RINEX Toolbox

For MATLAB®

User's Guide

Version 6.00

Constell, Inc.

RINEX Toolbox User's Guide, Version 6.00
November 2003

Constell, Inc.
P.O. Box 433
Philomont, VA 20131

(540) 338-0289 (voice)
(540) 338-0293 (fax)

http://www.constell.org (Web)

support@constell.org (e-mail information and support)

Table of Contents

Installation Instructions

The RINEX Toolbox is normally contained on a PC-based 3.5”, 1.44Mb floppy disk. Other formats such as an e-mail delivery are also available.

For PC users, the Toolbox is distributed as a self-extracting zip file. To install, double click on the self-extracting zip file named rinex6.zip. When prompted for an “unzip” directory, use the same root as the Constellation Toolbox (….\Constell for a typical Constellation Toolbox installation shown below) and add a subdirectory called RINEX. The resulting file structure should be:

...\constell	- contains Constellation Toolbox toolbox functions (previously installed)
...\constell\demo	- contains the Constellation Toolbox GUI-based demo (previously installed)
...\constell\tutorial	- contains the Constellation Toolbox tutorial code (previously installed)
...\constell\RINEX	- contains the RINEX Toolbox code (this installation)

Unix users will be provided a floppy disk with the Toolbox stored as a tar file. The directory structure described above should be created and the files copied into the appropriate directory from the installation disk.

Add the toolbox directory to your MATLAB path

The toolbox is accessible from all directories when on your MATLAB path.

Use the graphical path editing tool available in the base MATLAB environment to add the Constell\RINEX directory (following the example above) to your path. We recommend you add this to the back or end of your MATLAB path.
OR
Edit the pathdef.m file. For normal MATLAB installations, this file is found in the ..\matlabxx\toolbox\local directory.

Your RINEX Toolbox installation is now complete.

Quick Start

Try out the RINEX Toolbox using the following tutorials.

<u>In the ...\constell\RINEX directory:</u>

ex_rd_o.m – an example of reading and reformatting RINEX observation data

ex_wr_o.m – an example that generates simulated GPS observations with the Constellation Toolbox and writes RINEX observation data

ex_rdandwr_rx.m – an example of reading and writing RINEX data

Included in the installation package are sample RINEX files (base121A.00o and testrnx.00o) to support the example functions.

Introduction

The RINEX Toolbox for MATLAB® is a small collection of .m utilities that provide support to read and write RINEX-2 formatted GPS data files. The RINEX Toolbox requires the Constellation Toolbox for some support functions and formats data in a consistent manner with the Constellation Toolbox. This toolbox works with MATLAB version 5.3 or greater.

The Receiver Independent Exchange Format (RINEX) format is a standardized way to exchange GPS data. The Astronomical Institute of the University of Berne developed the format for a 1989 GPS campaign. The RINEX format supports four types of files:

1. Observation Data File
2. Navigation Message File
3. Meteorological Data File
4. GLONASS Navigation Message File

RINEX recommends a file naming convention as shown in Table 1.

Table 1 RINEX File Naming Convention

File name format	File name components	Description
ssssdddf.yyt	ssss	4-character station name designator
	ddd	day of the year of first record
	f	file sequence number within day. 0: file contains all the existing data of the current day
	yy	year
	t	file type: O: Observation file N: Navigation file M: Meteorological data file G: GLONASS Navigation file

For example, a file named base121A.00o (included in the RINEX Toolbox distribution) is described in Table 2.

Table 2 RINEX File Naming Example

File name format	File name components	Description
base121A.00o	ssss - base	4-character station name designator
	ddd - 121	day of the year of first record (April 15)
	f - A	file sequence number within day. A is the first in a sequence.
	yy - 00	year
	t - O	file type: Observation file

The RINEX Toolbox supports only the first two files, the observation data and navigation message files. For a complete description of the RINEX-2 format, see the file RINEX-2.txt included in the RINEX Toolbox installation.

