GRAVITY      =&world_data_gravity;

twst4_ANGLE   =&world_data_segtwst(4);
twst4_AXIS    =1;

blseg4_LEN    =&world_data_seglen(4);

aero5_BRKPTS  =&world_data_mrbp;
aero5_CHORD   =&world_data_segcor(5);
aero5_DEFIC   =&world_data_segdef(5);
aero5_LEN     =&world_data_segwid(5);
aero5_NALFA1  =&world_data_mrna1;
aero5_NALFA2  =&world_data_mrna2;
aero5_NMACH   =&world_data_mrnm1;
aero5_AOAT1   =&world_data_mraoat1;
aero5_MACHT1  =&world_data_mrmacht;
aero5_AOAT2   =&world_data_mraoat2;
aero5_CL1     =&world_data_clmr1;
aero5_CL2     =&world_data_clmr2;
aero5_CD1     =&world_data_cdmr1;
aero5_CD2     =&world_data_cdmr2;
aero5_CM1     =&world_data_cmmr1;
aero5_CM2     =&world_data_cmmr2;
```

```
     mass5_MASS    =&world_data_segmass(5);
     mass5_GRAVITY      =&world_data_gravity;

     twst5_ANGLE   =&world_data_segtwst(5);
     twst5_AXIS    =1;

     blseg5_LEN    =&world_data_seglen(5);

     tip_POS  =&world_data_tippos(1:3,1);

     feat_AXIS     =1;

     hof1_LEN      =&world_data_ho;

     rotr1_ANGLE   =3*world_data_pi/2;
     rotr1_AXIS    =3;

     flap_AXIS     =2;

     Lead_AXIS     =3;
     Lead_KK  =0;
     Lead_CC  =&world_data_lagdamper;
     Lead_ANG0     =0;


PARENTG
     mrspeed_AXIS  =3;

     sp_PSI   =&mrspeed_u(1);
     sp_PSID  =&mrspeed_u(2);
     sp_PSIDD      =&mrspeed_u(3);
     sp_A1S   =&world_data_a1s;
     sp_A1SD  =&world_data_a1sd;
     sp_A1SDD      =&world_data_a1sdd;
     sp_B1S   =&world_data_b1s;
     sp_B1SD  =&world_data_b1sd;
     sp_B1SDD      =&world_data_b1sdd;
     sp_NBLADES    =&world_data_nblades;
     sp_PHASE      =&world_data_phase;
     sp_THE   =&world_data_mtheta;
     sp_THED  =&world_data_mthetad;
     sp_THEDD      =&world_data_mthetadd;


PARENTG

GOTO inflow
     Wfi_ALPHA     =&world_data_alfiv;
     Wfi_ARG1      =&world_data_wfaost1;
     Wfi_ARG2      =&world_data_wfaoat1;
     Wfi_ARG3      =&world_data_wfaost2;
     Wfi_ARG4      =&world_data_wfaoat2;
     Wfi_ARG5      =&world_data_wfaost3;
     Wfi_ARG6      =&world_data_wfaoat3;
     Wfi_ARG7      =&world_data_wfchit;
     Wfi_ARG8      =&world_data_wfa1ft;
     Wfi_A1F  =&world_data_tippa(2);
     Wfi_BETA      =&world_data_betiv;
     Wfi_CHI  =&world_data_chimr;
```

```
Wfi_LAM  =&world_data_lam;
Wfi_OMEGA     =&world_data_rpmnom;
Wfi_RMR  =&world_data_rmr;
Wfi_ROW1      =&world_data_wfns1;
Wfi_COL1      =&world_data_wfna1;
Wfi_ROW2      =&world_data_wfns2;
Wfi_COL2      =&world_data_wfna2;
Wfi_ROW3      =&world_data_wfns3;
Wfi_COL3      =&world_data_wfna3;
Wfi_ROW4      =&world_data_wfnchi;
Wfi_COL4      =&world_data_wfna1f;
Wfi_ROW5      =&world_data_wfnchi;
Wfi_COL5      =&world_data_wfna1f;
Wfi_ROW6      =&world_data_wfnchi;
Wfi_COL6      =&world_data_wfna1f;
Wfi_TABLE1    =&world_data_wfqdyn;
Wfi_TABLE2    =&world_data_fvzwf;
Wfi_TABLE3    =&world_data_fvywf;
Wfi_TABLE4    =&world_data_rvxwfi;
Wfi_TABLE5    =&world_data_rvywfi;
Wfi_TABLE6    =&world_data_rvzwfi;
Wfi_TRANS     =&world_data_bodytr;
Wfi_VELIN     =&world_data_bodyvel;
Wfi_VELOUT    =&world_data_wfiv;

Hti_ALPHA     =&world_data_alfiv;
Hti_ARG1      =&world_data_htaost1;
Hti_ARG2      =&world_data_htaoat1;
Hti_ARG3      =&world_data_htaost2;
Hti_ARG4      =&world_data_htaoat2;
Hti_ARG5      =&world_data_htaost3;
Hti_ARG6      =&world_data_htaoat3;
Hti_ARG7      =&world_data_htchit;
Hti_ARG8      =&world_data_hta1ft;
Hti_A1F  =&world_data_tippa(2);
Hti_BETA      =&world_data_betiv;
Hti_CHI  =&world_data_chimr;
Hti_LAM  =&world_data_lam;
Hti_OMEGA     =&world_data_rpmnom;
Hti_RMR  =&world_data_rmr;
Hti_ROW1      =&world_data_htns1;
Hti_COL1      =&world_data_htna1;
Hti_ROW2      =&world_data_htns2;
Hti_COL2      =&world_data_htna2;
Hti_ROW3      =&world_data_htns3;
Hti_COL3      =&world_data_htna3;
Hti_ROW4      =&world_data_htnchi;
Hti_COL4      =&world_data_htna1f;
Hti_ROW5      =&world_data_htnchi;
Hti_COL5      =&world_data_htna1f;
Hti_ROW6      =&world_data_htnchi;
Hti_COL6      =&world_data_htna1f;
Hti_TABLE1    =&world_data_htqdyn;
Hti_TABLE2    =&world_data_fvzht;
Hti_TABLE3    =&world_data_fvyht;
Hti_TABLE4    =&world_data_rvxhti;
Hti_TABLE5    =&world_data_rvyhti;
Hti_TABLE6    =&world_data_rvzhti;
Hti_TRANS     =&world_data_bodytr;
Hti_VELIN     =&world_data_bodyvel;
Hti_VELOUT    =&world_data_htiv;
```

```
Vti_ALPHA     =&world_data_alfiv;
Vti_ARG1      =&world_data_vtaost1;
Vti_ARG2      =&world_data_vtaoat1;
Vti_ARG3      =&world_data_vtaost2;
Vti_ARG4      =&world_data_vtaoat2;
Vti_ARG5      =&world_data_vtaost3;
Vti_ARG6      =&world_data_vtaoat3;
Vti_ARG7      =&world_data_htchit;
Vti_ARG8      =&world_data_hta1ft;
Vti_A1F  =&world_data_tippa(2);
Vti_BETA      =&world_data_betiv;
Vti_CHI  =&world_data_chimr;
Vti_LAM  =&world_data_lam;
Vti_OMEGA     =&world_data_rpmnom;
Vti_RMR  =&world_data_rmr;
Vti_ROW1      =&world_data_vtns1;
Vti_COL1      =&world_data_vtna1;
Vti_ROW2      =&world_data_vtns2;
Vti_COL2      =&world_data_vtna2;
Vti_ROW3      =&world_data_vtns3;
Vti_COL3      =&world_data_vtna3;
Vti_ROW4      =&world_data_htnchi;
Vti_COL4      =&world_data_htna1f;
Vti_ROW5      =&world_data_htnchi;
Vti_COL5      =&world_data_htna1f;
Vti_ROW6      =&world_data_htnchi;
Vti_COL6      =&world_data_htna1f;
Vti_TABLE1    =&world_data_vtqdyn;
Vti_TABLE2    =&world_data_fvzvt;
Vti_TABLE3    =&world_data_fvyvt;
Vti_TABLE4    =&world_data_rvxhti;
Vti_TABLE5    =&world_data_rvyhti;
Vti_TABLE6    =&world_data_rvzhti;
Vti_TRANS     =&world_data_bodytr;
Vti_VELIN     =&world_data_bodyvel;
Vti_VELOUT    =&world_data_vtiv;

Tri_ALPHA     =&world_data_alfiv;
Tri_ARG1      =&world_data_vtaost1;
Tri_ARG2      =&world_data_vtaoat1;
Tri_ARG3      =&world_data_htaost2;
Tri_ARG4      =&world_data_htaoat2;
Tri_ARG5      =&world_data_vtaost3;
Tri_ARG6      =&world_data_vtaoat3;
Tri_ARG7      =&world_data_htchit;
Tri_ARG8      =&world_data_hta1ft;
Tri_A1F  =&world_data_tippa(2);
Tri_BETA      =&world_data_betiv;
Tri_CHI  =&world_data_chimr;
Tri_LAM  =&world_data_lam;
Tri_OMEGA     =&world_data_rpmnom;
Tri_RMR  =&world_data_rmr;
Tri_ROW1      =&world_data_vtns1;
Tri_COL1      =&world_data_vtna1;
Tri_ROW2      =&world_data_htns2;
Tri_COL2      =&world_data_htna2;
Tri_ROW3      =&world_data_vtns3;
Tri_COL3      =&world_data_vtna3;
Tri_ROW4      =&world_data_htnchi;
Tri_COL4      =&world_data_htna1f;
Tri_ROW5      =&world_data_htnchi;
```

```
Tri_COL5        =&world_data_htna1f;
Tri_ROW6        =&world_data_htnchi;
Tri_COL6        =&world_data_htna1f;
Tri_TABLE1      =&world_data_vtqdyn;
Tri_TABLE2      =&world_data_fvzht;
Tri_TABLE3      =&world_data_fvyvt;
Tri_TABLE4      =&world_data_rvxhti;
Tri_TABLE5      =&world_data_rvyhti;
Tri_TABLE6      =&world_data_rvzhti;
Tri_TRANS       =&world_data_bodytr;
Tri_VELIN       =&world_data_bodyvel;
Tri_VELOUT      =&world_data_triv;

u1_ALT   = &world_data_agl;
u1_DWTAU        =&world_data_dwtau;
u1_GEF1  =&world_data_gef1;
u1_GEF2  =&world_data_gef2;
u1_NBLADES      =&world_data_nblades;
u1_NSEG  =&world_data_nseg;
u1_RPMNOM       =&world_data_rpmnom;
u1_RMR   =&world_data_rmr;


PARENTG


PARENTG


PARENTG
```

```
////////////////////////////////////////////////////////////////
// file: psh.prolog
// date: 16 Jul 1992
//
// This script loads the data needed for psh model
////////////////////////////////////////////////////////////////

group data

//  Trim parameters

trimg = 1     "Percentage of trim change to apply on a control update";
nr    = 3     "Number of rotor revolutions between control updates";

//  Useful Constants

pi       = acos(-1)     "Ratio of diameter to circumferance";
d2r      = pi/180.0     "Degrees to radians conversion factor";
r2d      = 180.0/pi     "Radians to degrees conversion factor";
k2f      = 6076.115/3600"Knots to feet per second conversion factor";
f2k      = 3600/6076.115"Feet per second to knots conversion factor";
g        = 32.2;//      "Acceleration due to gravity (fpss)";
gravity  = [0 0 g]      "Inertial gravity vector";
dt       = 0.001  "Integration step size (sec)";
eps      = 5      "Solution convergence criteria on the Q's";
imax       = 20         "Maximun number of convergence iterations";

// Inflow data

gef1   =  0.0625         "Cheeseman Bennett ground effect parameter";
gef2   =  1.0       "Cheeseman Bennett ground effect parameter";
dwtau  =  0.01959   "Inflow time constant (sec)";
agl    = 90         "Alttitude above ground plane (ft)";
chimr  = 0          "Wake skew angle (rad)";
lam    = 0          "Inflow velocity (nd)";
nblades = 4         "Number of rotor blades";
nseg    = 5         "Number of blade segments";
```

```
// C G data

Vweight = 20000          "Total vehicle weight (lbs)";
ixx = 5000.0        "Total moment of inertia about x (sl-ft2)";
iyy = 40000.0            "Total moment of inertia about y (sl-ft2)";
izz = 35000.0            "Total moment of inertia about z (sl-ft2)";
ixy = 0.0           "Total cross product xy (sl-ft2)";
ixz = 0.0           "Total cross product xz (sl-ft2)";
iyz = 0.0           "Total cross product yz (sl-ft2)";

