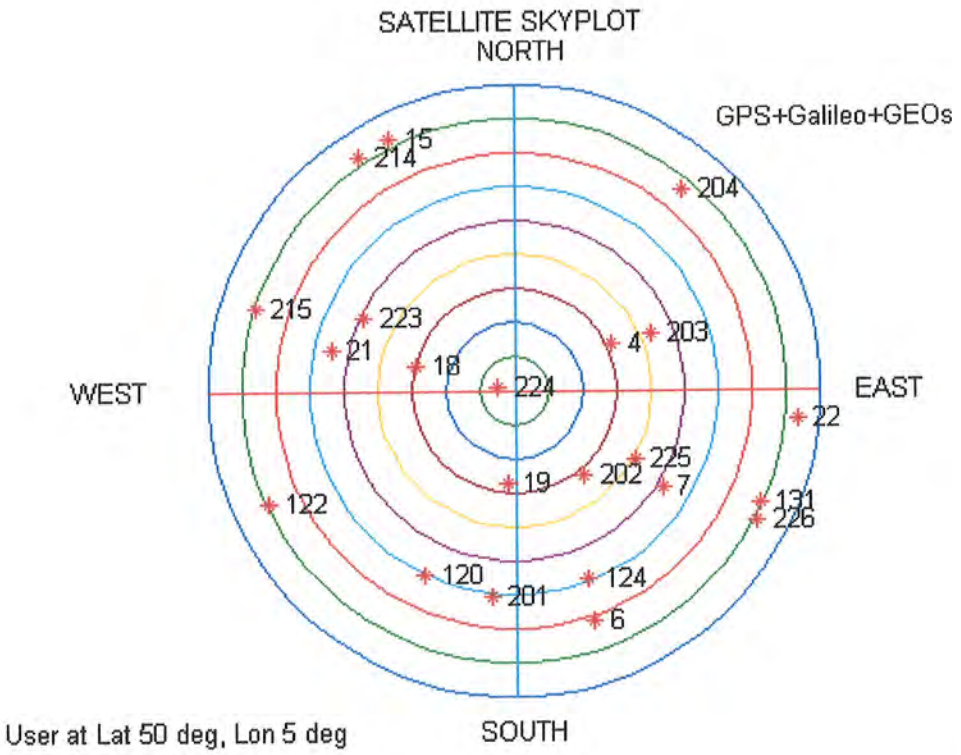
Satellite Navigation
TOOLBOX 3.0

For MATLAB®



User's Guide

GPSoft®

Satellite Navigation TOOLBOX 3.0

For MATLAB®

User's Guide

GPSoft®

Satellite Navigation Toolbox 3.0 User's Guide
September 2003

GPSoft®

P. O. Box 962
Athens, Ohio 45701

Phone: (740) 594-8610
FAX: (740) 594-8610

E-mail for general information: info@gpsoftnav.com
E-mail for technical support: tech@gpsoftnav.com

Table of Contents

1 Set-Up and Installation

For MATLAB 6.x:

The CD accompanying this manual contains a single subdirectory labeled `satnav3p0`. To install the toolbox, it is suggested that a subdirectory named `satnav3p0` be created off of the `c:\MATLAB6pX\toolbox` directory (where 'X' represents the MATLAB version number). Copy the entire contents of the `satnav3p0` directory on the CD into the `satnav3p0` directory on the hard disk (as an alternative, one may simply drag the `satnav3p0` directory on the CD onto the MATLAB `toolbox` directory thus creating the new directory and copying the contents simultaneously).

To complete the installation, do the following:

1) Start MATLAB
2) Open the Preferences dialog under the File menu
3) Click on 'Set Path'
4) Click on 'Add Folder'
5) Search for the newly created `satnav3p0` directory, click on it and then click 'OK'
6) At the bottom of the Set Path Dialog, click on SAVE then on CLOSE

MATLAB should automatically update the toolbox cache. However, one can type 'help toolbox_path_cache' for more information regarding updating the cache.

For MATLAB 5.x :

The CD accompanying this manual contains a single subdirectory labeled

satnav3p0. It is suggested that a subdirectory named satnav3p0 be created off of the c:\matlab\toolbox directory.

Copy the entire contents of the satnav3p0 directory on the CD into the satnav3p0 directory on the hard drive.

To complete the installation, make a back-up of the file pathdef.m. It can be found in the c:\matlab\toolbox\local\ subdirectory. After making the back-up, edit the file to add the satnav3p0 subdirectory to the MATLAB path. This will allow you to access the SatNav Toolbox files without placing them in your current directory.

After saving the updated pathdef.m, the SatNav Toolbox will be available for use the next time MATLAB is invoked.

The computational intensity will make many of the programs run slow with older CPU's. For best performance, a 500 MHz Pentium or better should be used.

A NOTE ABOUT HARD DISK SPACE

The vast majority of the files on the CD are M-files that are only a few kilobytes in size. There are, however, a few RINEX data files and their MATLAB MAT-file equivalents that are several megabytes in size each. These are data files used to support the demo programs sndemo39, sndemo40, sndemo41, sndemo50 and sndemo51. After one is comfortable with the process of acquiring RINEX data files and processing them using the SatNav Toolbox, these large data files may be deleted from the hard disk if so desired.

2 Tutorial

Introduction

Satellite-based navigation systems such as the United States Global Positioning System (GPS) have revolutionized many aspects of modern life. Telephone networks depend upon GPS for timing. Surveyors depend upon GPS for millimeter accuracy in relative positioning. Navigation applications (the primary purpose for the original Department of Defense system) abound as well with thousands upon thousands of units being installed in land vehicles, boats, ships and aircraft (both civilian and military).

A realtime centimeter-level system for, say, a flight reference system does not appear out of the ether. A significant investment of man-power and money usually accompany the development of any high tech application. There is an unfortunate, but inevitable, cycle associated with this development. The following scenario might be all too familiar: 1) software development; 2) hardware interface construction; 3) field testing; 4) consternation over poor results (if not total failure); 5) analysis; 6) return to step 1, 2 or 3.

The delays can be critical in an industrial setting. Even in an academic setting, such valuable 'laboratory' experience is cumbersome both for the instructor as well as the students. It is apparent that the ability to simulate the process (including the field testing) on a desktop computer would be helpful indeed.

The Satellite Navigation (SatNav) Toolbox was designed specifically for this purpose. The Toolbox allows one to simulate the satellite

constellation, the propagation environment, the receiver measurements and the data processing. The Toolbox supports the simulation of virtually all satellite navigation applications from stand-alone positioning up through ambiguity resolution for differential carrier-phase applications.

Since its introduction in 1996, the SatNav Toolbox has been expanded to incorporate the ability to simulate measurements of signals at different frequencies (such as the GPS L2 and L5 signals), different constellations (such as Galileo and Glonass) and now has the functions necessary to process data from real receivers (through the receiver independent exchange, RINEX, data file format).