Common Time, Position, and Other Vector Data

The .m files share a common input and output data format. The output of one function can easily serve as the input to another function. The files are highly vectorized, so most input and output data are in vector form. Therefore, a common vector format is defined. The standard time vector used here is gps_time, and has a two-column, n-row format for n time intervals:

$$gps\_time = \begin{bmatrix} week(1) & \sec(1) & rollover\_flag(1) \\ week(2) & \sec(2) & rollover\_flag(2) \\ week(3) & \sec(3) & rollover\_flag(3) \\ \ldots & \ldots & \ldots \\ week(n) & \sec(n) & rollover\_flag(n) \end{bmatrix}$$

where: week = GPS weeks since the GPS week rollover on Aug. 22, 1999.
Or since Jan 6, 1980 for times prior to the rollover.
sec = GPS seconds since midnight of the previous Saturday.
rollover_flag = Optional flag with default value of 1 indicating times since Aug. 22, 1999.
For times prior to Aug 22, 1999, include a rollover_flag of 0.

A single time point is defined by specifying both the week and second. Time represented in this manner keeps the numerical value of the seconds manageable in magnitude, and is a common representation in the GPS community. Time can also be represented in a linear manner with total_seconds (total seconds since Jan. 6, 1980 or since Aug. 22, 1999 depending on your value of *rollover_flag*):

total_seconds = weeks*86400*7 + sec.

Utilities are provided to convert between the 2-column or 3-column gps_time format and the 1-column total_seconds format. Time is occasionally needed in the 6 element UTC format of [year month day hour minute second], such as when entering wall clock time or computing Greenwich Sidereal Hour angle. Utilities are provided to convert between GPS time and UTC time. There is an integer second offset between GPS time and UTC time, commonly referred to as a leap second, which is used in this conversion. A data file, **leapsecs.dat**, contains the UTC time for each incremental leap second that has been added since the beginning of GPS time. This file must be updated manually or downloaded from the Constell web page (http://www.constell.org) when another leap second is added.

Positions are represented as row vectors in this toolbox to allow for convenient manipulation of large quantities of data. A single position vector is:

$$\vec{x} = [x, y, z].$$

Both a time vector and a corresponding position vector are needed to represent a position vector, x(t), over a time interval of n points:

$$gps\_time = \begin{bmatrix} week(1) & \sec(1) & rollover\_flag(1) \\ week(2) & \sec(2) & rollover\_flag(2) \\ week(3) & \sec(3) & rollover\_flag(3) \\ \ldots & \ldots & \ldots \\ week(n) & \sec(n) & rollover\_flag(n) \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x(1) & y(1) & z(1) \\ x(2) & y(2) & z(2) \\ x(3) & y(3) & z(3) \\ \ldots & \ldots & \ldots \\ x(n) & y(n) & z(n) \end{bmatrix}$$

Other vector quantities (e.g., velocities and angles) are represented in an identical manner.

Vectorization of Code

The Constellation Toolbox relies heavily on vectorization of the MATLAB code for speed optimization. Due to the nature of the MATLAB code design, vectorized code can run literally thousands of times faster than the same code implemented with a loop. Therefore, explicit loops are almost completely avoided.

For a specific example, consider one of the fundamental tasks of computing line-of-sight between two position vectors. To make this a real constellation type example, we will use a constellation of satellites and the GPS constellation. The orbits for both constellations have already been computed before this section of code. Here is the MATLAB code to do this the non-vectorized way. Notice the nested loops which slow MATLAB processing and make the code harder to read.

```
% Compute LOS by looping in time, GPS satellites, and constellation satellites
for i = 1:num_times
  for j = 1:num_gps
    % Find the index for the GPS satellite at this time and satellite number
    Ig = find(time_gps == times_unique(i) & prng == gps_sat_nums(j));

    % Find all of the user supplied constellation satellites at this time
    Ic = find(times_constellation == times _unique(i));

    % Loop over all of the constellation satellites and compute a LOS
    for k = 1:length(Ic)
      if ~isempty(Ig)
        this_los = xg(Ig,:) - xc(Ic(k),:);

            % Add this LOS, time and sat numbers to the outputs.  This is one of the
            % best way so concatenate data from a loop.  However, it's also very
            % inefficient.
        los_vect = [los_vect; this_los];
        los_t = [los_t; tg(Ig)];
        c_num = [c_num; prnc(Ic(k))];
        g_num = [g_num; prng(Ig)];
      end % if ~isempty(Ig)
    end % for k = 1:length(Ic)
  end % for j = 1:num_gps
end % for i = 1:num_times
```

