### REVISION RECORD

<table>
<thead>
<tr>
<th>Rev</th>
<th>Description</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>• Documents release 2008004&lt;br&gt;• <em>inms_get_data</em> updated to read TRND data.&lt;br&gt;• Documents re-write of <em>inmsIonSensitivity</em>&lt;br&gt;• Routines to read PDS spreadsheet added&lt;br&gt;• Support routines for reading spreadsheets documented.</td>
<td>4 Jan 2008</td>
<td>D. Gell</td>
</tr>
<tr>
<td>H</td>
<td>• Documents release 2007267&lt;br&gt;• Adds description of the mean and total spectra computations&lt;br&gt;• Adds description of the spectra accumulation process&lt;br&gt;• Keyword /POISSON deleted from <em>inmsComputeMeanSpectra</em>&lt;br&gt;• Adds routines to support widget GUI applications&lt;br&gt;• Adds routines to read ion calibration and related information&lt;br&gt;• Adds routines to save and restore data</td>
<td>24 Sep 2007</td>
<td>D. Gell</td>
</tr>
<tr>
<td>G</td>
<td>• Documents release 2007170&lt;br&gt;• Adds routines for SPICE kernel management&lt;br&gt;• Adds routine to make PDS labels&lt;br&gt;• Describes use of IIS frame kernel&lt;br&gt;• Adds geometry display routine&lt;br&gt;• Add general auxiliary value computation function&lt;br&gt;• Color tables now are not changed by <em>inmsPreparePlot</em> unless specifically commanded.&lt;br&gt;• /PORTRAIT keyword added to <em>inmsMakeWindow</em>&lt;br&gt;• Added routine to dump a structure into a CSV file&lt;br&gt;• Added table of auxiliary files to appendix B.&lt;br&gt;• Updated list of depreciated routines.</td>
<td>20 Jun 07</td>
<td>D. Gell</td>
</tr>
<tr>
<td>F</td>
<td>• Documents release 2007018&lt;br&gt;• Additional keywords defined for <em>inmsMakeIonSpectra</em>&lt;br&gt;• Keyword /ramangle added to <em>inmsPlotStackedSpectra</em>&lt;br&gt;• Added procedure <em>inmsTabulateSpectra</em>, which produces a table of information from the spectra</td>
<td>18 Jan 07</td>
<td>D. Gell</td>
</tr>
<tr>
<td>Rev</td>
<td>Description</td>
<td>Date</td>
<td>Author</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>--------</td>
</tr>
</tbody>
</table>
| E   | • Documents release 2007008  
     • /ingress and /egress keywords added to data selection  
     • Order of inms_make_ion_spectra arguments changed to conform to usage of other routines  
     • T17 ion spectra properly handled  
     • inms_plot_histogram now draws grid  
     • inms_plot_mass_profile has option to connect plot marker symbols | 8-Jan-07 | D. Gell |
| D   | • Documents release 2006341  
     • Arguments to inms_deconvolve & inms_make_profile changed  
     • Adds routines supporting REU calibration analysis  
     • Adds routines for ion sensitivity computations  
     • Adds support for energy scan mode  
     • Modifies the syntax for specifying calibration species  
     • Adds one field to the calibration data structure  
     • Adds control of window size and location to inms_make_window.  
     • Adds support in inms_prepare_plot for tiff and jpeg plots.  
     • Improves error handling and reporting  
     • Add instructions for directly plotting data in structures.  
     • Changes plot characteristics for ps, png, tiff & jpeg to improve legibility when including in electronic presentations (line and character weights, character size)  
     • Deprecated routines inms_plot_cal, inms_compute_density inms_init_ss_position and inms_ss_position | 4-Dec-06 | D. Gell |
| C   | • Adds /rate switch to inms_plot_mt_spectra inms_plot_mt_line, inms_plot_stacked_spectra and inms_plot_mass_history  
     • Deletes calfactor and /mole keywords from inms_deconvolve and documented its critfreq keyword  
     • Adds QUARTER type to inms_neat_ticks                                                                                                                                                               | 18-Nov-05 | D. Gell |
## REVISION RECORD

<table>
<thead>
<tr>
<th>Rev</th>
<th>Description</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
</table>
| B   | Adds support for MS Windows  
|     | Temperature keywords added to inms_ram_coefficient  
|     | Adds inms_select_cal, inms_make_profiles, inms_plot_density_profiles  
|     | Simply creation of multiple PNG files  
|     | Outlier removal added to inms_deconvolve  
|     | Augment spectra structure with additional ancillary data: latitude, west longitude, solar zenith angle, local solar time.  
|     | Auxiliary axes added to stacked spectra and mass history plots, inms_plot_stacked_spectra, inms_plot_mass_history  
|     | Added control of calibration data plot format  
|     | Bug fixes, see release notes included in distribution | 20-Sep-05 | D. Gell |
| A   | Initial Release of PDS Archive  
|     | Supports 0.125 AMU (high resolution) mass scans  
|     | Adds support for housekeeping packet files  
|     | inms_get_data always supplied data in chronological order | 1-Jul-05 | D. Gell |
| X3  | Documents TB support version  
|     | Adds descriptions of inms_grid_spectra, plot_stacked_spectra  
|     | Additional information on prerequisites and installation  
|     | Adds section on time manipulation routines | 5-Apr-05 | D. Gell |
| X2  | Correct typos and incorporate suggestions  
|     | Update argument lists for plotting routines | 11-Mar-05 | D. Gell |
| X1  | First Draft | 28-Jan-05 | D. Gell |
This page intentionally left blank
Contents

1. References ........................................ 9
2. Introduction ...................................... 9
   2.1 Conventions .................................. 9
   2.2 Prerequisites ................................ 10
3. Data Structures .................................. 10
4. Reading Data ...................................... 11
   4.1 Reading L1A, Housekeeping and Trend Files .................................. 11
   4.2 Reading Generic PDS Spreadsheet Files .................................. 13
5. Selecting Data and Forming Mass Spectra ......................... 14
   5.1 Selection Criteria Specification ................. 14
   5.2 Data Selection ................................ 16
   5.3 Mass Spectra Formation ......................... 16
6. Data Display ...................................... 19
   6.1 L1A Data Plots ................................ 20
   6.2 Spectra Plots .................................. 29
   6.3 Housekeeping Data Plots ......................... 36
   6.4 Direct Plotting ................................ 37
7. Calibration Data .................................. 38
   7.1 Neutral Gas Mode .............................. 38
   7.2 Ion Mode Calibration Data ....................... 43
8. Manipulating Data ................................ 45
   8.1 Data Validation ................................. 45
   8.2 Geometry ..................................... 46
   8.3 Instrument Response ........................... 47
   8.4 Spectral Calculations ......................... 47
8.5 Time Conversions and Manipulation .......................... 49
8.6 Miscellaneous .................................. 52
   9.1 Outline of the method .......................... 53
   9.2 The Deconvolution Procedure ................. 54
   9.3 Deconvolved Density Profiles .................. 56
   9.4 Profile Display (inms_plot_density_profiles) ...... 57
10. Determination of Ion Abundance ............................. 58
11. Support Routines .................................. 59
   11.1 Graphics Support ............................. 59
   11.2 PDS Spreadsheet Routines ...................... 68
   11.3 Programming Utilities ......................... 70
   11.4 Widget Support ............................... 73
   11.5 Spice Kernel Management ...................... 74
12. Installation Issues .................................. 77
   12.1 Data Distribution ............................. 77
   12.2 Data Location ................................. 77
   12.3 Platform Specific Information .................. 77
   12.4 Code Distribution, Installation and Support ...................... 77
A. Contents of INMS Analysis Library ........................ 79
B. Release Notes .................................... 84
C. Depreciated Routines ............................. 85
Figures

Figure 1, Example File Selection Dialog ........12
Figure 2, Example inms_query_l1a output .......15
Figure 3, Example inms_tabulate_data
   Output ........................................19
Figure 4, Example Plot Mass Time Spectra .......21
Figure 5, Example Multi-panel Mass-Time
   Spectra Plot ....................................22
Figure 6, Example Mass History ....................24
Figure 7, Example Time Series Plot ...............25
Figure 8, Example Instrument State Plot ...........27
Figure 9, Example Trajectory Geometry
   Plot ........................................28
Figure 10, Example Histogram Plot ...............30
Figure 11, Example Stacked Spectra Plot ...........31
Figure 12, Example Mass History Plot .............33
Figure 13, Example Mass Profile Plot .............34
Figure 14, Example Comparison Plot ...............36
Figure 15, Example Housekeeping Data
   Plot ........................................37
Figure 16, Example inms_list_cal_species
   Output ........................................40
Figure 17, Example Cracking Pattern Plot ......42
Figure 18, Example Sensitivity Plot ...............43
Figure 19, Example Graphical Output from
   inms_deconvolve ..............................56
Figure 20, Example Density Profile Plot ........58
Figure 21, Example Ion Density Spectra .........59
Figure 22, Color Tables ...........................63
Figure 23, Available Colors by Name ............65
Figure 24, Example inms_neat_ticks Usage .......66
Figure 25, Label Structure Example ...............69
Figure 26, Example inms_post_message
   Usage ........................................72
Figure 27, Example furnish Text Kernel ........75
Figure 28, Example inms_get_spice_kernels
   Output ........................................77

Tables

Table 1, Naming Convention for IDL
   Variables ......................................10
Table 2, Calibration Structure Contents ..........39
Table 3, Density Profile Structure Contents ....57
Table 4, PDS Label Template Tokens ..........73
Table 5, Contents of INMS Analysis Library ....79
Table 6, Auxiliary Files Included With INMS
   Analysis Library ...............................83
Alphabetical List of Procedures and Functions

inms_auxiliary_value .................................................. 46
inms_chebyshev .......................................................... 52
inms_close_windows ..................................................... 62
inms_compute_mean_spectra ........................................... 47
inms_compute_summed_spectra ........................................ 48
inms_deconvolve .......................................................... 54
inms_doy2date ............................................................. 50
inms_doy2julian ........................................................... 51
inms_doy2utc ............................................................... 50
inms_dump_structure ..................................................... 72
inms_format_time .......................................................... 50
inms_get_data .............................................................. 11
inms_get_series ........................................................... 16
inms_get_spectra ........................................................... 16
inms_get_spice_kernels .................................................. 76
inms_grid_spectra ........................................................... 17
inms/ion_sensitivity ....................................................... 44
inms/ion_transmission ..................................................... 44
inms/is_image ............................................................... 60
inms/is_publication ........................................................ 60
inms/kernel_list ............................................................ 75
inms/label_ticks ............................................................. 67
inms/list_cal_species ..................................................... 40
inms_make_data_structure .............................................. 70
inms_make/ion_spectra .................................................... 58
inms_make_pds_label ..................................................... 73
inms_make_profiles ....................................................... 56
inms_make_window .......................................................... 62
inms_neat_ticks ............................................................. 66
inms_plot_cal_prt ............................................................ 41
inms_plot_cal_sens ........................................................ 42
inms_plot_compare .......................................................... 34
inms_plot_density_profiles ............................................ 57
inms_plot_geom ............................................................. 27
inms_plot_histogram ....................................................... 29
inms_plot_hkg ............................................................... 36
inms_plot_mass_history .................................................. 32
inms_plot_mass_profile .................................................. 33
inms_plot_mt_line ........................................................... 22
inms_plot_mt_spectra ...................................................... 20
inms_plot_series ........................................................... 24
inms_plot_stacked_spectra .............................................. 30
inms_plot_state ............................................................ 25
inms_post_message ........................................................ 71
inms_post_status ........................................................... 74
inms_prepare_plot ........................................................ 59
inms_query_LIa ............................................................. 15
inms_ram_angle ............................................................. 46
inms_ram_coefficient ..................................................... 47
inms_read_cal .............................................................. 39
inms_read_cal/ion ............................................................ 45
inms_read_fnt_file ........................................................ 70
inms_read_jcamp ............................................................ 41
inms_read_label_file ..................................................... 69
inms_read_pds_sheets ..................................................... 13
inms_read_scvolts .......................................................... 45
inms_restore_data ........................................................... 71
inms_saturn_latitude ..................................................... 47
inms_saturn_longitude .................................................... 47
inms_save_data ........................................................... 70
inms_select_cal ............................................................. 40
inms_spectra_calculations .............................................. 49
inms/status_widget ........................................................ 74
inms_subtract_background .............................................. 49
inms_svd.solve ............................................................ 52
inms_tabulate_spectra .................................................... 18
inms_test ................................................................. 78
inms_utc_increment ........................................................ 51
inms_utc2date ............................................................... 50
inms_validate_cal_data ................................................... 45
inms_validate_hkg_data .................................................. 45
inms_validate_LIa_data .................................................... 45
inms_validate_spectra_data ............................................. 45
inms_validate_time ........................................................ 45
inms_weighted_mean ....................................................... 52
inms_write_image ........................................................ 61
sprl_colorplot ............................................................. 67
sprl_color_tria ............................................................ 64
sprl_cvt_jdate_md ...........
1. References

   SPRL File 089-0024, 26 January 2005

2. Gell, D.A. “INMS Open Source Ion Sensitivity Model”

2. Introduction

The INMS Archive Manager produces Level 1A (L1A) files, (Reference 1) and engineering housekeeping (HKG) files. These files are in the PDS compliant comma-separated value spreadsheet format. Each row of the L1A spreadsheet corresponds to one integration period. Data items in the row consist of ancillary data, geometric data, instrument configuration data and detector outputs. Each row of the housekeeping file corresponds to one HK packet.

This document describes a library of routines written in the IDL language for reading and manipulating these files. The routines form a toolkit, not an application. The user may develop applications using the routines, or may investigate the data interactively, by invoking routines from the IDL command line. The routines include those to read the data, extract subsets of the data based on supplied selection criteria, to aggregate data into mass spectra, manipulate and display the data. Routines are also supplied for the reading and display of calibration data. A facility for basic mass deconvolution is also included.

This document is intended for users of the INMS level 1A data archive and engineering housekeeping files. Additional help is available in the file inms_analysis_help.html which may be viewed with any web browser.

2.1 Conventions

In this document, commands written in the normal courier font are entered exactly as shown. Tokens that you replace with an appropriate value are shown in the italic courier font. Prompts that the programs present to you are shown in the bold courier font. In the body of the text, function and procedure names are shown in the italic font, thusly: inms_get_data. Optional portions of commands are enclosed in braces {like this}. Alternatives are separated by a vertical bar, /like this/. The routines written for this library comply with the INMS coding standard for IDL. In particular, a modified “Hungarian” notation is used for variable names. In this scheme, a variable name is preceded by a prefix indicating the nominal type of the variable. The most common prefixes used are listed in Table 1, below. Note that IDL is a weakly typed language, in that variable types may be changed dynamically. Because of this, some variables will occasionally be of a type that does not agree with their name prefix. This usually happens as part of exception handling. All procedure and function names are preceded with a project prefix to insure unique names. The prefixes are inms_ for routines written explicitly for this package and sprl_ for utility routines developed in other projects and ported to this package.
**Table 1, Naming Convention for IDL Variables**

<table>
<thead>
<tr>
<th>prefix</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>numerical, no distinction is made between integer, real and complex types</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>x</td>
<td>structure</td>
</tr>
<tr>
<td>a</td>
<td>array, precedes any of the above prefixes to indicate an array of items</td>
</tr>
<tr>
<td>p</td>
<td>pointer, precedes any of the above prefixes to indicate a pointer to a variable of that type</td>
</tr>
</tbody>
</table>

IDL is case insensitive, however to improve readability variable names are mixed case, with the type prefix lower case, and words within the variable name capitalized. Variable names are generally formed without underscores in the name, except for fields of the L1A data structure. Function and procedure names are lower case and, on platforms whose file naming is case sensitive, must be lower case.

### 2.2 Prerequisites

The routines described in this memo require IDL version 6.0 or latter to operate. An X window server is required for the production of image files. Postscript files can be produced without any additional software beyond that included with IDL and this package.

The PDS structure file that describes the columns of the data file must be included in the directory containing the data file, or its parent. If the structure file is absent defaults, which are included with the library distribution, are used. The data file labels are also required. The labels are files that share the same base name as the data file, but have the file name extension of LBL, replacing the CSV extension that identifies the data files. The label files are included in the data delivery packages obtained from the INMS Operations Network web site.

### 3. Data Structures

The INMS analysis library uses IDL structures to organize and pass data from one routine to another. In IDL a structure is an aggregate of fields. Each field has name, called a tag name. A field may be of any type, numerical or string. It may be an array or a nested structure. Arrays of structures are permitted. A structure field is referenced by the structure and tag names, with a dot ‘.’ separating the parts, for example

```idl
xSpect.anC1counts[3]
axL1Adata[1325].mass
```

In the first example an element of the array anC1counts in the xSpect structure is referenced. In the second, the mass field of an element of the axL1Adata array is referenced.
The analysis library makes frequent use of a number of structures, a level 1A (L1A) data structure, a spectra structure, calibration summary data structure and housekeeping packet structure. The L1A data structure contains a field for each column in the Level 1A data file. The tag names are identical to the PDS field names, and are therefore an exception to the naming convention. Reference 1 lists the items in the file. Items in the file include ancillary data, instrument configuration data, geometric data and counts. Geometric data item names are formed by appending “.t” or “.s” to a descriptive name to indicate whether the item is determined with respect to a target body or Saturn, respectively.

The spectra structure contains a spectra formed by aggregating data from one or more mass tables. Fields in the structure contain arrays of masses, counts, standard deviations and ancillary data. The calibration structure contains data describing the instrument sensitivity and cracking pattern for a species. Each of these structures may be elements of arrays of structures.

The housekeeping data structure contains one field for each item in the engineering housekeeping TM packets. Additional fields provide packet identification information.

4. Reading Data

4.1 Reading L1A, Housekeeping and Trend Files

The INMS Analysis library provides the procedure `inms_get_data`, to read data from the PDS compliant spreadsheet files produced by the INMS data system. This procedure will read one or more files based on the parameters passed to it and forms an array of data structures. A call to this procedure produces one data structure array. Files may be specified by name, time range or by use of a file selection dialog. In addition to the files being read, the `inms_get_data` procedure requires the presence of the associated PDS label file and the structure file which describes the columns of the spreadsheet. Both the label files and the structure file are distributed with each daily data set. The structure file, named L1A_STRUCT_vv.FMT for level 1A data or HKG_STRUCT_vv.FMT for housekeeping data, can be located in either the same directory as the data files or its parent. Default versions are included in the analysis library source code directory. Trend data is produced by ION, on demand, and consists of engineering data for extended periods of time.

You can specify which data you want to read by file name, time range or via a file selection dialog. To open and read a file, you use the `inms_get_data` command:

```
inms_get_data, axData {,type=‘L1A’|‘HKG’|‘TRND’} {,path=pathToFiles} {,files=[listOfFiles] | trange=[‘startTime’,’endTime’]} {,/yeardir | /doydir} {,/debug}
```

where you replace the token `axData` with the name of the variable to contain the data structure array. You specify whether to read L1A data, HKG data or TRND by supplying a value for the `type` keyword. If the `type` keyword is absent, L1A data is selected by default.

You can supply either a list of files or a time range using the `files` or `trange` keywords respectively. If you specify more than one file, the list should be in the form of a string vector. If neither the `files` nor the `trange` keywords are present, a file selection dialog is provided. Times are specified in the year, day-of-year format. For example, midnight UTC on 1 June 2005 is represented as 2005-152T00:00:00. Trend data files must be specified by file name or by use of the file selection dialog.

Data files are assumed to be in the current default directory unless you supply an alternative with the `path` keyword. The organization of the data directory is specified by the `/yeardir` or `/doydir` keywords. In the first case a subdirectory exists for each year, and in the second a
subdirectory exists of each day of the year in the year subdirectories. If neither keyword is present, all files are assumed to be in one directory. The /debug keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

If the structure file, which specifies the organization of the data file, is not present in any of its default locations a file selection dialog will appear so that you may locate it.

The simplest use of the command is to supply only a data variable, as shown below

\texttt{inms\_get\_data, axData}

The token “\texttt{axData}” may be any valid IDL variable name. A file selection dialog similar to that shown below appears. Only files of the type specified (L1A or HKG) will appear in the dialog.

![Example File Selection Dialog](image)

**Figure 1, Example File Selection Dialog**

You navigate through the directory structure by either entering a directory name in the text box or by clicking on items in the directories list box. Once you’ve specified the appropriate directory, you can select one or more files. Clicking the OK box initiates the data input. You can specify an initial path in the command, as follows,

\texttt{inms\_get\_data, axData, path='/the/path/to/data'}

in which case, the default directory in the dialog will be the one specified by the path keyword. If no path is supplied, the current working directory is the default.

