Constellation Toolbox

For MATLAB®

User's Guide

Version 7.00

Constell, Inc.

Constellation Toolbox User's Guide, Version 7.00
April 2006

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 Constellation Toolbox is normally contained on a PC-based compact disc. Other formats are available upon request. This toolbox works with MATLAB version 6.0 or greater.

For PC users, the Toolbox is distributed as a standard Windows install program. Run the set-up program on the disk and follow the on-screen instructions. The resulting directory structure after installation will be:

...\constell	- contains all toolbox functions other than tutorials
...\constell\almanacs	- contains YUMA almanacs
...\constell\tutorial	- contains the tutorial code

Unix users will be provided a compact disk with the Toolbox stored as a tar file. Mac users will be supplied the raw script files on a compact disk with the directory structure described above. 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 directory (using the example above) to your path. We recommend you add the back of your MATLAB path.
or
Edit the pathdef.m file. For normal installations, this file is found in the ..\matlabxx\toolbox\local directory.

Your Constellation Toolbox installation is now complete.

Quick Start

Try out the toolbox using the following tutorials.

In the ...\constell\tutorial directory:

vis_e.m – an example of dilution of precision (DOP) values and visibilities for a receiver fixed on the ground in Boulder, Colorado

vis_o.m – an example of visibilities for a satellite in low-Earth orbit

vis_ac.m – an example of a GPS receiver mounted on an aircraft

ex_dgps.m – an example of a differential GPS application

Introduction

The Constellation Toolbox for MATLAB® is a highly integrated collection of .m utilities that provide the capability to model, simulate, and analyze satellite constellations. The Toolbox provides specific modeling capabilities for the GPS and GLONASS constellations for a wide variety of navigation applications in addition to supporting user supplied constellations. A brief description of the toolbox capability is presented below.

- Read in GPS and GLONASS almanac data files in Yuma format
- Read in and process a data file containing GPS data stored in NMEA 0183 v. 2.0 format (most GPS receivers put this data stream out through a serial port)
- Calculate satellite locations as a function of time using orbit propagators based on almanac data
- Compute GPS/GLONASS satellite line-of-sight vectors and visibilities from a fixed ground site or from a moving vehicle, including both translational and rotational motion (car, airplane, rocket, satellite, etc.).
- Model antenna masking due to the Earth and discrete obstructions, for antennas both fixed and attached to rotating vehicles
- Calculate common DOP (dilution of precision) values as a function of time: GDOP, PDOP, VDOP, HDOP, and TDOP (geometric, position, vertical, horizontal, and time, respectively).
- Calculate pseudoranges and range-rates based on user position, velocity, masking, and error models
- Simulate user position and velocity measurements from models of pseudoranges and rates
- Simulate differential GPS errors
- Model selective availability errors and atmospheric path delays for the troposphere and ionosphere
- Evaluate satellite selection models and tracking algorithms
- Easily transform vectors between common reference frames: Earth-Centered-Earth-Fixed, Earth-Centered-Inertial, North-East-Down, and satellite local-level frames
- Convert between UTC and GPS time
- MATLAB GUI-based demonstration shows the toolbox capabilities for the most common applications:
  - calculate DOPs vs. time for a fixed ground station or a satellite in orbit
  - compute and plot GPS/GLONASS constellation visibilities
  - Simulate GPS signals with a breakdown of errors into component sources

Example functions later in the tutorial demonstrate how the toolbox functions are combined into common applications.

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) whenever 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

Example 1: Visibility and DOP Calculations for a Fixed Receiver

A common application of the Constellation Toolbox for GPS and GLONASS applications is to compute satellite visibility and DOP values over time for a receiver at a fixed location on or near the surface of the Earth. The function calling sequence and flow of data from the GPS/GLONASS satellite orbit descriptions to the calculation of visibility and DOPs is shown in Figure 1 and Figure 2 on the following pages. A sample MATLAB script file, vis_e.m shown on the succeeding three pages, illustrates how this application is mechanized with the Constellation Toolbox.

A simulation of the GPS/GLONASS visibilities and DOP computations begins by calculating the GPS satellite positions over the time interval and at the time step size of interest. The line-of-sight vectors from the user position are then calculated, the visible satellites determined, and the DOP values and visibilities computed. A formulation for this process is presented below.

1. The satellite information is read from an almanac file by the function readyuma.m (vis_e.m, line 38). This almanac data must be in Yuma format. Almanac data for the week nearest the time interval of interest should be used. The unhealthy satellites are immediately removed from further consideration (vis_e.m, line 41, 42).

2. The data is converted to the standard GPS ephemeris format by the function alm2geph.m (vis_e.m, line 45).

3. The positions and velocities of all GPS satellites are calculated over the time interval at the specified time step size by propgeph.m (vis_e.m, line 52). The results are expressed in the ECEF frame. The gps_times and satellite ID numbers (known as prn) are also returned by propgeph.m so a coherent set of data is available for each position vector.

4. The line-of-sight (LOS) vectors are calculated by subtracting the user position vector, in the ECEF frame, from each of the satellite positions in the function los.m (vis_e.m, line 55). Los.m assumes the GPS receiver is stationary with respect to the Earth. At this point in the formulation, LOS vectors are computed to all satellites for all times of interest; there are no obstructions modeled and the Earth is effectively transparent.

5. The LOS vectors are transformed from the ECEF frame to the North-East-Down (NED) frame based on the user's position (vis_e, line 61). The azimuth and elevation via the function ned2azel.m (vis_e.m, line 61) to each satellite are computed from the NED vectors and the local making is applied via the function vis_data.m (vis_e.m, line 65).

6. The dilution of precision (DOP) values and associated times are computed by ned2dops.m for each time step for the satellites that are inside the specified masking regions (vis_e.m, line 68). All visible satellites are used to compute these DOP values.

7. The number of visible satellites during the time interval, and associated times of visibility, are determined by num_vis.m (vis_e.m, line 80).

8. Use the function passdata.m to compute pass information for each satellite (e.g. rise time, set time, duration, etc.) (vis_e.m, line 84).

9. The remaining code in vis_e.m demonstrates the use of the function makeplot.m (vis_e.m, line 100). Makeplot.m is a fast and easy way to generate a common set out output plots that includes a sky plot, DOP, number of visible satellites, azimuth, and elevation figures.

Satellite Positions from Orbital Elements

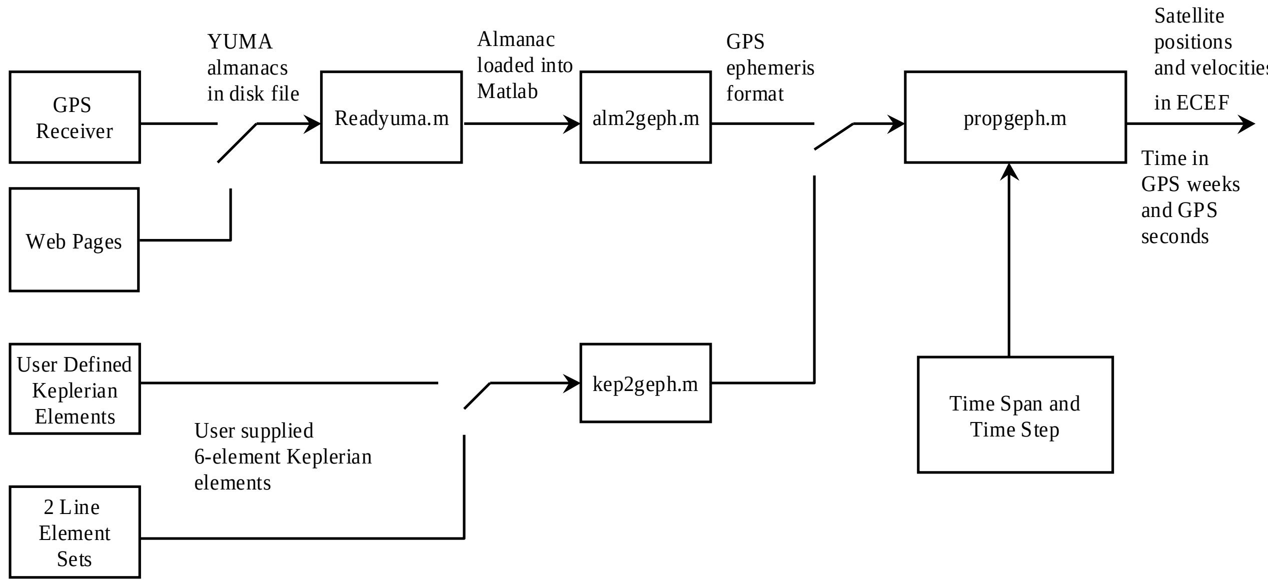


Figure 1: Satellite Positions from Orbital Elements

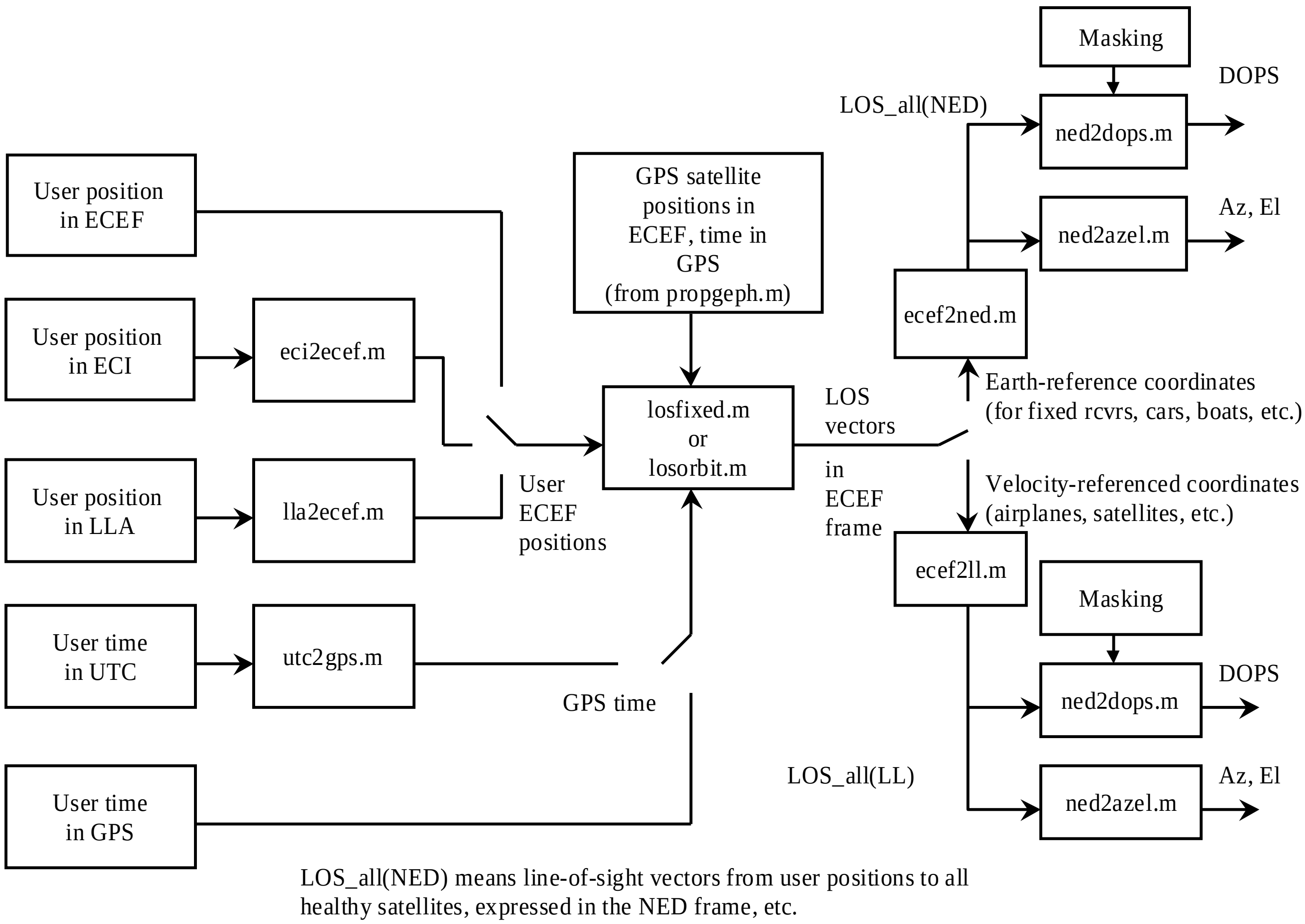


Figure 2: Positions to DOPS, Azimuth, and Elevation Angles

```
% vis_e.m
%
% Function to demonstrate the visibility and DOPS routines for a location fixed
% on the Earth.  Computes the DOPS, azimuth and elevation to the GPS
% satellites and displays an elevation vs. time, azimuth vs. time, # of satellites
% visible, and a sky plot showing azimuth/elevation pairs.
%
% See also vis_o.m

% Written by: Maria Evans Eagen
% Copyright (c) 1998 Constell, Inc.

% functions called: LLA2ECEF, READYUMA, ALM2GEPH, UTC2GPS,
%                   PROPGEPH, LOS, ECEF2LLA, ECEF2NED, NED2AZEL,
%                   VIS_DATA, NED2DOPS, GPST2SEC, NUM_VIS,
%                   PASSDATA, GPS2UTC, MAKEPLOT

clear       % clear all variables in workspace
close all   % close all open windows

% Set simulation parameters
sta_name = ['Boulder'];                                          % Station Name
location = [40.0*pi/180, -105.16*pi/180, 1700.0];   % Lat, long, alt (rad, m)
mask = 5.0*pi/180;                                % simple 5 deg elevation Mask (radians)
start_time = [2006 4 17 0 0 0];                   % Start Date (yr, mon, day, hr, mn, sc)
stop_time = [2006 4 14 4 0 0];                    % Stop Date (yr, mon, day, hr, mn, sec)
time_step = 30;                                                  % Time step (sec)
gps_start_time = utc2gps(start_time);
alm_file = find_alm(gps_start_time(1));           % GPS almanac file to be used here

%%%%% BEGIN ALGORITHM CODE %%%%%

% convert the station location from lat, long, alt. to ECEF vector
location_ecef = lla2ecef(location);

% load the GPS almanac for the given almanac week
alm_2_use = readyuma(alm_file);

% sort out the unhealthy satellites
I_gps_good = find(alm_2_use(:,2) == 0);
alm_2_use = alm_2_use(I_gps_good,:);

% convert the almanacs to ephemeris format
[gps_ephem] = alm2geph(alm_2_use);

% first convert the start and stop times to GPS time
start_gps = utc2gps(start_time);
stop_gps = utc2gps(stop_time);

% compute satellite positions in ECEF frame for the given time range and interval
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,start_gps,stop_gps,time_step);

% compute LOS vectors in ECEF frame
[t_los_gps, gps_los, los_ind] = los(t_gps(1,:), location_ecef, ...
                          t_gps, [prn_gps x_gps]);
prn_gps_los = prn_gps(los_ind(:,2));
```

```
location = ecef2lla(location_ecef(los_ind(:,1),:));

% convert LOS in ECEF to NED frame
[gps_los_ned] = ecef2ned(gps_los, location);

% Compute masking
[az, el] = ned2azel(gps_los_ned);
[az_el_pass, I_pass] = vis_data(mask, [az el]);

% Compute DOPS using Earth-fixed masking to determine the visible satellites
[dops,t_dops] = ned2dops(gps_los_ned(I_pass,:),t_los_gps(I_pass,:));

% Reset the data array to contain only visible data
if any(I_pass),
  % reset the arrays to contain only visible data
  t_los_gps = t_los_gps(I_pass,:);
  az = az(I_pass);
  el = el(I_pass);
  prn_gps_los = prn_gps_los(I_pass);
end;

% compute number of visible satellites
[t_vis, num_sats] = num_vis(t_los_gps);

% Make arrays for using the MAKEPLOT function
visible_data = [az_el_pass t_los_gps prn_gps_los];
[pass_numbers, pass_times, pass_summary] = passdata(t_los_gps, time_step, ...
              [ones(size(prn_gps_los)) prn_gps_los], visible_data(:,1:2));
number_vis = [t_vis,num_sats];
gps_dops = [t_dops dops];

% Compute pass information and print to screen
pass_times_utc = gps2utc(pass_times(:,2:3));
output_array = [pass_times_utc(:,2:3) pass_times_utc(:,1) pass_times_utc(:,4:6) ...
   pass_times(:,4)/60 pass_times(:,5:6) pass_summary(:,1,1)*180/pi ...
   pass_summary(:,2,1)*180/pi pass_summary(:,1,2)*180/pi ...
   pass_summary(:,2,2)*180/pi pass_summary(:,2,4)*180/pi];
fprintf('Start Time of Pass    Duration Ground S/C Rise Az. Elev.  Set Az. Elev. Max Elev.\n');
fprintf('       (UTC)           (min) Station PRN  (deg)   (deg)   (deg)  (deg)   (deg)\n');
fprintf('%2d/%2d/%4d %2d:%2d:%4.1f %7.2f %5d %5d %7.2f %6.2f %7.2f %6.2f %7.2f\n', ...
  output_array');

fig_handles = makeplot(visible_data, pass_numbers, number_vis, gps_dops,...
                       'GPS Visibility for Boulder, CO');

%%% End of plotting

% end of vis_e.m
```

End of Example 1

Example 2: Visibility And DOPs For a Moving Vehicle

The second example is similar to the first, except that the vehicle containing the GPS receiver is moving, rather that remaining fixed on the Earth. The moving vehicle in this case is a satellite in low Earth orbit. Any moving vehicle can be modeled using a process identical to the one described in this section.

The data flow diagrams that apply to this example are contained in Figure 1 and Figure 2. The example code, vis_o.m, follows this section. The formulation to compute the satellite visibility and DOP values for a moving vehicle is given below.

1. Compute the GPS satellite positions over the desired interval following steps 1-3 of Example 1.

2. Transform the user satellite data from a Keplerian 6-element set to GPS ephemeris format using kep2geph.m (vis_o.m, line 49). See also the lower left portion of Figure 1.

3. Compute the user satellite ECEF positions as a function of GPS time with the function propgeph.m (vis_o.m, line 52). Note that at this point, preexisting trajectory data in the ECEF frame for some other type of moving vehicle (e.g., a car, airplane, or boat) could be substituted for the ECEF satellite data.

4. Compute the LOS vectors between the user satellite and the GPS constellation, in the ECEF frame, via los.m (vis_o.m, line 59). Also get the Earth blocking information (obscure_data) from los.m. This information will be used by vis_data.m to determine which satellites are blocked by the Earth.

5. Convert the LOS vectors from ECEF to local-level by using ecef2ll.m with variable user positions and velocities (vis_o.m, line 66), rather that a fixed user location as in Example 1. Note that the x_user and v_user vectors need to be expanded in size in a manner such that they are synchronous with the LOS vector.

6. All subsequent steps are identical to steps 6-11 in Example 1.

End of Example 2.

```
% vis_o.m
%
% Function to demonstrate the visibility routines for orbiting satellites.
% Computes the azimuth and elevation to the GPS satellites and
% displays an elevation vs. time, azimuth vs. time, # of satellites
% visible, and a sky plot showing azimuth/elevation pairs.
%
% See also vis_e.m

% Written by: Maria Evans Eagen
% Copyright (c) 1998 Constell, Inc.

% functions called: READYUMA, ALM2GEPH, UTC2GPS, PROPGEPH, LOS,
%                   ECEF2LL, NED2AZEL, VIS_DATA, PASSDATA,
%                   MAKEPLOT, KEP2GEPH, LL2DOPS

clear
close all

% Set simulation parameters

% Kepler Elements for the user satellite (not the GPS satellites)
% [sv_num, a (meters), e, i, node, argp, M, epoch week, epoch sec]
kep_elems = [1 7000000, 0, 45*pi/180, 0*pi/180, 0*pi/180, 0*pi/180, 158, 0];

mask = -25.0*pi/180;                % 0 radians elevation Mask
start_time = [2006 4 17 7 0 0];     % Start Date (yr, mon, day, hr, mn, sc)
stop_time = [2006 4 17 7 30 0];     % Stop Date (yr, mon, day, hr, mn, sec)
time_step = 60;                     % Time step (sec)

[startweek startsec startday roll1] = utc2gps(start_time);
[stopweek stopsec stopday roll2] = utc2gps(stop_time);
if roll1==0,
   start_gps=[startweek startsec roll1];
   stop_gps=[stopweek stopsec roll2];
else
   start_gps=[startweek startsec];
   stop_gps=[stopweek stopsec];
end

% Find the almanac that is most recent to the start time
alm_file = find_alm(start_gps(1));

%%%%% BEGIN ALGORITHM CODE %%%%%

% load the GPS or GLONASS almanac for the given almanac week
alm_2_use = readyuma(alm_file);

% sort out the unhealthy satellites
I_gps_good = find(alm_2_use(:,2) == 0);
alm_2_use = alm_2_use(I_gps_good,:);

% convert the almanacs to ephemeris format
[gps_ephem] = alm2geph(alm_2_use);

% first convert the start and stop times to GPS time
```

```
start_gps = utc2gps(start_time);
stop_gps = utc2gps(stop_time);

% convert from the keplerian set to GPS ephemeris format
user_ephem = kep2geph(kep_elems);

% Compute user satellite positions in ECEF
[t_user,prn_user,x_user,v_user] = ...
        propgeph(user_ephem,start_gps,stop_gps,time_step);

% Compute navigation satellite positions in ECEF
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,start_gps,stop_gps,time_step);

% Compute LOS vectors in ECEF
[t_los_gps,los_vect,los_ind,obscure_info] = los(t_user, x_user, t_gps, [prn_gps x_gps]);

% Rename some variables for ease of use later
gps_prn_los = prn_gps(los_ind(:,2));
I_user = los_ind(:,1);

% convert LOS in ECEF to local-level
[los_ll] = ecef2ll(los_vect, x_user(I_user,:),v_user(I_user,:));

% compute az and els
[az el] = ned2azel(los_ll);

% Find indices to satellites above the mask and not obscured by the Earth
[az_el_prn, I_vis] = vis_data(mask, [az, el, gps_prn_los], obscure_info);
if any(I_vis),
  % reset the arrays to contain only visible data
  t_los_gps = t_los_gps(I_vis,:);
  los_ll = los_ll(I_vis,:);
  az = az(I_vis);
  el = el(I_vis);
  gps_prn_los = gps_prn_los(I_vis);
end;

% Compute DOPs
[dops, t_dops, num_sats] = ll2dops(los_ll, t_los_gps);

%%% Plotting Section %%%
% Make arrays for using the MAKEPLOT function
visible_data = [az el t_los_gps gps_prn_los];
[pass_numbers, pass_times, pass_summary] = passdata(t_los_gps, 600, ...
        [ones(size(gps_prn_los)) gps_prn_los], visible_data(:,1:2));
num_vis = [t_dops,num_sats];
gps_dops = [t_dops dops];

% Compute pass information and print to screen
pass_times_utc = gps2utc(pass_times(:,2:3));
output_array = [pass_times_utc(:,2:3) pass_times_utc(:,1) pass_times_utc(:,4:6) ...
   pass_times(:,4)/60 pass_times(:,5:6) pass_summary(:,1,1)*180/pi ...
   pass_summary(:,2,1)*180/pi pass_summary(:,1,2)*180/pi ...
   pass_summary(:,2,2)*180/pi pass_summary(:,2,4)*180/pi];
fprintf('Start Time of Pass    Duration  Obs.  S/C Rise Az. Elev.  Set Az. Elev. Max Elev.\n');
fprintf('       (UTC)            (min)   PRN   PRN  (deg)   (deg)   (deg)  (deg)   (deg)\n');
```

```
fprintf('%2d/%2d/%4d %2d:%2d:%4.1f %7.2f %5d %5d %7.2f %6.2f %7.2f %6.2f %7.2f\n', ...
  output_array');

fig_handles = makeplot(visible_data, pass_numbers, num_vis, gps_dops,...
                       'GPS Visibility from a=7000 km, i=45 deg.');

%%% End of plotting

% end of vis_o.m
```

End of Example 2

Example 3: GPS Visibility for a Vehicle with Varying Attitude and Body-Fixed Masking

The third example demonstrates how the toolbox can be used to compute the GPS visibility for an antenna attached to an aircraft. The antenna is fixed to the aircraft body and masking of the tail is modeled. This example also shows how to read in vehicle trajectory data from a disk file and combine it with the computed GPS satellite position data.

The airplane trajectory begins at the equator, flying North at 200 m/s (450 mph) for 60 sec. The airplane rolls 45 deg to the right and turns for 360 deg of heading (about 128 sec). The airplane then rolls back to zero and continues to fly North until 300 sec. have elapsed since the beginning of the flight. Data is stored at one second time steps. A block diagram of the data flow and function calling sequence is shown in Figure 3. The example MATLAB file, vis_ac.m, is shown on the three pages following Figure 3.

The formulation for computing the number of visible satellites and the DOP values for this example follows below.

1. Load the GPS almanac, select the healthy satellites, and convert to the ephemeris format (vis_ac.m, lines 49-57).

2. Load in the aircraft data file and assign absolute times to the aircraft data (vis_ac.m, lines 60-69).

3. Propagate the GPS orbits with time corresponding to the aircraft times (vis_ac.m, line 74).

4. Compute the line-of-sight vectors from the airplane to the GPS satellites, using los.m (vis_ac.m, line 84). When calling los.m for a trajectory like this, the Earth masking (obscure_data) is requested from the los.m function. This information will be used in vis_data.m to mask out GPS satellites that are blocked by the Earth.

5. Rotate the LOS vector from the ECEF to the NED frame. Then rotate from the NED frame to the body frame (vis_ac.m, lines 102-105).

6. The azimuth and elevation angles are determined in the body frame (line 105), and the body fixed masking and Earth blocking is applied with vis_data.m (vis_ac.m, line 118).

7. The LOS vectors that passed the Earth masking test are then rotated into the vehicle body frame using ned2body.m and the attitude matrix (vis_ac.m, line 118).