Now for the Constellation Toolbox vectorized way. Notice how much easier the calling structure is to read and the way the code is simplified.

```
[time_los, los_vect] = los(time_const,[sat_num x_cconst],time_gps,[prn x_gps]);
```

That's it. One line of code. Already vectorized and fast. The output line-of-sight vectors are the same for both function (e.g. los_vect is the same for both sets of code). The additional advantage of this approach is that GPS time vectors are used in the inputs and outputs. In the non-vectorized code, the nx2 GPS time vectors had to be converted to a 1-dimensional time vector to use the MATLAB FIND function effectively. The FIND function will do a two dimensional search, but the process is very slow compared to a 1-D search.

How much faster is the vectorized way? This code was run for a small set of test cases on a Pentium 200 laptop computer with 96 Mb of RAM. The results are shown here. This code is provided in the tutorial directory in function ex_vect.m. Try a few cases, and verify the performance.

# User Sats	# GPS Sats	# Times	# LOS	Non-Vectorized (sec)	Vectorized (sec)
2	27	201	10854	242.1	1.54
2	27	601	32454	2189.5	4.8
20	27	21	11340	208.82	0.94
20	27	51	27540	1268.5	2.3
20	27	101	54540	5012.7	4.12

The added speed of vectorization comes with a price and the price is memory. The speed is increased by doing all of the computations at a time. There are situations where large simulations cannot be run vectorized as shown above. The way to attack very large simulations is to break them up into smaller time chunks. This preserves much of the vectorization while allowing the problem to fit into the specific computer memory. An example of this technique is given the tutorial code named exdgpsac.m.

To tell if your machine is running out of memory, listen for it caching to the hard disk during a run. Another test is to time a small test case and a larger test case. If the execution time does not scale linearly, then the computer is probably caching to the hard drive. If this happens you will actually experience a performance decrease. However, this only occurs for very large simulations such as computing LOS once a second from constellation to constellation. Specific performance is also machine dependent. Check with your system administrator if you have memory allocation problems.

Notes About Outputs from Vectorized Functions

Because the functions are vectorized, they return data in matrix format. If you're unfamiliar with MATLAB matrices, this is a great way to learn how to harness the true computational power of MATLAB . All of the Constellation Toolbox outputs are designed to be two dimensional matrices. Although MATLAB supports multi-dimensional matrices, the 2-D kind are in general easier to understand and utilize. Some of the functions, such as LOS, use three dimensional matrices internally to achieve the vectorization. However, care has been taken to isolate the multi-dimensional arrays at the function level so the user can concentrate on solving problems instead of interpreting multi-dimensional arrays.

Throughout the Constellation Toolbox, inputs and outputs will be dimensioned nx2, nx3, or some other array size. The nx2 (or nx3) convention means that the row dimension can be variable in length, but the column dimension must be 2 or 3. A good example of this is GPS time, which is defined in weeks, seconds, and an optional rollover_flag. This is always input/output as an nx2 or nx3, where there can be n-number of times with each time stored in a row. GPS weeks are stored in the first column, GPS seconds are stored in the second column, and an optional rollover_flag may or may not be included in the third column.

Some variables are required to be a fixed dimension, for example 1x3. This means that the data should be in 1 row with 3 columns. A simple way to check the dimensions of matrices and variables within MATLAB is with the SIZE command. All of the Constellation Toolbox functions will use the modular error checking to verify that the variables given to a function have input dimensions that are correct.