In order to select data by time or file name, you supply values for the \texttt{trange} or the \texttt{files} keyword. Time ranges are specified in the year, day-of-year format, for example, noon UTC on June 1, 2005 would be specified as 2005-152T12:00:00. The hyphen delimiter separates the year from the day of year, and the “T” delimiter separates the date from the time. All fields are required. To read data for a period of one-half hour centered at the Titan A encounter at 2004-300T15:30:00, you would invoke the following command:
inms_get_data, axData, $
\text{trange}=[\text{‘2004-300T15:15:00’,} ‘2004-300T15:45:00’]\]

As specified above, the data files are expected to be in the current working directory. Alternative directories may be supplied by use of the path keyword. Also, the data directory can be organized in one of three ways, flat, by year, or by year and day-of-year. In the first case, all data files are in one directory, specified by the path keyword. When organized by the year, the path keyword specifies a parent directory containing a sub-directory for each year with data. The subdirectory names are the year, e.g. 2004, 2005. When organized by year and day of year, the path keyword specifies a parent directory containing year directories as in the organization by year scheme. Each year subdirectory has additional day-of-year subdirectories that contain the data files. The day-of-year subdirectory names are the day number, 001 for 1 January through 365 (or 366) for 31 December. This is the organization of data files in PDS archive volumes.

You specify the data organization by supplying either the /yeardir or /doydir keywords. If neither is supplied, the file organization is flat. If the /yeardir keyword is supplied, the files are organized by year, and if the /doydir keyword is supplied the files are organized by year and day. For example, if data directory is ~/inmsData and the files are organized by year and day of year, the example above becomes

inms_get_data, axData, /doydir, $ 
\text{path}='~/\text{inmsData}',$ 
\text{trange}=[\text{‘2004-300T15:15:00’,} ‘2004-300T15:45:00’]\]

Files may also be specified by name. In this case you supply a file name or list of file names via the files keyword. The path keyword is used to specify an alternate file location to the current working directory. For example, to read two files from the current directory, you would type

inms_get_data, axData, $ 
\text{files}=[\text{‘file1.csv’,} \text{file2.csv’}] \]

The structure axData contains, in this case all of the data from file1 with all of the file2 data appended to it. The time ranges may span one or more data files.

### 4.2 Reading Generic PDS Spreadsheet Files

The INMS L1A, Housekeeping and Trend data files are PDS spreadsheets. The routine described above, inms_get_data, is an example of a routine which reads these spreadsheet files. That routine is customized to find data within the INMS archive structure for particular dates. PDS spreadsheet files are useful for storing other data, such as calibration or ancillary data. You can use the routine inms_read_pds_spreadsheet to read any comma separated value file which is described by a detached PDS label.

To read a spreadsheet file, you type a command of the form

inms_read_pds_spreadsheet, axData {, asFiles}$ 
\{,\text{title=‘File Selection Dialog title’} \} {,/debug}$

The token axData is replaced with the name of a variable to contain the spreadsheet structure. The asFiles argument provides the paths of the files to be read. If more than one file is to be read, this argument is a vector of strings, each string being the path to one file. If the asFiles argument is absent, a file selection dialog is presented. You can supply a title for the file selection dialog using the title keyword expression. The detached label is assumed to have the same name as the data file, with the file name extension, .CSV replaced with .LBL and located in the same directory as the data file. If the label cannot be found in that location, a file selection dialog is presented so that the label file may be specified.
The `/debug` keyword controls the behavior of the procedure when an error occurs, and is not usually required.

The data returned by the procedure consists of an array of structures, with one array element for each record in the spreadsheet. The organization of the structure is specified by the PDS label. Each structure field is given the name specified in the label by the NAME keyword. The IDL data type assigned to the field is determined by the value of the DATA_TYPE keyword. Fields for which the ITEMS keyword is greater than 1 are vectors, with the number of elements specified by the value of the keyword.

5. **Selecting Data and Forming Mass Spectra**

The `inms_get_data` procedure described above returns a structure array containing all of the data within the specified files or time range. To further refine the data selection, the library contains three procedures. The first, `inms_get_series`, extracts a series of data records from the L1A data structure and forms a second L1A data structure from those records. For more complex data selection criteria, native IDL constructs, such as expressions using the `where` function may be used.

The other two data extraction procedures, `inms_get_spectra` and `inms_grid_spectra` form one or more spectra structures from the L1A data. The routine `inms_get_spectra` aggregates a series of data records from a L1A data structure to form an array of one or more spectra structures. The routine `inms_grid_spectra` interpolates the signals in each mass channel to a uniform time grid. All three routines use the same syntax for specifying selection criteria. The syntax permits any data item in the L1A record to be used for selection.

5.1 **Selection Criteria Specification**

The procedures `inms_get_series`, `inms_get_spectra` and `inms_grid_spectra` accept the same syntax for specifying selection criteria. Criteria based on the data are specified as a list of keyword value pairs. The keyword names are the names of the fields in the L1A structure. The value may be a scalar or a one-dimensional array. If the value is a scalar, the selection criterion is met for all records with the specified field equal to the specified value. For example, specifying the following keyword expression

```
target="titan"
```

selects all data records whose target field contains “titan”. String comparisons are case insensitive. Another example,

```
cyc_table=10
```

selects all data records containing data obtained while cycle table 10 was being executed.

To select records within a range of a value, you specify a two-element vector as the keyword value. The selection criterion is met for all records with the specified field greater than or equal to the first element of the vector and less than or equal to the second. For example, to select all data between 1200 and 1400 km altitude above the target body, you would specify

```
alt_t=[1200.,1400.]
```

To select records with values equal to one of a list, you specify a three or more element vector as the keyword value. The selection criterion is met for all records with the specified field equal to one of the elements in the supplied vector. For example, to select all records for data collected using mass table 12,13, or 14, you would specify
mass_table=[12, 13, 14]

If you wanted data for mass tables 1 and 15, specifying

mass_table=[1, 15]

would select all mass tables from 1 to 15, inclusive. To obtain only tables 1 and 15, one of the table numbers must be repeated in the keyword expression, as follows:

mass_table=[1, 1, 15]

If you supply more than one keyword expression, the selection criteria is met for each data record satisfying all of the expressions, in other words the selection criteria is the logical product (and) of all of the criteria. Since the selection criteria are supplied as IDL keyword arguments, all of the IDL restrictions on keyword arguments apply. In particular, keywords that partially match each other, like mass and mass_table, are considered ambiguous. When this happens, an error is reported. You resolve this by adorning one, but not both, of the variable names with a leading underscore. For example, if you must select on both mass and mass table number, you could supply the following keyword expressions to avoid the ambiguity:

_mass=16, mass_table=[12, 13, 14]

You can query the L1A data structure with the `inms_query_l1a` procedure to obtain a list of tables and other information. You invoke the procedure as follows:

```idl```
inms_query_l1a, axL1A```

You replace the token `axL1A` with the name of an L1A data structure. The routine produces a table of the targets, table set IDs, coadd counts, velocity compensation values and the sequence, cycle and mass tables. The output from this procedure is similar to that shown in Figure 2.

```
TARGET: "titan"
TABLE_SET_ID: "168-2"
COADD_CNT: 1
CYC_TABLE: 4, 6, 7, 9, 10
SEQ_TABLE: 8, 12, 60
MASS_TABLE: 2, 3, 4, 5, 6
SOURCE: "csn", "osi"
VELOCITY_COMP: 0.00, 5.88, 6.00
```

**Figure 2, Example inms_query_l1a output**

You can obtain the names of each of the fields in an L1A, housekeeping, calibration, or any other structure using the IDL `help,/str` command. For example, to obtain a list of all of the field names in an L1A data structure called `axData`, you would use the following statement:

```
help,/structure, axData
```

The list of fields in the structure is written to the standard output device.
5.2 Data Selection

5.2.1 Extracting Data Series (inms\_get\_series)

The procedure inms\_get\_series creates one level 1 data structure from another. The command syntax is

\[
inms\_get\_series, axL1A, axSeries, selector\_list \\
{,/ingress | /egress}
\]

where you supply the name of a L1A data structure to replace the token axL1A and the name of a variable to contain the subset for axSeries. The selector list is a list of one or more data selection keyword expressions as defined in 5.1 above. The keywords /ingress or /egress limits data to the inbound or outbound portion of the encounter respectively. For example, to select all data from altitudes between 1000 and 1500 km, you would use the following command:

\[
inms\_get\_series, axL1A, axSeries, alt_t=[1000, 1500]
\]

5.2.2 Native IDL facilities

To select data based on criteria more complex than those provided by the inms\_get\_series procedure, one can use the IDL where function with appropriate logical statements.

5.3 Mass Spectra Formation

5.3.1 Accumulating Spectra (inms\_get\_spectra)

The procedure inms\_get\_spectra creates a spectra structure from L1A data structure. A spectrum in this context is the set of all masses for a particular source sampled by one instance of a set of mass tables. Each instance of the first mass in the first table of the table set begins a new mass spectrum. The list of mass tables is supplied as one of the arguments of the procedure. The user can specify additional selection criteria using the syntax described above.

The command syntax is

\[
inms\_get\_spectra, axL1A, axSpectra, MassTableID=[mtid1, mtid2, ..., mtidn], \\
{,CoAddCnt=n}, source='osnb'|'osnt'|'osi'|'csn'}, /hires, \\
{,selector\_list}, {,/ingress | /egress}, {,/debug}
\]

where you supply the name of a L1A data structure to replace the token axL1A and the name of a variable to contain the array of spectra for axSpectra. The mass table numbers are specified as the value of the MassTableID, shown in the example as mtid1, mtid2, etc. If there is only one id specified, it may be specified as a scalar. The source of interest is specified as the value of the source keyword, supplied as a string. If the source keyword is absent, the default value is ‘osi’. To process high-resolution mass scans include the /hires keyword. Since measurements with unequal co-add counts may not be combined, you may select a value using the CoAddCnt keyword. If not supplied, the default value is one. To limit data to the inbound or outbound leg of a trajectory you can specify the /ingress or /egress keyword.

For example, the following command extracts closed-source, neutral spectra constructed using mass tables 16 and 17 for altitudes above the target between 1100 and 1230 km during the inbound portion of the encounter:

\[
inms\_get\_spectra, axData, axSpectra, source='csn', \\
masstableid=[16, 17], alt_t=[1100, 1230], /ingress
\]
The accumulated spectra contain the count rate in counts per integration period. Since a mass may be visited more than once, the rate is computed by accumulating the counts in each mass bin and counting the number of times each mass is sampled within the mass scan. Letting \( n \) be the co-add count and \( N \) be the number of visits to a mass, the rate is

\[
R = \frac{1}{nN} \sum_{k=1}^{N} S_k
\]

and the standard deviation for the rate is

\[
\sigma R = \frac{1}{nN} \sqrt{\sum_{k=1}^{N} S_k}
\]

5.3.2 **Gridding Spectra (inms\_grid\_spectra)**

The procedure `inms\_grid\_spectra` also creates a spectra structure from L1A data structure. It differs from `inms\_get\_spectra` in the way that the spectra are formed. This routine interpolates the signals in each mass channel to a uniform time grid then collects all the mass channel signals at a particular time point into a spectrum. The interpolation is performed in two steps. First, segments of the data for each mass channel are fit to a Chebyshev polynomial then the polynomial is evaluated at the time grid point. As in `inms\_get\_spectra`, only points for a specified set of mass tables, a specified ion source, and a specified co-add count are included in the data from which the spectra are formed. The user may also specify additional selection criteria using the syntax described in Section, 5.1 above.

The command syntax is

```plaintext
inms\_grid\_spectra, axL1A, axSpectra, $
   MassTableID=[mtid1,mtid2,...mtidn] \{,CoAddCnt=n\} $
   \{,source='osnb'|'osnt'|'osi'|'csn'\} $
   \{,stride=n\} \{,span=n\} \{,order=n\} \{,exclimit=nsig\} $
   \{,diagnostic=axDiagInfo\} \{,/verbose\} \{,$
   \{,selector\_list\} \{,/ingress | /egress \} \{,/debug\}
```

where you supply the name of a L1A data structure to replace the token `axL1A` and the name of a variable to contain the subset for `axSpectra`. The mass table numbers are specified as the value of the `MassTableID`, shown in the example as `mtid1, mtid2, etc`. If there is only one id specified, it may be specified as a scalar. The source of interest is specified as the value of the `source` keyword, supplied as a string. If the `source` keyword is absent, the default value is ‘csn’.

Since measurements with unequal co-add counts may not be combined, you may select a value using the `CoAddCnt` keyword. If not supplied, the default value is one. To limit data to the inbound or outbound leg of a trajectory you can specify the `/ingress` or `/egress` keyword.

The keywords `stride`, `span`, `order` and `exclimit` control the interpolation. The `stride` keyword specifies the spacing between points in the time grid and defaults to 5 seconds. The `span` keyword sets the length of a data segment used to determine the value at each time point. It is defaults to six times the span. The `order` keyword is used to set the degree of the polynomial to which the data is fit and defaults to 3. Numerical experiments indicate that a ratio of span to stride of 6 yields good results. Outlying points may be excluded from the fit by supplying a value to the `exclimit` keyword. The value specifies the deviation between a data point and the fit at that time beyond which the data point should be excluded from the fit, specified in standard deviations. The outlier exclusion proceeds iteratively, removing only the data point whose deviation is greatest and repeating the fit.
The keyword **diagnostic** is used to supply the name of a variable into which diagnostic information for each fit will be placed. The diagnostic information is provided in the form of an array of structures. Each structure contains the time, the counter number, mass channel, background, number of points included in the fit, the result of the fit, the reduced $\chi^2$ and number of singular values encountered in the solution of the normal equations for the fit. If the `/verbose` keyword is set, this information is displayed as the program runs.

For example, the following command forms the closed-source neutral spectra collected using mass tables 16 and 17 for altitudes above the target between 1100 and 1230 km with the default interpolation parameters:

```plaintext
inms_grid_spectra, axData, axSpectra,
masstableid=[16, 17], $\$
alt_t=[1100, 1230]$
```

### 5.3.3 Examining Spectral Data (**inms_tabulate_spectra**)  

Once a spectra is formed, you can examine the data using **inms_tabulate_spectra**. This routine can produce a formatted table of the ancillary data included in the spectra and optionally values of the signal at selected mass per charge ratios.

The syntax is

```plaintext
inms_tabulate_spectra, xSpectra, {anMasses},
{, file=sFileName} {, title=stitle}, {/debug}
```

You replace the token `xSpectra` with the name of a variable containing a spectra to examine. The argument `anMasses` is a vector of 1 or more mass values. If present, the values of the signal in counter 1 for those masses will be included in the formatted table. The `file` keyword expression specifies the name of a file to contain the output. If absent, the output is written to the standard output device, usually the terminal window. The `title` keyword is used to provide an optional title string used as a header in the output. An example output is shown in
Example Spectra Data

STYPE: Data, Mean of 48
SFIRSTDOYTIME: 2006-282T17:25:51.222
SFINALDOYTIME: 2006-282T17:34:17.967
STARGET: titan
NUTTIME: 63005744
NLAT_T: 60.8611
NWLon_T: 146.483
NSZA_T: 81.0147
NLST_T: 4.40155
NALT_T: 979.514
NSPEED: 5.96272
NANGLE: 0.0296765
NVELX: -5.96302
NVELY: -0.00100000
NVELZ: -0.00833281
NINTPERIOD: 0.0310000
NCOADDCNT: 0
SSOURCE: csn
NSEQTABLE: 8
NCYCTABLE: 56
Mass Tables: 42, 43

Figure 3, Example inms_tabulate_data Output

6. Data Display

The INMS analysis package provides a number of display options. These options consist of L1A data summary plots, spectral data plots, and housekeeping data plots.

L1A summary plots are produced by the inms_plot_mt_spectra, inms_plot_mt_line, inms_plot_series and inms_plot_state procedures. The first plots a mass-time spectra with the magnitude of the signal displayed as a color as a function of mass in the vertical axis and time on the horizontal. The second displays signal histories for selected masses for each source. The routine inms_plot_series plots the time series of individual items from the Level 1A. Items that have discrete values are plotted as a color bar by this routine. The operational state of the instrument is displayed the inms_plot_state routine which shows the transitions between sequence tables and between mass or cycle tables. The trajectory and pointing geometry is displayed with the inms_plot_geom procedure.

Spectra data is displayed by the routines inms_plot_histogram, inms_plot_stacked_spectra, inms_plot_mass_history, inms_plot_mass_profile and inms_plot_compare. The histogram plots produced by inms_plot_histogram display individual spectra, showing signal level as a function of mass bin. The routine inms_plot_stacked_spectra produces a plot similar to the mass-time spectra summary plot showing signal level as a function of mass channel and time. The routine inms_plot_mass_history plots the time series of one or more mass channels from an array of spectra structures and inms_plot_mass_profiles displays altitude profiles of the data. The routine inms_plot_compare is used to display the times series of one or more mass channels from both the L1A and spectra structure arrays.

Housekeeping data is displayed by inms_plot_hkg. This routine plots time series data from the housekeeping data files. Data items that have discrete values are plotted as color bars.
In addition to the plotting routines described above, the native IDL graphics routines and the low level graphics routines included in this library may be used to display data as the user wishes.

6.1 L1A Data Plots

6.1.1 Mass-time Spectra (inms_plot_mt_spectra)

To produce a mass time spectra, you use the `inms_plot_mt_spectra` procedure, which produces a plot similar to the example in Figure 4. The plot consists of the mass-time spectra and color scale in the center and annotative data above and below. Below the title, the points at which sequence tables switch are indicated by the numbers in the circular flags. The number is the that of the sequence table that is started. The color bar below shows the ion source selection as a function of time. The color of each vertical bar indicates the selected source during the period. Below the plot, auxiliary axis show the position of the spacecraft with respect to the target body or Saturn.

The command syntax is:

```idl
inms_plot_mt_spectra, axL1A [xrange=[time1, time2]]
{source=([osnb] [csn] [osi] [osnt])
{/noaux} {/noseq} {/noion} {/tres=nn}
{subtitle='An additional title string'} {/rate}
{/c2counts} {/samewindow} {/target} {/debug}
```

The token `axL1A` is replaced by the name of a level 1A data structure. The remaining keyword arguments are optional and control the format of the plot. The time range to show on the plot is set by the `xrange` keyword. If absent, the entire span of data in the input data structure is plotted. If set, only data that falls within the time range specified is plotted. Times are entered as strings in the year, day-of-year format. In this format, noon UTC on June 1, 2005 would be specified as 2005-152T12:00:00. The hyphen delimiter separates the year from the day-of-year and the "T" delimiter separates the date from the time.

If the `source` keyword is specified and one source mnemonic is supplied, the plot displays only counts for that one source. If a list of sources is supplied, a multi-panel plot is produced, with spectra for each source in the separate panels. An example of this is shown in Figure 5. When multiple panels are selected, the ion source color bar is omitted. The keywords `/noaux`, `/noseq`, and `/noion` control the presence of the auxiliary axis, the sequence table flags, and the ion source color bar, respectively. If the keyword is present, the corresponding item is omitted from the plot.

The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the subtitle keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.
Figure 4, Example Plot Mass Time Spectra
produced by the command inms_plot_mt_spectra, axData, /target

The time resolution of the plot is controlled by the `tres` keyword. The plot is built up by summing counts that fall in time and mass bins. The mass resolution is fixed at one Dalton. The time resolution is nominally 15 seconds. The value may be changed by specifying a value for the `tres` keyword. This value should be set with care, since a small value increases the processing effort needed to smooth the data. Remembering that there are about 1000 pixels across a computer display, one should not specify a time resolution so small that there are more than about 250 time bins. The example plot, made with the default time resolution has 250 time bins.

The data may be displayed either as counts per sample period or as counts per second. Supplying the keyword `/rate` cause the count rate to be displayed, if the keyword is absent counts per sample period are displayed. You choose the counter output to display with the `/c2counts` keyword. If absent, counter 1 is displayed, if present counter 2. The keyword `/target` specifies the body to which the auxiliary axes are referenced. If absent, the auxiliary axes specify position with respect to Saturn. If present, they refer to a target moon. The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures or the color plotting procedure `sprl_colorplot`. In particular you can control the z (color) axis with the `logsw` and `zrange` keywords to `sprl_colorplot`. By default, the color scale is logarithmic. Specifying `logsw=0` disables the log scale. The range for
the color scale can be changed using the \texttt{zrange} keyword, supplying a vector with the minimum and maximum values.

![Figure 5](image)

Figure 5, Example Multi-panel Mass-Time Spectra Plot
produced by the command

\texttt{inms\_plot\_mt\_spectra, \textit{axData}, \textit{source}=[‘csn’, ‘osnb’, ‘osi’], /target, /rate}

6.1.2 Mass Histories (\texttt{inms\_plot\_mt\_line})

To produce a mass time history plots, you use the \texttt{inms\_plot\_mt\_line} procedure, which produces a plot similar to the example in Figure 6. The plot consists of panels displaying the count rate produced using one or more of the ion sources. Ancillary data includes the sequence table switching points and position with respect to Saturn or a target body. Below the title, the points at which sequence tables switch are indicated by the numbers in the circular flags. The number is that of the sequence table that is started.

The command syntax is:

\texttt{inms\_plot\_mt\_line, \textit{axL1Adata}, [\textit{m1},\textit{m2},...]}  
\{,\textit{source}=[‘osnb’]{,’csn’}{,’osi’}{,’osnt’}\} {,/rate}  
\{,\textit{xrange}=[‘time1’,‘time2’]\} {,/files} {,/noaux}  
\{,/target\} {,/c2counts} {,/errorbars}  
\{,\textit{subtitle}=‘An additional title string’\}  
\{,/samewindow\} {,/debug}  
\{\textit{keyword expressions accepted by idl plot routine}\}
The token \textit{axL1A} is replaced by the name of a level 1A data structure. The masses to include are specified in the second argument, with the tokens \textit{m1, m2, \ldots} replaced by masses to display. The time range to show on the plot is set by the \texttt{xrange} keyword. If absent, the entire span of data in the input data structure is plotted. If set, only data that falls within the time range specified is plotted. Times are entered as strings in the year, day-of-year format. In this format, noon UTC on June 1, 2005 would be specified as 2005-152T12:00:00. The hyphen delimiter separates the year from the day-of-year and the “T” delimiter separates the date from the time.

The keyword \texttt{source} specifies the ion source for which data is to be plotted. Each plot may have one to four panels displaying the data. If the keyword is absent, the closed source neutral, open source ion, and open source neutral beam are displayed. The data may be displayed either as counts per sample period or as counts per second. Supplying the keyword \texttt{/rate} cause the count rate to be displayed, if the keyword is absent counts per sample period are displayed.

The keywords \texttt{/files, /noaux, /target} and \texttt{subtitle} control the annotation of the plot. When the \texttt{/files} keyword is present, a list of files from which the displayed data was read is added to the right margin of the plot. If the \texttt{/target} keyword is present, the auxiliary axis display position with respect to the target body rather than with respect to Saturn. If \texttt{/noaux} is present, the auxiliary axes are omitted. The keyword \texttt{subtitle} allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

You choose the counter output to display with the \texttt{/c2counts} keyword. If absent, counter one is displayed, if present counter 2 is displayed. The keyword \texttt{/target} specifies the body to which the auxiliary axes are referenced. If absent, the auxiliary axes specify position with respect to Saturn. If present, they refer to a target moon. The \texttt{/samewindow} keyword inhibits the creation of a new plot window for the figure. The keyword \texttt{subtitle} allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

The \texttt{/debug} keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required. The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures.
Figure 6, Example Mass History
produced by the command
inms_plot_mt_line, axData, [2, 14, 16, 28], /ylog, yrange=[0.1, 1e6]

6.1.3 L1A time series (inms_plot_series)

The plotting command, inms_plot_series, provides for the display of any level 1A quantity as a function of time similar to the example in Figure 7. Items, such as the ion source and the table identifiers, that take one of a few discrete values are displayed as color bars. Items such as counts, velocities or positions that assume continuous values are displayed as time histories.

The command syntax is