8. The visible data is then extracted from the original matrices (vis_ac.m, lines 124-129).

9. The DOP values, number of satellites tracked, and NED azimuth and elevation are plotted (vis_ac.m, lines 133-134).

10. Makeplot.m is called to generate the stanrd set of plots (vis_ac.m, line 142).

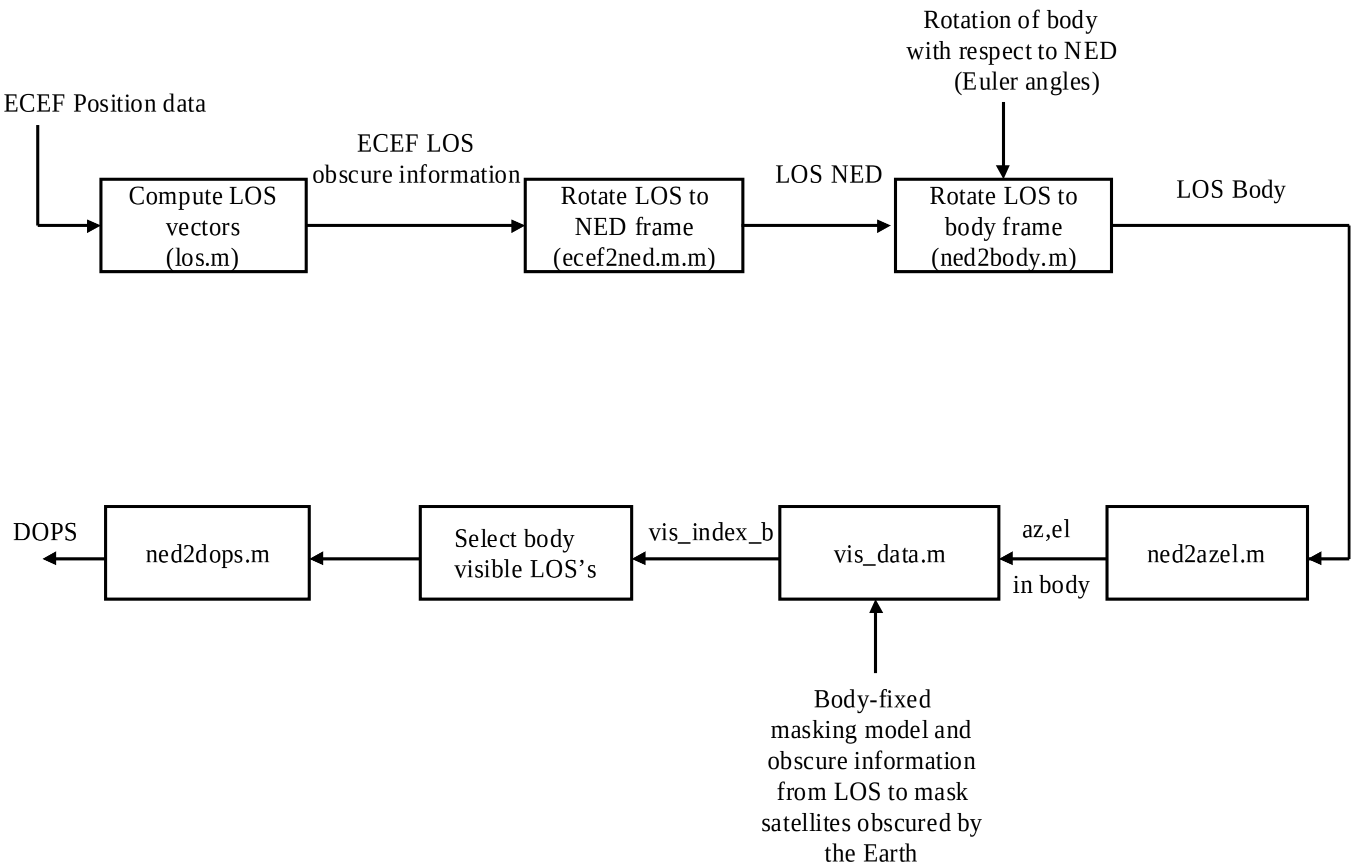


Figure 3: Implementation of Body-Fixed Masking for a Rotating Vehicle

```
% vis_ac.m
%
% Function to demonstrate body-fixed masking and Earth obscuring functions
% for an aircraft trajectory.
%
% An aircraft is modeled with an initial location at ECEF = (6378e3,0,0).  It
% has a ground speed of 200 m/s (450 mph).  The aircraft flies straight and
% level North for 60 sec, then banks to the right by 45 deg and executes a
% 360 deg. turn at constant altitude (about 128 sec).  At the end of the turn,
% the airplane levels back out and continues to fly North, straight and level,
% until 300 secs have elapsed since the beginning of the flight.  Aircraft
% position, velocity, and attitude data are stored in a file at 1 sec intervals.
% This data is read in and used in this example.
%
% A GPS antenna is mounted to the top of the airplane.  It is masked by the
% fuselage (elevation < 0) and by the vertical tail (elevation < 30 deg for
% an azimuth range of 170 to 190 deg, relative to the nose of the airplane).
% A 5 deg mask of the Earth is also used.

% Written by: Jimmy LaMance
% Copyright (c) 1998 Constell, Inc.

% functions called: READYUMA, ALM2GEPH, UTC2GPS, GPST2SEC, SEC2GPST,
%                   PROPGEPH, LOS, ECEF2NED, NED2BODY, NED2AZEL, VIS_DATA,
%                   NED2DOPS, MAKEPLOT

clear       % clear all variables in workspace
close all   % close all open windows

% Set simulation parameters
d2r = pi/180;

% Define the body fixed antenna mask
mask_b = [0 190 170;      % 0 elevation mask of fuselage between 190-170 deg
        30 170 190]*d2r; % 30 deg elevation mask of tail between 170-190 deg

% Define an absolute start time for the simulation.  The aircraft trajectory
% is given in relative time (seconds past the start of the trajectory).  Get
% the start time in seconds past the GPS epoch.
start_time = [2006 4 17 1 0 0];         % UTC start time
% Compute the stop time based on the start time and duration
% first convert the start time to GPS time
[startweek startsec startday roll] = utc2gps(start_time);
if roll==0,
   start_gps=[startweek startsec roll];
else
   start_gps=[startweek startsec];
end
start_sec = gpst2sec(start_gps);

%%%%% BEGIN ALGORITHM CODE %%%%%

% Find the almanac that is most recent to the start time
alm_file = find_alm(start_gps(1));

% Read in the almanac
```

```
alm_2_use = readyuma(alm_file);

% Sort out the unhealthy satellites
I_gps_good = find(alm_2_use(:,2) == 0);
alm_2_use = alm_2_use(I_gps_good,:);

% Convert the almanacs to ephemeris format
[gps_ephem] = alm2geph(alm_2_use);

% Read in airplane data:  time, position, velocity, attitude
load airplane.dat;               % aircraft data in an ascii file
ac_time = airplane(:,1);         % aircraft relative time: 0-300 sec
ac_pos = [airplane(:,2:4)];      % aircraft ECEF xyz position (m)
ac_vel = [airplane(:,5:7)];      % aircraft ECEF xyz velocity (m/s)
ac_att = [airplane(:,8:10)];     % aircraft attitude wrt NED (deg)
ac_att = ac_att*d2r;             % convert to radians

% Convert aircraft relative time to gps time, using start_time as time0
ac_time = ac_time + start_sec;  % absolute aircraft time (secs)
ac_time_s = sec2gpst(ac_time);  % GPS time version of ac_time

% Compute satellite positions in ECEF frame at the times from the aircraft
% trajectory.  This is a simple way to obtain time synced GPS positions and
% velocities if the desired output times are not in even steps.
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,ac_time_s);

% Compute LOS vectors from airplane to GPS satellites, in ECEF. The los_ind
% (line-of-site indices) will also be used to line up all of the data with
% the positions used to compute the LOS vector.   In addition, we will get the
% Earth obscuring information back from LOS to determine which satellites
% are blocked by the Earth.  Get the tangent altitude
% (how far above the surface of the Earth does the LOS vector pass) relative
% to the spheriod.
earth_model = 0;           % 1 = use ellipsoid, 0 = use spheriod
[t_los_gps, los_ecef, los_ind, obscure_info] = ...
    los(ac_time_s, ac_pos, t_gps, [prn_gps x_gps],earth_model);

% Rename some of the variables for easier use later.
prn_nav = prn_gps(los_ind(:,2));
gps_pos = x_gps(los_ind(:,2),:);
I_ac = los_ind(:,1);

% Build up an attitude matrix that uses the input matrix, ac_att, but
% is sync'd to the LOS vectors.  This is necessary since the LOS matrix
% is 4-12 times the size of the ac_att matrix.  The LOS matrix has several
% vectors at the same time point, and we need the attitudes to be correlated
% with these LOS vectors.  A large attitude matrix, ac_att_large, is built
% from ac_att using I_ac from losorbit.
ac_att_large = ac_att(I_ac,:);      % make the large attitude matrix
ac_pos_large = ac_pos(I_ac,:);  % do the same for the airplane position data

% Rotate the LOS's from ECEF to the NED frame
los_ned = ecef2ned(los_ecef,ecef2lla(ac_pos_large));

% Rotate the Earth-visible LOS's into the airplane body frame
los_body = ned2body(los_ned,ac_att_large);
```

```
% Convert LOS's in body frame to body-referenced az and el's
% (use ned2azel here since it is the same as body2azel would be, if we had one)
[az_b,el_b] = ned2azel(los_body);

% Apply the body-fixed masking model to the body az and el's.
% Use the obscure_info from LOS to mask out the satellite that are blocked by
% the Earth.  To demonstrate the use of this capability, we have chosen to
% mask all satellites that the LOS would be within 5 km of the Earth using
% a spherical Earth model.  This can also be done with an elliptical
% Earth model.
obscure_height = 5000;    % min height above the spheriod/ellipse (m)
[az_el, I_vis_b] = vis_data(mask_b,[az_b,el_b],obscure_info,obscure_height);

% Use this index of visible satellites in the body frame to build the LOS data
% set that is visible in the body frame, with all the maskings applied.
% Express this LOS data in the NED frame, however, to keep the DOPS and
% visibility calculations in the local-level frame.
if any(I_vis_b),
  los_ned = los_ned(I_vis_b,:);      % LOS vectors, visible, in NED frame
  los_body = los_body(I_vis_b,:);    % LOS vectors, visible, in body frame
  t_los_gps = t_los_gps(I_vis_b,:);  % times associated with above LOS's
  prn_nav = prn_nav(I_vis_b);        % prn's to match the LOS's
end;

% Finally, compute DOPS, number of satellites tracked, and the NED azimuth
% and elevation for plotting.
[dops,t_dops,num_sats_vis] = ned2dops(los_ned,t_los_gps);
[az_ned,el_ned] = ned2azel(los_ned);

% Make arrays for using the MAKEPLOT function
ac_vis_data = [az_ned el_ned t_los_gps prn_nav];
pass_numbers = ones(size(ac_vis_data,1),1);
num_ac_vis = [t_dops num_sats_vis];
gps_dops = [t_dops dops];

fig_handles = makeplot(ac_vis_data, pass_numbers, num_ac_vis, gps_dops,...
                       'Aircraft Example for GPS Visibility with Attitude');

% end of vis_ac.m
```

End of Example 3

Example 4: Simulated Raw Measurements from a Static GPS Receiver

This fourth example demonstrates how the toolbox can be used to generate simulated pseudo-range (PR), accumulated carrier phase, and Doppler (raw GPS) measurements. This example is for a single receiver fixed to the surface of the Earth. This example will be expanded to be a differential GPS (DGPS) problem in example 5. Models for Selective Availability (SA), the troposphere, ionosphere, receiver clock, receiver noise, GPS satellite clocks, GPS satellite movement during signal propagation, Earth rotation, and relativity can be modeled to achieve raw GPS measurements that are very realistic

The data flow diagram for this application is shown in Figure 4. This is the standard flow for computing GPS measurements, differential corrections, and navigation solutions. The data flow for computing a PR measurement is shown in Figure 5.

The formulation for computing simulated position measurements from a fixed ground station is shown below.

1. Define the simulation start and stop times as well as the GPS receiver locations. (ex_gps.m, lines 17-19).

2. Read in the most recent almanac file, remove unhealthy satellites, and convert the almanac to an ephemeris format. (ex_gps.m, lines 33-43).

3. Propagate the GPS orbits (ex_gps.m, line 46).

4. Set the fixed receiver time tag and receiver velocity (ex_gps.m, lines 49-53). Note that the base station has a single row. When this is input to pseudo_r.m to compute pseudo-ranges, the receiver will be assumed to be fixed ECEF coordinates and data will be generated for all of the GPS satellite times.

5. Generate PR and Doppler measurements for the base station (ex_gps.m, lines 69-80). Model SA, troposphere, ionosphere, receiver clock, and receiver noise. Do not model the other effects.

6. Compute the line-of-sight vectors that correspond to the PR using the indices returned from pseudo_r.m (ex_gps, line 88). Rotate this LOS into NED coordinates and compute the azimuth and elevation of the GPS satellites (ex_gps, lines 92-96).

7. Apply the base station masking and remove all of the non-visible data (ex_gps.m, lines 99-109).

8. Compute a position and velocity solution for the receiver using the PR measurements computed in step 5 with lsnav.m (ex_gps.m, line 119).

9. Compute the DOPs for this receiver using ned2dops.m (ex_gps.m, line 125).

10. Compute position and velocity errors for the receiver and rotate these errors to the NED frame (ex_gps.m, lines 133-139).

11. Plot a sampling of the data (ex_gps.m, lines 143-end).

Calculation of simulated velocity follows an identical formulation, and is shown in data flow form in Figure 4.

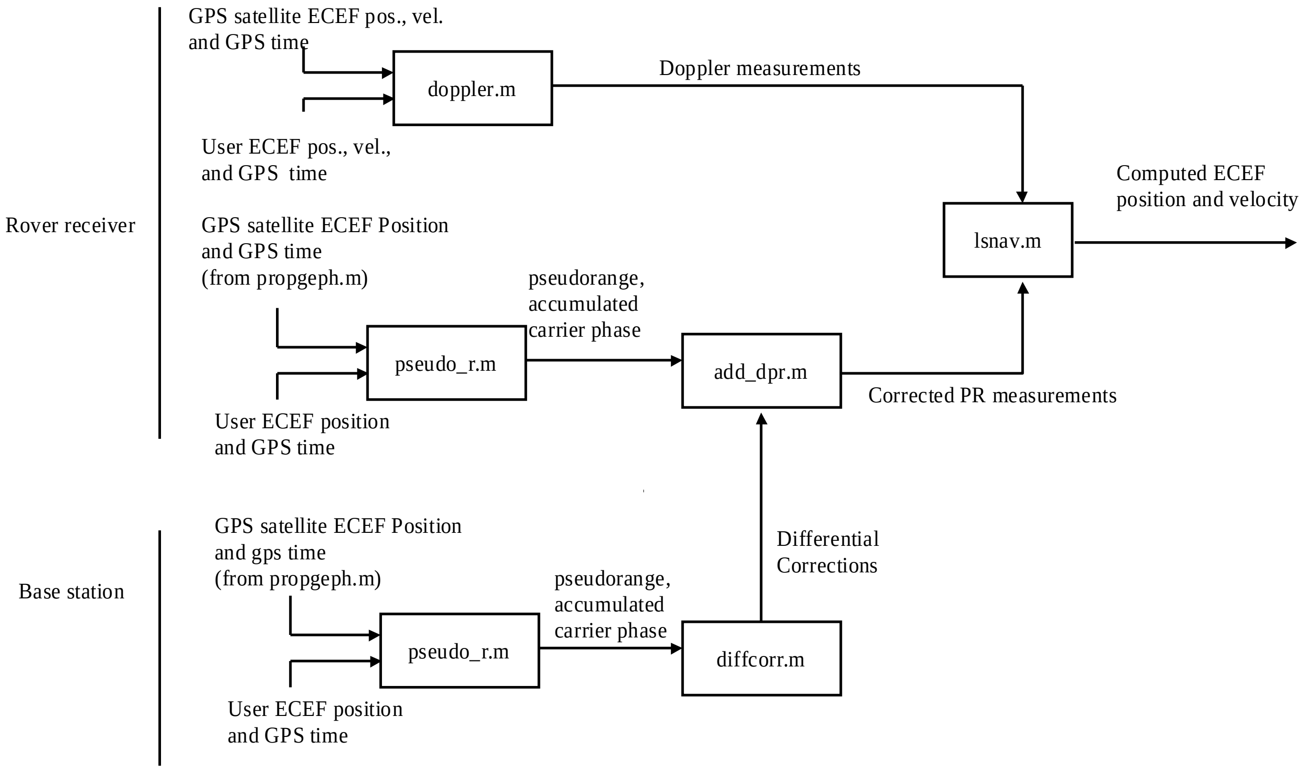


Figure 4: Simulated Position and Velocity Measurements from a GPS Receiver

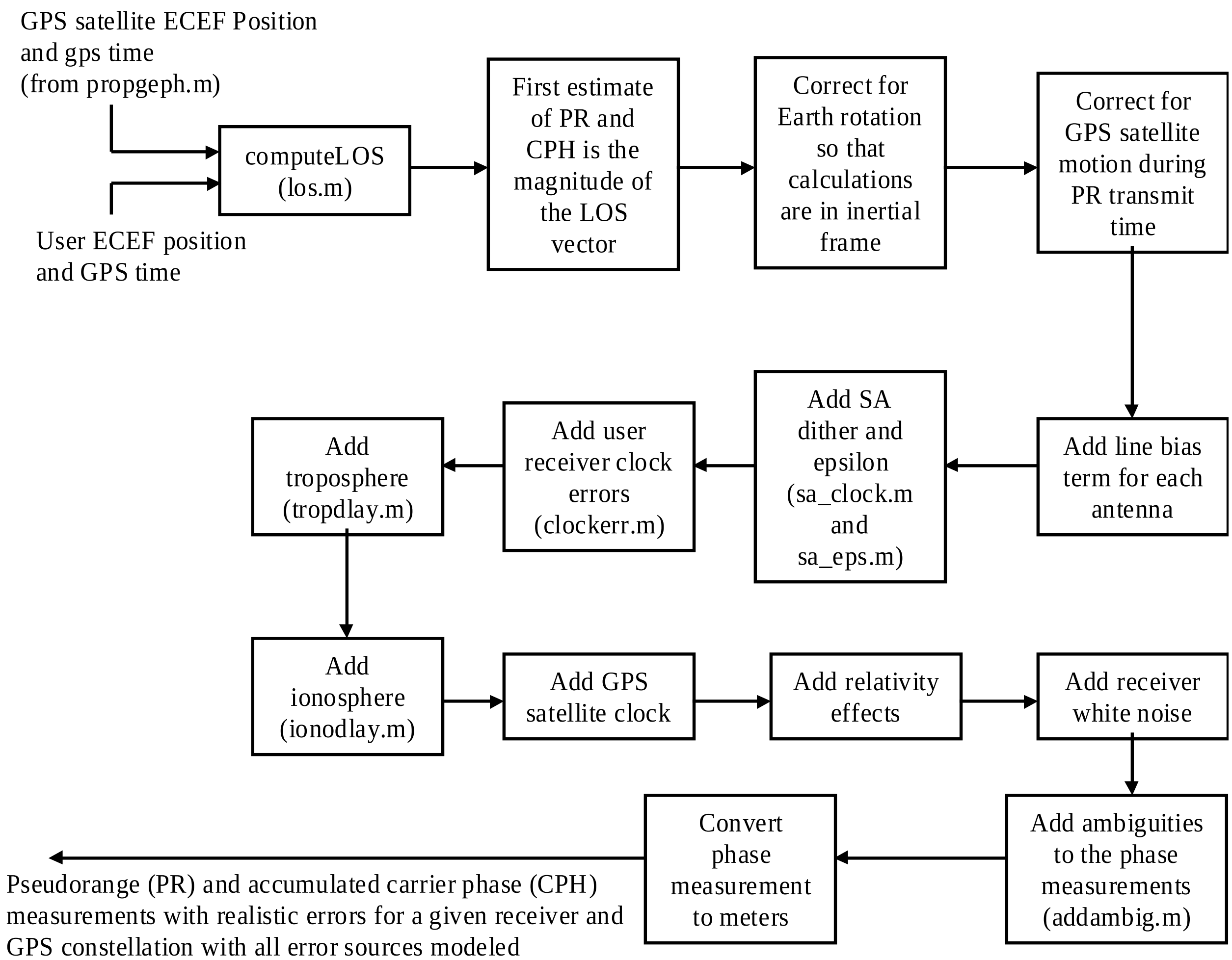


Figure 5: Simulating PR measurements with the Constellation Toolbox (function pseudo_r.m)

```
% ex_gps.m
%
% Example script for generating GPS measurements and computing position
% and velocity solutions using the raw data. This example is for a static
% receiver fixed to the surface of the Earth.

% Written by: Jimmy LaMance
% Copyright (c) 1998 by Constell, Inc.

% functions called: UTC2GPS, FIND_ALM, READYUMA, ALM2GEPH,
%                   PROPGEPH, ECEF2NED, PSEUDO_R, ECEF2LLA,
%                   NED2AZEL, VIS_DATA, DOPPLER, LSNAV,
%                   LOS, NED2DOPS, DOPS2ERR, PLOTPASS

% Variables to be used in the examples
start_time = [2003 7 4 0 0 0];        % UTC start time
duration = 1*3600;                    % 1 hour duration
base_station = [40*pi/180 255*pi/180 2000];      % base station coordinates

sample_rate = 20;                     % data sampling every 20 seconds

% Conversion from degree to radians
d2r = pi / 180;

% Compute the stop time based on the start time and duration
% first convert the start time to GPS time
[startweek startsec startday roll] = utc2gps(start_time);
if roll==0,
   start_gps=[startweek startsec roll];
   stop_gps=[start_gps(1) start_gps(2)+duration roll];
else
   start_gps=[startweek startsec];
   stop_gps=[start_gps(1) start_gps(2)+duration];
end

% check for week roll-overs
if stop_gps(2) > 604800
  stop_gps(1) = stop_gps(1) + fix(stop_gps(2) / 604800);
  stop_gps(2) = start_gps(2) + rem(stop_gps(2), 604800);
end % if stop_gps(2) > 604800

% Find the almanac that is most recent to the start time
alm_file = find_alm(start_gps(1));

% Read in the almanac
alm = readyuma(alm_file);

% Sort out the unhealthy satellites
I_healthy = find(alm(:,2) == 0);
alm = alm(I_healthy,:);

% Convert from almanac to ephemeris format
gps_ephem = alm2geph(alm);

% Compute satellite positions in ECEF
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,start_gps,stop_gps,sample_rate);
```

```
% Set the remote time matrix to be the same as the base station
t_base = t_gps(1,:);

% Convert the base and remote station coordinates to ECEF
x_base = ones(size(t_base,1),1) * lla2ecef(base_station);
v_base = zeros(size(x_base));

% Generate PR measurements for the base and remote stations using default
% values for masking, SA modeling, and random number seeding
% for the base station, force the modeling to have no receiver clock error
% to simulate the base station capability
base_mask = 10*d2r;

% Model SA, tropo, iono, and receiver clock.  Do not model satellite motion,
% Earth rotation, satellite clocks, line baises, or relativity
base_model = [0 0 1 1 1 1 0 0 0 0 0];   % turn all all error models
base_code_noise = .2;
base_carrier_noise = .01;
base_seed = 0;
max_latency = 2;           % set the maximum latency for a differential corr.

[t_pr_base,prn_base,pr_base,pr_errors_base,base_ndex] = ...
            pseudo_r(t_base,x_base,v_base,t_gps,[prn_gps x_gps],...
                  v_gps,base_model,base_seed, ...
                  base_code_noise, base_carrier_noise);

% Compute Doppler measurements at the site to be used
% in computing velocity solutions
doppler_model = [1 1 1];
dop_noise = .3;
[t_dop,prn_dop,dopp,dop_orb,dop_err] = ...
    doppler(t_base,x_base,v_base,t_gps,[prn_gps x_gps],v_gps,...
          doppler_model,base_seed,dop_noise);

% Compute masking for the base station.  Start by compute the LOS from
% the base to the satellites in ECEF.  This is easily done using the indices
% returned from PSEUDO_R where the LOS has been vectorized.  Masking is
% done external to PSEUDO_R so that it can be performed in any coordinate
% system.  This example is for a station on the surface of the Earth with
% the antenna boresight pointed up.  Therefore, the NED system is used.
los_base_ecef = x_gps(base_ndex(:,2),:) - x_base(base_ndex(:,1),:);

% Rotate the LOS to NED, using the base station as the reference for the NED
% coordinate system and rotation.
ref_lla = ecef2lla(x_base(base_ndex(:,1),:));
los_base_ned = ecef2ned(los_base_ecef,ref_lla);

% Compute the azimuth/elevation of the NED vectors
[az, el] = ned2azel(los_base_ned);

% Apply the masking in the NED coordinate system
[visible_data, I_pass] = vis_data(base_mask,[az el]);

% Remove all of the base station data that did not pass the masking test
t_pr_base = t_pr_base(I_pass,:);
```

```
prn_base = prn_base(I_pass,:);
pr_base = pr_base(I_pass,:);
dopp = dopp(I_pass,:);
dop_orb = dop_orb(I_pass,:);
pr_errors_base = pr_errors_base(I_pass,:);
base_ndex = base_ndex(I_pass,:);
x_ned = los_base_ned(I_pass,:);

% Rename the pr_errors for easier use later
pr_sa_errb = pr_errors_base(:,1) + pr_errors_base(:,2);
trop_base = pr_errors_base(:,3) + pr_errors_base(:,4);
iono_base = pr_errors_base(:,5);
clk_biasb = pr_errors_base(:,6);
clk_driftb = pr_errors_base(:,7);

% Compute a position and velocity solution at the remote site without DGPS
[t_nav,x_nav,num_remote,nav_index,v_nav] = ...
     lsnav(t_pr_base,pr_base(:,1),...
           [prn_base x_gps(base_ndex(:,2),:)],[x_base(1,:) 0],dopp, ...
            dop_orb(:,4:6),[0 0 0 0]);

% Compute DOPs at the remote station (with masking for the remote station)
[dops, t_dops, num_base] = ned2dops(x_ned,t_pr_base);

% Estimate position errors from DOPs
sigma_pr = 40;
[pos_err_dops] = dops2err(dops,sigma_pr);

% Compute the position errors with and without DGPS
% these vectors are in ECEF
pos_err = x_nav(:,1:3) - ones(size(x_nav,1),1) * x_base;
vel_err = v_nav(:,1:3) - ones(size(v_nav,1),1) * v_base;

% Rotate position difference to NED relative to the truth location
% for easier interpretation
[pos_err_ned] = ecef2ned(pos_err, ecef2lla(x_base));
[vel_err_ned] = ecef2ned(vel_err, ecef2lla(x_base));

% Generate a figure with 2 plots, one with the uncorrected position errors
% and one with differentially corrected position solutions
plot_time = t_nav(:,2) - t_nav(1,2);  % simple time in seconds

% generate the position error figure
clear fh
fh(1) = figure;
subplot(2,1,1)
plot(plot_time,pos_err_ned)
legend('North','East','Down',-1);
ylabel('Position Err (m)')
title_string = ...
  sprintf('Position Errors without DGPS for a Static Receiver');
title(title_string)

subplot(2,1,2)
plot(vel_err_ned)
legend('North','East','Down',-1);
```

```
ylabel('Velocity Err (m/s)')
title_string = ...
  sprintf('Velocity Errors without DGPS for a Static Receiver');
title(title_string)
xlabel('time past start (sec)')


% plot troposphere contribution to the PR error
fh(2) = plotpass(t_pr_base,trop_base,prn_base,...
        'Example of Troposphere Effects on Pseudorange Measurements',...
        'Tropo (m)');

% plot ionosphere contribution to the PR error
fh(3) = plotpass(t_pr_base,iono_base,prn_base,...
        'Example of Ionosphere Effects on Pseudorange Measurements',...
        'Iono (m)');

% plot the receiver clock bias for a single satellite
% the receiver clock bias is common to all satellites
I = find(prn_base == prn_base(2));
fh(4) = plotpass(t_pr_base(I,:),clk_biasb(I),prn_base(I),...
        'Example of Receiver Clock Bias on Pseudorange Measurements',...
        'Clock Bias (m)');

fh(5) = plotpass(t_pr_base(I,:),clk_driftb(I),prn_base(I),...
        'Example of Receiver Clock Drift on Pseudorange Measurements',...
        'Clock Drift (m/2)');

% generate a plot of the resulting position error from the DOPs
t_dop_plot = (t_dops(:,1) * 604800 + t_dops(:,2)) - ...
           (t_dops(1,1) * 604800 + t_dops(1,2));
fh(6) = figure;
plot(t_dop_plot,pos_err_dops);
legend('PDOP Error','HDOP Error','VDOP Error',0);
ylabel('Position Err (m)')
title('Estimated Position Errors Computed from DOPs')
xlabel('time past start (sec)')

% Stack the figures such that they are not on top of each other
x_wide = .4;
y_high = .4;
x_start = .03;
y_loc = .05;
x_step = .07;
for i = 1:length(fh)
  x_loc = x_start + (i-1) * x_step;
  pos = [x_loc y_loc x_wide y_high];
  set(fh(i),'Units','normalized');
  set(fh(i),'Position',pos);
end % for i = 1:length(fh)
```

End of Example 4

Example 5: Simulated Raw Measurements from a Differential GPS Base Station and a Static Rover Receiver

This fifth example demonstrates how the toolbox can be used to generate simulated pseudo-range (PR), accumulated carrier phase, and Doppler (raw GPS) measurements. This example is for two station fixed to the surface of the Earth. One station is modeled as a GPS differential base station and the other as a rover receiver. Models for Selective Availability (SA), the troposphere, ionosphere, receiver clock, receiver noise, GPS satellite clocks, GPS satellite movement during signal propagation, Earth rotation, and relativity can be modeled to achieve raw GPS measurements that are very realistic. This example also shows the improvements for differential GPS positioning over standard GPS positioning.