When multiple matrix variables are output from a function (such as LOS), the data that goes together will be in the same row. For example, the first two outputs of LOS are GPS time (nx2) and the line-of-sight vector (nx3). The output would look like the following …

```
t_los = [GPS_week(1) GPS_sec(1); …   los_vect = [los_x(1) los_y(1) los_z(1); …
         GPS_week(2) GPS_sec(2); …               los_x(2) los_y(2) los_z(2); …
         GPS_week(3) GPS_sec(3)]                 los_x(3) los_y(3) los_z(3)]
```

The vectorization in the Toolbox comes from using methods other than loops to achieve the looping functionality. For example, in the line-of-sight function mentioned above, the Toolbox LOS function is able to line up which constellation satellites, GPS satellites, and times are required for each computation without using loops. This is done through the use of much faster MATLAB functions such as UNIQUE, INTERSECT, and FIND in conjunction with multi-dimensional arrays and NaN. This is all part of the LOS function, but the output is simply a set of matrices with corresponding data in the same rows and the resulting indices that relate the input and the output variables.

Several routines return these matching indices. These indices are the key to making use of the resulting data. They relate the input data (for example azimuth and elevation data) to the output data. A good example of this usage is the function for computing which data are visible, called vis_data.m. vis_data.m takes a set of azimuth and elevation data (nx2) and returns an kx1 matrix with the indices corresponding to the rows of azimuth/elevation data that passed the masking test. This index is then used to obtain the data that has passed the masking test.

A 3-line set of sample MATLAB code is provided for clarification. In this case, the azimuth and elevation data are used for masking out data that is below the local horizon. The index to all of the data that has passed the masking test (above the horizon) is returned. This index is then used to obtain which line-of-sight vectors are above the horizon. In this example, I_pass is the variable with the indices passing the masking test.

```
mask = 0;          % minimum elevation mask of 0
[pass_az_el, I_pass] = vis_data(mask,[az el]);
los_above_horizon = los_vect(I_pass,:);
```

Now that the index to the data passing the masking test is available, any data that has corresponding rows can be edited and mapped, just like the line-of-sight vector above.   This is a simple way of bookkeeping the data with the indices instead of passing around more data than is required. The example functions provided in the tutorial make extensive use of this. A good place to start is with the tutorial function vis_e.m which computes GPS satellite visibility for a station fixed to the surface of the Earth.

General Toolbox Considerations

The GPS constellation uses the Earth-Centered-Earth-Fixed (ECEF) coordinate system. Therefore, the Constellation Toolbox functions generally operate in this system also. There are several toolbox utilities, however, that allow straightforward conversion from the ECEF to Earth-Centered-Inertial frame, and to a satellite local-level frame (and back to ECEF, of course).

Vectorization within the toolbox can require many large matrices when hundreds or thousands of data points are generated. Some liberties have been taken with generally accepted coding standards in order to keep the computer memory requirements as small as possible. The primary instance of this is when two variables are combined into a third. An example of standard programming practice is:

```
c = a+b
```

which maintains unique storage locations for all variables. If a and b are 10,000 elements long, however, and b is no longer needed, the following line is used:

```
b = a+b
```

thereby saving 10,000 storage units (up to 80,000 Kb, depending on the MATLAB version). Note that if b is passed into a function through a calling argument, the original value is not changed, since MATLAB calls by value, not by reference.

Format of Toolbox Files

Each file in the toolbox has an identical format. The file name is also the function name. The files are encapsulated, which means that all input and output data are passed through the calling and returning parameters. The only exception is a global debug flag. A sample file is shown below:

```
function [output data] = file_name(input data)

% [output data] = file_name(input data)
%
% A description of the function including usage, definitions of inputs and
% outputs
% See also …  (a list of related functions)
% Author and copyright information
% References
% A list of other functions called

%%%%% BEGIN VARIABLE CHECKING CODE %%%%%
global DEBUG_MODE

Various checks to see if the number and dimensions of the input variables are correct and some
bounds checking on the variables themselves. DEBUG_MODE defaults to 0 if not set, causing
incorrect inputs to trigger an error and the execution to stop. If DEBUG_MODE is set to anything
other than 0, a warning message is issued regarding the source of the problem, and execution
continues.
%%%%% END VARIABLE CHECKING CODE %%%%%

%%%%% BEGIN ALGORITHM CODE %%%%%
The actual algorithms to implement the functions
%%%%% END ALGORITHM CODE %%%%%

% end file_name
```

When the MATLAB command: **help file_name** is typed at the MATLAB prompt, the first set of comments is printed to the screen (the "% [output data] …" to the "% See also …" lines). The "% [output data] … " line appears on the screen without the leading % sign. This first line is included to allow easy and immediate execution of the function from the command prompt with input variables defined in the current workspace. This first line can be highlighted with a mouse, then copied and pasted at the location of the current MATLAB prompt. This copied line can be edited to use existing variable names and then immediately executed.