```
inms_plot_series, axData, ['item1', 'item2', ... 'itemN'],
{,/aux} {,/target}
{,subtitle='An additional title string'},
{,/samewindow} {,/debug}
{keyword expressions accepted by idl plot routine}
```

The token axL1A is replaced by the name of an array of level 1A data structures. The data items to display are specified in the second argument, with the tokens item1, item2... replaced by names of the L1A data items to display. The keyword /aux control the annotation of the plot. The /samewindow keyword inhibits the creation of a new plot window for the figure. The keyword subtitle allows additional information to be added to the plot title. It behaves
differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure. The /debug keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures.

Figure 7, Example Time Series Plot produced by the command

```
inms_plot_series, axData, [ 'source', 'qp_lens2', 'qp_lens4', 'seq_table', 'trap_table', 'sw_table'], /aux, /target
```

6.1.4 Instrument State Plots (inms_plot_state)

The plotting routine `inms_plot_state` is used to display a representation of the instrument’s operating mode. It produces a plot that displays when sequence table and mass or cycle table transitions occur, similar to the one in Figure 8. The plot displays a time history of the mass or cycle table, as selected by the user. Along the top edge of the plot, flags indicate transitions from one sequence table to another. The main portion of the plot displays the starting time of the mass or cycle table, with auxiliary axes showing altitude, time and longitude. At the side of the plot is a legend that includes a list of all table id numbers found.
To create a state plot, the command syntax is:

```idl
inms_plot_state, axL1A {, table='mt'|'ct'|'st'}{,/target}
{,/noaux}{,/samewindow}{,/debug}
{,subtitle='An additional string to include in title'}
{,keyword expressions accepted by idl plot routine}
```

The token `axL1A` is replaced by the name of a level 1A data structure. The remaining keyword arguments are optional and control the format of the plot. The table transitions to display are selected by the `table` keyword expression. You supply either `mt`, `ct`, or `st` to select the mass table, cycle table, or sequence table respectively. If the `table` keyword is absent, the mass table transitions are displayed by default.

The keywords `/target`, `/noaux` and `subtitle` control the annotation of the plot. If the `/noaux` is present, the auxiliary axes are omitted from the plot. If the `/target` keyword is present, the plots display altitude, latitude and longitude with respect to the target body. The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

The debug keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required. The plot produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures.
Figure 8, Example Instrument State Plot
produced by the command:
inms_plot_state, axData, /target

6.1.5 Trajectory and Geometry Plot (inms_plot_geom)

The plotting command inms_plot_geom is used to display the trajectory of the spacecraft and the INMS boresight direction with respect to the target body or Saturn, similar to that of the example in Figure 9. The plot displays the sub-spacecraft latitude, west longitude, local solar time, and solar zenith angle. Also displayed are the angle between the spacecraft velocity and the INMS boresight and the spacecraft altitude.
Figure 9, Example Trajectory Geometry Plot

Created using command:

```
inms_plot_geom axData, subtitle='Closest Approach Geometry'
```

The command syntax is

```
inms_plot_geom, axData {,/sasaturn}
   {,subtitle='An additional title string'}
   {,/samewindow} {,/debug}
   {keyword expressions accepted by idl plot routine}
```

where the token `axData` is replaced with the name of an L1A data structure containing the data to plot. The keyword `SATURN` is set to display the trajectory and geometry with respect to Saturn. If the keyword is absent the data with respect to the target body is displayed.

The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.
6.2 Spectra Plots

6.2.1 Histogram Plots (*inms_plot_histogram*)

The plotting command *inms_plot_histogram* creates a plot of a single mass spectrum, similar to that of example in Figure 10. The plot consists of a plot of the signal as a function of mass. Error bars are displayed as blue bars. The number of integration periods accumulated per spectra is displayed as the red dots.

The command syntax is

```
inms_plot_histogram, xSpectra {,/hires} {,/C2counts} {,/noylog} {,/noip} {,/errorbar} {,/replace} {,scale=nnn} {,refspec=xSpecRef} {,position=anPosVec} {,/samewindow} {,/debug} {,nogrid} {,subtitle='An additional title string'}
```

where the token `xSpectra` is replaced by the name of a spectra structure variable. Note that *inms_get_spectra* may return either a scalar or an array of spectra structures. In the latter case, you must specify an element of the array, for example `axSpectra[nHist]` where `nHist` is the index, as the argument of the *inms_plot_histogram* command. The keyword *hires* is set to display spectra collected at a resolution of 0.125 AMU. If absent, the mass bins are 1 AMU in width.

The keywords `/C2counts`, `/nolog`, `/noip`, `/errSw` and `/nogrid` control the format of the plot. Setting the keyword `/C2counts` plots the output of counter 2 rather than the default counter 1 values. Setting the `/nolog` keyword results in a linear scale for the signal level rather than the default log scale. The keyword `/noip` inhibits the display of the integration period count. Specifying `/errSw` adds error bars to the histogram. Specifying `/nogrid` suppresses the display of a grid at the major tick marks.

The keywords `/replace` and `scale` control the handling roll-off of the high sensitivity counter. When the counting rate exceeds approximately 1 MHz, the counter dead time and detector recharge time result in a reduction of the apparent count rate. When the `/replace` keyword is set, the values in saturated mass channels are replaced by counter two’s count rate, scaled by the ratio of the count rates. The default ratio is 5841, which may be changed with using the `scale` keyword.

You may add a second spectrum to the plot by supplying the name of a spectra structure variable with the `refspec` keyword. This reference spectrum is plotted in a contrasting color. The reference spectrum is not corrected for high counter-1 count rates.

The `position` keyword specifies the location of the plot within the plotting region. To specify a location, you supply a 4 element position vector in the same form as accepted by the IDL plot command. The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.
6.2.2 Mass-time spectra *(inms_plot_stacked_spectra)*

You can produce mass-time spectra from spectra structures using the *inms_plot_stacked_spectra* procedure. It produces a display similar to the summary plot produced by *inms_plot_mt_spectra* as shown in Figure 11. The plot consists of a mass-time spectra and color scale in the center of the plot. Below the plot, auxiliary axes show the position of the spacecraft with respect to the target body.

The command syntax is

```idl
inms_plot_stacked_spectra, axSpectra
    ;{,subtitle='An additional title string'}
    ;{,/rate} ;{,/c2counts} ;{,/wlon} ;{,/ramangle}
    ;{,/samewindow} ;{,/debug}
    {keyword expressions accepted by sprl_colorplot}
```

The token `axSpectra` is replaced by the name of an array of spectra data structures. The remaining keyword arguments are optional and control the format of the plot.

The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you
supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

The data may be displayed either as counts per sample period or as counts per second. Supplying the keyword /rate cause the count rate to be displayed, if the keyword is absent counts per sample period are displayed. You choose the counter output to display with the /c2counts keyword. If absent, counter 1 is displayed, if present counter 2. The longitudinal position variable included in the auxiliary axes is controlled by the /wlon keyword. If it is present, the auxiliary axes show altitude, west longitude and latitude. If the /wlon keyword is absent, the local solar time, in hours, is included replacing the west longitude. If the /ramangle keyword is present, the angle between the instrument boresight and the spacecraft velocity is plotted over the spectra. An example if this may be seen in Figure 21.

The /samewindow keyword inhibits the creation of a new plot window for the figure. The /debug keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures or the color plotting procedure sprl_colorplot. In particular you can control the z (color) axis with the logsw and zrange keywords to sprl_colorplot. By default, the color scale is logarithmic. Specifying logsw=0 disables the log scale. The range for the color scale can be changed using the zrange keyword, supplying a vector with the minimum and maximum values.

Figure 11, Example Stacked Spectra Plot
produced by the command inms_plot_stacked_spectra, axsSpectra
6.2.3 Mass History Plots (\textit{inms\_plot\_mass\_history})

The plotting command, \textit{inms\_plot\_mass\_history}, provides the display of the variation of one or more spectral bins with time similar to the example in Figure 12. Auxiliary axes displaying the position with respect to the target body are also shown. Data points that outside the limits of the vertical axis are plotted along the upper or lower edge of the plot frame.

The command syntax is

\begin{verbatim}
\texttt{inms\_plot\_mass\_history, axSpectra, [m1,m2,...]}
  {,/rate} {,/C2counts} {,/wlon} {,subtitle='A string'}
  {,/errorbar} {,/samewindow} {,/debug}
\end{verbatim}

The token \textit{axSpectra} is replaced by the name of an array of spectra structures. The masses to include are specified in the second argument, with the tokens \textit{m1, m2,...} replaced by masses to display.

The data may be displayed either as counts per sample period or as counts per second. Supplying the keyword \textit{/rate} cause the count rate to be displayed, if the keyword is absent counts per sample period are displayed. To display the output of the low sensitivity counter, supply the \textit{/C2counts} keyword. You can add 1-sigma error bars to the plot by including the \textit{/errorbar} keyword. The longitudinal position variable included in the auxiliary axes is controlled by the \textit{/wlon} keyword. If it is present, the auxiliary axes show altitude, west longitude and latitude. If the \textit{/wlon} keyword is absent, the local solar time, in hours, is included replacing the west longitude.

The keyword \textit{subtitle} allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure. The \textit{/samewindow} keyword inhibits the creation of a new plot window for the figure. The \textit{/debug} keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures. In particular, you can control the dependent variable axis with the keywords \textit{yrange} and \textit{ylog}. \


6.2.4 Altitude Profiles (*inms_plot_mass_profile*)

The plotting command, *inms_plot_mass_profiles*, provides the display of the variation of one or more spectral bins with altitude similar to the example in Figure 13.

The command syntax is

```
inms_plot_mass_profile, axSpectra, [m1,m2,...]
{/rate} {,/C2counts} {,subtitle='A string'}
{/errorbar} {,/connect} {,/samewindow} {,/debug}
{keyword expressions accepted by idl plot routine}
```

The token *axSpectra* is replaced by the name of an array of spectra structures. The masses to include are specified in the second argument, with the tokens *m1*, *m2*... replaced by masses to display. The data may be displayed either as counts per sample period or as counts per second. Supplying the keyword /rate cause the count rate to be displayed, if the keyword is absent counts per sample period are displayed. To display the output of the low sensitivity counter, supply the /C2counts keyword. You can add 1-sigma error bars to the plot by including the /errorbar keyword. You can connect the marker symbols with a line by including the /connect keyword to the command.
The keyword *subtitle* allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure. The */samewindow* keyword inhibits the creation of a new plot window for the figure. The */debug* keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The plots produced by this command can be further customized by supplying keywords accepted by the IDL plot procedures.

![INMS-Cassini Mass Profile](image)

**Figure 13, Example Mass Profile Plot**
produced by the command

```
inms_plot_mass_profile, axSpectra, [2,14,16,28], yrange=[950,1350], xrange=[100,1E6], /errorbar, /xlog, subtitle='Closed Source Neutral'
```

### 6.2.5 Spectra Comparisons (*inms_plot_compare*)

The plotting command *inms_plot_compare* displays a comparison of the detector signal level in the L1A data with that of the spectra data similar to that in Figure 14. This routine is useful for evaluating the effect of various interpolation parameters supplied to the *inms_grid_spectra* routine.
The command syntax is

```idl
inms_plot_compare, axL1A, axSpectra, mass=m1
{,xrange=['time1','time2']} {,/C2counts}
{,/naux} {,/target} {,/errorbar} {,/samewindow}
{,subtitle='An additional title string'} {,/debug}
{keyword expressions accepted by idl plot routine}
```

The tokens `axL1A` and `axSpectra` are replaced with the name of arrays of L1A data and spectra data structures, respectively. The mass to be displayed is supplied via the mass keyword expression. The time range to show on the plot is set by the `xrange` keyword. If absent, the entire span of data in the input data structure is plotted. If set, only data that falls within the time range specified is plotted. Times are entered as strings in the year, day-of-year format. In this format, noon UTC on June 1, 2005 would be specified as 2005-152T12:00:00. The hyphen delimiter separates the year from the day-of-year and the "T" delimiter separates the date from the time.

The keywords `/naux`, `/target`, and `subtitle` control the annotation of the plot. If the `/target` keyword is present, the auxiliary axis display position with respect to the target body rather than with respect to Saturn. If `/naux` is present, the auxiliary axes are omitted. The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure.

To display the output of the low sensitivity counter, supply the `/C2counts` keyword. You can add 1-sigma error bars to the plot by including the `/errorbar` keyword. The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.
6.3 **Housekeeping Data Plots**

6.3.1 **Trend Plots (inms_plot_hkg)**

To produce trends of housekeeping data, you use the `inms_plot_hkg` procedure. This procedure produces a plot similar to the example in Figure 15. In these plots, items that can take on only discrete values are displayed as color bars.

The command syntax is:

```
inms_plot_hkg, axH, asItems, 
{subtitle='subtitle added to title'}
{/samewindow} {,/debug}
{keyword expressions accepted by idl plot routine}
```

The token `axH` is replaced with the name of a housekeeping data structure. A string vector containing the names of the items to plot replaces the token `asItems`.

The keyword `subtitle` allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure. The `/samewindow` keyword inhibits the creation of a new plot window for the
The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

Figure 15, Example Housekeeping Data Plot
produced by the command

```
inms_plot_hkg, axHKG, ['bfill1on', 'bfill2on', 'bfill3on', 'bfill4on', 'd1current', 'd2current', 'd3current', 'd4current'], yrange=[-0.1, 1.2]
```

6.4 Direct Plotting

In addition to the plots produced by the routines included in the INMS analysis library, you can use the structures returned by the data access and mass spectra formation routines as arguments to the basic IDL plotting and analysis routines. For example, assume that the `inms_get_data` routine has been used to read data into an array of structures called `axL1Adata`. You can plot any field against any other field in the structure by naming them. If you wanted to display the velocity components as a function of altitude, the IDL statements would be of the following form:

```
plot, axL1Adata.sc_vel_t_x, axL1Adata.alt_t, ...

oplot, axL1Adata.sc_vel_t_y, axL1Adata.alt_t, ...
```

7. **Calibration Data**

The INMS instrument operates in two fundamental modes, neutral and ion. In the neutral gas mode, ambient gas is ionized either in the open source or, after ram density enhancement, in the closed source. The ionization products are directed into the quadrupole mass analyzer where they are filtered by mass-to-charge ratio and detected by the high and low sensitivity counters. For this mode, calibration data includes the sensitivity and the relative response to the various daughter products, the cracking patterns.

The ion mode passes ambient ions through the open source into the quadrupole mass analyzer. The ambient ions are filtered by mass-to-charge ratio and detected by the counters. For the ion mode calibration data includes the sensitivity as a function of ion kinetic energy, the effects of miss tuning the quadrupole switch lens, and the angular response.

7.1 **Neutral Gas Mode**

Calibration data appropriate to the open and closed source neutral gas modes are contained in a PDS compliant file, called the calibration summary file. The inms library provides routines to read this file, select the calibration data appropriate to a particular gas and instrument state and display the calibration data.

7.1.1 **Calibration Summary File Contents**

Neutral gas calibration data consisting of instrument sensitivity and dissociative fractionation patterns for a range of species is stored in a comma-separated value data file called a calibration summary file. A calibration summary file may have data from flight model, engineering model refurbished engineering model and National Institute of Science and Technology (NIST) measurements. The procedure inms_read_cal reads this data and places it an array of structures. There is one element for each calibration in the file. The definition of the structure is in Table 2. The fields are named in accordance with the IDL coding standards. All fields are scalars except for anFracMass and anFraction, which are one-dimensional arrays of up to 40 elements.
7.1.2 Reading Calibration Data (*inms_read_cal*)

The procedure *inms_read_cal* reads the contents of a calibration data file and stores it in a calibration data structure as described above. You read this data by typing a command of the following form

```
inms_read_cal, axCal {,file="file_or_directory"} {,/debug}
```

where you replace the token `axCal` with the name of a variable to contain the calibration structure array. If you omit the `file` keyword, you will select a calibration file using the presented dialog. If you specify a directory, it will be the default directory for the selection dialog. If you specify a full file path as the keyword’s value, that file will be opened if it exists. The `/debug` keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

The desired calibration data can be selected from the calibration structure array by use of a suitably constructed `where` statement. The `sFormula` field, which contains the chemical formula of the species, should be used to select a species. For example, testing this field against ‘CH4’ would choose calibration data for methane. To specify a species containing elemental isotopes, prefix the atomic symbol with the caret (^) and mass of the isotope. For example, the isotopic species \(^{13}\text{CH}_4\) would be indicated thusly:

\(^{13}\text{CH}_4\)

You select between source, and unit by testing against the values shown in Table 2. For an example see the source code for *inms_plot_cal*.

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sUnit</td>
<td>string</td>
<td>specifies the calibration source instrument, flight(FM), engineering(EM) refurbished engineering model (REU) or NIST(NT)</td>
</tr>
<tr>
<td>sSource</td>
<td>string</td>
<td>ion source, open(OS) or closed (CS)</td>
</tr>
<tr>
<td>sGas</td>
<td>string</td>
<td>name of calibration gas</td>
</tr>
<tr>
<td>sLabel</td>
<td>string</td>
<td>IDL label string with imbedded formatting</td>
</tr>
<tr>
<td>sFormula</td>
<td>string</td>
<td>molecular formula of calibration gas</td>
</tr>
<tr>
<td>nMolWt</td>
<td>integer</td>
<td>molecular weight of calibration gas</td>
</tr>
<tr>
<td>sFilament</td>
<td>string</td>
<td>filament, Primary, Secondary</td>
</tr>
<tr>
<td>nElecEnergy</td>
<td>integer</td>
<td>electron energy</td>
</tr>
<tr>
<td>nMajorPeak</td>
<td>integer</td>
<td>mass of major peak</td>
</tr>
<tr>
<td>nSensitivity</td>
<td>real</td>
<td>sensitivity at major peak</td>
</tr>
<tr>
<td>nSigmaSens</td>
<td>real</td>
<td>standard deviation of sensitivity</td>
</tr>
<tr>
<td>nPeakCount</td>
<td>integer</td>
<td>number of fragment peaks</td>
</tr>
<tr>
<td>anFracMass</td>
<td>integer</td>
<td>mass of fragment</td>
</tr>
<tr>
<td>anFraction</td>
<td>real</td>
<td>relative sensitivity of fragment, major=1.00</td>
</tr>
</tbody>
</table>
The procedure `inms_list_cal_species` produces a list of the species contained in a calibration structure. You produce a display by typing a command of the following form:

```
inms_list_cal_species, axCal
```

where you replace the token `axCal` with the name of a variable containing a calibration data structure. The procedure produces a table on the standard output device (not on the graphics device) listing the name, formula and ion source s for each species in the structure. An example output is shown in Figure 16.

