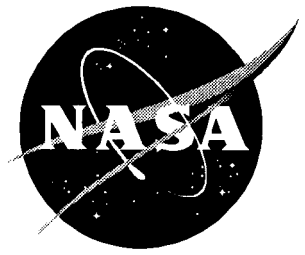
NASA Technical Memorandum 4790



PLATSIM: A Simulation and Analysis Package for Large-Order Flexible Systems (Version 2.0)

Peiman G. Maghami, Sean P. Kenny, and Daniel P. Giesy
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

December 1997

Chapter 1

Introduction

The software package PLATSIM provides time and frequency domain analysis of large-order generic space platforms. PLATSIM can perform open-loop analysis or closed-loop analysis with linear or nonlinear control system models. PLATSIM's generic control structure paradigm permits almost any type of control architecture to be implemented, for example: attitude control systems with or without flexible body controls, active isolation systems, and payload-instrument local control systems. In the time domain analysis, PLATSIM simulates the response of the space platform to disturbances and calculates the jitter and stability values from the response time histories. In the frequency domain analysis, PLATSIM calculates frequency response function matrices and provides the corresponding Bode plots. While PLATSIM was designed for analyzing space platforms, it only assumes that it has a finite element model of a structure that is being excited by force and/or torque inputs. Thus, any structure (e.g., aeronautical, automotive, structural, or mechanical) that fits this model can be analyzed by PLATSIM.

PLATSIM operates in the MATLAB® technical computing environment at MATLAB version 4.2 or 5.0. MATLAB, a product of The MathWorks, Inc., is a technical computing environment for high-performance numeric computation and visualization (ref. 1). PLATSIM also uses the Control System Toolbox and SIMULINK, which are additional products of The MathWorks, Inc. User input to PLATSIM is provided in the form of MATLAB readable data files and MATLAB function M-files.

PLATSIM allows the user to maintain a database of performance measurement outputs on the space platform and a database of disturbance scenarios. An individual run of PLATSIM can use all the performance outputs or a user-selected subset; the user selects one disturbance scenario for each run. Time domain analysis in PLATSIM provides on-screen plots of time histories at user-selected output locations (e.g., instrument boresight) due to user-selected disturbance scenarios, encapsulated PostScript files of these plots, tables of jitter-stability values due to disturbances for user-selected time window sizes, and files containing the time history data in either compressed or full form. Frequency domain analysis in PLATSIM provides on-screen Bode plots, encapsulated PostScript files of these plots, and files containing the plot data.

PLATSIM includes novel algorithmic features that provide efficiency in all calculations and, in some cases, are actually enabling technologies. PLATSIM exploits the particular form of sparsity (block diagonal with 2 by 2 blocks) of the plant matrices for both time analysis and frequency domain analysis. A new, original algorithm for the efficient computation of closed-loop (as well as open-loop) frequency response functions for large-order systems has been developed and is implemented within PLATSIM. This algorithm is an enabling technology for the analysis of large-order systems, in general, and flexible space systems, in particular. Furthermore, a novel and efficient jitter analysis routine which determines jitter and stability values from time simulations in a very efficient manner (speedup of three to four orders of magnitude in typical examples as compared to the brute force approach of sweeping minima and maxima) has been developed for and is incorporated in the PLATSIM package.

PLATSIM requires the following user inputs: modal data of the spacecraft as generated by finite element analysis, damping ratios for flexible modes, information about control actuators, measurement feedback sensors, and performance instrument outputs (e.g., boresight measurements), spacecraft disturbance data, and spacecraft control system model.

The program can be used in either a graphical user interface (GUI) mode or in a batch mode. The two modes differ in the way the required and optional flags and parameters that control execution are defined. In the GUI mode, the parameters and flags are chosen from pop-up MATLAB menus with a keypad and a mouse. In the batch mode, all flags and parameters are defined in MATLAB command lines, which can be placed in an ASCII input file.

Although PLATSIM was developed to analyze generic space platforms, the Earth Observing System EOS-AM-1 (ref. 2) is used throughout this manual as an example. Furthermore, several M-files and data files included in the PLATSIM distribution, some of which are listed in appendix A, correspond to the EOS-AM-1 spacecraft. These files are the spacecraft control system defined in `formscs.m`, the instrument types and connectivity data defined in `instdata.m`, the finite element data defined in `omega.mat` and `phi.mat`, the damping schedule defined in `mkdamp.m`, and the spacecraft disturbance data defined in `distdata.m` and its supporting routines. These files can serve as templates for the user-supplied files for other space platform applications.

PLATSIM Version 2.0 Enhancements

PLATSIM version 2.0 offers several substantial improvements in both capabilities and performance over the initial version 1.0 release. Improvements have been

made to all aspects of the software package, but the most significant improvements have been made to core analysis modules (time and frequency domain modules and the jitter analysis module) and to the graphical user interface. PLATSIM version 2.0 offers the following new features:

Nonlinear controllers are permitted.

- Seven nonlinear integration routines are available for controller state propagation.
- The user has direct access of controller states to facilitate solution integrity checking.

There is direct interface with SIMULINK controller models.

- Controller models may be SIMULINK block diagrams or generic S-functions.

Time-domain memory management has been greatly enhanced.

- Jitter calculations may be performed on-the-fly as opposed to postprocessing the entire time history matrices.

Time-domain analysis has been generalized.

- The control system can be modeled as continuous time or discrete time.
- Hybrid control system models, consisting of continuous and discrete states, are supported.
- Three methods are available for the propagation plant states.

Frequency domain analysis has been generalized.

- PLATSIM frequency domain analysis now has the capability of performing Bode analysis on 16 closed-loop and 9 open-loop transfer functions.
- The control system may be continuous time or discrete time.

The graphical user interface has been expanded.

- Complete graphical access to all new time and frequency domain features has been added, for example, transfer function definition (including input/output connections), controller implementation methods, plant integration methods, and others.
- Graphical interface for defining jitter windows is possible.
- Session file creation for convenient recovery of previous PLATSIM run-time variable setting can be used.

The manual begins with the description of input files and variables required by PLATSIM that include the finite element data, instrument and disturbance description data, and information on the control system (chapter 2). In chapter 3, the methodology and the assumptions behind the various analysis capabilities by the program are described, along with a discussion on the execution control parameters and options, which direct the type and extent of analysis performed by PLATSIM. Chapter 4 describes the graphical user interface execution mode as well as the batch execution mode. The various menus, buttons, and sliders in the graphical user interface that assign the execution control parameters, define or redefine the structural and controls models, or execute the program are discussed in detail. The description of the results or output of the various analyses performed by PLATSIM, in terms of file names, types, and contents are provided in chapter 5. These results and outputs include the time history plots, reduced time history data, jitter tables, gain and phase plots, and transfer functions. The run-time diagnostic messages generated by PLATSIM are listed in chapter 6. Appendixes A and B provide a listing of typical user-supplied routines required by the program. Appendix C provides an example of the type of routine required for the solution integrity check capability of PLATSIM. Finally, examples of typical PLATSIM outputs for a time domain analysis (including jitter analysis) and a frequency domain analysis are provided in appendix D.

Contents

Figures

Chapter 2

Input Files

The PLATSIM program requires user-supplied information from which it can construct the simulation or analysis model. PLATSIM requires

1. Information on the finite element model of the plant: modal frequencies, modal damping ratios, and mode shapes.
2. Information on the control system hardware and instrument outputs: location, direction, identification number, and types.
3. Information on disturbances: disturbance time histories, along with the corresponding location and direction of actions, and identifying names.
4. Information on the control system mathematical model: whether the controller is linear or nonlinear; for linear control systems, the control system matrices; for nonlinear control systems, the nonlinear controller function, along with discrete-state updates and output vector.
5. Information on user-defined and optional checking algorithm to be used in the validation of the integration of the nonlinear controller.

Specifically, PLATSIM requires users to provide the following input files:

- modal frequency data file (`omega.dat` or `omega.mat`)
- mode shape data file (`phi.dat` or `phi.mat`)
- modal damping schedule file (`mkdamp.m`)
- instrument data file (`instdata.m`)
- disturbances data file (`distdata.m`)
- disturbances files (arbitrarily named M-files)
- control system file (arbitrarily named M-file)
- solution check file (optional) (`solchk.m`)

Note: These files should be placed in MATLAB's directory path.

The remainder of this chapter is devoted to presenting the details of these input files.

Modal Frequency Data File

The finite element modal frequency data are to be provided in ASCII file `omega.dat`, or in MATLAB binary file `omega.mat`.

The ASCII file `omega.dat` contains one frequency per line:

$$\omega_1$$
$$\omega_2$$
$$\vdots$$
$$\omega_p$$

where p is the number of modes in the spacecraft model.

The MATLAB binary file `omega.mat` should contain a variable `omega` which is a $p \times 1$ vector containing the frequencies from `omega.dat` as described above.

The user running PLATSIM in its graphical user interface (GUI) mode has the opportunity, via the menu item "Modify Plant Model" under the "Options" button, to interactively change the plant model's frequencies. (See "Program Execution" on page 21.)

Mode Shape Data File

The finite element mode shape data are to be provided in ASCII file `phi.dat`, or in MATLAB binary file `phi.mat`. Use s to denote the number of grid points at which mode shape data are to be given in file `phi.dat` and label these grid points with grid numbers $N_1, N_2, \ldots, N_s$. Then `phi.dat` has $(p \times s)$ lines in the format shown in figure 1 on page 4, where X_j^i, Y_j^i, and Z_j^i are the modal translations in X, Y, and Z, and Θ_j^i, Φ_j^i, and Ψ_j^i are rotations about X, Y, and Z for mode i at grid point N_j. Note that grid points that are not needed to model instruments, the control system, or disturbances do not need to be included in file `phi.dat`. However, if they are included, care must be taken to ensure that each mode block contains data for exactly the same grid points.

Note: If p frequencies are defined in `omega.dat`, modal amplitudes for exactly p modes should be defined in `phi.dat`, and the i-th mode in `phi.dat` should correspond to frequency ω_i.

The MATLAB binary file `phi.mat` should contain a variable `phi`, which is a $(p \times s)$ by 7 matrix containing the grid numbers and mode shape displacements and mode slope displacements from `phi.dat` as described previously. Users are encouraged to create the binary file `phi.mat` to improve the efficiency of loading model data.

Modal Damping Schedule File

PLATSIM determines the damping ratios of the flexible spacecraft modes by calling the user-supplied

$$
\begin{array}{ccccccc}
N_1 & X_1^1 & Y_1^1 & Z_1^1 & \theta_1^1 & \phi_1^1 & \psi_1^1 \\
N_2 & X_2^1 & Y_2^1 & Z_2^1 & \theta_2^1 & \phi_2^1 & \psi_2^1 \\
 & & & & & & \\
N_s & X_s^1 & Y_s^1 & Z_s^1 & \theta_s^1 & \phi_s^1 & \psi_s^1 \\
\end{array}
$$

$$
\begin{array}{ccccccc}
N_1 & X_1^2 & Y_1^2 & Z_1^2 & \theta_1^2 & \phi_1^2 & \psi_1^2 \\
N_2 & X_2^2 & Y_2^2 & Z_2^2 & \theta_2^2 & \phi_2^2 & \psi_2^2 \\
 & & & & & & \\
N_s & X_s^2 & Y_s^2 & Z_s^2 & \theta_s^2 & \phi_s^2 & \psi_s^2 \\
\end{array}
$$

$$\vdots$$

$$
\begin{array}{ccccccc}
N_1 & X_1^p & Y_1^p & Z_1^p & \theta_1^p & \phi_1^p & \psi_1^p \\
N_2 & X_2^p & Y_2^p & Z_2^p & \theta_2^p & \phi_2^p & \psi_2^p \\
 & & & & & & \\
N_s & X_s^p & Y_s^p & Z_s^p & \theta_s^p & \phi_s^p & \psi_s^p \\
\end{array}
$$

Figure 1. Format of `phi.dat`.

MATLAB function `mkdamp` in file `mkdamp.m`. This function has the following form:

```
function [d]=mkdamp(omega)
```

The input argument `omega` is a vector containing the modal frequencies $\omega_1, \omega_2, \ldots, \omega_p$. The user must set the output `d` to a vector containing the corresponding damping ratios. Appendix A contains a sample file `mkdamp.m` for the EOS-AM-1 mission. This file defines a damping ratio of 0.2 percent for structural modes with frequencies under 15 Hz, a damping ratio of 0.25 percent for structural modes with frequencies between 15 Hz and 50 Hz, and a damping ratio of 0.3 percent for structural modes with frequencies over 50 Hz.

The user running PLATSIM in its graphical user interface (GUI) mode can use "Modify Plant Model" under the "Options" menu to change the damping schedule. (See "Program Execution" on page 21.)

Instrument Data File

The data defining the location, direction, type, and identification of the control system actuators and sensors, as well as the performance outputs of the system, must be provided by the user through a user-supplied MATLAB function `instdata` in a file named `instdata.m`. This function must have the following form:

```
function [act,mout,pout,instr]=instdata
```

The following is a description of the user-defined output parameters in file `instdata`. See appendix A for a sample file `instdata.m` for the EOS-AM-1 spacecraft.

Instrument data parameter `act`. Parameter `act` is used to define the location, direction, identification, and scaling of the control system actuator models. Parameter `act` is a matrix of four rows by as many columns as

necessary to represent the actuators. An example of act is given as

$$\texttt{act} = \begin{bmatrix} 625 & 1211 & 1212 & 827 \\ 4 & 5 & 5 & 6 \\ 300 & 550 & 550 & 400 \\ 1.0 & 0.5 & 0.5 & 1.0 \end{bmatrix} \rightarrow \begin{bmatrix} \text{Grid Point No.} \\ \text{Direction No.} \\ \text{Identification No.} \\ \text{Weighting/Scale Factor} \end{bmatrix}$$

- The first row of act defines the finite element grid points upon which the actuators are acting.
- The second row of act defines the directions at which the actuators are acting. Direction numbers are limited to 1 through 6, which correspond to x-translational, y-translational, z-translational, x-rotational, y-rotational, and z-rotational directions, respectively.
- The third row of act defines the user-defined positive identification numbers associated with the actuators. **Note: Each actuator must have a unique identification number. A single actuator may require more than a one-column description.**
- The fourth row of act defines the weighting-scale factors associated with the actuators. This factor allows for distributing the effect of an actuator within grid points. For example, in the act example, actuator 550 exerts its torque about the Y-axis on two grid points (1211 and 1212), each with a weighting of 0.5; that is, half the torque generated by this actuator is applied to grid point 1211 and the other half to grid point 1212. On the other hand, actuator 400 exerts its torque about the Z-axis solely at grid point 827. This factor also allows for independent scaling of the actuator output.
- Each column of act indicates a contribution of an actuator (force or torque) to a grid point. **Note: Each actuator must be represented by at least one column.**
- Note: Internally, PLATSIM reorders the actuator numbers according to the sorted (low-to-high) identification numbers provided in the third row of act. In the previous example, PLATSIM considers actuator 300 as the first actuator, actuator 400 as the second, and actuator 550 as the third and last actuator.

Instrument data parameter mout. Parameter mout is used to define the location, direction, identification, scaling, and measurement type of the control system sensor models. Parameter mout is a matrix of five rows by as many columns as necessary to represent the measurement outputs. An example of mout is given as

$$\texttt{mout} = \begin{bmatrix} 111 & 121 & 122 & 827 \\ 5 & 6 & 6 & 4 \\ 11 & 11 & 11 & 120 \\ 0.5 & 0.25 & 0.25 & 1.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} \text{Grid Point No.} \\ \text{Direction No.} \\ \text{Identification No.} \\ \text{Weighting/Scale Factor} \\ \text{Type Flag No.} \end{bmatrix}$$

- The first row of mout defines the finite element grid points whose responses contribute to the sensor measurements.
- The second row of mout defines the direction associated with the grid point responses that contribute to the sensor measurements. Direction numbers are limited to 1 through 6, which correspond to x-translational, y-translational, z-translational, x-rotational, y-rotational, and z-rotational directions, respectively.
- The third row of mout defines the user-defined positive identification numbers associated with the measurement sensors. **Note: Each measurement sensor must have a unique identification number. A single sensor may require more than a one-column description.**
- The fourth row of mout defines the weighting-scale factor associated with the sensors. This factor allows for the contribution of responses from multiple grid points and directions to a measurement signal. For example, in the mout example above, measurement sensor 11 measures a combined response from rotation about the Y-axis at grid point 111, rotation about the Z-axis at grid point 121, and rotation about the Z-axis at grid point 122, with corresponding weighting factors of 0.5, 0.25, and 0.25. In other words, the measurement signal of sensor 11 is comprised of half the rotational response (about the Y-axis) at grid 111, a quarter of the rotational response (about the Z-axis) at grid point 121, and a quarter of the rotational response (about the Z-axis) at grid point 122. Note that this weighting-scale factor also allows for independent scaling of the sensor output.
- The fifth row of mout defines the measurement sensor type flag numbers which take the value 0 for displacement and the value 1 for rate. PLATSIM is programmed only to handle displacement and rate sensors as input to the control system; however, acceleration feedback to the control system can be implemented by the addition of a judiciously selected prefiltering of a velocity signal.

- Each column of mout indicates a contribution of a grid point (in a specified direction and type) to a measurement sensor. **Note: Each measurement sensor must be represented by at least one column.**
- Note: Internally, PLATSIM reorders the sensor numbers according to the sorted (low-to-high) identification numbers provided in the third row of mout. In the previous example, PLATSIM considers sensor 11 as the first sensor and sensor 120 as the second and last sensor.

Instrument data parameter pout. Parameter pout defines the location, direction, identification, scaling, and measurement type of the performance output models. Parameter pout is a matrix of 5 rows by as many columns as necessary to represent the performance outputs. An example of pout is given as

$$\texttt{pout} = \begin{bmatrix} 101 & 314 & 567 \\ 5 & 6 & 4 \\ 1 & 113 & 1120 \\ 2.0626 \times 10^5 & 0.6188 \times 10^5 & 2.0626 \times 10^5 \\ 0 & 1 & 2 \end{bmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{bmatrix} \text{Grid Point No.} \\ \text{Direction No.} \\ \text{Identification No.} \\ \text{Weighting/Scale Factor} \\ \text{Type Flag No.} \end{bmatrix}$$

- Note: The rows and columns of parameter pout follow exactly the definitions given for that of parameter mout with one exception:

  The elements in the fifth row of pout**, which define the performance output types, may also take values of 2 for acceleration output.**
- A weighting factor is included with the prior data. One use of this weighting factor is for conversion of units. The performance output which is being calculated in radians is converted to arc seconds by multiplying the original weighting factors [1 0.3 1] by $(3600 \times 180)/\pi$ to obtain the fourth row in the example pout.

Instrument data parameter instr. Parameter instr is a matrix of character strings that provides names for the performance outputs. The number of rows in instr must be equal to the number of performance outputs, that is, the number of *distinct* entries in the third row of pout. An example of instr, corresponding to the previous example of pout, is given as

$$\texttt{instr} = \begin{bmatrix} \text{1|SWIR Pitch|SWIR|arc-sec} \\ \text{1120|VNIR Yaw Acceleration|VNIR|arc-sec/sec}^2 \\ \text{113|TIR Roll|TIR|arc-sec} \end{bmatrix}$$

- Each element of instr is a string consisting of three or four fields separated from each other by the vertical bar (|) character.
- The first field is an integer which must correspond to one of the performance output identification numbers defined in the third row of pout. **Note: Every identification number in that row must be referenced in** instr**.**
- The second field is one or more alphanumeric words which provide a unique name for the corresponding performance output. These names are used in PLATSIM on menu labels and to identify information in output tables and graphs. They are also used as parts of file names after embedded blanks have been replaced by underscores (_).

  No characters (except blanks that will be replaced by underscores) should be used that would confuse the computer operating system when used in a file name.

  For purposes of determining whether character strings are distinct, embedded blanks should be considered the same as underscores (e.g., "SWIR Pitch" and "SWIR_Pitch" should be considered the same).
- The third field is used to name the menu button under which this instrument will be found when running PLATSIM in the GUI mode. By using the same name in the third field of several entries, the entries will be grouped in a submenu under the same menu heading. In the example shown previously, all performance outputs are found under separate menus labeled: "SWIR", "VNIR", and "TIR".
- If the optional fourth field is present, PLATSIM will add it to the second field in generating the vertical legend of time history plots; this can be used, for example, to give the units of the output.

Disturbance Data

In the development of the PLATSIM package, it was assumed that spacecraft simulations and analyses would be performed for a variety of different disturbances scenarios and that each user would maintain a disturbance database. Easy interactive access to the disturbance database is provided in PLATSIM through a graphical user interface. The operation and capabilities of the disturbance GUI will be described in "Program Execution" on page 21.

The PLATSIM disturbance data are communicated to the main package through a user-supplied MATLAB function `distdata` in file `distdata.m`. This user-supplied data file provides a complete description of all spacecraft disturbance scenarios. A spacecraft disturbance scenario may consist of several disturbance events. A disturbance event is a force or torque time history acting on the platform.

The actual structure of the user disturbance data is described herein. **Note: The specific structure of the function call has been modified for version 2.0. The variable `tdflag` has been added to the input list.** The first line(s) of file `distdata.m` must have the following form:

```
function [dist,w,period,cnames,dnames,
instdat,mapping]
=distdata(casenum,tdflag)
```

The data the user returns in the first three parameters (`dist`, `w`, and `period`) will depend on `casenum`, while the data returned in the last four parameters (`cnames`, `dnames`, `instdat`, and `mapping`) must be the same, independent of the value of `casenum`. The data returned in `dist`, `w`, and `period` will model the one or more disturbance events of the disturbance scenario identified by `casenum`.

`distdata.m` input

`casenum`: Used to select a particular disturbance scenario from the user-provided database.

- If PLATSIM calls `distdata` with `casenum` = 0, the user need only return the last four parameters (`cnames`, `dnames`, `instdat`, and `mapping`).
- If `casenum` is input as a positive integer between the values of 1 and the number of rows in character matrix `dnames`, then all seven parameters should be returned with the values in the first three corresponding to the disturbance scenario whose name is in row `casenum` of `dnames`. See input variable `tdflag` for exceptions.
- Note: All other values of `casenum` are invalid.
- For all nonbatch mode operations, the value of `casenum` is set by the disturbance module GUI. In batch mode, the value of `casenum` is set in the batch mode input file.