// Rotor data
lagdamper = 4000 "Lag Damping Coefficient (lbs sec /rad)";
imr = 0 "Longitudinal Shaft Tilt + Forward (rad)";
mrloc = [-0.5 0 -7.5]    "Main Rotor Location (ft)";
rpmnom  = 21.6667 "Nominal main rotor speed (r/s)";
rmr  = 30.0         "Main rotor radius (ft)";
naz      = 24       "Number of azimuth steps/rev";
dpsi     = 360/naz "Change in azimuth angle/integration step (deg)";

dt       = d2r*dpsi/rpmnom; // Integration step size

tippos   = zeros(3,nblades)"Inertial positions of blade tips (ft)";
tippa    = zeros(3,1)    "Tip path plane angles a0 a1f b1f (rad)";



//   Load main rotor blade segment aero tables
//

//  Rotor
rtable1= read("c_mrot1.tab");

clmr1(1:25,1:1) = rtable1(:,1)..
"Main rotor blade segment cl(alpha,mach) table (nd): high res/low
angle";
cdmr1(1:25,1:1) = rtable1(:,2) ..
"Main rotor blade segment cd(alpha,mach) table (nd): high res/low
angle";
cmmr1(1:25,1:1) = rtable1(:,3) ..
"Main rotor blade segment cm(alpha,mach) table (nd): high res/low
angle";

clear(rtable1);

rtable2= read("c_mrot2.tab");
clmr2    = rtable2(:,1) ..
"Main rotor blade segment cl(alpha) table (nd): low res/high angle";
cdmr2    = rtable2(:,2) ..
"Main rotor blade segment cd(alpha) table (nd): low res/high angle";
cmmr2    = rtable2(:,3) ..
"Main rotor blade segment cm(alpha) table (nd): low res/high angle";
clear(rtable2);

mrbp = 30.0*d2r ..
"Main rotor blade segment angle of attack transition angle (rad)";
mraoat1 = [d2r*[-30   30 2.5]  25] ..
"Main rotor blade segment angle of attack breakpoint table for high res
tables";
mraoat2 = [d2r*[-180 180 5] 73] ..
```

```
"Main rotor blade segment angle of attack breakpoint table for low res
tables";
mrmacht = [0 0 1 1] ..
"Main rotor blade segment Mach number breakpoint table for high res
tables";
mrna1   = mraoat1(4)    " of rows in main rotor blade segment high res
tables";
mrna2   = mraoat2(4)    " of rows in main rotor blade segment low res
tables";
mrnm1   = mrmacht(4)    " of cols in main rotor blade segment high res
tables";

exec("blade_seg_geom",1);// Compute the blade segment geometry

// C G reference Data (DMASS)

fscg = 296.0       "Fuselage Station of cg (in)";
blcg = 0.0         "Buttline Station of cg (in)";
wlcg = 113.0       "Waterline Station of cg (in)";

xcgf = 0  "Rigid body fuselage  station offset (ft)";
ycgf = 0  "Rigid body buttline  station offset (ft)";
zcgf = 0  "Rigid body waterline station offset (ft)";

cgloc = [xcgf ycgf zcgf] "Inertial cg offset (ft)";


fmass     = (Vweight - nblades*bw)/g"Mass of the fuselage (sl)";
finertia  = [ixx ixy ixz
             ixy iyy iyz
             ixz iyz izz] "Inertia matrix of the fuselage (sl-ft2)";

// Aero Wing/Fuselage data  (AERO3DS and INTERFER)

fswf = 302.0 "Fuselage  station of wing/fuselage (in)";
blwf =   0.0 "Buttline  station of wing/fuselage (in)";
wlwf = 119.0 "Waterline station of wing/fuselage (in)";

fswfft = (fscg - fswf)/12.0;
blwfft = (blcg - blwf)/12.0;
wlwfft = (wlcg - wlwf)/12.0;

wfloc=[fswfft blwfft wlwfft] "Aero fuselage location (ft)";

exec("c_wfaero.exc",1);     // Fuselage Aerodynamics

wabp = 25.0*d2r   "Wing/fuselage angle of attack transition angle
(rad)";
wbbp = 25.0*d2r   "Wing/fuselage angle of sideslip transition angle
(rad)";

alfwf  = 0    "Wing/fuselage angle of attack (rad)";
betwf  = 0    "Wing/fuselage sideslip angle (rad)";
alfiv  = 0    "Angle of attack for the interference effects (rad)";
betiv  = 0    "Sideslip angle for the interference effects (rad)";

wfiv = zeros(3,1) "Interference velocities on the wing/fuselage (fps)";
exec("wf_intf.exc",1); // W/F interference

// Horizontal Stabilizer (AERO2D3D and INTERFER)
```

```
fsht = 692.0 "Fuselage  station of horizontal stabilizer (in)";
blht =   0.0 "Buttline  station of horizontal stabilizer (in)";
wlht = 95.0  "Waterline station of horizontal stabilizer (in)";

fshtft = (fscg - fsht)/12.0;
blhtft = (blcg - blht)/12.0;
wlhtft = (wlcg - wlht)/12.0;

htloc = [fshtft blhtft wlhtft] "Horizontal tail location (ft)";

htincang = -0.052 "horizontal incidence angle + up (rad)";
htable1 = read("c_htail1.tab");

clh1(1:13,1:2)  =  htable1(:,1) ..
"Horizontal stabilizer cl(alpha,mach) table (nd): high res/low angle";
cdh1(1:13,1:2)  =  htable1(:,2) ..
"Horizontal stabilizer cd(alpha,mach) table (nd): high res/low angle";
cmh1(1:13,1:2)  =  0.0*ones(26,1) ..
"Horizontal stabilizer cm(alpha,mach) table (nd): high res/low angle";

clear(htable1);

htable2 = read("c_htail2.tab");

clh2   =  htable2(:,1) ..
"Horizontal stabilizer cl(alpha) table (nd): low res/high angle";
cdh2   =  htable2(:,2) ..
"Horizontal stabilizer cd(alpha) table (nd): low res/high angle";
cmh2   =  0.0*ones(19,1) ..
"Horizontal stabilizer cm(alpha) table (nd): low res/high angle";

clear(htable2);

hbp = 30.0*d2r "Horizontal stabilizer angle of attack transition angle
(rad)";

haoat1 = [d2r*[-30 30  5]  13] ..
"Horizontal stabilizer angle of attack breakpoint table for high res
tables";

haoat2 = [d2r*[-90 90 10]  19] ..
"Horizontal angle of attack breakpoint table for low res tables";

hmacht = [0 1 1 2] ..
"Horizontal stabilizer Mach number breakpoint table for high res
tables";

hna1   = haoat1(4);     // # of rows in clh1,cdh1,cmh1
hna2   = haoat2(4);     // # of rows in clh2,cdh2,cmh2
hnm1   = hmacht(4);     // # of cols in clh1,cdh1,cmh1

hchord = 2   "Chord length of horizontal stabilizer (ft)";
hlen   = 9   "Span of horizontal stabilizer (ft)";
hdefic = 1   "Lift deficiency factor (nd)";
htiv = zeros(3,1) "Interference velocities on the horizontal stabilizer
(fps)";
exec("ht_intf.exc",1); // H tail interference

// Vertical Tail (AER2D3D and INTERFER)
```

```
fsvt = 716.0;           // Fuselage Station of Vertical Tail
blvt = 0.0;       // Buttline Station of Vertical Tail
wlvt = 149.0;           // Waterline Station of Vertical Tail

fsvtft = (fscg - fsvt)/12.0;
blvtft = (blcg -blvt)/12.0;
wlvtft = (wlcg -wlvt)/12.0;

vtloc = [fsvtft blvtft wlvtft] "Vertical Tail location (ft)";
vtable1= read("c_vtail1.tab");

clv1(1:13,1:2)   =  vtable1(:,1) ..
"Vertical tail cl(alpha,mach) table (nd): high res/low angle";
cdv1(1:13,1:2)   =  vtable1(:,2) ..
"Vertical tail cd(alpha,mach) table (nd): high res/low angle";
cmv1(1:13,1:2)   =  0.0*ones(26,1) ..
"Vertical tail cm(alpha,mach) table (nd): high res/low angle";

clear(vtable1);

vtable2= read("c_vtail2.tab");

clv2   =  vtable2(:,1) ..
"Vertical tail cl(alpha) table (nd): low res/high angle";
cdv2   =  vtable2(:,2) ..
"Vertical tail cd(alpha) table (nd): low res/high angle";
cmv2   =  0.0*ones(19,1) ..
"Vertical tail cm(alpha) table (nd): low res/high angle";

clear(vtable2);

vbp = 30.0*d2r;         // Transition AoA fro clv1 to clv2
vaoat1 = [d2r*[-30 30  5]  13];// AoA break pts for clv1,cdv1,cmv1
vaoat2 = [d2r*[-90 90 10]  19]; // AoA break pts for clv2,cdv2,cmv2
vmacht = [0 1 1 2];     // Mach break pts for clv1,cdv1,cmv1
vna1   = vaoat1(4);     // # of rows in clv1,cdv1,cmv1
vna2   = vaoat2(4);     // # of rows in clv2,cdv2,cmv2
vnm1   = vmacht(4);     // # of cols in clv1,cdv1,cmv1

vchord = 33/7.7;        // Chord length of v tail
vlen   = 7.7;           // Width of the v tail
vdefic = 1;             // No lift deficiency

vtiv   = zeros(3,1);
exec("vt_intf.exc",1); // No V tail interference

// Bailer Tail Trotor (BAILEY)

fstr = 740.0;           // Fuselage Station of tail rotor
bltr = 24.0;      // Buttline Station of tail rotor
wltr = 185.0;           // Waterline Station of tail rotor

fstrft = (fscg - fstr)/12.0;
bltrft = (blcg -bltr)/12.0;
wltrft = (wlcg -wltr)/12.0;

trloc = [fstrft bltrft wltrft] "tail rotor location (ft)";

atr    = 5.73;          // Lift curve slope
ttr    = 1.00;          // Tail rotor thrust
cdtr   = 0.0;           // Rotor head drag coefficient
```

```
d0tr   = 0.0087;  // Taylor series drag coeff.
d1tr   =-0.0216;  // Taylor series drag coeff.
d2tr   = 0.4000;  // Taylor series drag coeff.
biastr = 14.0;          // Blade pitch bias
trblades = 3;           // Number of blades
btltr  = 0.92;          // Blade tip loss
bvttr  = 1.0;           // Blockage effect parameter
bvt1tr = 1.0;           // Blockage effect parameter
chordtr= 1;       // Blade chord
delttr = 0.001455;      // Partial of coning wrt thrust
omegatr= 100;           // Tail rotor speed
rtr    = 6.5;           // Tail rotor radius
thettr = 0.0;           // Tail rotor collective pitch
twsttr = -5;            // Tail rotor blade twist
td3tr  = -0.5774; // Tan of delta 3 angle
vbvttr = 30*k2f;  // Blockage velocity parameter
xibtr  = 0.67;          // Mass moment of inertia
xlamtr = 1.0;           // Tail rotor inflow (inital value)

triv   = zeros(3,1);
exec("tr_intf.exc",1);// No Tail rotor interference

// Atmosphere data  (ATMOS)

atmtab   = read("atmo.tab");// Get ARDC62 atmosphere tables

densityt =  atmtab(:,1);// Air density as function of altitude
ssoundt  =  atmtab(:,2);// Speed of sound as f(altitude)
natmo    = prod(size(densityt));// Size of data tables
altt     = [0 240000 2000 121]';// Alitude break points
clear(atmtab);

wind = zeros(3,1);      // No wind
bodytr = eye(3);
pos = [0 0 -90];
bodyvel = [10 0 0]';




//  Data for the UH60 control system
//
// Control system data

     mtheta   = 17.25*d2r;// main rotor pitch (rad)
     mthetad  = 0;       // main rotor pitch (rad)
     mthetadd = 0;       // main rotor pitch (rad)
     a1s      = -1.1*d2r;// lateral cyclic   (rad)
     a1sd     = 0;       // lateral cyclic   (rad)
     a1sdd    = 0;       // lateral cyclic   (rad)
     b1s      = -0.78*d2r;// long cyclic      (rad)
     b1sd     = 0;       // long cyclic      (rad)
     b1sdd    = 0;       // long cyclic      (rad)
        phase    =-4.0*d2r;// Swash plate phase angle