Satellite Constellation Emulation

Let's start off by looking at how the satellites are emulated. To determine the location of the satellites, time must be specified. Often, we are only interested in the satellites that are visible at a particular location. This can be accomplished by the following:

```
t = 39600;
usrllh = [40*pi/180  -90*pi/180  0];
usrxyz = llh2xyz(usrllh);
```

For this arbitrary example, 39600 seconds (or 11 hours) after midnight at the beginning of the GPS week has been chosen for the specific epoch. For most of the routines in the SatNav Toolbox, time is specified in terms of GPS time of week in seconds. This is GPS time and not local time. A user in western Europe, for example, will need to add one hour (during Winter time) or two hours (during Summer time) to convert from GPS time to local time.

We have specified a user location at 40 degrees latitude, -90 degrees

longitude and 0 meters above the 1984 World Geodetic System (WGS-84) reference ellipsoid. This places the user approximately in the middle of the State of Illinois in the United States (the local time in this location is 6 hours behind GPS in Winter and 5 hours behind GPS in summer). The conversion to radians is trivial in this particular example. `llh2xyz` converts the position to the WGS-84, earth-centered, earth-fixed Cartesian coordinate system (note: in this system, the x-axis emanates from the center of the earth and intersects the equator at the prime meridian; the z-axis coincides with the north pole and the y-axis completes the right-hand coordinate system).

Now that we have specified the time and the user position, we can determine which satellites are in view:

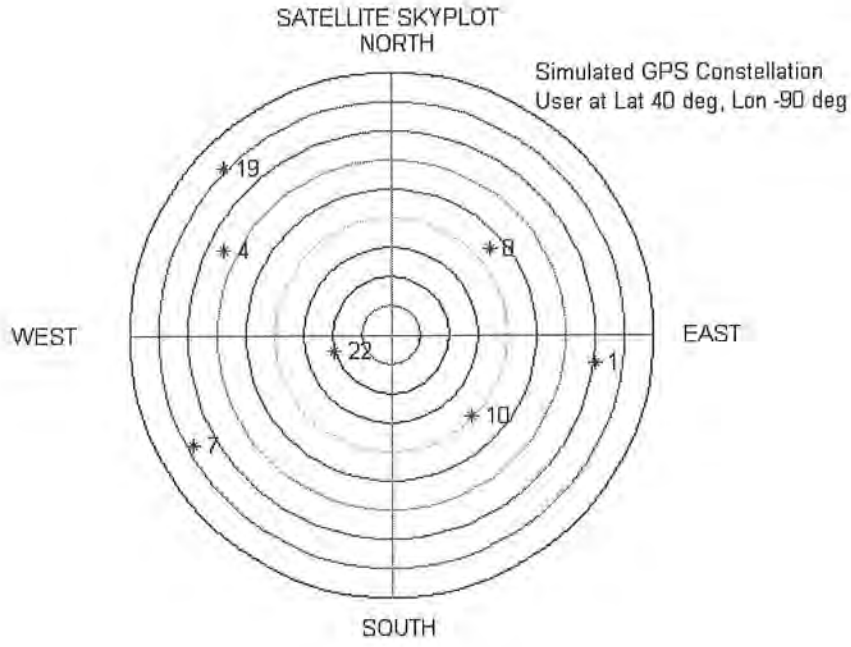
```
loadgps
[svxyzmat,svid] = gensv(usrxyz,t,0);
```

`loadgps` is an m-file which loads Keplerian orbital parameters for the GPS constellation (Galileo parameters may be loaded with `loadgalileo`). Note the default parameters assume perfectly circular Keplerian orbits. The small perturbations associated with the actual satellites are ignored and generally have no effect for most simulation purposes.

`gensv` determines the locations of all satellites (specified in this case by `loadgps`) and saves the positions of the visible ones in `svxyzmat` (in Cartesian coordinates). Note that the third parameter passed to `gensv` specifies the mask angle in degrees (0 in this case). The identification numbers of the satellites specified in `svxyzmat` are given in `svid`.

Finally, a satellite sky plot may be generated simply by:

```
skyplot(svxyzmat,svid,usrxyz)
```



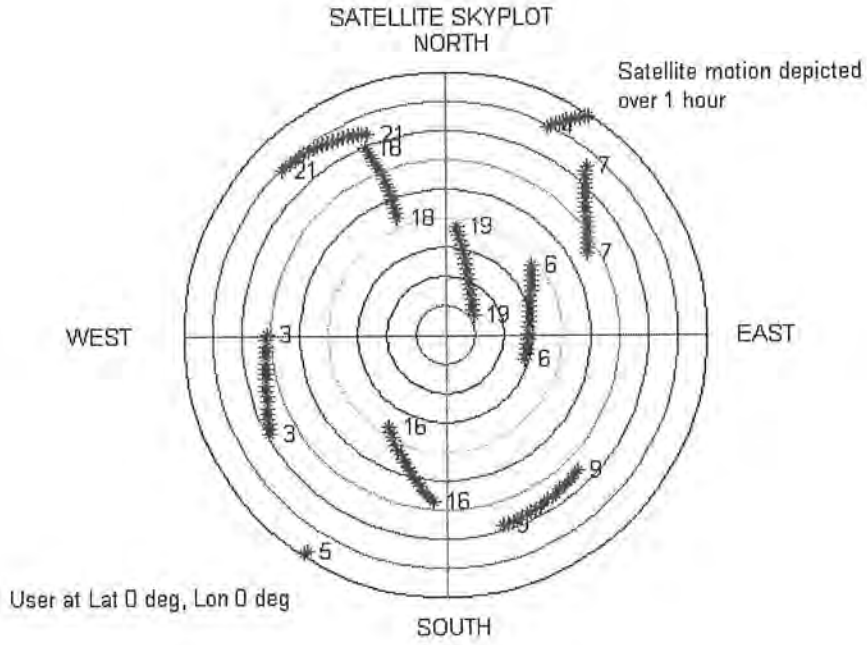
The rings represent elevation angle. The outer ring signifies the horizon (0 degrees) and each successive inner ring represents an increment of 10 degrees.

The complete code listing is given in `sndemo01.m`.

Satellite Motion

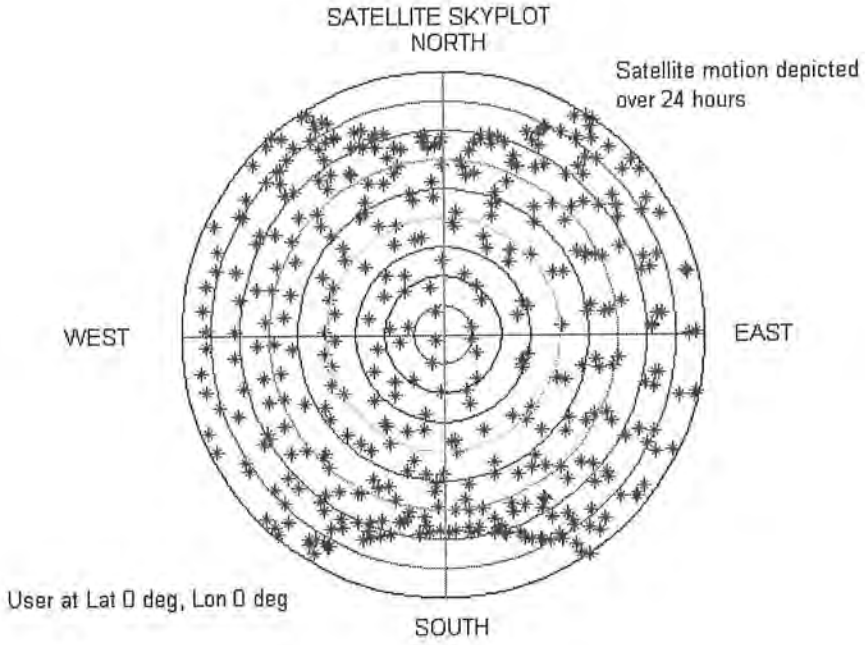
It is instructive to watch the paths of the satellites in the sky. `sndemo02.m` illustrates the motion of the satellites (visible to a user at 0 degrees latitude and longitude) over a one hour period (200 second time interval):

```
%  sndemo02.m          Short example of satellite
%                      motion through sky
clear all
close all
%
usrllh = [0*pi/180 0*pi/180 0];
usrxyz = llh2xyz(usrllh);
loadgps
for t = 0:200:3600,
  [svxyzmat,svid] = gensv(usrxyz,t,0);
  pause(0.1)
  skyplot(svxyzmat,svid,usrxyz,0,1)
  hold on
end
```