`tdflag`: A string variable set to 'yes' for time-domain analysis and to 'no' for frequency domain analysis. This input is a version 2.0 modification which allows users to avoid unnecessary calculations when frequency domain analysis is requested.

- When `tdflag='no'`, output variables `w`, and `period` do not have to be defined.

`distdata.m` outputs

`dist`: A matrix of four rows by as many columns as necessary to describe one disturbance event represented by a column of the matrix `w`. An example of `dist` is given as

$$\texttt{dist} = \begin{bmatrix} 6325 & 121 & 112 \\ 1 & 5 & 5 \\ 100 & 200 & 200 \\ 1.0 & 0.6 & 0.4 \end{bmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{bmatrix} \text{Grid Point No.} \\ \text{Direction No.} \\ \text{Identification No.} \\ \text{Weighting/Scale Factor} \end{bmatrix}$$

- The first row of `dist` defines the finite element grid points upon which the disturbances are acting.
- The second row of `dist` defines the directions at which the disturbances are acting. Direction numbers are limited to the integers 1 through 6, which correspond to x-translational, y-translational, z-translational, x-rotational, y-rotational, and z-rotational directions, respectively.
- The third row of `dist` defines the user-defined positive identification numbers associated with the disturbances. **Note: Each disturbance event must have a unique identification number. A single disturbance event may require more than a one-column description.**
- The fourth row of `dist` defines the weighting-scale factor associated with the disturbances. This factor allows for distributing the effect of a disturbance across multiple grid points. For example, in the `dist` illustration, disturbance 200 exerts its torque about the Y-axis on two grid points (121 and 112), with weighting factors of 0.6 and 0.4, respectively. In other words, 60 percent of the torque exerted by this disturbance is applied to grid point 121, and the other 40 percent is applied to grid point 112. On the other hand, disturbance 100 exerts its force in the x-direction solely at grid point 6325. This factor also allows for independent scaling of the disturbance levels.

- Each column of dist indicates a contribution of a disturbance (force or torque) to a grid point. **Note: Each disturbance must be represented by at least one column.**
- Note: Internally, PLATSIM reorders the disturbance numbers according to the sorted (low-to-high) identification numbers provided in the third row of dist. In the previous example, PLATSIM considers disturbance 100 as the first disturbance event and disturbance 200 as the second and last disturbance event.

w: A matrix of force and/or torque disturbance time history profiles. The number of rows in w is the number of time steps, and the number of columns in w is the number of disturbance events. This output is optional for frequency domain analysis.

period: A disturbance sampling period, which is also used as the update rate of the simulation solution. Note that the same discretization period is used for all events in a single disturbance scenario. This output is optional for frequency domain analysis.

cnames: A string matrix that contains the menu heading labels for the disturbance GUI. The disturbance GUI will create a pull-down menu button for each row in cnames. The entries in cnames are used for menu button labeling only and will not appear in output documentation.

dnames: A string matrix of disturbance scenario case names. The rows in dnames are used for labeling pull-down menu items in the disturbance GUI, for labeling plot figures, and for constructing some PLATSIM output file names.

- If the PLATSIM user selects a disturbance scenario from the disturbance GUI that was in row "j" of dnames, then PLATSIM will call distdata with casenum = 'j'. The user who wrote distdata.m must return the disturbance case which corresponds to the name in row "j" of dnames by testing casenum and acting on the value found there.
- Individual disturbance scenarios may be "commented out" by editing distdata.m to return the corresponding row of dnames with an asterisk (*) as the first character. The scenario will still show on the disturbance GUI menu but in a distinctive grey type, and it will not be selectable.

instdat: The variables instdat and mapping together define which dnames row entries will appear under a particular cnames disturbance GUI pull-down menu button. The vector instdat may be any length, and its entries may be in any numerical order.

- **Note: The entries in** instdat **must correspond to valid row numbers in the** dnames **string matrix.**

mapping: The vector mapping must contain non-negative integers and define the partitioning of the instdat vector required to map the entries in dnames to the disturbance GUI labels created from cnames.

- The number of entries in the mapping variable must equal the number of rows in cnames.
- The sum of all entries in mapping must equal the number of elements in instdat. For example, if instdat has 30 elements, the MATLAB command sum(mapping) must also equal 30.

To illustrate the use of the parameters cnames, dnames, instdat, and mapping, suppose the file distdata.m contains these commands:

```
cnames = str2mat('Calibration','Tracking');

dnames = str2mat('Solar Array','Telescope Cal','Antenna Cal');

dnames =str2mat(dnames,'Telescope Track','Antenna Track','Antenna Sweep');

instdat=[1 2 3 1 4 5 6];

mapping=[3 4];
```

This disturbance database should contain six disturbance cases whose names are given in the six lines of the dnames matrix. When PLATSIM is run in GUI mode, there will be two buttons on the disturbances menu with labels Calibration and Tracking. The Calibration button will invoke a submenu showing three entries (mapping(1) = 3) which will be taken from the first three rows of the dnames matrix. The first three entries of instdat are [1 2 3]. The Tracking button will invoke a submenu showing four entries (mapping(2) = 4) which will be taken from rows 1, 4, 5, and 6 of dnames. The next four entries of

instdat are [1 4 5 6]). Note that the same disturbance scenario (Solar Array in this example) may be included in more than one category.

Control System File

The control system model is accessed by a user-supplied MATLAB function whose name is in the string variable nlcon. Depending on the linear or nonlinear nature of the controller, the user-supplied file may be either a PLATSIM version 1.0 style, that is, a function that returns a linear time-invariant state space representation, or a MATLAB S-function. It should be emphasized that any linear or nonlinear control system can be represented, such as a spacecraft attitude control system, a flexible body control for payload isolation or disturbance rejection, robotic control, and many others.

Linear and time-invariant control system. If the control system is linear and time invariant, information on the control system may be provided in one of two ways:

1. A user-supplied MATLAB function must provide the matrices that define the control system in state-space form. This form was required by PLATSIM version 1.0.
   - The first line of this type controller file must have the form function[ascs,bscs,cscs,dacs] =xxyyzz where the output parameters ascs, bscs, cscs, and dacs denote the state matrix, the input influence matrix, the output influence matrix, and the feed-through matrix of the control system, respectively. The name of the controller file, xxyyzz.m in this example, is arbitrary and is passed to PLATSIM in the execution control parameter (p. 16 ff) nlcon as nlcon= 'xxyyzz'.
   - An example file named formscs.m is given in appendix A. This file corresponds to a simplified continuous-time model of the attitude control system of the EOS-AM-1 spacecraft.
2. A user-supplied MATLAB S-function may be provided in one of three ways:
   - As a user-supplied MATLAB function which follows the input/output requirements for S-functions. Type help sfunc at the MATLAB prompt for a complete description of S-functions. An example of a user-supplied S-function corresponding to a continuous-time model of the attitude control system of the EOS-AM-1 spacecraft is given in appendix B.
   - A user-supplied SIMULINK block diagram. If the controller exists as a block in a SIMULINK simulation file, a MATLAB S-function may be obtained by isolating the controller block, by changing its input and output connections to Inports and Outports, respectively, and by saving the updates.
   - A user-supplied C or FORTRAN function which follows the input/output requirements of S-functions. This function is subsequently converted to a MATLAB "mex" file through the MATLAB fmex or cmex routines.

Note: Controller file names are arbitrary and can be modified from default values during program execution in GUI mode or by setting string parameter nlcon in the batch mode.

General assumptions and comments:

1. The linear controller can be either a continuous or a digital controller. Note that the user declares the continuous or digital nature of the controller in flags used during the program execution.
2. It is assumed that the control system is implemented with a negative feedback connection.
3. It is assumed that the number of control system inputs is equal to the number of measurement outputs of the plant. Moreover, there is a straight connection between the measurement outputs of the plant and the inputs of the control system.
4. It is assumed that the number of control system outputs is equal to or greater than the number of control inputs to the plant. However, it is assumed that only control system outputs 1 through the number of plant inputs are used in the feedback connection. Moreover, the feedback connection is a straight connection.

Nonlinear or time variant control system. If the controller model is nonlinear or time variant, the necessary information on the control system model is provided solely by a user-supplied MATLAB S-function which can be defined in one of three ways, as discussed in the previous section. An example of a nonlinear control system corresponding to a continuous-time model of the attitude control system of the EOS-AM-1 spacecraft with reaction wheel stiction is provided in user-supplied MATLAB S-function stiction.m.

General assumptions and comments:

1. The nonlinear controller may have both continuous and discrete states. (See S-function definitions.)
2. It is assumed that the control system is implemented with a negative feedback connection.

3. It is assumed that the number of control system inputs is equal to the number of measurement outputs of the plant. Moreover, there is a straight connection between the measurement outputs of the plant and the inputs of the control system.
4. It is assumed that the number of control system outputs is equal to or greater than the number of control inputs to the plant. However, it is assumed that only control system outputs 1 through the number of plant inputs are used in the feedback connection. Moreover, the feedback connection is a straight connection.

Solution Check File

PLATSIM provides the capability to accept or reject a solution of the nonlinear controller integration through the use of the user-supplied `solchk.m` input file. By using this function (given the information on the previous as well as the current controller states and outputs), the user can implement a checking of the solution; that is, if the conditions are not met, the current solution is rejected, and a smaller recommended step size is provided by `solchk.m`. The first line of file `solchk.m` must have the form

```
function [han]=solchk(z2,u2,z1,u1,h)
```

where the input parameters `z2`, `u2`, `z1`, `u1`, and `h` are defined as

`z2`: current controller state vector

`u2`: current controller output vector

`z1`: previous controller state vector

`u1`: previous controller output vector

`h`: integration step size

The output parameter `han` must be set to `h`, if the user accepts the solution. If the solution is rejected, `han` should contain the recommended new integration step size, which must be less than `h`.

Nonlinear Reaction Wheel Friction

As an example of controller solution checking, consider the problem of nonlinear reaction wheel friction model (stiction); see figure 2, "Reaction wheel friction characteristics."

Nonstiction condition. If the wheel speed is nonzero, or if the magnitude of the applied torque is greater than or equal to the stiction torque (static friction torque), the reactive torque is given as

$$T_{react} = -J_w \dot{\omega}_w$$

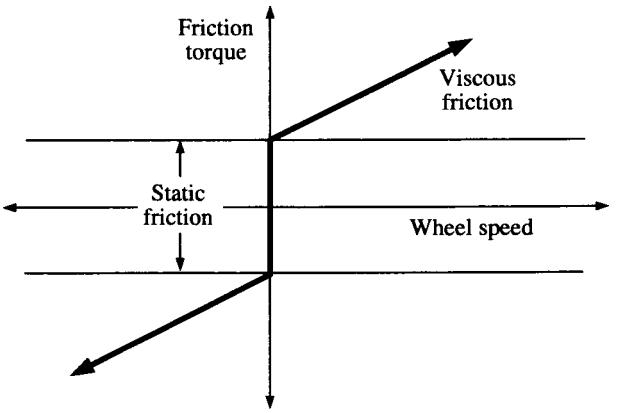


Figure 2. Reaction wheel friction characteristics.

where T_{react} denotes the reactive torque (output torque), J_w represents the inertia of the reaction wheel, and $\dot{\omega}_w$ represents the wheel speed rate.

Stiction condition. If the wheel speed is zero and the magnitude of applied torque is also less than or equal to the stiction torque (see fig. 3, "Stiction Condition"), the reactive torque is given as

$$T_{react} = 0$$

The wheel speed would stay at zero until the applied torque becomes larger then the stiction torque (static friction torque).

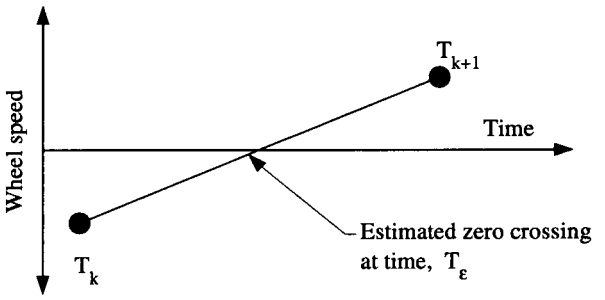


Figure 3. Stiction condition.

From the previous equations, it is obvious that the reactive torque is not a continuous function of $\dot{\omega}_w$. Moreover, it is quite possible that in the process of advancing the states of the nonlinear control system, which includes the nonlinear model of the reaction wheel, an integration step may come about wherein (a) a zero-crossing of the wheel speed occurs, and (b) the applied torque at the time of wheel speed zero-crossing is

less than or equal to the stiction torque; that is, stiction has occurred. Thus, the current solution of the integration is not valid because it did not model the behavior of the wheel properly by predicting and taking the stiction of the wheel into account. For example, if from the states and outputs it is determined that a stiction condition may have occurred in the vicinity of point "s", at time T_ε from the previous step, the user may recommend retrying the integration with a step size of T_ε or some fraction of T_ε.

The solution check is provided to give the user the capability of altering integration step sizes and state values to accommodate for discontinuities associated with some nonlinear devices. A solution check file for the stiction of reaction wheels in the EOS-AM-1 example is provided in appendix C. Note that in this example file, the checks on the zero crossing of the wheel speed and the friction torque crossing of the applied torques are performed by using linear interpolation between the previous and current values.

Chapter 3

Analysis Methodology and Options

PLATSIM includes novel algorithmic features that provide efficiency of all calculations while constituting enabling technology in some cases. PLATSIM exploits the particular form of sparsity of the plant matrices both in the continuous-time and discrete-time forms, as used by the time domain analysis and the frequency domain analysis.

Time Domain

In the time domain analysis, PLATSIM performs a time simulation of the system by using the user-provided input data files (which define the model) together with the execution control parameters selected by the user. This simulation is optionally followed by a jitter analysis at the discretion of the user. PLATSIM assumes the following with regard to the plant and control system models:

1. The plant model is assumed to be linear with model information provided, as described in chapter 2.
2. The control system model can be either linear or nonlinear. See chapter 2 for the proper input file format.
3. The control system can be modeled in continuous time or in discrete time.
4. Hybrid control system models, consisting of continuous-time and discrete-time modules, are also permitted (in the nonlinear analysis only).
5. The information needed to declare the type of control system is provided by the user through the linearity flag (execution control parameter `linflag`) and the continuous-time flag (execution control parameter `ctflag`). See "Execution Control Parameters" on page 16.

Once the plant and the control system models have been defined, PLATSIM allows for two types of time analysis, linear and nonlinear. The user defines the type of analysis desired through the implementation flag (`impflag`). See "Execution Control Parameters" on page 16. PLATSIM takes advantage of the sparsity of the spacecraft dynamic model system matrix to perform the linear or nonlinear analysis very efficiently. The reader should refer to reference 3 for a detailed description of this sparsity. To take advantage of the sparsity, PLATSIM assumes that the measurement outputs of the plant, which are inputs to the control system, are processed through a sample and hold analog-to-digital conversion; this is not a restrictive assumption because it is a routine procedure in almost all modern practical applications.

Linear analysis. When using the previously mentioned assumption regarding the sample and hold applied to control inputs, the following applies to linear analysis:

1. If the control system model is discrete time with a sampling period `tsc`, then time simulation is performed through algebraic propagation of the plant and the controller states, with the output updates occurring at the appropriate times (every `tsd` for the plant and every `tsc` for the controller).
2. If the control system model is continuous time, the controller is still implemented in a discrete-time form. However, the sampling period `tsc` is chosen automatically by PLATSIM to be small enough; that is, the controller, for all practical purposes, is continuous time. Then, time simulation is performed through algebraic propagation of the plant and the controller states, with the output updates occurring at the appropriate times (every `tsd` for the plant and every `tsc` for the controller).
   - The controller sampling rate is chosen to be one decade larger than the crossing frequency with the line 20 dB below the H-infinity norm (of the controller). A 40 dB line is used if the roll-off rate is less than 20 dB/decade.
   - Users may define their own value for `tsc` (the value must be different from the default) to override its automatic computation. (See "Execution Control Parameters" on page 16.)
3. The disturbances are interpolated by using linear interpolation as necessary.

Nonlinear analysis. When using the assumption regarding the sample and hold applied to control inputs, the following applies to nonlinear analysis:

1. In the nonlinear analysis, the sampled measurement outputs are used in one of seven nonlinear integration routines to propagate the states of the controller. These routines are as follows:
   - Euler integration algorithm
   - Second-order Runge-Kutta-Heun integration algorithm (with no error control)
   - Second-order Runge-Kutta integration algorithm (with third-order error control)
   - Third-order Runge-Kutta algorithm by Bogacki-Shampine (with error control)
   - Fourth-order Runge-Kutta-Hall integration algorithm (with error control)
   - Fifth-order Runge-Kutta integration algorithm by Fehlberg (with error control)
   - Modified Rosenbrock algorithm for stiff systems (with error control)

The desired integration routine is selected from the first element of vector `imthd`. (See "Execution Control Parameters" on page 16.)

2. Three methods are available for the propagation of the plant states:
   - Zero-order hold
   - First-order hold
   - Fourth-order Runge-Kutta-Hall integration algorithm (with error control)

   The desired integration routine is selected from the second element of vector `imthd`. (See "Execution Control Parameters" on page 16.)

3. If the control system does not have discrete states, and if `tsc` is not defined by the user, then `tsc` is set equal to `tsd` by PLATSIM to perform controller state propagation, as needed.

4. Note: If the control system has discrete states, then `tsc` should be defined by the user.

5. The user is provided with the ability to accept or reject a solution of the nonlinear controller integration through the use of the user-supplied `solchk.m` input file. (See "Solution Check File" on page 10.) The idea is that given information on the previous, as well as the current controller states and outputs, the user can implement a desired checking of the solution; that is, if the conditions are not met, the current solution is rejected, and a smaller recommended step size is provided by `solchk.m`.

6. The disturbances are interpolated by using linear interpolation as necessary.

Jitter analysis. The calculation of jitter by PLATSIM depends on a user-provided window (time interval). Each performance output time history is scanned from beginning to end by moving this window along it. At each window position in the scan, the maximum peak-to-peak variation of the portion of the time history within the window is noted. The biggest of these is observed as the window moves to scan the entire time history; this is the measure of jitter in this time history corresponding to this window.

In typical applications of this technology, time histories can contain on the order of a hundred thousand to a million points, and the various windows used in the analysis (there are typically several of them) can cover anywhere from a few hundred points to the entire time history. If jitter is calculated in the obvious way (by placing the window at the beginning of the time history, finding the peak-to-peak variation under the window, moving it one time step, finding the new peak-to-peak variation, and repeating until the end of the time history is reached), then about $k(n-k)$ references are being made to points in the time history for each window, where n is the number of points in the time history and k is the number of points under the window. The number of references to time history points can constitute a serious computational burden.

PLATSIM uses an alternate algorithm (ref. 4), which has been shown to improve computational speed by three to four orders of magnitude, when compared with the obvious way of calculating using typical engineering problems. A single pass is made through the time history. If the jitter value is desired for more than one window, all are done in the same pass. A running tally is kept of jitter attributable to points which have passed out of the moving windows, and lists are kept of points still under any of the windows which might be significant in future parts of the jitter calculation. This single pass calculation can be done by making only $(ma+b)n$ references to points in the time history vector, where n is still the number of points in the time history, m is the number of windows for which jitter is being calculated, and a and b are constants which are reasonably small and are independent of the actual lengths of the windows. Empirical observation has shown that a is about $0.3(a+b)$, which implies that the time to calculate jitter for each window after the first is about 30 percent of the time that is taken for the first.

Note: The user should refer to the section "Execution Control Parameters" at the end of this chapter for a complete listing and description of time domain and jitter analyses parameters.

Frequency Domain

In frequency domain analysis, PLATSIM computes the frequency response function matrix from a set of user-defined inputs to a set of user-defined outputs for a set of frequency points selected by the user. A new, original algorithm for the efficient computation of closed-loop (as well as open-loop) frequency response functions for large-order systems has been developed and is implemented within PLATSIM. This algorithm exploits the particular form of sparsity (block diagonal with 2 by 2 blocks) of the plant state matrices in both the continuous and discrete forms used by the frequency analysis, as well as the sparsity in the control input and disturbances influence matrices in the continuous form. This algorithm is an enabling technology for the analysis of large-order systems, in general, and flexible space systems, in particular. A detailed description of a sample algorithm is provided in reference 5. Figures 4 and 5 on page 15 are block diagrams of open-loop and closed-loop systems with the various input and output types identified.

PLATSIM frequency domain inputs

1. `Ref`, Attitude or rate reference commands: valid with both open-loop and closed-loop analyses
2. `D`, Disturbances at the plant input: valid with both open-loop and closed-loop analyses
3. `W`, External disturbances: valid with both open-loop and closed-loop analyses
4. `M`, Measurement noise: valid only with closed-loop analysis

PLATSIM frequency domain outputs

1. `Err`, Tracking errors: valid only with closed-loop analysis
2. `U`, Control input: valid with both open-loop and closed-loop analyses
3. `Yper`, Performance outputs: valid with both open-loop and closed-loop analyses
4. `Y`, Measurement outputs: valid with both open-loop and closed-loop analyses

PLATSIM makes the following assumptions and restrictions in frequency domain analysis:

1. Control implementation must be linear (parameter `impflag` set to 'yes').
2. Control system may be continuous time or discrete time.
3. Input or output types cannot be combined; for example, input: measurement noise and input: external disturbances cannot be analyzed simultaneously.

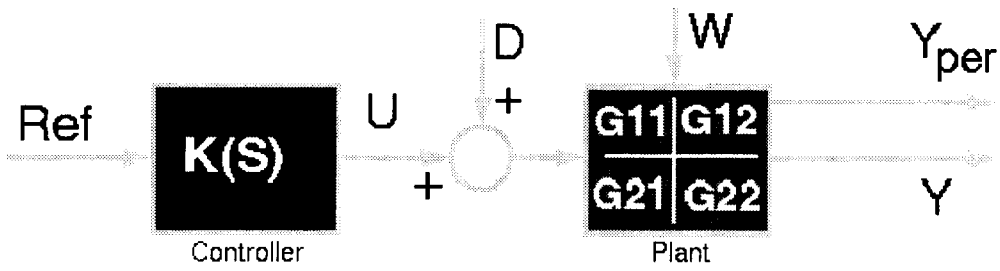


Figure 4. Open-loop block diagram.