The data flow diagram for this application is shown in Figure 4. This is the standard flow for computing GPS measurements, differential corrections, and navigation solutions.

The formulation for computing simulated position measurements from a fixed ground station is shown below.

1. Define the simulation start and stop times as well as the GPS receiver locations. (ex_dgps.m, lines 17-20).

2. Read in the most recent almanac file, remove unhealthy satellites, and convert the almanac to an ephemeris format. (ex_dgps.m, lines 34-44).

3. Propagate the GPS orbits (ex_dgps.m, line 47).

4. Set the base station time tag and base station velocities (ex_dgps.m, lines 50-57). Note that the base station and rover time/position/velocity matrices have a single row. When this is input to pseudo_r.m to compute pseudo-ranges, these will be assumed to be fixed ECEF coordinates and data will be generated for all of the GPS satellite times.

5. Generate PR measurements for the base station (ex_dgps.m, lines 78-82). Model SA, troposphere, ionosphere, receiver clock, and receiver noise. Do not model the other effects.

6. Compute the line-of-sight vectors that correspond to the PR using the indices returned from pseudo_r.m (ex_dgps, line 89). Rotate this LOS into NED coordinates and compute the azimuth and elevation of the GPS satellites (ex_dgps, lines 93-98).

7. Apply the base station masking and remove all of the non-visible data (ex_dgps.m, lines 101-108).

8. Compute PR for the rover receiver with different noise characteristics from the base staiton (ex_dgps.m, line 127). Rotate to NED and mask, same as for the base station (ex_dgps.m, lines 145-167).

9. Compute position and velocity for the rover receiver using the raw PR and Doppler measurements with lsnav.m (ex_dgps.m, line 177).

10. Compute differential corrections based on the known location of the base station using the diffcorr.m function (ex_dgps.m, line 189).

11. Apply the differential corrections to the rover PR measurements using add_dpr.m (ex_dgps.m, line 199).

12. Compute a navigation solution for the rover receiver using the differentially corrected PR measurements with lsnav.m (ex_dgps.m, line 203).

13. Use the add_dpr.m function to find the common troposphere and ionosphere errors for plotting later (ex_dgps.m, lines 212-219) . This is a useful way to find how much troposphere and ionosphere (or other error source) are common to the base and rover receivers.

14. Compute position errors for the remote receiver and rotate these errors to the NED frame (ex_dgps.m, lines 242-248).

15. Plot a sampling of the data (ex_dgps.m, lines 255-end).

```
% ex_dgps.m
%
% Example script for generating GPS measurements and computing position
% and velocity solutions using the raw data. This example is for a static
% receiver fixed to the surface of the Earth with a base station located
% approximately 250 miles away.

% Written by: Jimmy LaMance
% Copyright (c) 1998 by Constell, Inc.

% functions called: UTC2GPS, FIND_ALM, READYUMA, ALM2GEPH,
%                   PROPGEPH, ECEF2NED, PSEUDO_R, ECEF2LLA,
%                   NED2AZEL, VIS_DATA, DOPPLER, LSNAV, DIFFCORR,
%                   ADD_DPR, LOS, NED2DOPS, DOPS2ERR, PLOTPASS

% Variables to be used in the examples
start_time = [2006 4 17 0 0 0];      % UTC start time
duration = 1*3600;                   % 1 hour duration
base_station = [40*pi/180 255*pi/180 2000];     % base station coordinates
remote_station = [42*pi/180 254*pi/180 2000];   % remote station coordinates

sample_rate = 20;                 % data sampling every 20 seconds

% Conversion from degree to radians
d2r = pi / 180;

% Compute the stop time based on the start time and duration
% first convert the start time to GPS time
[startweek startsec startday roll] = utc2gps(start_time);
if roll==0,
   start_gps=[startweek startsec roll];
   stop_gps=[start_gps(1) start_gps(2)+duration roll];
else
   start_gps=[startweek startsec];
   stop_gps=[start_gps(1) start_gps(2)+duration];
end
% check for week roll-overs
if stop_gps(2) > 604800
  stop_gps(1) = stop_gps(1) + fix(stop_gps(2) / 604800);
  stop_gps(2) = start_gps(2) + rem(stop_gps(2), 604800);
end % if stop_gps(2) > 604800

% Find the almanac that is most recent to the start time
alm_file = find_alm(start_gps(1));

% Read in the almanac
alm = readyuma(alm_file);

% Sort out the unhealthy satellites
I_healthy = find(alm(:,2) == 0);
alm = alm(I_healthy,:);

% Convert from almanac to ephemeris format
gps_ephem = alm2geph(alm);

% Compute satellite positions in ECEF
```

```
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,start_gps,stop_gps,sample_rate);

% Set the remote time matrix to be the same as the base station
t_base = t_gps(1,:);
t_remote = t_base;

% Convert the base and remote station coordinates to ECEF
x_base = ones(size(t_base,1),1) * lla2ecef(base_station);
v_base = zeros(size(x_base));
x_remote = ones(size(t_remote,1),1) * lla2ecef(remote_station);
v_remote = zeros(size(x_remote));

% Compute the base line length
base_line_ecef = x_base(1,:) - x_remote(1,:);
base_line_ned = ecef2ned(base_line_ecef,ecef2lla(x_base(1,:)));
base_line_km = norm(base_line_ned) / 1000;

% Generate PR measurements for the base and remote stations using default
% values for masking, SA modeling, and random number seeding
% for the base station, force the modeling to have no receiver clock error
% to simulate the base station capability
base_mask = 10*d2r;

% Model SA, tropo, iono, and receiver clock.  Do not model satellite motion,
% Earth rotation, satellite clocks, line baises, or relativity
base_model = [1 1 1 1 1 1 0 0 0 0 0];   % turn all all error models
base_code_noise = .2;
base_carrier_noise = .01;
base_seed = 0;
max_latency = 2;            % set the maximum latency for a differential corr.

[t_pr_base,prn_base,pr_base,pr_errors_base,base_ndex] = ...
              pseudo_r(t_base,x_base,v_base,t_gps,[prn_gps x_gps],...
                     v_gps,base_model,base_seed, ...
                     base_code_noise, base_carrier_noise);

% Compute masking for the base station.  Start by compute the LOS from
% the base to the satellites in ECEF.  This is easily done using the indices
% returned from PSEUDO_R where the LOS has been vectorized.  Masking is
% done external to PSEUDO_R so that it can be performed in any coordinate
% system.  This example is for a station on the surface of the Earth with
% the antenna boresight pointed up.  Therefore, the NED system is used.
los_base_ecef = x_gps(base_ndex(:,2),:) - x_base(base_ndex(:,1),:);

% Rotate the LOS to NED, using the base station as the reference for the NED
% coordinate system and rotation.
ref_lla = ecef2lla(x_base(base_ndex(:,1),:));

los_base_ned = ecef2ned(los_base_ecef,ref_lla);

% Compute the azimuth/elevation of the NED vectors
[az, el] = ned2azel(los_base_ned);

% Apply the masking in the NED coordinate system
[visible_data, I_pass] = vis_data(base_mask,[az el]);

```

```
% Remove all of the base station data that did not pass the masking test
t_pr_base = t_pr_base(I_pass,:);
prn_base = prn_base(I_pass,:);
pr_base = pr_base(I_pass,:);
pr_errors_base = pr_errors_base(I_pass,:);
base_ndex = base_ndex(I_pass,:);

% Rename the pr_errors for easier use later
pr_sa_errb = pr_errors_base(:,1) + pr_errors_base(:,2);
trop_base = pr_errors_base(:,3) + pr_errors_base(:,4);
iono_base = pr_errors_base(:,5);
clk_biasb = pr_errors_base(:,6);
clk_driftb = pr_errors_base(:,7);

remote_mask = 5*d2r;
% Model SA, tropo, iono, and receiver clock.  Do not model satellite motion,
% Earth rotation, satellite clocks, line baises, or relativity.  The LSNAV
% function used to compute navigation solutions does not currently support
% this PR modeling.  However, these effects could be modeled and then absorbed
% into the differential correction.
remote_model = [1 1 1 1 1 1 0 0 0 0 0];
remote_code_noise = 1;
remote_carrier_noise = .01;
remote_seed = 0;
[t_pr_remote,prn_remote,pr_remote,pr_errors_remote,remote_ndex] = ...
          pseudo_r(t_remote,x_remote,v_remote,t_gps,[prn_gps x_gps],v_gps,...
                remote_model,remote_seed,...
                remote_code_noise, remote_carrier_noise,gps_ephem);

% Save the remote orbits associate with this PR data into it's own
% matrix for easier booking keeping later.
pr_orb_remote = x_gps(remote_ndex(:,2),:);

% Compute Doppler measurements at the remote site to be used
% in computing velocity solutions
remote_doppler_model = [1 1 1];
remote_dop_noise = .3;
[t_dop,prn_remote_dop,dopp,dop_orb,dop_err] = ...
   doppler(t_remote,x_remote,v_remote,t_gps,[prn_gps x_gps],v_gps,...
      remote_doppler_model,remote_seed,remote_dop_noise);

% Compute masking for the remote receiver.  Here again, the NED system is used.
los_remote_ecef = x_gps(remote_ndex(:,2),:) - x_remote(remote_ndex(:,1),:);

% Rotate the LOS to NED, using the base station as the reference for the NED
% coordinate system and rotation.
ref_lla = ecef2lla(x_remote(remote_ndex(:,1),:));

los_base_ned = ecef2ned(los_base_ecef,ref_lla);

% Compute the azimuth/elevation of the NED vectors
[az, el] = ned2azel(los_base_ned);

% Apply the masking in the NED coordinate system
[visible_data, I_pass] = vis_data(remote_mask,[az el]);
```

```
% Remove all of the base station data that did not pass the masking test
t_pr_remote = t_pr_remote(I_pass,:);
prn_remote = prn_remote(I_pass,:);
pr_remote = pr_remote(I_pass,:);
pr_orb_remote = pr_orb_remote(I_pass,:);
pr_errors_remote = pr_errors_remote(I_pass,:);
remote_ndex = remote_ndex(I_pass,:);
dopp = dopp(I_pass,:);
dop_orb = dop_orb(I_pass,:);

% Rename the pr_errors for easier use later
pr_sa_errr = pr_errors_remote(:,1) + pr_errors_remote(:,2);
trop_remote = pr_errors_remote(:,3) + pr_errors_remote(:,4);
iono_remote = pr_errors_remote(:,5);
clk_biasr = pr_errors_remote(:,6);
clk_driftr = pr_errors_remote(:,7);

% Compute a position and velocity solution at the remote site without DGPS
[t_nav,x_nav,num_remote,nav_index,v_nav] = ...
    lsnav(t_pr_remote,pr_remote(:,1),...
          [prn_remote x_gps(remote_ndex(:,2),:)],[x_remote(1,:) 0],dopp, ...
           dop_orb(:,4:6),[0 0 0 0]);

% Compute differential corrections given the base station locations.  Model
% the GPS satellite clocks, relativistic effects, satellite motion, and
% Earth rotation.  SA, ionosphere, troposphere, and receiver clock are not
% modeled in the PR for computing differential corrections.  If a term is
% not modeled in the correction, but was included in the PR measurement, it
% will effectively be absorbed into the differential correction.
diff_model = [0 0 0 0 0 0 0 0 0 0 0];
[t_dpr, dpr] = ...
       diffcorr(t_pr_base, [prn_base pr_base], x_gps(base_ndex(:,2),:), ...
                v_gps(base_ndex(:,2),:), x_base(1,:),diff_model,gps_ephem);

% Set the differential correction rate of change to zero
dprr = zeros(size(dpr,1),1);

% Add the corrections to the remote PR measurements
% the return variable cpr_index is an index into the input remote
% site pseudoranges that have differential corrections applied to them
[t_cpr, prn_cpr, cpr, cpr_index] = ...
      add_dpr(t_pr_remote,[prn_remote pr_remote],t_dpr,dpr,dprr,max_latency);

% Compute a LS nav solotuion at the remote site with DGPS
[t_nav_dgps,x_nav_dgps,num_dgps] = ...
    lsnav(t_cpr,cpr(:,1),[prn_remote(cpr_index) pr_orb_remote(cpr_index,:)],...
          [x_remote(1,:)+50 0]);

% Compute residual troposphere errors using the ADD_DPR function.  this function
% handles all of the common visibility analysis.  Instead of pseudoranges and
% pseudorange corrections, we will send it troposphere errors.  To obtain the
% troposphere differences, we will send in the negative of one set of tropo
% errors.
[t_dtrop, prn_dtrop, dtrop, dtrop_index] = ...
      add_dpr(t_pr_remote,[prn_remote trop_remote],t_pr_base,...
             [prn_base -trop_base],dprr,0);
```

```
% Compute the residual ionospheric errors the same way
[t_diono, prn_diono, diono, diono_index] = ...
     add_dpr(t_pr_remote,[prn_remote iono_remote],t_pr_base,...
          [prn_base -iono_base],dprr,0);

% Compute LOS at remote station in ECEF frame
[t_los, los_vect, los_ind] = los(t_gps(1,:), x_remote(1,:), t_gps, [prn_gps x_gps]);

veh_num = prn_gps(los_ind(:,2));

% Rotate ECEF los to NED
[x_ned] = ecef2ned(los_vect, ecef2lla(x_remote(1,:)));

% Compute masking
[az, el] = ned2azel(x_ned);
[az_el, I_pass] = vis_data(remote_mask, [az el]);

% Compute DOPs at the remote station (with masking for the remote station)
[remote_dops, t_dops, num_base] = ned2dops(x_ned(I_pass,:),t_los(I_pass,:));

% Estimate position errors from DOPs
sigma_pr = 40;
[pos_err_dops] = dops2err(remote_dops,sigma_pr);

% Compute the position errors with and without DGPS
% these vectors are in ECEF
pos_err = x_nav(:,1:3) - ones(size(x_nav,1),1) * x_remote;
pos_err_dgps = x_nav_dgps(:,1:3) - ones(size(x_nav,1),1) * x_remote(:,1:3);

% Rotate position difference to NED relative to the truth location
% for easier interpretation
[pos_err_ned] = ecef2ned(pos_err, ecef2lla(x_remote));
[pos_err_ned_dgps] = ecef2ned(pos_err_dgps, ecef2lla(x_remote));

% Generate a figure with 2 plots, one with the uncorrected position errors
% and one with differentially corrected position solutions
plot_time = t_nav(:,2) - t_nav(1,2);  % simple time in seconds

% generate the position error figure
clear fh
fh(1) = figure;
subplot(2,1,1)
plot(plot_time,pos_err_ned)
legend('North','East','Down',-1);
ylabel('Position Err (m)')
title_string = ...
  sprintf('Position Errors without DGPS with a %d km Baseline',...
         round(base_line_km));
title(title_string)

subplot(2,1,2)
plot(pos_err_ned_dgps)
legend('North','East','Down',-1);
ylabel('Position Err (m)')
title_string = ...
```

```
  sprintf('Position Errors with DGPS with a %d km Baseline',...
          round(base_line_km));
title(title_string)
xlabel('time past start (sec)')

% plot troposphere contribution to the PR error
fh(2) = plotpass(t_pr_base,trop_base,prn_base,...
        'Example of Troposphere Effects on Pseudorange Measurements',...
        'Tropo (m)');

% plot differential troposphere contribution to the differential PR error
fh(3) = plotpass(t_dtrop,dtrop,prn_dtrop,...
        'Example of Differential Troposphere Effects on Pseudorange Measurements',...
        'Delta Tropo (m)');

% plot ionosphere contribution to the PR error
fh(4) = plotpass(t_pr_base,iono_base,prn_base,...
        'Example of Ionosphere Effects on Pseudorange Measurements',...
        'Iono (m)');

% plot differential ionosphere contribution to the differential PR error
fh(5) = plotpass(t_diono,diono,prn_diono,...
        'Example of Differential Ionosphere Effects on Pseudorange Measurements',...
        'Delta Iono (m)');

% plot the receiver clock bias for a single satellite
% the receiver clock bias is common to all satellites
I = find(prn_base == prn_base(2));
fh(6) = plotpass(t_pr_base(I,:),clk_biasb(I),prn_base(I),...
        'Example of Receiver Clock Bias on Pseudorange Measurements',...
        'Clock Bias (m)');

fh(7) = plotpass(t_pr_base(I,:),clk_driftb(I),prn_base(I),...
        'Example of Receiver Clock Drift on Pseudorange Measurements',...
        'Clock Drift (m/2)');

% generate a plot of the resulting position error from the DOPs
t_dop_plot = (t_dops(:,1) * 604800 + t_dops(:,2)) - ...
             (t_dops(1,1) * 604800 + t_dops(1,2));

fh(9) = figure;
plot(t_dop_plot,pos_err_dops);
legend('PDOP Error','HDOP Error','VDOP Error',0);
ylabel('Position Err (m)')
title('Estimated Position Errors Computed from DOPs')
xlabel('time past start (sec)')

% Stack the figures such that they are not on top of each other
x_wide = .4;
y_high = .4;
x_start = .03;
y_loc = .05;
x_step = .07;

for i = 1:length(fh)
  x_loc = x_start + (i-1) * x_step;
```

```
  pos = [x_loc y_loc x_wide y_high];
  set(fh(i),'Units','normalized');
  set(fh(i),'Position',pos);
end % for i = 1:legnth(fh)

if exist('save_plot_data')
  % save the data to a mat file for regeneration later
  save demoplt1 plot_time pos_err_ned pos_err_ned_dgps pr_base ...
       t_pr_base pr_sa_errb clk_biasb clk_driftb trop_base trop_remote ...
       iono_base iono_remote ...
       prn_base base_line_km remote_dops t_dop_plot pos_err_dops ...
       t_nav num_remote num_base num_dgps ...
       t_diono prn_diono diono ...
       t_dtrop prn_dtrop dtrop
end % if save_plot_data == 1

```

End of Example 5

Example 6: DGPS Modeling and Large Simulation Time Chunking for a Vehicle with Varying Attitude and Body-Fixed Masking

The sixth example demonstrates many aspects of utilizing the Toolbox. This example demonstrates how to a) compute DGPS measurements for a receiver in an aircraft in body coordinates, b) how to how to break a simulation into time chunks, and c) how to change almanac data during a simulation. This example is an extension of the aircraft visibility given in Example 3. The antenna is fixed to the aircraft body and masking of the tail is modeled.

The airplane trajectory begins at the equator, flying North at 200 m/s (450 mph) for 60 sec. The airplane rolls 45 deg to the right and turns for 360 deg of heading (about 128 sec). The airplane then rolls back to zero and continues to fly North until 300 sec. have elapsed since the beginning of the flight. Data is stored at one second time steps. A block diagram of the data flow and function calling sequence is shown in Figure 3. The example MATLAB file, vis_ac.m, is shown on the three pages following Figure 3.

The simulation is broken into several time steps to demonstrate the chunking concept. In addition, at each time step the almanac data is changed. This is an effective way to simulate satellites becoming unhealthy during a simulation. The time chunking does not have to be done on almanac changes and can be achieved using a number of methods.

The formulation for computing the raw GPS measurements, common view satellites with the base station, masking in body coordinates, and navigation solutions is given below.

1. Set up the simulation start and stop times (exdgpsac.m, lines 58-63) and define the different almanacs to be used in each time chunk (exdgpsac.m, lines 66-100).

2. Load in the aircraft data file and assign absolute times to the aircraft data (exdgpsac.m, lines 103-112).

3. Define the base station location and velocity (exdgpsac.m, lines 116-117).

4. Initialize the data matrices to store the output from the simulation (exdgpsac.m, lines 135-144).

5. Start looping over the number of time chunks (exdgpsac.m, line 147).

6. Set the start and stop times for this time chunk, convert the appropriate almanac to an ephemeris, and propagate the GPS orbits. (exdgpsac.m, lines 151-172).

7. Compute PR measurements for the base station (exdgpsac.m, line 177).

8. Compute base station masking and remove all of the non-visible data (exdgpsac.m, lines 184-202).

9. Compute differential correction for the base station PR data (exdgpsac.m, line 209).

10. Compute PR measurements for the aircraft receiver (exdgpsac.m, line 221).

11. Compute body fixed masking for the aircraft PR measurements (exdgpsac.m, lines 244-281).

12. Compute a navigation solution without differential correction (exdgpsac.m, line 284).

13. Apply the differential corrections (exdgpsac.m, line 288).

14. Compute a navigation solution with differential correction (exdgpsac.m, line 292).

15. Compute DOPs (exdgpsac.m, line 297).

16. Code to write data from each time step to a data file. This code is commented. To utilize this functionality, uncomment the desired lines of code (exdgpsac.m, lines 316-328).