//  Pilot Controls, Trim Signals, Fps and Sas

Xa    = 0.0;
Xb    = 0.0;
```

```
Xc     = 0.0;
Xp     = 0.0;

Xatrm = 4.95;
Xbtrm = 1.196;
Xctrm = 10.817;
Xptrm = 1.925;

// Control bias inputs

A1sbias = -10.625;
B1sbias = -10.0;
Th0bias = 1.25;
Thrbias = 31.875;

// Swashplate test inputs

A1schk = 0.0;
B1schk = 0.0;
Th0chk = 0.0;
Thrchk = 0.0;

// Swashplate limits
A1sll = -11.0;
A1sul = 7.0;
B1sll = -10.0;
B1sul = 20.0;
Th0ll = 1.25;
Th0ul = 19.0;
Thrll = -15.0;
Thrul = 36.875;

//  Stick to swashplate scale factor

kxaa1s = 17.5/9      "Lateral pitch degrees per inch control movement";
kxbb1s = 30/10       "Longitudinal pitch degrees per inch control
movement";
kxcth0 =17.75/12     "Collective pitch degrees per inch control
movement";
kxpthr =-46.875/5.5 "Tail rotor pitch degrees per inch control
movement";


// Pilot input stops

xcll = (Th0ll-Th0bias)/kxcth0    "Collective stick lower stop (in)";
xcul = (Th0ul-Th0bias)/kxcth0    "Collective stick upper stop (in)";
xall = (A1sll-A1sbias)/kxaa1s    "Lateral cyclic stick lower stop (in)";
xaul = (A1sul-A1sbias)/kxaa1s    "Lateral cyclic stick upper stop (in)";
xbll = (B1sll-B1sbias)/kxbb1s    "Long.   cyclic stick lower stop (in)";
xbul = (B1sul-B1sbias)/kxbb1s    "Long.   cyclic stick upper stop (in)";
xpll = (Thrul-Thrbias)/kxpthr    "Pedal lower stop (in)";
xpul = (Thrll-Thrbias)/kxpthr    "Pedal upper stop (in)";

parentg

//
// End of psh.prolog data script
```

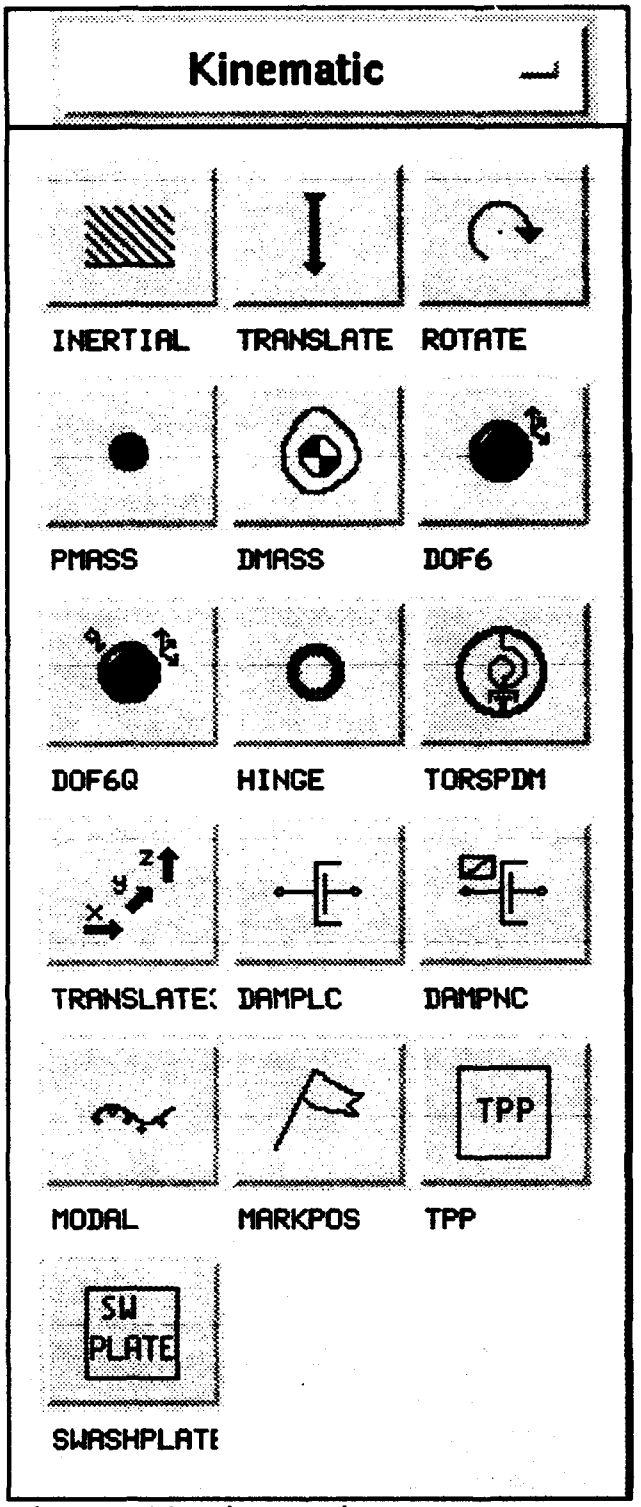

Figure 22 Kinematic Components

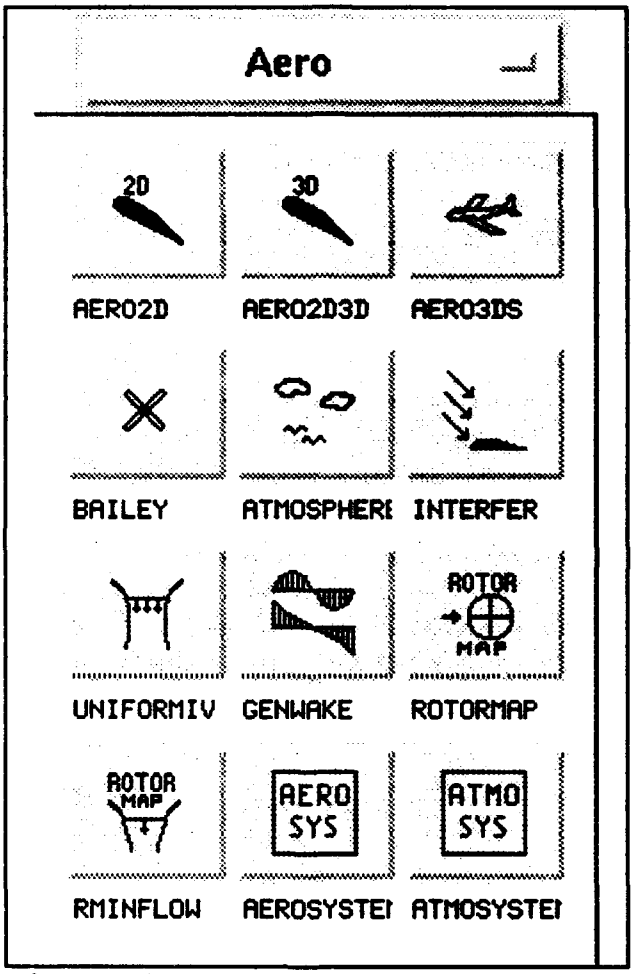

Figure 23 Aero. Components

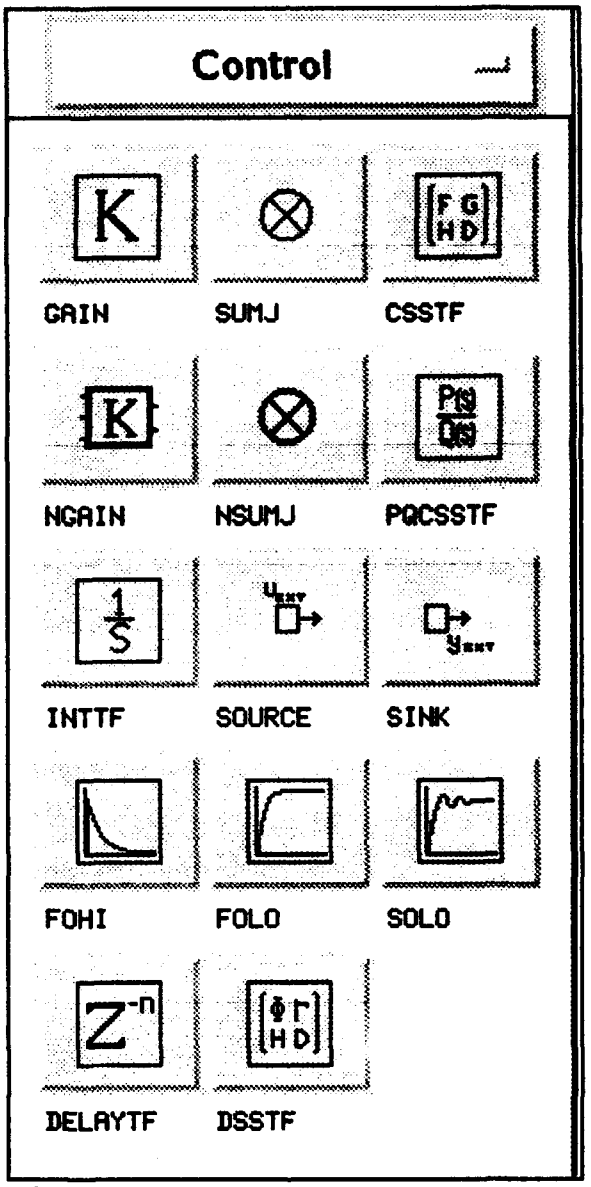

Figure 24 Control Components

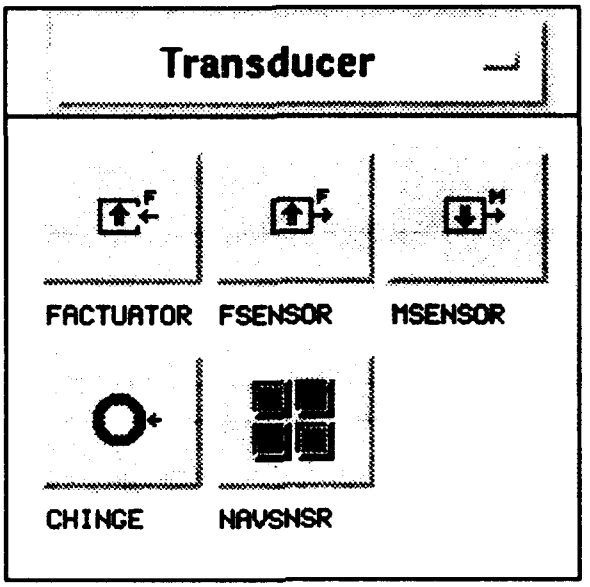

Figure 25 Transducer Comp.

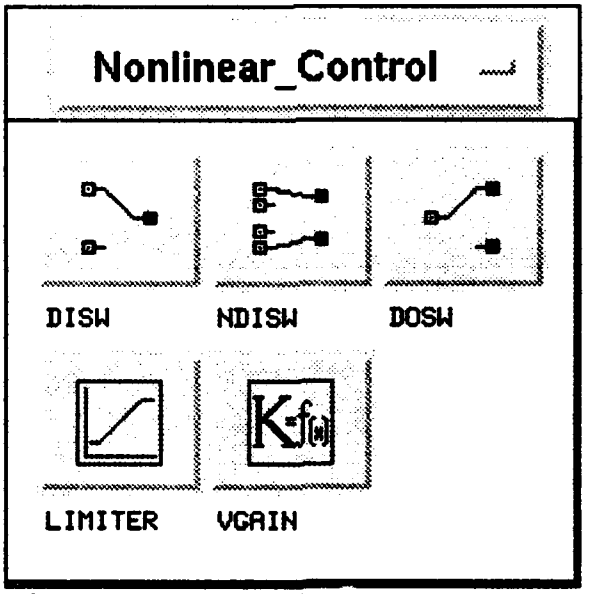

Figure 26 Non linear Control Components

APPENDIX B

```
////////////////////////////////////////////////////////////////////////
//  file: psh.def                                                     //
//  Definition file for psh with 4 rigid blades and                   //
//  five segments and the primary flight control system               //
////////////////////////////////////////////////////////////////////////