```
identifier mass species name source
C2H2 26 Acetylene CS
C2H3CN 53 Acrylonitrile CS
NH3 17 Ammonia CS
^12CH4 16 Methane 12 CS
^13CH4 17 Methane 13 CS
^14N2 28 Nitrogen 14 CS
^14N^15N 29 Nitrogen CS
^15N2 30 Nitrogen 15 CS
^40AR 39 Argon 40 CS
```

*Figure 16, Example `inms_list_cal_species` Output*

### 7.1.3 Selecting Calibration Data (`inms_select_cal`)  

The `inms_select_cal` function extracts the calibration data for one species from the array of calibration data structures returned by `inms_read_cal`. If no calibration meets the supplied criteria, a value of scalar zero is returned and an optional warning message is posted. The command syntax is:

```
xCal = inms_select_cal(axCal, species="formula"
    {,unit="fm"|"em"|"nt"|"*"} {,source="cs"|"os"}
    {,energy=nn} {,filament="pri"|"sec"}
    {,/multiple} {,/silent}
```

The token `formula` is replaced by the formula for the species of interest. The available species in the axCal structure may be obtained by invoking the `inms_list_cal_species` procedure, as described above. You further identify the calibration of interest by supplying values for the unit, source, energy and filament keywords. The unit keyword specifies the instrument unit whose calibration is to be supplied. The flight model is denoted by a value of FM, the engineering model by EM, the refurbished engineering model by REU and NIST data converted to flight model sensitivities by NT. If an asterisk is supplied or if the `unit` keyword is absent, flight model data will be returned, if present. If no flight model data is available, engineering model data will be returned. If neither flight nor engineering data is in the structure, NIST data is returned.

The source keyword specifies whether to select closed source (cs) or open source (os) data. The default is closed source. The `energy` keyword supplies the value of the electron energy in the ion source to select. The default value is 70 eV. The filament keyword specifies which ion source filament is to be selected, primary (pri) or secondary (sci). The default is the primary filament.

If the `/multiple` keyword is not present, the function returns either 0 or the first match found in the file. If it is present and a source is selected with the `source` keyword, cracking patterns for all instances of the specified species for the specified source will be returned. This is useful when
multiple calibrations for a specific species have been included in a calibration file. The /silent keyword inhibits messages indicating that the selection failed.

7.1.4 Reading NIST Mass Spectra (inms_read_jcamp)

NIST mass spectra are presented in text files that follow the jcamp-dx format. In this file, data is stored in named records. The routine `inms_read_jcamp` reads these files and returns a structure containing the data of each named record as a field whose name is the name of the record. To read such a file you invoke a command of the form:

```
inms_read_jcamp, axData {,/sFile} {,/debug}
```

You replace the token `axData` with the name of a variable to receive the data structure. The parameter `sFile` is the name of the file to read, if absent a file selection dialog is presented. The `/debug` keyword controls the behavior of the routine when an error occurs and is normally not required.

7.1.5 Plotting Cracking Patterns (inms_plot_cal_ptrn)

The fractionation patterns in the calibration data file can be displayed with the `inms_plot_cal_ptrn` procedure. It produces plots similar to Figure 17, below. The thicker black line is a linear plot of the relative response and the thinner red line is a logarithmic plot. Each panel is annotated with the species, unit, filament, electron energy and sensitivity.

To plot the cracking pattern data you type a command of the form

```
inms_plot_cal_ptrn, axCal, asSpecies
   {,/FM | ,/EM | ,/REU | ,/NIST}
   {,filament= "primary" | "secondary" }
   {,energy=vv} {,source= "OS" | "CS"}
   {,/columns} {,/multiple} {,/samewindow}
{keyword expressions accepted by idl plot routine}
```

You replace the tokens `axCal` with the name of the calibration data structure array and `asSpecies` with a string or string array containing the formula of the species to plot. If you specify `/FM`, `/EM`, `/REU` or `/NIST` only data collected using that instrument will be displayed. If none of those choices are made, data from the flight model calibration will be displayed if present, otherwise engineering model data or NIST data will be displayed if present.

The keywords `filament`, `energy` and `source` are used to choose the calibration configuration for which data is to be displayed. By default data for the closed ion source primary filament at 70 eV is displayed. Specifying values for any of these keywords changes the selection. If the `/multiple` keyword is specified and a source is selected with the `source` keyword, cracking patterns for all instances of the specified species for the specified source will be displayed. This is useful when multiple calibrations for a specific species have been included in a calibration file.

The `/columns` keyword controls the format of the plot. If the keyword is present, the data is displayed in two columns of eight plots per page or window. If absent, the data is displayed in one column of four plots per page or window. The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The plot may be further customized by specifying additional keyword expressions accepted by IDL plotting procedures.
7.1.6 Plotting Sensitivity Data (*inms_plot_cal_sens*)

The sensitivity of the instrument to neutral gases can be displayed using the `inms_plot_cal_sens` procedure. This procedure produces a plot similar to the one shown in Figure 18. A vertical bar indicates the sensitivity to each of the gas species selected for display. Each bar is labeled with the species formula. It also permits the comparison of sensitivities obtained from two data structures.

To plot the sensitivity data, you type a command of the form

```
inms_plot_cal_sens, axCal1 {, axCal2}
{,species=asSpecies} {,/FM | /EM | /NIST | /REU}
{,filament= "primary" | "secondary" }
{,energy=vv} {,source= "OS" | "CS"}
{,multiple} {,/nodates} {,/samewindow}
{keyword expressions accepted by idl plot routine}
```

You replace the token `axCal1` with the name of a calibration data structure array containing the data to plot. The optional argument `axCal2` is the name of a second calibration data array used for comparison, sensitivities obtained from this array are shown in red. The keyword `species` supplies a string or string array containing the formula of the species to plot. If you specify `/FM`, `/EM`, `/REU` or `/NIST` only data collected using that instrument will be displayed. If none of those choices are made, data from the flight model calibration will be displayed if present, otherwise engineering model data or NIST data will be displayed if present.
The keywords `filament`, `energy` and `source` are used to choose the calibration configuration for which data is to be displayed. By default data for the closed ion source primary filament at 70 eV is displayed. Specifying values for any of these keywords changes the selection. If the `/multiple` keyword is specified and a source is selected with the `source` keyword, sensitivity values for all instances of the specified species will be displayed. This is useful for comparing sensitivities measured in a series of calibration measurements.

The `/nodates` keyword controls annotation of each sensitivity bar with the data that the calibration for that species was performed. If absent, the dates are shown in parenthesis following the species name, if present the dates are omitted. The `/samewindow` keyword inhibits the creation of a new plot window for the figure. The plot may be further customized by specifying additional keyword expressions accepted by IDL plotting procedures.

![Instrument Sensitivity Plot](image)

**Figure 18, Example Sensitivity Plot**
produced by the command
```
inms_plot_cal_sens, axCal, species=asSpecies, /nodates
```

### 7.2 Ion Mode Calibration Data

Ion calibration data consists of a formula that describes the variation of sensitivity as a function of ion kinetic energy, models of the change in sensitivity due to mistuning of the quadrupole switching lens and the angular response of the instrument. Since the quantity of calibration data required for these calculations is modest, no calibration data file is required. Two routines are provided to perform these calculations, `inms_ion_sensitivity` and `inms_ion_transmission`. The first performs the sensitivity calculation accounting for switching lens mistuning and the second performs the angular response modeling.
The full sensitivity calculation, accounting for particle kinetic energy, quadrupole switching lens tuning and instrument pointing is performed by multiplying together the results obtained from these two functions.

7.2.1 On-axis Sensitivity (inms_ion_sensitivity)

The function inms_ion_sensitivity computes the sensitivity of the INMS instrument as a function of mass, velocity, and the date of the measurement. The mass and velocity are used to determine the ion kinetic energy while the date of the measurement is used to select the parameters required to model the switching lens tuning. The command syntax is

\[
\text{nSens} = \text{inms}_\text{ion}_\text{sensitivity}\left(\text{nMass},\ \text{nSpeed}\right)\ \text{, sDate} \\
\left\{,\text{scvolt=}\text{nSCvolt}\right\} \left\{,\text{velcomp=}\text{nCompVel}\right\} \left\{,\text{calFile='filePat'}\right\} \left\{,/\text{test}\right\} \left\{,/\text{debug}\right\}
\]

where the parameter nMass is the mass of the species of interest, nSpeed is the spacecraft velocity relative to the target body, and sDate is the date of the measurement in the PDS compliant format. These three parameters may be scalars or arrays. If more than one is an array, each of the arrays must be the same shape. The mass parameter is required, while the speed defaults to 6.0 km s\(^{-1}\) and the date defaults to 2006-250T00:00:00.

The keywords scvolt, velcomp, and calfile, provide information needed to perform the computation. The scvolt keyword is used to supply a value for the spacecraft potential in volts. The velcomp keyword is used to supply the compensation velocity specified in the operations table in use while the measurements of interest were made. These values default to -0.5v and 6.0 km s\(^{-1}\), respectively. The calfile keyword supplies the path to the ion calibration data file. If absent, a file selection dialog is presented.

Setting the /test keyword causes the routine to return the sensitivity for the case of perfect switching lens tuning. The /debug keyword controls the behavior of the procedure when an error occurs and is not normally required.

7.2.2 Angular Response(inms_ion_transmission)

The function inms_ion_transmission computes the change in ion transmission due to off-nominal instrument pointing. The result is a multiplicative factor to be applied to the sensitivity, reducing its value.

The command syntax is

\[
\text{nFactor}=\text{inms}_\text{ion}_\text{transmission}\left(\text{nMass},\ \text{nXvel},\ \text{nYvel},\ \text{nZvel}\right)\ \left\{,/\text{debug}\right\}
\]

where the argument nMass is the mass of the species of interest, and the arguments nXvel, nYvel, and nZvel are the three components of the spacecraft velocity with respect to the target body. The arguments can be scalars or vectors. If the argument nMass is a vector, the velocity components must be either all scalars or all vectors of the same length as the mass argument. If the argument nMass is a scalar, the velocity components must be either all scalars or all vectors of the same length.

The /debug keyword controls the behavior of the procedure when an error occurs and is not normally required.

The return value is the factor by which the sensitivity is multiplied to account for the reduction in instrument throughput due to the off-nominal pointing.
7.2.3 **Spacecraft Potential File (inms_read_scvolts)**

The electrical potential of the spacecraft with respect to its local environment can change during an encounter due to a number of factors. The change in spacecraft potential can effect the open source ion throughput. This routine reads a comma-separated value file containing date, time and spacecraft potential, returning an array of spacecraft potential structures, each structure having the following form:

```
xVolt0 = { nDate: 0L, $ ;; date YYYYddd
         nUTtime: 0L, $ ;; time of day, ms
         nVolts: 0.0 } ;; spacecraft potential volts
```

You invoke the procedure with a statement of the following form:

```
inms_read_scvolts, axVolts {,file=file} {,/debug}
```

The parameter `axVolts` is the name of a variable to receive the spacecraft potential structure. You specify the file to read either by supplying the `file` keyword expression or by using the file selection dialog displayed when the keyword expression is absent.

The `/debug` keyword controls the behavior of the procedure when an error occurs and is not normally required.

7.2.4 **Ion Mode Calibration File (inms_read_cal_ion)**

The ion mode calibration file contains the results of in-flight energy scans needed to adjust ion throughput for quad lens tuning errors.

You invoke the procedure with a statement of the following form:

```
inms_read_cal_ion, axCalIon, {,file=file} {,/debug}
```

The parameter `axCalIon` is the name of a variable to receive the data structure. You specify the file to read either by supplying the `file` keyword expression or by using the file selection dialog displayed when the keyword expression is absent. Ion calibration data files are PDS compliant spreadsheets. The contents of the file, and the organization of the structure are specified by the detached label associated with the data file.

The `/debug` keyword controls the behavior of the procedure when an error occurs and is not normally required.

8. **Manipulating Data**

The INMS analysis library contains a number of routines for manipulating data and performing calculations. These routines include routines for data verification, geometric calculation, instrument response calculations and spectral calculations. Each of these types of routines are further described in the following subsections.

8.1 **Data Validation**

The analysis library makes use of a number of data structures and data types. One should confirm that a data structure is of the expected type prior to using it. Also, some data items, such as time, can contain nonsense values that are valid IDL values, but not useable. The library includes the functions `inms_validate_cal_data`, `inms_validate_hkg_data`, `inms_validate_spectra_data` and `inms_validate_l1a_data` that confirm that their argument is a structure of the correct type. The function `inms_validate_time` confirms that its argument is a properly formatted year, day-of-year
time string. Each of these functions accepts one argument and return 1 if the argument is valid and 0 if not. For example, to confirm that a data array is a level 1A structure, you use the statement as shown below

```plaintext
if inms_validate_l1a_data(axData) eq 1 then begin
  ;; perform processing of valid data
endif else begin
  ;; axData in valid, perform recovery / error handling
endelse
```

The function `sprl_is_numeric` confirms that its string argument is a valid numeric string. A valid string is of the form

```
{+|-}n{.n}
```

where \( n \) is a string of one or more digits from 0 through 9. If the argument of the function is a string variable containing a valid numeric string, the function returns 1 otherwise it returns 0.

### 8.2 Geometry

#### 8.2.1 Geometry Computations (inms_auxiliary_value)

The function `inms_auxiliary_value` may be used to compute a number of auxiliary geometric quantities that are not included in the level 1A data. The quantities that can be computed are latitude, west longitude, boresight ram angle, and speed. The quantities may be computed with respect to a target body (e.g. Titan, Enceladus,...) or with respect to Saturn. The syntax is

```plaintext
anResult = inms_auxiliary_value(axL1A,
  (/lat | /wlon | /ram | /speed)
  {,/saturn} {,/debug}
)
```

You supply one of the following keywords to choose the quantity to be returned, `/lat` (latitude), `/wlon` (west longitude), `/ram` (boresight ram angle), `/speed` (spacecraft speed). The angles are returned in degrees and the speed is returned in km-sec\(^{-1}\). The result is a vector containing an entry for each entry in the input L1A data array.

This routine replaces `inms_ram_angle`, `inms_saturn_latitude` and `inms_saturn_wlongitude` and should be used in place of those routines.

#### 8.2.2 Ram Angle (inms_ram_angle)

The analysis library contains a function to compute the angle between the spacecraft velocity and the instrument boresight. This function accepts a L1A data structure and returns a vector containing the boresight ram angle in radians. The syntax is