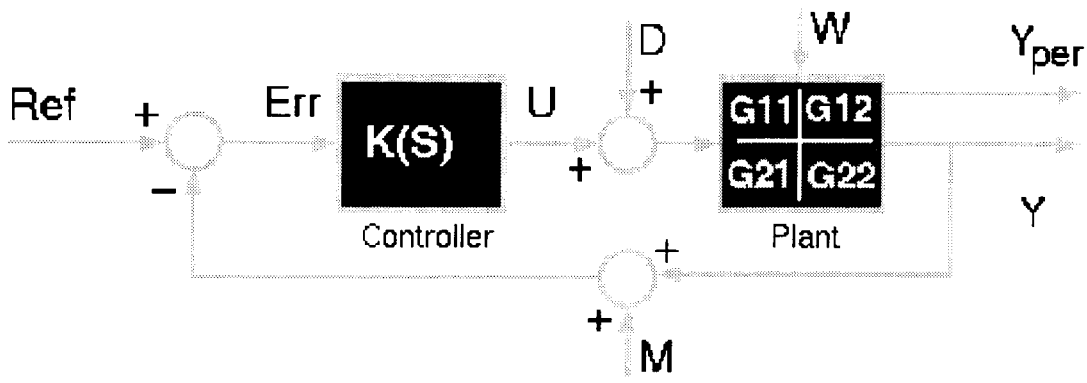


Figure 5. Closed-loop block diagram.

4. The smallest frequency used in the analysis must not approach zero because the frequency analysis problem becomes ill conditioned in the presence of rigid-body modes (or pure integrators) as frequency approaches zero.

5. Each flexible mode either must have positive damping, or its frequency must not be included in the frequency vector used for the analysis.

Execution Control Parameters

There are 38 execution control parameters within PLATSIM. These are MATLAB work space variables that are used to define the type, extent, methods, and options of analysis required by the user, as well as to control the output of the program. Some of these parameters have meaning to both time and frequency domain analysis, some only to time domain, and some only to frequency domain. Most of these parameters can be defined easily by using an interactive PLATSIM GUI (chapter 4). Execution control parameters can also be set by using commands from a prewritten script to run PLATSIM in the background. (See "Batch Mode" on page 32.) PLATSIM assigns a default value to all execution control parameters, with the exception of the parameter `casenum`. If PLATSIM is running in GUI mode, `casenum` is set by user responses to the "Disturbances" menu. In batch mode, `casenum` must be set before PLATSIM is called. The default values for most execution control parameters are specified in the user-changeable file `defaults.m`. Users are encouraged to customize `defaults.m` to fit their particular analysis needs. See appendix A for a listing of `defaults.m`.

Table 1. Specifications for Individual Execution Control Parameters

Name	Usage	Type	Description
casenum	both	integer	This variable is set to the number of the disturbance scenario to be used. PLATSIM will pass this value to user-supplied MATLAB function `disdata.m`. Can be set from GUI mode.
clflag	both	string	This flag controls whether closed-loop or open-loop analysis is performed. For closed-loop analysis, set `clflag='yes'`, and for open-loop analysis, set `clflag='no'`. Can be set from GUI mode.
desint	both	integer vector	Determines which performance outputs will be used. If left undefined, all will be used. To use, set "`desint`" to a vector of instrument identification numbers. (These numbers appear in the third row of the `pout` matrix returned by user-supplied MATLAB function `instdata`.) Can be set from GUI mode.
impflag	both	string	Defines whether linear or nonlinear implementation of the control system is used. For a nonlinear implementation of the control system, set `impflag='no'`. Can be set from GUI mode. The necessary control system files are automatically created by PLATSIM; that is, 1. If the controller is linear, `impflag` is set to '`yes`', and an S-function is provided for the control system, an equivalent state space representation is created, and `nlcon` is set to `formscs.m`. A file `formscs.mat` is also created to hold the controller data. 2. If the controller is linear, `impflag` is set to '`yes`', and a state space representation is provided for the control system; no action is taken. 3. If the controller is linear, `impflag` is set to '`no`', and a state space representation is provided for the control system, an equivalent S-function is created, and `nlcon` is set to `platc_*.m`, where the asterisk is replaced by the lowest integer that defines a unique file name in the current directory.

Table 1. Continued

Name	Usage	Type	Description
impflag	both	string	4. If the controller is nonlinear, `impflag` is set to 'no', and an S-function is provided for the control system; no action is taken. 5. If the controller is nonlinear, `impflag` is set to 'yes', and an S-function is provided for the control system, an equivalent state space representation is created, and `nlcon` is set to `formscs.m`. A file `formscs.mat` is also created to hold the controller data.
linflag	both	string	If the control system is linear, set `linflag='yes'`. For nonlinear control systems, set `linflag='no'`. Can be set from GUI mode.
phold	both	real scalar	A mnemonic for *plot hold*, `phold` is the number of seconds a time history, Bode plot, or jitter analysis table will remain on-screen before being cleared for the next plot. See note 1 at the end of table.
pltflag	both	string	In time domain analysis, `pltflag='yes'` causes reduction of time history data for plotting and writing of MAT-file with reduced time histories. In both analyses, causes plots (time history or Bode) to be displayed on-screen. Disable by setting `pltflag='no'`. Can be set from GUI mode.
prtflag	both	string	If `pltflag` is set to 'yes', this causes the encapsulated PostScript forms of the plots to be written to files. Disable by setting `prtflag='no'`. Can be set from GUI mode.
nlcon	both	string	The name of the MATLAB M-file which contains the S-function that describes the linear or nonlinear controller. Can be set from GUI mode.
runmode	both	string	Determines what interface mode PLATSIM runs in. Set `runmode='batch'` before entering the `platsim` command to run in batch mode. Only the first character of `runmode` is significant and may be of either case.
nmode	both	integer scalar, or vector	Which structural modes to use in the analysis. The value `0` means "all modes". A positive integer `n` means "modes 1 through n". A negative integer `-n` means "only mode n". A MATLAB vector such as `[1:6,10,13:16]` means "use exactly the mode numbers in the vector". Can be set from GUI mode.
tdflag	both	string	For time domain analysis, set `tdflag='yes'`. Set `tdflag='no'` for frequency domain analysis. Can be set from GUI mode.
tsc	both	real scalar	Parameter `tsc` defines the controller sampling period. If the controller is discrete time, the user should provide the sampling time of the controller (or the default value would be used). The value of `tsc` for a continuous time controller is based upon the following: if the user has provided a preferred value, it will be used; otherwise, a sufficiently small sampling time is automatically computed and used. Can be set from GUI mode.

Table 1. Continued

Name	Usage	Type	Description
abstolc	time	real scalar	Absolute error tolerance used in the nonlinear integration of the controller. See note 1 at end of table.
abstolp	time	real scalar	Absolute error tolerance used in the nonlinear integration of the plant. See note 1 at end of table.
ctflag	both	string	If `ctflag` is set to 'yes', the controller provided by the user is assumed to be continuous (no discrete states). If the controller has any discrete states, then set `ctflag='no'`. Can be set from GUI mode. Note: PLATSIM assumes that the input to the controller is discrete (sampled and held for one sampling period). See "Time Domain" on page 13.
imthd	time	integer vector	Determines the solution techniques used in the propagation of the controller and plant states. When controller implementation is nonlinear, `impflag` is set to 'no'. Parameter `imthd` is a 2-element integer vector, with the first element identifying the solution technique used in the propagation of the controller states and the second element identifying that used for the plant. Can be set from GUI mode. The options for the first element of `imthd` are 1. Euler integration algorithm 2. 2nd-order Runge-Kutta-Heun integration algorithm (with no error control) 3. 2nd-order Runge-Kutta integration algorithm (with 3rd-order error control) 4. 3rd-order Runge-Kutta algorithm by Bogacki-Shampine (with error control) 5. 4th-order Runge-Kutta-Hall integration algorithm (with error control) 6. 5th-order Runge-Kutta integration algorithm by Fehlberg (with error control) 7. Modified Rosenbrock algorithm for stiff systems (with error control) The options for the second element of `imthd` are 0. zero-order-hold integration 1. first-order-hold integration 2. fourth-order Runge-Kutta-Hall integration algorithm (with error control)
jtrflag	time	string	Causes jitter analysis to be performed on time histories. Disable by setting `jtrflag='no'`. Can be set from GUI mode.
lowmemflag	time	string	If `lowmemflag` is set to 'yes', and time-domain analysis or jitter analysis is requested, PLATSIM implements a memory efficient simulation and jitter analysis algorithm which reduces the required memory drastically, while paying a slight penalty on computational efficiency. Can be set from GUI mode.
maxiter	time	integer scalar	Maximum number of iterations allowed for convergence at each step of the nonlinear integration. See note 1 at end of table.

Table 1. Continued

Name	Usage	Type	Description
multflag	time	string	If `multflag='no'`, a separate time simulation for each event in the disturbance scenario is performed. To simulate the effect of all disturbances simultaneously, set `multflag='yes'`. Can be set from GUI mode.
nplpts	time	real scalar	The parameter `nplpts` is used in reducing the time histories for plotting. The data points to be plotted are divided into `nplpts` nonoverlapping groups of consecutive points, the groups containing, as nearly as possible, the same number of points. See note 1 at end of table.
options	time	real vector	`options` is a 3-element vector that defines the most commonly used parameters in the nonlinear integration. The first element defines the relative integration error tolerance, the second element defines the minimum step size allowed, and the third defines the maximum step size allowed. Can be set from GUI mode.
pmflag	both	string	If `pmflag='yes'` a performance meter is shown during simulation and that indicates what percentage of the calculation has been completed. Can be set from GUI mode.
saveflag	time	string	To save full, that is, not reduced time histories, set `saveflag='yes'`. Can be set from GUI mode.
solchkflag	time	string	`solchkflag` is valid only with the nonlinear controller implementation in time domain analysis. If `solchkflag` is set to 'yes', the nonlinear integration solution is checked against a set of user-defined conditions (provided in a MATLAB function in file `solchk.m`; see "Solution Check File" on page 10); that is, if the conditions are met, the solution is accepted; otherwise, a smaller integration step is suggested by the user (as the output of `solchk.m`). Can be set from GUI mode.
tclip	time	real scalar	The parameter `tclip` is used for clipping the time histories. If the value entered is not zero, any data point corresponding to a time before `tclip` will be removed; that is, it will not be used for jitter computation. Units of `tclip` must match the units of parameter `period` returned by user-supplied MATLAB function `distdata`. This option is useful in the jitter analysis of steady-state disturbance sequences. May be set from GUI mode.
window	time	real vector	A vector whose nondecreasing and positive elements define the time windows to be used in the jitter analysis. `window` must be defined if jitter analysis is requested. May be set from GUI mode.
bfrax	frequency	string	If `bfrax` is set to 'yes', the Bode plot frequency axis units are set to rad/sec; otherwise, they are set to Hz. Can be set from GUI mode.
bmagax	frequency	string	If `bmagax` is set to 'yes', the Bode magnitude plot axis is presented in decibels; otherwise, it is presented in a logarithmic scale. Can be set from GUI mode.
bodemthd	frequency	integer vector	Determines the types of inputs and outputs for which a frequency response function is desired. `bodemthd` is a 2-element integer vector, with the first element identifying the type of input and the second identifying the type of output. Can be set from GUI mode. The options for the first element of `bodemthd` are 1. attitude or rate reference commands (open loop and closed loop)

Table 1. Concluded

Name	Usage	Type	Description
bodemthd	frequency	integer vector	2. disturbances at the plant input (open loop and closed loop) 3. external disturbances (open loop and closed loop) 4. measurement noise (closed loop only) The options for the second element of `bodemthd` are 1. tracking errors (closed loop only) 2. control inputs (open loop and closed loop) 3. performance outputs (open loop and closed loop) 4. measurement outputs (open loop and closed loop)
desinti	frequency	integer vector	Determines which elements from the type selected (see "Frequency Domain" on page 14) will be used as the inputs for the frequency response analysis. If undefined, all inputs will be used. To use, set `desinti` to a vector of identification numbers of the desired elements. Can be set from GUI mode.
desinto	frequency	integer vector	Determines which elements from the type selected (see "Frequency Domain" on page 14) will be used as the outputs for the frequency response analysis. If undefined, all outputs will be used. To use, set `desinto` to a vector of identification numbers of the desired elements. Can be set from GUI mode.
frqflag	frequency	string	If `frqflag` is set to 'yes', model frequencies and, if defined, a user-prescribed frequency vector, provided in MATLAB variable `usrfrq`, are added to the frequency vector used in the frequency domain analysis. Can be set from GUI mode.
il	frequency	real scalar	The smallest frequency at which frequency domain analysis will be done is 10^il. Can be set from GUI mode.
iu	frequency	real scalar	The largest frequency at which frequency domain analysis will be done is 10^iu. Can be set from GUI mode.
npts	frequency	real scalar	The number of frequency points at which frequency domain analysis will be done is `npts`. Can be set from GUI mode.
usrfrq	frequency	real vector	`usrfrq` is a MATLAB vector which contains the user-defined frequencies that are also used in the frequency domain analysis. `frqflag` must be set to 'yes' for this parameter to be used by PLATSIM. Can be set from GUI mode.

Note 1. To run PLATSIM in GUI mode with a nondefault value for this parameter, it must be set by MATLAB assignment statement prior to invoking `platsim`.

Chapter 4

Program Execution

Overview

As already stated, PLATSIM operates in the MATLAB technical computing environment of MATLAB version 4.2. The MathWorks, Inc. products, Control System Toolbox and SIMULINK, must also be available. In order to run PLATSIM, one must first start MATLAB. Furthermore, MATLAB must have access to the PLATSIM source code and the user-supplied data and M-files. Source code access is typically defined by a `startup.m` file that sets the MATLAB path variable automatically on initialization.

GUI Mode

A graphical user interface (GUI) has been developed for PLATSIM. PLATSIM's GUI uses MATLAB's Handle Graphics available in MATLAB version 4 or 5. The objective of this GUI is to provide the user with a convenient and intuitive way to set PLATSIM execution control parameters. A screen image of the PLATSIM GUI is shown in figure 6.

Top-level graphical interface. The top-level interface consists of five menu buttons (MATLAB `uimenu` functions) and two slider controls (MATLAB `uicontrol` functions). Additional, low-level, supporting interfaces have been developed to provide complete graphical access to all PLATSIM simulation and analysis features. See "Supporting graphical interfaces" on page 25.

The Menu-Driven (GUI) Mode is the default execution mode and is invoked by typing `platsim` in the MATLAB command window. Normal execution of the PLATSIM GUI will create a file called `platsim.mat`, which contains parameters associated with various GUI functions. A complete description of all PLATSIM GUI features is given in this chapter.

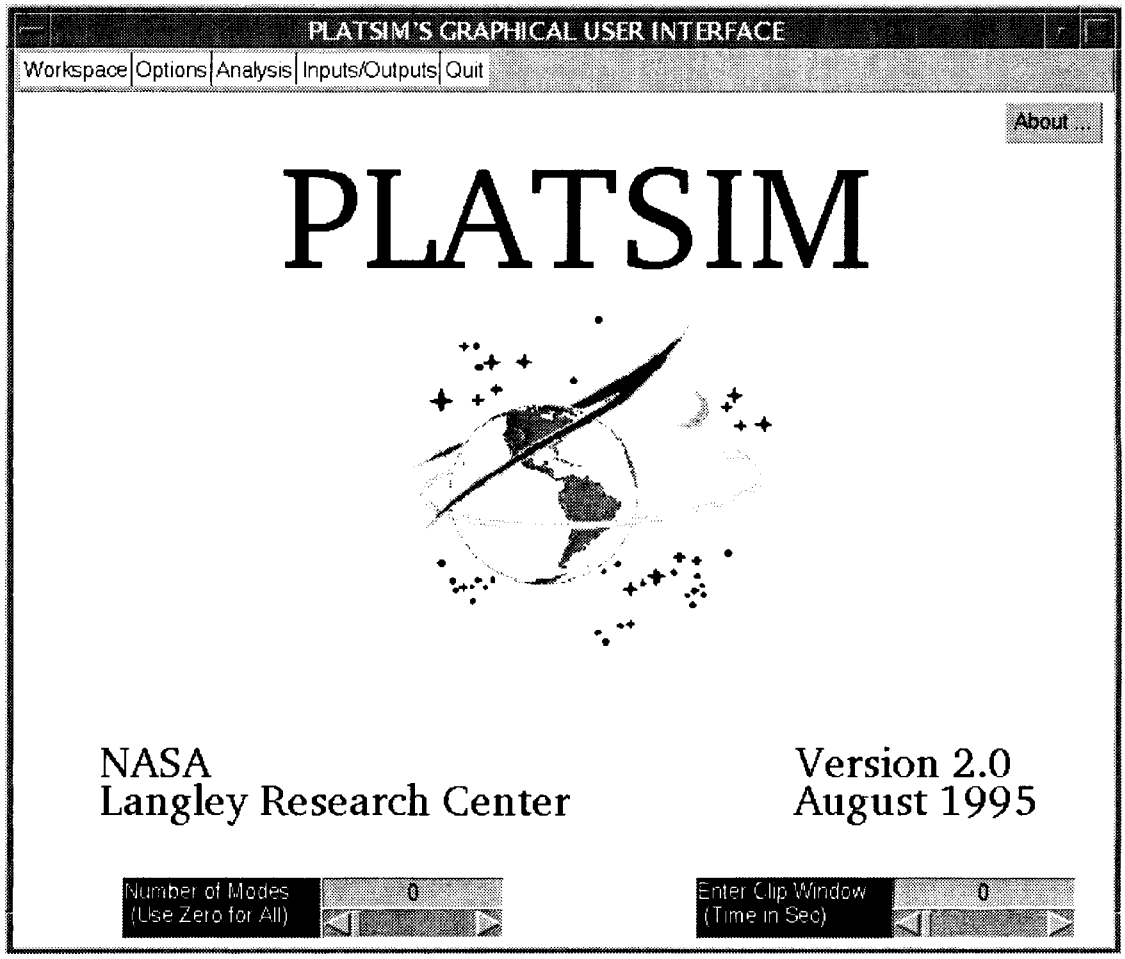


Figure 6. Graphical user interface for the PLATSIM package.

■ Workspace (see fig. 7, "Workspace button pull-down menu.")

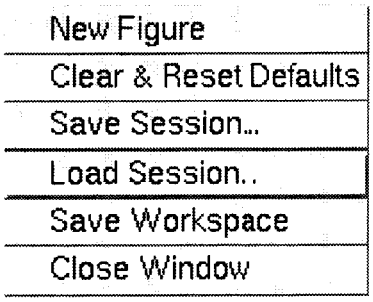


Figure 7. Workspace button pull-down menu.

- New Figure

  The "New Figure" menu button simply executes the MATLAB command figure. The creation of a new figure is useful if graphical preprocessing or postprocessing of PLATSIM data is desired.

- Clear & Reset Defaults

  This menu button clears all (work space and global) variables and resets them to their default values. (See file `defaults.m`.) This menu option must be invoked between successive PLATSIM runs.

- Save Session

  Selecting this menu button saves the current values of all PLATSIM variables to a binary "MAT-file". This utility, along with the "Load Session..." menu button, allows users to quickly recover previous PLATSIM run-time variable settings.

- Load Session

  Loads a previously saved session file and sets all GUI object properties accordingly.

- Save Workspace

  This menu button simply executes the MATLAB save command. All work space variables are saved on disk using the default file name `matlab.mat`.

- Close Window

  This menu button closes the main PLATSIM GUI window.

■ Options (see fig. 8, "Options button pull-down menu.")

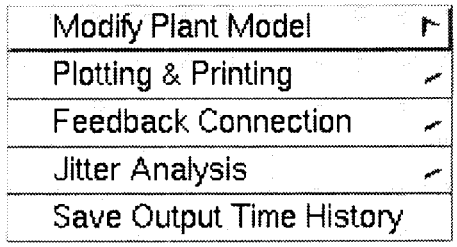


Figure 8. Options button pull-down menu.

- Modify Plant Model

  This menu button has two submenu selection options: "Frequencies" and "Damping Ratios". These submenu items allow the user to graphically modify or define both frequency uncertainties and modal damping ratios. Details for using these submenu options are described in "Supporting graphical interfaces" on page 25.

- Plotting & Printing

  The "Plotting & Printing" submenu button provides the user with three plot control options: "No Plotting", "Plot Results", and "Plot with Hardcopy". The "Plot with Hardcopy" option will plot the analysis results on the screen and save data-reduced versions of the plots in encapsulated PostScript files.

- Feedback Connection

  This menu button allows the user to select between open-loop, `clflag='no'`, and closed-loop, `clflag='yes'`, analysis modes. Specific controller models are prescribed in the "Set Analysis Parameters" graphical interface under the "Analysis" menu button.

- Jitter Analysis

  This menu button has three submenu selection options: "Perform Jitter Analysis", "Memory Conservative Mode", and "Jitter Windows". Selecting "Perform Jitter Analysis" sets `jtrflag='yes'` which invokes the baseline postprocessing jitter analysis algorithms. If the "Memory Conservative Mode" (`lowmemflag='yes'`) is selected, jitter values are calculated as time history data are generated. Therefore, creating and postprocessing potentially very large time history response vectors are not required. The combination of `lowmemflag='yes'` and `saveflag='yes'` is not valid because time history results are not available to write to disk. Selecting "Jitter Windows" opens an additional graphical interface (see fig. 9, "Jitter window interface," on page 23) for the selection of various jitter windows from

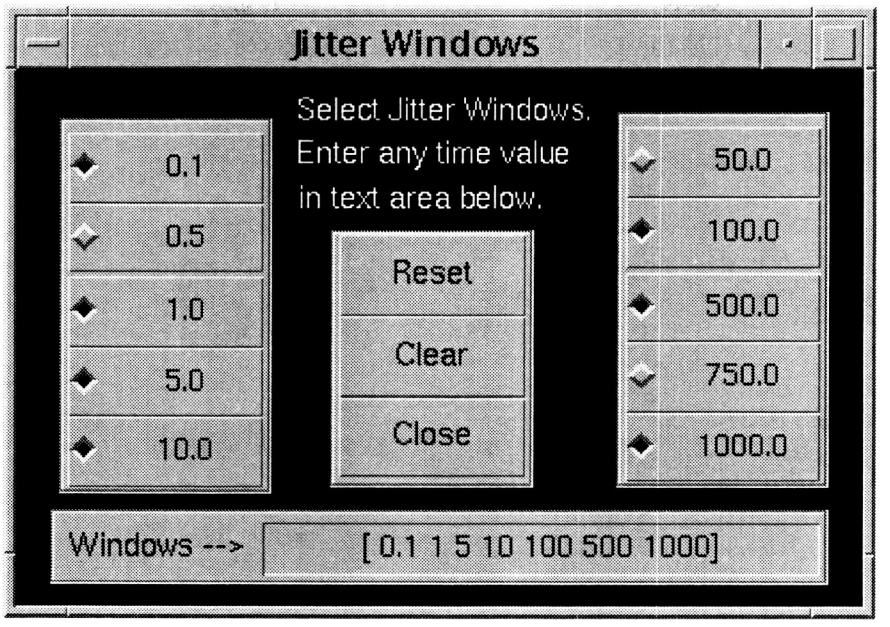


Figure 9. Jitter window interface.

either the predefined push buttons or by entering arbitrary time values in the text field at the bottom of the interface.