The MATLAB code for each file is split into two major sections: the ***variable checking section*** and the ***algorithm section***. Note that each section is delineated by five % signs for ready identification by a browser search function, e.g.,

```
%%%%% BEGIN VARIABLE CHECKING CODE %%%%%
%%%%% BEGIN ALGORITHM CODE %%%%%
```

The ***variable checking*** code typically checks for the correct number of inputs, that the dimensions of the inputs are correct and consistent, and that the variable conforms to the data type described for the function (e.g. if a variable is supposed to be a string, a check is performed to see if it is a string). All of the error checking is contained with the function **err_chk.m**.

An example of the error checking logic is provided here. If a function requires input vectors of gps_time and position, the number of input arguments should be two, the gps_time vector dimensions should be nx2 or nx3, and the position vector dimensions should be nx3. The GPS time vector would also be checked for sanity of the input GPS weeks and GPS seconds.

If an input error is detected, a warning message is printed to the screen and the function continues to execute. The exception to this case is if the DEBUG_FLAG is set to a non-zero value and an error is detected. When the DEBUG_FLAG is set, a warning will be issued but the function will continue to execute. Normally, an error condition will stop execution and report the error.

Function Reference

Examples and General Support Function Summary

contents	Display contents of the RINEX Toolbox.
ex_rd_rx	Example of reading and reformatting RINEX observation data.
ex_rdandwr_rx	Example of reading and writing RINEX data.
ex_wr_rx	Example that generates simulated GPS observations with the Constellation Toolbox and writes RINEX observation data.
refrinex	Reformats MATLAB vectors from the RINEX style (organized by time tag) to a more MATLAB friendly style (organized by data type). For example, in the RINEX style pseudorange and carrier phase measurements are combined into a single set based on time tag. This function separates them into separate vectors for ease of vectorizing other internal MATLAB functions and for easier plotting.

RINEX Reading Functions

rd_rnx_o	Reads RINEX formatted observation data (‘O’ files).
readeph	Reads RINEX formatted epehemeris/navigation data (‘N’ files).

RINEX Writing Functions

wr_rnx_o	Writes RINEX formatted observation data (‘O’ files).
wr_rnx_e	Writes RINEX formatted epehemeris/navigation data (‘N’ files).

Alphabetical Listing of Functions

Each function is listed alphabetically with descriptions of inputs, outputs, algorithms, and references.

rd_rnx_o

Purpose Read RINEX-II observation files.

Syntax [obs, obs_qual, type_str, clk_off] = rd_rnx_o(file_name)

```
Input:
         file_name = file name of RINEX-II data file to read (1xn) (string).

Output:
         obs      = observation data (nxm) (number_of_obs x 3+number_of_data_types)
                    The columns of the obs matrix are defined as ...
                           1              2        3       4        5        6
                     [GPS_week, GPS_sec, PRN, data(1), data(2), data(3), ...]
                     data fields depend on RINEX header file.  Use type_str
                     variable to identify each observation type.  Observations are
                     loaded in the data(n) variable in the same order as the typ_str.
                     Example obs matrix with 4 observation types (C1    L1    D1    P2)
                        1       2       3          4              5           6          7
                      Week   Sec    PRN     C1            L1         D1        P2
                     [1023  856792  2   20855400.126 109595864.844 -969.622 20855402.416; ...
                      1023  856792  10  23539064.233 123698609.620  486.471 23539068.706; ...
                      1023  856792  18  21600694.212 113512400.206 2518.909 21600696.656; ...
                      1023  856792  28  20883879.485 109745525.139  853.284 20883880.933; ...
                      1023  856793  2   20855215.771 109594895.988 -969.455 20855217.99]
         obs_qual = observation quality matrix (nx2) [LLI Signal_strength]
                        LLI - loss of lock indicator (0-7) see RINEX format for meaning
                        Signal strngth (1-9, min-max), 0 - unknown/don't care
         type_str = observation type string (num_obs_types x 2) character string.
         clk_off = clock offset (nx1) seconds if available.
```