```plaintext
anResult = inms_ram_angle(axL1A)
```

The result is a vector containing the ram angle corresponding to each element in the input L1A data structure array.
8.2.3 Saturn Coordinates (inms_saturn_latitude, inms_saturn_wlongitude)

The analysis library contains functions to determine the sub-spacecraft position, inms_saturn_latitude and inms_saturn_wlongitude. These functions accept a L1A data structure and return a vector containing the planetocentric latitude and west longitude respectively. The vector has one element for each data point in the argument supplied to the function. The statements have the following syntax:

\[
\text{anResult} = \text{inms_saturn_latitude}(\text{axL1A}) \\
\text{anResult} = \text{inms_saturn_wlongitude}(\text{axL1A})
\]

8.3 Instrument Response

The function inms_ram_coefficient computes the closed-source ram enhancement factor. This is the increase in density within the closed-source antechamber due to the motion of the spacecraft relative to the atmosphere. The syntax of the function is

\[
\text{anResult} = \text{inms_ram_coefficient}(\text{anSpeed}, \text{anTheta}, \text{anMass} \\
\{, \text{tambient}=N\} \{, \text{tsource}=N\})
\]

The result is an array of the same shape as the function arguments. The argument anSpeed is the speed of the ambient gas with respect to the spacecraft in km s\(^{-1}\). The argument anTheta is the angle between the ram direction and the instrument boresight in radians. The final argument anMass is the mass of the species in AMU. Each of these arguments may be scalar or an array, however if more than one of the arguments is an array, then the arrays must be of the same shape and size. The optional keyword parameters tambient and tsource are used to supply values for the temperature of the ambient gas flowing into the instrument and the temperature of the ion source, respectively. The default value of tambient is 273K and of tsource is 300K

8.4 Spectral Calculations

The analysis library provides a set of procedures to manipulate mass spectra structures. With the routines you can compute a mean spectra, co-add a set of spectra, remove a background spectra or perform arithmetic operations. The four procedures provided for these operations are described below.

8.4.1 Averaging Spectra (inms_compute_mean_spectra)

The procedure inms_compute_mean_spectra is used to compute the mean and standard deviation of a collection of spectra. To perform this operation use a command of the form

\[
\text{inms_compute_mean_spectra, axSpectra, xMean} \\
\{,/debug\}
\]

The argument axSpectra is an array of spectra structures to be averaged and the argument xMean is a spectra structure containing the result. If the input array contains only one element, that structure is returned as the mean. The signal in each mass channel is assumed to follow Poisson statistics. The mean and standard deviation are computed based on the following derivation.

The inms_get_spectra routine returns the count rate in counts per IP for each mass bin in the spectra. With A the accumulated counts in one mass bin for N visits to that bin in the spectra, the values returned by inms_get_spectra are given by
\[ R = \frac{A}{nN} \]  
\[ \sigma R = \frac{1}{nN} \sqrt{A} \]  

where \( n \) is the co-add count. The average rate of \( M \) spectra is simply

\[ R_{\text{ave}} = \frac{1}{M} \sum_{i=1}^{M} R_i \]  

(2)

This can also be expressed in terms of the accumulated counts

\[ R_{\text{ave}} = \frac{1}{nNM} \sum_{i=1}^{M} A_i \]  
\[ = \frac{1}{nNM} T \]  

(3)

where \( T \) is the total counts accumulated for \( M \) spectra in one mass bin. Using Poisson statistics, the standard deviation of \( T \) is \( T^{1/2} \) so that the standard deviation of \( R_{\text{ave}} \) is

\[ \sigma R_{\text{ave}} = \frac{1}{nNM} \sqrt{T} \]  
\[ = \frac{1}{nNM} \sqrt{nNM R_{\text{ave}}} \]  
\[ = \sqrt{\frac{R_{\text{ave}}}{nNM}} \]  
\[ = \sqrt{\frac{R_{\text{ave}}}{nNM}} \]  

(4)

The routine evaluates equation (2) and (5) to compute the average rate and its standard deviation.

The ancillary data included in the output spectra records are the values of the quantities evaluated at the midpoint time of the data.

8.4.2 Summing Spectra (\texttt{inms\_compute\_summed\_spectra})

The procedure \texttt{inms\_compute\_summed\_spectra} is used to co-add a set of spectra. To perform this operation use a command of the form

\[
\texttt{inms\_compute\_summed\_spectra, axSpectra, xResult } \{,/\text{debug}\}
\]

The argument \texttt{axSpectra} is an array of spectra structures to be summed and the argument \texttt{xResult} contains the result of the summation. If the input array contains only one element, that structure is returned as the result of the summation. The standard deviation for each summed mass channel is computed as \( \sqrt{\Sigma c} \) where \( c \) is the signal in the channel and the sum is over the \( N \).
measurements being summed. The /debug keyword controls the behavior of the procedure when an error occurs. It is not normally required.

The ancillary data included in the output spectra records are the values of the quantities evaluated at the midpoint time of the data.

8.4.3 Spectra Arithmetic (inms_spectra_calculations)

The function inms_spectra_calculations is used to apply arithmetic operations to spectra. One can add, subtract, multiply or divide one spectra by another or by a scalar. To perform spectra arithmetic, use a command of the form

\[
\text{axResult}=\text{inms\_spectra\_calculations}(\; \text{axArg1, xArg2,} \\
\quad \{,/\text{add} \mid ,/\text{subtract} \mid ,/\text{multiply} \mid ,/\text{divide} \;
\quad \{,/\text{sigma0\_M} \} \{,/\text{sigma0\_S}\})
\]

The function returns an array of spectra structures of the same size as the first argument, axArg1. The first two arguments axArg1 and xArg2, specify the operands. The first must be a spectra structure and may be an array. The second argument may be either a spectra structure or a scalar number. The keywords /add, /subtract, /multiply and /divide specify the operation to be performed. The final two keywords, /sigma0_M and /sigma0_S are set if the first or second argument, respectively, are to be treated as exact.

8.4.4 Background Removal (inms_subtract_background)

The function inms_subtract_background subtracts one spectra from one or more spectra. It is less general than inms_spectra_calculations described above and is meant for performing background removal. To remove a background spectra use a command of the form

\[
\text{axResult}=\text{inms\_subtract\_background}(\text{axSpectra, xBackground})
\]

The function returns an array of background corrected spectra of the same size as the first input argument, axSpectra. The background to be subtracted is provided as the second argument, xBackground. The standard deviations contained in the result are computed based on propagation of errors as \( \sigma_r = \sqrt{\sigma_s^2 + \sigma_b^2} \), where \( \sigma_s \) is the signal standard deviation and \( \sigma_b \) is the background standard deviation.

8.5 Time Conversions and Manipulation

The INMS level 1A data contains time information in two formats, the PDS compliant time string, and the time of day in milliseconds. A number of routines are provided to convert, format and manipulate time.

The first time format, the PDS compliant time string, has the following form,

\[
\text{yyyy-dddThh:mm:ss.fff}
\]

where the token yyyy is the year, ddd is the day of year, hh:mm:ss.fff is the time of day in hours, minutes, and seconds. The hyphen, colons, and letter T are required delimiters.

The second time format is the time of day in milliseconds. This requires a 32 bit integer to represent it. A day consists 86400000 milliseconds.

Two additional time formats are supported in this package. The first Julian Date and the second is the ordinal date. The Julian day number (JDN) is the number of days between noon GMT of -4712 January 1, Julian proleptic calendar and noon of the day of interest. The epoch represented
in the Gregorian (current civil) calendar is noon GMT of -4713 November 23. The Julian Date (JD) for a specific instant is the Julian day number for the preceding noon plus the fraction of a day since that instant. In order to reduce the magnitude of Julian Dates, it is common to form Modified Julian Dates (MJD) by subtracting 2500000.5. A day of MJD begins at midnight of the UT day.

The second additional time format is the ordinal date. In the analysis library it is represented by two integers. The first is the date formed by adding the day of year to 1000 times the year. The second is the time of day in milliseconds.

8.5.1  Time Conversions

Four IDL functions are included that perform conversions between the time formats in the L1A data files: inms_doy2date, inms_doy2utc, inms_utc2date, and inms_format_time. The function inms_doy2date converts the date portion of the PDS compliant date string to a day month and year string. For example, the command

\[
asResult = \text{inms\_doy2date}(['2004-001', '2004-031'])\]

converts the two dates to

01-Jan-2004  31-Jan-2004

The function inms_doy2utc converts the PDS date to a structure containing the ordinal date and time of day. For example the command

\[
axResult = \text{inms\_doy2utc('2004-300T15:31:29')}
\]

converts the date to the structure:

\[
\text{help,axResult,/str}
\]

** Structure <21006e0>,2 tags,length=8,data
 length=8,refs=1:
 NDATE LONG 2004300
 NMSECS LONG 55889000

The function inms_utc2date is the inverse of inms_date2utc. This routine converts the UTC time representation into PDF compliant date strings. The syntax is

\[
asResult = \text{inms\_utc2date}(axUTC)
\]

The argument axUTC is an array of UTC structures whose definition is shown above. The result is an array of strings the same shape as the input argument array containing the date strings.

The function inms_format_time, converts a time of day in milliseconds to a string with hours minutes and seconds. For example:

\[
\text{print,inms\_format\_time}(43200000L)
12:00:00.000
\]

Additional functions sprl_cvt_jdate_odate, sprl_cvt_jdate_mdy, sprl_cvt_jtime_tod, sprl_cvt_odate_jdate and inms_doy2julian are included that deal with the Julian Date. To prevent loss of significance, Julian dates should be stored in double precision floating point numbers. The first routine, sprl_cvt_jdate_odate, converts a Julian day number to an ordinal date. The command syntax is
\[ anOdate = \text{sprl\_cvt\_jdate\_odate}(anJDN \{,/MJD\}) \]

where \( anJDN \) is an array of integer Julian day numbers. The result, \( anOdate \), is an array the same size and shape as \( anJDN \) containing the dates corresponding to the Julian day number. An ordinal date is the day-of-year plus the year times 1000. The ordinal date corresponding to 1 July 1999 is 1999182. If the keyword /MJD is present, the input is assumed to be the modified Julian day number.

The function \( \text{sprl\_cvt\_jdate\_mdy} \) determines the month, day-of-month and year corresponding to midnight UTC on the specified Julian date. This function has the following syntax:

\[ \text{axMDY} = \text{sprl\_cvt\_jdate\_mdy}(anJdate \{,/MJD\}) \]

The value returned is an array of structures the same shape and size as the argument. The structure has three fields named \( nYear \), \( nMonth \), and \( nDay \) containing the values corresponding to the input Julian dates.

The function \( \text{sprl\_cvt\_jtime\_tod} \) converts the fractional portion of a Julian date to the time of day, expressed in hours minutes and seconds. The function has the following syntax:

\[ anTOD = \text{sprl\_cvt\_jtime\_tod}(anJdate) \]

The function returns an array, \( anTOD[npts,3] \), whose first dimension is the number of elements in the \( anJdate \) array. The sub-arrays \( anTOD[*,0] \) contains the hours, \( anTOD[*,1] \) the minutes and \( anTOD[*,2] \) the seconds.

The function \( \text{sprl\_cvt\_odate\_jdate} \), converts the ordinal date and time into the corresponding Julian Date. This function has the following syntax:

\[ \text{anJDate} = \text{sprl\_cvt\_odate\_jdate}(anOdate, anTimeMS \{,/MJD\}) \]

where \( anOdate \) is an array of ordinal dates and \( anTimeMS \) is an array of times-of-day in milliseconds. If the arguments are both arrays, they must be the same size and shape, however one of the arguments may be a scalar. In that case, the scalar is combined with each array element to form the output arrays. If the keyword /MJD is present, the output is modified Julian Dates.

The final data conversion function, \( \text{inms\_doy2julian} \), converts a PDS compliant date string to a Julian date. The function has the following syntax:

\[ \text{anJDate} = \text{inms\_doy2julian}(as\_Dates \{,/MJD\}) \]

where \( as\_Dates \) is an array of time strings. If the keyword /MJD is present, the function returns the modified Julian Date.

### 8.5.2 Time and Date Arithmetic

The analysis library contains one function that may be used for date arithmetic, \( \text{inms\_utc\_increment} \). This function has the following syntax:

\[ \text{axResult} = \text{inms\_utc\_increment}(ax\_UTC, anINC) \]

The argument \( ax\_UTC \) is an array of UTC time structures and \( anINC \) is an array of increments in seconds. Positive increments increase the time and negative decrease the time. The \( anINC \) argument must be either an array the same shape as the UTC time array or a scalar. When the second argument is a scalar, it is applied to all elements of the UTC time array.
An alternative method of date and time arithmetic is possible when the times are represented as Julian Dates. In this case, ordinary arithmetic operators may be applied to the JD values.

Neither method of date arithmetic considers leap seconds in their operation.

8.6 Miscellaneous

8.6.1 Computing a Weighted Mean (inms_weighted_mean)

The function inms_weighted_mean computes the average and standard deviation of a vector of measurements and the corresponding standard deviations. This function has the following syntax:

\[
an\text{Result}=\text{inms\_weighted\_mean}(\text{anValues}, \text{anSigmas})
\]

The arguments, which must be arrays of the same shape, supply the data to be averaged. The first argument \text{anValues}, is the array of values to be averaged. The second argument \text{anSigmas}, is the array of corresponding standard deviations. The result is a two-element vector whose first element is the weighed mean of the values, and whose second element is the standard deviation of the value.

8.6.2 Computing Chebyshev Polynomials (inms_chebyshev)

The function inms_chebyshev computes Chebyshev polynomials by recursion. It was written to be used as a call-back function in curve fitting procedures. This requires that a method to supply values required by the algorithm prior to executing the curve fit be provided. In particular, the polynomials are defined only on the interval \([-1,1]\), so the range of the independent variable must first be supplied so that a transformation of the independent variable may be made. To supply the range, the function is first invoked as follows:

\[
n\text{Dummy}=\text{inms\_chebyshev}(\text{range}=[\text{nXmin}, \text{nXmax}])
\]

The tokens \text{nXmin} and \text{nXmax} are replaced with the minimum and maximum values of the independent variable. The Chebyshev polynomials are computed by a second call to the function with the following syntax:

\[
an\text{Result}=\text{inms\_chebyshev}(\text{anXvalues}, nM)
\]

The token \text{anXvalues} is replaced by a vector of the independent variable values at which the polynomials are to be evaluated. The parameter \text{nM} specifies the number of polynomials required.

The function returns an array consisting of one column for each value in the \text{anXvalues} vector and \text{nM} rows. The elements in each row are the values of Chebyshev polynomials of order 0 through \text{nM}-1 evaluated for the corresponding independent variable value.

8.6.3 Singular Value Decomposition (inms_svd_solve)

The procedure inms_svd_solve solves the matrix equation

\[
b = Ax
\]

for \text{x} in a least squares sense using singular decomposition. It is the equation solving engine used in inms_deconvolution. The \text{m} row by \text{n} column matrix \text{A} is decomposed into 3 matrices, so the equation becomes

\[
b = U \Sigma W^T x
\]
The matrix $U$ is an square $m$ row column-orthogonal matrix, $W$ is a square $n$ row diagonal matrix and $V$ is an square $n$ row orthogonal matrix. Orthogonally provides that

$$U^T U = V^T V = I \quad (3)$$

Using these conditions equation (1) can be solved for $x$

$$x = VW^{-1}U^T b \quad (4)$$

The method is more fully described in *Numerical Recipes* by Press et.al.

The syntax of the equation is

```plaintext
inms_svd_solve, anX, anMatrix, anB {,anMeasureErrors} 
{,status=nStatus} {,sigma=anSigmaX} {,chisqr=anChisqr} 
{,svalues=anSvalues} {,/silent} {,/debug}
```

The first arguments, $anX$, $anMatrix$ and $anB$ are required. They are the $n$-element solution vector $x$, $m$-row by $n$-column matrix $A$, and $m$-element vector $b$ from equation 1, respectively. You can specify statistical weighting using the argument $anMeasureErrors$. If this argument is absent, equal weighting is assumed. If present, the measurements are weighted by the reciprocal values.

The remaining keyword arguments are used to retrieve diagnostic information. You specify the name of a variable as the value of each. The keyword $status$ returns the status of the solution. A negative value indicates an error, 0 or positive values indicate success. The positive value is the number of singular values encountered in the solution. The keyword $sigma$ returns a vector containing the 1-$\sigma$ uncertainties in $anX$. The value of $\chi^2$ is returned through the $chisqr$ keyword and the singular values through $svalues$.

The keyword /silent controls the output of status messages to the standard output. If absent, messages are produced, if present they are suppressed. The /debug keyword controls the behavior of the procedure when an error occurs and is not normally required.

9. **Deconvolution of Neutral Gas Mass Spectra**

9.1 **Outline of the method**

The general procedure is based on the fact that the signal in any mass channel is a linear combination of the signal at that mass per charge due to each species and dissociation product of that mass per charge. A forward model is constructed using the calibration data, sensitivity, and for the closed source, the ram enhancement factor. This model takes the form of a $M$ row by $N$ column matrix, with each column corresponding to one parent species. Obtaining the densities is then, in principle, the solution of the linear system

$$c = Kn \quad (1)$$

where $c$ is a vector of signals in $M$ mass bins, $n$ is a vector of $N$ densities and $K$ is the kernel matrix.

Once the forward model (equation 1) is defined, determining the abundances of the various species becomes an inversion problem. The equation is solved using the Singular Value Decomposition (SVD). First, to include the measurement errors in the formulation, equation 1 is multiplied by the statistical weights.
\[ \mathbf{S} \mathbf{c} = \mathbf{S} \mathbf{K} \mathbf{n} \]  

(2)

where \( \mathbf{S} \) is a diagonal matrix whose elements are the reciprocals of the measurement errors.

Defining \( \mathbf{y} = \mathbf{S} \mathbf{c} \) and \( \mathbf{A} = \mathbf{S} \mathbf{K} \) the equation that must be solved is

\[ \mathbf{y} = \mathbf{A} \mathbf{n} \]  

(3)

The matrix \( \mathbf{A} \) is, in general, rectangular rather than square, so the equation cannot be solved by mere inversion. Even if it could be, inversion might be numerically difficult if the matrix is poorly conditioned. An alternative is to solve by least squares. An equivalent and more robust method is solution by singular value decomposition (SVD). In this method, the matrix \( \mathbf{A} \) may be factored into three matrices. The densities and their standard deviations are computed by forming products of the appropriate matrix factors.

9.2 The Deconvolution Procedure

The \textit{inms\_deconvolve} procedure implements the deconvolution algorithm outlined above. The procedure operates on the contents of a spectra record and requires calibration data obtained using \textit{inms\_read\_cal}. It returns a number of items, both the results of the deconvolution and additional diagnostic information. You execute the deconvolution with a statement of the form:

\[
\text{inms\_deconvolve, axResult, xspectra, axCal}
\]

\[
\{\text{,species=asSpeciesList}\} \{\text{,plot}\} \{\text{,noannotate}\}
\]

\[
\{\text{,annotate=list}\} \{\text{,critFreq=nFreq | /critFreq}\}
\]

\[
\{\text{,c2Factor=nScaleFactor}, \text{,chisqr=nValue}\}
\]

\[
\{\text{,model=anModelSpectra}\} \{\text{,residual=anResiduals}\}
\]

\[
\{\text{,kernel=xKernel}\} \{\text{,verbose}\} \{\text{,debug}\}
\]

The first three arguments, \textit{axResult}, \textit{xSpectra}, and \textit{axCal} are required. The first, \textit{axResult} returns the density results. It is an array of structures with one element for each species’ retrieved density. Each element is defined as follows,

\[
\text{axResult} = \$
\]

\[
\{\text{sSpecies: 'xx'}, \text{,nAlt: 0.0,}\} \{\text{nDensity: 0.0,}\} \{\text{nSigma: 0.0,}\} \{\text{nMoleFcn: 0.0,}\} \{\text{nMoleSig: 0.0,}\} \{\text{nSingValue:0.0}}\} \{;\text{ the species formula}\}
\]

The argument \textit{xSpectra} contains the spectra to be deconvolved in the form returned by \textit{inms\_get\_spectra}. The final required argument, \textit{axCal}, contains the calibration data as supplied by \textit{inms\_read\_cal}. The keyword \textit{species} is used to specify a list of species to be retrieved. If absent, \(^{14}\text{N}_2\) and \(^{12}\text{CH}_4\) are retrieved. You supply a string array containing the formula of the species of interest. The procedure \textit{inms\_list\_cal\_species} (section 7.1.2 above) produces a list (Figure 16 is an example) from which you may select the species.

The keyword \textit{critFreq} is used to specify the frequency in Mhz in counter 1 above which it is replaced by the scaled counter 2 values. If the keyword is absent, no substitution is performed. If present as a switch (/critFreq), the default value of 1.75 MHz is used; otherwise the value specified by the keyword is used. The keyword parameter \textit{c2Factor} supplies the scale factor.
used to convert counter 2 (low sensitivity) values to count 1 (high sensitivity values). If the keyword is absent, the default value of 5841 is used.

The keywords /plot, /annotate, and /noannotate, control graphical output that the procedure may produce. If the /plot keyword is present, a graphical display of the results are produced. The plot, similar to that in Figure 19, consists of a histogram of the input spectra overlaid with the result of reconstructing the spectra using the computed densities. A second panel displays the absolute value of the residuals scaled by the channels’ signal. The plot is annotated with the density values obtained by the deconvolution and additional diagnostic information. If the keyword /noannotate is supplied, only the histogram with the reconstruction is displayed. The keyword annotate provides more control of the annotation. You supply a list of annotation elements to include on the plot. The elements may be “residual”, “observation”, “deconvolution” or “none”, which control the residual plot, the table of observation conditions, and the table of deconvolution results, or turns off the annotation. Each element name may be abbreviated by its first letter.

The keyword parameters chisqr, kernel, model, and residual are used to obtain optional diagnostic information. You supply variable names as the value of each of these keywords. The variable supplied as the value of chisqr will be set to the reduced $\chi^2$ of the fit. The variable supplied with the kernel keyword will contain a structure holding the kernel matrix and the mass values that make up the forward model. The structure is defined as

```
kernel = $
{\text{anKernel: anKernel, } }$
{\text{anMass: intarr(nMassCount)}}
```

The field anKernel contains the matrix $K$ from equation 1, and anMass is a vector containing the list of masses. The variable supplied with the model keyword returns a vector of the counter 1 counts reconstructed using the calculated densities. The residual keyword specifies a variable to contain the difference between the reconstructed spectra and the input.

If the /verbose keyword is present, additional output is displayed on your terminal. The /debug keyword controls the behavior of the procedure when an error occurs. It is not normally required.

The following IDL code fragment illustrates the use of inms_deconvolve. The first two statements read the calibration data and level 1A data. The next two statements extract a set of spectra from the data and average them. The final statement invokes the deconvolution, which produces the image shown in Figure 19:

```
inms_read_cal, axCal
inms_get_data, axData
inms_get_spectra, axData, axSpectra, source='csn', $
masstableid=[16, 17], $
utt ime=[01, 55810432L], $
alt_t=[1100, 1230]
inms_compute_mean_spectra, axSpectra, xSpectra0, /poisson
inms_deconvolve, axDens, $
xSpectra0, $ 
axCal, $
/critfreq, $ 
species=asSpecies, $ 
chisqr = nChiSqr, $ 
model=anModel, $ 
/plot, kernel=xKernel
```
9.3 Deconvolved Density Profiles

In order to obtain a density profile, the \textit{inms\_deconvolve} procedure must be called repeatedly, once for each altitude in the profile. The routine \textit{inms\_make\_profiles} is provided to simplify this process. The command syntax is similar to that of \textit{inms\_deconvolve}, where the keywords \texttt{species}, \texttt{critfreq}, \texttt{c2factor}, \texttt{plot}, \texttt{annotate}, \texttt{noannotate}, and \texttt{no28} are passed directly to \textit{inms\_deconvolve}. To use this routine, you use a command of the form:

\begin{verbatim}
inms_make_profiles, axSpectra, axCal, axProfile, anChisqr, anResidual, anFit, */,species=asSpeciesList}
{,critfreq=nFreq | /critfreq} {,c2Factor=nScaleFactor}
{,plot} {,noannotate} {,annotate=list}
{,print} {,file="filename"} {,verbose}
{,debug}
\end{verbatim}

The argument \texttt{axSpectra} contains an array of spectra to be deconvolved. The argument \texttt{axCal} contains the calibration data as supplied \textit{inms\_read\_cal}. The argument \texttt{axProfile} is the name of the variable to contain the array of structures which compose the profile of densities. The contents of each structure in the array are specified in Table 3. The sub-array \texttt{axProfile[n,*]} contains the profile of the \texttt{n}\textsuperscript{th} species.
Table 3, Density Profile Structure Contents

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSpecies</td>
<td>string</td>
<td>Identifies the species to which the profile pertains</td>
<td>—</td>
</tr>
<tr>
<td>nAlt</td>
<td>real</td>
<td>Altitude</td>
<td>km</td>
</tr>
<tr>
<td>nDensity</td>
<td>real</td>
<td>Species abundance at the specified altitude</td>
<td>cm³</td>
</tr>
<tr>
<td>nSigma</td>
<td>real</td>
<td>Standard deviation of the density</td>
<td>cm³</td>
</tr>
<tr>
<td>nMoleFcn</td>
<td>real</td>
<td>Species mole fraction at the specified altitude</td>
<td>—</td>
</tr>
<tr>
<td>nMoleSig</td>
<td>real</td>
<td>Standard deviation of the mole fraction</td>
<td>—</td>
</tr>
</tbody>
</table>

The argument *anchisqr* is the name of a one-dimensional vector which receives the $\chi^2$ statistic for the fit at each altitude. The arguments *anResidual* and *anFit* are two-dimensional arrays that receive the residual of the fit and the reconstructed spectra respectively. They are organized so that the $[*,N]$ element is the data for the $n^{th}$ altitude in the profile.

The keywords */print* and *file* control additional output. If */print* is present, the altitude profiles are displayed on the screen. The *file* keyword is used to supply the name of a file to contain the profiles. The data is written as a comma-separated-value file, with a row of headers indicating the contents of the file.

**9.4 Profile Display (inms_plot_density_profiles)**

To produce plots of the density profile you use the *inms_plot_density_profiles* procedure. This procedure produces a plot similar to the example in Figure 20. The plot displays either the abundance or the mixing ratio of selected species. The standard deviation of the values is displayed with error bars. The command syntax is

```
inms_plot_density_profiles, axProfile, asSpecies
  {,/molefraction} {,/samewindow}
  {,subtitle='An additional title string'}
  {keyword expresions accepted by idl plot routines}
```

The token *axProfile* is replaced by the name of the density profile array produced by *inms_make_density_profiles*. The parameter *asSpecies* is a string or array of strings containing the species to include in the plot. If the keyword */molefraction* is supplied the plot will display the mixing ratio of the species rather than the abundance.

The keyword *subtitle* allows additional information to be added to the plot title. It behaves differently than the keyword to the IDL plot routines. Unlike the IDL supplied routines, if you supply a subtitle string through this keyword, an additional line is added to the title shown at the top of the figure. The */samewindow* keyword inhibits the creation of a new plot window for the figure.
10. **Determination of Ion Abundance**

Ion abundance determination is, in principal, simpler than the neutral gas decomposition. No species produces counts in more than one channel and therefore all that is required is to determine the sensitivity and apply it to the mass spectra. You can use the routine `inms_make_ion_spectra` to perform these calculations. The routine computes the nominal sensitivity based on kinetic energy then adjusts for quadrupole lens tuning and ram-angle misalignment. To convert a raw ion spectra in counts per IP into an ion abundance spectra, you use the command