17. Plot data from the run showing uncorrected and corrected GPS PR measurements (exdgpsac.m, lines 334-end).

```
% exdgpsac.m
%
% Function to demonstrate body-fixed masking, efficient breaking up of
% the vectorized functions to conserve memory, changing GPS almanacs,
% and computing DGPS coverage and navigation solutions for an aircraft
% which models aircraft atitude and it's effects on DGPS satellite visibility.
%
% This example has a large body-fixed masking (20 degress over about 1/2 of the
% sky) to demonstrate the effects of bad satellite geometry.  The effects
% on the differentially corrected solutions and the non-differentially
% corrected solution is apparent.  DGPS errors in excess of 20 meters
% and raw GPS errors in excess of 600 meters are shown.  These large
% errors result from bad geometry.  To test this, set the aircraft masking
% to -10 deg over the entire azimuth and see the improvement. However, because
% of the aircraft maneuvers and the body-fixed masking, the satellite geometry
% (observed as DOP values) is still poor.
%
% An aircraft is modeled with an initial location at ECEF = (6378e3,0,0).  It
% has a ground speed of 200 m/s (450 mph).  The aircraft flies straight and
% level North for 60 sec, then banks to the right by 45 deg and executes a
% 360 deg. turn at constant altitude (about 128 sec).  At the end of the turn,
% the airplane levels back out and continues to fly North, straight and level,
% until 300 secs have elapsed since the beginning of the flight.  Aircraft
% position, velocity, and attitude data are stored in a file at 1 sec intervals.
% This data is read in and used in this example.
%
% A GPS antenna is mounted to the top of the airplane.  It is masked by the
% fuselage (elevation < 0) and by the vertical tail (elevation < 30 deg for
% an azimuth range of 170 to 190 deg, relative to the nose of the airplane).
% A 5 deg mask of the Earth is also used.

% Written by: Jimmy LaMance
% Copyright (c) 1998 Constell, Inc.

clear       % clear all variables in workspace
close all  % close all open windows

% Set simulation parameters
d2r = pi/180;
mask_base = 0;            % simple 0 deg elevation Earth Mask (radians)
mask_b = [0 190 170;       % 0 elevation mask of fuselage between 190-170 deg
          20 170 190]*d2r;  % 30 deg elevation mask of tail between 170-190 deg

% Input some receiver modeling parameters
ac_rec_code_noise = 3;       % 3 meters of code noise, typical mid quality C/A code
ac_rec_carrier_noise = .1;   % 10 cm of carrier noise

% Set up the PR modeling for the aircraft.  Model SA epsilon, SA dither,
% troposphere, ionosphere, receiver clock, and receiver white noise (1's).
% Do not model line bias, Earth rotation, satellite motion,
% satellite clocks, or relativity.  This set of modeling will be used to
% simulate PR measurements for both the base station and the aircraft.
ac_pr_err_model = [0 0 1 1 1 1 0 0 0 0 0];   % model all PR errors

% Set the seed value to use in generate the PR model errors
seed = 0;
```

```
sim_start_time_utc = [2006 4 17 0 0 0];   % Start Date (yr, mon, day, hr, mn, sec)
sim_stop_time_utc = [2006 4 17 0 5 0];    % Stop Date (yr, mon, day, hr, mn, sec)
sim_start_time_gps = utc2gps(sim_start_time_utc);  % Sim start in GPS time
sim_stop_time_gps = utc2gps(sim_stop_time_utc);    % Sim stop in GPS time
sim_start_time_sec = gpst2sec(sim_start_time_gps); % Sim start in GPS seconds
sim_stop_time_sec = gpst2sec(sim_stop_time_gps);   % Sim stop in GPS seconds

time_step = 1;          % Time step (sec)
gps_start_time = utc2gps(sim_start_time_utc);
alm_file ; find_alm(gps_start_time(1));        % GPS almanac file to be used here

% Set up the parameters to chunk up the run into multiple sections
% to add the capability of changing almanac/ephemeris data sets during
% the simulation.  This allows for setting specific satellite healthy
% and un-healthy during the simulation.  This same technique can be used
% break up a simulation into time chunk to save RAM for very large simulations.
% See the example function for the large constellation broken into
% time chunk while maintaining the vecotrization.

% Use Matlab 5 structures for readibility in almanac manipulation.  Start
% by developing 4 separate almanacs, all with the same satellite information.
almanac_data(1).alm = readyuma(alm_file);
almanac_data(2).alm = almanac_data(1).alm;
almanac_data(3).alm = almanac_data(1).alm;
almanac_data(4).alm = almanac_data(1).alm;

% Define start times for each almanac.  The start time will be used
% to determine when this almanac is switched into the simulation.  Beacuse
% this is a short simulation (300 seconds), almanac start times will be
% from the beginning of the sim (ie 0 -> 300 seconds).
almanac_data(1).start_time = 0;      % Start this alm at the beginning
almanac_data(2).start_time = 60;     % Start this alm at 1 minute
almanac_data(3).start_time = 240;    % Start this alm at 4 minutes
almanac_data(4).start_time = 250;    % Use this alm for the rest

% Set some satellites unhealthy for the different almanacs.  The satellite
% ID is in column #1 of the almanac, the health bit is coulmn #2.  The
% value of 0 is healthy, anything else is unhealthy. For this example,
% we will set satellites 17 and 30 unhealty.
I_17 = find(almanac_data(1).alm(:,1) == 17);
I_30 = find(almanac_data(1).alm(:,1) == 30);
almanac_data(2).alm(I_17,2) = 1;    % set prn 17 unhealthy in almanac 2
almanac_data(3).alm(I_17,2) = 1;    % set prn 17 unhealthy in almanac 3
almanac_data(4).alm(I_30,2) = 1;    % set prn 30 unhealthy in almanac 4

% Read in airplane data:  time, position, velocity, attitude
load airplane.dat;                 % aircraft data in an ascii file
ac_time = airplane(:,1);           % aircraft relative time: 0-300 sec
ac_pos_all = [airplane(:,2:4)];    % aircraft ECEF xyz position (m)
ac_vel_all = [airplane(:,5:7)];    % aircraft ECEF xyz velocity (m/s)
ac_att_all = [airplane(:,8:10)];   % aircraft attitude wrt NED (deg)
ac_att_all = ac_att_all*d2r;       % convert to radians

% Convert aircraft relative time to gps time, using start_time as time0
ac_time = ac_time + sim_start_time_sec;   % absolute aircraft time (secs)
```

```
ac_time_s = sec2gpst(ac_time);           % GPS time version of ac_time

% Set the base station location to be at the aircraft starting location.
base_loc_ecef = ac_pos_all(1,:);

% Compute the base station location in LLA coordinates.
base_lla = ecef2lla(base_loc_ecef);

% Move the base station away from the aircraft starting position.
% 1 deg at the equator ~ 110km at the equator
base_lla = base_lla + [0*d2r 1*d2r -100];    % offset 5 deg in long, -100m in height

% Convert the base location back to ECEF
base_loc_ecef = lla2ecef(base_lla);

base_vel = zeros(size(base_loc_ecef));

% Find out how many chunks to break the simulation into.
num_chunks = size(almanac_data,2);

%%%%% BEGIN ALGORITHM CODE %%%%%

% Initialize variables for storing the output for all the chunks
t_nav_gps_all = [];
t_nav_dgps_all = [];
x_nav_gps_all = [];
x_nav_dgps_all = [];

t_vis_all = [];
t_dops_all = [];
dops_all = [];
num_sats_all = [];
num_dgps_all = [];

% Loop over each data chunk
for ijk = 1:num_chunks

  % Set the start and stop time for this chunk
  this_start_sec = sim_start_time_sec + almanac_data(ijk).start_time;
  if ijk < num_chunks
    this_stop_sec = sim_start_time_sec + ...
                almanac_data(ijk+1).start_time - time_step;
  else
    this_stop_sec = sim_stop_time_sec;
  end % if ijk < num_chunk

  start_gps = sec2gpst(this_start_sec);
  stop_gps = sec2gpst(this_stop_sec);

  % Load the GPS almanac for the given almanac week
  alm_2_use = almanac_data(ijk).alm;

  % Sort out the unhealthy satellites
  I_gps_good = find(alm_2_use(:,2) == 0);
  alm_2_use = alm_2_use(I_gps_good,:);
```

```
% Convert the almanacs to ephemeris format
[gps_ephem] = alm2geph(alm_2_use);

% Compute satellite positions in ECEF frame for the given time range and interval
[t_gps,prn_gps,x_gps,v_gps] = propgeph(gps_ephem,start_gps,stop_gps,time_step);

% Compute pseudo-range measurements for the base station.  Get the indicies
% used to compute the PR (which base station position goes with which
% GPS satellite position) and the Earth obscure information.
[t_pr_base,prn_base,pr_base,pr_base_err,prn_ndex] = ...
          pseudo_r(start_gps,base_loc_ecef, base_vel, ...
          t_gps,[prn_gps x_gps],v_gps,ac_pr_err_model,seed);

% Use the indices from PSEUDO_R to get the base station and GPS satellite
% positions for use in creating the LOS.  This will be used to compute
% satellite masking.
base_los = x_gps(prn_ndex(:,2),:) - base_loc_ecef(prn_ndex(:,1),:);

% Convert the ECEF LOS to NED
base_los_ned = ecef2ned(base_los,ecef2lla(base_loc_ecef,1));

% Compute azimuth and elevation from the NED vectors
[az, el] = ned2azel(base_los_ned);

% Compute masking in the NED frame.  Since this is a base station, no obscure
% information needs to be used.  Instead a simple az/el masking for the
% antenna pattern is modeled.
[visible_data, I_pass] = vis_data(mask_base, [az el]);

% Eliminate the PR data not passing the masking test
t_pr_base = t_pr_base(I_pass,:);
prn_base = prn_base(I_pass,:);
pr_base = pr_base(I_pass,:);
pr_base_err = pr_base_err(I_pass,:);
prn_ndex = prn_ndex(I_pass,:);

% Compute differential corrections at the base station.  Do not use any
% pseudo-range models (SA, atmospheric, satellite motion, etc.).  All
% of these parameters that are modeled in the raw PR measurements will be
% effectively removed from the base station PR with the differential
% corrections.
[t_dpr, dpr] = diffcorr(t_pr_base, [prn_base pr_base], ...
                 x_gps(prn_ndex(:,2),:), v_gps(prn_ndex(:,2),:),...
                 base_loc_ecef);

% Compute pseudo-ranges for the aircraft receiver.  Use increased code and
% carrier noise to more closely imitate an aircraft type receiver.
% Model the base station and remote PR at the same time in the simulation
% and with the same seed.  Otherwise, the SA errors, because of the algorithms
% used to generate them, will not be common.  If the SA errors are not common,
% the DGPS solutions will not have any meaning. Also get the Earth obscuring
% information back from PSEUDO_R.  This will allow the Earth to be masked
% out with VIS_DATA.  The default Earth obscuring model is a spherical Earth.
[t_pr_ac,prn_ac,pr_ac,pr_ac_err,prn_ndex,obscure_info] = ...
          pseudo_r(ac_time_s, ac_pos_all, ac_vel_all, ...
          t_gps,[prn_gps x_gps],v_gps,...
```

```
            ac_pr_err_model,seed,ac_rec_code_noise, ...
            ac_rec_carrier_noise);

    % Rename some variables for ease of use later
    prn_los = prn_gps(prn_ndex(:,2));
    ac_pos = ac_pos_all(prn_ndex(:,1),:);
    gps_pos = x_gps(prn_ndex(:,2),:);
    I_ac = prn_ndex(:,1);
    I_gps = prn_ndex(:,2);
    los_ac = gps_pos - ac_pos;

    % Build up an attitude matrix that uses the input matrix, ac_att, but
    % is sync'd to the LOS vectors.  This is necessary since the LOS matrix
    % is 4-12 times the size of the ac_att matrix.  The LOS matrix has several
    % vectors at the same time point, and we need the attitudes to be correlated
    % with these LOS vectors.  A large attitude matrix, ac_att_large, is built
    % from ac_att using I_ac from losorbit.
    ac_att = ac_att_all(I_ac,:);  % make the large attitude matrix

    % Convert LOS in ECEF to NED
    [los_ned] = ecef2ned(los_ac, ecef2lla(ac_pos));

    % Compute az and el in Earth frame
    [az_e el_e] = ned2azel(los_ned);

    % Rotate the Earth-visible LOS's into the airplane body frame
    los_body = ned2body(los_ned,ac_att); % Earth vis. LOSs in b frame

    % Convert LOS's in body frame to body-referenced az and el's
    % using ned2azel here since it is the same as body2azel.  This assumes that the
    % azimuth is defined relative to the body x-axis and elevation is relative
    % the the x-y body plane.
    [az_b,el_b] = ned2azel(los_body);

    % Apply the body-fixed masking model to the body az and el's.  Use the
    % obscure information from the PSEUDO_R routine to eliminate data that is
    % masked by the Earth.  The default (no input for minimum tangent altitude)
    % is at the surface of the Earth.  See vis_ac for using a minimum tangent
    % altitude with the obscure ifnromation.
    [azel_vis, I_vis_b] = vis_data(mask_b,[az_b el_b],obscure_info);

    % Use this index of visible satellites in the body frame to build the LOS data
    % set that is visible in the body frame, with all the maskings applied.
    % Express this LOS data in the NED frame, however, to keep the DOPS and
    % Visibility calculations in the local-level frame.
    if any(I_vis_b),
      t_pr_ac = t_pr_ac(I_vis_b,:);     % PR times for the aircraft data
      prn_ac = prn_ac(I_vis_b);         % PR PRN for the aircraft data
      pr_ac = pr_ac(I_vis_b,:);         % PR for the aircraft data
      pr_orb_ac = gps_pos(I_vis_b,:);   % PR orbits for the aircraft data
      los_ned = los_ned(I_vis_b,:);     % LOS vectors, in NED, visible in body
      los_body = los_body(I_vis_b,:);   % LOS vectors, in body frame, visible in body
      prn_los = prn_los(I_vis_b);       % visible GPS PRN
      gps_pos = gps_pos(I_vis_b,:);     % visible GPS orbit positions
    else
      fprintf('No visibile satellites during aircraft body masking.\n');
```

```
        return
      end;

      % Compute a navigation solution with these PR measurements, no DGPS
      [t_nav_gps,x_nav_gps,num_dgps] = ...
        lsnav(t_pr_ac,pr_ac(:,1),[prn_ac pr_orb_ac],[ac_pos_all(1,:)+50 0]);

      % Apply the differential corrections
      [t_cpr, prn_cpr, cpr, cpr_index] = ...
         add_dpr(t_pr_ac,[prn_ac pr_ac],t_dpr,dpr);

      % Compute a differential GPS (DGPS) navigation solution
      [t_nav_dgps,x_nav_dgps,num_dgps] = ...
        lsnav(t_cpr,cpr(:,1),[prn_ac(cpr_index) pr_orb_ac(cpr_index,:)],...
              [ac_pos_all(1,:) 0]);

      % Finally, compute DOPS and number of satellites tracked
      [dops,t_dops,num_sats] = ned2dops(los_ned,t_pr_ac);       % no masking

      [az_eb,el_eb] = ned2azel(los_body);       % az and el's for the body masking

      % Save the data for this time/almanac chunk
      t_nav_gps_all = [t_nav_gps_all; t_nav_gps];
      t_nav_dgps_all = [t_nav_dgps_all; t_nav_dgps];
      x_nav_gps_all = [x_nav_gps_all; x_nav_gps];
      x_nav_dgps_all = [x_nav_dgps_all; x_nav_dgps];

      % Store the data to be stored for all of the chunks.  These variables are
      % initialized before the start of the chunking loop.  This is where you add
      % additional variable to be stored for output.  Be sure to initialize
      % the same way these are.
      t_vis_all = [t_vis_all; t_dops];
      num_sats_all = [num_sats_all; num_sats];
      num_dgps_all = [num_dgps_all; num_dgps];
      t_dops_all = [t_dops_all; t_dops];
      dops_all = [dops_all; dops];

      % Writing out data to a file...
      % To write all of the data out to a file uncomment the following code
        %write_string = sprintf('save exdata%02d.mat',ijk);
        %eval(write_string);

      % To write a part of the data out to a file uncomment the following code
      % and add the variables you're interested in.
         %write_string = sprintf('save exdata%02d.mat t_vis_all dops_all',ijk);
         %eval(write_string);

      % To read in this data and put it back together, use the load command
      % and the same concatenation procedure as above to generate matrices with
      % all of the data.

    end % for ijk = 1:num_chunks

    %%% Plotting Section %%%
    % Verify that there are common satellites visibile at every time step
    if (size(num_dgps_all,1) ~= size(num_sats_all))
```

```
    fprintf('There are not common view satellites at each time step.\n')
    fprintf('Move the base station closer to the aircraft trajectory.\n')
    fprintf('No plots will be generated.\n');
    return;
  end % if (size(num_dgps_all,1) ~= size(num_sats_all))

  % generate the plot of visible satellites versus time
  fh(1) = plotpass(t_vis_all,num_dgps_all,ones(size(num_sats_all)),...
                     'Number of Common View (Base & A/C) Satellites', '# Visible');

  % plot DOPS
  fh(2) = figure;
  tm = gpst2sec(t_dops_all);       % time past GPS epoch (1980 in seconds)
  tm = (tm - tm(1))/ 60;           % time past start in minutes
  plot_handle = plot(tm,dops_all);
  title('DOPS for Non-Differential Aircraft Navigation Solution')
  xlabel('Time Past Aircraft Epoch (min)')
  ylabel('DOPS')
  legend('GDOP','PDOP','HDOP','VDOP','TDOP',0);

  % Plot GPS errors
  gps_err = x_nav_gps_all(:,1:3) - ac_pos_all;
  gps_err_ned = ecef2ned(gps_err,ecef2lla(ac_pos_all));
  fh(3) = figure;
  tm = gpst2sec(t_nav_gps_all);
  tm = (tm - tm(1))/60;
  plot(tm, gps_err_ned);
  title('GPS Errors for Aircraft Flight Profile')
  xlabel('Time Past Aircraft Epoch (min)')
  ylabel('meters')
  legend('North','East','Down',0)

  % Plot GPS errors
  dgps_err = x_nav_dgps_all(:,1:3) - ac_pos_all;
  dgps_err_ned = ecef2ned(dgps_err,ecef2lla(ac_pos_all));
  fh(4) = figure;
  plot(tm, dgps_err_ned);
  title('DGPS Errors for Aircraft Flight Profile')
  xlabel('Time Past Aircraft Epoch (min)')
  ylabel('meters')
  legend('North','East','Down',0)

  % Stack the figures such that they are not on top of each other
  x_wide = .6;
  y_high = .6;
  x_start = .03;
  y_loc = .08;
  x_step = .1;

  for i = 1:length(fh)
    x_loc = x_start + (i-1) * x_step;
    pos = [x_loc y_loc x_wide y_high];
    set(fh(i),'Units','normalized');
    set(fh(i),'Position',pos);
  end % for i = 1:length(fh)

```

```
%%% End of plotting

% end of exdgpsac.m

```

End of Example 6

Example 7: Mechanization of a Receiver Satellite Selection Algorithm

The seventh example, shown in Figure 6, is only a data flow diagram suggesting a way to mechanize a receiver satellite selection algorithm. The line-of-sight vectors are computed in the usual way, in the NED frame with masking. A selection algorithm is then applied to the visible satellites. In many cases, all satellites are used. In other cases, a subset of the visible satellites is used for tracking, in which case, a selection algorithm must be implemented to choose the desired subset. These selection algorithms are generally dependent on the specific receiver hardware used. One example is to use the 4 line-of-sights that produce the best DOP values.

End of Example 7.

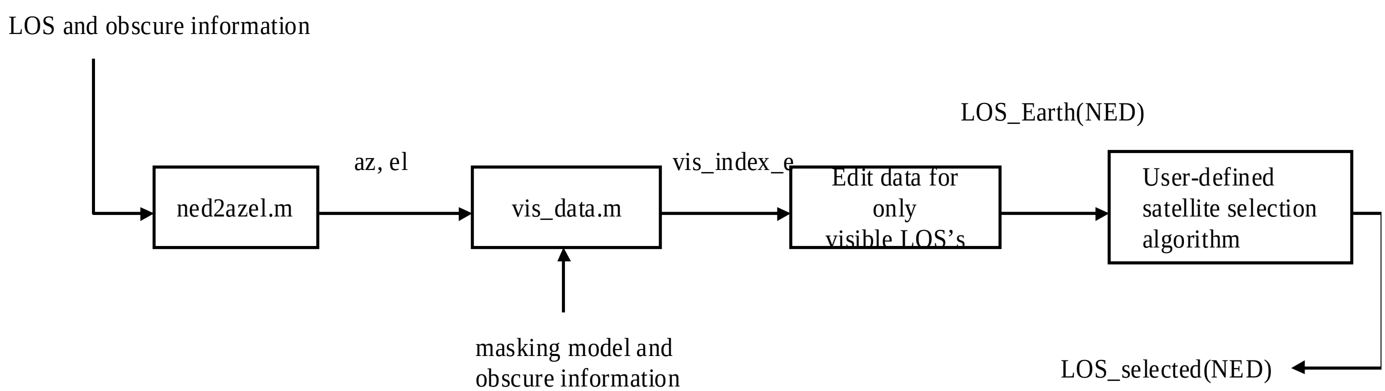


Figure 6: Implementation of a GPS Receiver Satellite Selection Algorithm

Examples and General Support Function Summary

contents	Display contents of the toolbox.
err_chk	Input variable error checking.
ex_dgps	Example script for differential GPS positioning.
exdgpsac	Example for differential GPS for an aircraft.
ex_dop_m	Example program to generate a movie of GPS DOP values.
ex_gps	Example program for generating GPS data nd computing solutions for a statis receiver fixed to the surface of the Earth.
ex_nmea	Example program for reading and plotting NMEA data.
normvect	Normalize n-dimensional vectors to have length of 1.
vis_ac	Example program demonstrating the visibility and DOPs routines for a receiver mounted on an aircraft
vis_e	Example program demonstrating the visibility and DOPs routines for observers fixed on the Earth
vis_o	Example function demonstrating the visibility and DOPS routines for orbiting observers.

YUMA Almanacs, Ephemeris Formats, and Satellite Propagation Function Summary

alm2geph	Converts YUMA almanacs to GPS ephemeris format.
find_alm	Search the Matlab path for the most recent GPS and/or GLONASS almanacs.
kep2geph	Converts from a Keplerian 6-element set to GPS ephemeris format (22 column format).
keplr_eq	Iteratively solve Kepler's equation for eccentric anomaly.
propgeph	Computes satellite positions in ECEF coordinates from a GPS ephemeris
readyuma	Read in YUMA formatted GPS and/or GLONASS almanacs.
writyuma	Create a YUMA formatted almanac file.

Visibility Function Summary

los	Compute line-of-sight (LOS) vectors from a group of objects to another group of objects.
num_vis	Computes the number of satellites visible as a function of time.
passdata	Determines pass numbers for data that has passed the masking tests in **vis_data**.
vis_data	Finds azimuth/elevation pairs passing the masking tests and returns the GPS data corresponding to those azimuth/elevation pairs.
vis_e	Example program demonstrating the visibility and DOPs routines for observers fixed on the Earth
vis_o	Example function demonstrating the visibility and DOPS routines for orbiting observers.

Graphical Output Function Summary

makeplot	Function to create the five most used plots, azimuth, elevation, sky plot, number of visible satellites, and DOPS for a single object.

orb_anim	Animate the orbits and stations over a Mercator map of the Earth.
playmov	Plays Matlab movies in a new figure window while retaining the original window size and optionally the original color map.
plotpass	Plot data pertaining to passes over observers(azimuth, elevation, etc.)
plotsky	Create a polar plot of the azimuth/elevation (sky plot) for a single observer of a constellation of satellites.
writemov	Play a Matlab movie and write it to an MPEG movie file. Retains the original window size and color map properties.
writempg	Supporting function for writemov to write an MPEG file.

Dilution of Precision Function Summary

dops2err	Converts from Dilution of Precision (DOP) to equivalent position error.
ll2dops	Compute DOP values from Local Level (LL) LOS vectors.
ned2dops	Compute DOP values from NED LOS vectors.

Position Error Analysis and Simulation Function Summary

addambig	Adds a random set of integer ambiguities (N) to the accumulated phase (CPH) measurements.
clockerr	Simulates the user clock bias and drift of a GPS receiver.
doppler	Computes Doppler measurements given a user trajectory and the GPS/GLONASS satellite positions and velocities.
ionodlay	Computes the ionospheric group delay for the L1 frequency.
lsnav	Computes a least squares PR navigation (position) solution or a least squares position and velocity solution if the Doppler data is provided.
pseudo_r	Computes pseudorange (pr) and accumulated carrier phase (cph) measurements given a user trajectory and the GPS and/or GLONASS satellite positions.
sa_clock	Simulate the S/A clock dither contribution to the pseudo-range (PR) and PR rate (PRR) models using a 2nd order Gauss-Markov process.
sa_eps	Simulate the epsilon contribution of S/A to the pseudo-range (PR) and accumulated carrier phase (CPH) measurement error using the RTCA standard model.
tropdlay	Hopfield model to compute the wet and dry troposphere delays.
tropunb1	Univeristy of New Brunswick troposphere model for altitude dependent wet and dry troposphere delays.

Data Processing Function Summary

nmeanext	Get the next field in an NMEA message string.
parsegga	Parsing function for NMEA GGA messages.
parsegsa	Parsing function for NMEA GSA messages.
parsegsv	Parsing function for NMEA GSV messages.
parsnmea	Driver parsing function for NMEA messages.
readnmea	Read NMEA data from a file.
readyuma	Read in YUMA formatted GPS and/or GLONASS almanacs.
writyuma	Create a YUMA formatted almanac file.

Differential Processing Function Summary

add_dpr	Adds differential correction to pseudo-range (PR) measurements.
diffcorr	Computes differential corrections.

Coordinate Transformation Function Summary

azel2ned	Converts from azimuth and elevation to North-East-Down (NED) coordinates.
body2ned	Converts from body coordinates to NED coordinates
ecef2eci	Convert position and velocity vectors from Earth Centered Earth Fixed (ECEF) to Earth Centered Inertial (ECI) coordinates.
ecef2ll	Convert vectors from ECEF to Local Level (LL) coordinates.
ecef2lla	Compute geodetic latitude, longitude, and altitude from ECEF coordinates.
ecef2ned	Convert vectors from ECEF to North-East-Down (NED) coordinates.
eci2ecef	Convert position and velocity vectors from ECI to ECEF coordinates.
eci2ll	Convert vectors from ECI to LL coordinates.
ll2ecef	Convert vectors from Local Level (LL) to ECEF coordinates.
ll2eci	Convert vectors from Local Level (LL) to Earth-Centered-Inertial (ECI) coordinates.
lla2ecef	Compute ECEF coordinates from geodetic latitude, longitude, and altitude.
ned2azel	Convert a North, East, Down (NED) vector into azimuth and elevation.
ned2body	Transform a vector from the NED local-level frame to a vehicle body frame.
ned2ecef	C vectors from NED coordinates to ECEF coordinates.
sidereal	Computes the Greenwich sidereal time (GST) from UTC time.

Time Transformation Function Summary

gps2utc	Converts GPS time into equivalent UTC time.
gpst2sec	Utility function to convert a GPS time structure to linear seconds.
leapsecs.dat	Data file containing UTC time of each leap second increment.
sec2gpst	Convert from 1-dimensional linear GPS seconds to a two-dimensional GPS weeks and GPS seconds of week structure.
utc2gps	Converts a UTC time matrix to the equivalent time in GPS weeks, GPS seconds, and GPS days.
utc2leap	Determines the number of leap seconds of offset between GPS and UTC time.

Alphabetical Listing of Functions

Each function is listed alphabetically with descriptions of inputs, outputs, algorithms, and references. It is a more complete description of each function than is found by typing help function name at the command prompt.

add_dpr

Purpose

Applies differential corrections to pseudo-range (PR) and accumulated carrier phase (CPH) measurements.