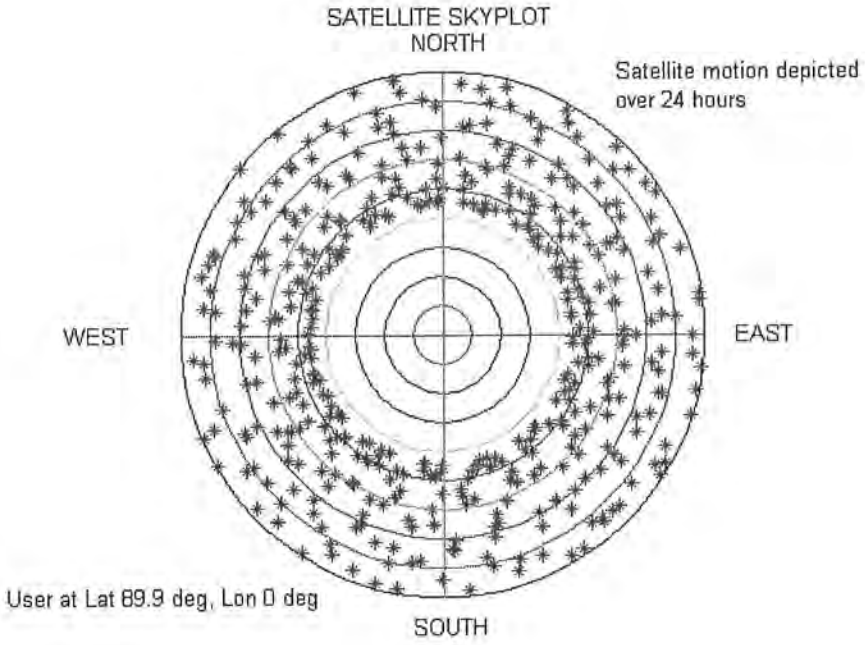


This plot is made possible by successively determining satellite positions over the interval, followed by plotting via `skyplot` (the MATLAB command `hold on`, has been issued to retain the results of previous plotting). Note that some of the satellites stay at essentially the same elevation angle as they move through the sky. Conversely, others hold essentially the same azimuth as they rise or set.

One might wonder if over a twenty-three hour and fifty-six minute period (which is the satellite ground track repeat time) the satellites may be found at all locations in the sky. `sndemo03.m` does this for the aforementioned user location:



It is interesting to note that there are regions to the north and south in which satellites never appear. This is due to the fact that all GPS orbital planes are inclined at 55 degrees with respect to the equator. This may be confirmed by running `sndemo04.m` (which repeats `sndemo03` but with a user located near the north pole):



As might be expected, for a user near the north pole, the satellites never exceed an elevation angle of 50 degrees. Note that they do not reach 55 degrees since the user is well above the equatorial plane.

Stand-Alone Positioning

Now that we can emulate the satellite constellation, pseudorange measurements are needed to compute the user state (receiver clock offset and three-dimensional position). Let's simulate half an hour of data collection and processing:

```
mpmat = mpgen(24,3600,1,54321);
usrllh = [0*pi/180 0*pi/180 0];
```

```
usrxyz = llh2xyz(usrllh);
loadgps
i=0;
randn('state',9083247);
bar1 = waitbar(0,'Calculating Position...    ');
for t = 41000:10:42800,
    i=i+1;
    [svxyzmat,svid] = gensv(usrxyz,t,2);
    [prvec,adrvec] =
genrng(1,usrxyz,svxyzmat,svid,t,…
                [1 1 0 1 1],[],mpmat);
    estusr = olspos(prvec,svxyzmat);
    enuerr(i,:) = ( xyz2enu(estusr(1:3),usrxyz)
)';
    terr(i) = estusr(4);  % true clk bias is zero
    waitbar(i/180)
 end
```

`mpgen` produces range-domain multipath errors for the entire constellation. `genrng` produces pseudorange (prvec) and integrated Doppler (adrvec) 'measurements' (note that `vec` is used simply to remind us that it is a vector of measurements; `adr` represents accumulated Delta range which is another term for integrated Doppler; `id` is not used for integrated Doppler since one might confuse it with identification number). The integrated Doppler values are not used in this example and have been included in order to demonstrate how they may be obtained. The vector:

[1 1 0 1 1]

tells `genrng` to add thermal noise and tropospheric delay, set Selective Availability to zero, and add multipath error and ionospheric delay to the pseudoranges (see `genrng` in the reference section for a complete explanation of the input parameters).

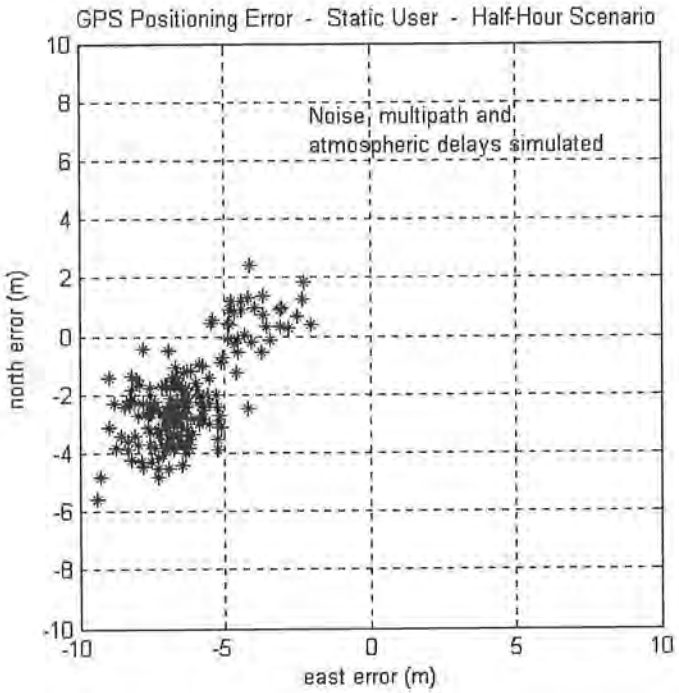
`olspos` computes the ordinary least-squares position solution (more

specifically, user state solution) for the given satellite positions and corresponding pseudorange measurements.