//  Set path to directories where files are located
path("analysis/")
path("functions/")
path("data-tables/");
path("operations/")
path("system/")
path("solution/")
path("rotors/")

// load the functions used
exec("cycle.fun",1);     // Load the cycle function
exec("odli.fun",1);      // Load the 1d linearinzation function

// Model construction:
goto world;         // Go to the top group of the scope program

exec("psh.prolog",1)    // Load the data file for the psh
exec("psh.exc",1)       // Load the model file for the psh
exec("psh.epilog",1)    // equivalence data for solution and system
components
exec("system.exc",1);           // Create and connect the system
component
exec("solution.exc",1);         // Setup solution

init;               // Links, equivalences, and analyzes data-flow

exec("mbc.exc",1);       //  Setup a MBC transformation

exec("configure.exc",1) // Configure for reporting and display

world_data_omega = 650/30  // Set main rotor speed

init;                     // Relink since we equivalenced fields in
configure.exc

world::setup        // One time internal initialization and method setup

world::reset        // Set i.c. for states/derivatives and invokes model

exec("assemble.exc");    //  setup the model for running

//// end of psh.def ////////
```

```
////////////////////////////////////////////////////////////////////////////
//  file: psh.epilog                                                      //
//  Epilog file for psh rigid blade simulation                            //
////////////////////////////////////////////////////////////////////////////

world_data_alfwf = &world_model_heli_body_wf_alpha;
world_data_betwf = &world_model_heli_body_wf_beta;
world_data_wfiv  = &world_model_heli_body_wf_dwash;
world_data_triv  = &world_model_heli_body_trotor_dwash;
world_data_vtiv  = &world_model_heli_body_vstab_dwash;
world_data_htiv  = &world_model_heli_body_hstab_dwash;

world_data_bodytr  = &world_model_heli_body_bodydof_cfrt;
world_data_bodyvel = &world_model_heli_body_bodydof_cfrl(4:6);
world_data_pos     = &world_model_heli_body_bodydof_xi(1:3);
world_data_pos = [0 0 -90];

world_data_chimr   = &world_model_heli_inflow_u1_chi;
world_data_lam     = &world_model_heli_inflow_u1_x;

world_data_tippa   = &world_model_heli_rotor_tppc_tippp;


world_model_drivetrain_rai_xic = world_data_rpmnom;
world_model_heli_rotor_mrspeed_u(2)=world_data_rpmnom;

world_model_heli_body_bodydof_xiic(1:3) = world_data_pos;

world_model_heli_body_bodydof_xicf=ones(6,1);
world_model_heli_body_bodydof_xcf=ones(6,1);
world_model_heli_body_bodydof_xdcf=ones(6,1);

world_model_heli_inflow_u1_xic = 0.05;
world_model_heli_inflow_u1_x   = 0.05;

//  Control connection

init(model_cont);

world_data_alfiv   = &world_model_cont_sensors_alpha_y;
world_data_betiv   = &world_model_cont_sensors_beta_y;

world_data_b1s     = &world_model_cont_lng_B1s_y;
world_data_a1s     = &world_model_cont_lat_A1s_y;
world_data_mtheta  = &world_model_cont_coll_mtheta_y;
world_data_thettr  = &world_model_cont_dir_Thettr_y;


//  Motion sensor gain matrix

world_model_heli_body_dofsensor_k(1,10)=1;
world_model_heli_body_dofsensor_k(2,11)=1;
world_model_heli_body_dofsensor_k(3,12)=1;
world_model_heli_body_dofsensor_k(4,13)=1;
world_model_heli_body_dofsensor_k(5,14)=1;
world_model_heli_body_dofsensor_k(6,15)=1;

// End of uh60.epilog
```

```
////////////////////////////////////////////////////////////////////////
//  file: system.exc                                                  //
//  This script creates and connects the system component             //
////////////////////////////////////////////////////////////////////////

goto world

system  sys
connect sys to world

init;

// End of system.exc script

////////////////////////////////////////////////////////////////////
//  file: solution.exc                                            //
//  This file creates the solution configuration for psh          //
//  date: 19 July 1993                                            //
////////////////////////////////////////////////////////////////////

pushg(world);


     HSOLVE  helisolve;
          helisolve_dt     =& world_data_dt;
                helisolve_delta =  1/2**14;
                helisolve_epsilon=& world_data_eps;
                helisolve_maxiter=& world_data_imax;
                helisolve_reassemble=  1;

     CSOLVE  drivesolve;
          drivesolve_dt    =& world_data_dt;
                drivesolve_delta=  1/2**14;
                drivesolve_epsilon=& world_data_eps;
                drivesolve_maxiter=& world_data_imax;
                drivesolve_reassemble=  0;
                drivesolve_ycf  =  0;

     CSOLVE  contsolve
          contsolve_dt     =& world_data_dt;
                contsolve_delta =  1/2**14;
                contsolve_epsilon=& world_data_eps;
                contsolve_maxiter=& world_data_imax;
                contsolve_reassemble=  0;
                contsolve_ycf    =  0;

        HSOLVE  rotorsolve;
                rotorsolve_dt    =& world_data_dt;
                rotorsolve_delta=  1/2**14;
                rotorsolve_epsilon=& world_data_eps;
                rotorsolve_maxiter=& world_data_imax;
                rotorsolve_reassemble=  1;

     HSOLVE  rhsolve;
          rhsolve_dt       =& world_data_dt;
                rhsolve_delta    =  1/2**14;
                rhsolve_epsilon  =& world_data_eps;
                rhsolve_maxiter  =& world_data_imax;
                rhsolve_reassemble=  1;
```

```
      FSOLVE  topsolve;
          topsolve_dt      =& world_data_dt;
                 topsolve_delta   =  1/2**14;
                 topsolve_epsilon=& world_data_eps;
                 topsolve_maxiter=& world_data_imax;
                 topsolve_reassemble=  0;
                 topsolve_ycf     =  0;

      // Low-level solution groups:

      connect contsolve to model_cont;
      connect contsolve to model_heli_body_dofsensor;

      connect drivesolve to model_drivetrain;

      connect helisolve to model_heli;

         connect rotorsolve to model_heli_rotor;
         connect rotorsolve to model_heli_inflow_u1;

      // Mid-level: RHSOLVE combines ROTOR and HELI groups
      connect rhsolve to helisolve;
      connect rhsolve to rotorsolve;

      // Top-level:
      connect topsolve to drivesolve;
      connect topsolve to rhsolve;
      connect topsolve to contsolve;

      // Default architecture:
      // Iterate between RH, DRIVE, and CONT solution groups:
      topsolve_cf   = 1;
      rhsolve_cf    = 0;

      // These must always be zero, as they are bottom-level groups:
      drivesolve_cf      = 0;
      contsolve_cf = 0;
      helisolve_cf = 0;
      rotorsolve_cf      = 0;

popg

exec("load_operations.exc",1)

// End of solution.exc

////////////////////////////////////////////////////////////////
// file: load_operations.exc                                  //
// This file loads the simulation opns that propagate the model//
// date: 22 July 1993                                         //
////////////////////////////////////////////////////////////////

//  Load the simulation operations

exec("step.op",1)
exec("steprotor.op",1)
exec("stepnc.op",1)

// End of load_operations.exc
```

```
////////////////////////////////////////////////
// File:  step.op
// Author:   Joe English
// Created:  3 May 1993
////////////////////////////////////////////////

operation step
//
// STEP operation for helicopter model:
// Propagate forward one time step and solve for the new states,
// using CONTSOLVE, DRIVESOLVE, and RHSOLVE.
//

    // Integrate everything:
    [contsolve::integ, rhsolve::integ, drivesolve::integ];

    // Drivetrain and control system outputs:
    [drivesolve::geny, contsolve::geny];

    // propagate the control system again so the nonlinear components
converge:
    world_contsolve::genq;
    world_contsolve::geny;

    // Rotor / helicopter:
    [rhsolve::genq, rhsolve::geny]

    // convergence iterations on the helicopter:
    repeat 2,
     rhsolve::solve;
     rhsolve::genq;
    end
end

// End of step.op

////////////////////////////////////////////
// File:  steprotor.op
// Author:   Joe English
// Created:  3 May 1993
////////////////////////////////////////////
operation steprotor
//
// STEPROTOR operation for helicopter
// Runs the rotor alone (just drivetrain and rotorsolve)
// Iterates on ROTORSOLVE twice
    // Integrate everything:
    [drivesolve::integ, rotorsolve::integ];
    [drivesolve::geny];
    [helisolve::genq];

    // Rotor / helicopter:
    [rotorsolve::genq, rotorsolve::geny]

    // two convergence iterations on the rotor is enough
    repeat 2,
     rotorsolve::solve;
     rotorsolve::genq;
    end
end
// End of steprotor.op
```

```
////////////////////////////
// File:  stepnc.op
// Author:    Joe English
// Created:  3 May 1993
////////////////////////////

operation stepnc
//
// STEPNC operation for helicopter model:
// Propagate everything but the control system
// using DRIVESOLVE and RHSOLVE.
//

    // Integrate everything:
    [rhsolve::integ, drivesolve::integ];

    // Drivetrain outputs:
    [drivesolve::geny, contsolve::geny];

    // Rotor / helicopter:
    [rhsolve::genq, rhsolve::geny]

    // convergence iterations on the helicopter:
    repeat  2,
     rhsolve::solve;
     rhsolve::genq;
    end
end

// End of stepnc.op

////////////////////////////////////////////////////////
//  file: mbc.exc
//  date: 21 July 1993
//  This script sets up a multi blade coordinate
//  transformation of the rotor states to improve the speed and
//  accuracy of a rotating blade states for the
//  psh rigid blade element model
////////////////////////////////////////////////////////
pushg(world_model_heli_rotor);

describe("MBC: Rigid blade element")

nblades   =& world_data_nblades;
ncomp = 2;    // flapping, lead-lag
ndof = 1;

mbcxfrm mbc 1,nblades*ncomp;
     mbc_nblades   = &nblades;
     mbc_ncomp     = ncomp;
     mbc_ndof      = ndof;
     mbc_psi       = &mrspeed_u(1);
     mbc_omega     = &mrspeed_u(2);
     mbc_mbcf      = 0; // leave off initially
j=1;
for i=1:nblades
     connect mbc(j) to blade$i_flap; j=j+1;
     connect mbc(j) to blade$i_lead; j=j+1;
end
popg;
// End of mbc.exc
```

```
////////////////////////////////////////////////////
//  file: configure.exc
//  date: 2 September 1993
//  This file configures the parameter group by selectively
//  addressing relevant model and data information
////////////////////////////////////////////////////

pushg(world); group results; popg
pushg(world); group cpg;


g    = &world_data_g;
d2r  = &world_data_d2r;
r2d  = &world_data_r2d;
k2f  = &world_data_k2f;
f2k  = &world_data_f2k;
pi   = &world_data_pi;
naz  = world_data_naz;

// Rigid body data

fmass    = &world_data_fmass;
finertia = &world_data_finertia;
rbpfrl   = &world_model_heli_body_body_pfrl(1:3);
rblf     = &world_model_heli_body_body_pfo;
trb      = &world_model_heli_body_body_pfrt;

//  Aircraft Center of Gravity

pos  = &world_model_heli_body_bodydof_xi(1:3);
ean  = &world_model_heli_body_bodydof_xi(4:6);
ivel = &world_model_heli_body_bodydof_xid(1:3);
eand = &world_model_heli_body_bodydof_xid(4:6);
bvel = &world_model_heli_body_bodydof_xd(1:3);
brat = &world_model_heli_body_bodydof_xd(4:6);
bacc = &world_model_heli_body_bodydof_xdd;
bata = &world_model_heli_body_bodydof_xdd(1:3);
bara = &world_model_heli_body_bodydof_xdd(4:6);
tran = &world_model_heli_body_bodydof_cfrt;

ixx  = &world_data_ixx;
iyy  = &world_data_iyy;
izz  = &world_data_izz;
ixz  = &world_data_ixz;
ixy  = &world_data_ixy;
iyz  = &world_data_iyz;

// Atmospheric data

tamb = &world_model_heli_atmos_tamb;
rho  = &world_model_heli_atmos_rho;
rho0 = world_data_densityt(1);

// Aircraft station data

mrloc = &world_data_mrloc;
cgloc = &world_data_cgloc;
wfloc = &world_data_wfloc;
htloc = &world_data_htloc;
trloc = &world_data_trloc;
vtloc = &world_data_vtloc;
```