- Save Output Time History

  The "Save Output Time History" submenu button provides the user with the option to save the complete output time history data; that is, variables `y` and `instr` will be saved if this option is selected; that is, `saveflag='yes'`. Note that selecting this option has the potential of creating a very large file(s) and therefore should be used with caution. The output data will be saved in the file(s) labeled `y1.mat` and, if necessary, `y2.mat`, `y3.mat`, and so on—as many as necessary so that there is one for each disturbance event.

■ Analysis (see fig. 10, "Analysis button pull-down menu.")

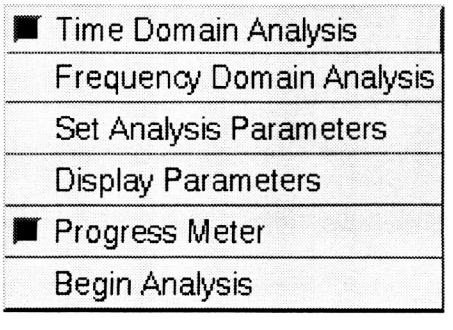


Figure 10. Analysis button pull-down menu.

- Time Domain Analysis

  Selecting this menu button specifies that time domain analysis will be performed; that is, `tdflag='yes'`. The time domain and frequency domain modes are mutually exclusive options; that is, PLATSIM does not allow for simultaneous time and frequency domain analyses.

- Frequency Domain Analysis

  This button specifies that frequency domain analysis will be performed; that is, `tdflag='no'`. When frequency domain analysis is selected, the clip window slider and text in the lower right corner of the main PLATSIM GUI will disappear.

- Set Analysis Parameters

  This menu button invokes supporting graphical interfaces that are used to define various integration and analysis parameters. The interface that is displayed when this button is pressed is dependent on the status of `tdflag` and `clflag`. See "Supporting graphical interfaces" on page 25.

- Display Parameters

  Selecting the "Display Parameters" button will echo the current values of PLATSIM parameters in the MATLAB command window.

- Progress Meter

  If "Progress Meter" is selected, a graphic meter is displayed showing the percent completion of the

simulation of the response to the current disturbance event or of the current frequency domain analysis.

- Begin Analysis

  Selecting "Begin Analysis" executes the MATLAB script file `plattime.m` or `platfreq.m` for time or frequency domain analysis, depending on the analysis requested. Once an analysis has been started, all GUI functions are disabled. User control of the GUI will not be returned until the analysis has been completed. As for MATLAB version 4.2 or 5.0, the user may terminate the analysis at any time by sending an interrupt signal (`Control C` on most UNIX workstations) from the MATLAB command window. If MATLAB is unresponsive to the interrupt signal, most UNIX systems will respond to the suspend signal (typically, `Control Z`). The user resorting to this expedient should remember to `kill` the suspended process.

■ Inputs/Outputs (fig. 11).

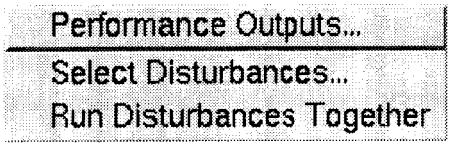


Figure 11. Inputs/outputs button pull-down menu.

- Performance Outputs

  Selecting the menu item labeled "Performance Outputs" provides access to the performance output selection figure window (see "Performance output selection interface" on page 31). See "Supporting graphical interfaces" on page 25.

- Select Disturbances

  This menu button allows the user to interactively select a disturbance model to be used in the PLATSIM analysis. See "Supporting graphical interfaces" on page 25.

- Run Disturbances Together

  This menu button controls whether several simulations are run, one for each individual disturbance event, or whether all disturbance events acting simultaneously are simulated. *If jitter analysis is performed without selecting "Run Disturbances Together," the jitter contributions from each of the individual disturbance events are added together to give a total jitter.* For the purpose of determining overall response at various output locations, the "Run Disturbances Together" option may provide more useful results. Selecting "Run Disturbances Together" sets the work space variable `multflag='yes'`. ***Warning: Running disturbances separately requires a separate simulation for each disturbance event. Disturbance scenarios which have many separate events may require excessive CPU time to complete.***

■ Quit

- Quit MATLAB

  This button simply quits the current MATLAB session without saving any work space variables.

■ Number of Modes

A slider control which is situated at the lower left-hand side of the top-level interface may be used to set the number of modes (PLATSIM variable `nmode`) which is to be included in the analysis. The slider allows the user to move, with a mouse, a sliding bar which sets the numeric input to the value that appears in the text field of the slider. The mode(s) to be used in the analysis may also be changed by clicking the mouse in the text field and typing the desired scalar or vector values. A scalar entry of `100` is equivalent to `[1:100]` in MATLAB notation; that is, the first 100 modes will be used in the simulation. Vector entries are used when multiple mode ranges are desired or when mode range starting points do not include the first mode. All vector entries must follow MATLAB's syntax. For example, to capture the rigid body response plus the flexible body response between modes 100 and 200, the user would enter `[1:6 100:200]` in the slider's text field. Negative scalar values are also permitted, and are used to indicate that a **single** mode will be used in the simulation, for example, if a text field entry of `-100` is used, only mode 100 will be used in the simulation. A slider value of `0` implies all available modes will be used in the simulation.

■ Enter Clip Window

This GUI item is used to define the clip window (`tclip`) used in time domain analysis. The functionality of this slider is similar to the mode slider. Definition of the `tclip` execution control parameter is provided in "Execution Control Parameters" on page 16.

Supporting graphical interfaces. Supporting graphical interfaces are accessed from either the top-level menu buttons or from other support interfaces. All top-level menu button labels ending with the ellipsis symbol (...) access a supporting interface. Instructions for each supporting interface are presented below in an itemized list, with each item label specifying the menu path from the top-level menu buttons. For example, the description of the frequency modification interface has this label: Options -> Modify Plant Model -> **Frequencies...**, which implies that it is accessed from the top-level "Options" button under the "Modify Plant Model" submenu button.

All PLATSIM's GUI routines operate in an asynchronous input mode, with the exception of the disturbance selection GUI, which is a synchronous process. Synchronous processing temporarily suspends other MATLAB processes and waits for the user to respond to the current synchronous process before continuing. Therefore, once the "Select Disturbances..." button is pressed, a disturbance selection should be made before attempting to access other PLATSIM features.

■ Options -> Modify Plant Model -> Frequencies...

Selecting the "Frequencies" submenu button of the "Modify Plant Model" option provides the user access to the Spacecraft Modal Uncertainty Graphical User Interface tool shown in figure 12, "Spacecraft modal uncertainty interface." **Note: Each time the "Frequencies" submenu button is selected, frequency values as defined in file(s)** `omega.dat` **or** `omega.mat` **are used as the initial values.** This interface tool has two options for adding modal uncertainties to the plant structural model. The two types of uncertainties considered are constant scaling and random scaling. The constant scaling uncertainties are defined as follows:

$$\omega_i^* = \omega_i^o + \omega_i^o \times S_j$$

Whereas the random scaling uncertainties are defined

$$\omega_i^* = \omega_i^o + \omega_i^o \times R_i \times S_j$$

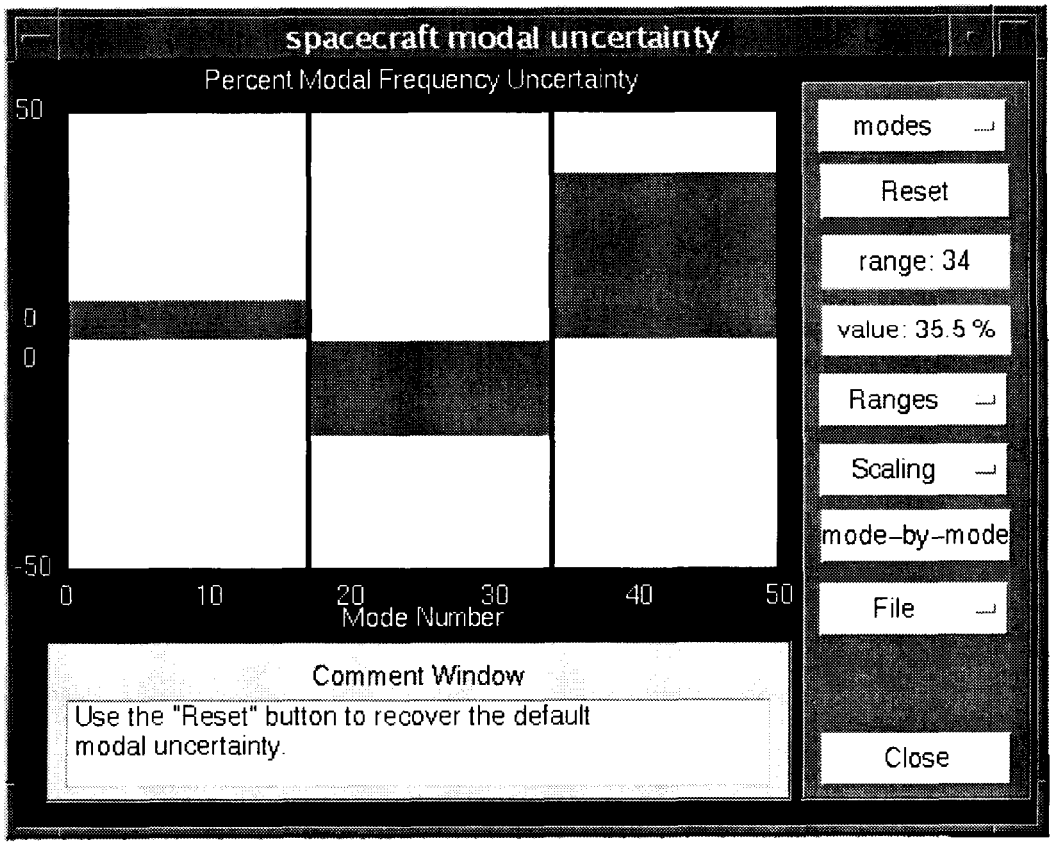


Figure 12. Spacecraft modal uncertainty interface.

The variable S_j in the above equations represents user-defined constant scale factor(s) for each prescribed range, and R_i represents a set of random numbers, one for each mode, uniformly distributed over the interval (0,1) (by the MATLAB random number generator `rand`).

- modes (frequency)

  The frequency spectrum axis may be changed from "Mode Number" (which is the default setting) to "Frequency (Hz)" by using the top push-button on the interface.

- Reset

  The "Reset" button resets the frequency uncertainty scale factors to zero for all modes and sets the number of ranges to three.

- range

  The frequency spectrum, as defined in file(s) `omega.dat` or `omega.mat`, may be subdivided into several intervals (ranges) to allow for different levels of uncertainties for different modes within the frequency spectrum. A "range", as defined by the interface, is denoted by the interval (a,b], where "a" and "b" are the lower and upper bounds on the interval, respectively. For example, if a = 10, and b = 20, all modes starting from 11 through 20 (including mode 20) will have the same S factor. The lower and upper bounds ("range" label on the GUI) for the interval may be changed by using a mouse to "click-and-drag" on the vertical lines that are used to represent interval ranges.

- value

  The scale factors ("value" label on the GUI) may also be changed by using a mouse to "click-and-drag" on the shaded regions between the vertical lines. Both "range" and "value" may be changed by using keyboard entries once the corresponding graphical element has been activated. A "range" or "value" element is activated by using the mouse to "click" on a vertical line or shaded region, respectively. Once a graphical element has been activated, its numeric value will be displayed in the "range" or "value" text area of the graphical interface.

- Ranges

  The user may change the number of ranges by selecting the interface button labeled "Ranges".

- Scaling

  A user may choose between the two allowable types of scaling by selecting "constant" or "random" from the "Scaling" button (constant is the default setting).

- mode by mode

  See item: Options -> Modify Plant Model -> Frequencies... -> **mode by mode**

- File

  The "File" button allows the user either to save the current frequency uncertainties to a ".mat" file or load a preexisting ".mat" file containing frequency uncertainties.

- Close

  This menu button closes the Spacecraft Modal Uncertainty Graphical User Interface.

■ Options -> Modify Plant Model -> Frequencies... -> **mode by mode**

Selecting the "mode-by-mode" button from the Spacecraft Modal Uncertainty Graphical User Interface provides access to a graphical interface which allows convenient mode-at-a-time frequency modifications. Figure 13 on page 27 shows a screen image of the mode-by-mode graphical interface. This interface gives the user a list of frequencies of all modes in a *read only* text window displayed on the right side of the interface.

- Frequency of Interest (Hz)

  The user can scan the list of frequencies to select a particular mode using the "Frequency of Interest" text field. In figure 13, "Mode-by-mode frequency modification interface," on page 27, a value of 1 Hz has been entered as the frequency of interest, and mode 25 was determined to be the closest mode, 1.0184 Hz.

- Desired Value (Hz)

  In figure 13 on page 27, the value of the currently selected mode, 1.0184 Hz, may be changed by either deleting the text and reentering a new value, or by scaling it using any arithmetic operator, for example, +, –, * (for multiplication), or / (for division).

- Change Selection

  The user can also scroll through the list of frequencies using the "Change Selection" buttons. The value in Hz of the currently selected mode will be displayed in the "Desired Value" text field. The currently selected mode is the mode that is between the two horizontal dashed lines.

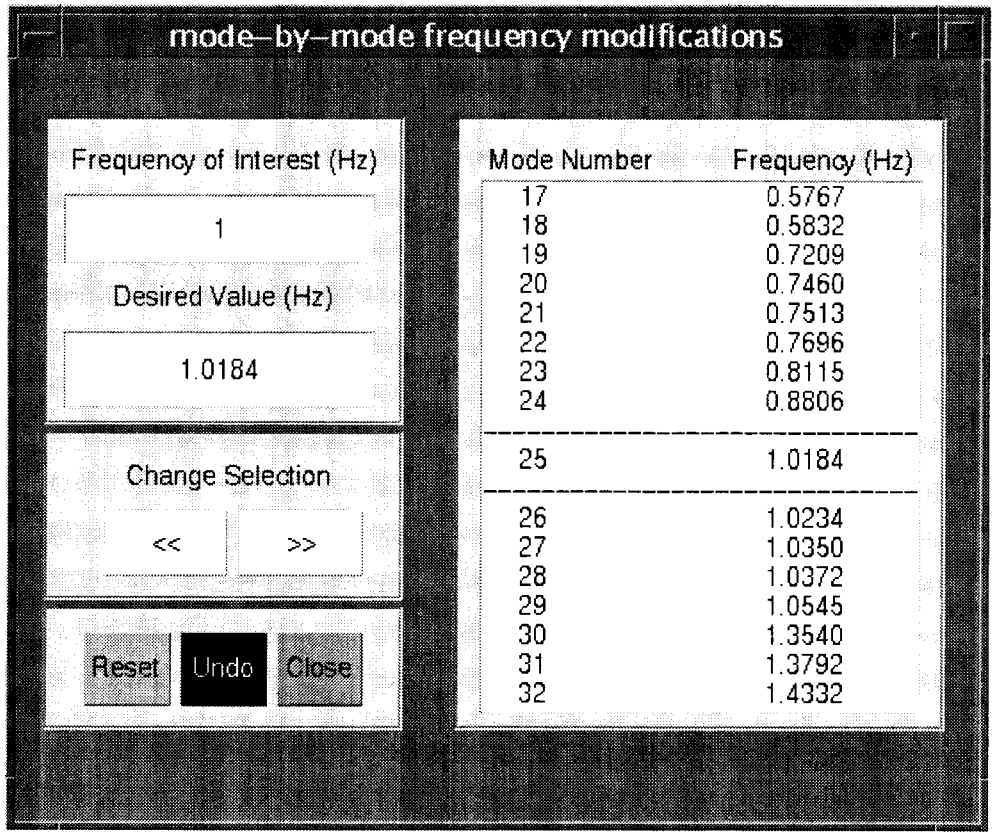


Figure 13. Mode-by-mode frequency modification interface.

- Reset

  The "Reset" button restores the modal frequencies to their nominal values, that is, those defined with the interface in figure 12 on page 25.

- Undo

  The "Undo" button clears only the last frequency modification.

- Close

  The "Close" button closes the interface and accepts all frequency changes.

■ Options -> Modify Plant Model -> **Damping Ratios...**

Selecting the "Damping Ratios" button from the "Modify Plant Model" menu provides the user access to a graphical tool for defining/modifying the spacecraft modal damping schedule. The operation of the damping schedule graphical interface is very similar to the frequency modification graphical interface. The graphical damping tool allows users to easily modify the structural damping ratios. Structural damping ratios and ranges are specified by using a mouse to "click and drag" on graphical elements that represent modal damping ratios and ranges. Keyboard entries are also permitted once a modal damping range or value element has been activated by a mouse click (see above frequency modification description for definition of ranges, values, and active elements). Damping schedules may also be saved or loaded by using the "File" button. This tool uses, as its default damping schedule, the schedule defined in the user-changeable file `mkdamp.m`. If `mkdamp.m` has not been defined, then a default damping schedule of 0.25 percent is assumed for all modes.

■ Analysis -> **Set Analysis Parameters...** (time domain analysis)

This interface provides access to various PLATSIM time domain analysis parameters. See figure 14 on page 28 for a screen image of this interface. As mentioned in "Top-level graphical interface" on page 21, the interface that is displayed when the "Set Analysis Parameters" button is pressed is dependent on the status of `tdflag` and `clflag`. The following five time domain analysis features are accessible with this interface:

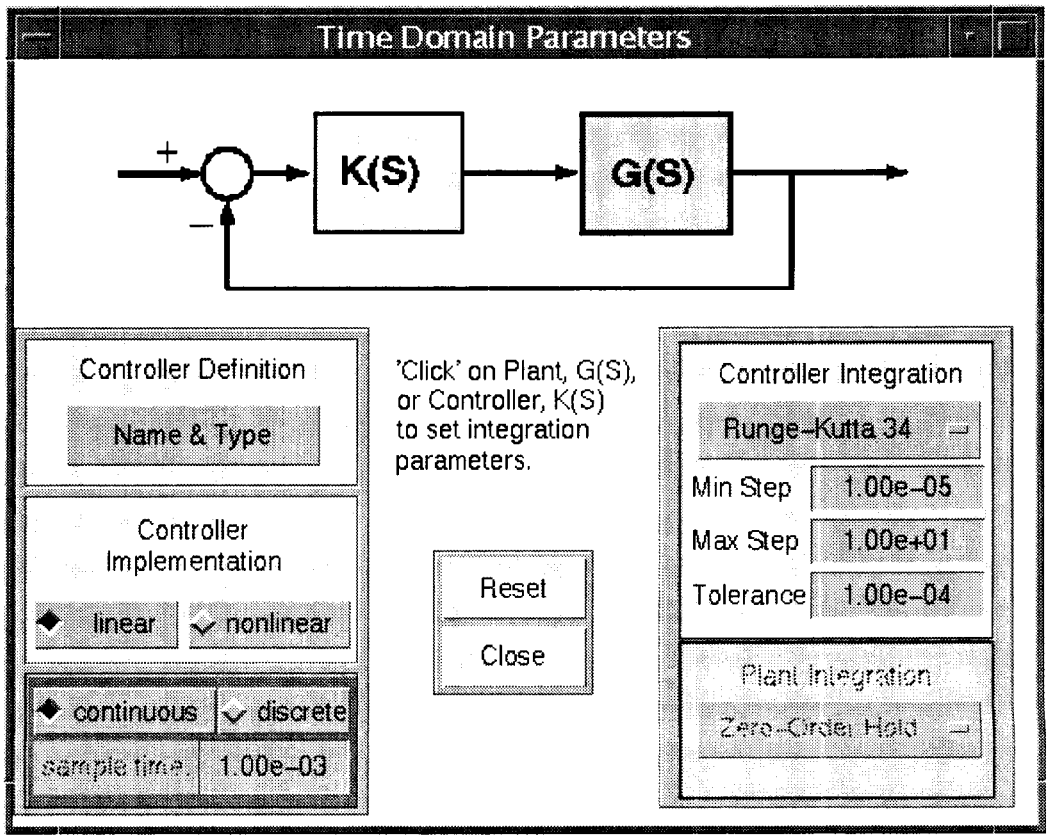


Figure 14. Set analysis parameters interface (time domain).

- Controller Definition

  Pressing the "Name & Type" button opens the GUI shown in figure 15 on page 29. In this window, the user must enter the controller file name and select whether it is a linear or nonlinear controller. The file name is stored in string variable `nlcon` and the type is in string variable `linflag`.

- Controller Implementation

  The linear or nonlinear implementation methods (`impflag`) can be selected by using this interface. See "Time Domain" on page 13, and "Execution Control Parameters" on page 16 for a complete description of controller implementation methods.

- Continuous- or Discrete-Time Controller

  PLATSIM variable `ctflag` is set using the buttons labeled: "continuous", "discrete", and variable `tsc` is defined in the "sample time" text field. Select "discrete" (`ctflag='no'`) for mixed-time controllers, that is, for controllers with both continuous and discrete states.