See Also **readeph, refrinex**

Notes

- Data that is not available will be filled with NaN.
- Does not handle mixed GPS/GLONASS RINEX data file.
- GPS time is kept without rollovers (e.g. week 1025 is week 1 with a rollover).
- Data read in will be saved in a *.mat file with the same file prefix as the input data file (e.g. sample.obs data will be saved in sample.obs.mat). When RD_RNX_O is called subsequent time, the mat file will be loaded to same time on reading in the raw data. Remove the *.mat file to force the raw data to be read from scratch.

Reference “RINEX: The Receiver Independent Exchange Format Version 2”, Werner Gurtner, Astronomical Institute, University of Berne.

readeph

Purpose Read RINEX-II navigation files. The navigation files contain the satellite ephemeris information.

Syntax eph = readeph(file_name)

Input:

    file_name = file name of RINEX-II data file to read (1xn) (string).

Output:

```
gps_ephem = ephemeris matirx for all satellites (nx24), with columns of
            [prn,M0,delta_n,e,sqrt_a,long. of asc_node at GPS week epoch,
             i,perigee,ra_rate,i_rate,Cuc,Cus,Crc,Crs,Cic,Cis,Toe,IODE,
             GPS_week,Toc,Af0,Af1,Af2,perigee_rate]
             Ephemeris parameters are from ICD-GPS-200 with the
             exception of perigee_rate.
```

See Also **rd_rnx_o**

Notes See propgeph for complete details on the ephemeris matrix.

Reference "RINEX: The Receiver Independent Exchange Format Version 2", Werner Gurtner, Astronomical Institute, University of Berne.

refrinex

Purpose Reformat RINEX data.

Syntax [gps_time, prn, data1, data2, data3, ..., datan] = refrinex(obs, obs_qual, clk_off)

```
Input:
        obs = observation data (nxm) (number_of_obs x 3+number_of_data_types)
                        1           2       3     4       5        6
                  [GPS_week, GPS_sec, PRN, data(1), data(2), data(3), ...]
                  data fields depend on RINEX header file.  Use type_str
                  variable to identify each observation type.  Observations are
                  loaded in the data(n) variable in the same order as the typ_str.
                  Example obs matrix with 4 observation types (C1   L1   D1   P2)
                    1        2     3         4               5              6           7
                  [1023  856792  28  20855400.126 109595864.844 -969.622 20855402.416]
        obs_qual = observation quality matrix (nx2) [LLI Signal_strength]
                   LLI - loss of lock indicator (0-7) see RINEX format for meaning
                   Signal strngth (1-9, min-max), 0 - unknown/don't care
        clk_off = clock offset (nx1) seconds if available (optional)

Output:
        gps_time = GPS time tag for each time [GPS_week GPS_sec] (kx2).
                   Each element corresponds to a row in thedata matrices.
                   k is the number of obseration times in the data set.
                   If the clock offset is provided in the input (3 input parameters),
                   a 3-rd column is added to the time matrix that has the clock offset
                   [GPS_week GPS_sec clk_off] (kx3)
        prn = PRN numbers for each column in the satellite data matrices (jx1).
        data1 = data matrix of observations corresponding to the first data type
                 in the obs matrix (column 4 of obs) (kxj) (e.g. L1, C1, D1, etc.)
        data2 = data matrix of observations corresponding to the first data type
                 in the obs matrix (column 5 of obs) (kxj) (e.g. L1, C1, D1, etc.)
        …….
        datan = data matrix of observations corresponding to the first data type
                 in the obs matrix (column 4+n-1 of obs) (kxj) (e.g. L1, C1, D1, etc.)
```

Description Function to reformat RINEX data from RD_RNX_IN (obs matrix) into separate matrices. This function takes the output from rd_rnx_o and reformats the observation matrix into a separate matrix for each observation type. The observation type is defined in the type_str variable.