```
mrpfrl = &world_model_heli_rotor_hub_cfrl(1:3);
wfpfrl = &world_model_heli_body_wf_pfrl(1:3);
htpfrl = &world_model_heli_body_hstab_pfrl(1:3);
vtpfrl = &world_model_heli_body_vstab_pfrl(1:3);
trpfrl = &world_model_heli_body_trotor_pfrl(1:3);
cgpfrl = &world_model_heli_body_bodydof_cfrl(1:3);
rbpfrl = &world_model_heli_body_body_pfrl(1:3);


//  Data for inflow and tip path plane data

tippa = &world_data_tippa;
chimr = &world_data_chimr;
dwash = &world_model_heli_inflow_u1_x;
mu    = &world_model_heli_inflow_u1_mu;
cta   = &world_model_heli_inflow_u1_ct;
gndef = &world_model_heli_inflow_u1_gndef;
ahubf = &world_model_heli_inflow_u1_faa(1:3);
ahubm = &world_model_heli_inflow_u1_faa(4:6);

psimr  = &world_model_heli_rotor_mrspeed_u(1);
omega  = &world_model_heli_rotor_mrspeed_u(2);
omegad = &world_model_heli_rotor_mrspeed_u(3);

hublf = &world_model_heli_rotor_hub_cfo;
thubf = &world_model_heli_rotor_hub_cfo(1:3);
thubm = &world_model_heli_rotor_hub_cfo(4:6);
hbtr  = &world_model_heli_rotor_hub_cfrt;


// Fuselage aerodynamic data

wfcv   = &world_model_heli_body_wf_pfrl(4:6);
wfiv   = &world_data_wfiv;
wftv   = &world_model_heli_body_wf_vel;
wfalfa = &world_model_heli_body_wf_alpha;
wfbeta = &world_model_heli_body_wf_beta;
wfqdyn = &world_model_heli_body_wf_qdyn;
cdwf   = &world_model_heli_body_wf_cd;
cywf   = &world_model_heli_body_wf_cy;
clwf   = &world_model_heli_body_wf_cl;
crwf   = &world_model_heli_body_wf_cr;
cmwf   = &world_model_heli_body_wf_cm;
cnwf   = &world_model_heli_body_wf_cn;
wflf   = &world_model_heli_body_wf_pfo;
wffor  = &world_model_heli_body_wf_pfo(1:3);
wfmom  = &world_model_heli_body_wf_pfo(4:6);
wfif   = &world_model_heli_body_wf_faero;
wfifor = &world_model_heli_body_wf_faero(1:3);
wfimom = &world_model_heli_body_wf_faero(4:6);
twf    = &world_model_heli_body_wf_pfrt;
```

```
// Horizontal tail aerodynamic data

htcv   = &world_model_heli_body_hstab_pfrl(4:6);
htiv   = &world_data_htiv;
httv   = &world_model_heli_body_hstab_vel;
htalfa = &world_model_heli_body_hstab_alpha;
htbeta = &world_model_heli_body_hstab_beta;
htqdyn = &world_model_heli_body_hstab_qdyn;
cdht   = &world_model_heli_body_hstab_cd;
clht   = &world_model_heli_body_hstab_cl;
cmht   = &world_model_heli_body_hstab_cm;
htlf   = &world_model_heli_body_hstab_pfo;
htfor  = &world_model_heli_body_hstab_pfo(1:3);
htmom  = &world_model_heli_body_hstab_pfo(4:6);
htif   = &world_model_heli_body_hstab_faero;
htifor = &world_model_heli_body_hstab_faero(1:3);
htimom = &world_model_heli_body_hstab_faero(4:6);
tht    = &world_model_heli_body_hstab_pfrt;

// Vertical tail aerodynamic data

vtcv   = &world_model_heli_body_vstab_pfrl(4:6);
vtiv   = &world_data_vtiv;
vttv   = &world_model_heli_body_vstab_vel;
vtalfa = &world_model_heli_body_vstab_alpha;
vtbeta = &world_model_heli_body_vstab_beta;
vtqdyn = &world_model_heli_body_vstab_qdyn;
cdvt   = &world_model_heli_body_vstab_cd;
clvt   = &world_model_heli_body_vstab_cl;
cmvt   = &world_model_heli_body_vstab_cm;
vtlf   = &world_model_heli_body_vstab_pfo;
vtfor  = &world_model_heli_body_vstab_pfo(1:3);
vtmom  = &world_model_heli_body_vstab_pfo(4:6);
vtif   = &world_model_heli_body_vstab_faero;
vtifor = &world_model_heli_body_vstab_faero(1:3);
vtimom = &world_model_heli_body_vstab_faero(4:6);
tvt    = &world_model_heli_body_vstab_pfrt;

// Tail rotor data

trthr  = &world_model_heli_body_trotor_t;
trtorq = &world_model_heli_body_trotor_q;
trcv   = &world_model_heli_body_trotor_pfrl(4:6);
triv   = &world_data_triv;
trtv   = &world_model_heli_body_trotor_vel;
trlam  = &world_model_heli_body_trotor_xlam;
tromega= &world_model_heli_body_trotor_omeg;
trrad  = &world_model_heli_body_trotor_r;
trlf   = &world_model_heli_body_trotor_pfo;
trfor  = &world_model_heli_body_trotor_pfo(1:3);
trmom  = &world_model_heli_body_trotor_pfo(4:6);
trif   = &world_model_heli_body_trotor_faero;
trifor = &world_model_heli_body_trotor_faero(1:3);
trimom = &world_model_heli_body_trotor_faero(4:6);
trtr   = &world_model_heli_body_trotor_pfrt;
```

```
// Rotor data which includes inflow, tip path plane and hub
for i=1:world_data_nblades,
   flap$i     = &world_model_heli_rotor_blade$i_flap_x;
   flapd$i    = &world_model_heli_rotor_blade$i_flap_xd;
   flapdd$i   = &world_model_heli_rotor_blade$i_flap_xdd;
   lag$i      = &world_model_heli_rotor_blade$i_lead_x;
   lagd$i     = &world_model_heli_rotor_blade$i_lead_xd;
   lagdd$i    = &world_model_heli_rotor_blade$i_lead_xdd;
   feat$i     = &world_model_heli_rotor_blade$i_feat_u(1);
   featd$i    = &world_model_heli_rotor_blade$i_feat_u(2);
   featdd$i   = &world_model_heli_rotor_blade$i_feat_u(3);
end
i = 1;
for j = 1:world_data_nseg,
   alfmr$i$j   = &world_model_heli_rotor_blade$i_aero$j_alpha;
   mnmr$i$j    = &world_model_heli_rotor_blade$i_aero$j_mach;
   clmr$i$j    = &world_model_heli_rotor_blade$i_aero$j_cl;
   cdmr$i$j    = &world_model_heli_rotor_blade$i_aero$j_cd;
   cmmr$i$j    = &world_model_heli_rotor_blade$i_aero$j_cm;
   qdyn$i$j    = &world_model_heli_rotor_blade$i_aero$j_qdyn;
   tv$i$j      = &world_model_heli_rotor_blade$i_aero$j_vel;
   iv$i$j      = &world_model_heli_rotor_blade$i_aero$j_dwash;
   cv$i$j      = &world_model_heli_rotor_blade$i_aero$j_pfrl(4:6);
   cfor$i$j    = &world_model_heli_rotor_blade$i_aero$j_pfo(1:3);
   cmom$i$j    = &world_model_heli_rotor_blade$i_aero$j_pfo(4:6);
   ifor$i$j    = &world_model_heli_rotor_blade$i_aero$j_faero(1:3);
   imom$i$j    = &world_model_heli_rotor_blade$i_aero$j_faero(4:6);
end
// Control system outputs
xatrm= &world_data_xatrm;
xbtrm= &world_data_xbtrm;
xctrm= &world_data_xctrm;
xptrm= &world_data_xptrm;

b1s  = &world_model_cont_lng_b1s_y;
a1s  = &world_model_cont_lat_a1s_y;
mth  = &world_model_cont_coll_mtheta_y;
tth  = &world_model_cont_dir_thettr_y;
iht  = &world_data_htincang;
xa       = &world_data_xa;
xb       = &world_data_xb;
xc       = &world_data_xc;
xp       = &world_data_xp;

lower = zeros(6,1);
upper = zeros(6,1);
lower(1) =-10*d2r;
lower(2) = -5*d2r;
lower(3) = world_data_xcll;
lower(4) = world_data_xall;
lower(5) = world_data_xbll;
lower(6) = world_data_xpll;

upper(1) = 10*d2r;
upper(2) =  5*d2r;
upper(3) = world_data_xcul;
upper(4) = world_data_xaul;
upper(5) = world_data_xbul;
upper(6) = world_data_xpul;
popg
// End of configure.exc
```

```
////////////////////////////////////////////////////
//  file: assemble.exc
// Assemble the helicopter model:
//  date: 21 July 1993
////////////////////////////////////////////////////

topsolve_cf       = 1;
    rhsolve_cf    = 0;
     rotorsolve_cf      = 0;
     helisolve_cf = 0;
    drivesolve_cf = 0;
    contsolve_cf  = 0;

drivesolve_reassemble = 0;
drivesolve::assemble;
drivesolve::reset;

contsolve_reassemble = 0;
contsolve::assemble;
contsolve::reset;
// Run to steady state:
for i=1:36; contsolve::step, contsolve::iter; end
contsolve::saveic;

exec("mbc_setup.exc",1);

rotorsolve_reassemble = 0;
rotorsolve::reset;
rotorsolve::assemble;

rhsolve_reassemble      = 0;
rhsolve::reset;
rhsolve::assemble;
for i=1:10, rhsolve::genq; rhsolve::solve; end
rhsolve::assemble;
rhsolve_niter=0; rhsolve::iter, rhsolve_niter

// End of assemble.exc

////////////////////////////////////////////////////
// file: mbc_setup.exc
// This file sets up the MBC transformation for the rigid rotor
// date: 22 July 1993
////////////////////////////////////////////////////

mbcset(model_heli_rotor_mbc, 1);

// End of mbc_setup.exc
```

APPENDIX C

```
////////////////////////////////////////////////////////////////////////
//  file: trimsweep.def                                               //
//  Definition file to conduct trim sweep for psh with rigid blades   //
////////////////////////////////////////////////////////////////////////

// execute the psh.def file to assemble the model for running
exec("psh.def",1)

// load additional functions used
exec("limitchange.fun",1); // function to limit control change size
exec("plotutils.exc",1)  // plotting utilities

exec("Trim_Sweep.exc") // Conduct trim sweep for the range of airspeeds
desired

// display trim control plots if desired
showplt = Enter("Enter a 1 to plot control psn vs airspeed or 0 to
exit:");
if showplt == 1
plot("clear")
plot("title = Control Position vs Airspeed,xlabel = Veq(kts)")
plot("ylabel =Collective (inches)")
plot(cpg_stc(1,:)',cpg_stc(4,:)')
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("ylabel =Longitudinal Cyclic(inches)")
plot(cpg_stc(1,:)',cpg_stc(6,:)')
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("ylabel =Lateral Cyclic(inches)")
plot(cpg_stc(1,:)',cpg_stc(5,:)')
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("ylabel =Pedal (inches)")
plot(cpg_stc(1,:)',cpg_stc(7,:)')
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
elseif showplt == 0
disp("Trim Control Matrix is saved in file called Trim_Controls.rbe")
disp("Trim Matrix is saved in file called Trim_Matrix.rbe")
end
disp("end of trimsweep.def file")
/////// end of trimsweep.def////////
```

```
////////////////////////////////////////////////////////////////////
// file: limitchange.fun
//  This file limits the change of control during an update
// date: 27 August 1993
////////////////////////////////////////////////////////////////////

function y = limitchange(x(n),pct,range(n))
//
//
// Y = LIMITCHANGE(X,PCT,RANGE)
//
// Inputs:
// ------
// X        - Input vector
// PCT      - Allowable percentage change (0-1)
// RANGE    - Total range of X (must be same size as X)
//

// Outputs:
// -------
//
// Y        - X limited to the allowable percentage of the range
//
// This function limits the amount the signal can change to a
// specified percentage of the total range
//

function sign;

   y = x;
   for i=1:n
      if abs(x(i)) > pct*range(i)
         y(i) = sign(pct*range(i),x(i));
         disp("Limiting Change on "$i)
      end
   end
end
//////////end of limitchange.fun /////////////
```