- Controller Integration

  The desired nonlinear controller integration routine may be selected from the pop-up menu on the "Controller Integration" panel. The additional integration parameters that can be set with this interface are minimum step size, maximum step size, and relative error tolerance.

- Plant Integration

  The choice of three plant integration routines are available in this release of PLATSIM. The pop-up menu on the "Plant Integration" panel provides the interface to these choices: zero-order hold, first-order hold, and Runge-Kutta 34.

■ Analysis -> **Set Analysis Parameters...** (frequency domain analysis)

This interface provides access to various PLATSIM frequency domain analysis parameters. See figure 16, "Set analysis parameters (closed loop, frequency domain)," on page 29 for a screen image of this interface. The following six frequency domain analysis features are accessible with this interface:

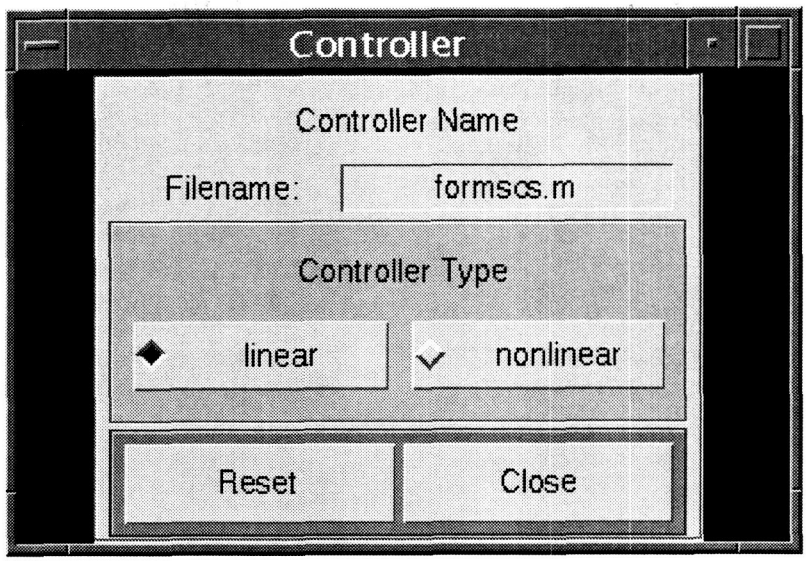


Figure 15. Controller definition interface.

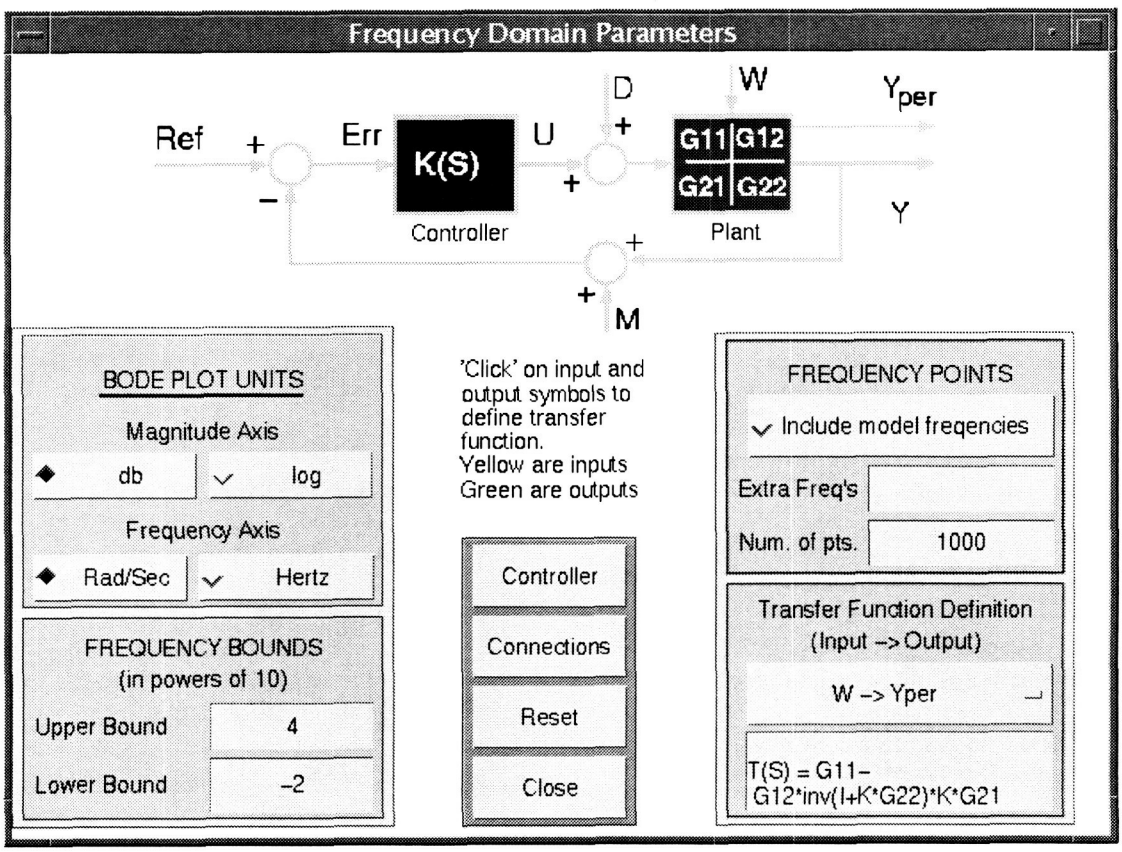


Figure 16. Set analysis parameters (closed loop, frequency domain).

- Bode plot units

  The magnitude and frequency units on the Bode plots are defined by using this interface. The variables corresponding to the magnitude and frequency units are `bmagax` and `bfrax`, respectively. See "Execution Control Parameters" on page 16.

- Frequency bounds

  Upper and lower frequency bounds are defined using this interface. The variables corresponding to the upper and lower bounds are `iu` and `il`, respectively. Both entries are in powers of ten, that is, 10^{il} and 10^{iu}. The units for the frequency bounds are defined by the parameter `bfrax`.

- Frequency points

  This panel has three options: include model frequencies (`frqflag`), extra frequency points (`usrfrq`), and the number of points (`npts`) at which frequency domain analysis will be done. Selecting the "Include model frequencies" button sets variable `frqflag='yes'`, which augments the nominal frequency vector with model frequencies. The nominal frequency vector is defined by using the MATLAB command: `logspace(il,iu,npts)`. See "Frequency Domain" on page 14, for a list of assumptions and restrictions for frequency domain analysis.

- Transfer function definition

  PLATSIM has the capability of performing Bode analysis on 16 closed-loop and 9 open-loop transfer functions. Transfer functions may be selected from the pop-up menu on the "Transfer Function Definition" panel or selected graphically by using the mouse to point and click on the block diagram input/output symbols. See "Frequency Domain" on page 14 for a description of transfer function inputs and outputs.

- Controller

  Pressing the "Controller" button opens the same GUI shown in figure 15 on page 29. In this window, the user must enter the controller file name and whether it is a linear or nonlinear controller. The file name is stored in string variable, `nlcon` and the type is in string variable, `linflag`.

- Connections

  Selecting the button labeled "Connections" provides access to the graphical interface shown in figure 17, "Input/output connections interface," on page 31. Once a transfer function has been defined, specific input/output connections may be specified with this interface. Variables `desinti` and `desinto` (see "Execution Control Parameters" on page 16) are defined with this interface. The interface will display all possible inputs and outputs for a given transfer function. Inputs and outputs are labeled as `input_xyz` for inputs and `output_xyz` for outputs, where `xyz` is the unique identification number as specified in data files `instdata.m` and `distdata.m`. See "Instrument Data File" on page 4 for descriptions of identification numbers. To use the interface, select the input/output pairs and press the "Connect" button. Connections are made from each activated input to all activated outputs. In figure 17, "Input/output connections interface" on page 31, two separate sets of connections have been made; that is, `input_2` is connected to `outputs 1, 3` and `6`, and `input_6` is connected to `outputs_3` and `4`. (Also see fig. 18 on page 31, "Performance output selection interface.") Input and output buttons can be activated individually by pressing each button separately, or in groups by using the middle mouse button and dragging a rectangle around the desired inputs and/or outputs. The "Delete All" button breaks all connections and sets `desinti=[]` and `desinto=[]`. The "Delete" button, while activated, permits connections to be broken from individual inputs or outputs. Transfer functions using external disturbances as inputs require a disturbance scenario to be specified so that specific inputs may be correctly displayed. Therefore, if the "Connections" button is pressed before a disturbance is chosen, the user will be required to make a disturbance scenario selection using the interface shown in figure 19 on page 31 before individual connections can be specified.

■ Inputs/Outputs -> **Performance Output...**

Selecting the submenu item labeled "Performance Outputs" provides access to the performance output selection figure window shown in figure 18 on page 31. The performance output window provides a menu-driven method of selecting a full or partial set of output locations from those defined in the user-defined function `instdata.m`. This utility allows for a large database of output locations to be maintained in function `instdata.m`, without the computational expense of solving for all these outputs in every analysis. The labeling and grouping of the menu items for the performance output window are determined by the third field in the work space variable `instr`, which itself is defined in function

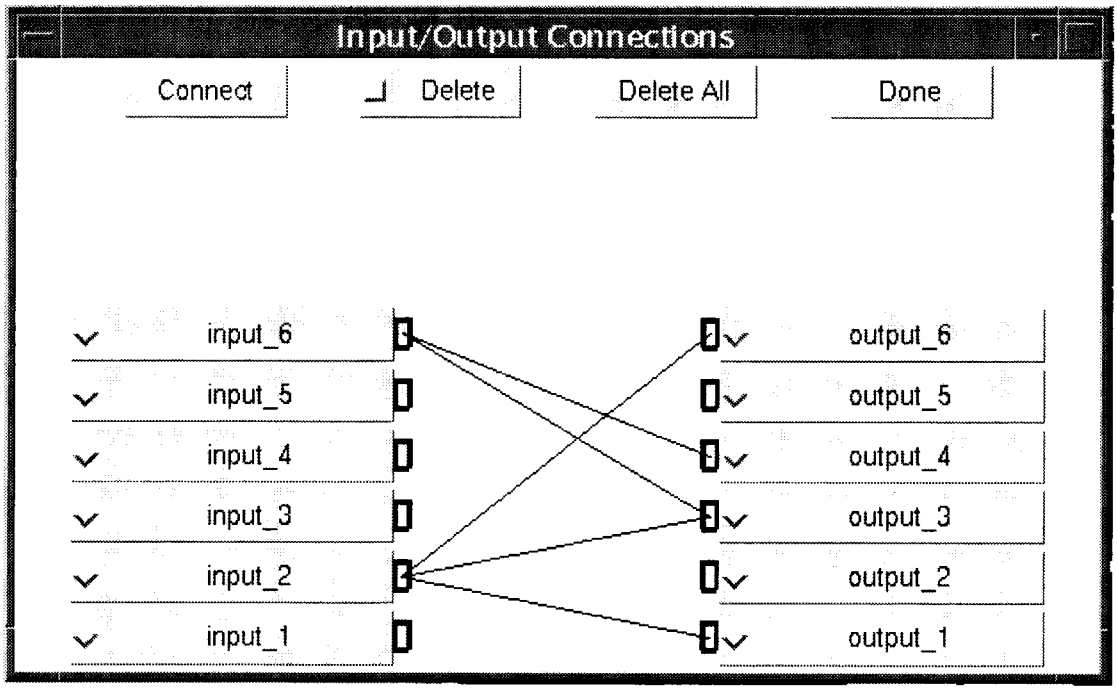


Figure 17. Input/output connections interface.

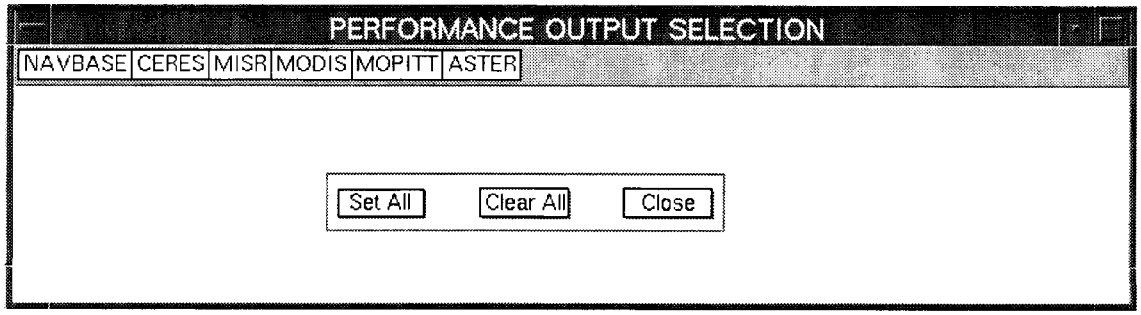


Figure 18. Performance output selection interface.

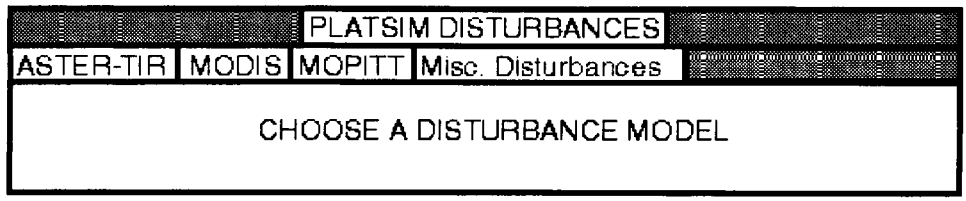


Figure 19. Disturbance module interface.

instdata.m. All entries that have exactly the same character string in the third field of instr will be under the same pull-down menu, and this common character string will be used to title the menu group. See "Instrument Data File" on page 4 for more information on the instrument data file. The performance output selection figure window has three pushbutton functions. These buttons are used to select all outputs, deselect all outputs, or close the selection window, and are labeled "Set All", "Clear All", and "Close", respectively. A user may select performance outputs individually, or as groups of outputs using the "Select All" option for each group, or all groups using the master "Set All" button. Any currently selected output may be deselected by simply reselecting the item, or using the "Clear All" button. An additional level of print control is available for each group of performance outputs using the "Print" button at the bottom of each group's pull-down menu. Deselecting the "Print" button disables printing and plotting for all performance outputs under that menu heading.

■ Inputs/Outputs -> **Select Disturbances...**

When the pull-down menu button labeled "Select Disturbances" is selected, an additional figure window with a top-level menu bar will appear; see figure 19, "Disturbance module interface," on page 31. The disturbance window top-level menu bar is used to display labels pertaining to disturbance scenario groups that may consist of one or more disturbance scenarios. A single instrument/system may have multiple disturbance scenarios. For example, a scanning telescope may have a disturbance scenario that describes the forces/torques related to the motion of a scanning mirror and a separate disturbance scenario for calibration operations. The entries in the cnames and dnames matrices (see "Disturbance Data" on page 7 for more details) provide the names for the disturbance scenario groups and the disturbance scenarios, respectively. A disturbance scenario may be selected from the pull-down menu with a single click from a mouse. Upon selection, the appropriate disturbance models, as defined in the user provided file distdata.m, will be executed and instrument/system disturbance data will be loaded into the MATLAB work space. This process may take several seconds to complete, depending on the number of data points and the number of individual force and/or torque profiles that make up the disturbance scenario. Note that if the user attempts to run a simulation without first selecting a disturbance model, a warning message will be displayed with instructions to select a disturbance followed by the display of the disturbance module figure window.

Batch Mode

In batch mode, each PLATSIM execution control parameter is set either by the user with MATLAB assignment statements before the platsim command is entered or it is used with its default value. Execution control parameters may be set in an interactive MATLAB window, but it is primarily intended to provide a way to run PLATSIM in the background by using commands from a prewritten script.

Batch Mode Operation

Batch mode operation will be demonstrated with two batch mode examples.

Example 1. MATLAB is started in a directory containing two files which are listed here, startup.m and runplat.m:

```
::::::::::::::
startup.m
::::::::::::::
format compact
path('/scb3/usr5/eos/platdir',path)
path('/scb3/usr5/eos/platdir/eos_eg',path)
::::::::::::::
runplat.m
::::::::::::::
diary            % generate a diary of the run
runmode='b';     % run in batch mode
tdflag = 'n';    % perform frequency domain analysis
nmode = 40;      % use first 40 modes
clflag = 'n';    % run Bode plot open loop
casenum = 7;     % use 'MODIS static imbalance' disturbance
                 % from the EOS-AM-1 example
```

```
prtflag = 'n';   % do not write .eps files
desinto = 14;    % only output is 'MODIS Pitch'
ndist = -2;      % use only the second event from the
                 % 'MODIS static imbalance' disturbance
phold = 120;     % give the user at least 2 minutes to look at plot
platsim          % This should now run a single input/single output
                 % Bode plot using 1000 points logarithmically spaced
                 % between .01 and 10,000.
```

MATLAB is started and the command

```
>> runplat
```

is given. MATLAB performs the necessary calculations and displays the Bode plot. To exit from MATLAB, the command

```
>> quit
```

is given. Now two new files are in the directory, `MODIS_static_imbalance_freq.mat`, which contains the results of the Bode calculation in MATLAB readable form, and `diary`, which shows what occurred:

```
PLATSIM initialization in progress...

Selected disturbance case: MODIS
static imbalance

Initiating Bode Plot for bodemthd
=[3 3]

Task completed in 13.05 seconds

Plot and print Bode plots Task
completed in 8.16 seconds

PROGRAM COMPLETED Total cpu time =
22.72 seconds >> quit

2204653 flops.
```

Example 2 uses a batch operation utility such as the UNIX `at` command. First a command script is prepared and written to a file, for example, `doit` (line numbers are not present in the script; they have been added here for reference purposes):

1. `setenv DISPLAY`
2. `/usr/local/matlab-4.2/bin/matlab << EOF > mat-out`
3. `diary`
4. `runmode='b';`
5. `casenum=7;`
6. `pmflag='n';`
7. `platsim`
8. `quit`
9. `EOF`

A UNIX command such as SUN Solaris command `at -m -f doit 20:00` will wait until 8 p.m. and then start executing the commands in `doit`. Line 1 of `doit` may get around a problem MATLAB has on some systems while trying to run plotting commands without running them from a logged-in terminal. Line 2 starts MATLAB running and uses the following lines as input to MATLAB until it finds the line (9) matching the `EOF` in line 2. MATLAB then executes lines 3 through 8, causing it to run PLATSIM in batch mode by using disturbance number 7 (MODIS static imbalance) and by using defaults for all other run parameters except `pmflag`. When the job is done, the user can harvest results from the files created by PLATSIM. (See chapter 5, "PLATSIM Output.")

Chapter 5

PLATSIM Output

PLATSIM returns its results in two general manners, interactive and permanent. Interactive output includes MATLAB work space variables which are set to the results of PLATSIM calculations, plots, and tables displayed in MATLAB figure windows. Permanent output consists of a variety of files. The most useful of the PLATSIM work space variables can be captured in MATLAB MAT-files. Plots can be written as encapsulated PostScript files. Tables of jitter values are written as both ASCII files and in PostScript form.

General file-naming conventions will be given first, and the modifications necessary to fit PC file-naming limitations will be given at the end of each section.

Time Domain Analysis

Full-time histories. The results of each simulation are contained in a variable `y` which contains one column for each performance measurement output and one row for each row in the user-supplied disturbance time histories used to drive the simulation. If the user needs these full-time histories, the user can elect to have them written to MAT-files (`saveflag` on page 19 and "Save Output Time History" on page 23).

1. File `y1.mat` will contain the time history by using the first disturbance event or the simulation by using all events simultaneously.
2. If there is more than one disturbance event and they are being run separately, the additional time histories are in files `y2.mat`, `y3.mat`, ... , up to as many as are needed.
3. Each file contains the time history in variable `y` as previously stated, a scalar variable `period` that contains the time increment between discrete points of the simulation, and a character array `instr` whose rows contain the names of the performance outputs that are being simulated.
4. **Note: Depending on the numbers of performance outputs, the time steps of simulation, and the disturbance events, these files can be very large and can require significant disk space.**

Time history plots and reduced time history data. The user can elect to have the time histories plotted on-screen and further can elect to write encapsulated PostScript files of the on-screen plots. If the latter is chosen, a reduced form of the data is plotted. Although some simulations involved a hundred thousand or even a million time steps, the reduced data typically involve vector lengths of less than ten thousand. The reduction is performed in such a manner (ref. 3) that the visual effect of plotting the reduced data is virtually identical to that of plotting the full time history. Thus, the results are faster on-screen plotting, faster writing of encapsulated Adobe PostScript files, smaller PostScript files, and faster printing of these files by a PostScript printer.

The following rules define how time history plot variable names are represented in the MATLAB work space:

1. If each disturbance event is simulated separately, there is one time history plot variable for each combination of performance measurement output and disturbance event.
2. If the disturbance events are run together, there is one time history plot variable per performance measurement output.
3. The reduced data from a typical time history is contained in a MATLAB variable with a name which looks like `p113_1`.
   - The `113` is a performance output identification number taken from the third row of the user-supplied `pout` matrix. (See "Instrument data parameter `pout`" on page 6.)
   - The number `1` indicates that this is the response to the first event in the chosen disturbance scenario if events are run separately, or that this is a simulation using all the events simultaneously.
   - If there is more than one event in the disturbance scenario, and if the events are being run separately, the reduced response to the second event will use the number `2` after the underscore, and so forth.
   - The variable `p113_1` (or like variables) is a 2-column matrix. Column 1 contains time data, and column 2 is the performance output data.
   - The plots are made by plotting the second column of this variable as a function of the first column.

The following rules define the file-naming convention used for the previously mentioned time history plot variables. All these variables are saved in a MATLAB MAT-file with a name such as `MODIS_static_imbalance_1_time.mat`.

1. The `MODIS_static_imbalance` part of the name comes from replacing any blanks with underscore characters in the name for the disturbance used in this simulation.
2. The number `1` is used if these data come from the first disturbance event in the disturbance scenario when events are being run separately, and it is also used if the events are run simultaneously.

3. If the events are being run separately, the second event would have the number 2.

4. The time in the name distinguishes this file from a similarly named file from the frequency domain analysis part of PLATSIM.