`xyz2enu` converts the user position to a local-level Cartesian coordinate system (with the x-axis pointing East, the y-axis pointing North and the z-axis pointing Up (local vertical), relative to the WGS-84 geoid). The second parameter passed to `xyz2enu` is the origin of the local system expressed in WGS-84 Cartesian coordinates. Since we have specified the true user location to be the origin, `enuerr` thus gives the east, north and vertical errors associated with the computed position.

Note that `genrng` does not emulate receiver clock offset. The computed clock offset is thus entirely error. We could have added a given number to all elements of `prvec` and `adrvec` to emulate a clock offset.
To finish this example, let's plot the horizontal positioning errors:

```
close
plot(enuerr(:,1),enuerr(:,2),'*')
axis([-10 10 -10 10])
axis('square')
axis('equal')
grid
title('GPS Positioning Error  -  Static User  -
Half-Hour Scenario')
ylabel('north error (m)')
xlabel('east error (m)')
```



The errors are dominated primarily by the effect of the ionosphere. This shows the beauty of a simulation. Any of the error sources can be simulated by in isolation to determine the impact on the position solution.

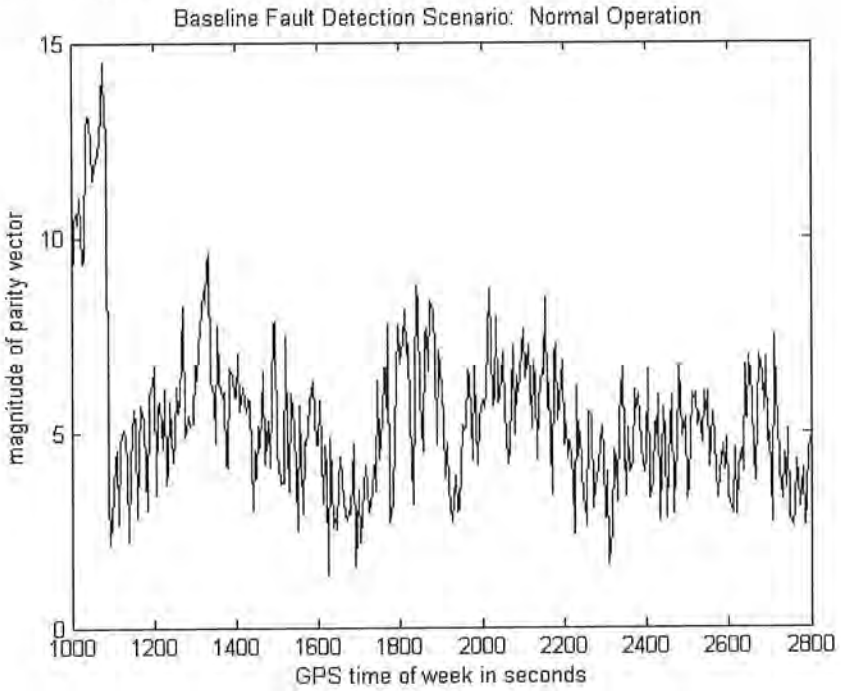
This example is given in `sndemo05.m`.

Fault Detection Via Parity Space Technique

Satellite-based navigation systems, as with all high tech marvels, are not perfect. In addition to the nominal error levels, occasionally large excursions are encountered. RAIM (receiver autonomous integrity monitoring) is the satellite-based navigation community's term for fault detection. The main idea is to examine the consistency in a redundant set of measurements. One way to do this is to decompose the measurement

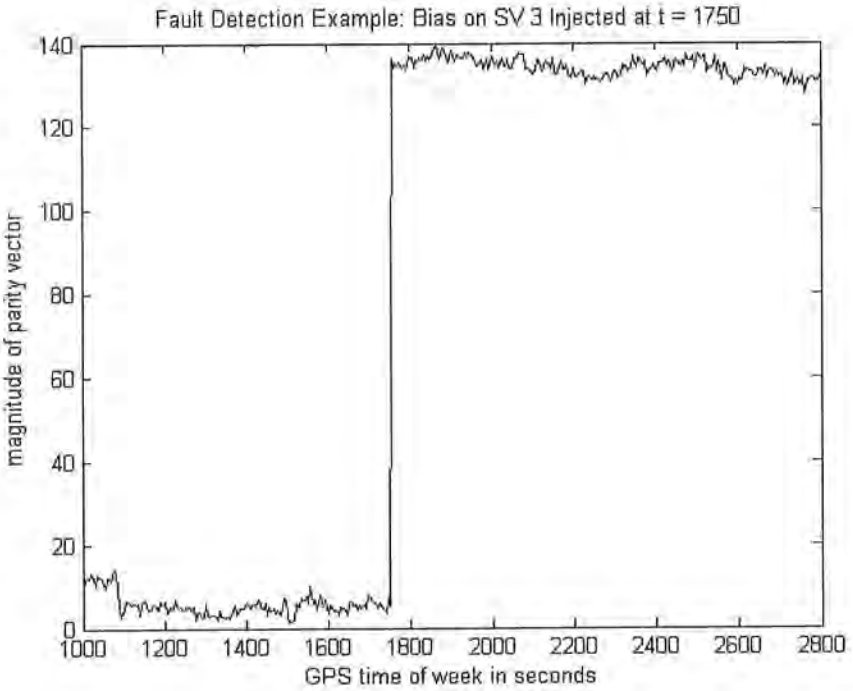
space into an estimation space and an orthogonal estimation error space. The estimation error space is also known as the parity space. The SatNav toolbox function `parityvec` (`raim` in previous versions of the toolbox) performs the necessary computations.

`sndemo06.m` illustrates what happens under normal circumstances. The inconsistency metric here is the norm of the parity vector (see `parityvec` and references [1] and [2] for more detail). Having just the nominal errors, the parity vector is well-behaved:

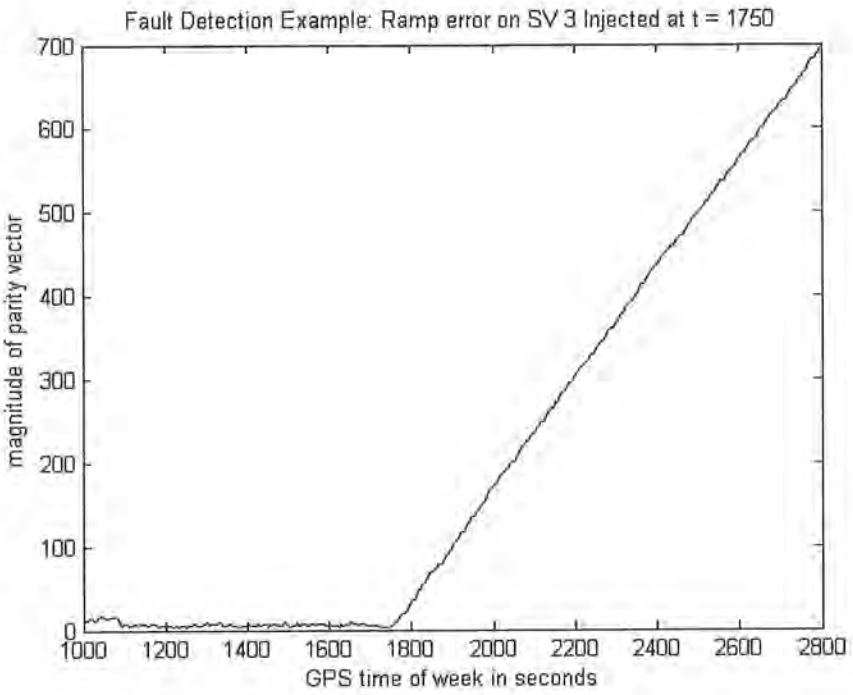


Notice the discontinuity in the plot after approximately 100 seconds. This is due to a change in the number of satellites being tracked.