```
////////////////////////////////////////////////////////
// file: Trim_Sweep.exc
// date: 31 August 1993
//  This file performs a trim sweep over a range of
//  airspeeds in increment specified by user
////////////////////////////////////////////////////////
pushg(world_cpg)

nprev   = world_data_naz;//# of azimuth steps/revolution
nrevss  = 4; //# of revolutions to reach steady state
nrevavg = 1; //# of revolutions for obtaining average
converg = 0.0001;
bias    = zeros(6,1);
Trimg   = ones(bias) "Trim Gain (nd)";
//exec("Compute_Trim_Matrix.exc",1)
//disp("Trim Matrix")
//disp( (trm) )
stc = [];
output([])
output(bacc)
popg

exec("Set_Airspeed.exc",1) // Set airpseed for initial setup
  exec("Steady_State.exc",1) // Run model to steady state
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)
  exec("Steady_State.exc",1)

pushg(world_cpg)
  vstart = veq;
  vinc   = Enter("Enter increment velocity :");
  vend   = Enter("Enter ending    velocity :");

  tc = [];
  trmd=[];
  output([])
  output(bata,bara)


  for ijk= vstart:vinc:vend
    veq = ijk;
    trimg =  ones(trimg);
    exec("Compute_Trim_Matrix.exc",1)
    exec("Save_Trim_Matrix.exc",1)
    disp((trm))
    exec("Trim.exc",1)
    tc=[veq ean(2) ean(1) xctrm xatrm xbtrm xptrm mth a1s b1s tth gamvr
gamhr]';
    exec("Save_Controls.exc",1)
  end
  save("Trim_Controls.rbe",stc)
  save("Trim_Matrix.rbe",trmd)
popg

// End of Trim_Sweep.exc ////
```

```
////////////////////////////////////////////////////
// file: Compute_Trim_Matrix.exc
//  This script computes a diagonal matrix which is
//  the partial of one acceleration which is coupled to
//  one control.
// date: 27 July 1993
////////////////////////////////////////////////////

//  This script computes a diagonal matrix which is
//  the partial of one acceleration which is coupled to
//  one control.
//
//  th ~ ud   (pitch attitude couples with longitudinal accel)
//  ph ~ vd   (roll  attitude couples with lateral      accel)
//  xc ~ wd   (collective     couples with vertical     accel)
//  xa ~ pd   (lateral cyclic couples with roll         accel)
//  xb ~ qd   (long.   cyclic couples with pitch        accel)
//  xp ~ rd   (pedal          couples with yaw          accel)
//
// |d(ud)/d(th)      0            0            0            0            0       |
// |     0      d(vd)/d(ph)       0            0            0            0       |
// |     0           0       d(wd)/d(xc)       0            0            0       |
// |     0           0            0       d(pd)/d(xa)       0            0       |
// |     0           0            0            0       d(qd)/d(xb)       0       |
// |     0           0            0            0            0       d(rd)/d(xp) |
//

pushg(world_cpg)

// Steady_State.exc scripts computes the average values
// of the output if outputs have been selected.  The results
// are returned in a0.

output([])
output(bacc)

exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
a00 = a0;

x0 = ean(2);

trm = zeros(6,6);



i = 1;
ean(2) = ean(2) + 0.01;
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
ean(2) = ean(2) - 0.01;

i = 2;
ean(1) = ean(1) + 0.01;
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
ean(1) = ean(1) - 0.01;

i = 3;
xctrm = xctrm + 0.01;
```

```
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
xctrm = xctrm - 0.01;


i = 4;
xatrm = xatrm + 0.01;
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
xatrm = xatrm - 0.01;

i = 5;
xbtrm = xbtrm + 0.01;
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
xbtrm = xbtrm - 0.01;

i = 6;
xptrm = xptrm + 0.01;
exec("Steady_State.exc",1)
trm(i,i)= (a0(i) - a00(i))/0.01;
xptrm = xptrm - 0.01;

trm = inv(trm);
trm = - trm;

save("Trim_Matrix.rbe",trm)

popg
// End of Compute_Trim_Matrix.exc

////////////////////////////////////////////////////
// file: Trim.exc
// This script applies an extremely simplistic trim algorithm
// to drive the accelerations to zero.
// date: 3 Setember 1993
////////////////////////////////////////////////////
// This script applies an extremely simplistic trim algorithm to
// drive the accelerations to zero.
pushg(world_cpg)
// Set body velocities from prescribed flight condition
    exec("Compute_Body_Axis_Velocity.exc",1)
    world_rhsolve::assemble
    world_topsolve::iter;

// Run to steady state
    output([])
    output(world_cpg_bacc)
    disp("Run to steady state");
    exec("Steady_State.exc",1);
//**********************TRIM LOOP 1*********************************
//
    itercnt=0;
    trimfail=0;
    disp("Perform gradient trim");
    while norm(a0-bias) > converg && itercnt < 20
        itercnt = itercnt + 1;
        exec("Update_CW.exc",1);
     disp("Iteration count is "$itercnt$" at "$veq$" knots");
     disp("Average accelerations are");
     disp((a0'))
```

```
     if (itercnt==20)
         trimfail=1;
     end
    end
//
//if trim fails, reduce gradient step by 50% and try once more
//
    if (trimfail==1)
     world_cpg_trimg=.5*world_cpg_trimg;
//
//**********************TRIM LOOP 2*********************************
//
        itercnt=0;
        trimfail=0;
        disp("Perform gradient trim");
        while norm(a0-bias) > converg && itercnt < 20
            itercnt = itercnt + 1;
            exec("Update_CW.exc",1);
         disp("Iteration count is "$itercnt$" at "$veq$" knots");
            disp("Average accelerations are");
         disp((a0'))
            if (itercnt==20)
                trimfail=1;
            end
        end
//*********************************************************************
    end
//
//if trim fails, reduce gradient step by 50% and try once more
//
    if (trimfail==1)
     world_cpg_trimg=.5*world_cpg_trimg;
//
//**********************TRIM LOOP 3*********************************
//
        itercnt=0;
        trimfail=0;
        disp("Perform gradient trim");
        while norm(a0-bias) > converg && itercnt < 20
            itercnt = itercnt + 1;
            exec("Update_CW.exc",1);
         disp("Iteration count is "$itercnt$" at "$veq$" knots");
            disp("Average accelerations are");
         disp((a0'))
            if (itercnt==20)
                trimfail=1;
            end
        end
//*********************************************************************
    end

popg
// End of Trim.exc

////////////////////////////////////////////////////////////////
// file: Update_CW.exc
// This file update controls and modified trim weights
// date: 27 August 1993
////////////////////////////////////////////////////////////////

a00 = a0;               // Save the last average accels
```

```
exec("Update.exc",1)    // Update controls
for i=1:6               // If the sign of the accels changed
    sa1 = a0(i)/abs(a0(i));// Then decrease that trim gain by
    sa0 = a00(i)/abs(a00(i));// by 10 percent.  Lower limit is 0.5
    if sa1 <> sa0
     trimg(i) = max(0.5,0.9*trimg(i));
    end
end
// End of Update_CW.exc

////////////////////////////////////////////////////////////////////
// file: Update.exc
//  This file update the controls
// date: 27 July 1993
////////////////////////////////////////////////////////////////////

pushg(world_cpg)
  deltac = trm*(trimg.*a0);
  deltac = limitchange(deltac,0.02,abs(upper-lower));

  ean(2) = min(upper(1),max(lower(1),ean(2) + deltac(1)));
  ean(1) = min(upper(2),max(lower(2),ean(1) + deltac(2)));
  xctrm  = min(upper(3),max(lower(3),xctrm  + deltac(3)));
  xatrm  = min(upper(4),max(lower(4),xatrm  + deltac(4)));
  xbtrm  = min(upper(5),max(lower(5),xbtrm  + deltac(5)));
  xptrm  = min(upper(6),max(lower(6),xptrm  + deltac(6)));

  disp("New control setting (theta phi xc xa xb xp)")
  disp([[ean(2) ean(1)]*r2d xctrm xatrm xbtrm xptrm])
  exec("Compute_Body_Axis_Velocity.exc",1)
popg

exec("Steady_State.exc",1)

// end of Update.exc

////////////////////////////////////////////////////////////
//  file: Compute_Body_Axis_Velocity
//  This file uses the equivalent airspeed and the flight path
//  angle sines and cosines to set the aircraft body axis velocity
//  date: 4 Nov 1992
////////////////////////////////////////////////////////////
pushg(world_cpg)
   world_rhsolve::genq;
   vt = veq*sqrt(rho0/rho)*k2f;
   cgv  = cos(gamvr); sgv = sin(gamvr);
   cgh  = cos(gamhr); sgh = sin(gamhr);
   vi = vt*[cgv*cgh cgv*sgh -sgv]';
   tib(1:3,1:3) = tran;
   bvel = tib*vi;
   world_rhsolve::genq

popg
// End of  Compute_Body_Axis_Velocity
```

```
////////////////////////////////////////////////////////////
// file: Save_Trim_Matrix.exc
// date: 2 September 1993
// Saves the trim matrix for each airspeed as a vector
////////////////////////////////////////////////////////////
pushg(world_cpg)

trmd=[trmd [veq diag(trm)']'];
describe("Saved Trim Matrix: veq d(ud)/d(th) d(vd)/d(ph) d(wd)/d(xc)
d(pd)/d(xa) d(qd)/d(xb) d(rd)/d(xp)");

popg
// End of Save_Trim_Matrix.exc

////////////////////////////////////////////////////////////
// file: Save_Controls.exc
// date: 27 July 1993
//  Include the trim controls which you wish to save
//  in this list
////////////////////////////////////////////////////////////

pushg(world_cpg)

stc=[stc [veq ean(2) ean(1) xctrm xatrm xbtrm xptrm mth a1s b1s tth
gamvr gamhr]'];
describe("Saved Trim Controls: veq ean(2) ean(1) xctrm xatrm xbtrm xptrm
mth a1s b1s tth gamvr gamhr");

popg
// End of Save_Controls.exc

////////////////////////////////////////////////////////
// file: Set_Airspeed.exc
// User specifies airspeed in knots
//  vertical and horizontal flight path angle in degrees
// date: 3 August 1993
////////////////////////////////////////////////////////
pushg(world_cpg)
veq   = Enter("Enter equivalent airspeed (knots)      : ");
gamvr = Enter("Vertical flight path angle(deg) + down: ")*d2r;
gamhr = Enter("Horizontal flight path angle(deg) + r : ")*d2r;
exec("Compute_Body_Axis_Velocity.exc",1)
popg
// End of Set_Airspeed.exc

////////////////////////////////////////////////////////
// file: Steady_State.exc
// This script runs the model to steady state
// date: 27 July 1993
////////////////////////////////////////////////////////
pushg(world_cpg)
y = nrun(nprev*nrevss);
if y <> []
  plot("xlabel=Time steps for nrevss,ylabel=average body accels")
//  plot(nprev*nrevss*world_data_dt,y)
plot(y)
  sk = ((nrevss-nrevavg)*nprev+1):(nrevss*nprev);
  a0 = sum(y(sk,:))'/(nprev*nrevavg);
end
popg
// End of Steady_State.exc
```

APPENDIX D

```
////////////////////////////////////////////////////////////////////////
//  file: analysis.def                                                //
//  Definition file to analyze the psh with rigid blades              //
//  using the linearization routine and sample tests                  //
////////////////////////////////////////////////////////////////////////

// execute the psh.def file to assemble the model for running
exec("psh.def",1)

//  add path to directory where test files are located
path("tests/")

// load additional functions
exec("qsreduce.fun",1); // linear reduction function
exec("plotutils.exc",1)  // plotting utilities
exec("logspace.fun.fun",1);  // function for creating frequency range
exec("freq.fun",1);      // function for the freq sweep test

exec("Restore_Trim.exc") // Restores the model to trim at selected
airspeed
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)

// Run the model and determine the non linear characteristic matrix

frh=nlmodel(rhsolve);  // open loop system matrix based on rhsolve
solution

//  To compare the nonlinear results with a reduced
//  order linear model, you need to create a reduced
//  order linear model.

cpg_havembc=1;  //flag to indicate use of multi-blade coord system

exec("6dof_linearize.exc",1); // returns linear matrix for 6dof model

// Where
// full f matrix in f
// full g matrix in g
// reduced f matrix in fq
// reduced g matrix in gq
// change ndof to 10, 11, 15 in 6dof _linearize.exc to get other
// reduced models