Syntax

[t_cpr, prn_cpr, cpr, cpr_index] = add_dpr(t_pr, pr, t_dpr, dpr, *dprr*, *max_latency*)

Input:

t_pr = time associated with the PR, [GPS_week GPS_sec] (nx2) or [GPS_week GPS_sec rollover_flag] (nx3). Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].
pr = satellite number and PR (and CPH) for each t_pr [prn pr] (nx2) or [prn pr cph] (nx3). (pr is in meters and cph is in cycles (or same units as dpr).
t_dpr = time associated with the differential correction (DPR), [GPS_week GPS_sec] (kx2) or [GPS_week GPS_sec rollover_flag] (kx3)
dpr = satellite number and DPR (and dCPH) for each t_dpr [dprn dpr] (kx2). dpr units are meters and dcph is in cycles (or same units as pr).
dprr = *optional* differential correction rate associated with the corresponding dpr and t_dpr (kx1) default: dprr = 0, rate is in meters/s (or same units as pr per second).
max_latency = *optional* maximum latency for differential corrections to be applied (s), default = 10

Output:

t_cpr = time associated with the corrected pseudo-range (CPR), [GPS_week GPS_sec] (mx2) or [GPS_week GPS_sec rollover_flag] (mx3)
prn_cpr = satellite number for base station CPRs (mx1)
cpr = pseudo-range correction associated with the corresponding t_cpr and prn_cpr (mx1) or (mx2) if carrier phase corrections are computed.
cpr_index = index matrix that relates the corrected pseudo-ranges to the input pr measurements.

Description

Differential corrections are used to remove common errors in GPS measurements by using a base station at a known location. The most common standard for differential corrections is the Radio Technical Commission for Maritime Service (RTCM) Recommended Standards developed by Special Committee Number 104. This function applies the PR correction using the formulation from the RTCM Recommended Standard Version 2.1 for correction types 1 and 9.

Algorithm

Differential corrections are seldom available at the precise time the PR measurement is taken. Therefore a method of mapping PR correction forward in time was developed by the RTCM as follows.

$$DPR(t) = DPR(t_0) + DPRR * [t - t_0]$$

DPR is the differential correction computed at the base station at time t_0, and DPRR is the estimated rate of change of the differential correction.

The correction PR measurement is then computed as

$$PR_{corr}(t) = PR_{raw}(t) + DPR(t)$$

See Also

diffcorr, pseudo_r, sa_clock, clockerr

Notes This function can also be used to apply accumulated carrier phase (cph) differential corrections. Substitute the cph measurements and corrections for the pr measurements and corrections. Do not use a correction rate when applying correction for the cph measurements.

References "RTCM Recommended Standards for Differential NAVSTAR GPS Service", Version 2.1, January 3, 1994. Developed by RTCM Special Committee No. 104.

addambig

Purpose Adds a random set of integer ambiguities (N) to the accumulated carrier phase (CPH) measurements. The ambiguities will between -1e6 and 1e6.

Syntax [cphnew, ambig] = addambig(cph, *seed*);

Input:

cph = satellite number and associated CPH measurement to apply N ambiguities (nx2)
[prn cph] (cycles)
seed = *optional* seed value for random number generator.
Default = 0.

Output:

cphnew = CPH measurements with N integer ambiguities added (nx1) (cycles)
ambig = N integer ambiguities (nx1)

Description The value of the N ambiguities will be between $-1x10^{-6}$ and $1x10^{6}$ and will be constant for a given satellite. No simulation of loss-of-lock is done within this routine. The equation for accumulated carrier phase is

$$cph = \left| \vec{X}_{GPS} - \vec{X}_{user} \right| + SA_{epsilon} + SA_{dither} + B_{rx\_clock} + T - I + N + \varepsilon_{carrier}$$

This function computes the N in the above equation.

See Also **pseudo_r, sa_clock, clockerr, tropdlay**

alm2geph

Purpose Converts from an almanac format to a GPS ephemeris format.

Syntax [gps_ephem] = alm2geph(alm)

Input:

alm = Yuma almanac format (nx13), with columns of [ID, health, e, epoch t0, i, asc_rate, sqrt(a), long. of asc_node at weekly epoch, perigee, M, af0, af1, epoch week] units are seconds, meters, and radians

Output:

gps_ephem = ephemeris matrix for all satellites (nx22), with columns of [prn, M0, delta_n, e, sqrt_a, long. of asc_node at weekly epoch, i, perigee, ra_rate, i_rate, Cuc, Cus, Crc, Crs, Cic, Cis, Toe, IODE, GPS_week, Af0, Af1, perigee_rate].

Description GPS ephemeris format is needed by **propgeph** for propagation of orbits. See description of **propgeph** for more details. Ephemeris parameters are from ICD-GPS-200 with the exception of perigee_rate. Perigee rate is set to zero for almanac data. See **kep2geph** for use of this parameter with Keplerian elements. The gps_ephem output variable will be filled with inf values for any almanac element set that is out bounds.

See Also **propgeph, kep2geph**

References ICD-GPS-200

azel2ned

Purpose Convert azimuth and elevation angles into a North-East-Down unit vector

Syntax [ned] = azel2ned(az, el)

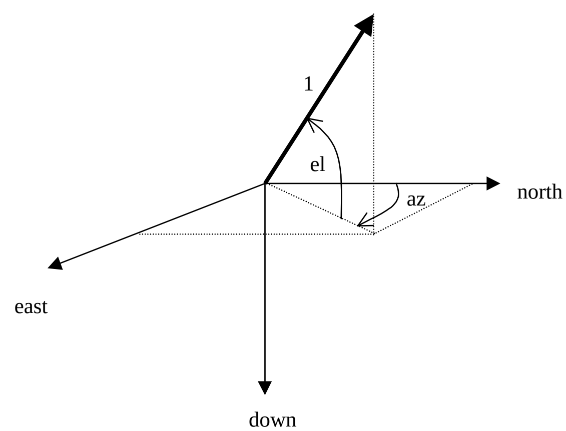
Inputs:
    az = vector azimuth (radians) (nx1).
        Valid values are -2*pi -> 2*pi
    el = vector elevation (radians) (nx1
        Valid values are -pi/2 -> pi/2
Output:
    ned = unit vector in local North, East, and Down coordinates (nx3)

Description Azimuth is the rotation angle of a vector about local vertical in the local-horizontal plane. Zero azimuth corresponds to North, East is pi/2, etc. Elevation is the rotation angle of a vector above or below the local-horizontal plane (positive or negative elevation angles respectively).

See the diagram below.

See Also **ned2azel**

$$n = north = \cos(el)\cos(az)$$
$$e = east = \cos(el)\sin(az)$$
$$d = down = -\sin(el)$$



body2ned

Purpose Transform a vector from a vehicle body frame to the North-East-Down local-level frame.

Syntax [x_ned] = body2ned(x_body, euler)

Inputs:

x_body = vector in body frame (nx3), n = number of input vectors
euler = 3-2-1 Euler angle sequence from NED to body frame (radians)

Note: The euler input can be either (1x3) or (nx3), where the 1st col. is yaw, 2nd col. is pitch, 3rd col. is roll. If 1x3, then the same Euler angles are used for each vector. If nx3, then one set of angles is used for each input vector.

Output:

x_ned = vector in NED frame [x_ned(x), x_ned(y), x_ned(z)] (nx3)

Description The body frame is related to the NED frame by a 3-2-1 euler angle rotation sequence. The rotation angles and axes of rotation are given by:

NED to NED' frame: yaw about Down (3rd or z axis)
NED' to Body' frame: pitch about E' (2nd or y axis)
Body' to Body frame: roll about body-x axis (1st axis)

The transformation from the body to NED frame is the reverse of the above rotation sequence, and is the inverse function of ned2body.

See Also **ned2body, ecef2ned, ned2ecef**

clockerr

Purpose Simulates the user clock bias and drift of a GPS receiver.

Syntax [clk_bias, clk_drift] = clockerr(t_pr, *clk_model*, *seed*);

Input:

t_pr = GPS time of pseudoranges (PR) (nx2)
[GPS_week GPS_sec] or [GPS_week GPS_sec rollover_flag] (nx3). Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

clk_model = *optional* input for the user clock model. 1x2 matrix in the form [S_b S_f] where the S_b and S_f are white noise spectral densities in s and s/s. These values can easily be related to Allan variances (see algorithm section for details). Default is based on a typical crystal oscillator with values of [4e-19 1.58e-18]

seed = *optional* seed value for random number generator.
Default = 0.

Output:

clk_bias = User clock bias (m) (nx1)
clk_drift = User clock drift rate (m/s) (nx1)

Description The clock in a GPS receiver has a bias and drift that are characteristic of the oscillator used in the receiver. The bias and drift of the clock effect the generation of raw receiver measurements (pseudorange, accumulated carrier phase, and Doppler). To generate raw measurements that have error characteristics that represent real data, the receiver clock should be modeled.

Algorithm The clock model used is a 2nd order process with correlated white noise represented by the following equations.

$$\mathbf{x}_c = \Phi_c(\Delta t)\mathbf{x}_c(k-1) + \mathbf{w}_c(k-1)$$

Where

$$\mathbf{x}_c = \begin{bmatrix} b \\ f \end{bmatrix} \qquad \Phi_c(\Delta t) = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$Q_c = E[\mathbf{w}_c \mathbf{w}_c^T] = \begin{bmatrix} S_b \Delta t + S_f \dfrac{\Delta t^3}{4} & S_f \dfrac{\Delta t^2}{2} \\ S_f \dfrac{\Delta t^2}{2} & S_f \Delta t \end{bmatrix}$$

The parameters S_b and S_f can be related to Allend variance parameters using the following equations

$$S_f = 2h_0 \qquad S_b = 8\pi^2 h_{-2}$$

See Also **pseudo_r**, **doppler**, **sa_eps**, **sa_clock**, **tropdlay**, **ionodlay**

References "Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 417-418.

contents

Purpose Display all of the files contained in the CONSTELLATION Toolbox.

Syntax help contents or help constellation (or the directory holding the CONSTELLATION Toolbox)

Description A one line description of each file is written to the screen when **help contents** is entered.

diffcorr

Purpose Compute differential pseudorange (PR) corrections (DPR) at a base station with known coordinates.

Syntax [t_dpr, dpr] = diffcorr(t_pr, pr, pr_orb, pr_orb_vel, base, *base_model, ephem*);

Input:

t_pr = GPS time associated with the PR collected at a base station [GPS_week GPS_sec] (nx2) or [GPS_week GPS_sec rollover_flag] (nx3). Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

pr = satellite number and PR for base station pseudo-ranges (nx2) [prn pr] PR is in meters, or (nx3) if using carrier phasedata [prn pr cph] where chp is in cycles

pr_orb = GPS satellite positions as a function of time associated with this base station PR (nx3) (meters)

pr_orb_vel = GPS satellite velocities at t_pr times (nx3) (m/s)

base = base station vector location (1x3) (meters)

base_model = *optional* flags controlling which contributions to the PR errors are modeled (1x11). [sa_eps dither trophosphere ionosphere receiver_clock receiver_noise line_bias sat_motion sat_clock earth_rotation relativity]. A value of 1 indicates useage of that model. Use values of 2 to implement user supplied models. See the code for where to insert the user models. A warning is given if a user model is selected and none is supplied. Default = [0 0 0 0 0 0 0 0 0 0 0].

ephem = *optional* ephemeris matrix for all satellites (nx24). Used to compute satellite clock. If not provided, no GPS satellite clock effects will be computed.

Note: Coordinate systems for the base and pr_orb data must be the same. Either both in ECEF or ECI frame in meters.

Output:

t_dpr = GPS time associated with the differential correction, [GPS_week GPS_sec] (nx2) or [GPS_week GPS_sec rollover_flag] (nx3)

dpr = satellite number and differential correction (nx2) [prn_diff dpr] correction (dpr) in meters

Description Differential corrections are used to remove common errors in GPS measurements by using a base station at a known location. The most common standard for differential corrections is the Radio Technical Commission for Maritime Service (RTCM) Recommended Standards developed by Special Committee Number 104. This function computes the PR correction using the formulation from the RTCM Recommended Standard Version 2.1 for correction types 1 and 9.

The base station does not model the troposphere or ionosphere, and those errors are assumed to be common between the base and the remote receivers. The base station clock bias is removed to reduce the size of the corrections. However, this clock bias is not critical when using DGPS because the base station and remote clock bias are inseparable.

Algorithm The following equation is used to compute PR differential corrections.

$$dpr = \left| \vec{X}_{GPS} - \vec{X}_{user} \right| - PR_{measured} + B_{est\_rx\_clock}$$

Coordinate systems for the base and pr_orb data must be the same, either both in ECEF or ECI frame in meters.

See Also **add_dpr**, **pseudo_r**, **sa_clock**, **clockerr**

References "RTCM Recommended Standards for Differential NAVSTAR GPS Service", Version 2.1, January 3, 1994. Developed by RTCM Special Committee No. 104.

doppler

Purpose Computes Doppler measurements given a user trajectory and the GPS/GLONASS satellite positions and velocities.

Syntax [t_dop, prn, doppler, *dop_orb*, *dop_err*] = doppler(t_user, x_user, …
v_user, t_gps, x_gps, v_gps, *model*, *seed*, *dop_noise*)

Input:

t_user = GPS time vector for user trajectory [GPS_week, GPS_sec] (nx2)) or [GPS_week GPS_sec rollover_flag] (nx3). Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].
x_user = ECEF/ECI position vectors for the user vehicle [x,y,z] (nx3) (m)
v_user = ECEF/ECI vel. vectors for the user vehicle [xv,yv,zv] (nx3) (m/s)
t_gps = GPS time vector for GPS positions [GPS_week, GPS_sec] (mx2) or [GPS_week GPS_sec rollover_flag] (mx3)
x_gps = ECEF/ECI position vectors for GPS satellites [prn,x,y,z] (mx4) (m)
v_gps = ECEF/ECI velocity vectors for GPS satellites [xv,yv,zv] (mx3) (m/s)
model = *optional* flags controlling which contributions to the Doppler errors are modeled. (1x3) [sa_dither user_clock receiver_noise]. A value of 1 indicates usage of the model and a value of zero indicates no use of that model.
Default = [0 1 1].
seed = *optional* seed value for random number generator.
Default = 0.
dop_noise = *optional* 1 sigma estimate of the receiver Doppler noise. (1x1) (m/s).
Default = 0.3 m/s.

Output:

t_dop = GPS time associated with the Doppler measurements, [GPS_week GPS_sec] (kx2), or [GPS_week GPS_sec rollover_flag] (kx3),
k = num_time_steps x number of visible satellites
prn = satellite number for this Doppler measurement (kx1)
doppler = Doppler measurements associated with the corresponding t_dop and prn (kx1) (m/s)
dop_orb = *optional* GPS/GLONASS satellite positions and velocities associated with this measurement (kx3) (m, m/sec)
dop_err = optional modeled errors added to the Doppler measurement (m/s) (kx3) [sa_dither user_clock receiver_noise]

Description The Doppler measurement is sometimes used by GPS receivers for instantaneous velocity computations. The Doppler measurement is dependent on the user position, velocity, and clock state. However, the observation geometry for the Doppler measurements is not as strong as for range measurements.

Algorithm The Doppler measurement is constructed using the following equation.

$$\dot{\rho} = \frac{\vec{X}_{GPS} - \vec{X}_{user}}{\left|\vec{X}_{GPS} - \vec{X}_{user}\right|} \bullet (\vec{V}_{GPS} - \vec{V}_{user}) + f_{SA\,dither} + f_{rx\_clock} + \varepsilon_{Doppler}$$

The GPS satellite clock frequency offset is taken from the **sa_clock** function and the receiver clock offset is taken from the **clockerr** function. When generating coherent pseudorange, accumulated carrier phase, and Doppler measurements, the same seed value should be used when generating the measurements. This is the default case for the models as the seed value is set to zero if not provided.

See Also **pseudo_r**, **sa_eps**, **sa_clock**, **clockerr, tropdlay, ionodlay, vis_data**

References 'GPS: Theory and Practice', Hoffman-Wellenhoff, pages 92-93, 182.

"Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 411-412.

dops2err

Purpose Converts from Dilution of Precision (DOP) to equivalent position error.

Syntax [pos_err] = dops2err(dops, *sigma_pr*);

Input:

dops = DOP values (nx5) [GDOP PDOP HDOP VDOP TDOP]
sigma_pr = *optional* sigma on the pseudo-range measurements (1x1) (m). Default = 30.

Output:

pos_err = estimate of the position error based on the DOPs and the sigma on the pseudo-range (nx3) [total_pos_err horizontal_err vertical_err]

Description This is a first order approximation that takes the DOP value and multiplies by the sigma on the PR measurement to provide an estimate of the position error components in the same reference frame as the DOP values.

Algorithm

$$pos\_err = PDOP * \sigma_{PR}$$
$$horiz\_err = HDOP * \sigma_{PR}$$
$$vert\_err = VDOP * \sigma_{PR}$$

See Also **ned2dops**, **ll2dops**, **pseudo_r**, **diffcorr**

ecef2eci

Purpose Function to convert position and velocity vectors from Earth Centered Earth Fixed (ECEF) to Earth Centered Inertial (ECI) coordinates.

Syntax [x_eci, *v_eci*] = ecef2eci(GPS_time, x_ecef, *v_ecef*)

Inputs:

GPS_time = GPS time (nx2) [GPS_week GPS_sec] or (nx3) [GPS_week GPS_sec rollover_flag]). Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].
x_ecef = ECEF position in m (nx3) [xx_ecef, xy_ecef, xz_ecef]
v_ecef = *optional* ECEF velocity in m/s (nx3) [vx_ecef, vy_ecef, vz_ecef]

Note: If v_ecef is not provided, position only is converted from ECEF to ECI. v_eci will return filled with inf.

Outputs:

x_eci = ECI position in m (nx3) [xx_eci, xy_eci, xz_eci]
v_eci = *optional* ECI velocity in m/s (nx3) [vx_eci, vy_eci, vz_eci]

Description The ECEF x and y positions are rotated about the North Pole unit vector by the Greenwich sidereal hour angle. The z component remains unchanged:

$$\vec{x}_{eci} = C_{e2i}\vec{x}_{ecef}$$

$$where: \quad C_{e2i} = \begin{bmatrix} \cos\Phi & -\sin\Phi & 0 \\ \sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$and \qquad \Phi = SiderealTime(radians)$$

The ECEF velocity vector is similarly transformed, but the effect of the rotating coordinate frame with respect to inertial space is also added:

$$\vec{v}_{eci} = C_{e2i}\vec{v}_{ecef} + \vec{\Omega}\times\vec{x}_{eci}$$

where $\vec{\Omega}$ is the Earth spin rate vector expressed in the ECI frame.

See Also **eci2ecef, sidereal, ecef2lla, ecef2ned**

ecef2ll

Purpose Convert vectors from ECEF to Local Level (LL) coordinates.

Syntax [x_ll] = ecef2ll(x_vect, x_ecef, v_ecef)

Input:

x_vect = vector (ECEF, meters) to be converted to LL in meters [x, y, z] (nx3)
x_ecef = satellite position (ECEF) in meters [xe, ye, ze] (1x3 or nx3)
v_ecef = satellite velocity (ECEF) in m/s [vxe, vye, vze] (1x3 or nx3)

Output:

x_ll = x_vect converted to LL position in meters [xll, yll, zll] (nx3)

Description The local-level (LL) coordinate system is used as a reference frame for Earth-pointing satellites. The formulation must have both a position and a velocity vector to define the local-level axes. The LL frame used here is defined as:

$$\hat{z}_{LL} = \frac{-\vec{R}}{\left|\vec{R}\right|}$$

$$\hat{y}_{LL} = \frac{-\vec{R}\times\vec{V}}{\left|\vec{R}\times\vec{V}\right|}$$

$$\hat{x}_{LL} = \hat{y}_{LL} \times \hat{z}_{LL}$$

Here, $\hat{z}_{LL}$ is down, $\hat{y}_{LL}$ is anti-orbit-normal, $\hat{x}_{LL}$ is more or less along the velocity vector, $\vec{R}$ is pos_ecef, and $\vec{V}$ is vel_ecef. If x_ecef and v_ecef are 1x3, then these vectors are used throughout the transformations. If nx3, then each different row of x_vect uses a corresponding row of x_ecef and v_ecef.

See Also **ll2ecef, ecef2ned, ned2ecef**

References Hughes, Peter, “Spacecraft Attitude Dynamics”, John Wiley and Sons, 1986, page 282.

ecef2lla

Purpose Compute geodetic latitude, longitude, and altitude from ECEF coordinates.

Syntax [lla] = ecef2lla(x_ecef, *method*)

Inputs:

x_ecef = ECEF position in meters (nx3) [x_ecef, y_ecef, z_ecef]

method = *optional* flag indicating method to be used. Use 0 for fast but approx. (lat, long errors < 1e-4 rad, alt_err < 100m),1 for slower, more accurete, iterative method (lat, long errors < 1e-14 rad, alt_err < 1e-6m). Default = 0.

Outputs:

lla = geodetic lat, lon, and altitude (nx3) [lat, lon, altitude], Latitude is $-\pi/2$ to $+\pi/2$ radians. Longitude is 0 to 2π radians. Altitude is meters above the WGS-84 ellipsoid.

Description

See Also **lla2ecef, ecef2ned, ecef2eci, sidereal**

References Hoffman-Wellenhoff, “GPS: Theory and Practice”, pages 33, 255-257.

ecef2ned

Purpose Convert vectors from ECEF (Earth-Centered-Earth-Fixed) to NED (North-East-Down) coordinates.

Syntax [x_ned] = ecef2ned(x_ecef, ref_lla)

Input:

x_ecef = ECEF position in meters (nx3) [x_ecef, y_ecef, z_ecef]

ref_lla = reference lat, lon, and alt for NED coordinate system base, Latitude is -π/2 to +π/2 radians. Allowable longitude is -π to 2π radians. ref_lla must have dimensions of (1x3 or nx3) [lat lon hgt]. or (1x2 or nx2) [lat lon]. Note that altitude is provided as part of the standard lla matrix, but is not used by this function.

Output:

x_ned = x_ecef transformed to NED position in meters (nx3) [x_ned, y_ned, z_ned]

Description This function is typically used to transform line-of-sight vectors, computed from a fixed ground station to GPS/GLONASS satellites, from the ECEF to NED frame. The lla vector can be either in geodetic (computed in the **ecef2lla** function) or geocentric (compute by **cart2sph**, a Matlab provided function). If geodetic latitudes are provided, down is defined as normal to the ellipsoid. If geocentric latitudes are provided, down is defined as towards the center of the Earth.

$$\vec{x}_{ned} = c_{ecef\,2ned}\,\vec{x}_{ecef}$$

where: $$c_{ecef\,2ned} = \begin{bmatrix} -\sin lat \cos lon & -\sin lat \sin lon & \cos lat \\ -\sin lon & \cos lon & 0 \\ -\cos lat \cos lon & -\cos lat \sin lon & -\sin lat \end{bmatrix}$$

See Also **ned2ecef, ecef2lla, lla2ecef**

References Bate, Mueller, and White, "Fundamentals of Astrodynamics", page .101

eci2ecef

Purpose Convert position and velocity vectors from Earth-Centered-Inertial (ECI) to Earth-Centered-Earth-Fixed (ECEF) coordinates.

Syntax [x_ecef, *v_ecef*] = eci2ecef(GPS_time, x_eci, *v_eci*);

Inputs:

GPS_time = GPS time (nx2) [GPS_week GPS_sec]
or (nx3) [GPS_week GPS_sec rollover_flag]
x_eci = ECI position in meters (nx3) [xi, yi, zi]
v_eci = *optional* ECI velocity in m/s (nx3) [vxi, vyi, vzi]

Note: If *v_eci* is not provided, only position is converted from ECI to ECEF. *v_ecef* will return filled with inf.

Outputs:

x_ecef = ECEF position in meters (nx3) [xe, ye, ze]
v_ecef = *optional* ECEF velocity in m/s (nx3) [vxe, vye, vze]

Description The ECEF x and y positions are the ECI x and y positions rotated about the North Pole unit vector by the Greenwich sidereal hour angle. The z component remains unchanged. The inertial to Earth transformation is:

$$\vec{x}_{ecef} = C_{i2e}\vec{x}_{eci}$$

$$where: \quad C_{i2e} = \begin{bmatrix} \cos\Phi & \sin\Phi & 0 \\ -\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$and \qquad \Phi = SiderealTime(radians)$$

The ECI velocity vector is similarly transformed, but the effect of the rotating coordinate frame with respect to inertial space is also added:

$$\vec{v}_{ecef} = C_{i2e}\vec{v}_{eci} - \vec{\Omega} \times \vec{x}_{eci}$$

where $\vec{\Omega}$ is the Earth spin rate vector expressed in the ECI frame.

The ECI velocity vector is similarly transformed, but the effect of the rotating coordinate frame with respect to inertial space is also subtracted:

See Also **ecef2eci, sidereal, ecef2lla, ecef2ned**

eci2ll

Purpose Convert vectors from Earth Centered Inertial (ECI) to Local Level (LL) coordinates.

Syntax [x_ll] = eci2ll(x_vect, x_eci, v_eci)

Input:

x_vect = ECI position vector (meters) to be converted to LL in meters [x, y, z] (nx3)
x_eci = satellite ECI position (meters) [xi, yi, zi] 1x3 or nx3
v_eci = satellite ECI velocity (m/s) [vxi, vyi, vzi]1x3 or nx3

Output:

x_ll = LL position (meters) [xll, yll, zll] (nx3)

Description The local-level (LL) coordinate system is used as a reference frame for Earth-pointing satellites. The formulation must have both a position and a velocity vector to define the local-level axes. The LL frame used here is defined as:

$$\hat{z}_{LL} = \frac{-\vec{R}}{\left|\vec{R}\right|}$$

$$\hat{y}_{LL} = \frac{-\vec{R}\times\vec{V}}{\left|\vec{R}\times\vec{V}\right|}$$

$$\hat{x}_{LL} = \hat{y}_{LL}\times\hat{z}_{LL}$$

Here, $\hat{z}_{LL}$ is down, $\hat{y}_{LL}$ is anti-orbit-normal, $\hat{x}_{LL}$ is more or less along the velocity vector, $\vec{R}$ is pos_eci, and $\vec{V}$ is vel_eci. If x_eci and v_eci are 1x3, then these vectors are used throughout the transformations. If nx3, then each different row of x_vect uses a corresponding row of x_eci and v_eci.