`sndemo07.m` considers the same case only now a bias error of 200 meters is injected onto one satellite midway through the run:



Finally we consider the case of a ramp error of 1 meter per second injected onto the same satellite at the same time as before (`sndemo08.m`):



It should be pointed out that the behavior of the parity vector is dependent not only on the pseudorange errors but also on the satellite geometry. In safety critical navigation applications, the key is in setting thresholds such that the error is detected before an unacceptable position error is encountered.

Dilution of Precision (DOP)

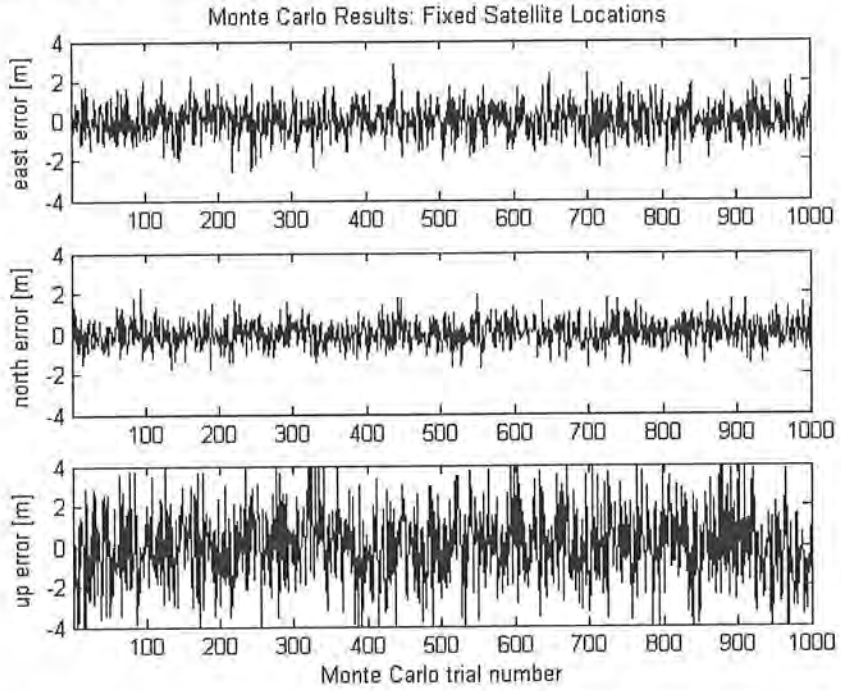
As was mentioned in the last section, satellite geometry plays a key role in the impact of pseudorange errors in the calculation of user state. To see this, consider `sndemo09.m`:

```
%  sndemo09.m   Compute DOP's and compare with
```

```
%                Monte Carlo trials
clear all
close all
%
time = 0;
usrllh = [0*pi/180 0*pi/180 0];
usrxyz = llh2xyz(usrllh);
loadgps
[svxyzmat,svidvec] = gensv(usrxyz,time,15);
randn('state',7912345);
for i = 1:1000,
    [prvec,adrvec] = genrng(1,usrxyz,svxyzmat,…
                      svidvec,time,[1 0 0 0 0]);
    estusr = olspos(prvec,svxyzmat);
    err(i,1:3) = (xyz2enu(estusr(1:3),usrxyz))';
    terr(i) = estusr(4);  % true clk bias is zero
    waitbar(i/1000)
 end
for j = 1:length(svidvec),
    svenumat(j,:) = …
              (xyz2enu(svxyzmat(j,:),usrxyz))';
end
dopvec = dops(svenumat,[0 0 0]);
subplot(311)
plot(1:1000,err(:,1))
axis([1 1000 -4 4])
ylabel('east error [m]')
title('Monte Carlo Results: Fixed Satellite
Locations')
subplot(312)
plot(1:1000,err(:,2))
axis([1 1000 -4 4])
ylabel('north error [m]')
subplot(313)
plot(1:1000,err(:,3))
axis([1 1000 -4 4])
ylabel('up error [m]')
```

```
xlabel('Monte Carlo trial number')
```

Note that time is fixed in order to maintain exactly the same geometry for the entire run. In order to verify the DOP calculations easily, thermal noise is the only error source emulated. Note that when the thermal noise flag is set to 1 in `genrng`, the standard deviation of pseudorange noise is 1 meter. Position error is computed by converting the user position solution result from WGS-84 Cartesian coordinates to local-level (east-north-up) coordinates with the true position at the origin. When the demo is executed, the east, north and vertical errors are plotted:



It is clear the satellite geometry favors horizontal positioning rather than vertical positioning in this case. It is widely known that satellite-based navigation systems perform worse for vertical positioning than for

horizontal positioning.

This can be quantified using the notion of dilution of precision (DOP). The key relationship is as follows (see reference [2] for a derivation):

```
xerr = xdop * prerr
```

where: `xerr` is the standard deviation of the errors in the x direction;
`xdop` is the dilution of precision in the x direction;
`prerr` is the standard deviation of the independent, identically Gaussian distributed pseudorange errors;

Although stated for the x direction, the relation holds for y, z, and t as well. Note that in reality, errors such as multipath, ionospheric delay and tropospheric delay are not independent or identically distributed. The DOP relations are therefore useful metrics of geometric quality, but not exact relationships in the real world.

In the demo, the satellite positions are converted from WGS-84 Cartesian coordinates to east-north-up coordinates relative to the true user position. This is done prior to the computation of the DOPs. In this manner, the x, y and z DOPs are east-DOP, north-DOP and up-DOP (better known as vertical DOP or VDOP). Since the standard deviation of the ranging error has been set to 1 meter in the demo, the equation above indicates the standard deviation of the errors should be equal to the DOPs. The demo results are:

```
ans =

   0.8248    0.6302    1.8718


stdxerr =
```

```
    0.8247

stdyerr =

    0.6345

stdzerr =

    1.8798
```

Where 'ans' is listing the first three elements of DOPVEC. Clearly there is good agreement between the theoretical results and the Monte Carlo results. Note the root-sum-square of the east and north position error (or standard deviation) is equal to the horizontal position error (or standard deviation). The horizontal position error is related to the ranging error through horizontal dilution of precision (HDOP). When the satellite positions have been converted to local level coordinates (such as in this demo), DOPVEC(4) gives the HDOP.