```
inms_make_ion_spectra, xRawSpec, xResult
    {,calfile=file_path} {,/noqlens} {,/angle} {,/debug}
```

You replace the token `xRawSpec` with the name of the input spectra and `xResult` with the name of a variable to hold the abundance spectra. The keywords `/noqlens` and `/angle` and `/test` control the adjustments to sensitivity for quadrupole switching lens and ram angle effects. If the `/noqlens` keyword is included, the quadrupole tuning adjustment is disabled. To enable the ram angle adjusted, set the `/angle` keyword. The `/debug` keyword controls the behavior of the routine when an error occurs and is not normally required.
The routine uses the correct quadrupole lens tuning parameters to adjust the instrument sensitivity and the velocity components to compute the adjustment due to angle between the ram direction and the normal to the inlet plane. The resulting spectrum is contained in a spectra structure of the same form as the input structure. This structure can be used as input to the `inms_plot_histogram` or any of the other routines that consume the spectra records. For example, you can plot an array of ion density spectra using the `plot_stacked_spectra` routine to produce a plot similar to the example in Figure 21. The `/ramangle` option may be used to plot the angle between the spacecraft velocity and the instrument boresight to determine where the densities are valid.

![Cassini-INMS Stacked Abundance Spectra](image)

Figure 21, Example Ion Density Spectra
produced by the command `inms_plot_stacked_spectra, axDenSpec, /ramangle`

11. **Support Routines**

11.1 **Graphics Support**

11.1.1 **Graphics Device Configuration** (inms_prepare_plot)

The plotting procedures described above direct their output through the current IDL graphics device. If the device is the postscript device a plot file can be produced. To simplify the management of the graphics device, the analysis library includes the `inms_prepare_plot` procedure which you use to switch between devices. The procedure makes the necessary changes to the IDL
plotting environment to insure that plots are nearly identical in appearance regardless of the plotting device. The procedure also manages the files used to store graphics files. To create graphics files, you use the `inms_prepare_plot` command twice, once prior to the plotting commands to initialize the plotting device and once afterwards to save the file.

You use `inms_prepare_plot` to capture graphics output in a file. You set up the file capture by issuing the command with the `init` keyword as follows:

```
inms_prepare_plot init="device" |/init {,/black | /white} {,/publication} {,resolution=[hh,vv]} {,/portrait} {,path="dir/path"} {,file="filename"}, {,sequential{"tablename"}|/divergent}
```

where the token `device` may be replaced with, X, WIN, NULL, PS, IMAGE, PNG, TIFF, or JPG. Supplying the `init` keyword a value of PS results in the creation of a postscript file from subsequent graphics commands. Providing the value X makes the graphics device the X window server, WIN makes the graphics device the Microsoft Windows device, NULL disables graphics output and the PS value yields a postscript file. The value IMAGE yields a portable network graphics (PNG) file, while the PNG, TIFF and JPG values yield the corresponding graphics files. The postscript device can produce graphics files containing more than one page of graphics, while the image files may contain only one. When producing displays on an X window server or on Microsoft Windows platform, each plot appears in a new window by default. Specifying the /init keyword without a device sets the IDL plotting device to the X device on UNIX and Mac OSX systems and the WIN device on MS Windows systems. When one of the image devices is initialized, a flag is set that may be examined using the `inms_is_image` which returns 1 if the plot is directed to a PNG, TIFF or JPG file and 0 otherwise.

The keywords `/black`, `/white`, `resolution`, `/portrait` and `/publication` provide control of the plots appearance. The first two set the background color for the X, WIN or image devices but is not supported for postscript. To plot on a black background you include `/black` keyword with the `init` keyword, including the `/white` keyword results in plots on a white background and is the default. The `resolution` keyword is used to change the resolution for the image file devices. The default is 1280 by 960. To change it, supply a two-element vector value to the `resolution` keyword, the first element of which is the horizontal resolution and the second the vertical. The `/portrait` keyword applies to the postscript device resulting in a portrait orientation plot.

The `/publication` keyword is used to indicate that “publication quality” plots are desired. The routine `inms_is_publication` may be used to determine the value of this switch. The behavior of `inms_plot_histogram` and `inms_plot_mt_spectra` are modified by this option as well. In each, the main title and the date of preparation are omitted from the plot. You can test whether the `/publication` keyword has been set using the `inms_is_publication` function, which returns 1 if the keyword has been set, and zero if not.

Color display is controlled by the `sequential` and `/divergent` keywords. Supplying a color table name, blue, red or spectrum with the `sequential` keyword selects the specified table (See the examples in Figure 22). The `/divergent` keyword specifies a color table that varies from blue to white to red. The color table that you select becomes the default for future calls and defaults initially to the blue sequential table.

The remaining keywords are used to specify the name and location of the resulting plot files. By default, files are placed in the current working directory and are named `INMSplot_nnnnn.type`, where nnnnn is a unique identifying number and type is either PS, PNG, TIFF or JPG. To change the directory in which the plot files will be saved you supply the directory path as the `path` keyword value. Once set, it becomes the new default until changed or
the IDL session is terminated. To supply a file name you use the file keyword, specifying the name of the file, excluding the file type extension.

Once all of the plotting commands are executed, you save the plotting file by issuing the `inms_prepare_plot` command again with either the /done, /spool or /next keywords.

```
inms_prepare_plot, /done | /spool | /next
```

To simply close the file, you use the /done keyword. The file is saved and the plotting environment is returned to the configuration prior to the initial `inms_prepare_plot` command. The /spool keyword performs the same actions as the /done keyword and additionally spools the resultant file to the printer. The following example illustrates preparing and printing a postscript file:

```
inms_prepare_plot,init='ps',path="~/plotdir"
inms_plot_histogram,xSpec
inms_prepare_plot,/spool
```

To save the file without printing substitute /done for /spool.

The /next keyword is only applicable to the image devices. Unlike a postscript file, an image file may only contain one plot. To simplify creating multiple image plots, you can use the `inms_prepare_plot` command with the /next keyword to write the current image to a file and prepare for another plot, replacing two `inms_prepare_plot` calls with one. In this case the file name is augmented by the string "-Pnnn" where nnn is replaced by a three digit page counter. The files are collected in a subdirectory of that supplied with the path keyword whose name is the basic filename supplied by `inms_prepare_plot`. After completing the last plot of the series, use the /done keyword. The following example shows the use of the next command to make a series of plots:

```
inms_prepare_plot,init='image',path="~/plotdir",file='HIST'
for nI=0,10 do begin
  inms_plot_histogram,axSpec[nI]
inms_prepare_plot,/next
enddo
inms_plot_stacked_spectra,axSpec
inms_prepare_plot,/done
```

This example plot 11 histograms followed by a spectra. The files are named HIST-P00.PNG, HIST-P001.PNG...HIST-P010.PNG and stored in the directory ~/plotdir/HIST. You can also specify both the /next and /spool keywords to spool the intermediate files to the printer.

11.1.2 Creating Image Files (`inms_write_image`)

In order to create an image file, PNG, TIFF or JPG, it is necessary to first create the image as either a pixel map or in a Z-buffer. Once this is done, you must read the pixel map or buffer then write the resulting array to a file. IDL provides a routine called `write_buffer` to perform the final step. The INMS library contains a routine similar to the IDL supplied program which is customized for use with the rest of the library. In particular it assumes that the image is a pixel map and uses the IDL routine `tvrd` to read the image out of the pixel map. It also properly distinguishes between pseudo-color and true-color pixel maps, forming the image correctly in each case.

To use this routine, you must first create an image as a pixel buffer. You can use `inms_prepare_plot` to do this set-up or you can issue the proper IDL commands yourself. Once the pixel buffer preparation is completed and all the plotting performed, you invoke the routine as follows:...
The token `sFilePath` is replaced with the name of the file to contain the image. The type keyword is used to specify the type of image file to create. You may also pass image-specific keyword to the IDL output procedures.

### 11.1.3 Window Management

The display procedures place each plot in a new X window by default. The windows are numbered and titled to make it easier to bring windows of interest to the front for viewing. The creation of these windows is performed by `inms_make_window`. This procedure may be used directly, when you want to place an ad-hoc plot in a new window. To create a new window, invoke the command

```
inms_make_window {,"title"} {,samewindow} {,/animate}
```

where “title” is a title string to place in the border of the X window. If the parameter `samewindow` is nonzero, the window is, in effect, reused. In order to re-title the window, it is deleted and recreated with the new title. The keyword `/animate` causes the current window to be actually re-used, inhibiting the renaming of the window.

You can also use the routine `inms_make_window` to set the size and location of the windows. To set these properties you invoke the procedure in the following manner:

```
inms_make_window /winset, {winsize=nFracSize} 
{,winpos=nFracPos} {,/portrait}
```

The keyword `winsize` specifies the vertical size of the window as a fraction of the screen width, the height is computed to make the window aspect ratio the same as a 8.5 x 11 sheet in the landscape orientation. The `winpos` keyword specifies portion of the display to be used for windows. If absent, the entire display is used. If present, the value `nFracPos`, is the fraction of the screen starting at the upper left corner that will contain windows. When the keyword `/portrait` is present the window is oriented in the vertical, portrait orientation, otherwise it is oriented in the horizontal, landscape orientation.

After creating a number of plots, many X windows may be open consuming significant computer resources. Closing them all interactively can be tedious. The command `inms_close_windows` will close all open windows.

### 11.1.4 Color Table Management

The INMS library includes routines to load the color table and to determine the color index corresponding to a named color. The routine `sprl_load_colors` may be used to load one of five predefined color tables. The four continuous color tables are shown in Figure 22. The command syntax is

```
sprl_load_colors, /divergent | /categorical 
| sequential(=blue|red|spectrum)
```

The keywords specify the color table to load. If the `/divergent` keyword is present, a color table that varies from blue to white to red is loaded, shown in the left hand column of Figure 22. The `sequential` keyword specifies either a blue or red monotonic scale or a spectra scale. These scales are shown in the three columns to the right in Figure 22. The blue sequential color table is
the table used for all of the examples in this memo. The /categorical keyword results in loading the first 87 entries in the table with specific colors.

The divergent and monotonic sequential scales are based on color definitions by C.A. Brewer at the following web site http://www.personal.psu.edu/~cab38. General information on choosing color scales that permit the widest audience of viewers to perceive the images as intended may be found at http://geography.uoregon.edu/datagraphics.

![Color Tables](image)

**Figure 22, Color Tables**

### 11.1.5 Selecting Discrete Colors

Two routines are provided to obtain a numerical value corresponding to a specific color. The first, `sprl_find_color_index`, returns the 8 bit color index or the 24 bit color value for a named color. The second, `sprl_color_triad`, returns a three element vector containing 8 bit red, green and blue color values. The colors and their names are shown in the color swatch (Figure 23).

To obtain the index of a specific color, you use the `sprl_find_color` function, whose syntax is:

```
  nResult = sprl_find_color( {sName} {,/index} {,/swatch} )
```

You replace the token *sName* by the name of one of the available colors. The case and white space within the name is not significant. If no name is specified, a list of the available colors is produced. If the */swatch* keyword is specified, the example color lists shown in Figure 23 is produced. For example the command

```
nResult = sprl_find_color('darkgoldenrod')
```

returns the color index for color 54, Dark Goldenrod.

If the current device is a 24 bit display, the return value in *nResult* is the value that corresponds to the requested color. For 8 bit color devices such as postscript, the return value is 254 and that entry in the color table is loaded with the requested color. In the */index* keyword is present, the index of the color closest to the request in the color table is returned. The */index* keyword has no effect for 24 bit color displays.

IDL object graphics use a different way to specify colors. Instead of an index or 24 bit color number, object graphics methods use a 3 element color triad containing the red, green and blue values. To obtain a color triad corresponding to one of the named colors you use the *sprl_color_triad* function.

The command syntax is

```
anResult = sprl_color_triad(sName)
```

where *sName* is the name of the color. The return value *anResult* is a three element byte vector containing the color triad.
11.1.6 Tick Mark Formatting

The library includes two routines for creating time axis labels, `inms_neat_ticks` and `inms_label_ticks`, which are used together to create axis labels at convenient locations. By using these routines, tick marks can be placed on even minutes, hours, days, months and years. Furthermore, the remaining tick marks on the axis will also be on the same interval boundaries.

You use the routine `inms_neat_ticks` to determine the values to supply to the `xrange`, `xv`, `xticks`, and `xminor` keywords for the IDL `plot` or `axis` routines. The routine `inms_label_ticks` is a call-back routine whose name is supplied to the IDL `plot` or `axis` routines to produce the string used as tick mark labels. The following code fragment illustrates their use:

Figure 23, Available Colors by Name
### Figure 24, Example `inms_neat_ticks` Usage

Note that these routines require that the Julian date be used as the independent variable for the plots. The first section of the code provides an example of conversion from the PDS date and time string to the Julian date. Once the time is converted to Julian dates, the `inms_neat_ticks` function determines where the tick formatting quantities. Next the `inms_label_ticks` function is called to set the format to be used for the tick mark labels. Finally, the plot is produced with the values for the `xrange`, `xtickv`, `xticks` and `xminor` keywords returned by `inms_neat_ticks` and providing the name of the callback routine `inms_label_ticks` as the value for the `xtickformat` keyword. An example of the use of these routines may be found in `inms_plot_hkg`.

The command syntax for `inms_neat_ticks` is

\[
\text{xTickDef} = \text{inms_neat_ticks}( \text{anRange}, \text{tickinterval}='\text{type}')
\]

The token `xTickDef` is replaced with the name of the IDL variable to contain the tick definition structure. You replace `anRange` with a two element vector containing the minimum and maximum value of Julian date to be plotted. The token `type` is replaced with the time unit on which the ticks are to be placed. It may have YEAR, MONTH, DAY, HOUR, QUARTER or MINUTE for a value. The value QUARTER results in tick marks on quarter-hour boundaries. The contents of the returned structure is shown in the example.
The command syntax for `inms_label_ticks`, when called directly, is

```c
void = inms_label_ticks(format='type')
```

The function does not return a useful value when called directly. When called by the `plot` procedure, it returns a character string to use as the tick mark label. The value supplied to the format keyword specifies what those labels should look like. The keyword may take the values `CAL`, `JD`, `DOY` or `TOD`. The `CAL` keyword results in two line labels, with the first line containing the date as month, day and year and the second line containing the time of day. The `JD` keyword results in the Julian Date values as labels. The `DOY` keyword results in two line labels, with the first line containing the date as year and day-of-year and the second line containing the time of day. Finally, the `TOD` keyword results in a one line label with the time-of-day in hours, minutes and seconds being displayed.

11.1.7 Color Plots (`sprl_colorplot`)

The mass-time plots both use the routine `sprl_colorplot` to produce a plots of a function of two variables. The procedure accepts a number of keywords that can be passed to it from the higher level routines. The `sprl_colorplot` command syntax is

```c
sprl_colorplot, anZ {,anXin} {,anYin}
{,zrange=[zmin,zmax]} {,zttitle='title for scale'}
{,/logsw} {,/xwrap}, {,/ywrap} {,/noerase}
{,/contour} {,smooth=n | smooth=[hs,vs]} {,missing=n}
{,region=[x0,z0,x1,y1]} {,margins=[lt,lo,rt,up]}
{,gutter=n},
{keyword expressions accepted by plot and contour}
```

The token `anZ` is replaced by the name of a two dimensional array containing the function to plot. The array should be organized so that increasing the first index corresponds to increasing values of the first independent variable, `X` and increasing the second index corresponds to increasing values of the second, `Y`. The optional vectors `anXin` and `anYin` specify the values of the independent variables. If they are supplied as two element vectors they specify the minimum and maximum values. If the vectors are absent, the dependent variable is plotted as a function of its indices.

The keyword `zrange` is used to specify the range of the independent variable to include in the plot. Values less than the minimum are displayed as the first color in the color table, while values greater than the maximum are displayed as the last color in the table. If the `/contour` keyword is set to add contour lines to the plot, values outside of the specified range are ignored. The value supplied via the `missing` keyword is used to indicate a data cell whose contents are missing. The value must be outside the range specified by `zrange`.

The keyword `zttitle`, `/logsw`, `smooth` `/xwrap`, and `/ywrap`, control the format of the plot. You use the `zttitle` keyword to specify a title for the z-axis color scale. This title is placed to the side of the color scale. If the `/logsw` keyword is present, the z-axis is logarithmic. If absent, the axis is scaled linearly. The smooth keyword controls the smoothing of the dependant variable before plotting. If present, and a scalar, the data is smoothed horizontally and vertically with a boxcar smoothing algorithm using a smoothing window of the specified number of pixels. If a vector, the elements `hs` and `vs` specify the horizontal and vertical smoothing windows, respectively. The `/xwrap` and `/ywrap` control how the routine handles the boundaries. If set, the data is assumed to be periodic in the corresponding axis for the purpose of smoothing and contouring.

The keyword `/noerase` prevents the erasure of the plotting device prior the making the color plot. You would use this keyword when placing more than one color plot an a page.
The keywords region, margin and gutter control the placement of the plot on the page. The region keyword is used to specify the entire plot region, which encompasses the plot, the color scale, and the titles. It is expressed in normal units as a four element vector \([x_0, y_0, x_1, y_1]\), with \(x_0, x_1\) specifying the left and right edge of the region and \(y_0, y_1\) specifying the bottom and top. The margin keyword specifies the margins between the left side of the color plot and the plot region, the lower margin, the upper margin, and the margin on the right between the color bar and the right edge of the plot regions. The keyword gutter specifies the spacing between the color plot and the color bar. Margin and gutter are specified as fractions of the plot region given by region.

In addition to the keywords explicitly defined for the sprl_colorplot procedure, you may include any keywords accepted by the IDL plot or contour procedures. Of particular use are those that supply axis titles, control tick mark formatting or contour intervals.

11.2 PDS Spreadsheet Routines

The INMS analysis library uses structures and arrays of structures to contain data read from PDS compliant spreadsheet files. The PDS spreadsheet files are labeled with detached labels, a file containing a description of the data and of the spreadsheet. In addition to the routine inms_read_pds_spreadsheet, described in section 4.2, above, the library contains three additional routines which support PDS spreadsheet access, inms_read_label_file, inms_read_label_file, and inms_make_data_structure. The first two routines obtain file meta data relating to PDS spreadsheets and the third creates IDL structures to contain the spreadsheet data.

The user will not normally need to call these routines explicitly, since they are invoked where needed in the spreadsheet reading routines. They are documented here for completeness and to assist those who need to read other spreadsheets.
11.2.1 Reading PDS Labels (*inms_read_label_file*)

The routine *inms_read_label_file* is used to read detached PDS labels. Labels consist of a series of keyword-value pairs. For a full explanation of PDS standards, the reader is encouraged to visit the PDS web site.

To read a PDS label, you issue the *inms_read_label_file* command as follows:

```
inms_read_label, xLabel{}, file='FilePath', /debug
```

The token `xLabel` is replaced by the name of the format structure. The file to read is specified by the `file` keyword. If absent, a file selection dialog is presented. The `/debug` switch controls the behavior of the routine when an error occurs.

The structure returned contains a field for each keyword-value pair found in the label file. All structure fields are strings, except for object keywords. In that case the structure field is an imbedded structure, with fields for each keyword describing the object. Figure 25 is example of the returned structure.

The routine ignores any field keywords found, as those are parsed by the *inms_read_fmt_file* procedure, documented below. The fields `PTR_HEADER` and `PTR_SPREADSHEET` contain the location of the header row of the spreadsheet and the spreadsheet data in terms of file, and byte location. The `SPREADSHEET` field, also shown in the figure, specifies the size and format of the spreadsheet data. The field `PTR_STRUCTURE` in the spreadsheet points to a file that will

---

**Structure <2913a04>, 27 tags, length=420, data length=420, refs=1:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>STRING</td>
<td>'200732300_HKG_01.LBL'</td>
</tr>
<tr>
<td>PDS_VERSION_ID</td>
<td>STRING</td>
<td>'PDS3'</td>
</tr>
<tr>
<td>DATA_SET_ID</td>
<td>STRING</td>
<td>'CO-S-INMS-2-PKT-U-V1.0'</td>
</tr>
<tr>
<td>STANDARD_DATA_PRODUCT_ID</td>
<td>STRING</td>
<td>'INMS_PKT_HKG'</td>
</tr>
<tr>
<td>PRODUCT_ID</td>
<td>STRING</td>
<td>'200732300_HKG_01'</td>
</tr>
<tr>
<td>PRODUCT_TYPE</td>
<td>STRING</td>
<td>'DATA'</td>
</tr>
<tr>
<td>PRODUCT_VERSION_ID</td>
<td>STRING</td>
<td>'01'</td>
</tr>
<tr>
<td>PRODUCT_CREATION_TIME</td>
<td>STRING</td>
<td>'2007-325T17:55:11'</td>
</tr>
<tr>
<td>RECORD_TYPE</td>
<td>STRING</td>
<td>'STREAM'</td>
</tr>
<tr>
<td>FILE_RECORDS</td>
<td>STRING</td>
<td>'1042'</td>
</tr>
<tr>
<td>MD5_CHECKSUM</td>
<td>STRING</td>
<td>'99f5a757e9a038e1f50b26dd367951a'</td>
</tr>
<tr>
<td>START_TIME</td>
<td>STRING</td>
<td>'2007-325T00:00:00'</td>
</tr>
<tr>
<td>STOP_TIME</td>
<td>STRING</td>
<td>'2007-325T23:59:59'</td>
</tr>
<tr>
<td>INSTRUMENT_HOST_NAME</td>
<td>STRING</td>
<td>'CASSINI ORBITER'</td>
</tr>
<tr>
<td>INSTRUMENT_HOST_ID</td>
<td>STRING</td>
<td>'CO'</td>
</tr>
<tr>
<td>MISSION_PHASE_NAME</td>
<td>STRING</td>
<td>'TOUR'</td>
</tr>
<tr>
<td>TARGET_NAME</td>
<td>STRING</td>
<td>'TITAN'</td>
</tr>
<tr>
<td>INSTRUMENT_NAME</td>
<td>STRING</td>
<td>'ION AND NEUTRAL MASS SPECTROMETER'</td>
</tr>
<tr>
<td>INSTRUMENT_ID</td>
<td>STRING</td>
<td>'INMS'</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>STRING</td>
<td>'This file contains INMS housekeepi...'</td>
</tr>
<tr>
<td>NOTE</td>
<td>STRING</td>
<td>''</td>
</tr>
<tr>
<td>PTR_HEADER</td>
<td>STRING</td>
<td>'(&quot;200732300_HKG_01.CSV&quot;,1 &lt;BYTES&gt;)'</td>
</tr>
<tr>
<td>HEADER</td>
<td>STRUCT</td>
<td>&lt;Anonymous&gt; Array[1]</td>
</tr>
<tr>
<td>PTR_SPREADSHEET</td>
<td>STRING</td>
<td>'(&quot;200732300_HKG_01.CSV&quot;,1596 &lt;BYTES&gt;)'</td>
</tr>
<tr>
<td>SPREADSHEET</td>
<td>STRUCT</td>
<td>&lt;Anonymous&gt; Array[1]</td>
</tr>
</tbody>
</table>

**Structure <2272194>, 6 tags, length=72, data length=72, refs=2:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTTYPE</td>
<td>STRING</td>
<td>'SPREADSHEET'</td>
</tr>
<tr>
<td>ROWS</td>
<td>STRING</td>
<td>'1041'</td>
</tr>
<tr>
<td>ROW_BYTES</td>
<td>STRING</td>
<td>'925'</td>
</tr>
<tr>
<td>FIELDS</td>
<td>STRING</td>
<td>'150'</td>
</tr>
<tr>
<td>FIELD_DELIMITER</td>
<td>STRING</td>
<td>'COMMA'</td>
</tr>
<tr>
<td>PTR_STRUCTURE</td>
<td>STRING</td>
<td>'HKG_STRUCT_01.FMT'</td>
</tr>
</tbody>
</table>

---

Figure 25, Label Structure Example
describe each item in the spreadsheet.

11.2.2 Reading PDS Structure Files (inms_read_fmt_file)

A PDS structure file functions like an include file in a programming language. They are used to contain text that repeats verbatim in multiple files. In the case of PDS labels, they are used to contain the, possibly lengthy, list of spreadsheet fields. Each field is described by a field object. The inms_read_fmt_file procedure, reads a structure file, parsing each field object that it contains. When the label file itself contains the field descriptions, this routine is used to read the label file to extract the spreadsheet fields creating a template to be used to create a data structure.

To read a PDS structure file, you issue the inms_read_fmt_file command as follows:

```
inms_read_fmt_file, axTemplate{}, file='FilePath'
   {,/verbose} {,/debug}
```

The token axTemplate is the name of a variable to contain the format information. If the keyword file is absent, and no format was previously read, a file selection dialog will appear. If the file keyword is absent and a format file has been read, the cached format will be returned. The /verbose keyword causes additional status output to be displayed. The /debug keyword controls the program behavior when an error occurs.

The structure array returned by the procedure contains one structure element for each field object in the file. The structure elements contain the information used by inms_make_data_structure to create a variable to contain the spreadsheet data.

11.2.3 Creating Data Structures (inms_make_data_structure)

The function inms_make_data_structure returns a structure with one field for each field in a PDS spreadsheet file according to the format contained in the PDS label. Optionally, an array of structures can be returned, creating an array to contain multiple spreadsheet records. You create a spreadsheet data structure by issuing the command

```
axData = inms_make_data_structure(axTemplate {,nRecords}
   {,/debug}
```

The variable axData is an empty structure created to hold spreadsheet data according to the prescription contained in the axTemplate structure. The argument nRecords, if present, sets the number of elements in the axData array. The /debug keyword controls the behavior of the function in the event of an error.

11.3 Programming Utilities

11.3.1 Creating IDL Save Sets (inms_save_data)

The INMS analysis library makes use of a number of global data items, that are not normally accessible at the command prompt. As a result, using the IDL save procedure fails to copy those variables to the save set. This routine creates save sets that include that data. The data is restored by the inms_restore_data, described in the following section.

To save data, you issue the inms_save_data command as follows:

```
inms_save_data, sVariableList {,/file=file}{,/verbose}
```

The parameter sVariableList, a comma-separated list of variables to save. These variables may be any variables defined at the current program level. The file keyword expression is used to specify the path to the save file. If absent, the standard IDL default “idlsave.dat” is created in
the current working directory. The /verbose keyword causes an information message to be written to the console device for each variable saved.

11.3.2 Restoring IDL Save Sets (inms_restore_data)

Save sets created by the inms_save_data procedure contain data that needs to be restored to common blocks. This can be done at the command line by creating an instance of the common block prior to using the IDL restore procedure. To simplify this process, you can use the inms_restore_data procedure.

The command syntax is

```
inms_restore_data [,file=filename] {,/verbose} {,/debug}
```

The name of the file to restore is supplied with the file keyword expression. If the expression is omitted, the file “idlsave.dat” in the current working directory is restored. The /verbose keyword causes an informational message to be displayed for each restored data item. The /debug keyword results in additional informational messages to be displayed.

Variables with the same name as those in the save file may already exist at the main program level. In this event, you must confirm the replacement of the existing variable with the variable in the save file by clicking “Yes” on the confirmation dialog that is presented.

11.3.3 Displaying Error Dialogs (inms_post_message)

The normal IDL error message output is mediated by the IDL message procedure. This procedure causes a message to be displayed on the standard output device and the action specified via the on_error procedure to occur. In cases where a number of windows are open, this may be insufficient because the message is overlooked. The inms_post_message procedure was written to address this. It determines whether or not the program is running in an interactive windowing environment. If so, it uses the dialog_message function to display a message, otherwise it writes the message to the standard output. To redirect the output from the message procedure, the routine must be called from an error handler. Basically, when an error handler is present, the message procedure results in control being transferred to the error handler, at which point user code, including inms_post_message, may be used to respond to the error.

The syntax of the inms_post_message procedure is

```
inms_post_message {,sMessage} {,/console} {,/traceback}
   {,/error} {,/warning} {,/information}
```

The token sMessage is replaced by the message to display in the dialog. If absent, the message is obtained from the IDL !error_state variable which will contain the message passed as an argument to the message procedure. The /console keyword forces the message to be written to the standard output regardless of whether a dialog may be posted. The /traceback keyword adds additional diagnostic output to the message. The final keywords /error, /warning, and /information specify the type of message icon displayed in the dialog. If none of these three keywords are present, the default message type is ERROR.

The code fragment in Figure 26 illustrates the use of this routine. Code similar to this should be among the first executable statements in a routine that will use this facility to display dialogs. Lines 1, 2 and 3 establish an error handler. On initial execution, the code between lines 3 and 7 will not be executed because the first line set the error code to the value for no error. Should an error occur or the message procedure be executed, control is transferred back through line 2 of this fragment placing a non-zero value into the variable nErrCode. Line 4 ensures that an error
following that line will not result in an infinite loop. Line 5 posts the dialog, including the call tree and line 6 returns control to the routine that called the routine containing this fragment.

```plaintext
1  nErrCode = 0
2  catch,nErrCode
3  if nErrCode ne 0 then begin
4    catch,/cancel
5    inms_post_message,/traceback
6    return
7    endif
8... ;; continuation of program
```

Figure 26, Example inms_post_message Usage

In addition to invoking the routine within an error handler, it can be called anywhere within a program to post an informational message. An example of this usage is as follows:

```
inms_post_message, 'AnInformationMessageString', /info
```

The first argument is a string scalar containing the message to display, while the /information keyword overrides the default error message type.

You must use caution when using this routine while creating image files. Since the image is first produced in a pixel map, the inms_post_message routine will attempt to use the dialog rather than the desired behavior of writing the message to the standard output device. To ensure that the message is correctly routed, use the /console keyword when producing pixel maps for output to an image file. The inms_prepare_plot routine sets a flag to indicate that a pixel map is the output for the graphics. The flag is accessible using the function inms_is_image so using the following keyword expression ensures that the message is properly directed.

```
console=inms_is_image()
```

For a more sophisticated example of the use of inms_post_message, you can examine the source code of inms_plot_series.

11.3.4 Structure Output (inms_dump_structure)

The analysis library makes use of many structures. You may occasionally want to save one of these structures in a non-proprietary format. The IDL save procedure is not adequate because it saves the structure in a binary format only useful with the restore procedure. The routine inms_dump_structure copies the contents of a structure into a comma-separated data file, with column headings and an optional note. To produce a structure dump file, you use the following command:

```
inms_dump_structure, axStruct {,file=filepath}
{,note="note text to include in file"} {,/debug}
```

The token axStruct is replaced by the name of the structure to be copied to a file. You specify the file to contain the data using the file keyword expression. If the keyword expression is absent, you will be presented with a file selection dialog. If the file exists, you will be presented with a dialog to confirm that the file may be overwritten. The optional note keyword expression is used to supply a descriptive note to include in the file.
11.3.5 PDS Label Construction (\texttt{inms\_make\_pds\_label})

Files that are PDS compliant require a data file, a structure file and a label. For a group of files containing data in the same format, the structure file does not change while the label is different for each file. These labels can be created with a text editor, but this process is both tedious and error-prone. The routine may be used to create label files from a template. The template contains all of the label text, properly formatted, that is constant from one data file to the next. Items that depend on the data file are represented in the template by tokens. The tokens are replaced with values obtained by reading the specific data file for which a label is being created. Supported tokens are listed in Table 4. Example label templates are included with the library.

<table>
<thead>
<tr>
<th>token</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>!&lt;file&gt;!</td>
<td>name of the pds spreadsheet file</td>
</tr>
<tr>
<td>!&lt;records&gt;!</td>
<td>the number of records in the file</td>
</tr>
<tr>
<td>!&lt;MD5&gt;!</td>
<td>the MD5 checksum</td>
</tr>
<tr>
<td>!&lt;rows&gt;!</td>
<td>the number of rows in the spreadsheet</td>
</tr>
<tr>
<td>!&lt;hdrlen&gt;!</td>
<td>the number of rows in the header, defaults to 1</td>
</tr>
<tr>
<td>!&lt;note&gt;!</td>
<td>a text note, defaults to empty</td>
</tr>
<tr>
<td>!&lt;startbyte&gt;!</td>
<td>the starting byte of the spreadsheet</td>
</tr>
<tr>
<td>!&lt;createdate&gt;!</td>
<td>the file creation date stamp</td>
</tr>
</tbody>
</table>

The syntax for this command is

\begin{verbatim}
inms\_make\_pds\_label, sPDSfile, sTemplateFile
{},hdrlen=nCnt{,note=\textit{note text string}}{,/debug}
\end{verbatim}

The first two arguments are the path to the PDS file and the path to the template file. The \texttt{hdrlen} keyword expression provides the number of rows in the header portion of the file. The \texttt{note} keyword expression is used to supply a value for the NOTE PDS keyword. The \texttt{/debug} keyword controls the behavior of the procedure when an error occurs and additional output for debugging. It is not normally required.

11.4 Widget Support

The INMS library contains two routines to support applications with a widget GUI. One routine, \texttt{inms\_post\_status}, puts lines of text into a scrolling text widget created by \texttt{inms\_status\_widget}. Widget programming is beyond the scope of this document, for details see “User Interface Programming” in the IDL manual set or on-line help.
11.4.1 Status Widget Creation (*inms_status_widget*)

The function *idl_status_widget* is used within a widget application to create a text widget known to *inms_post_status* for the purpose of displaying status messages. You create this widget by including the following statement in the widget application:

```idl
wiStatus = inms_status_widget, wiParent {,xsize=xsize}
           {,ysize=ysize} {,value=value} {,/wrap}
```

The token *wiParent* is replaced with the identifier of the widget which will contain the status widget. The *xsize* and *ysize* keyword expressions provide control for the width and height of the widget. You can supply an initial message using the *value* keyword expression. Including the */wrap* keyword causes the text strings to wrap at the right margin, otherwise a horizontal scroll bar is presented.

11.4.2 Displaying Messages (*inms_post_status*)

Once the status widget is created using *inms_status_widget*, you can post messages to it using the *inms_post_status* procedure. This procedure checks for the existence of the widget created with *inms_post_status*. If it exists, the message is displayed in the widget’s scrolling text window. If the widget does not exist, the data is written to the IDL standard output. The procedure *inms_post_message* uses this procedure to support widget applications.

The command syntax is

```idl
inms_post_status, sStatus
```

where the parameter *sStatus* is a string variable or constant containing the string to be displayed.

11.5 Spice Kernel Management

The INMS level 1A data contains only those position, velocity and pointing data considered most generally useful. Additional quantities can be obtained using the SPICE toolkit and kernel files. While the details of such computations are beyond the scope of the data analysis library and this user’s guide, a routine for confirming the presence of required kernel files and a routine for obtaining kernel files are included. The routine *inms_kernel_list* provides a list of present or absent kernel files. The routine *inms_get_spice_kernels* obtains kernel files from the NAIF that are not present on the user’s local system. Kernel files are loaded by the *furnish* SPICE toolkit procedure. This procedure accepts the name of a text file, called a *furnish* text kernel which specifies the paths to the kernel files. An example text kernel file shown in Figure 27. The keywords *PATH_VALUES* and *PATH_SYMBOLS* provide symbolic file path names.

The library also contains a frames text kernel that defines ionospheric interaction system (IIS) coordinate frames for Titan and Enceladus. When this frame is loaded using the SPICE *furnish* procedure, you can use the SPICE toolkit routines to translate between the IIS frames and any other frame supported by the toolkit.
\begindata
PATH_VALUES = (  
'/usr/local/spice/kernels/generic',  
'/usr/local/spice/kernels/cassini'  )

PATH_SYMBOLS = ('GEN', 'CAS')

KERNELS_TO_LOAD = (  
'GEN/lsk/naif.tls',  
'GEN/pck/pck.tpc',  
'CAS/sclk/cas.tsc',  
'CAS/fk/cas.tf',  
'CAS/spk/981005_PLTEPH-DE405S.bsp',  
'CAS/spk/sat164.bsp',  
'CAS/ik/cas_inms_v02.ti',  
'CAS/spk/060323AP_SCP5E_06082_08222.bsp',  
'CAS/ck/07087_07162pc_port2.bc',  
'CAS/ck/07124_07162pe_psiv2.bc',  
'CAS/ck/07087_07124pe_psiv2.bc',  
'CAS/ck/07124_07134pa_fsiv_livpud_DOY_132.bc',  
'CAS/ck/07102_07107ra.bc',  
'CAS/ck/07107_07112ra.bc',  
'CAS/ck/07112_07117ra.bc',  
'CAS/ck/07129_07131ra.bc'  )
\begintext

Figure 27, Example *furnish* Text Kernel

11.5.1 Checking Spice Kernel Presence (*inms_kernel_list*)

This routine reads and parses a SPICE text kernel containing a list of kernel files to be loaded. Normally, this file is an argument to the *furnish* routine in the SPICE library, however missing kernel files do not cause an error until data within them is required. The *inms_kernel_list* routine checks for missing files and can be used before loading them. Its output is a list of files from the furnish kernel that are absent from the system on which the routine is run.

The syntax for this routine is

\[
\text{inms_kernel_list, } sFile \{,/present|present=fileList\} \{,/absent | absent=fileList\} \{,/verbose\}, \{,/debug\}, \{,symbol=path\} \{,symbol=path\}...
\]

The token *sFile* is replaced with the full path to a furnish text kernel. The keywords /present and /absent control the output. If neither is specified or if the /absent keyword is specified, the program produces a list of kernels that are not present on the system. If the /present keyword is specified a list of those kernels in the file that are on the system is produced. If a
variable name is supplied to either the present or absent keyword, the list of files is returned in
the named variable.

The symbol=path keyword expressions provide a mechanism for replacing path symbols
within the furnish kernel. The token symbol is replaced with a path symbol found in the furnish
kernel and the path token is replaced by a file path to be substituted for the symbol in the kernel
file. Multiple symbol-path pairs may be specified in this manner.

The /verbose keyword results in additional status messages list the number of kernel files
found. The /debug keyword controls the behavior of the routine when an error occurs and is not
normally required.

11.5.2 Kernel Downloading (inms_get_spice_kernels)

This routine reads and parses a SPICE text kernel containing a list of kernel files to be loaded.
Normally, this file is an argument to the furnish routine in the SPICE library. Any kernel files that
are absent from the user’s local machine are obtained from the NAIF ftp server. The furnish
kernel specifies the path to the kernels on the user’s local machine unless over-ridden by symbol-
path pairs. The location of the files on the NAIF server is built into the routine. If the target
directory on the local machine does not exist, it is created. Once the ftp transfers are complete, the
routine re-checks the local machine for the existence of the kernel files and reports those files that
were not successfully transferred.

The syntax for this routine is

\texttt{inms\_get\_spice\_kernels, sFile \{,/verbose\}, \{,/debug\},
\{,symbol=path\} \{,symbol=path\}…}

The token sFile is replaced with the full path to a furnish text kernel. The symbol=path
keyword expressions provide a mechanism for replacing path symbols within the furnish kernel.
The token symbol is replaced with a path symbol found in the furnish kernel and the path
token is replaced by a file path to be substituted for the symbol in the kernel file. Multiple
symbol-path pairs may be specified in this manner.

The /verbose keyword results in additional status messages. The /debug keyword controls the
behavior of the routine when an error occurs and is not normally required.

A sample output is shown in Figure 28, where the furnish kernel text kernel is the one used in the
previous example. In this case a number of directories were created and all but four files were
downloaded. This example illustrates one shortcoming of the routine. ION uses symbolic links to
a few kernel files that are frequently updated so that as new versions of the files are available, the
entries in the furnish text kernel do not need to be changed. The four files listed at the end of the
example as absent are examples. At the time of this writing, the most current leap second kernel
was naif0008.tls. To overcome this restriction, one can edit the furnish text kernel, replacing the
generic file names naif.tlk, pck.tpc, cas.tsc and cas.tf with the names of the most recent versions of
the leap-second, planetary constants, spacecraft clock and instrument frames kernels, respectively.
12. **Installation Issues**

12.1 **Data Distribution**

Level 1A data may be obtained from the INMS Operations Network (ION). Using a web browser visit https://ion.space.swri.edu/ion/index.jsp. Select the “Analysis” tab and you will be presented with a calendar. Select the year, month and day of interest to get a list of files. The list includes both browse product PNG files and a compressed archive of the data files. You can examine the browse product by clicking the name. To download the data file, click the archive file name. Once the file has been downloaded, move it to a convenient location and expand it using `gunzip` and `tar` or the equivalent on your system. The result will be a directory named `yyyyddd_L1A_vv` for L1A data or `yyyyddd_HKG_vv` for housekeeping data, containing the data.

12.2 **Data Location**

The analysis library works on level 1A archive files and requires some auxiliary files containing file format information and calibration data. These files may be located in any directory. However, for ease of use, it is recommended that the Level 1A archive files and the file format file, `L1A_STRUCT_vv.FMT` be located in the same directory. As noted in the `inms_get_data` description, the data files may be organized by year, by year and day, or undifferentiated.

The calibration summary spreadsheet also may be located in any directory. The reader, `inms_read_cal`, accepts the name of the file as an argument and will display a file selection dialog if the specified file is not readable.

12.3 **Platform Specific Information**

This software has been tested on the Apple MAC OSX (version 10.3 and 10.4) and on Windows XP Professional platforms. On the Windows platform the color bars which display discrete data items, such as the ion source on the Mass-Time Spectra appear as curved line segments on the screen and in PNG files, they are displayed correctly in postscript files.

12.4 **Code Distribution, Installation and Support**

The INMS analysis library is distributed as a gzip’ed tar archive file. The archive file contains all