If the user chooses the option "Plot with Hardcopy", PLATSIM writes encapsulated PostScript files of the plots, which the user can send to the printer. These files have names such as MODIS_Yaw_1_time.eps.

1. The MODIS_Yaw part of the name comes from replacing any blanks with underscore characters in the name for the performance output used in this simulation.

2. The number 1 and time follow the same convention as described in the previous paragraph.

Jitter results and plots. If the calculation of jitter is elected, the results of the jitter calculation are available to the user in several forms. The following rules define how jitter work space variable names and file names are generated.

1. If disturbance events are run separately, the results of the jitter calculation for the first event are in MATLAB variable JITTER1; those for the second event (if any) are in JITTER2, and so forth.

2. Each of these variables is a matrix with one column for each jitter window and one row for each performance output (the order of the rows is determined by the numerical order of the identification numbers of the performance outputs).

3. The "total jitter" in the form of the sum of the individual jitter calculations is given in the variable JITTER.

4. If the disturbance events are run simultaneously, the results are in JITTER1 and JITTER, which, in this case, are identical.

5. The variable JITTER1, JITTER2 (if defined), and so forth are preserved in the same files containing the reduced time histories (see the previous section). For example, JITTER2 would be in file MODIS_static_imbalance_2_time.mat following the example in the previous section.

6. The variable JITTER would be written in a file named MODIS_static_imbalance_time.mat.

The table of jitter values is displayed on the screen.

1. If the events are being run separately, a table is presented for each event and another is presented for the jitter totals.

   - The worst value in each column is displayed in a contrasting color for emphasis.
   - These tables are also written in both PostScript and ASCII files.
   - The PostScript format tries to emulate what is shown on the screen; however, if too many performance measurement outputs exist, the PostScript file prints two or more pages.
   - The PostScript files for the individual disturbance event jitter results have names such as MODIS_static_imbalance_1_jitr.ps and MODIS_static_imbalance_2_jitr.ps, while the table with overall totals is in MODIS_static_imbalance_jitr.ps.
   - For the ASCII files, the names are the same except that the ps extension is replaced by out.

2. If there is only one event in the disturbance scenario, or if the disturbance events are run simultaneously,

   - The one jitter result is written to the files with names such as MODIS_static_imbalance_jitr.ps and MODIS_static_imbalance_jitr.out (no event number).

Example. An example of time domain analysis and jitter analysis, which is based on the EOS-AM-1 spacecraft, is presented in appendix D. In the example, PLATSIM runs use the default values of all execution time parameters, along with the disturbance scenario MODIS static imbalance.

File-naming conventions for PC's. PC's running under DOS (Disk Operating System) have limitations on which character strings may be file names. Letters are mapped to upper case. The file name may have, at most, eight characters, optionally followed by a period and an extension of one to three characters. Consequently, PLATSIM output files are given alternate names which conform to PC DOS restrictions, if the program is run on a PC:

1. The file which was called, for example, MODIS_static_imbalance_1_time.mat is now called JITTER1.mat. The file which was called, for example, MODIS_static_imbalance_time.mat is now called JITTER.MAT.

2. The file which was called, for example, MODIS_static_imbalance_1_jitr.out is now called JITR_1.OUT.

3. The file which was called, for example, `MODIS_static_imbalance_1_jitr.ps` is now called `JITR.PS`.
4. The file which was called, for example, `CERES2_Yaw_2_time.eps` is now called `P9_2_T.EPS`.
   - The number 9 comes from the identification number used for the instrument named `CERES2 Yaw`.
   - The number 2 indicates the disturbance event number.
   - The letter `T` indicates that the file contains a time history.
   - Only single or double digit identification numbers are allowed for the performance outputs.
   - Up to 99 events in a disturbance scenario are allowed.

Frequency Domain Analysis

Frequency response matrix. In frequency domain analysis, PLATSIM calculates either the open-loop or closed-loop frequency response matrix from a set of user-selected inputs to a set of user-selected outputs. As mentioned in the Instrument Data File on page 4, the inputs may be in the form of attitude or attitude rate reference commands, disturbances at the plant input, external disturbances, or sensor noise, if closed-loop response is requested, and in the form of attitude or attitude rate reference commands, disturbances at the plant input, or external disturbances, if open-loop response is requested. Similarly, the outputs may be in the form of tracking error, control effort, performance output, or measurement output, if closed-loop response is requested, and in the form of control effort, performance output, or measurement output, if open-loop response is requested. Moreover, all or a user-specified subset of the inputs or outputs may be chosen by the user. The vector of frequency points is generated from the execution control parameters defined by the user. (See parameters: `il`, `iu`, `npts`, `clflag`, `usrfrq`, and `frqflag` in table 1.)

The frequency response matrix is stored in a MATLAB work space array `g`. The size of array `g` is k by $\tau \times p$, where k is the number of elements in the frequency vector, τ is the number of selected inputs, and p is the number of selected outputs. Subsequently, the k by $\tau \times p$ gain matrix in decibels is stored in MATLAB variable `m`, and the k by $\tau \times p$ wrapped phase matrix in degrees is stored in MATLAB variable `p`. The frequency vector is stored in a MATLAB variable `frq`.

The variables `frq`, `g`, `m`, and `p` are work space variables that are available to the user when PLATSIM completes the frequency domain calculations. These variables are also written in MATLAB binary form to a MAT-file following naming convention:

1. If the inputs are reference commands, the file name would be `reference_command_freq.mat`.
2. If the inputs are disturbances at the plant input, the file name would be `input_disturbance_freq.mat`.
3. If the inputs are external disturbances, the file name would be `MODIS_static_imbalance_freq.mat`. The `MODIS_static_imbalance` part of the name comes from replacing any blanks with underscore characters in the name for the disturbance used in this analysis.
4. If the inputs are measurement noise, the file name would be `measurement_noise_freq.mat`.

Bode plots. If plotting is requested, the Bode plots for the selected outputs in response to the selected inputs are plotted on-screen. If "Plot with Hardcopy" is selected, in addition to on-screen plots, PLATSIM writes encapsulated Adobe PostScript files of the plots. These files have the following naming convention:

1. If the outputs are tracking errors, the file name would be `tracking_error_1_4_freq.eps`. The number 1 corresponds to the identification number associated with the measurement output used in the analysis. The "4" corresponds to the identification number associated with the selected input. For example, if the input is an external disturbance, the number 4 indicates that the input used to generate the Bode plot corresponds to the disturbance event with identification number 4 for the disturbance scenario chosen.
2. If the outputs are control effort, the file name would be `control_input_1_4_freq.eps`. The number 1 corresponds to the identification number associated with the actuator used in the analysis. The number 4 corresponds to the identification number associated with the selected input. For example, if the input is a reference command, the number 4 indicates that the input used to generate the Bode plot corresponds to a reference command for measurement output with identification number 4.
3. If the outputs are performance outputs, the file name would be `MODIS_Yaw_4_freq.eps`. The `MODIS_Yaw` part of the name comes from replacing any blanks with underscore characters in the name for the performance output, supplied by the user, that is used in this analysis. The number 4 corresponds to the identification number associated with the selected

input. For example, if the input is an external disturbance, the number 4 indicates that the input used to generate the Bode plot corresponds to the disturbance event with identification number 4 for the disturbance scenario chosen.

4. If the outputs are measurement outputs, the file name would be `measurement_output_1_4_freq.eps`. The number 1 corresponds to the identification number associated with the measurement output used in the analysis. The number 4 corresponds to the identification number associated with the selected input. For example, if the input is measurement noise, the number 4 indicates that the input used to generate the Bode plot corresponds to a measurement noise in the measurement output with identification number 4.

An example of frequency domain analysis, which is based on the EOS-AM-1 spacecraft, is presented in appendix D. In the example, PLATSIM runs use the default values of all execution parameters except that `tdflag` = 'n' selects frequency domain analysis, along with the disturbance scenario `MODIS static imbalance`.

File-naming conventions for PC's. As in the previous section on time domain analysis, the file-naming conventions just given will not work for PC's. The conventions used on PC's are exemplified here:

1. The frequency response matrix, the gain and phase matrices, and the frequency vector are stored in file `FREQ.mat`.
2. The Bode plots, irrespective of the type of input or output, will be `o1i4_f.eps`. The letters `o` and `i` are prefixes that denote output and input, respectively. The number 1 denotes the identification number associated with the output (any form of output). The number 4 denotes the identification number associated with the input (any form of input).
3. Using this naming convention, the identification numbers on inputs and outputs can each range from 1 to 99 without violating the PC DOS file-naming limitation of 8 characters before the period.

Chapter 6

Diagnostic Messages

The error and diagnostic messages generated by PLATSIM are given below. The format used below gives a brief description of a scenario that may cause the error, the actual error message as seen in the MATLAB command window, followed by a recommended solution.

- In the GUI execution mode, if the "begin analysis" button is pushed before a disturbance is selected from the disturbance menu, PLATSIM displays the "PLATSIM DISTURBANCES" menu and the following message appears in the MATLAB window.

A disturbance case was not specified, please select one now.
Solution: Choose a disturbance from the pop-up menu.

- In the batch execution mode, if PLATSIM is executed without the execution control parameter `casenum` having been defined, the following message appears in the MATLAB window, and the execution of PLATSIM terminates.

The batch mode disturbance variable '`casenum`' **is not defined. Please select a valid case number from file:** '`distdata.m`' **and resubmit** '`platsim`'.
Solution: Assign a valid value to the execution control parameter `casenum` and restart the execution of PLATSIM.

- In the batch execution mode, if PLATSIM is executed with the execution control parameter `casenum` having been set to an invalid value, a message of the following form appears in the MATLAB window, and the execution of PLATSIM terminates.

`casenum=200`, **is not a valid disturbance case number. Select an integer between 1 and 15.**

Check batch mode input data.
Solution: Assign a valid value to the execution control parameter `casenum` and restart the execution of PLATSIM.

- If the variables `mapping`, `instdat`, `cnames`, and `dnames`, as defined in the user-provided file `distdata.m`, do not satisfy specific relationships, the following message will appear in the MATLAB window.

Disturbance data is not correct!

Solution: See "Disturbance Data" on page 7 for a complete description of `distdata.m` variables.

- In the GUI execution mode, if the scalar parameter `tclip` contains a value greater than the simulation time, then a message of the following form appears in the MATLAB window.

`tclip (=1000)` **is larger than** `tfinal (=999.974)`
input a new value for `tclip -->`
Solution: Type in a valid value for `tclip` after the arrow, followed by a carriage return.

- If the scalar parameter `tclip` defined is greater than the simulation time and the execution mode is batch, the following message will appear:

`tclip` **is larger than** `tfinal`

`tclip` **is set to** `0.5*tfinal`.

Solution: Adjust `tclip` or increase the simulation time.

- If any mode number chosen for analysis is greater than the number of modes available in the file `omega.dat` or `omega.mat`, the following message will appear in the MATLAB window and program execution will stop.

An error has been detected in file `formplnt`. **Maximum mode number chosen is greater than number of modes available. Program termination in** `formplnt`.

Solution: Check the mode numbers and restart the program.

- If the number of rows of array `phi` (in file `phi.mat` or `phi.dat`) is not an integer multiple of the number of modal frequencies defined in file `omega.mat` or `omega.dat`, the following error message will appear in the MATLAB window and program execution will stop.

The number of rows of "phi" (in file `phi.mat` **or** `phi.dat`**) must be an integer multiple of the number of elements of `omega' (in file** `omega.mat` **or** `omega.dat`**). Program termination in** `formplnt`.

Solution: Make the `phi` and `omega` files consistent and restart the program.

- If a grid point defined in file `instdata.m` for the spacecraft control system input is not defined in array `phi` (in file `phi.mat` or `phi.dat`), the following error message will appear in the MATLAB window and program execution will stop.

An error has been detected in file `formplnt.m`. **A FEM SCS input grid number as referenced in file** `instdata.m` **is not available in file** `phi.dat` **or** `phi.mat`. **Program termination in** `formplnt`.

Solution: Check the first row of parameter `act` in file `instdata.m`, check file `phi.dat` or `phi.mat`, and restart the program.

- If a grid point used in file instdata.m for the measurement outputs is not defined in array phi (in file phi.mat or phi.dat), the following error message will appear in the MATLAB window and program execution will stop.

An error has been detected in file formplnt.m. **A FEM measurement output grid number as referenced in file** instdata.m **is not available in file** phi.dat **or** phi.mat. **Program termination in** formplnt.

Solution: Check the first parameter mout in file instdata.m and check file phi.dat or phi.mat, and then restart the program.

- If a grid point used in file instdata.m for the performance outputs is not defined in array phi (in file phi.mat or phi.dat), the following error message will appear in the MATLAB window and program execution will stop.

An error has been detected in file formplnt.m **A FEM performance output grid number as referenced in file** instdata.m **is not available in file** phi.dat **or** phi.mat. **Program termination in** formplnt.

Solution: Check the first row of parameter pout in file instdata.m, check file phi.dat or phi.mat, and restart the program.

- If a grid point used in file distdata.m for a disturbance sequence is not defined in array phi (in file phi.mat or phi.dat), the following error message will appear in the MATLAB window and program execution will stop.

An error has been detected in file formplnt.m. **A FEM disturbance grid number as referenced in file** distdata.m **is not available in file** phi.dat **or** phi.mat. **Program termination in** formplnt.

Solution: Check file distdata.m and file phi.dat or phi.mat, and restart the program.

- If a step size smaller than the minimum step size, defined by the user, is required in the controller state or plant state propagation, the following error message will appear in the MATLAB window and program execution will stop.

A step size smaller than the minimum is required in filename.m.

Note: Filename may take the name of any of the six routines that perform controller state or plant state propagation with step size control: rk23, rk32, rk34, rk45, rstiff, nlplnt4.

Solution: (1) Decrease the minimum step size allowed from the GUI window or in variable option; (2) Increase the integration error tolerance from the GUI window or in variable option.

- If the number of iterations in one integration step for the controller state or plant state propagation goes beyond the user-defined maximum number of iterations allowed, the following error message will appear in the MATLAB window and program execution will stop.

Maximum number of iterations reached without convergence in filename.m

Note: Filename may take the name of any of the six routines that perform controller state or plant state propagation with step size control: rk23, rk32, rk34, rk45, rstiff, nlplnt4.

Solution: (1) Increase the maximum number of iterations allowed per step from the GUI window or in variable option; (2) Increase the integration error tolerance from the GUI window or in variable option.

- If the number of iterations in the solution check of the nonlinear controller goes beyond the user-defined maximum number of iterations allowed, the following error message will appear in the MATLAB window and program execution will stop.

Maximum number of iterations reached without convergence in filename.m **due to solchk.**

Note: Filename.m may take the name of any of the four routines that perform controller state propagation: nlsim.m, nlsim2.m, nlsim_m.m, nlsim_m2.m.

Solution: Increase the maximum number of iterations allowed per step from the GUI window or in variable option.

- If an attempt is made to use the memory conservative feature of PLATSIM without having compiled the MEX-file for lmtime or trplomem, one of the following error messages will appear in the MATLAB window and program execution will stop.

LMTIME: **MEX-file for** lmtime **not found, cannot proceed.**

trplomem.m: **For `help' only, must run the MEX-file from** trplomem.c

Solution: Compile the MEX-files for lmtime and trplome or do not attempt to run the memory conservative option.

- If PLATSIM function `nlinitcl.m` or `nlinitol.m` executes one of the commands `exist('nlsim')`, `exist('nlsim_m')`, `exist('nlsim2')`, `exist('nlsim_m2')`, or `exist('propgateo')`, and the answer is neither 2 (the name represents an M-file) nor 3 (the name represents a MEX-file), one of the following error messages will appear in the MATLAB window and program execution will stop.

```
NLINITCL: failure setting ismex
NLINITOL: failure setting ismex
```

Solution: Make sure that none of these variables are being used in the wrong way (such as for a compiled SIMULINK function) and that the M-files (and MEX-files, if they exist) of these names are on the `matlabpath`.

NASA Langley Research Center
Hampton, VA 23681-2199
August 27, 1997

Appendix A

User-Supplied Routines for Earth Observing System EOS-AM-1 Example

The following listings are examples of the routines to be supplied to PLATSIM by the user. These examples are based on the Earth Observing System EOS-AM-1 spacecraft and are distributed with PLATSIM. The user may want to use these examples as templates for writing the user-supplied routines for the platform that the user wishes to analyze.

mkdamp.m

```
function [d]=mkdamp(omega)
%
% function [d]=mkdamp(omega)
%
% purpose: to assign modal damping ratios
%
% input variables:
%
% omega : vector containing the natural frequencies
%
% output variables:
%
% d : vector of damping ratios
%
%

% Author: Peiman G. Maghami
%         Spacecraft Controls Branch
%         NASA Langley Research Center
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% damping schedule for the EOS-AM-1 Spacecraft
%
% damping ratio = 0.2% for modes with frequency less than 15Hz
% damping ratio = 0.25% for modes with frequency greater than 15Hz
%                 but less than 50Hz
% damping ratio = 0.3% for modes with frequency greater than 50Hz
for i=1:max(size(omega))
  if omega(i)< 30.0*pi
    d(i)=0.002;
  elseif omega(i)>=30.0*pi&omega(i)<100.0*pi
    d(i)=0.0025;
  else
    d(i)=0.003;
  end
end
d=d';
```

instdata.m

```
  function [act,mout,pout,instr]=instdata
%
% function [act,mout,pout,instr]=instdata
%
% purpose: a user-defined routine to provide the grid
```

```
% point numbers, directions distribution/contribution
% factors, and identification numbers for the
% spacecraft instruments. It also provides names for
% the performance outputs.
%
%
% output variables:
%
%
% act    : control input information matrix
% mout   : measurement output information matrix
% pout   : performance output information matrix
% instr  : list of names for performance outputs
%
%
%

% Author: Peiman G. Maghami
%         Spacecraft Controls Branch
%         NASA Langley Research Center
%         August, 1993
%
% modified: P. G. Maghami
%           March, 1994
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following parameters are associated with
% EOS-AM-1 Spacecraft.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% define the spacecraft control input information
% matrix (the ACS input at the RWA)
%
act=[155003,155003,155003;
     4,     5,     6;
     1,     2,     3;
     1.0,   1.0,   1.0];
%
% define grid points for the measurement feedbacks at
% the NAVBASE
%
mout=[111091,111091,111091,111091,111091,111091;
      4,     4,     5,     5,     6,     6;
      1,     2,     3,     4,     5,     6;
      1.0,   1.0,   1.0,   1.0,   1.0,   1.0;
      0,     1,     0,     1,     0,     1];
%
% define grid points for performance outputs
% [NAVBASE, CERES1, CERES2, MISR, MODIS-N, MOPITT,
% SWIR, TIR, VNIR]
%
pout=[111091,111091,111091,350420,350420,350420,...
```

```
351420,351420,351420,333498,333498,333498,361203,...
361203,361203,396400,396400,396400,329722,329722,...
329722,326989,326989,326989,325647,325647,325647;
4,5,6,4,5,6,4,5,6,4,5,6,4,5,6,4,5,6,4,5,6,4,5,6,...
4,5,6;
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,...
20,21,22,23,24,25,26,27;
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,...
1,1,1;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
0,0,0];
%
% convert the performance output units from rads
% to arcsec
%
pout(4,:)=(180.0*3600.0/pi)*pout(4,:);
%
%
% define the performance output names, display
% and unit labels
%
instr=str2mat('1|NAVBASE Roll|NAVBASE|arcsec','2|NAVBASE Pitch|NAVBASE|
arcsec');
instr=str2mat(instr,'3|NAVBASE Yaw|NAVBASE|arcsec');
instr=str2mat(instr,'4|CERES1 Roll|CERES|arcsec','5|CERES1 Pitch|CERES|
arcsec');
instr=str2mat(instr,'6|CERES1 Yaw|CERES|arcsec');
instr=str2mat(instr,'7|CERES2 Roll|CERES|arcsec','8|CERES2 Pitch|CERES|
arcsec');
instr=str2mat(instr,'9|CERES2 Yaw|CERES|arcsec');
instr=str2mat(instr,'10|MISR Roll|MISR|arcsec','11|MISR Pitch|MISR|arcsec');
instr=str2mat(instr,'12|MISR Yaw|MISR|arcsec');
instr=str2mat(instr,'13|MODIS Roll|MODIS|arcsec','14|MODIS Pitch|MODIS|
arcsec');
instr=str2mat(instr,'15|MODIS Yaw|MODIS|arcsec');
instr=str2mat(instr,'16|MOPITT Roll|MOPITT|arcsec','17|MOPITT
Pitch|MOPITT|arcsec');
instr=str2mat(instr,'18|MOPITT Yaw|MOPITT|arcsec');
instr=str2mat(instr,'19|SWIR Roll|ASTER|arcsec','20|SWIR Pitch|ASTER|
arcsec');
instr=str2mat(instr,'21|SWIR Yaw|ASTER|arcsec');
instr=str2mat(instr,'22|TIR Roll|ASTER|arcsec','23|TIR Pitch|ASTER|arcsec');
instr=str2mat(instr,'24|TIR Yaw|ASTER|arcsec');
instr=str2mat(instr,'25|VNIR Roll|ASTER|arcsec','26|VNIR Pitch|ASTER|
arcsec');
instr=str2mat(instr,'27|VNIR Yaw|ASTER|arcsec');
%
```