See Also **ll2ecef, ll2eci, ecef2ll**

References Hughes, Peter, “Spacecraft Attitude Dynamics”, John Wiley and Sons, 1986, page 282.

err_chk

Purpose Performs error checking on function inputs.

Syntax [stop_flag] = err_chk(estruct)

Input:

estruct - error checking structure
estruct.func_name = (string), function name
estruct.variable(i).name = (string), i-th variable name from input
estruct.variable(i).req_dim = (nxm); i-th input required dimensions (optional)
estruct.variable(i).var = (jxk); i-th variable
estruct.variable(i).type = (type_string); i-th variable type (optional)

Output:

stop_flag - terminal condition flag, 1 = stop the called function, otherwise continue execution

Description Checks for matrix dimensions meeting required input dimension, common matrix dimensions between variables, and NaN, inf, and real checking. All errors report messages to the screen. Dimension failures cause the stop_flag to be set. NaN, inf, and real failues cause a warning message only.

The required dimensions (req_dim) is nxm where multiple options are allowed. For example, a valid input could be 1x3 or 1x4 which would lead to the req_dim = [1 3; 1 4]; For matching inputs between multiple variables, use a number greater than 900 to indicate that the dimensions must match. For example if variable #1 and variable #2 must match in the row dimension, but have different requirements the column dimension, it would be handled as follows

estruct.variable(1).req_dim = [901 2];
estruct.variable(2).req_dim = [901 3; 901 4];

This input means that the variables #1 and #2 must match in the row dimension and the column dimensions are 2 and 3 or 4, respectively. There is no restriction on the number of variables, the number of variables that must match a given dimension, or the number of allowed dimensions in the required dimension (req_dim) field.

The 'type' of variable input is used for the sanity checking of the values of that variable. Following are the current valid types and their bounds.

LLA_RAD:
latitude bounds = [-pi/2 to +pi/2]
longitude bounds = [-pi to +2*pi]
GPS_TIME:
GPS week bounds = [0 to 3640]
GPS sec bounds = [0 to 604800]
rollover_flag = [0 to 1]
ANGLE_RAD:
radian based angle bounds = [-2*pi to +2*pi]
ELEVATION_RAD:
elevation angle in radians bounds = [-pi/2 to +pi/2]
KEPLER_ORBIT
Keplerian orbit check, input [a e], (1x2) or (nx2)
a is in meters, e is dimensionless. Checks perigee and apogee to be above the surface of the Earth. If neither apse is above the surface of the Earth a warning message is issued.

STRING
Variable must be a string

If an invalid or unknown type is specified a warning message will be issued.

The type checking is completely optional.

If the global variable DEBUG_MODE is set to 1, the stop_flag will not be set and execution will continue. Error messages will be printed to the screen.

find_alm

Purpose Search the Matlab path for the most recent GPS and/or GLONASS almanacs.

Syntax [gps_alm_file, glo_alm_file] = find_alm(*GPS_week_start*);

Inputs:

GPS_week_start = *optional* GPS week number to begin almanac search. If no start week is given, the computer clock time will be used. The files that are searched for have the naming convention of gps###.alm or yuma###.txt for GPS almanacs, and glo###.alm, where ### is the week number,. GPS weeks are limited to 0 to 1024.

Outputs:

gps_alm_file = most recent GPS almanac file name (string).
glo_alm_file =most recent GLONASS almanac file name (string).

Description The search starts with the GPS_week_start and proceeds backward in time. The file name convention is important since this is the way the function will expect the file name in the search algorithm. An example file name for a GPS almanac from week 878 would be gps878.alm. The week number can be 1-4 characters (i.e. week 1 through 3640 are accepted).

See Also **readyuma**

Example To find the GPS almanac that is most recent use the following command
> [gps_alm_file] = find_alm;

To find the GPS and GLONASS almanacs that are most recent to the start of a simulation that starts on 6/1/03 use the following commands
> [start_week, start_sec] = utc2gps([2003 6 1 0 0 0]);
> [gps_alm_file, glo_alm_file] = find_alm(start_week);

The GPS and GLONASS almanacs found may be from different weeks.

genconst

Purpose Converts information about satellite constellations into the corresponding 6-element Keplerian element sets for each spacecraft.

Syntax [constell_elems] = genconst(constell_define);

Inputs:

constell_define = Array with each row defining a constellation (nx5, nx6, …, or nx11) (meters, radians)
[nsats = # of satellites in constellation,
nplanes = # of orbit planes,
a = semimajor axis ,
e = eccentricity,
i = inclination,
ASC_i = optional longitude of ascending node of 1st plane,
w_i = optional argument of perigee of 1st plane,
M_i = optional mean anomaly of 1st satellite in 1st plane,
delta_ASC = optional longitude of ascending node spacing between adjacent planes,
delta_w = optional argument of perigee spacing between adjacent planes,
delta_M = optional delta mean anomaly between satellites in adjacent planes,

Note: If only 5 parameters are input, then ASC_i=0, w_i=0, M_i=0, orbit planes will be evenly spaced, all sv's will have same w, and sv's in adjacent planes will have same M.

Outputs:

constell_elems = Constellation ephemeris [sv, a, e, i, long. of asc node, w, M] (meters, radians)

Note: sv's start counting at 101.

Description This function allows easy generation of orbital elements for large or small constellations of satellites. To generate elements for a 24 satellite constellation in 4 orbital planes, 7000000 meter circular orbit, 45 degrees inclination, use the following command. The 4 planes will be evenly spaced at ascending nodes of 0, 90, 180, and 270 degrees.

```
>  [constell_elems] = genconst([24, 4, 7000000, 0, 45*pi/180]);
```

The resulting output would be (shown here in degrees)

```
101  7000000  0  45     0   0      0
102  7000000  0  45     0   0     60
                 :
124  7000000  0  45   270   0    300
```

gps2utc

Purpose Converts GPS time into equivalent UTC time.

Syntax [UTC_time, *leap_sec*] = gps2utc(GPS_time, *offset*)

Input:

GPS_time = GPS time (nx2) [GPS_week GPS_sec] or (nx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

offset = *optional* leap seconds for the GPS times (1x1 or nx1). If not provided, leap seconds are computed based on the data in **leapsecs.dat.**

Output:

UTC_time = matrix of the form [year month day hour minute second] with 4-digit year (1980), nx6 matrix

leap_sec = *optional* leap seconds applied to UTC relative to GPS

Description Vectorized function to convert any number of GPS times to UTC time format. Leap second offsets should only be provided for special applications or times that are not covered by **leapsecs.dat**. If not provided, the actual number of leap seconds is read from **leapsecs.dat** for each GPS time to be converted. Any invalid inputs times will result in the UTC equivalent time being filled with inf (infinity) and a warning will be issued. If all of the GPS time input is invalid, the function will terminate with an error.

Limitations Valid GPS weeks are 0 - 3640
Valid GPS secs are 0 - 604800
Valid offset values are 0 - 500

See Also **utc2gps, utc2leap, leapsecs.dat**

gpst2sec

Purpose Utility function to convert a GPS time structure to linear seconds.

Syntax [total_gps_seconds] = gpst2sec(GPS_time)

Input:

GPS_time = GPS time (nx2) [GPS_week GPS_sec] or (nx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

Output:

total_gps_seconds = gps_weeks*86400*7 + gps_secs

Description This utility is useful when generating linear time intervals that may cross a week boundary. Since GPS standard convention keeps time since the GPS week rollover, the rollover_flag is not included in the computation.

See Also **sec2gpst**

ionodlay

Purpose Computes the ionospheric group delay for the L1 frequency.

Syntax iono_delay = ionodlay(t_gps, lla, az, el, *iono_params*);

Input:

t_gps = GPS time vector for az/el pairs [GPS_week, GPS_sec] (nx2)] or (nx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

lla = matrix of geodetic position (1x3 or nx3) [lat, lon, height] or (1x2 or nx2) [lat, lon]. lat and lon are in rad, height is not used with the Klobuchar (GPS-ICD-200) ionosphere model. Valid latitudes are -pi/2 -> pi/2. Valid longitudes are -pi -> 2*pi

az = azimuth (-2*pi to 2*pi rad) (nx1)

el = elevation (-pi/2 to pi/2 rad) (nx1)

iono_params = *optional* input data for the ionosphere model. 1x8 matrix in the form [alpha_0 alpha_1 alpha_2 alpha_3 beta_0 beta_1 beta_2 beta_3] where the alpha and beta are transmitted by the GPS satellites. These parameters are also available from RINEX2 data files available on the World Wide Web at

http://www.ngs.noaa.gov/~don/Data4.html

in the RINEX navigation files (these end with an n).

Default = [0.1211D-07 0.1490D-07 -0.5960D-07 -0.1192D-06 0.9626D+05 0.8192D+05 -0.1966D+06 -0.3932D+06]. These default values may or may not be representative of the current state of the ionosphere. Download new alpha and beta values for current times to reflect the current ionosphere.

Output:

iono_dlay = Ionospheric delay at L1 (m) (nx1)

Description This implements the standard GPS ionosphere model (Klobuchar) using the data broadcast in the subframe 4 data. This delay caused by the ionosphere is an apparent bias in the pseudo-range (pr) and accumulated carrier phase (cph) measurements collected by receivers that observe the GPS satellites through the ionosphere.

The ionosphere extends from about 50 km to 1000 km altitude. The ionosphere is a dispersive medium and is therefore frequency dependent. The frequency dependence is linear to a first order approximation. Therefore, the L2 delay is obtained by multiplying the L1 delay by $\gamma = 1.646944444$.

See Also **tropdlay**

References "Global Positioning System: Theory and Applications", Parkinson et. al, Vol. 1, pp. 144-149.

ICD-GPS-200, July 1, 1992

kep2geph

Purpose Converts from a Keplerian 6-element set to GPS ephemeris format (22 column format).

Syntax [gps_ephem] = kep2geph(kep_elems, *j2_flag*)

Input:

kep_elems = Keplerian element set (nx9), with columns of [sv_num, a, e, i, long. of asc_node, arg. of perigee, M, epoch GPS_week, epoch GPS_sec] units are seconds, meters, and radians (mod 2*pi)

j2_flag = *optional* flag to indicate the application of J2 effects on ascending node, mean motion, and argument of perigee, 0 for no J2 effect, 1 to apply J2 terms. Default = 1 to apply J2 terms.

Output:

gps_ephem **-** ephemeris matrix for all satellites (nx22), with columns of [prn, M0, delta_n, e, sqrt_a, long. of asc_node at weekly epoch, i, arg. of perigee, ra_rate, i_rate, Cuc, Cus, Crc, Crs, Cic, Cis, Toe, IODE, epoch GPS_week, Af0, Af1, perigee_rate] . Ephemeris parameters are from ICD-GPS-200 with the exception of perigee_rate. The gps_ephem will be filled with inf values for any kep_elems element set that is out bounds.

Description GPS ephemeris format is needed by **propgeph** for propagation of the GPS or GLONASS orbits. See description of **propgeph** for more details. The gps_ephem output variable will be filled with inf values for any almanac element set that is out bounds.

Algorithm If J2 perturbations are to be included, the following equations describe the computation of the J2 effects on ascending node, mean motion, and argument of perigee, respectively.

See Also **propgeph, alm2geph**

References "Global Positioning System: Theory and Applications Vol. 1", Spilker et al. or ICD-GPS-200 for details on the meaning of the ephemeris variables.

"Methods of Orbit Determination", Escobal for J2 perturbations.

$$\dot{\Omega} = -1.5n\frac{J_2 R_e^2}{a^2(1-e^2)^2}\cos i$$

$$\bar{n} = n\left[0.75\frac{J_2 R_e^2\sqrt{1-e^2}}{a^2(1-e^2)^2}(3\cos^2 i - 1)\right]$$

$$\dot{\omega} = 0.75n\frac{J_2 R_e^2}{a^2(1-e^2)^2}\left[5\cos^2 i - 1\right]$$

keplr_eq

Purpose Iteratively solve Kepler's equation for eccentric anomaly.

Syntax [E] = keplr_eq(M, e)

Input:
  M = mean anomaly (rad) (nxm)
  e = eccentricity (dimensionless) (nxm)
Output:
  E = eccentric anomaly (rad) (nxm)

Description Kepler's equation is given by

$$M = E - e\sin E$$

Eccentric anomaly, *E*, is solved for using iterative techniques. A maximum number of iterations of 10 and a convergence tolerance of $1x10^{-12}$ is used. A warning message is issued by the function if the tolerance is not achieved within the maximum number of iterations.

See Also **propgeph**

leapsecs.dat

Purpose Data file containing UTC time of each leap second increment.

Description Format of file is [UTC time (1x6) Leap seconds (1x1)]. One row for each leap second added since GPS time began.

Notes This file must be updated each time a leap second is added. It may be updated manually or downloaded from the Constell, Inc. Web page at www.constell.org.

ll2dops

Purpose Compute DOP values from Local Level (LL) LOS vectors.

Syntax

[dops, t_out, num_sats] = ll2dops(x_ll, t);

Input:

x_ll = line-of-sight (LOS) vector in LL coordinates (nx3) unit vectors
t = GPS time corresponding to each row in the LOS vectors, [GPS_week GPS_sec] (nx2) or (nx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

Output:

dops = dilution of precision for each time (kx5), where k = number of time steps, [GDOP, PDOP, HDOP, VDOP, TDOP]
t_out = output time matrix (kx2) [GPS_week GPS_sec] or (kx3) [GPS_week GPS_sec rollover_flag].
num_sats = number of satellites used in the DOP computation (kx1)

Description Computes the Dilution of Precision (DOP) values for Precision (PDOP), Geometric (GDOP), Horizontal (HDOP), Vertical (VDOP), and Time (TDOP). The DOP is a measure of the strength of the satellite geometry for obtaining position or velocity solutions. Smaller values of DOP indicate a better ability to resolve position and velocity.

The local-level (LL) coordinate system is used as a reference frame for Earth-pointing satellites. The formulation must have both a position and a velocity vector to define the local-level axes. The LL frame used here is defined as:

$$\hat{z}_{LL} = \frac{-\vec{R}}{\left|\vec{R}\right|}$$

$$\hat{y}_{LL} = \frac{-\vec{R}\times\vec{V}}{\left|\vec{R}\times\vec{V}\right|}$$

$$\hat{x}_{LL} = \hat{y}_{LL} \times \hat{z}_{LL}$$

Here, $\hat{z}_{LL}$ is down, $\hat{y}_{LL}$ is anti-orbit-normal, $\hat{x}_{LL}$ is more or less along the velocity vector, $\vec{R}$ is the position vector, and $\vec{V}$ is the velocity vector

See Also **vis_data, ned2dops**

Note If only 3 satellites are found at a given time, the altitude is assumed fixed and only the HDOP and TDOP are filled with values. If fewer than 3 satellites are found at a given time, the DOPs values are filled with inf for that time.

Reference Reference: Parkinson and Spilker,"Global Positioning System: Theory and Applications", vol. 1, page 413-414. Modified to operate in LL coordinates.

ll2ecef

Purpose Convert vectors from Local Level (LL) to ECEF coordinates.

Syntax [x_ecef] = ll2ecef(x_ll, pos_ecef, vel_ecef);

Inputs:

x_ll = vector (LL) to be converted to ECEF (m)
    [xl, yl, zl], (nx3)
pos_ecef = satellite position (ECEF) (m)
    [xe, ye, ze], (1x3 or nx3)
vel_ecef = satellite velocity (ECEF) (m/s)
    [vxe, vye, vze], (1x3 or nx3)

Note: If pos_ecef and vel_ecef are 1x3, then these values are used for all transformations; if nx3, then each row of x_ll will be transformed based on the corresponding rows of pos and vel.

Output:

x_ecef = x_ll in ecef frame [xe, ye, ze] (nx3) (m)

Description The local-level (LL) coordinate system is used as a reference frame for Earth-pointing satellites. The formulation must have both a position and a velocity vector to define the local-level axes. The LL frame used here is defined as:

$$\hat{z}_{LL} = \frac{-\vec{R}}{\left|\vec{R}\right|}$$

$$\hat{y}_{LL} = \frac{-\vec{R}\times\vec{V}}{\left|\vec{R}\times\vec{V}\right|}$$

$$\hat{x}_{LL} = \hat{y}_{LL} \times \hat{z}_{LL}$$

Here, $\hat{z}_{LL}$ is down, $\hat{y}_{LL}$ is anti-orbit-normal, $\hat{x}_{LL}$ is more or less along the velocity vector, $\vec{R}$ is pos_ecef, and $\vec{V}$ is the vel_ecef.

See Also **ecef2ll, ecef2ned, ll2azel**

ll2eci

Purpose Convert vectors from Local Level (LL) to Earth-Centered-Inertial (ECI) coordinates.

Syntax [x_eci] = ll2eci(x_ll, pos_eci, vel_eci)

Inputs:
- x_ll = vector (LL) to be converted to ECI (m)
  - [xl, yl, zl], (nx3)
- pos_eci = satellite position (ECI) (m)
  - [xe, ye, ze], (1x3 or nx3)
- vel_eci = satellite velocity (ECI) (m/s)
  - [vxe, vye, vze], (1x3 or nx3)

Output:
- x_eci = x_ll in eci frame [x, y, z] (nx3) (m)

Description The local-level (LL) coordinate system is used as a reference frame for Earth-pointing satellites. The formulation must have both a position and a velocity vector to define the local-level axes. The LL frame used here is defined as:

$$\hat{z}_{LL} = \frac{-\vec{R}}{\left|\vec{R}\right|}$$

$$\hat{y}_{LL} = \frac{-\vec{R}\times\vec{V}}{\left|\vec{R}\times\vec{V}\right|}$$

$$\hat{x}_{LL} = \hat{y}_{LL} \times \hat{z}_{LL}$$

Here, $\hat{z}_{LL}$ is down, $\hat{y}_{LL}$ is anti-orbit-normal, $\hat{x}_{LL}$ is more or less along the velocity vector, $\vec{R}$ is pos_eci, and $\vec{V}$ is vel_eci.

See Also **eci2ll, ll2ecef**

lla2ecef

Purpose Compute ECEF coordinates from geodetic latitude, longitude, and altitude.

Syntax [x_ecef] = lla2ecef(lla)

Inputs:

lla = matrix of geodetic parameters (nx3) [lat, lon, altitude]. lat and lon are in radians, altitude in meters above the WGS-84 ellipsoid. Valid latitudes are -pi/2 -> pi/2. Valid longitudes are -pi -> 2*pi

Output:

x_ecef = ECEF position in meters (nx3) [x_ecef, y_ecef, z_ecef]

Description

See Also **ecef2lla, ecef2ned, ecef2eci, sidereal**

References Hoffman-Wellenhoff , et al, “GPS: Theory and Practice”, pages 33, 255-257.

los

Purpose

Compute line-of-sight (LOS) vectors from a group of objects to another group of objects, each identified by object number within the group. This function is used for LOS computation for all ground assets, satellites, and constellation combinations.

Syntax

[t_los, los_vect, los_indices, obscure_info] = los(t1, x1, t2, x2, *earth_model_flag*);

Inputs:

t1 = GPS time vector for group #1 objects (mx2) [GPS_week GPS_sec]) or (mx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

x1 = positions for group #1 vehicles (mx3) [x y z] or (mx4) [veh_num1 x y z].

t2 = GPS time vector for group #2 objects (nx2) [GPS_week GPS_sec]) or (nx3) [GPS_week GPS_sec rollover_flag]. Use the nx3 format with rollover_flag of zero only for times preceeding the GPS week rollover on Aug. 22, 1999. For times since Aug. 22, 1999, include only [GPS_week GPS sec].

x2 = positions for group #2 vehicles (nx3) [x y z] or (nx4) [veh_num2 x y z],

earth_model_flag = *Optional.* 0 for spherical earth, 1 for oblate earth (1x1). Default is spherical earth. Used only if obscure_info is requested as output. The tangent of the los_vect to the earth is computed. Note: For oblate earth, input vectors x1 and x2 must be in ECEF coordinates. Note used for visibility from a ground-based site.

Note: veh_num is a vehicle identification number, not required for either group #1 or group #2.

Note: If a single time for a given object number is given for either input set (e.g. x1 or x2 data), it is assumed to be an object fixed to the surface of the Earth. This objects position will be fixed and LOS computed for each of the times in the other object (e.g. if the ground station is entered in x1 data, then the times from x2 will be used for the LOS computations). The Earth fixed objects are assumed to be in ECEF coordinates. Care should be taken when inputting Earth fixed objects in ECI or other non-ECEF coordinate frames.

Note: Group #1 and group #2 positions must be in the same reference system (e.g. ECEF or ECI).

Outputs:

t_los = GPS time vector corresponding to the LOS (kx2) [GPS_week GPS_sec] or (kx3) [GPS_week GPS_sec rollover_flag].

los_vect = line of sight vector at t_los from group #1 object to group #2 object (kx3). LOS are computed only for matching times in t1 and t2.

los_indices = indices to obtain relationship between output LOS vectors and the input positions [x1_ind, x2_ind] (kx2)

obscure_info = contains information needed to determine whether the earth obscures the line-of-sight (kx3) [tangent_radius, alpha, beta].

Description

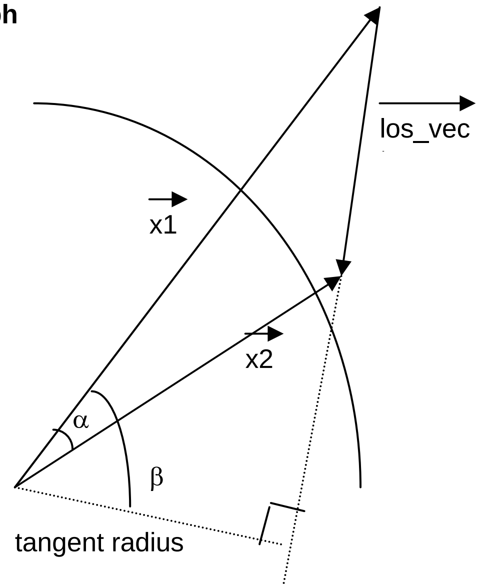
Compute line-of-sight (LOS) vectors from a group of objects to another group of objects, each identified by object number within the group. For example, this function allows computation of LOS vectors from a satellite (group #1) to the GPS constellation of satellites (group #2). Group #2 could also be a group of ground stations. This function is used for LOS computation for all ground assets, satellites, and constellation combinations.

The los_indices can be used to track which velocities or other data associated with the vehicle trajectory go with each line of sight. For example, to rotate the ECEF LOS vector to a satellite local level coordinate system via **ecef2ll**, the position and velocity of the vehicle are required. The **ecef2ll** function call would be the following assuming the original vehicle position and

velocity matrices were x1 & v1 [x_ll] = ecef2ll(los_vect, x1(los_indices(:,1),:), v1(los_indices(:,1),:));
See help on **ecef2ll** for details about these inputs.

Algorithm Alpha is the angle between the x1 and x2 vectors. Beta is the angle between the x1 vectors and the radius to the tangent point of the los_vect vectors. An observation is obscured if the tangent radius is below the users tolerance, and alpha is greater than beta.

See Also **propgeph**



lsnav

Purpose Computes a least squares PR navigation solution or a least squares position and velocity solution if the Doppler data is provided.

Syntax [t_nav, x_nav, num_sats, nav_index, *v_nav*] = lsnav(t_pr, pr, gps_orb,
init_pos, *doppler*, *gps_vel*, *init_vel*);

Input:

t_pr = time associated with the PR and GPS orbit [GPS_week GPS_sec] (kx2)
or [GPS_week GPS_sec rollover_flag] (kx3) for dates prior to August 22, 1999.
rollover_flag assumed to be 1 indicating times since August 22, 1999.
valid GPS_week values are 1-1024.
valid GPS_sec values are 0-604799.
pr = pseudo-range (PR) corresponding to t_pr time (kx1) (m)
gps_orb = GPS/GLONASS satellite orbits corresponding to pr, (kx4) (m) ECEF [sv_num x y z]
init_pos = initial estimate of the user position and clock bias (1x4) (m) ECEF [x y z clk_bias]
doppler = *optional* Doppler measurements corresponding to t_pr time (kx1) (m). No velocity solution is returned if not provided.
gps_vel = *optional* GPS/GLONASS satellite velocities corresponding to pr and Doppler measurements (kx3) (m/s) ECEF [vx vy vz] (required when using Doppler data)
init_pos = *optional* initial estimate of the user velocity and clock drift (1x4) (m/s) ECEF [vx vy vz clk_drift] (required when using Doppler data)

Output:

t_nav = time of navigation solutions (nx2) [GPS_week GPS_sec] or [GPS_week GPS_sec rollover_flag] (nx3)
x_nav = navigation position solution for each time (nx4) (m) ECEF, [x y z clk_bias]
num_sats = number of satellites used in the nav computation (nx1), n is the number of resulting navigation solutions
nav_index = index that relates the t_nav matrix to the t_pr matrix (kx1) i.e. [1 1 1...2 2 2...3 3 3 3 3...] where all of the 1s refer to the first t_nav/t_pr, the 2s the next, etc.
v_nav = *optional* navigation velocity solution for each time (nx4) (m) ECEF, [vx vy vz clk_drift]

Description To compute the position and velocity solution, a least squares navigation to the measurement equations is used at each time step. There is no correlation in the navigation solution between time steps. This process is done in a batch format with no looping. This speeds execution and improves the readability of the algorithm.

Algorithm For each satellite in the solution, the predicted pseudorange is computed as

$$\hat{\rho} = \left| \vec{X}_{GPS} - \vec{X}_{user} \right| + B_{rx\_clock}$$

After forming the predicted pseudorange measurement, the measurement residual is formed using the measured pseudorange and the predicted pseudorange as

$$\Delta\rho = \hat{\rho} - \rho_{measured}$$

The measurements are then combined into a set of normal equations as

$$\Delta\rho = G\Delta\mathbf{x}$$

where

$$\Delta\rho = \begin{bmatrix} \Delta\rho_1 \\ \Delta\rho_2 \\ \vdots \\ \Delta\rho_n \end{bmatrix} \quad G = \begin{bmatrix} los_1^N & los_1^E & los_1^D & 1 \\ los_2^N & los_2^E & los_2^D & 1 \\ \vdots & \vdots & \vdots & \vdots \\ los_n^N & los_n^E & los_n^D & 1 \end{bmatrix} \quad \Delta\mathbf{x} = \begin{bmatrix} \vec{X}_{user} \\ B_{rx\_clock} \end{bmatrix}$$

These equations are solved for a correction, $\Delta\mathbf{x}$ to the solution from the apriori estimate (init_pos) using the following equation

$$\Delta\hat{\mathbf{x}} = (G^T G)^{-1} G^T \Delta\rho$$

This process is repeated until $\Delta\mathbf{x}$ converges to .01 meters or the maximum number of iterations is exceeded (10). If the maximum number of iterations is exceeded, a warning is issued to the screen.