// Setup matrices to compare linear and nonlinear model response
//  Select the states (u,v,w,p,q,r,phi,theta) as outputs
//  by setting the hq matrix to identity and the dq matrix
//  to zero;

hq = eye(fq);
dq = zeros(gq);
// Construct a linear system matrix
sq = [fq gq
      hq dq];
```

```
// Discretize the linear system matrix (convolution integral)

sqd = disc(sq,8,topsolve_dt);


// Make a plot of the eigenvalues for the linear system matrix sq

plot("clear")
eigmod=eig(sq(1:8,1:8));
plot("title=Eigenvalues Linear Model,xlab =Real Axis,ylab=Imag Axis")
plot("mark=1,line=0");
plot(real(eigmod),imag(eigmod));
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("clear")

disp("The reduced linear characteristic matrix");disp((fq));pause
disp("The reduced linear control matrix");disp((gq));pause

{V,D}=eig(fq); //eigenvectors and eigenvalues of linear matrix
ceqn=poly(fq); //characteristic equation of linear matrix

disp("The characteristic equation coefficients of the linear matrix fq")
disp((ceqn))
pause
disp("The eigenvectors of the linear characteristic matrix")
disp((V))
pause
disp("The eigenvalues of the linear characteristic matrix")
disp((diag(D)))
pause

// conduct longitudinal axis tests

exec("Long_Impulse.exc",1) //use 60 sec, 0.025 sec delay, 1 inch

// Propagate the linear model, the input vector must
// have the same dimensionality as the gq matrix
// which is [xb xa xp xc]'

yllng = filp(sqd,[ulng 0*ulng 0*ulng 0*ulng]);

// Get the bias of the nonlinear run
x0 = ylng(1,1:8);
//  Add the nonlinear bias to the linear model results
//  and plot the results
for i=1:8
  yllng(:,i) = yllng(:,i) + x0(i);
end
//  Plot the response of the linear and nonlinear models
plot("clear")
plot("title=Phugoid Response")
plot("upper,xlab=time (sec),ylab=theta (deg)")
plot(tv,ylng(:,8).*world_data_r2d,tv,yllng(:,8).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,title= ,ylab=velocity (fps)")
plot(tv,ylng(:,1),tv,yllng(:,1))
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
// end of longitudinal impulse test
```

```
// conduct lateral axis tests

exec("Lat_Step.exc",1) //use 15 sec,.025 delay,1.0 inch,2.5 sec, .025
rise/fall

// Propagate the linear model and
// Get the bias of the nonlinear run
//  Add the nonlinear bias to the linear model results
//  and plot the results

yllat = filp(sqd,[0*ulat ulat 0*ulat 0*ulat]);
for i=1:8
  yllat(:,i) = yllat(:,i) + x0(i);
end
plot("clear")
plot("title=Roll Response")
plot("upper,xlab=time (sec),ylab=Phi (deg)")
plot(tv,ylat(:,7).*world_data_r2d,tv,yllat(:,7).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,title= ,ylab=Roll Rate (deg/sec)")
plot(tv,ylat(:,4).*world_data_r2d,tv,yllat(:,4).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end

exec("Lat_Impulse.exc",1) //use 30 sec, 0.025 sec delay, 1 inch right
stick

// Propagate the linear model and
// Get the bias of the nonlinear run
//  Add the nonlinear bias to the linear model results
//  and plot the results

yllat2 = filp(sqd,[0*ulat2 ulat2 0*ulat2 0*ulat2]);
x0 = ylat2(1,1:8);
for i=1:8
  yllat2(:,i) = yllat2(:,i) + x0(i);
end
plot("clear")
plot("title=Spiral Response")
plot("upper,xlab=time (sec),ylab=Phi (deg)")
plot(tv,ylat2(:,7).*world_data_r2d,tv,yllat2(:,7).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,title= ,ylab=Roll Rate (deg/sec)")
plot(tv,ylat2(:,4).*world_data_r2d,tv,yllat2(:,4).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end

//conduct vertical axis tests

exec("Ped_Doublet.exc",1) // use 20 sec,0.025,1 inch,1.5,.025,.025,.025

// Propagate the linear model and
// Get the bias of the nonlinear run
//  Add the nonlinear bias to the linear model results
//  and plot the results

ylped = filp(sqd,[0*uped 0*uped uped 0*uped]);
x0 = yped(1,1:8);
```

```
for i=1:8
  ylped(:,i) = ylped(:,i) + x0(i);
end
plot("clear")
plot("title=Dutch-roll Response")
plot("upper,xlab=time (sec),ylab=Phi (deg)")
plot(tv,yped(:,7).*world_data_r2d,tv,ylped(:,7).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,left,title= ,xlab=time (sec),ylab=Roll Rate (deg/sec)")
plot(tv,yped(:,4).*world_data_r2d,tv,ylped(:,4).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,right,title= ,xlab=time (sec),ylab=Yaw Rate (deg/sec)")
plot(tv,yped(:,6).*world_data_r2d,tv,ylped(:,6).*world_data_r2d)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end

// conduct freq response test
// simplify system matrix to have only the input column desired for the
test
// you want and the only the ouputs for that test
// example frequency response of q/long stick and theta/long stick

[f,g,h,d]=split(sq,8);
g=g(:,1);  // Pick up only the column associated with long stick

//  Output index for pitch rate is 5 (u v w p q r phi theta) and 8 for
// theta, so
sk = [8]
h = h(sk,:);
[no,ns]=size(h);
[ns,ni]=size(g);
d=zeros(no,ni);
s=[f g;
   h d];

w=logspace(0.1,100,200);
[amp,phase]=freq(s,8,w);

// The scope plot package does not handle log plots very well, so to get
// equivalent system, convert the data to a log base 10 scale.  The
// x axis now represent powers of 10.

sc=1/log(10);      //conversion from ln to log base 10
scg=20/log(10); //conversion from ln to db

// plot pitch attitude response to longitudinal freq sweep
plot("clear")
plot("upper,title=frequency response theta vs delta long,xlab=
,ylab=Gain(db)")
plot(sc*log(w),scg*log(amp))
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
plot("lower,title= ,xlab=Frequency ((rad/sec) in power of 10),ylab=Phase
(deg)")
plot(sc*log(w),phase)
pli = Enter("Enter 1 to print plot 0 to continue:");
if pli == 1; plot("print");elseif pli == 0;disp("plot not printed");end
disp("end of analysis.def file")
//end of analysis.def
```

```
////////////////////////////////////////////////////////////////////
// file: Restore_Trim.exc
// date: 27 July 1993
// This script restores the trim matrix and trim control matrix
// and runs the model to steady state for the selected a/s
////////////////////////////////////////////////////////////////////

pushg(cpg)
load("Trim_Controls.rbe")
load("Trim_Matrix.rbe")
[r,c]=size(cpg_stc);
test=diag(ones(c));for i = 1:c;test(i)=test(i)*i;;end
disp(([test cpg_stc(1,:)']))
Ti = Enter("Enter the row number of the airspeed to analyze:");
world_cpg_veq    = cpg_stc(1,ti);
world_cpg_ean(2) = cpg_stc(2,ti);
world_cpg_ean(1) = cpg_stc(3,ti);
world_cpg_xctrm  = cpg_stc(4,ti);
world_cpg_xatrm  = cpg_stc(5,ti);
world_cpg_xbtrm  = cpg_stc(6,ti);
world_cpg_xptrm  = cpg_stc(7,ti);
world_cpg_gamvr  = cpg_stc(12,ti);
world_cpg_gamhr  = cpg_stc(13,ti);

trm=diag(cpg_trmd(2:7,ti)');
disp("Trim Matrix="); disp((trm))
tc=cpg_stc(:,ti)';
disp("Trim Control Matrix=");disp((tc))

nprev   = world_data_naz;
nrevss  = 4;
nrevavg = 1;
converg = 0.0001;
bias    = zeros(6,1);
Trimg   = ones(bias) "Trim Gain (nd)";

stc = [];
output([])
output(bacc)

veq   = cpg_tc(1,1);
disp("Equivalent Arispeed =");disp((veq))
gamvr = Enter("Enter Vertical flight path angle(deg) + down: ")*d2r;
gamhr = Enter("Enter Horizontal flight path angle(deg) + r : ")*d2r;
exec("Compute_Body_Axis_Velocity.exc",1)

popg

output([])
output(world_cpg_bacc)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
disp("Average accelerations")
disp((world_cpg_a0))
// End of Restore_Trim.exc
```

```
////////////////////////////////////////////////////////////////
// file: Long_Impulse.exc
// This file sets up a longitudinal impulse
// date: 3 August 1993
////////////////////////////////////////////////////////////////

//  This file sets up a Longitudinal Impulse

pushg(world_cpg)
  exec("Select_Outputs.exc",1);
  input([])
  input(xb);
popg

tend = Enter("Enter run duration  (sec)        : ");
tdly = Enter("Enter delay time of impulse (sec): ");
ampl = Enter("Enter impulse amplitude pos+ (inches) : ");

ulng = ipulse(world_topsolve_dt,tend,tdly,ampl) "Impulse input signal
(in)";
[r,c]=size(ulng);
tv = world_topsolve_dt*(0:r-1)' "Time Vector (sec)";
plot("clear")
plot("xlab=Time (sec),ylab=Long. Input Signal")
plot(tv,ulng)
world::saveic
exec("Unfreeze.exc",1)
ylng = nrun(ulng) "Nonlinear model response to longitudinal impulse";
exec("Freeze.exc",1)
plot("clear")
plot("xlabel = Time (sec),ylabel = Theta (deg) ")
plot(tv,ylng(:,8)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
plot("ylabel = Velocity (fps) ")
plot(tv,ylng(:,1))
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
world::restoreic;
pushg(world_cpg)
   output([])
   output(bacc)
popg
plot("clear")
exec("Steady_State.exc",1)
disp("end of Long_Impusle.exc")
// End of Long_Impulse.exc
```

```
////////////////////////////////////////////////////
// file: Lat_Step.exc
//  This file sets up a Lateral Step
// date: 3 August 1993
////////////////////////////////////////////////////

//  This file sets up a Lateral Step

pushg(world_cpg)
  exec("Select_Outputs.exc",1);
  input([])
  input(xa);
popg

tend = Enter("Enter run duration  (sec)          : ");
tdly = Enter("Enter delay time of step     (sec): ");
ampl = Enter("Enter step amplitude (inches)     : ");
delta= Enter("Enter step duration (sec)         : ");
trise= Enter("Enter rise time (sec)             : ");
tfall= Enter("Enter fall time (sec)             : ");

ulat = istep(world_topsolve_dt,tend,tdly,delta,trise,tfall,ampl);
describe(ulat,"Step input signal (in)");
[r,c]=size(ulat);
tv = world_topsolve_dt*(0:r-1)' "Time Vector (sec)";
plot("clear")
plot("xlab=Time (sec),ylab=Input Signal")
plot(tv,ulat)
world::saveic
 exec("Unfreeze.exc",1)
ylat = nrun(ulat) "Nonlinear model response to lateral step";
exec("Freeze.exc",1)
plot("clear")
//plot("title = NL Roll Response @ "$veq$" kts")
plot("xlabel = Time (sec),ylabel = Phi (deg) ")
plot(tv,ylat(:,7)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
plot("ylabel = Roll Rate (deg/sec)")
plot(tv,ylat(:,4)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
world::restoreic;
pushg(world_cpg)
  output([])
  output(bacc)
popg
plot("clear")
exec("Steady_State.exc",1)
disp("end of Lateral_Step.exc")
// End of Lat_Step.exc
```

```
////////////////////////////////////////////////////////
// file: Lat_Impulse.exc
//  This file sets up a Lateral Impulse
// date: 3 August 1993
////////////////////////////////////////////////////////