distdata.m

```
function [dist,w,dt,cnames,dnames,instdat,...
          mapping]=distdata(casenum,tdflag)
%function [dist,w,dt,cnames,dnames,instdat,...
           mapping]=distdata(casenum,tdflag)
%
```

```
% Author: Sean P. Kenny
%         NASA Langley Research Center
%         Spacecraft Controls Branch
% Created:  2/14/94
%
%------------------------------------------------
if nargin == 0
  casenum=0;
  tdflag='yes';       % This will not affect the
                      % analysis, just the amount
                      % of data generated (time spent)
                      % here.
elseif nargin == 1
  tdflag='yes';
end
%
%
%
% Individual Instrument Disturbances:
%
% Defines labels for pull-down menus, also labels
% for jitter tables, and time history plots.
%
% Note:
%
% A GUI menu item may be disabled by including a
% preceding asterisk in the string variable, e.g.,
% s23='* High Gain Antenna', will be displayed,
% but cannot be selected with the mouse. This feature
% pertains to GUI mode ONLY ! Batch mode will allow
% the selection of all entries.
%
%
%
s1='TIR repoint';
s2='TIR calibrate';
s3='TIR scan';
s4='TIR chopper';
s5='TIR cryocooler LDE';
s6='MODIS scan mirror';
s7='MODIS static imbalance';
s8='MODIS dynamic imbalance';
s9='MOPITT mirror scan';
s10='MOPITT cryocooler LDE';
s11='MOPITT pressure modulated cells';
s12='Reaction Wheel Assembly case 1';
s13='* Reaction Wheel Assembly case 2';
s14='Solar Array Drive';
s15='Solar Array Thermal Snap';
%
%
%
% Combine individual cases into a matrix form using
% function "str2mat".
% Note "str2mat" allows a maximum of 10 arguments per
```

```
% call, therefore, multiple calls may be required.
% Each individual case defined above represents a
% row entry within the "dnames" string matrix.
%
dnames=str2mat(s1,s2,s3,s4,s5,s6,s7,s8,s9,s10);
dnames=str2mat(dnames,s11,s12,s13,s14,s15);
%
%
%--------------------------------------------------
%
%
%            Disturbance Category Labels
%  (labels for top-level menu items on pop-up figure)
%
ss1='ASTER-TIR';
ss2='MODIS';
ss3='MOPITT';
ss4='Misc. Disturbances';
%
% Combine Category Labels into a string matrix:
%
cnames=str2mat(ss1,ss2,ss3,ss4);
%
%
%
% Setup mapping between category labels and
% disturbance case numbers.
%
% instdat[i] corresponds to the ith row in "cnames",
% e.g., the vector instdat5 contains all case numbers
% that are associated with the fifth row in "cnames".
% The elements of the instdat[i] vector are the row
% indices within the "dnames" string matrix. For
% example, if the 4th, 5th, and 10th row entries in
% "dnames" correspond to the category in the 5th row
% of "cnames", then instdat5=[4,5,10].
%
% The vector variable "mapping" is a pointer that
% defines the number of cases in each category, e.g.,
% mapping(1)=8; implies that there are eight
% disturbance cases associated with the first
% category.
%
%
  mapping=[];
instdat1=[1,2,3,4,5];
  linst=length(instdat1);
  mapping=[mapping,linst];
instdat2=[6,7,8];
  linst=length(instdat2);
  mapping=[mapping,linst];
instdat3=[9,10,11];
  linst=length(instdat3);
  mapping=[mapping,linst];
```

```
instdat4=[12,13,14,15];
  linst=length(instdat4);
  mapping=[mapping,linst];
%
instdat=[instdat1,instdat2,instdat3,instdat4];
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Based upon menu selection, create the proper
% disturbance input vector(s).
%
%
% TIR Mirror Repointing
%
if (casenum == 1)
% 326990 ==> scanner
igrid=[326990];
idir=[4];
inum=[1];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
% get torque/force profiles for only time-domain
if strcmp(tdflag(1),'y')
  [dt,torque] = tir1;
  w(:,1)=torque';
end
%
% TIR Mirror Calibration
%
elseif (casenum == 2)
% 326944 ==> chopper
igrid=[326990];
idir=[4];
inum=[1];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = tir2;
  w(:,1)=torque';
end
%
% TIR Scanner
%
elseif (casenum == 3)
% 326990 ==> scanner
igrid=[326990 326990 326990 326990 326990 326990];
idir=[1 2 3 4 5 6];
inum=[1:6];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = tirscan1;
  w(:,1)=torque';
  [dt,torque] = tirscan2;
  w(:,2)=torque';
```

```
  [dt,torque] = tirscan3;
  w(:,3)=torque';
  [dt,torque] = tirscan4;
  w(:,4)=torque';
  [dt,torque] = tirscan5;
  w(:,5)=torque';
  [dt,torque] = tirscan6;
  w(:,6)=torque';
end
%
% TIR Chopper
%
elseif (casenum == 4)
% 326944 ==> chopper
igrid=[326944 326944];
idir=[1 2];
inum=[1:2];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = tirchop1;
  w(:,1)=torque';
  [dt,torque] = tirchop2;
  w(:,2)=torque';
end
%
% TIR Compressor/Displacer "Low Distortion Electronics" (LDE)
%
elseif (casenum == 5)
% 326992 ==> compressor
% 326993 ==> displacer
igrid=[326992 326992 326992 326992 326992 326993 326993 326993 326993 326993];
idir=[1 2 4 5 6 2 1 4 5 6];
inum=[1:10];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = tirca1;
  w(:,1)=torque';
  [dt,torque] = tirca2;
  w(:,2)=torque';
  [dt,torque] = tirca3;
  w(:,3)=torque';
  [dt,torque] = tirca4;
  w(:,4)=torque';
  [dt,torque] = tirca5;
  w(:,5)=torque';
  [dt,torque] = tirda1;
  w(:,6)=torque';
  [dt,torque] = tirda2;
  w(:,7)=torque';
  [dt,torque] = tirda3;
  w(:,8)=torque';
  [dt,torque] = tirda4;
  w(:,9)=torque';
```

```
  [dt,torque] = tirda5;
  w(:,10)=torque';
end
%
%
% MODIS scan mirror
%
elseif (casenum == 6)
% 3601 ==> averaged interface
% 361203 ==> scan mirror center
% 361342 ==> scan mirror motor/encoder
% 355349 ==> solar door 8/12/93 spk
igrid=[361342];
idir=[4];
inum=[1];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = modis1;
  w(:,1)=torque';
end
%
% MODIS static imbalance
%
elseif (casenum == 7)
% 361342 ==> scan mirror motor/encoder
igrid=[361342 361342];
idir=[2 3];
inum=[1:2];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,for1,for2] = modis2;
  w(:,1)=for1';
  w(:,2)=for2';
end
%
% MODIS dynamic imbalance
%
elseif (casenum == 8)
% 361342 ==> scan mirror motor/encoder
igrid=[361342 361342];
idir=[5 6];
inum=[1:2];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,tor1,tor2] = modis3;
  w(:,1)=tor1';
  w(:,2)=tor2';
end
%
% MOPITT mirror scan
%
elseif (casenum == 9)
```

```
% 396400 ==> scan motor 1
% 396403 ==> scan motor 2
% 3608 ==> Avg. interface
igrid=[396400];
idir=[4];
inum=[1];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = mopitt;
  w(:,1)=torque';
end
%
% MOPITT Compressor/Displacer "Low Distortion
% Electronics" (LDE)
%
elseif (casenum == 10)
% 396416 ==> compressor
% 396417 ==> displacer
igrid=[396416 396416 396416 396417 396417 396417];
idir=[2 1 5 2 1 5];
inum=[1:6];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = mopittc1;
  w(:,1)=torque';
  [dt,torque] = mopittc2;
  w(:,2)=torque';
  [dt,torque] = mopittc3;
  w(:,3)=torque';
  [dt,torque] = mopittd1;
  w(:,4)=torque';
  [dt,torque] = mopittd2;
  w(:,5)=torque';
  [dt,torque] = mopittd3;
  w(:,6)=torque';
end
%
% MOPITT pressure modulated cell (PMC)
%
elseif (casenum == 11)
% 396412 ==> PMC #1
% 396413 ==> PMC #2
igrid=[396412 396413];
idir=[1 1];
inum=[1:2];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,pmc1,pmc2] = mopmc;
  w(:,1)=pmc1';
  w(:,2)=pmc2';
end
%
```

```
%
% Reaction Wheel Assembly (RWA) Imbalance: Case 1
%
% ** static imbalance in wheel #1 **
%
elseif (casenum == 12)
% 50600 ==> RWA averaged interface
% 155003 ==> RWA 1 (farthest from C.G.)
igrid=[155003 155003 155003];
idir=[1 2 3];
inum=[1:3];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,rwax,rway,rwaz] = rwa1;
  w(:,1)=rwax';
  w(:,2)=rway';
  w(:,3)=rwaz';
end
%
% Reaction Wheel Assembly (RWA) Imbalance: Case 2
%
elseif (casenum == 13)
igrid=[155003 155003 155003];
idir=[1 2 3];
inum=[1:3];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
[dt,rwax,rway,rwaz] = rwa2;
w(:,1)=rwax';
w(:,2)=rway';
w(:,3)=rwaz';
%
% Solar Array Harmonic Drive
%
elseif (casenum == 14)
% 69090 ==> solar array drive (SAD)
igrid=[69090];
idir=[5];
inum=[1];
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = sadhd;
  w(:,1)=torque';
end
%
% Solar Array Thermal Snap
%
elseif (casenum == 15)
% 69090 ==> SAD
% 60024 ==> SA
igrid=[69090 60024 60024];
idir=[4 1 3];
inum=[1:3];
```

```
ifac=ones(size(inum));
dist=[igrid;idir;inum;ifac];
if strcmp(tdflag(1),'y')
  [dt,torque] = sa1;
  w(:,1)=torque';
  [dt,torque] = sa2;
  w(:,2)=torque';
  [dt,torque] = sa3;
  w(:,3)=torque';
end
%
end
%
% End distdata.m
```

formscs.m

```
  function [aacs,bacs,cacs,dacs]=formscs

%
% function [aacs,bacs,cacs,dacs]=formscs
%
% Purpose: To form the continuous-time spacecraft
% control system (SCS) for the space platform
%
%          Currently, this function forms the attitude
%          control system for the EOS-AM-1 Spacecraft.
%
%
% output variables:
%
% aacs  : the SCS state matrix (continuous)
% bacs  : the SCS input influence matrix (continuous)
% cacs  : the SCS output influence matrix
%

% Author: P. G. Maghami
%         Spacecraft Controls Branch
%         NASA Langley Research Center
%         December, 1992
%
% Modified: March, 1994
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 third-order Butterworth filter
kf=0.25005;
af=1.0;
bf=1.2600;
cf=0.7938;
df=kf;
numf=kf;denf=[af,bf,cf,df];
% wide-band notch filter
an=0.52335;
bn=0.19678;
cn=1.0;
```

```
kn=0.57408;
numn=[an,bn,cn];denn=[an,kn,cn];
% rate gyro
kg=1.0;
wg=12.5664;
zg=0.7071;
numg=[kg*wg*wg];deng=[1.0,2.0*zg*wg,wg*wg];
%zero order hold
taw=0.512;
%numh=taw;denh=[0.125*taw*taw,taw,1.00];
numh=1.00;denh=[0.125*taw*taw,taw,1.00];
%delay
numd=[0.125*taw*taw,taw,1.00];
dend=[0.125*taw*taw,taw,1.00];
%
krp=[4284.6;6696.4;8322.5];
kri=[30.604;40.761;105.68];
%
% rate loop compensator-roll
numrr=[krp(1),kri(1)];denrr=[1.00,0.00];
%
% rate loop compensator-pitch
numrp=[krp(2),kri(2)];denrp=[1.00,0.00];
%
% rate loop compensator-yaw
numry=[krp(3),kri(3)];denry=[1.00,0.00];
%
kp=[0.051604;0.049064;0.047159];
ki=[1.5199e-4;1.6984e-4;3.3011e-4];
%
% position loop compensator-roll
numpr=[kp(1),ki(1)];denpr=[1.00,0.00];
%
% position loop compensator-pitch
numpp=[kp(2),ki(2)];denpp=[1.00,0.00];
%
% position loop compensator-yaw
numpy=[kp(3),ki(3)];denpy=[1.00,0.00];
%
% rate loop total compensation
%
%
% transform (numg,deng) to state-space
%
[a1,b1,c1,d1]=tf2ss(numg,deng);
%
%
% transform the position-loop combined TFs to
% state-space
%
[ap1,bp1,cp1,dp1]=tf2ss(numpr,denpr);
[ap2,bp2,cp2,dp2]=tf2ss(numpp,denpp);
[ap3,bp3,cp3,dp3]=tf2ss(numpy,denpy);
%
%
```

```
% combine the position loop and rate loop filters
%
[ac1,bc1,cc1,dc1]=...
                append(ap1,bp1,cp1,dp1,a1,b1,c1,d1);
[ac2,bc2,cc2,dc2]=...
                append(ap2,bp2,cp2,dp2,a1,b1,c1,d1);
[ac3,bc3,cc3,dc3]=...
                append(ap3,bp3,cp3,dp3,a1,b1,c1,d1);
%
cc1=cc1(1,:)+cc1(2,:);dc1=dc1(1,:)+dc1(2,:);
cc2=cc2(1,:)+cc2(2,:);dc2=dc2(1,:)+dc2(2,:);
cc3=cc3(1,:)+cc3(2,:);dc3=dc3(1,:)+dc3(2,:);
%
% combine the butterworth filter and the notch filter
%
[num1,den1]=series(numf,denf,numn,denn);
%
% combine the rate loop compensator in series
% with the notch filter
%
[num31,den31]=series(numrr,denrr,num1,den1);
[num32,den32]=series(numrp,denrp,num1,den1);
[num33,den33]=series(numry,denry,num1,den1);
%
% add the zero order hold in series
%
[num41,den41]=series(num31,den31,numh,denh);
[num42,den42]=series(num32,den32,numh,denh);
[num43,den43]=series(num33,den33,numh,denh);
%
% add the time delay in series
%
[numinr,deninr]=series(num41,den41,numd,dend);
[numinp,deninp]=series(num42,den42,numd,dend);
[numiny,deniny]=series(num43,den43,numd,dend);
%
% transform the rate loop TFs to state-space
%
[ar1,br1,cr1,dr1]=tf2ss(numinr,deninr);
[ar2,br2,cr2,dr2]=tf2ss(numinp,deninp);
[ar3,br3,cr3,dr3]=tf2ss(numiny,deniny);
%
%
% Form the SCS state-space model
%
%
[aacs1,bacs1,cacs1,dacs1]=...
             series(ac1,bc1,cc1,dc1,ar1,br1,cr1,dr1);
[aacs2,bacs2,cacs2,dacs2]=...
             series(ac2,bc2,cc2,dc2,ar2,br2,cr2,dr2);
[aacs3,bacs3,cacs3,dacs3]=...
             series(ac3,bc3,cc3,dc3,ar3,br3,cr3,dr3);
%
[aacs,bacs,cacs,dacs]=append(aacs1,bacs1,...
                cacs1,dacs1,aacs2,bacs2,cacs2,dacs2);
```

```
[aacs,bacs,cacs,dacs]=append(aacs,bacs,...
                cacs,dacs,aacs3,bacs3,cacs3,dacs3);
%
% apply the scale factor KW of the reaction wheels
% (units in lb-in/ssc)
%
kw=0.22*12.00;
cacs=cacs*kw;
%
```

Appendix B

S-Function Representation of Earth Observing System EOS-AM-1 Attitude Control System

The following listing is an example of an S-function representation of an attitude control system. This example is based on the Earth Observing System EOS-AM-1 spacecraft and is distributed with PLATSIM. The user may want to use this example as a template for writing the S-function routines for the spacecraft that the user wishes to analyze.

stiction.m

```
function [yout]=stiction(x,u,t,flag)
%
% Purpose: A MATLAB S-function that defines the
% EOS-AM-1 attitude control system with the reaction
% wheel models included. The reaction wheel friction
% nonlinearities and stiction are modeled. However,
% an uncontrolled model of the wheel is used (no
% wheel speed controller).
%
% Inputs: Standard MATLAB S-function inputs.
%
% Outputs: Standard MATLAB S-function outputs.
%
% Author: Peiman G. Maghami
%         Guidance and Control Branch
%         NASA Langley Research Center
%         Hampton, VA 23681-0001
%
% Created: September, 1995
% Modified: March, 1996
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global PLAT_ac PLAT_bc PLAT_cc PLAT_nc
%
if flag==0
%
% if flag=0, the number of the continuous states,
% discrete states, outputs,and inputs are required.
%
 sizes=[51,0,6,6,0,0];
end
%
%
% call "formscs.m" the first time to generate the
% ACS system matrices
%
% The first "nc" states correspond to the states of
% the linear ACS controller.
% The states "nc+1" to "nc+3" represent the reaction
% wheel speeds.
%
if exist('PLAT_ac')~=1
[PLAT_ac,PLAT_bc,PLAT_cc,PLAT_dc]=formscs;
nc=size(PLAT_ac,1);
end
```

```
%
% compute the required torques
%
uc=PLAT_cc*x(1:nc);
%
% compute the reaction wheel drag torques
%
ws=x(nc+1:nc+3);
tdrag=0.0232215-0.0209812*exp(-0.0035787*abs(ws));
izws=find(x(nc+1:nc+3)==0);
inzws=find(x(nc+1:nc+3)~=0);
tdrag(inzws)=12*sign(ws(inzws)).*tdrag(inzws);
tdrag(izws)=12*sign(uc(izws)).*tdrag(izws);
%
if flag==1
%
% if flag=1, the derivatives of the continuous-time
% variables are required.
%
yout=zeros(size(x));
%
% compute controller state derivatives
%
ydot=PLAT_ac*x(1:nc)+PLAT_bc*u;
%
% compute reaction wheel accelerations
%
 wdot=(1/(12*0.103))*(uc-tdrag);
%
 for i=1:3
%
% if a wheel is under stiction, zero out the wheel
% accelerations
%
  if abs(uc(i))<=12*(0.0232215-0.0209812) & abs(x(nc+i))<1.00e-12;
    wdot(i)=0;
  end
 end
%
yout=[ydot;wdot];
%
elseif flag==2
%
% if flag=2, the system outputs are required.
%
% The first three outputs are the reaction torques
% and the next three outputs are the requested
% torques.
%
yout=zeros(6,1);
%
% compute reaction torques
%
 for i=1:3
%
```

```
% if a wheel is under stiction, set the reaction
% torque to zero.
%
  if abs(uc(i))<=12*(0.0232215-0.0209812) & abs(x(nc+i))<1.00e-12;
   yout(i)=0;
  else
   yout(i)=(uc(i)-tdrag(i));
  end
 end
%
  yout(4:6)=uc(1:3);
end
```

Appendix C

Solution Integrity Checking File

The following listing is an example of solution integrity checking. This file corresponds to the example given in the Solution Check File on page 10. The user may want to use this example as a template for writing a solution integrity checking file for the spacecraft that the user wishes to analyze.

solchk.m

```
  function [hout,z2,u2,isolok]=solchk(z1,u1,z2,u2,h)
%
% function [hout,z2,u2,isolok]=solchk(z1,u1,z2,u2,h)
%
% purpose: Generally, to determine if a user-defined
% solution discontinuity condition has occurred in the
% nonlinear control system during the last
% integration update, and take action as necessary
% to reset the integration step size or to redefine
% the controller states and outputs. This particular
% file is written to handle the discontinuities that
% are brought on due to the reaction wheel stiction
% for the EOS-AM-1 spacecraft.
%
%
% input variables:
%
% z1 : last controller state vector
% u1 : last controller output vector
% z2 : current controller state vector
% u2 : current controller output vector
% h  : integration step size
%
% output variables:
%
% hout   : new integration step size to be tried by
%          the integration algorithm.
% z2     : updated current controller state vector
% u2     : updated current controller output vector
% isolok : an integer scalar. If isolok=1, the
%          current solution is accepted, otherwise, the
%          solution is rejected and the integration is
%          to be performed again with hout.
%
%

% Author: Peiman G. Maghami
%         Guidance and Control Branch
%         NASA Langley Research Center
%
% Created : March, 1995
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global PLAT_flag
%
```

```
% if PLAT_flag(i)=0, wheel no. "i" is in non-stiction
% condition with potential for entering stiction
% condition
% if PLAT_flag(i)=1, wheel no. "i" is in stiction
% condition with potential for leaving stiction
% condition
%
% nwheels : number of reaction wheels
%
nwheels=3;
%
% nc : the position of the wheel speed variables in
% the nonlinear controller state vector
nc=[49,50,51];
%
% nrt : the position of the reaction torque levels of
% the wheels in the nonlinear controller output
% vector
%
nrt=[1,2,3];
%
% nat : the position of the applied torque levels to
% the wheels in the nonlinear controller output
% vector
%
nat=[4,5,6];
%
% define the stiction torque
%
tstic=12*(0.0232215-0.0209812);
%
% define minimum step for the cross-over from one
% condition to the next
%
hmin=1.00e-5*h;
%
% initialize the wheels flags. If the initial wheel
% speed is zero, assume stiction condition, and set
PLAT_flag(i)=1; otherwise, set PLAT_flag(i)=0.
%
if exist('PLAT_flag')~=1
 for i=1:nwheel
  if z1(nc(i))==0
   PLAT_flag(i)=1;
  else
   PLAT_flag(i)=0;
  end
 end
end
%
% loop around the number of reaction wheels
%
for i=1:nwheel
   hout(i)=h;
%
```

```
   if PLAT_flag(i)==0
%
% check wheel speed zero crossing
%
      if (z1(nc(i))*z2(nc(i)))<=0
%
% compute estimated location of the wheel speed zero
% crossing
%
         if (z2(nc(i))-z1(nc(i)))~=0
            wc=-z1(nc(i))*h/(z2(nc(i))-z1(nc(i)));
         else
            wc=h;
         end
%
% compute estimated requested torque @ wc
%
         uc=u1(nt(i))+((u2(nt(i))-u1(nt(i)))/h)*wc;
%
% if estimated requested torque @ wc is less than the
% stiction torque, stiction has occurred.
%
         if abs(uc)<tstic
%
% if stiction has occurred at the end of the time
% interval, set the appropriate flag and accept the
% solution
%
            if wc==h
               PLAT_flag(i)=1;
               isoloki(i)=1;
%
% if the stiction condition is estimated to have
% occurred at a time between hmin and h-hmin, reject
% the solution, and retry the last step with step
% size set at 0.9*wc.
%
            elseif wc>=hmin & wc<(h-hmin)
               isoloki(i)=0;
               hout(i)=wc*0.9;
%
% if the stiction condition is estimated to have
% occurred at a time between 0
% and hmin, assume the cross-over from non-stiction
% to stiction has taken place;
% accept the solution, and zero out the wheel speed
% and reaction torque
%
            elseif wc<hmin & wc>0
               PLAT_flag(i)=1;
               isoloki(i)=1;
               hout(i)=wc;
               z2=z1;
               z2(nc(i))=0;
               u2=u1;
               u2(nrt(i))=0;
```

```
%
% if the stiction condition is estimated to have
% occurred at a time between h-hmin and h, assume
% the cross-over from non-stiction to stiction has
% taken place; accept the solution, and zero out the
% wheel speed and reaction torque
%
            elseif wc>=h-hmin
               PLAT_flag(i)=1;
               isoloki(i)=1;
               z2(nc(i))=0;
               u2(nrt(i))=0;
            end
         else
            isoloki(i)=1;
         end
      else
         isoloki(i)=1;
      end
 %
   elseif PLAT_flag(i)==1
%
% check the requested torque levels
%
      if abs(u2(nat(i)))>=tstic
%
% compute estimated location of the requested torque
% crossing the stiction torque
%
         if (u2(nat(i))-u1(nat(i)))~=0
            tc=(sign(u2(nat(i)))*tstic-u1(nat(i)))...
               *h/(u2(nat(i))-u1(nat(i)));
         else
            tc=h;
         end
%
% if non-stiction condition has occurred at the end
% of the time interval, set the appropriate flag and
% accept the solution
%
         if tc==h
            PLAT_flag(i)=0;
            isoloki(i)=1;
%
% if the non-stiction condition is estimated to have
% occurred at a time between hmin and h-hmin, reject
% the solution, and retry the last step with step
% size set at 0.9*tc.
%
         elseif tc>=hmin & tc<(h-hmin)
            isoloki(i)=0;
            hout(i)=tc*0.9;
%
% if the non-stiction condition is estimated to have
% occurred at a time between 0 and hmin, assume the
```

```
% cross-over from stiction to non-stiction has taken
% place; accept the solution, and zero out the wheel
% speed and reaction torque, and set the applied
% torque to tstic
%
         elseif tc<hmin & tc>0
            PLAT_flag(i)=0;
            isoloki(i)=1;
            hout(i)=tc;
            z2=z1;
            z2(nc(i))=0;
            u2=u1;
            u2(nrt(i))=0;
            u2(nat(i))=sign(u1(nat(i)))*tstic;
%
% if the non-stiction condition is estimated to have
% occurred at a time between h-hmin and h, assume the
% cross-over from stiction to non-stiction has taken
% place; accept the solution, and zero out the wheel
% speed and reaction torque, and set the applied
% torque to tstic
%
         elseif tc>=h-hmin
            PLAT_flag(i)=0;
            isoloki(i)=1;
            z2(nc(i))=0;
            u2(nrt(i))=0;
            u2(nat(i))=sign(u1(nat(i)))*tstic;
         end
      else
         isoloki(i)=1;
      end
%
% end if-loop on PLAT_flag
%
   end
%
% end loop on i
%
end
%
% Determine the new step size to be taken by the
% integration. If no changes in the condition of any
% of the wheels are observed, e.g., any wheel that
% was in stiction remains in stiction, or any wheel
% that was not in stiction condition remain outside
% the stiction region, the solution is accepted
% (isolok is set to 1) and the new step size is the
% same as the old one. Otherwise, the solution is
% rejected (isolok is set to 0), and the recommended
% new step size is set to the minimum of the step
% sizes for the each of three wheels.
%
hout=min(h,min(hout));
isolok=min(isoloki);
%
```