This same formulation is used to compute velocity solutions using Doppler data. The pseudorange measurement and measurement equation is replaced with the Doppler measurement equation

$$\hat{\dot{\rho}} = \frac{\vec{X}_{GPS} - \vec{X}_{user}}{\left| \vec{X}_{GPS} - \vec{X}_{user} \right|} \bullet (\vec{V}_{GPS} - \vec{V}_{user}) + f_{rx\_clock}$$

$\Delta\mathbf{x}$ is modified such that the clock bias term is replaced with a clock drift term, Doppler measurement residuals are computed, and the process is again iterated until converged. The position and velocity solutions are uncoupled.

Notes Times without 3 or more visibile satellites will be filled with the initial position and velocity estimates provided in init_pos and init_vel. For initial altitude estimates of greater than 10 km, the vehicle trajectory is assumed to be a satellite and a fixed altitude solution with only 3 satellites is not computed. All other trajectories will compute an alititude hold navigation solution when only 3 satellites are available.

See Also **propgeph**, **pseudo_r**, **doppler**

References "Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 410-415.

makeplot

Purpose Function to create the five most used plots, azimuth, elevation, sky plot, number of visible satellites, and DOPS for a single object.

Syntax fig_handles = makeplot(visible_data, pass_numbers, num_sats_vis, gps_dops, label_string, *plot_selection, min_elev, vertical_offset*);

Input:

visible_data = [azimuth, elevation, GPS week, GPS seconds, PRN] that are visible (nx5) (radians). For dates prior to the GPS week rollover on Aug 22, 1999, the format includes the rollover flag. [azimuth, elevation, GPS week, GPS seconds, rollover_flag, PRN] (nx6) (radians)

pass_numbers = pass number of each observation (nx1)

num_sats_vis = [GPS week, GPS Sec, number of visible satellites] (mx3). For times prior to Aug 22, 1999, include rollover_flag with GPS time [GPS week, GPS Sec, rollover_flag, number of visible satellites] (mx4).

gps_dops = [GPS week, GPS Sec, GDOP, PDOP, HDOP, VDOP, TDOP] (jx7). For times prior to Aug 22, 1999, include rollover_flag with GPS time [GPS week, GPS Sec, rollover_flag, GDOP, PDOP, HDOP, VDOP, TDOP] (jx8).

label_string =string identifying object to be plotted (1x??) i.e. 'Boulder' for ground site at Boulder, 'User Satellite 1', etc.

plot_selection = *Optional* (1x9) array to limit the plots that are created [1=plot sky plot, 1=plot elevation, 1=plot azimuth, 1=plot # of visible satellites, 1=plot PDOP, 1=plot GDOP, 1=plot HDOP, 1=plot VDOP, 1=plot TDOP]. Default = all plots.

min_elev = *Optional* minimum elevation to be plotted (radians). Used to set axis lower limit. Default is set by plot function.

vertical_offset = *Optional.* Figures are plotted above the previously plotted figure. (1x1) This integer identifies the number of plots to shift upward. Default is 0 or overlying figures.

Output:

fig_handles = Handles to figures created.

See Also **plotsky, plotpass**

ned2azel

Purpose Convert a North, East, Down vector into azimuth and elevation.

Syntax [az, el] = ned2azel(ned);

Input:

ned = vector in north, east, and down coordinates (nx3) or could be used with any other coordinate system where north and east are aligned with two local-level unit vectors such that north $\times$ east = down, e.g., an aircraft navigation frame or a satellite local-level frame)

Output:

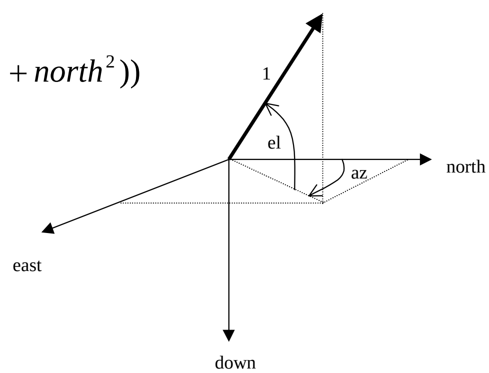
az = azimuth (rad) (nx1): rotation of vector in local North-East plane, clockwise about down, positive beginning at North.
$0 \le az \le 2\pi$

el = elevation (rad) (nx1): angle of vector above North-East plane, positive for a vector with a negative down component (i.e., for a vector with a positive up component).
$-\pi/2 \le el \le \pi/2$

Algorithm

$$az = \mathrm{atan2}(east, north)$$

$$el = \mathrm{atan}(-down / sqrt(east^2 + north^2))$$



See Also **azel2ned, ned2body, body2ned, ecef2ned, ned2ecef**

ned2body

Purpose Transform a vector from the North-East-Down local-level frame to a vehicle body frame.

Syntax [x_body] = ned2body(x_ned, euler);

Inputs:

x_ned = vector in NED frame (nx3) [North East Down]

euler = 3-2-1 Euler angle sequence from NED to body frame (radians). This input can be either (1x3) or (nx3), where the 1st col. is yaw, 2nd col. is pitch, 3rd col. is roll. If 1x3, then the same Euler angles are used for each vector.

Output:

x_body = transformed vector in body frame [xb, yb, zb] (nx3)

Description The body frame is transformed from the NED frame by a 3-2-1 Euler angle rotation sequence. The rotation angles and axes of rotation are given by:

NED to NED' frame: yaw about Down (3rd or z axis)
NED' to Body' frame: pitch about E' (2nd or y axis)
Body' to Body frame: roll about body-x axis (1st axis)

This function can be used to transform a vector through any 3-2-1 Euler angle sequence, not just from NED to body.

See Also **body2ned, ecef2ned, ned2ecef**

ned2dops

Purpose To compute DOP values from North, East, Down line-of-sight (LOS) vectors.

Syntax [dops, t_out, num_sats] = ned2dops(ned, t);

Input:

- ned = LOS vector in NED coordinates (nx3) unit vectors
- t = GPS time corresponding to each row in the LOS vectors, [GPS_week GPS_sec] (nx2), or (nx3) [GPS_week GPS_sec rollover_flag] Valid GPS_week values are 1-1024. Valid GPS_sec values are 0-604799. GPS week values are kept in linear time accounting for 1024 rollovers. Include a rollover_flag of 0 for any times prior to August 22, 1999. Default rollover_flag=1 indicating time since August 22, 1999.

Output:

- dops = dilution of precision for each time (kx5), where k = number of time steps, [GDOP, PDOP, HDOP, VDOP, TDOP]
- t_out = output time matrix (kx2) [GPS_week GPS_sec] or (kx3) [GPS_week GPS_sec rollover_flag] for dates prior to Aug. 22, 1999.
- num_sats = number of satellites used in the DOP computation (kx1)

Description Computes the Dilution of Precision (DOP) values for Position (PDOP), Geometric (GDOP), Horizontal (HDOP), Vertical (VDOP), and Time (TDOP). The DOP is a measure of the strength of the satellite geometry for obtaining position or velocity solutions. Smaller values of DOP indicate a better ability to resolve position and velocity.

See Also **vis_data, ll2dops**

Note If only 3 satellites are found at a given time, the altitude is assumed fixed and only the HDOP and TDOP are filled with values. If fewer than 3 satellites are found at a given time, the DOPs values are filled with inf for that time.

Reference Reference: Parkinson and Spilker,"Global Positioning System: Theory and Applications", vol. 1, page 413-414. Modified to operate in NED coordinates.

ned2ecef

Purpose

Function to convert vectors from NED (North-East-Down) coordinates to ECEF (Earth-Centered-Earth-Fixed) coordinates.

Syntax

[x_ecef] = ned2ecef(x_ned, ref_lla);

Inputs:

x_ned = NED position in meters [x_ned, y_ned, z_ned] (nx3)

ref_lla = reference lat, lon, and altitude for NED coordinate system. lat and lon in rad, altitude in meters, ref_lla must have dimensions of 1x3 or nx3 [lat lon alt] or 1x2 or nx2 [lat lon]. Note that the standard lla matrix is passed as input; however, altitude is not used in this routine. Valid latitudes are -π/2 to + π/2. Valid longitudes are -π to + 2π.

Output:

x_ecef = ECEF position in meters [x_ecef, y_ecef, z_ecef](nx3)

Description

This is the inverse transformation from ecef2ned.

$\vec{x}_{ecef} = c_{ned2ecef}\vec{x}_{ned}$

where: $$c_{ned2ecef} = \begin{bmatrix} -\sin lat \cos lon & -\sin lon & -\cos lat \cos lon \\ -\sin lat \sin lon & \cos lon & -\cos lat \sin lon \\ -\cos lat & 0 & -\sin lat \end{bmatrix}$$

See Also

ecef2ned, ecef2lla, lla2ecef

nmeanext

Purpose To get the next field in an NMEA message string.

Syntax [data, index] = nmeanext(line, start_index);

Inputs:
  line = single line of NMEA data (string)
  start_index = index into the line to start searching for the data (1x1)
Outputs:
  data = next number or character string contained in the NMEA message
  index = starting index to be used to retrieve the next piece of data

Description The NMEA data format is a variable length ASCII string with the data separated by commas. The data contained in the message is a combination or numeric and character data. This routine searches a line of data starting at a given location to find the next piece of data to be extracted. A sample NMEA data set is included in the CONSTELLATION Toolbox and is called nmeadata.dat.

See Also **ex_nmea, readnmea, parsnmea, parsegga, parsegsa, parsegsv**

normvect

Purpose Normalize n-dimensional vectors to have length of 1.

Syntax [x_norm, x_mag] = normvect(x)

Input:
    x = vector to be normalized (n x m)
Output:
    x_norm = normalized vector, magnitude = 1, same direction as x (n x m)
    x_mag = magnitude of the vector (n x 1)

Description Inputs can be vector, scalar, positive, or negative.

num_vis

Purpose Computes the number of satellites visible as a function of time.

Syntax [t_out, num_sats, id_num_sats] = num_vis(t_vis, *observer_ids*);

Input:

t_vis = time that observed satellites are visible in GPS format (nx2) [GPS_week GPS_sec] or (nx3) [GPS_week GPS_sec rollover_flag]. Valid GPS_week values are 1-1024. Valid GPS_sec values are 0-604799. GPS week values are kept in linear time accounting for 1024 rollovers. Include a rollover_flag of 0 for any times prior to August 22, 1999. Default rollover_flag=1 indicating time since August 22, 1999.

observer_ids = *Optional* identification numbers for observers (i.e. PRN, ground station #) (nx1). If not provided, it is assumed that there only 1 observer, and id_num_sats will be filled with ones.

Output:

t_out = output time matrix (kx2) [GPS_week GPS_sec] or (kx3) [GPS_week GPS_sec rollover_flag]

num_sats = number of satellites visible at each output time (kx1)

id_num_sats = id number of the observer for each time step (kx1)

Description This algorithm requires the output of **vis_data** as input. Only data that has been determined to be visible to an observer should be input, since no re-computation of the masking information is performed within this routine.

The output time and number of satellites visible is a reduction of the time matrix to only the time transitions. Therefore if there were 6 different observer_ids for 100 different times (t = 600x2), the t_out would be 100x2.

See Also **vis_data**

orb_anim

Purpose Animate the orbits and ground stations over a Mercator map of the Earth.

Syntax current_fig = orb_anim(alm, start_time, *animation_dt, sta_loc, sta_name, mask*);

Inputs:

alm = almanac (nx13) or ephemeris (nx22) data for satellites to be animated
start_time = GPS start time, [GPS_weeks GPS_seconds] (1x2)
animation_dt = *optional* time step for the animation (1x1) (sec), default = 60
sta_loc = *optional* Earth station locations [lat, lon, hgt] (nx3), lat and lon (East) are in rad, hgt is in meters
sta_name = *optional* Earth station names (nxm) (string)
mask = *optional* station visibility mask (rad), default = 0.0, mask can be 1x1, nx4, nx5 in the form
[sta_num min_el min_az max_az] (nx4)
[sta_num min_el max_el min_az max_az] (nx5) with n elevation/azimuth triples, where sta_num corresponds to the sta_loc/sta_name index. If overlapping visibility masks are given, the least stringent mask will be applied

Output:

current_fig = handle to the resulting figure

Description This function shows animated orbits and ground stations. Any number of satellite orbits and ground stations can be displayed. Satellites visible from a ground station will plot in green, with non-visible satellites in red.

See Also **plotazel, writempg**

Example For a quick look at this function, use the CONSTELLATION. If you just want to try it the manual way, use the following commands …

```
> alm = readyuma;               % read in most recent GPS YUMA almanac
> orb_anim(alm,[923 11]);       % start the animation Sept. 14, 1997
```

To add a station with an interesting mask …

```
> sta_loc = [0.6981 4.4506 2000];  % station  lat, lon, hgt (rad and m)
> sta_name = 'Boulder';            % station name of Boulder
> mask = [1 pi/6 0 pi/2; 1 pi/4 pi 3*pi/2];   % masking in radians
> orb_anim(alm,[923 11],600,sta_loc,sta_name,mask);
```

parsegga

Purpose Parse GGA NMEA messages.

Syntax gga_out = parsegga(line);

Input:

line = string with one NMEA GGA data message

Output:

gga_out = output data message for GGA data type, [1, gga_data], where 1 is the internal GGA message type and gga_data is an nx12 matrix with columns [hours, mins, secs, lat, lon, hgt, fix_qual, num_svs, HDOP, geoid_height, DGPS_latency, DGPS_station_number]. lat and lon are in deg, hgt and geoid_height are in m, fix_quality is 1 = GPS, 2 = DGPS.

Description The NMEA $GPGGA data message is a primary message for position information. This message also contains a time tag which the $GPGSV and $GPGSA messages do not.

See Also **readnmea, parsnmea, parsegsa, parsegsv**

Note Invalid or empty values will be filled with inf.

parsegsa

Purpose Parse GSA NMEA messages.

Syntax gsa_out = parsegsa(line);

Input:

line = a string with one NMEA GSA data message

Output:

gsa_out = output data message for GSA data type [3, gsa_data] 3 is the internal GSA data type where gsa_data is an nx17 matrix with columns [auto, fix_dim, prn_1, prn_2, …, prn_12, PDOP, HDOP, VDOP]. auto is 0 = auto 2/3-D mode, 1 = manual 2/3-D mode, -1 = unknown. fix_dim is the dimension fix for this data, 2 = 2-D, 3 = 3-D.

Description The NMEA $GPGSA data message is a primary message containing the satellites that are being tracked and used in the position solution. It also contains information about the current DOPs.

See Also **readnmea, parsnmea, parsegga, parsegsv**

Note

Note: prn values that are not filled in the GSA message will be filled with -1. PDOP values for 2D fixes will also be filled with -1. When the fix is not 2D or 3D, the HDOP and VDOP will be filled with -1.

parsegsv

Purpose Parse GSV NMEA messages.

Syntax gsv_out = parsegsv(line, *fid*);

Input:

line = a string with one NMEA GSV data message

fid = *optional*. File handle to the data file (required for processing GSV message that span more than a single line).

Output:

gsv_out = output data message for GSV message type

[2, gsv_data]   2 is the internal GSV message type

where gsv_data is an nx49 matrix with columns

[num_sats, prn_1, el_1, az_1, snr_1, prn_2, ..., snr_12]

num_sats - number of satellite in view, elevation (el) and azimuth (az) are in deg, signal-to-noise ration in dB (definition varies with receiver)

Description The NMEA $GPGSV data message is a primary message containing the satellites that are in view. It also contains satellite location and the signal-to-noise (SNR) ratio for that satellite, if it is currently being tracked.

See Also **readnmea, parsnmea, parsegga, parsegsa**

Note Any unavailable satellite data will be filled with inf.

parsnmea

Purpose Parse a single line NMEA message data.

Syntax parse_out = parsnmea(line, *fid*);

Input:

line = string with NMEA data for one message

fid = *optional* file handle to the data file (required if processing a GSV message because this message can span up to 3 lines for a complete message)

Output:

parse_out =output data message, varies with message type [message_type, message_data]. Supported message types include GPGGA, GPGSV, GPGSA. message_type is 1 = GPGGA , 2 = GPGSV, 3 = GPGSA. message_data for each NMEA message supported is described below.

gga -> message_data is an nx12 matrix with columns [hours, mins, secs, lat, lon, hgt, fix_qual, num_svs, HDOP, geoid_height, DGPS_latency, DGPS_station_number]. lat and lon are in deg, hgt and geoid_height are in m, fix_quality is 1 = GPS, 2 = DGPS.

gsa -> message_data is an nx17 matrix with columns [auto, fix_dim, prn_1, prn_2, …, prn_12, PDOP, HDOP, VDOP]. auto is 0 = auto 2/3-D mode, 1 = manual 2/3-D mode, fix_dim is the dimension fix for this data, 2 = 2-D, 3 = 3-D.

gsv -> message_data is an nx49 matrix with columns [num_sats, prn_1, el_1, az_1, snr_1, prn_2, ..., snr_12]. num_sats - number of satellite being tracked, elevation (el) and azimuth (az) are in deg, signal-to-noise ration in dB (definition varies with receiver).

Description The function serves as a master parsing function for the NMEA reading routines.

See Also **readnmea, parsegga, parsegsa, parsegsv**

passdata

Purpose Determines the pass numbers for data that has passed the masking tests in **vis_data**. Also provide summary information about each pass.

Syntax [pass_numbers, pass_times, pass_summary] = passdata(gps_time, *pass_dt, id_nums, other_vis_data*);

Input:

gps_time = GPS times that targets are visible to observers [GPS week, GPS_sec] (nx2) or (nx3) [GPS_week GPS_sec rollover_flag]. Valid GPS_week values are 1-1024. Valid GPS_sec values are 0-604799. GPS week values are kept in linear time accounting for 1024 rollovers. Include a rollover_flag of 0 for any times prior to August 22, 1999. Default rollover_flag=1 indicating time since August 22, 1999.

pass_dt = *Optional* minimum number of seconds between consecutive data determining whether this data is part of the previous datas visibility pass. (1x1) (seconds). Should be a number greater than your propagation step size. If not provided, all passes will be given the pass number 1.

id_nums = *Optional* identification numbers of observer and target objects (i.e. PRN, ground station #) for two objects (nx2). Used to differentiate pass numbers based on which objects were involved in the observation.

other_vis_data = *Optional* array of other visible data i.e. azimuth, elevation, range, etc. (nxm) (any units).

Output:

pass_numbers = Pass number of each data point. (nx1).

pass_times = [Pass number, GPS week, GPS sec at start of pass, duration of pass (sec), id_num1, id_num2] (jx6) or [Pass number, GPS week, GPS sec at start of pass, rollover_flag, duration of pass (sec), id_num1, id_num2] (jx7) for times prior to Aug. 22, 1999.

pass_summary = Summary of other_vis_data for each pass (jxmxk) where j = number of passes, m = number of input variables in other_vis_data, k = 4 [value at start of pass, value at end of pass, minimum during pass, maximum during pass]. Units match input units in other_vis_data.

Description Any time there is a time jump greater than or equal to pass_dt, the pass number will be incremented. Similarly, if the id number of either the observer or target object changes, then pass number is incremented. This allows plotting of large time sets of data to show individual passes.

See Also **vis_data**

playmov

Purpose Plays Matlab movies in a new figure window while retaining the original window size and optionally the original color map.

Syntax fig_handle = playmov(M, *num_plays*, *frames_per_second*, *color_map*);

Input:

M = Matlab Movie matrix
num_plays = *optional* number of time to play through the movie, (1x1). Default = 1
frames_per_second = *optional* number of frames per second to play the movie, (1x1).
Default = 2
color_map = *optional* color map used for this movie.
Default = Matlab default color map

Output:

fig_handle = figure handle to the resulting figure

Description Generate a movie of DOPs over the surface of the Earth using the **ex_dop_m** function. This will be a coarse resolution movie to speed generation.
> [M, color_map] = ex_dop_m([1997 9 1 0 0 0],600,60,20);

To play the movie in a new figure window, 5 times, at a rate of 2 frames per second use
> playmov(M, 5, 2, color_map);

See Also **writemov**, **writempg**

Notes This is a wrapper function for the Matlab function **movie**. However, this implementation has the added feature of retaining the figure size and color attributes so that movies are exactly reproduced.

plotpass

Purpose Function to plot y data as a function of time for a single observer and return the plot handles.

Syntax p_handle = plotpass(t, plot_data, *prn_pass, plot_title, y_label, y_scale*);

Input:

t = time in GPS format (nx2) [GPS_week GPS_sec] or (nx3) [GPS_week GPS_sec rollover_flag]. GPS_week values may range from 0 to 1024. GPS_sec values may range from 0 to 604800. Include a rollover_flag of 0 for any times prior to August 22, 1999. rollover_flag assumed to be 1 indicating times since August 22, 1999.

plot_data = data to be plotted (nxm). each column is a different data set, i.e. [azimuth, elevation, range]

prn_pass = *Optional* [satellite number, pass number], nx1 or nx2. Pass numbers are computed by **passnums**. If prn_pass is not provided, a single line is drawn indicating a single pass of one object over the observer. If only satellite number is provided, then any occurrence of that satellite number is assumed to be part of the same pass.

plot_title = *Optional* title of plot (1xj string or mxj string). Default = 'Data Plot'

y_label = *Optional* label for the y-axis, (1xk string or mxk string). Default = 'Y Data'

y_scale = *Optional* min and max limits of y scale (1x2 or mx2). Units match plot_data. If provided, data will be truncated when passing from one limit to the other. Example: Azimuth data that goes from 360 to 0 degrees during a pass will not have a vertical line connecting those 2 points.

Note: if only 1 plot_title, y_label, or y_scale is provided for multiple sets of plot_data, then it will be used for all plots.

Output:

p_handle = graphics handles to the figures (mx1)

Description Each data arc generated by a satellite over the observer is colored and labeled with the satellite number. Multiple passes from the same satellite will be colored the same. A new figure is generated each time this function is called. Zoom capability is added to the figure. See help on ZOOM for more information

See Also **plotsky**

plotsky

Purpose Create a polar plot of the azimuth/elevation for a single observer of a constellation of satellites.

Syntax p_handle = plotazel(az, el, *prn_pass, plot_title, spoke_label*)

Input:

az = azimuth angle in -2π to +2π radians (nx1)
el = elevation angle in -π/2 to +π/2 radians (nx1)
prn_pass = *Optional* [satellite number, pass number], nx1 or nx2. Pass numbers are computed by **passnums**. If prn_pass is not provided, a single line is drawn indicating a single pass of one object over the observer. If only satellite number is provided, then any occurrence of that satellite number is assumed to be part of the same pass.
plot_title = *Optional* title of the plot (1xm string). Default = 'Sky Plot'
spoke_label = *Optional* labels for the North, East, South, and West spokes of the plot. (4xn string).
Default = str2mat('N','E','S','W')

Output:
p_handle = control handle to the graphic

Description Each azimuth/elevation arc generated by a satellite is colored and labeled with the satellite number. Multiple passes from the same satellite will be colored the same. This function will use the current figure if one is available. Otherwise, a figure will be created. Zoom capability is added to the figure. See help on ZOOM for more information. Starting points are not labeled and end points are labeled with 'x'.

Azimuth is the rotation angle of a vector about down in the local-horizontal plane. Zero azimuth corresponds to North, East is pi/2, etc, for a site fixed on the Earth. Azimuth may also be relative to a local level coordinate system for satellites or rockets. Elevation is the rotation angle of a vector above or below the local-horizontal plane (positive or negative elevation angles respectively).

See Also **plotpass**

propgeph

Purpose Computes satellite positions in ECEF coordinates from GPS ephemeris format.

Syntax [t_out, prn, x, v] = propgeph(gps_ephem, t_start, *t_stop, dt*);

Input:

gps_ephem = ephemeris matrix for all satellites (mx22), with columns [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, Af0, Af1, perigee_rate]. Ephemeris parameters are from ICD-GPS-200 with the exception of perigee_rate term which has been added to accommodate the J2 effect on the perigee for other than GPS satellites.

t_start = start time in GPS format (1x2) or (nx2) [GPS_week GPS_sec] If t_stop is not provided, orbits will be computed at each of the t_start times. Must be 1x2 if t_stop is given. Or (nx3) [GPS_week GPS_sec rollover_flag]. Valid GPS_week values are 1-1024. Valid GPS_sec values are 0-604799. GPS week values are kept in linear time accounting for 1024 rollovers. Include a rollover_flag of 0 for any times prior to August 22, 1999. Default rollover_flag=1 indicating time since August 22, 1999.

t_stop = *optional* stop time in GPS time format (1x2) [GPS_week GPS_sec] or (lx3) [GPS_week GPS_sec rollover_flag]

dt = *optional* output interval, seconds. Default = 2 sec

Output:

t_out = GPS time vector output [GPS_week, GPS_sec] (nx2) or [GPS_week GPS_sec rollover_flag] (nx3)

prn = S/C PRN or ID number (nx1)

x = S/C position in ECEF meters (nx3)

v = S/C velocity in ECEF meters/second (nx3)

Note: Output are in time order with all of the satellites at time #1 first, followed by the satellites at time #2, etc. The following matrix shows sample output format for 1 second step size up to 600 second duration for 24 satellites. The position and velocity correspond to the times and satellite numbers in the corresponding columns. The t_out and prn matrices have the following form.

```
                     wk  s  prn   X     Y     Z      Vx     Vy     Vz
[t_out prn x v] = [933 1    1   x11  y11  z11  vx11  vy11  vz11
                   933 1    2   x12  y12  z12  vx12  vy12  vz12
                    .    .    .    .    .    .    .    .
                    .    .    .    .    .    .    .    .
                   933 1   24  x124 y124 z124 vx124 vy124 vz124
                   933 2    1   x21  y21  z21  vx21  vy21  vz21
                   933 2    2   x22  y22  z22  vx22  vy22  vz22
                    .    .    .    .    .    .    .    .
                   933 600 24   .    .    .    .    .    .  ]
```

Description This is the orbit propagator used as the standard for computing GPS orbits. The propagator accepts data in addition to the 6-element Keplerian set to account for other than two-body orbit perturbations.