See Also **rd_rnx_o**

wr_rnx_e

Purpose Write RINEX-II ephemeris data files.

Syntax wr_rnx_e(file_name, sat_data, info_struct)

```
Inputs:
        file_name = file name of RINEX-II ephemeris data file to write (1xn) (string)
        eph_data = ephemeris matrix for all satellites (nx24), with columns of
                     1  2        3   4     5                  6
                  [prn,M0,delta_n,e,sqrt_a,long. of asc_node at GPS week epoch,
                   7   8           9     10    11    12   13   14  15  16   17  18
                   i,perigee,ra_rate,i_rate,Cuc,Cus,Crc,Crs,Cic,Cis,Toe,IODE,
                      19        20  21  22  23     24
                   GPS_week,Toc,Af0,Af1,Af2,perigee_rate]
                   Ephemeris parameters are from ICD-GPS-200 with the
                   exception of perigee_rate.
        info_struct = structure with information for the RINEX header (optional)
             Reasonable default values have been selected for all parameters.  Any
             of this structure may be left blank.  All fields must be strings.

             info_struct.ion_alpha         - Ionosphere parameters A0-A3 of almanac,
                                    (page 18 of subframe 4), (1x4)
             info_struct.ion_beta    - Ionosphere parameters B0-B3 of almanac (1x4)
             info_struct.comment     - comments for comment line
Output:
        none
```

See Also **wr_rnx_o**

Notes See propgeph for complete details on the ephemeris matrix.

wr_rnx_o

Purpose Write RINEX-II observation files.

Syntax function wr_rnx_o(file_name, sat_data, info_struct)

```
Inputs:
        file_name = file name of RINEX-II data file to write (1xn) (string)
        sat_data = satellite measurement data structure.  This structure
             contains all of the measurement data to be output to the RINEX file.
             sat_data.meas_types - measurement types that follow in the data
                           (e.g. L1, C1, D1, L2, D2, C2). (nx2) (string)
                           the data member of this structure must contain
                           the same number of arrays.
             sat_data.data{1}    - measurement data for the first measurement type.
                           [GPS_week GPS_sec prn data] (nx4)
             sat_data.data{2}    - measurement data for the second measurement type.
                           [GPS_week GPS_sec prn data] (nx4)
                  .
                  .
             sat_data.data{n}    - measurement data for the n-th measurement type.
                           [GPS_week GPS_sec prn data] (nx4)
        info_struct = structure with information for the RINEX header (optional)
             Reasonable default values have been selected for all parameters.  Any
             of this structure may be left blank.  All fields must be strings.

             info_struct.data_type - satellite system. 'G' = GPS, 'R' = GLONASS
             info_struct.agency - agency name creating the file
             info_struct.observer - observer creating the file
             info_struct.comment - comments for comment line
             info_struct.marker_name - antenna marker name
             info_struct.marker_num - antenna marker number
             info_struct.rectype - receiver type
             info_struct.recnum - receiver number
             info_struct.recver - receiver version
             info_struct.anttype - antenna type
             info_struct.antnum - antenna number
             info_struct.antposx - approximate antenna position ECEF WGS-84 - X (m) F14.4
             info_struct.antposy - approximate antenna position ECEF WGS-84 - Y (m) F14.4
             info_struct.antposz - approximate antenna position ECEF WGS-84 - Z (m) F14.4
             info_struct.antdelh - antenna Height (H) above marker - Delta H/E/N (m) F14.4
             info_struct.antdelh - antenna East (E) from marker - Delta H/E/N (m) F14.4
             info_struct.antdelh - antenna North (N) from marker - Delta H/E/N (m) F14.4

Output:
        none
```

See Also **wr_rnx_o**

Notes See the RINEX-2 standard for a description of each info_struct parameter.

Reference "RINEX: The Receiver Independent Exchange Format Version 2", Werner Gurtner, Astronomical Institute, University of Berne.