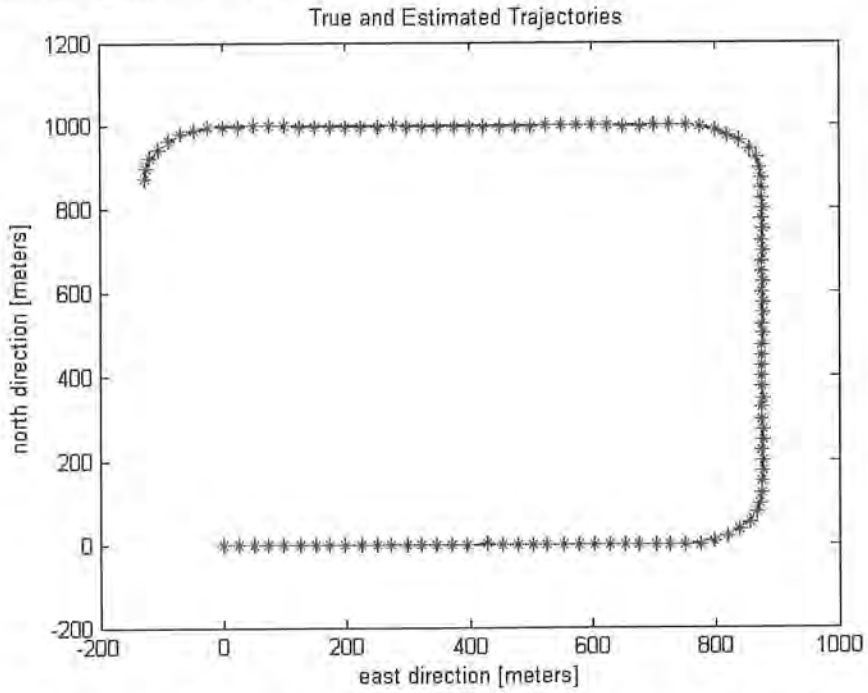
Vehicle Motion

It is often necessary to be able to emulate motion of a given vehicle. `pathgen` allows one to generate a horizontal vehicle trajectory with user-defined straight segments and constant-radius turns. Consider the example in `sndemo10`:

```
%   sndemo10.m
```

```
%
%  Demonstrate vehicle path generator
%
clear all
close all
%
mpmat=mpgen(24,3600,1,54321);
orgllh = [0*pi/180 0*pi/180 0];
orgxyz = llh2xyz(orgllh);
loadgps
startt=1000; t = startt; deltat=5;
segp = [150 90 .2; 150 90 .2; 150 90 .2];
usrenu = pathgen([0 0 0],[5 0],segp,deltat);
EndLoop = max(size(usrenu));
bar1 = waitbar(0,'Generating User Solutions...
');
for i = 1:EndLoop,
    t = t + deltat;
    usrxyz=enu2xyz(usrenu(i,:),orgxyz);
    [svxyzmat,svid] = gensv(usrxyz,t,5);
    [prvec,adrvec] = …
              genrng(1,usrxyz,svxyzmat,…
              svid,t,[1 1 0 1 1],[],mpmat);
    estusr = olspos(prvec,svxyzmat);
    estenu(i,:) = ( xyz2enu(estusr(1:3),orgxyz)
)';
    err(i,1:3) = estenu(i,1:3) - usrenu(i,:);
    terr(i) = estusr(4);  % true clk bias is zero
      waitbar(i/EndLoop)
  end
close(bar1);
plot(usrenu(:,1),usrenu(:,2),'-
',estenu(:,1),estenu(:,2),'*')
title('True and Estimated Trajectories')
ylabel('north direction [meters]')
xlabel('east direction [meters]')
```

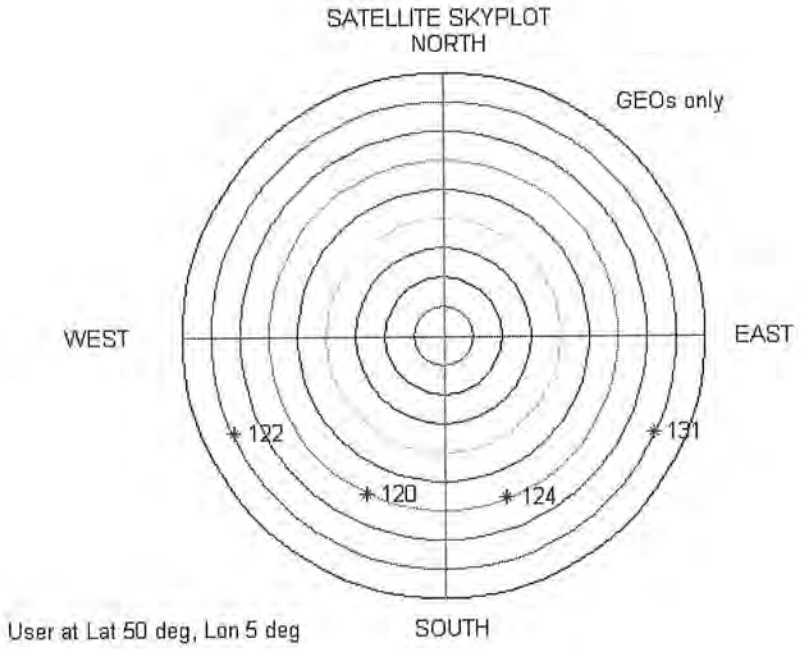
The program is similar to previous ones discussed in this tutorial with the exception of the user position. The first argument ([0 0 0]) passed to `pathgen` specifies that the generated path will start at the local origin. The second argument ([5 0]) specifies that the initial velocity is 5 meters/second in the east direction. The third argument specifies that there shall be three straight segments, each 150 seconds long and each shall be followed by 90 degree left hand turns governed by centripetal accelerations of 0.2 meters/second-squared. The actual path and the GPS positioning results are plotted when the program is run:



Since the errors are small relative to the scale of the user path, the true position and the GPS estimated position traces are very similar.