When using this propagator with GPS almanac data converted to the GPS ephemeris format or other satellite data converted to the GPS ephemeris format, it functions as a standard J2 orbit propagator. There is no degradation of the orbits caused by the propagator.

The terms in the ephemeris format are adopted directly from the GPS ICD standard. Note that the ascending node term is the longitude of the ascending node at the start of the GPS week. When using this propagator for a full set of ephemeris data collected from a GPS receiver (all the terms in the ephemeris are filled), the orbits will degrade quickly. New ephemeris data from the receiver should be used when available. To obtain the full accuracy of < 1.5m (bit 17 of subframe 2 set to 0 indicating the orbit fit is for four hours), the ephemeris data is only good for 2 hours.

See Also **alm2geph, kep2geph**

References "Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 132-138. (Correction to the typo in y_k equation in Table 8 has been applied).

ICD-GPS-200, July 1, 1992.

pseudo_r

Purpose Computes pseudorange (pr) and accumulated carrier phase (cph) measurements given a user trajectory and the GPS/GLONASS satellite positions.

Syntax [t_pr, prn, pr, *pr_orb*, *pr_errors, obscure_info*] = pseudo_r(t_user, x_user, v_user, t_gps, x_gps, v_gps, *model*, *seed*, *code_noise*, *carrier_noise, ephem,model_data*);

Input:

t_user = GPS time vector for user trajectory [GPS_week, GPS_sec] (nx2) or [GPS_week GPS_sec rollover_flag] (nx3) for times prior to Aug. 22, 1999. There must be coincident times with the times in the GPS time vector. If there are no coincident times, no pseudo-range or phase data will be generated and the output matrices will be empty.

x_user = ECEF/ECI position vectors for the user vehicle [x,y,z] (nx3) (m)

v_user - ECEF/ECI velocity vectors for the user vehicle [x,y,z] (nx3) (m/s)

t_gps = GPS time vector for GPS positions [GPS_week, GPS_sec] (mx2) or [GPS_week GPS_sec rollover_flag] (nx3)

x_gps = ECEF/ECI position vectors for GPS satellites [prn,x,y,z] (mx4) (m)

mask = *optional* satellite mask_vis information (rad). Default = 0. See help on VIS_DATA for details on the mask format

model = *optional* flags controlling which contributions to the PR errors are modeled. (1x11) [sa_eps dither troposphere ionosphere receiver_clock receiver_noise line_bias sat_motion sat_clock earth_rotation relativity]. A value of 1 (or 2 for the tropo) indicates usage of the model and a value of zero indicates no use of that model. Use values of 2 to implement user supplied models. See the code for where to insert the user models. A warning is given if a user model is selected and none is supplied. Default =[0 1 1 1 1 1 1 0 0 0 0].

seed = *optional* seed value for random number generator. Default = 0.

code_noise = *optional* 1 sigma estimate of the receiver code noise (1x1) (m). Default = 1.

carrier_noise = *optional* 1 sigma estimate of the receiver carrier noise (1x1) (m). Default = 0.01.

ephem = *optional* ephemeris matrix for all satellites (nx24). Used to compute the satellite clock. If not provided, no GPS satellite clock effects will be computed. Ephemeris parameters are from ICD-GPS-200 with the exception of perigee_rate.

model_data = *optional* structure with data for each of the models. If not provided, model defaults will be used. Valid model structure elements are

model_data.sa_eps (1x1) see SA_EPS for details.
model_data.sa_dither (1x3) see SA_CLOCK for details
model_data.tropo (1x3) see TROPDLAY for details
model_data.iono (1x8) see IONODLAY for details
model_data.rec_clcok (1x2) see CLOCKERR for details
model_data.line_bias (nx3) [rec_num ant_num line_bias_sigma] default = 1 meter for all line biases
Note: For the troposphere model, a value of 1 = modified Hopfield model, a value of 2 = UNB1 model (altitude dependent)

Output:

t_pr = GPS time associated with the PR, [GPS_week GPS_sec] (kx2), k = num_time_steps x number of visible satellites

prn = satellite number for this pseudo-range (kx1)

pr = pseudo-range and accumulated carrier phase measurements associated with the corresponding t_pr and prn (kx2) [pr cph] (m and cycles)

pr_errors = *optional* modeled errors added to the geometric range to obtain a pseudorange and carrier phase measurement (m or m/s) (kx13), [sa_eps_err sa_clk_err trop_wet trop_dry iono clk_bias clk_drift code_white carrier_white line_bias sat_motion sat_clock relative].

pr_ndex = *optional*. Indices to obtain relationship between output PR matrix and the input positions and velocity [x_user_inc, x_gpsind] (kx2)

obscure_info – optional Contains information needed to determine whether the earth obscures the line-of-sight (kx3) [tangent_altitude, alpha, beta]. Alpha is the angle between the x1 and x2 vectors. Beta is the angle between the x1 vectors and the radius to the tangent point of the los vectors. An observation is obscured if the tangent vector magnitude is below the users tolerance, and alpha is greater than beta.

Description Computes PR and CPH measurements in the presence of masking and errors. Errors that can be included in the measurements include S/A (epsilon and dither), troposphere, ionosphere, receiver clock bias and clock drift, and receiver code and carrier tracking noise.

Algorithm The pseudorange and accumulated carrier phase measurements are computed using the following formulae

$$pr = \left| \vec{X}_{GPS} - \vec{X}_{user} \right| + SA_{epsilon} + SA_{dither} + B_{rx\_clock} + T + I + \varepsilon_{code}$$

$$cph = \left| \vec{X}_{GPS} - \vec{X}_{user} \right| + SA_{epsilon} + SA_{dither} + B_{rx\_clock} + T - I + N + \varepsilon_{carrier}$$

where

$B \equiv$ Clock Bias
$T \equiv$ Troposphere
$I \equiv$ Ionosphere
$\varepsilon \equiv$ Noise
$N \equiv$ Integer Carrier Cycles

See Also **doppler**, **sa_eps**, **sa_clock**, **clockerr**, **tropdlay**, **ionodlay**, **vis_data**

readnmea

Purpose Read NMEA data from a file.

Syntax [gga_out, gsa_out, gsv_out] = readnmea(file_name);

Inputs:

file_name = name of file to read NMEA data from (string)

Outputs:

gga_out = NMEA data from $GPGGA messages and is an nx10 matrix with columns [hours, mins, secs, lat, lon, alt, diff_flag, num_used, HDOP, dgps_age]. lat and lon are in deg, alt is in m

gsa_out = NMEA data from $GPGSA messages in an nx17 matrix with columns [auto, fix_dim, prn_1, prn_2, ..., prn_12, PDOP, HDOP, VDOP]. auto is 0 = auto 2/3-D mode, 1 = manual 2/3-D mode, fix_dim is the dimension fix for this data, 2 = 2-D, 3 = 3-D.

gsv_out = NMEA data from $GPGSV messages in an nx49 matrix with columns [num_sats, prn_1, el_1, az_1, snr_1, prn_2, ..., snr_12]. num_sats - number of satellite being tracked, elevation (el) and azimuth (az) are in deg, signal-to-noise ration in dB (definition varies with receiver)

Description The function serves as a master driver for the NMEA reading and parsing routines.

See Also **ex_nmea, parsnmea, parsegga, parsegsa, parsegsv**

readyuma

Purpose Read in YUMA formatted GPS, GLONASS, or user created almanacs.

Syntax [gps_alm, glo_alm] = readyuma(*filename*);

Inputs:

filename = *optional* name of almanac file to read or GPS week number. The files must have the naming convention of gps###.alm and glo###.alm if using the week number to specify a file. If a filename instead of a GPS week number is specified, only the gps_alm matrix will be returned and it will contain the data for the given file. If a user created almanac file name is provided, gps_alm will contain the user almanac data. If no input is provided, the FIND_ALM function will locate the most recent almanac(s). This function can also be used to read user created almanacs by providing the full almanac name and extension. The data is returned in the gps_almfield. Almanac names are stored as mod(1024) weeks to conform to YUMA standards.

Outputs:

gps_alm = GPS almanac data matrix for all the satellites found in the specified almanac file. (nx13)

glo_alm = GLONASS almanac data matrix for all the GLONASS satellites found in the specified almanac file. (Supported when an almanac week number is provided.) (mx13)

Description Yuma almanac files provide current information on the status of the constellation and are available each week. GPS almanac files are available from the U.S. Coast Guard Web Site at

http://www.navcen.uscg.mil/

Yuma formatted GLONASS almanac files are available from the Lincoln Lab GLONASS site at

http://satnav.atc.ll.mit.edu/

Links to these sites are provided at the Constell, Inc. Web Site at

http://www.constell.org/

The internal format of Yuma almanac file format is as follows...
[sv_num, health, ecc, GPS_sec, inc, asc_node_rate, sqrt_a, longitude of asc. node at weekly epoch, perigee, mean_anomaly, Af0, Af1, GPS_week] with units of rad, s, and meters.

See Also **find_alm**, **kep2geph**

sa_clock

Purpose Simulate the S/A clock dither contribution to the pseudo-range (PR) and pseudo-range rate (PRR) models using a 2nd order Gauss-Markov process.

Syntax [PR_error, PRR_error] = sa_clock(t_pr, prn, *sa_model_data*, *seed*);

Input:

t_pr = GPS time of pseudoranges (PR) or carrier phase (CPH) measurements, (nx2) [GPS_week GPS_sec] or [GPS_week GPS_sec rollover_flag] (nx3) for times prior to Aug. 22, 1999. rollover_flag defaults to 1 for times since. Use rollover_flag=0 for times prior.

prn = GPS satellite numbers corresponding to t_pr (nx1)

sa_model_data = *optional* input for the dither model (1x3 or 3x1) matrix in the form [sigma_pr sigma_prr tau] where the sigma's on the PR and PR-rate are in m and m/s and tau is the decorrelation time in seconds. Default is the RTCA proposed values of [23 .28 118]

seed = *optional* seed value for random number generator. Default = 0.

Output:

PR_error = Pseudo-range error (nx1) (m), n = num_time_steps x num_sats

PRR_error = Pseudo-range rate error (nx1) (m/s)

Description Selective Availability (SA) was turned off in 2000. It is no longer employed. Selective Availability (SA) is the intentional degradation of the GPS signal and navigation data to deny the full position and velocity accuracy to the unauthorized. Authorized users have access to the Precise Positioning Service (PPS). Most users in the civilian community have access to the Standard Positioning Service (SPS) which is corrupted with SA.

Two different methods are used to deny the full precision of the GPS system. The first is inducing errors in the message used for computation of the GPS satellite positions. This is sometimes referred to as the ε-process. The second method is achieved by inducing errors in the GPS satellite clock frequency, also referred to as the δ-process or clock dither.

The dither process can be modeled by a 2nd Order Gauss-Markov process. This is the process adopted by the RTCA (formerly the Radio Technical Commission for Aeronautics). The default values provided for this model are specified by the RTCA.

This SA model is a non-real time representation of the dither effect of SA. It is representative in the sense that the statistics of the process are representative of an SA process that would be observed in a GPS receiver.

Algorithm See the reference for a complete description on the algorithm used to implement the 2nd Order Gauss-Markov SA Model.

Limitations For the 2nd Order Gauss-Markov process the input PRs should be evenly spaced in time. If they are not a warning message will be issued and the inputs will be assumed to be evenly spaced. This will result in PR errors that are skewed based on the time tags.

See Also **pseudo_r**, **sa_eps**, **clockerr**, **tropdlay**

References "Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 605-608.

sa_eps

Purpose Simulate the epsilon contribution of S/A to the pseudo-range (PR) and accumulated carrier phase (CPH) measurement error using the RTCA standard model.

Syntax [pr_sa_eps, sa_eps_err] = sa_eps(pr, *sigma*, *seed*);

Input:

pr = satellite number and associated PR measurement to corrupt with S/A epsilon errors (m) (nx2) [prn pr]

sigma = *optional* standard deviation of the bias to apply (1x1). Default is the RTCA proposed value of 23 meters.

seed = *optional* seed value for random number generator. Default = 0.

Output:

pr_sa_eps = sa_eps corrupted PR measurements (m) (nx1)

sa_eps_err = sa_eps bias added to PR measurements (m) (nx1)

Description The epsilon (ε) process can modeled as constant random biases chosen from a Guassian distribution with zero mean added to the satellite range measurements. This is the process adopted by the RTCA (formerly the Radio Technical Commission for Aeronautics). The default value for the sigma of 23 meters provided for this model is specified by the RTCA.

This SA model is a non-real time representation of the epsilon effect of SA. It is representative in the sense that the statistics of the process are representative of an SA process that would be observed in a GPS receiver.

See the Description section of the **sa_clock** function for a more complete description of SA.

Limitations This sa_eps model does not reflect the slowly varying effect of the sa_eps error which has periods on the order of a few hours.

See Also **propgeph**, **sa_clock**, **clockerr**, **tropdlay**

References "Global Positioning System: Theory and Applications", Volume 1, Parkinson and Spilker, pages 605.

sec2gpst

Purpose Convert from 1-dimensional linear GPS time to a two-dimensional structure.

Syntax [GPS_time] = sec2gpst(total_gps_secs)

Input:

total_gps_seconds = gps_weeks*86400*7 + gps_secs (nx1) (seconds)

Output:

GPS_time = [gps_weeks gps_secs] (nx2)

where gps_secs is from the beginning of the week (week begins at midnight Saturday night)

Description Utility function to convert gps time in seconds from Jan. 6, 1980 to a 2-element matrix of gps_week and gps_sec from beginning of week.

See Also **gpst2sec**

sidereal

Purpose Computes the Greenwich sidereal time (GST) from UTC time.

Syntax [gst] = sidereal(UTC_time);

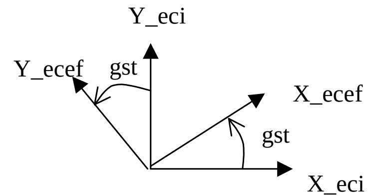
Input:

UTC_time = matrix of the form [year month day hour minute second] (nx6) with 4-digit (1980) years, valid years are 1900 - 2079

Output:

gst = angle between the ECI and ECEF coordinate system (rad) (nx1)

Description GST is defined as the angle (or time converted to angle) between the inertial x-axis, xi (first point of Aries), and the Greenwich meridian (which is the x-axis for the ECEF frame).



See Also **eci2ecef, ecef2eci, gps2utc, utc2gps**

tropdlay

Purpose Computes the wet and dry troposphere delays using a Hopfield model.

Syntax [trop_dry, trop_wet] = tropdlay(elev, *trop_model*);

Input:

elev = elevation angle to GPS satellites (rad) (nx1). Valid elevations are between 0 and pi/2.

trop_model = *optional* input for the troposphere model. (1x3 or nx3) matrix in the form [p T e], where p is the surface atmospheric pressure in mb, T is the surface temperature in degrees K, and e is the partial pressure of water vapor in mb. Default values are [1013.25 288.15 11.691].

Output:

trop_dry = Dry troposphere component (m) (nx1)
trop_wet = Wet troposphere component (m) (nx1)

Description The contribution to the path delay of the GPS radio signals from the neutral atmosphere (the nonionized part) is denoted as the troposphere delay. Although, the neutral atmosphere is made up of more than the troposphere, the troposphere is the dominant error source.

The neutral atmosphere is nondispersive medium with respect to radio waves at the GPS frequencies. Thus the propagation is frequency independent. Therefore, a distinction between the L1 and L2 frequencies is not required.

The troposphere is generally separated into two components the dry part coming from the dry atmosphere and the wet part coming from water vapor suspended in the atmosphere. The dry troposphere is the dominant source of delay accounting for about 90% of the troposphere delay.

The Hopfield models assumes a troposphere with a mean height of 11 km and extending to about 40 km in altitude. The is no accommodation or scaling for user altitudes located within the troposphere region.

See Also **tropunb1, pseudo_r**

References "GPS Theory and Practice", Hofmann-Wellenhof, Lichtenegger, and Collins, pages 110-113.

tropunb1

Purpose Computes the wet and dry troposphere delays using the University of New Brunswick (UNB1), altitude dependent, troposphere model. Use this model instead of the Hopfield model when altitude variation is being modeled.

Syntax [trop_dry, trop_wet] = tropunb1(lla,t_gps,elev);

```
Input:
  lla     - matrix of geodetic position (1x3 or nx3) [lat, lon, height]
             lat and lon are in rad, height is in m
             valid latitudes are -pi/2 -> pi/2
             valid longitudes are -pi -> 2*pi
  t_gps  - GPS time vector for az/el pairs [GPS_week, GPS_sec] (nx2) or
              [GPS_week GPS_sec rollover_flag] (nx3) for times prior to Aug. 22, 1999.
                valid GPS_week values are 1-3640 (years 1980-2050)
                valid GPS_sec values are 0-604799
  elev       - elevation angle to GPS satellites (rad) (nx1)
                Valid elevations are between -pi/2 and pi/2.
                Elevations below .1 degree will have the a delay mapped to .1 degree.
                No mapping below zero degree elevation is modeled.
Output:
  trop_dry   - Dry troposphere component (m) (nx1)
  trop_wet   - Wet troposphere component (m) (nx1)
```

Description The contribution to the path delay of the GPS radio signals from the neutral atmosphere (the nonionized part) is denoted as the troposphere delay. Although, the neutral atmosphere is made up of more than the troposphere, the troposphere is the dominant error source.

The neutral atmosphere is nondispersive medium with respect to radio waves at the GPS frequencies. Thus the propagation is frequency independent. Therefore, a distinction between the L1 and L2 frequencies is not required.

The troposphere is generally separated into two components the dry part coming from the dry atmosphere and the wet part coming from water vapor suspended in the atmosphere. The dry troposphere is the dominant source of delay accounting for about 90% of the troposphere delay.

The UNB1 model is a composite model that uses the explicit forms of Saastamoinen's delay algorithms combined with Niell's mapping functions. The surface and lapse rate parameters are from the U.S. 1976 Standard Atmosphere. There will be a several centimetre bias at the poles and the equator unless surface met data is used. This model was developed at the University of New Brunswick under contract to Nav Canada.

See Also **tropdlay, pseudo_r**

References

Collins, J.P. and R.B. Langley. "A Tropospheric Delay Model for the User of the Wide Area Augmentation System." Final contract report prepared for Nav Canada, Department of Geodesy and Geomatics Engineering Technical Report No. 187, University of New Brunswick, Fredericton, N.B., Canada.

Available in PDF format from http://gauss.gge.unb.ca/papers.pdf/waas.tropo.oct96.pdf

utc2gps

Purpose Converts a UTC time matrix to the equivalent time in GPS weeks, GPS seconds, and GPS days.

Syntax [GPS_week, GPS_sec, *GPS_day, rollover_flag*] = utc2gps(UTC_time, *leap_sec*);

Input:

UTC_time = matrix of the form [year month day hour minute second] (nx6) with 4-digit (1980) or 2-digit (80) years, valid years are 1980 - 2079 (2-digit 80-79)

leap_sec = *optional* leap seconds applied to UTC relative to GPS can be a 1x1 or an nx1. If not entered, the function will use a look-up table to determine the number of leap seconds

Output:

GPS_week = GPS week (nx1) (if 0 or 1 output parameters are used, this is filled with [GPS_week GPS_sec] (nx2).

GPS_sec = seconds into the week measured from Sat/Sun midnight (nx1)

GPS_day = *optional* days since the beginning of GPS time. (nx1)

rollover_flag = Flag indicating whether the GPS week rollover has occurred. (nx1) (optional). Rollover_flag = 0 for time before Aug. 22, 1999. Rollover_flag = 1 for time since Aug. 22, 1999.

Description Vectorized function to convert any number of UTC times to GPS time format. Leap second offsets should only be provided for special applications or times that are not covered by **leapsecs.dat**. If not provided, the actual number of leap seconds is read from **leapsecs.dat** for each UTC time to be converted. Any invalid input times will result in the GPS equivalent time being filled with ∞ (infinity) and a warning will be issued. If all of the UTC time input is invalid, the function will terminate with an error.

Limitations Maximum values for each UTC time field are = [2079 12 31 24 60 60]
Minimum values for each UTC time field are = [1980 1 0 0 0 0]
Values for leap_sec are limited to 0 - 500

See Also **gps2utc, utc2leap**

utc2leap

Purpose Determines the number of leap seconds of offset between GPS and UTC time.

Syntax [leap_sec] = utc2leap(UTC_time)

Input:

UTC_time = matrix of the form [year month day hour minute second] with 4-digit year (i.e. 1980), (nx6)

Output:

leap_sec - leap seconds relating UTC to GPS time (nx1)

Description Accesses the data file **leapsecs.dat** to determine the leap seconds of offset between GPS and UTC time.

See Also **gps2utc, utc2gps**

vis_data

Purpose Finds azimuth/elevation pairs passing the masking tests described in the mask variable. Returns only that data that passes the masking test.

Syntax [visible_data, I_pass] = vis_data(mask, all_data, *obscure_info, obscure_altitude*)

Input:

mask = masking information (rad) (1x1, nx3, or nx4). Default = 0;
1x1 form is minimum elevation only [min_el]
nx3 form is min elevation and azimuth bounds
[min_el start_az stop_az]
nx4 form is elevation and azimuth bounds
[min_el max_el start_az stop_az]
Azimuth start and stop are assumed to be clockwise.
Examples:
minimum elevation mask of 5 degrees (rad) (1x1)
mask = .0873
minimum elevation and azimuth bound triples (nx3)

```
mask = [.0873   pi/2   pi;   ( 5 deg min el, 90->180 azimuth)
          .1745   pi    pi/4]  (10 deg min el, 180->90 azimuth wraps through 0)
```

elevation and azimuth bound quadruples

```
mask = [.0873  .5236  0    pi;   (5->30 deg el, 0->180 azimuth)
          .1745  pi/4   pi  2*pi]   (10->45 deg el, 180->360 az)
```

all_data = raw observation data [azimuth (rad), elevation (rad), other corresponding data (optional)]. Minimum size (nx2) up to (nxm). Valid values for azimuth are -2π to $+2\pi$. Valid values for elevation are $-\pi/2$ to $+\pi/2$. Note: The corresponding data is optional and can be any data corresponding to the az/el pairs (nxk) where k is the number of other data type included. Units are parameter dependent and are not used within this function. Examples of data that might be included would be time, range, satellite number, etc.

obscure_info = Optional. Contains information needed to determine whether the earth obscures the line-of-sight (nx3) [tangent_radius, alpha, beta]. (meters, radians). Only include the following variables if you wish to use the most restrictive of the mask info or earth obscuring the view. Not used for visibility from a ground-based site. See Description below for angle definition.

obscure_altitude = Optional altitude above earth to include in earth obscuration (meters) (nx1). Default = 0 meters. Not used for visibility from a ground-based site.

Output:

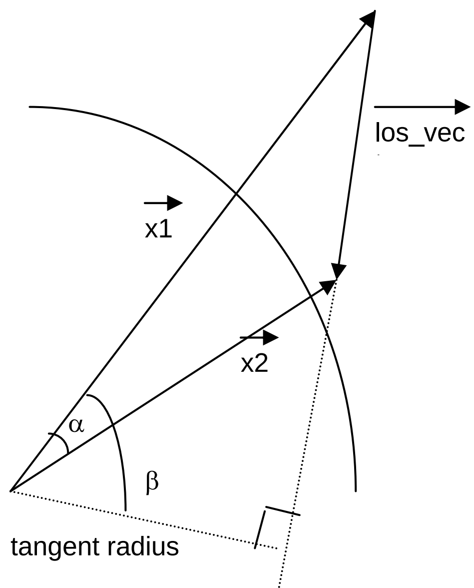
visible_data = all_data that passed the masking test and is not obscured by the Earth (jxm)

I_pass = index to data that passed the masking test (jx1)

Description This function is used to sort visible data from non-visible data. It can be used for any data type such as pseudorange or Doppler measurements, troposphere errors, SA errors, or any other data that is satellite dependent. The use of the standard azimuth/elevation masking combination is used.

Alpha is the angle between the observer (x1) and target (x2) vectors. Beta is the angle between the x1 vectors and the radius to the tangent point of the line-of-sight vectors. An observation is obscured if the tangent radius is below the users tolerance, and alpha is greater than beta.

See Also **los**



writemov

Purpose Play a Matlab movie and write the movie to an MPEG movie file while retaining the original figure window size and color map properties.

Syntax writemov(M, file_name);

Input:
- M = Matlab Movie matrix
- file_name = file name to write the MPEG Movie (string)

Output:
- None

See Also **writempg**, **playmov**

writyuma

Purpose Write YUMA formatted almanacs for any constellation.

Syntax [err_string] = writyuma(ephemeris, *filename, over_write*);

Input:

ephemeris = GPS ephemeris formatted data to use to create the YUMA almanac (nx22), with columns of [prn, M0, delta_n, e, sqrt_a, longitude of asc_node at weekly epoch, i, perigee, ra_rate, i_rate, Cuc, Cus, Crc, Crs, Cic, Cis, Toe, IODE, GPS_week, Af0, Af1, perigee_rate]. Use **kep2geph** to convert Keplerian elements into the ephemeris format or **alm2geph** to convert from existing YUMA almanac format. Ephemeris parameters are from ICD-GPS-200 with the exception of perigee_rate. The gps_ephem will be filled with inf values for any almanac element set that is out bounds.

filename = *Optional* name of almanac file to write. The file naming convention of gps###.alm and glo###.alm is preferable for GPS and GLONASS constellations. User defined constellation may want to use usr###.alm.

over_write = *Optional* flag indicating that the file should be overwritten without a warning message if it already exists.
over_write = 1 to suppress the warning if the file exists.
over_write = 0 to display the warning if the file exists.
Default = 0 to display the warning.

Outputs:

err_string = error message issued if the write failed. This string is empty if writing the YUMA almanac was successful

Description For user created constellations (like from **genconst**), this function can be used to create an almanac formatted file so that **readyuma** can be used by other functions. Then, a user created constellation is indistinguishable from a GPS or GLONASS constellation.

See Also **alm2geph, kep2geph**