Appendix D

Time Domain Output and Frequency Domain Output for Earth Observing System EOS-AM-1 Example

Time Domain Output for EOS-AM-1

As an example, suppose that PLATSIM runs using the default values of all execution time parameters and the example data distributed with the PLATSIM software, which is based on the EOS-AM-1 spacecraft. Suppose further that the disturbance scenario selected is "MODIS static imbalance"; then the following files are generated by PLATSIM:

MODIS_static_imbalance_1_time.mat	MODIS_static_imbalance_2_time.mat
MODIS_static_imbalance_time.mat	
CERES1_Pitch_1_time.eps	MISR_Pitch_1_time.eps
CERES1_Pitch_2_time.eps	MISR_Pitch_2_time.eps
CERES1_Roll_1_time.eps	MISR_Roll_1_time.eps
CERES1_Roll_2_time.eps	MISR_Roll_2_time.eps
CERES1_Yaw_1_time.eps	MISR_Yaw_1_time.eps
CERES1_Yaw_2_time.eps	MISR_Yaw_2_time.eps
CERES2_Pitch_1_time.eps	
CERES2_Pitch_2_time.eps	
CERES2_Roll_1_time.eps	
CERES2_Roll_2_time.eps	
CERES2_Yaw_1_time.eps	
CERES2_Yaw_2_time.eps	
MODIS_Pitch_1_time.eps	MOPITT_Pitch_1_time.eps
MODIS_Pitch_2_time.eps	MOPITT_Pitch_2_time.eps
MODIS_Roll_1_time.eps	MOPITT_Roll_1_time.eps
MODIS_Roll_2_time.eps	MOPITT_Roll_2_time.eps
MODIS_Yaw_1_time.eps	MOPITT_Yaw_1_time.eps
MODIS_Yaw_2_time.eps	MOPITT_Yaw_2_time.eps
NAVBASE_Pitch_1_time.eps	SWIR_Pitch_1_time.eps
NAVBASE_Pitch_2_time.eps	SWIR_Pitch_2_time.eps
NAVBASE_Roll_1_time.eps	SWIR_Roll_1_time.eps
NAVBASE_Roll_2_time.eps	SWIR_Roll_2_time.eps
NAVBASE_Yaw_1_time.eps	SWIR_Yaw_1_time.eps
NAVBASE_Yaw_2_time.eps	SWIR_Yaw_2_time.eps
TIR_Pitch_1_time.eps	VNIR_Pitch_1_time.eps
TIR_Pitch_2_time.eps	VNIR_Pitch_2_time.eps
TIR_Roll_1_time.eps	VNIR_Roll_1_time.eps
TIR_Roll_2_time.eps	VNIR_Roll_2_time.eps
TIR_Yaw_1_time.eps	VNIR_Yaw_1_time.eps
TIR_Yaw_2_time.eps	VNIR_Yaw_2_time.eps
MODIS_static_imbalance_1_jitr.out	MODIS_static_imbalance_1_jitr.ps
MODIS_static_imbalance_2_jitr.out	MODIS_static_imbalance_2_jitr.ps
MODIS_static_imbalance_jitr.out	MODIS_static_imbalance_jitr.ps

The reduced time histories for the EOS-AM-1 instruments, namely, CERES1, CERES2, MISR, MODIS, MOPITT, NAVBASE, SWIR, TIR, and VNIR, are provided in an encapsulated PostScript form in the corresponding files with extension ".eps" for roll, pitch, and yaw axes. For example, the file `CERES1_Roll_1_time.eps` provides the roll time history for the CERES1 instrument due to the first MODIS static imbalance disturbance element (vector), and `VNIR_Yaw_2_time.eps` provides the yaw time history for VNIR instrument due to the second MODIS static imbalance disturbance element (vector).

A typical time history output as encapsulated in file `MISR_Roll_2_time.eps` is shown in figure D1. This time history was generated using the 703-mode model of the EOS-AM-1 spacecraft being distributed with the PLATSIM software.

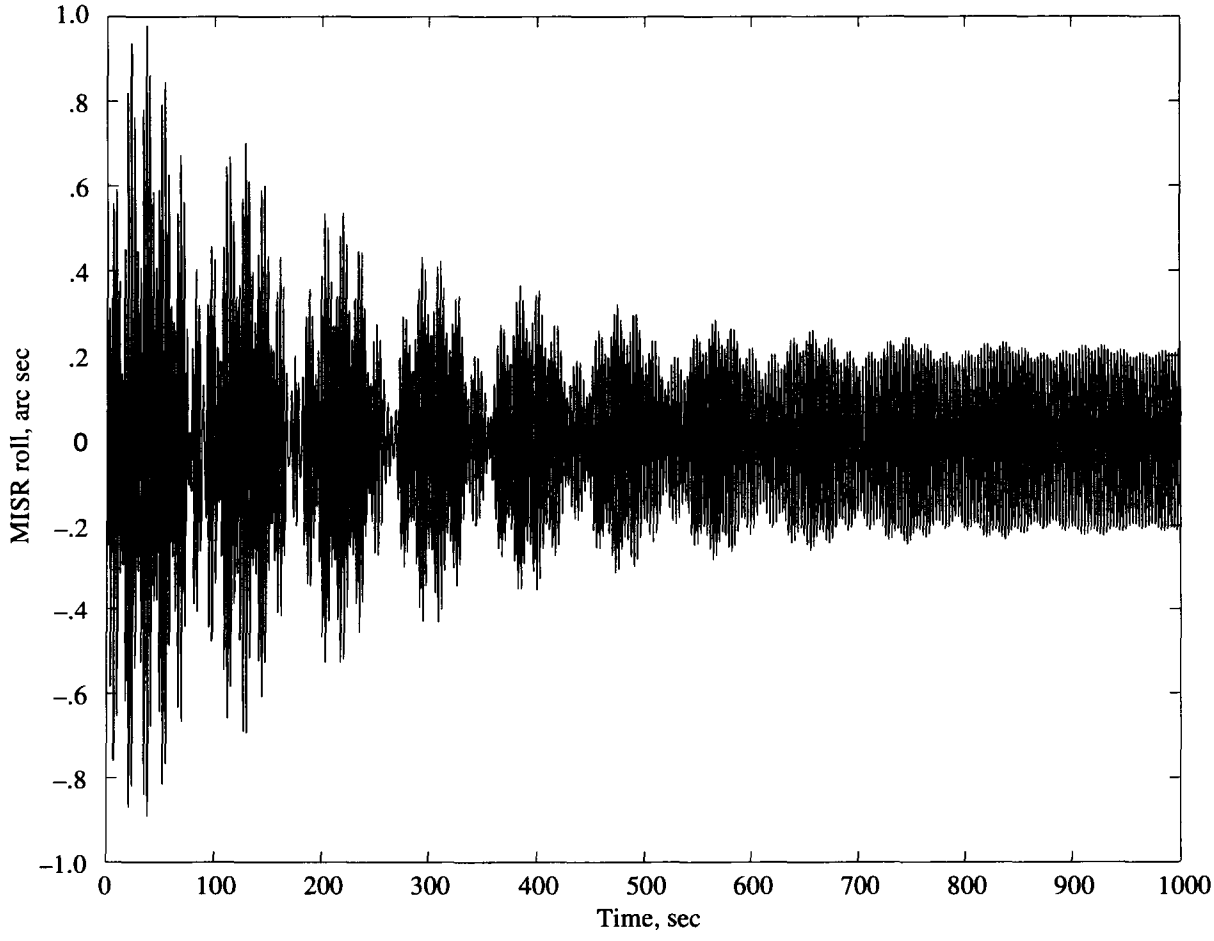


Figure D1. Example of time history plot. Time response for disturbance sequence MODIS static imbalance (2); 29 Apr 1994.

Files `MODIS_static_imbalance_1_time.mat` and `MODIS_static_imbalance_2_time.mat` are MATLAB binary files which include the reduced time history data for all EOS-AM-1 instruments and the corresponding jitter values (in variables `JITTER1` and `JITTER2`) for the first and the second MODIS static imbalance disturbance elements (vectors), respectively. File `MODIS_static_imbalance_time.mat` is a MATLAB binary file which has the overall jitter values (in variable `JITTER`). Files `MODIS_static_imbalance_1_jitr.out` and `MODIS_static_imbalance_2_jitr.out` are ASCII files which contain the jitter values, in a tabular form, for the first and the second MODIS static imbalance disturbance elements (vectors), respectively. File `MODIS_static_imbalance_jitr.out` is an ASCII file containing the total jitter values in a tabular form, obtained through the direct summation of jitter values in files `MODIS_static_imbalance_1_time.mat` and `MODIS_static_imbalance_2_time.mat`. The jitter data written to file `MODIS_static_imbalance_1_jitr.out` looks as follows:

Disturbance Source: MODIS static imbalance(1)

Window	1.00	1.80	9.00	55.00	420.00	480.00	1000.00
NAVBASE Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
NAVBASE Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
NAVBASE Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
CERES1 Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
CERES1 Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
CERES1 Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
CERES2 Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
CERES2 Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
CERES2 Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
MISR Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
MISR Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
MISR Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
MODIS Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
MODIS Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
MODIS Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
MOPITT Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
MOPITT Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
MOPITT Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
SWIR Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
SWIR Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
SWIR Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
TIR Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
TIR Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
TIR Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
VNIR Roll	3.19	3.82	7.55	12.50	12.50	12.50	12.50
VNIR Pitch	0.42	0.55	1.40	2.20	2.20	2.20	2.20
VNIR Yaw	3.97	4.96	14.19	26.88	26.88	26.88	26.88
RUN DATE: 29-Apr-94							

Files `MODIS_static_imbalance_1_jitr.ps`, `MODIS_static_imbalance_2_jitr.ps`, and `MODIS_static_imbalance_jitr.ps` are equivalent to their "`.out`" counterpart, except that they are in PostScript form.

Frequency Domain Output for EOS-AM-1

As an example, suppose that PLATSIM runs using the default values of all execution time parameters, except `tdflag = 'n'` (which selects frequency domain analysis). Suppose also that PLATSIM runs using the example data distributed with the PLATSIM software which is based on the EOS-AM-1 spacecraft. Suppose further that the disturbance scenario selected is "MODIS static imbalance"; then the following files are generated by PLATSIM:

MODIS_static_imbalance_freq.mat	
CERES1_Pitch_1_freq.eps	MISR_Pitch_1_freq.eps
CERES1_Pitch_2_freq.eps	MISR_Pitch_2_freq.eps
CERES1_Roll_1_freq.eps	MISR_Roll_1_freq.eps
CERES1_Roll_2_freq.eps	MISR_Roll_2_freq.eps
CERES1_Yaw_1_freq.eps	MISR_Yaw_1_freq.eps
CERES1_Yaw_2_freq.eps	MISR_Yaw_2_freq.eps
CERES2_Pitch_1_freq.eps	
CERES2_Pitch_2_freq.eps	
CERES2_Roll_1_freq.eps	
CERES2_Roll_2_freq.eps	
CERES2_Yaw_1_freq.eps	
CERES2_Yaw_2_freq.eps	
MODIS_Pitch_1_freq.eps	MOPITT_Pitch_1_freq.eps
MODIS_Pitch_2_freq.eps	MOPITT_Pitch_2_freq.eps
MODIS_Roll_1_freq.eps	MOPITT_Roll_1_freq.eps
MODIS_Roll_2_freq.eps	MOPITT_Roll_2_freq.eps
MODIS_Yaw_1_freq.eps	MOPITT_Yaw_1_freq.eps
MODIS_Yaw_2_freq.eps	MOPITT_Yaw_2_freq.eps
NAVBASE_Pitch_1_freq.eps	SWIR_Pitch_1_freq.eps
NAVBASE_Pitch_2_freq.eps	SWIR_Pitch_2_freq.eps
NAVBASE_Roll_1_freq.eps	SWIR_Roll_1_freq.eps
NAVBASE_Roll_2_freq.eps	SWIR_Roll_2_freq.eps
NAVBASE_Yaw_1_freq.eps	SWIR_Yaw_1_freq.eps
NAVBASE_Yaw_2_freq.eps	SWIR_Yaw_2_freq.eps
TIR_Pitch_1_freq.eps	VNIR_Pitch_1_freq.eps
TIR_Pitch_2_freq.eps	VNIR_Pitch_2_freq.eps
TIR_Roll_1_freq.eps	VNIR_Roll_1_freq.eps
TIR_Roll_2_freq.eps	VNIR_Roll_2_freq.eps
TIR_Yaw_1_freq.eps	VNIR_Yaw_1_freq.eps
TIR_Yaw_2_freq.eps	VNIR_Yaw_2_freq.eps

The Bode plots for the EOS-AM-1 instruments, namely, CERES1, CERES2, MISR, MODIS, MOPITT, NAVBASE, SWIR, TIR, and VNIR, are provided in an encapsulated PostScript form in the corresponding files with extension ".eps" for roll, pitch, and yaw axes. For example, the file `CERES1_Roll_1_freq.eps` provides the Bode plot for the roll response of the CERES1 instrument due to the first MODIS static imbalance disturbance element (vector), and `VNIR_Yaw_2_time.eps` provides the Bode plot for the yaw response of the VNIR instrument due to the second MODIS static imbalance disturbance element (vector). A Bode plot output would look as in figure D2 on page 71. This Bode plot was generated by using the 703-mode model being distributed with the PLATSIM software.

File `MODIS_static_imbalance_freq.mat` is a MATLAB binary file containing the results of the frequency domain calculation in the form indicated in "Frequency response matrix" on page 37.

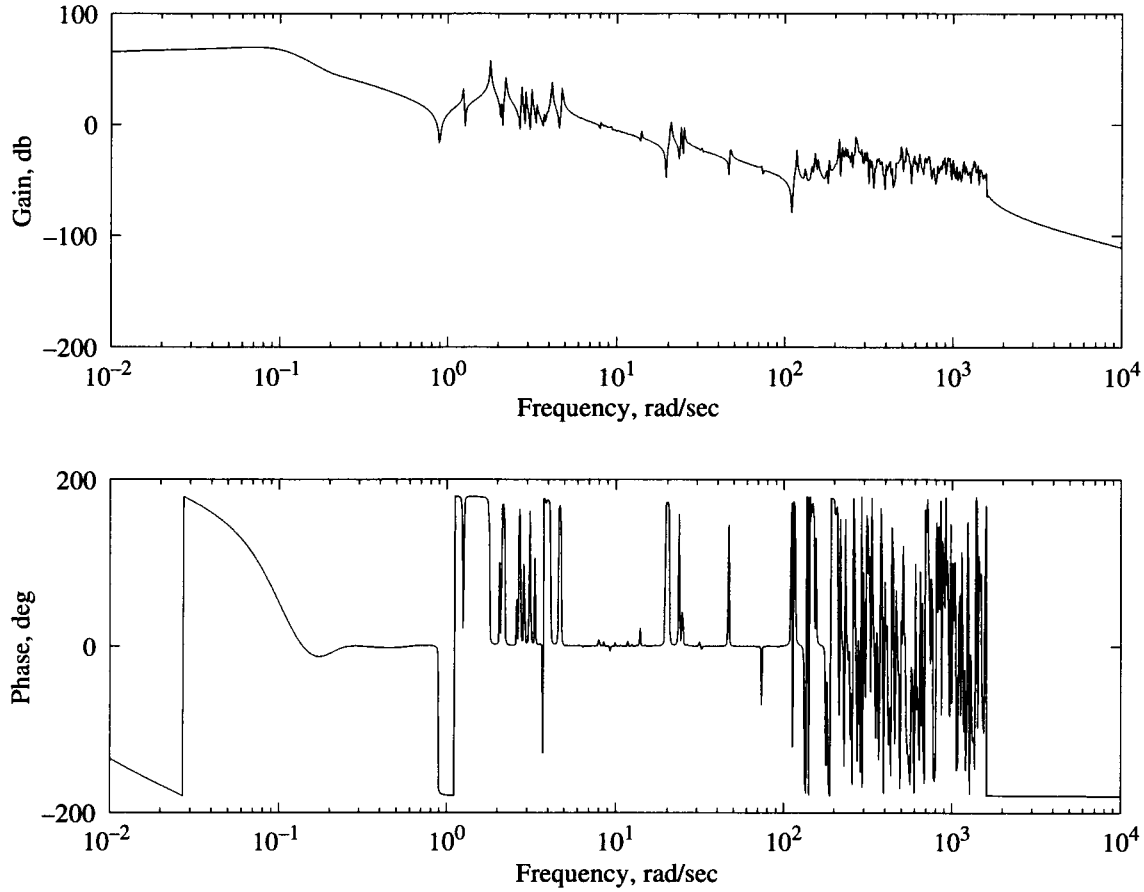


Figure D2. Example of Bode plot. Frequency response at MISR roll, arcsec. Disturbance: MODIS static imbalance (2); 29 APR 1994.

References

1. *MATLAB Reference Guide*. The MathWorks, Inc., 1993.
2. Asrar, Ghassem; and Dokken, David Jon, eds.: *1993 Earth Observing System Reference Handbook*. NASA NP-202, 1993.
3. Maghami, Periman; Kenny, Sean P.; and Giesy, Daniel P.: PLATSIM: *An Efficient Linear Simulation and Analysis Package for Large-Order Flexible Systems*. NASA TP-3519, 1995.
4. Giesy, D. P.: Efficient Calculation of a Jitter/Stability Metric. *J. Spacecr. & Rockets*, vol. 34, no. 4, July–Aug. 1997, pp. 549–557.
5. Maghami, P. G.; and Giesy, D. P.: Efficient Computation of Closed-Loop Frequency Response for Large-Order Flexible Systems. *J. Guid., Control, & Dyn.*, vol. 19, no. 4, July 1996, pp. 780–786.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1997	Technical Memorandum

4. TITLE AND SUBTITLE
PLATSIM: A Simulation and Analysis Package for Large-Order Flexible Systems (Version 2.0)

5. FUNDING NUMBERS
WU 233-10-14-05

6. AUTHOR(S)
Peiman G. Maghami, Sean P. Kenny, and Daniel P. Giesy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17608

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-4790

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 18
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The software package PLATSIM provides efficient time and frequency domain analysis of large-order generic space platforms. PLATSIM can perform open-loop analysis or closed-loop analysis with linear or nonlinear control system models. PLATSIM exploits the particular form of sparsity of the plant matrices for very efficient linear and nonlinear time domain analysis, as well as frequency domain analysis. A new, original algorithm for the efficient computation of open-loop and closed-loop frequency response functions for large-order systems has been developed and is implemented within the package. Furthermore, a novel and efficient jitter analysis routine which determines jitter and stability values from time simulations in a very efficient manner has been developed and is incorporated in the PLATSIM package. In the time domain analysis, PLATSIM simulates the response of the space platform to disturbances and calculates the jitter and stability values from the response time histories. In the frequency domain analysis, PLATSIM calculates frequency response function matrices and provides the corresponding Bode plots. The PLATSIM software package is written in MATLAB script language. A graphical user interface is developed in the package to provide convenient access to its various features.

14. SUBJECT TERMS
PLATSIM; Time domain analysis; Frequency domain analysis; Jitter analysis; Linear control; Nonlinear control; Flexible structures; Large-order systems

15. NUMBER OF PAGES
80

16. PRICE CODE
A05

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