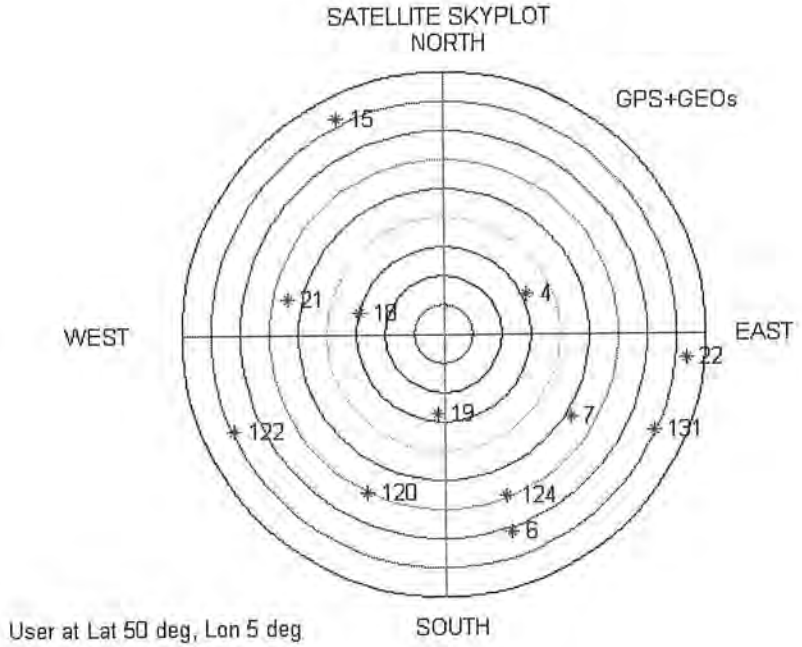
Geostationary Satellites

A variety of augmentations have been or are being developed for satellite-based navigation systems. Space-based augmentation systems (SBAS) involve geostationary satellites which broadcast GPS-like ranging signals and which also provide satellite health information and corrections for satellite ephemeris, clock and ionospheric errors. The SBAS developed by the United States is known as the Wide Area Augmentation System (WAAS) and its European counterpart is known as the European Geostationary Navigation Overlay Service (EGNOS). The geostationary satellites (GEOs) utilized by WAAS and EGNOS may be simulated through the `loadgeo` function. For a user at latitude 50 degrees North and longitude 5 degrees East, the skyplot of the SBAS GEOs is given by `sndemo11.m`:



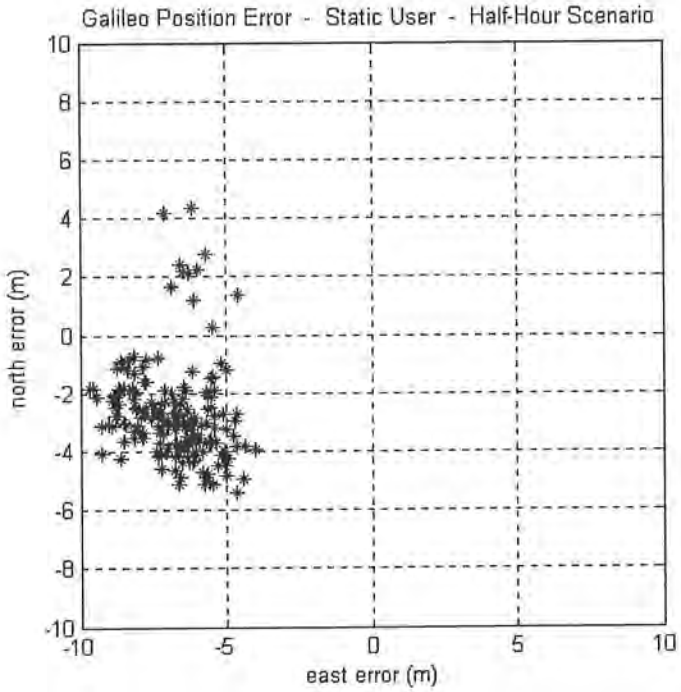
All, of course, are in a southerly azimuth (relative to this user in western Europe) and their elevation angles range from about 10 to about 30 degrees.

We can simulate these GEOs and GPS at the same time simply by loading both of their orbital files (`loadgeo` and `loadgps`) such as is done in `sndemo11.m`:



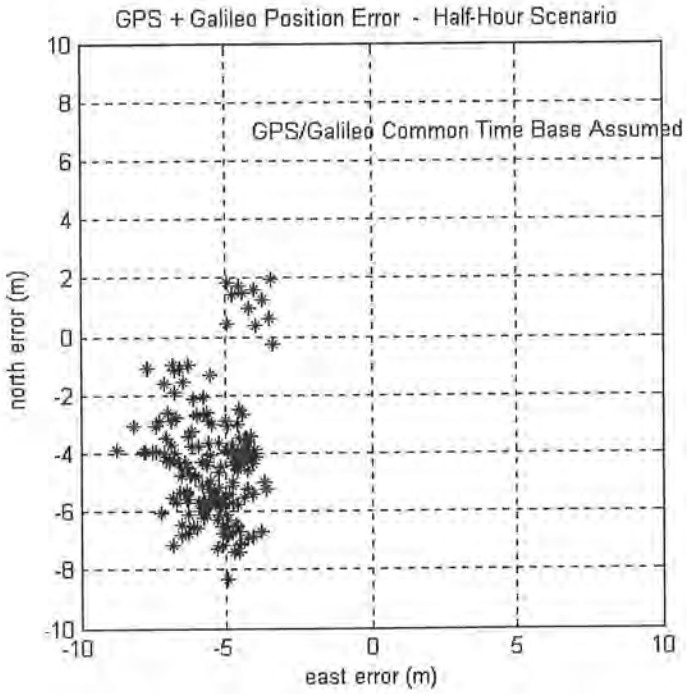
Galileo

The planned European counterpart to GPS is emulated by the Toolbox as well. `sndemo13` is the Galileo equivalent to `sndemo05` discussed earlier. The two programs are virtually identical with only one exception. The command `loadgalileo` is used (instead of `loadgps`) to load the Kepler parameters for a prototype Galileo constellation. The performance is similar to that of GPS shown earlier:



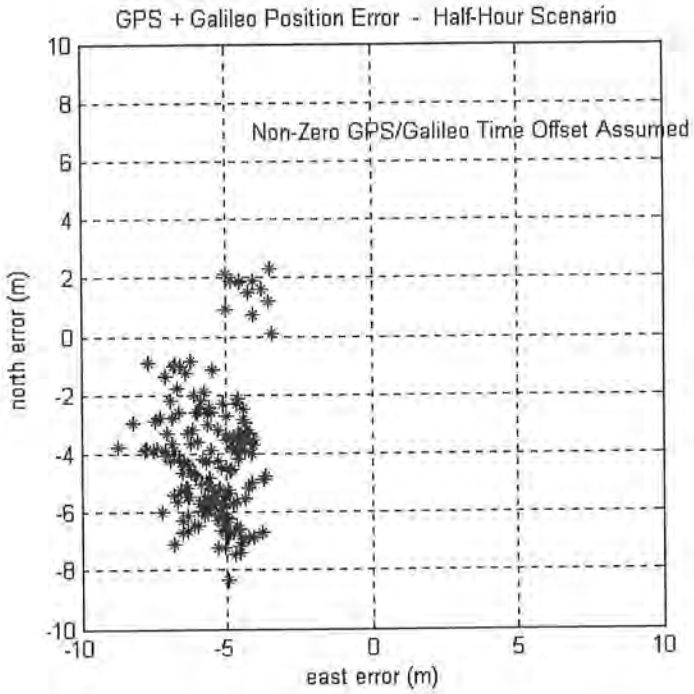
As with GPS, the position error is due to noise, uncorrected ionospheric and tropospheric delays and multipath.

If we treat GPS and Glonass satellites as pure equals, we can simply use Glonass as additional pseudoranging sources. This is done in `sndemo14`:



The results are only marginally better than with the use of either system separately.