```plaintext
inms_get_spice_kernels,’/spiceKernels/kernels.txt’,$
gen=’/spiceKernels/generic’,cas=’/spiceKernels/cassini’
% INMS_GET_SPICE_KERNELS: 16 of 16 kernel files listed in text kernel file
 ’/spiceKernels/kernels.txt’ are missing
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/generic/lsk”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/generic/pck”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/cassini/sclk”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/cassini/fk”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/cassini/spk”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/cassini/ik”
% INMS_GET_SPICE_KERNELS: creating directory ”/spiceKernels/cassini/ck”
% INMS_GET_SPICE_KERNELS: 4 of 16 kernel files listed in text kernel file
 ’/spiceKernels.txt’ were not successfully obtained from
NAIF
Files Absent: $GEN/lsk/naif.tls,$GEN/pck/pck.tpc,$CAS/sclk/cas.tsc,$CAS/fk/cas.tf
You must manually obtain the most recent version number of each file
% INMS_GET_SPICE_KERNELS: Processing Complete
```

Figure 28, Example `inms_get_spice_kernels` Output
of the IDL routines required to perform the functions described in this document. Also included are the default files required by the \texttt{inms-get-data} and \texttt{inms-read-cal} routines. The archive file may be downloaded from the ION website from the “analysis -> data sets -> File System. The example plot find INMS\_TEST\_RESULTS.PDF may also be found at that location.

Installation consists of expanding the archive. Copy the archive to the directory in which you wish to place the source code routines. Next, make that directory the working directory and expand the archive. The commands required on UNIX based systems are:

\begin{verbatim}
cd the/source/directory
gunzip yyyyddd-inmsAnalysis.tgz
tar -xvf yyyyddd-inmsAnalysis.tar
\end{verbatim}

On Mac OSX systems, once the archive is copied to the desired folder, you can use Stuffit Expander to unpack the source code.

Once you’ve unpacked the archive you can verify the installation by using the program \texttt{inms-test} to processes data from the T19 Titan encounter (2006-282T17:30). Download the data for that date, place it in a convenient location, which need not be the same as the source code, and expand it. Once that is done, invoke IDL and type the command

\begin{verbatim}
IDL> .run inms_test
\end{verbatim}

A dialog should appear to permit the selection of the TA data file and housekeeping file, then a series of plots should appear on your screen. Compare these to those in the file INMS\_TEST\_RESULTS.PDF or the figures in this document. They should be very similar.

You might get the message “% Error opening file. file inms_test”. This indicates that the directory containing the INMS analysis package is not in the IDL path. This can be confirmed by inspecting the output of the command

\begin{verbatim}
IDL> print, !PATH
\end{verbatim}

There are several ways to set the IDL path, including environment variables, executable IDL statements, or setting preferences in the IDL development environment (IDLDE). Check the IDL documentation for the method appropriate in your situation.

The library is supplied “as-is”. Please direct questions or comments by email to david.gell@swri.org. Continuing development is planned, but other INMS activities will be at a higher priority.
A. Contents of INMS Analysis Library

The analysis library contains a number of routines in addition to the routines described in the body of this document. These routines are listed, with a brief statement of their purpose in Table 5. Each routine in the library contains a documentary prolog specifying the use of the routine and defining arguments and parameters. The prologs are also available in the on-line help file inms_analysis_help.html, which may be viewed with any web browser.

The library distribution also contains a number of auxiliary files. These files and a brief statement of their purpose are listed in Table 6.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>inms_add_aux_axes</td>
<td>adds auxiliary axes to a plot</td>
</tr>
<tr>
<td>inms_auxiliary_value</td>
<td>computes lat, lon, ram angle, from data</td>
</tr>
<tr>
<td>inms_build_locator</td>
<td>creates a selector expression for use in a where statement</td>
</tr>
<tr>
<td>inms_chebyshev</td>
<td>evaluate chebyshev polynomials by recursion</td>
</tr>
<tr>
<td>inms_close_windows</td>
<td>closes all open windows</td>
</tr>
<tr>
<td>inms_compare_utc</td>
<td>compares two UTC time structures</td>
</tr>
<tr>
<td>inms_compute_density</td>
<td>Simplified density calculation (deprecated)</td>
</tr>
<tr>
<td>inms_compute_mean_spectra</td>
<td>computes the mean of a collection of spectra</td>
</tr>
<tr>
<td>inms_compute_summed_spectra</td>
<td>sums spectra contained in a collection</td>
</tr>
<tr>
<td>inms_create_l1a_template</td>
<td>creates template used in reading L1A files (Obsolete)</td>
</tr>
<tr>
<td>inms_deconvolve</td>
<td>extracts species abundances by mass deconvolution</td>
</tr>
<tr>
<td>inms_define_xSpecRec.pro</td>
<td>defines the contents of a spectra record</td>
</tr>
<tr>
<td>inms_doy2date</td>
<td>converts a day of year numeric date to a calendar date string</td>
</tr>
<tr>
<td>inms_doy2julian</td>
<td>converts calendar strings to Julian dates</td>
</tr>
<tr>
<td>inms_doy2utc</td>
<td>converts calendar strings to UTC time structure</td>
</tr>
<tr>
<td>inms_dump_structure</td>
<td>Creates a CSV file filled with data from a structure</td>
</tr>
<tr>
<td>inms_file_format</td>
<td>obtains the file format for a specified file type</td>
</tr>
<tr>
<td>inms_format_time</td>
<td>converts time of day in milliseconds to hours, minutes, seconds</td>
</tr>
<tr>
<td>inms_format_time_tick</td>
<td>a callback function to format time axis tick labels</td>
</tr>
<tr>
<td>inms_get_data</td>
<td>reads a Level 1A INMS data file</td>
</tr>
<tr>
<td>inms_get_series</td>
<td>extracts a subset of data from a Level 1A data structure</td>
</tr>
<tr>
<td>inms_get_spectra</td>
<td>Extracts one or more spectra from Level 1A data</td>
</tr>
<tr>
<td>inms_get_spice_kernels</td>
<td>reads a spice furnish text kernel and gets missing files</td>
</tr>
<tr>
<td>inms_grid_spectra</td>
<td>interpolates mass spectra to uniform time grid</td>
</tr>
<tr>
<td>inms_hkg_labels</td>
<td>provides list of axis labels for housekeeping data</td>
</tr>
<tr>
<td>Name</td>
<td>Purpose</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inms_idl_type</td>
<td>determines IDL type based on PDS type name and value range</td>
</tr>
<tr>
<td>inms_init_ss_position</td>
<td>initializes the spice toolkit for use by inms_ss_position</td>
</tr>
<tr>
<td>inms_is_publication</td>
<td>returns publication quality output flag</td>
</tr>
<tr>
<td>inms_julian_utc</td>
<td>converts a Julian date to a utc structure</td>
</tr>
<tr>
<td>inms_kernel_list</td>
<td>produces a list of SPICE kernels present or missing</td>
</tr>
<tr>
<td>inms_L1A_files_read</td>
<td>provides list of Level 1A files last read by inms_get_data</td>
</tr>
<tr>
<td>inms_L1a_labels</td>
<td>provides list of axis labels for L1A data</td>
</tr>
<tr>
<td>inms_label_ticks</td>
<td>produces formatted time strings for use as tick labels</td>
</tr>
<tr>
<td>inms_labels</td>
<td>provides list of axis labels for a data structure</td>
</tr>
<tr>
<td>inms_list_cal_species</td>
<td>lists species contained in calibration data structure</td>
</tr>
<tr>
<td>inms_list_files</td>
<td>annotates a plot with a list of names</td>
</tr>
<tr>
<td>inms_make_data_structure</td>
<td>Creates a data structure based on an input template</td>
</tr>
<tr>
<td>inms_make_pds_label</td>
<td>make a pds label file</td>
</tr>
<tr>
<td>inms_make_profiles</td>
<td>computes density profiles by deconvolution</td>
</tr>
<tr>
<td>inms_make_scalar</td>
<td>returns an initialized scalar of a specified type</td>
</tr>
<tr>
<td>inms_make_window</td>
<td>creates a new X window for graphics</td>
</tr>
<tr>
<td>inms_mdy2doy</td>
<td>converts a date from YYYYMMDD to YYYYDDD</td>
</tr>
<tr>
<td>inms_neat_ticks</td>
<td>determines values for tick labeling plot keywords</td>
</tr>
<tr>
<td>inms_parse_file_name</td>
<td>extracts constituent fields from INMS archive file names</td>
</tr>
<tr>
<td>inms_parse_L1A_name</td>
<td>extracts constituent fields from L1A file names</td>
</tr>
<tr>
<td>inms_parse_time</td>
<td>extracts fields from date/time strings</td>
</tr>
<tr>
<td>inms_plot_cal_ptn</td>
<td>plots calibration cracking patterns curves</td>
</tr>
<tr>
<td>inms_plot_cal_sens</td>
<td>plots sensitivities as function of species</td>
</tr>
<tr>
<td>inms_plot_compare</td>
<td>plots count rate time series from Level 1A data and spectra</td>
</tr>
<tr>
<td>inms_plot_density_profiles</td>
<td>plots one or more density profiles with altitude</td>
</tr>
<tr>
<td>inms_plot_geom</td>
<td>plots geometric quantities as function of time</td>
</tr>
<tr>
<td>inms_plot_histogram</td>
<td>plots a mass spectra histogram</td>
</tr>
<tr>
<td>inms_plot_hkg</td>
<td>plots one or more housekeeping data items</td>
</tr>
<tr>
<td>Name</td>
<td>Purpose</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inms_plot_mass_history</td>
<td>plots count rate time series from spectra arrays</td>
</tr>
<tr>
<td>inms_plot_mass_profile</td>
<td>plots signal altitude profiles from spectra</td>
</tr>
<tr>
<td>inms_plot_mt_line</td>
<td>plots count rate time series from Level 1A data</td>
</tr>
<tr>
<td>inms_plot_mt_spectra</td>
<td>plots mass time spectra from Level 1A data</td>
</tr>
<tr>
<td>inms_plot_series</td>
<td>plots one or more housekeeping data items</td>
</tr>
<tr>
<td>inms_plot_stacked_spectra</td>
<td>plots a collection of spectra as a color coded mass/time spectra</td>
</tr>
<tr>
<td>inms_plot_state</td>
<td>Plots mass, cycle, or sequence table transitions</td>
</tr>
<tr>
<td>inms_post_message</td>
<td>Posts a message as a dialog if possible</td>
</tr>
<tr>
<td>inms_post_status</td>
<td>posts a message to the status widget</td>
</tr>
<tr>
<td>inms_prepare_plot</td>
<td>initializes plotting for X, PS or PNG</td>
</tr>
<tr>
<td>inms_put_annotations</td>
<td>writes annotations on plot in specified location</td>
</tr>
<tr>
<td>inms_query_11a</td>
<td>Extracts configuration data from a Level 1A data structure</td>
</tr>
<tr>
<td>inms_ram_angle</td>
<td>computes the ram angle (Depreciated use inms_auxiliary_value)</td>
</tr>
<tr>
<td>inms_ram_coefficient</td>
<td>computes density enhancement in closed source due to velocity</td>
</tr>
<tr>
<td>inms_read_cal</td>
<td>reads a calibration data spreadsheet</td>
</tr>
<tr>
<td>inms_read_cal_ion</td>
<td>reads the ion calibration data</td>
</tr>
<tr>
<td>inms_read_fmt_file</td>
<td>Reads PDS compliant structure file</td>
</tr>
<tr>
<td>inms_read_jcamp</td>
<td>reads mass spectrometer data in the JCAMP/DX format</td>
</tr>
<tr>
<td>inms_read_label</td>
<td>reads a PDS compliant label</td>
</tr>
<tr>
<td>inms_read_pds_spreadsheet</td>
<td>reads a PDS compliant spreadsheet file</td>
</tr>
<tr>
<td>inms_read_scvolts</td>
<td>Reads the spacecraft potential files</td>
</tr>
<tr>
<td>inms_reduce_qb_scan</td>
<td>reduce quad bias scan data</td>
</tr>
<tr>
<td>inms_reduce_ql_scan</td>
<td>performs analysis of quad lens scan</td>
</tr>
<tr>
<td>inms_remove_quotes</td>
<td>removes enclosing quotation marks</td>
</tr>
<tr>
<td>inms_restore_data</td>
<td>restores data previously saved by inms_save_data</td>
</tr>
<tr>
<td>inms_saturn_latitude</td>
<td>computes the latitude w.r.t Saturn (Depreciated use inms_auxiliary_value)</td>
</tr>
<tr>
<td>inms_saturn_wlongitude</td>
<td>computes the west longitude w.r.t. Saturn (Depreciated use inms_auxiliary_value)</td>
</tr>
<tr>
<td>inms_save_data</td>
<td>Prepares a save file containing inms data</td>
</tr>
<tr>
<td>inms_select_cal</td>
<td>selects a calibration record from the calibration structure</td>
</tr>
<tr>
<td>Name</td>
<td>Purpose</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inms_set_charsize</td>
<td>returns scaled character size</td>
</tr>
<tr>
<td>inms_spectra_aux_values</td>
<td>determines values of ancilliary data</td>
</tr>
<tr>
<td>inms_spectra_calculations</td>
<td>performs arithmetic with spectra</td>
</tr>
<tr>
<td>inms_ss_position</td>
<td>computes the position of the subsolar point (depreciated)</td>
</tr>
<tr>
<td>inms_status_widget</td>
<td>prepares a scrolling status window for use</td>
</tr>
<tr>
<td>inms_subtract_background</td>
<td>performs background removal</td>
</tr>
<tr>
<td>inms_svd_solve</td>
<td>Solves aX=b using Singular Value Decomposition</td>
</tr>
<tr>
<td>inms_tabulate_spectra</td>
<td>Writes the contents of the spectra record</td>
</tr>
<tr>
<td>inms_test</td>
<td>executes a sequence of routines to verify installation</td>
</tr>
<tr>
<td>inms_utc2date</td>
<td>converts a UTC time to a calendar date string</td>
</tr>
<tr>
<td>inms_utc_increment</td>
<td>increments a UTC time by a specified amount</td>
</tr>
<tr>
<td>inms_validate_cal_data</td>
<td>confirms a structure is a calibration data structure</td>
</tr>
<tr>
<td>inms_validate_hkg_data</td>
<td>confirms validity of housekeeping data structures</td>
</tr>
<tr>
<td>inms_validate_l1a_data</td>
<td>confirms a structure is a valid Level 1A data structure</td>
</tr>
<tr>
<td>inms_validate_spectra_data</td>
<td>confirms a structure is a spectra data structure</td>
</tr>
<tr>
<td>inms_validate_time</td>
<td>confirms that a string is a properly formatted date/time</td>
</tr>
<tr>
<td>inms_weighted_mean</td>
<td>computes the estimate of the mean &amp; std deviation</td>
</tr>
<tr>
<td>inms_write_image</td>
<td>transfers an image from a pixel map and saves to a file</td>
</tr>
<tr>
<td>l_getdim</td>
<td>transforms the results of where() to dimensions</td>
</tr>
<tr>
<td>sprl_color_triad</td>
<td>returns the a 3 element byte array corresponding to the named color</td>
</tr>
<tr>
<td>sprl_colorplot</td>
<td>plots a function of two variables in color</td>
</tr>
<tr>
<td>sprl_create_list</td>
<td>forms a list of unique elements in an array</td>
</tr>
<tr>
<td>sprl_cvt_jdate_mdy</td>
<td>converts the specified Julian date to the month, day and year</td>
</tr>
<tr>
<td>sprl_cvt_jdate_odate</td>
<td>converts a Julian day number to an ordinal date</td>
</tr>
<tr>
<td>sprl_cvt_jtime_tod</td>
<td>converts the fractional part of a Julian date to time</td>
</tr>
<tr>
<td>sprl_cvt_odate_jdate</td>
<td>converts an ordinal date/time to a Julian date</td>
</tr>
<tr>
<td>sprl_date_plotted</td>
<td>annotates a plot with the current date</td>
</tr>
<tr>
<td>sprl_discrete_color_list.pro</td>
<td>Definitions of discrete colors</td>
</tr>
<tr>
<td>sprl_draw_flag</td>
<td>draws a flag at a specified location</td>
</tr>
<tr>
<td>sprl_draw_scale</td>
<td>draws a color bar scale at the specified location</td>
</tr>
<tr>
<td>sprl_error_plot</td>
<td>Plots data points with x &amp; y error bars</td>
</tr>
<tr>
<td>sprl_find_color</td>
<td>returns the numerical value corresponding to the named</td>
</tr>
</tbody>
</table>
### Table 5, Contents of INMS Analysis Library

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>sprl_is_decomposed</td>
<td>indicates if a true color display is in use</td>
</tr>
<tr>
<td>sprl_isnumeric</td>
<td>Tests if a string represents a valid number</td>
</tr>
<tr>
<td>sprl_load_blue_sequential</td>
<td>loads the blue sequential color table</td>
</tr>
<tr>
<td>sprl_load_catagorical_colors</td>
<td>loads the discrete colors defined in SPRL_DISCRETE_COLOR_LIST.PRO</td>
</tr>
<tr>
<td>sprl_load_colors</td>
<td>interface to set of color table loading routines</td>
</tr>
<tr>
<td>sprl_load_divergent_colors</td>
<td>loads a divergent color table (blue)</td>
</tr>
<tr>
<td>sprl_load_red_sequential</td>
<td>loads the red sequential color table</td>
</tr>
<tr>
<td>sprl_load_spectrum_sequential</td>
<td>loads the spectrum sequential color table</td>
</tr>
<tr>
<td>sprl_plot_sym</td>
<td>generated user defined plot symbols</td>
</tr>
</tbody>
</table>

### Table 6, Auxiliary Files Included With INMS Analysis Library

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalysisGuide.pfd</td>
<td>This document</td>
</tr>
<tr>
<td>inms_analysis_help.html</td>
<td>On-line help file, view with any browser</td>
</tr>
<tr>
<td>AAAReleaseNotes.txt</td>
<td>Release note</td>
</tr>
<tr>
<td>2004001_CAL_05.CSV</td>
<td>Default thermal gas calibration summary</td>
</tr>
<tr>
<td>2004001_CAL_05.LBL</td>
<td>Label file for calibration summary</td>
</tr>
<tr>
<td>CAL_SUMMARY_02.FMT</td>
<td>Calibration summary file format</td>
</tr>
<tr>
<td>HKG_STRUCT_01.FMT</td>
<td>Default housekeeping data file format</td>
</tr>
<tr>
<td>L1A_STRUCT_05.FMT</td>
<td>Default level 1A data file format</td>
</tr>
<tr>
<td>IIS_frame_definitions.fk</td>
<td>Frames text kernel defining Ionospheric Interaction System frame</td>
</tr>
<tr>
<td>HKG_TEMPLATE_01.LBL</td>
<td>Housekeeping file label template for use with inms_make_pds_label</td>
</tr>
<tr>
<td>L1A_TEMPLATE_05.LBL</td>
<td>Level 1A file label template for use with inms_make_pds_label</td>
</tr>
</tbody>
</table>
B. **Release Notes**

THE INMS ANALYSIS LIBRARY IS SUPPLIED “AS-IS”
WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
Version 2008004

This release revises the ion abundance calculations. The sensitivity data is read from a PDS compliant spreadsheet. The OSI mode throughput correction for QL3 mistuning is updated to include the effect of spacecraft charging.

This release also provides support for housekeeping trend file plotting. Trend files are customized housekeeping data files that span multiple days produced by ION.

The routines that read PDS labels have been enhanced. They can now read the structure information directly from the label, when there is no structure file pointer defined.

B.1 **New Routines**

- `inms_read_pds_spreadsheet` Reads a PDS compliant spreadsheet, using information from its label and structure files.
- `inms_make_data_structure` Creates a data structure to hold spreadsheet records. Supports vector fields specified by ITEMS keyword.
- `inms_plot_label` Generic form of `inms_l1A_labels`

B.2 **Changed Routines:**

- `inms_ion_sensitivity` Reads QL3 tuning information from a spreadsheet file. Modified to support revised `inms_ion_sensitivity` routine. Angle correction disabled by default.
- `inms_make_ion_spectra` Inverted the output.
- `inms_read_cal_ion` Revised to read data needed to compute QL3 tuning effect from spreadsheet.
- `inms_parse_time` Time delimiter may be either upper or lower case.
- `inms_plot_stacked_spectra` Restored ram angle plot functionality.
- `inms_read_fmt_file` Reads format from label file as well as format file. Recognizes the ITEMS keyword.
- `inms_test` Updated version, added an additional ion density plot example.
- `inms_validate_time` Increased permissiveness, accepts either t or T delimiter.
- `inms_spectra_aux_values` Corrects units of nAngle field, now degrees.
- `inms_spectra_calc` Corrects variable names.
- `inms_file_format` Supports trend data.
- `inms_reduce_qb_scan` Replaces amoeba search with curvefit when fitting data to sigmoid curve. Skips fitting when signal is low.
- `inms_deconvolve` Insure correct units of ram angle passed to.
- `inms_ram_coefficient` Delete isotopic ratio calculation and the /no14 and /no28 switches.
- `inms_auxiliary_value` Fix non-conforming matrix shapes.
- `inms_get_data` Modified to read housekeeping trend files.
- `inms_post_message` Improved message format.
- `inms_make_pds_label` Correct array reference syntax.
- `inms_reduce_qi_scan` Continued development.
- `sprl_find_color` Correct error when the NULL plotting device is active.
C. **Depreciated Routines**

C.1 **inms_plot_cal**

The routine to display the cracking patterns from the calibration summary file in the previous release, `inms_plot_cal`, has been replaced in release 2006338 with the routine `inms_plot_cal_sens` (see 7.1.4 above). This change was made to avoid ambiguous procedure names. Please use the new routine for new code.

C.2 **inms_compute_density, inms_density_file**

The procedure `inms_compute_density` has been removed from the library. It performed a simplistic algorithm for the determination of N\textsubscript{2}, CH\textsubscript{4} and Ar densities using compiled in constants for branching ratios and sensitivities. In its place, one should use `inms_deconvolve` or other code specifically developed to retrieve densities under the specific data set constraints.

The routine `inms_density_file` is a driver for `inms_compute_density`, which may be replaced by `inms_make_profiles`, a driver for `inms_deconvolve`.

C.3 **inms_ss_postions, inms_init_ss_position**

The function `inms_ss_position` computes the sub-solar latitude and west longitude with respect to a target body coordinate frame. The current version of the L1A data includes the values of these parameters that should be used instead of this routine. The `inms_init_ss_position` initializes the spice system to perform the computations and is therefore also unnecessary.

C.4 **inms_saturn_latitude, inms_saturn_wlongitude, inms_ram_angle**

The function of these routines have been replaced and augmented by `inms_auxiliary_value`, which should be used in new work.

C.5 **inms_create_l1a_template**

This routine, which was called by `inms_get_data`, is no longer required. It was called to further process the file format structure returned by `inms_read_fmt_file`. That later routine has been modified to perform the action of the former, which was merely to assign IDL variable type codes to the data items defined by the format file.