//  This file sets up a Lateral Impulse
pushg(world_cpg)
  exec("Select_Outputs.exc",1);
  input([])
  input(xa);
popg

tend = Enter("Enter run duration  (sec)        : ");
tdly = Enter("Enter delay time of impulse (sec): ");
ampl = Enter("Enter inpulse amplitude right+ (inches) : ");

ulat2 = ipulse(world_topsolve_dt,tend,tdly,ampl) "Impulse input signal
(in)";
[r,c]=size(ulat2);
tv = world_topsolve_dt*(0:r-1)' "Time Vector (sec)";
plot("clear")
plot("xlab=Time (sec),ylab=Input Signal")
plot(tv,ulat2)

world::saveic
exec("Unfreeze.exc",1)
ylat2 = nrun(ulat2) "Nonlinear model response to longitudinal impulse";
exec("Freeze.exc",1)
plot("clear")
//plot("title = NL Spiral Response @ "$veq$" kts")
plot("xlabel = Time (sec),ylabel = Phi (deg) ")
plot(tv,ylat2(:,7)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
plot("ylabel = Roll Rate (deg/sec)")
plot(tv,ylat2(:,4)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
world::restoreic;
pushg(world_cpg)
   output([])
   output(bacc)
popg
plot("clear")
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
disp("end of Lateral_Impulse.exc")
// End of Lat_Impulse.exc
```

```
////////////////////////////////////////////////////////
// file: Ped_Doublet.exc
//  This file sets up a Pedal Doublet
// date: 3 August 1993
////////////////////////////////////////////////////////

//  This file sets up a Pedal Doublet

pushg(world_cpg)
  exec("Select_Outputs.exc",1);
  input([])
  input(xp);
popg

tend = Enter("Enter run duration  (sec)         : ");
tdly = Enter("Enter delay time of doublet (sec): ");
ampl = Enter("Enter step amplitude (inches)     : ");
delta= Enter("Enter step duration (sec)         : ");
trise= Enter("Enter rise time (sec)             : ");
tfall= Enter("Enter fall time (sec)             : ");
tnxt = Enter("Enter time between steps (sec)    : ");

uped =
idoublet(world_topsolve_dt,tend,tdly,delta,trise,tfall,tnxt,ampl);
describe(uped,"Doublet input signal (in)");
[r,c]=size(uped);
tv = world_topsolve_dt*(0:r-1)' "Time Vector (sec)";
plot("clear")
plot("xlab=Time (sec),ylab=Input Signal")
plot(tv,uped)
world::saveic
exec("Unfreeze.exc",1)
yped = nrun(uped) "Nonlinear model response to pedal doublet";
exec("Freeze.exc",1)
plot("clear")
plot("xlabel = Time (sec) ");plot("ylabel = Phi (deg) ")
plot(tv,yped(:,7)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
plot("ylabel =  Roll Rate (deg/sec)")
plot(tv,yped(:,4)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
plot("ylabel = Yaw Rate (deg/sec)")
plot(tv,yped(:,6)*world_data_r2d)
//pli = Enter("Enter 1 to print plot 0 to continue:");
//if pli == 1; plot("print");elseif pli == 0;disp("plot not
printed");end
world::restoreic;
pushg(world_cpg)
  output([])
  output(bacc)
popg
plot("clear")
exec("Steady_State.exc",1)
exec("Steady_State.exc",1)
disp("end of Pedal_Doublet.exc")
// End of Ped_Doublet.exc
```

```
////////////////////////////////////////////////////////
// file: Select_Outputs.exc
// This file selects the outputs for the tests
// date: 3 August 1993
////////////////////////////////////////////////////////
output([])
output(bvel,brat,ean)
for i=1:world_data_nblades
   output(flap$i)
end
for i=1:world_data_nblades
   output(lag$i)
end
describe("Outputs 1-3: Body axis translational velocities u,v,w
(ft/sec)");
describe("Outputs 4-6: Body axis rotational    velocities p,q,r
(rad/sec)");
describe("Outputs 7-9: Euler angle roll, pitch yaw (rad)");
describe("Outputs 10-13: Flap response of blades 1,2,3,4 (rad)");
describe("Outputs 14-17: Lag  response of blades 1,2,3,4 (rad)");

// End of Select_Outputs.exc

////////////////////////////////////////////////////////////
// File: 6dof_linearize
// This file generates a 6dof linear model for psh:
// date: 27 July 1993
////////////////////////////////////////////////////////////
[f,g]=LinearModel(topsolve);
[ns,ns]=size(f);
// Fetch state and input indices of relevant components:
ix6=StateIndices(topsolve,world_model_heli_body_bodydof) "Body state
indices";
ix6r=ix6([13:18 4 5]) "Reduced body state indices";
iup=[  InputIndices(topsolve,world_model_cont_lng_xb);
       InputIndices(topsolve,world_model_cont_lat_xa);
       InputIndices(topsolve,world_model_cont_dir_xp);
       InputIndices(topsolve,world_model_cont_coll_xc)] "Input indices";
nup=prod(size(iup));
savg=[f g(:,iup)];

// average linear model if required

    nrun(3);
    [f,g]=LinearModel(topsolve);
    s=[f g(:,iup)];
    savg=savg+s;
    nrun(3);
    [f,g]=LinearModel(topsolve);
    s=[f g(:,iup)];
    savg=(savg+s)/3;
f=savg(1:ns,1:ns);
gp=savg(1:ns,ns+1:ns+nup);

//Set up number of DOFs for reduced order model

ndof=6;

//Perform quasistatic reduction
exec("qsreduce.exc",1);
// end of 6dof_linearize.exc
```

```
////////////////////////////////////////////////////////////////////////
// File:         qsreduce.exc
// Author:       Ron DuVal and Joe English
// Description: Computes quasistatically reduced linear model of
helicopter.
// See-Also:     linearize.exc
// Notes:        Most of the inputs to this script are produced by
linearize.exc
////////////////////////////////////////////////////////////////////////

//  INPUTS:
//      NDOF     -- # degrees of freedom in reduced model
//                  Must be set to:
//
//           6   -- DOF6 body states only
//          10   -- DOF6 body states + flapping
//          11   -- DOF6 body states + flapping + inflow
//          15   -- DOF6 body states + flapping + lead-lag + inflow
//           0   -- Do not do any reduction


//  (computed in linearize.exc:)
//      F(ns,ns) -- full state matrix
//      GP(ns,4) -- full control matrix (inputs are trim controls)
//      H(14,ns)
//      DP(14,4)  -- output matrices for body velocities & accelerations
(???)
//
//  OUTPUTS:
//      FQ(n,n)
//      GQ(n,4)    -- state and control matrix of reduced model if ndof
<> 0
//      HQ(???,n)
//      DQ(???,4) -- steady state output matrices for reduced model

//Compute indices for quasistaic reduction

ix6=StateIndices(topsolve,world_model_heli_body_bodydof) "Body state
indices";
ix6r=ix6([13:18 4 5]) "??? body state indices";
idrtrn=StateIndices(topsolve,drivesolve) "Drivetrain state indices";
if (world_cpg_havembc == 1),
    irotor=StateIndices(topsolve,model_heli_rotor_mbc);
    iflap=irotor([1:4 9:12]) "Flapping state indices";  // %%% will
change
    ilag=irotor([5:8 13:16]) "Lead-lag state indices";  // %%% will
change
else
    irotor=[];
    iflap=[];
    ilag=[];
end
iinflow=StateIndices(topsolve,model_heli_inflow_u1) "Inflow state
indices";

// Determine dynamic states based on NDOF:

if ndof == 6,
       idyn=[ix6r];
elseif ndof == 10,
       idyn=[ix6r;iflap];
```

```
elseif ndof == 11,
       idyn=[ix6r;iflap;iinflow];
elseif ndof == 15,
       idyn=[ix6r;iflap;ilag;iinflow];
elseif ndof == 0,
       // ???
else
       error("NDOF = "$ndof$"; must be 6,10,11 or 15");
end

// Always eliminate drivetrain and ??? dof6 states

ielim=[idrtrn;ix6([1:3 6:12])];

// Compute reduced model:

if(ndof <> 0),
    [fq,gq,hq,dq]=qsreduce(f,gp,idyn,ielim,0);
end;

// end of qsreduce.exc
```

```
////////////// results of analysis.def /////////////////
Trim Matrix=
    0.0312    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000   -0.0312    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.1276    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000   -0.5613    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    1.8720    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000   -0.5594

The reduced linear characteristic matrix
   -0.0300    0.0091   -0.0591   -1.2212    1.5360   -0.2055    0.0000
-32.0124
    0.0019    0.0307   -0.0095   -0.7664   -0.0879    0.4489   32.0036
-0.0810
   -0.0456   -0.0067   -0.7201   -2.1623   -1.2583    2.7010    0.7461
3.4700
    0.0015   -0.0462    0.0134   -6.6040   -1.1185    1.2524    0.0000
0.0000
    0.0036    0.0100   -0.0025    0.2859   -1.6059   -0.3555    0.0000
0.0000
   -0.0057    0.0149   -0.0193    0.0024    0.0888   -0.5162    0.0000
0.0000
    0.0000    0.0000    0.0000    1.0000    0.0025   -0.1084    0.0000
0.0000
    0.0000    0.0000    0.0000    0.0000    0.9997    0.0233    0.0000
0.0000

The reduced linear control matrix
    1.8661   -0.1586    0.4224   -0.8579
    0.3245    0.2813   -2.4906   -0.2769
    6.7358   -0.0642    1.0253   -8.1918
    0.1939    1.8587   -2.1082    0.1150
   -0.5083    0.0599    0.6609    0.1455
    0.1236   -0.0548    1.7156    0.0738
    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000


The characteristic equation coefficients of the linear matrix fq
 1.0000 + 0.0000i
 9.4455 + 0.0000i
21.4968 + 0.0000i
18.5608 + 0.0000i
 9.8123 + 0.0000i
 4.5450 - 0.0000i
 1.4522 - 0.0000i
 0.2193 + 0.0000i
 0.0759 + 0.0000i
```

```
The eigenvectors of the linear characteristic matrix
 columns 1 thru 4
-0.1651 + 0.0627i  0.1248 - 0.9841i -1.2950 - 0.0458i  0.4863 - 1.0047i
-0.5704 + 0.2168i  0.0238 - 0.1875i -0.6219 + 0.0176i  0.4434 - 1.4703i
-0.2499 + 0.0950i -0.0241 + 0.1903i -0.3135 + 0.9051i -0.1009 + 0.0154i
-0.6684 + 0.2540i  0.0023 - 0.0181i -0.0164 + 0.0062i -0.0052 + 0.0069i
 0.0396 - 0.0151i -0.0104 + 0.0820i  0.0140 - 0.0073i  0.0036 - 0.0059i
 0.0001 - 0.0000i  0.0007 - 0.0053i -0.0849 + 0.0114i -0.0101 - 0.0243i
 0.1017 - 0.0387i -0.0013 + 0.0101i  0.0130 - 0.0039i  0.0206 + 0.0087i
-0.0060 + 0.0023i  0.0060 - 0.0476i -0.0213 + 0.0048i -0.0141 - 0.0071i
 columns 5 thru 8
 3.3901 + 6.8954i  9.2088 - 0.1581i  4.6426 -24.9243i 21.3797 -53.2807i
-2.2410 - 0.6718i  4.4177 + 0.2052i 11.6962 -32.8607i -16.3188 + 6.2655i
-0.4637 - 0.5540i  2.1109 + 6.4724i  1.6378 + 1.6409i -3.1432 + 4.3881i
-0.0040 - 0.0015i  0.1155 + 0.0459i  0.0036 + 0.1960i -0.0288 + 0.0132i
 0.0155 + 0.0141i -0.0984 - 0.0538i  0.0132 - 0.1568i  0.1073 - 0.1140i
-0.0889 - 0.0234i  0.6018 + 0.0922i  0.5115 - 0.3090i -0.6494 + 0.2244i
 0.0065 - 0.0198i -0.0921 - 0.0294i -0.4941 - 0.1196i  0.0596 + 0.1438i
 0.0544 - 0.0416i  0.1509 + 0.0365i  0.3539 + 0.0597i  0.4285 + 0.2791i

The eigenvalues of the linear characteristic matrix

-6.5720 - 0.0000i
-1.7182 - 0.0000i
-0.6057 + 0.1953i
-0.0069 + 0.4620i
 0.0349 + 0.2765i
-0.6057 - 0.1953i
-0.0069 - 0.4620i
 0.0349 - 0.2765i

// end of analysis results //
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria VA 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey CA 93943-5002	2
3.	Advanced Rotorcraft Technologies 1685 Plymouth St. Suite 250 Mountain View CA 94043	1
4.	Dr. E.R. Wood Naval Postgraduate School Code AA/Wd Monterey CA 93943-5002	5
5.	Tony Cricelli, Code 31 Naval Postgraduate School Code AA/Tc Monterey CA 93943-5002	1
6.	Gary P. McVaney 4201 Crosstimbers Road Flowermound TX 75028	3