It is likely, however, there will be a system time offset between GPS and Galileo. The user state vector thus has five elements: x, y, z user position, receiver clock offset and GPS/Galileo system time offset. This additional unknown must be accounted for when linearizing the pseudorange observation prior to performing the ordinary least-squares solution. The procedure is straightforward and merely consists of an additional column vector being added to the direction cosine (H) matrix. The elements of this column are zero for the rows corresponding to GPS measurements and are one for Galileo. `olsposgg` performs this function as is illustrated in `sndemo15`:

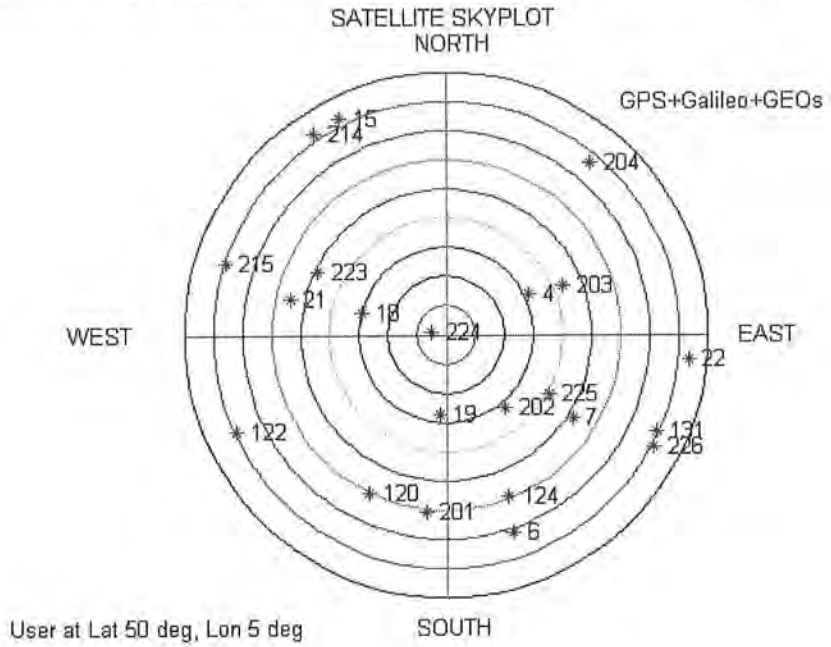


Strictly speaking, the need to solve for an additional unknown given the same number of measurements should result in a degradation of the estimate of all variables. This is not noticeable here since we are considering the full complement of both constellations. Five measurements are required but typically we would have have over thirteen.

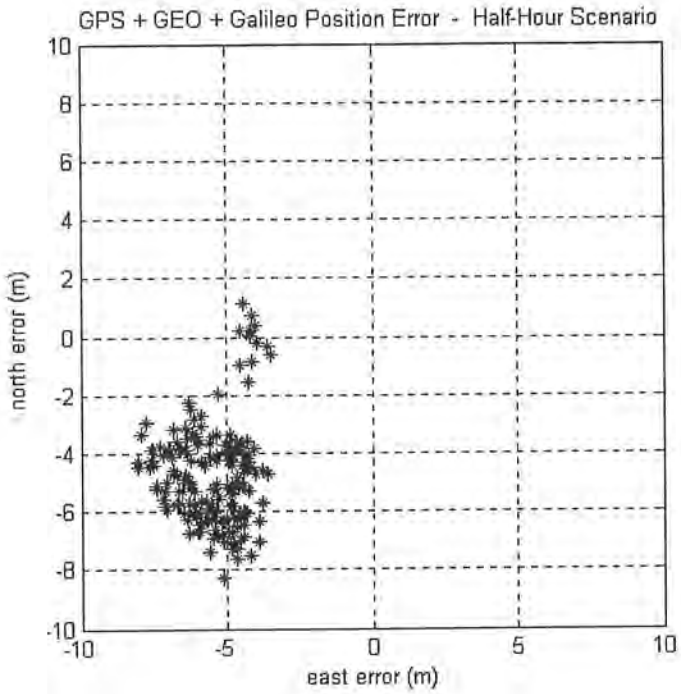
It should be pointed out that besides requiring at least five measurements, there is an additional requirement of at least one measurement from each system. Practically speaking, however, at least two are required from each system. If only one is provided, that measurement is simply used to estimate the system time offset.

GPS + Galileo + GEOs

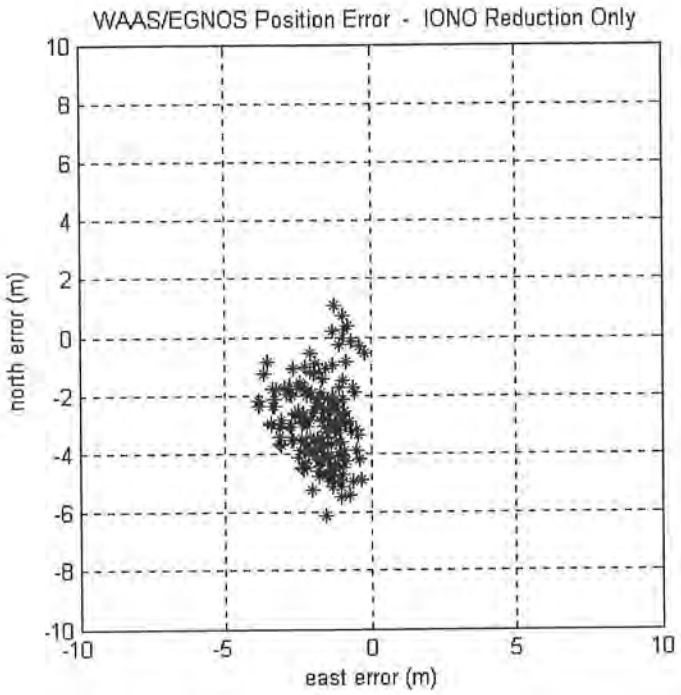
`sndemo16` provides an example skyplot for a western European user (50 degrees North, 5 degrees East) with all three types of satellites:



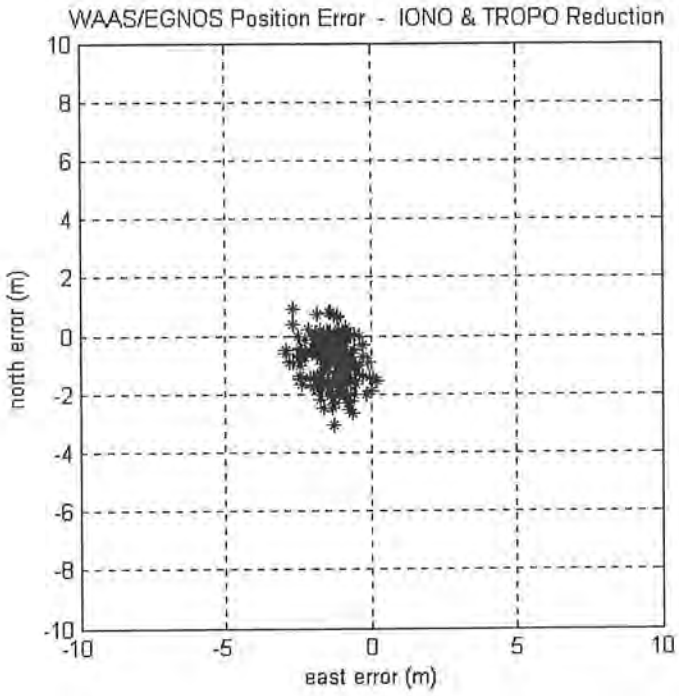
`sndemo17` simulates positioning results accordingly:



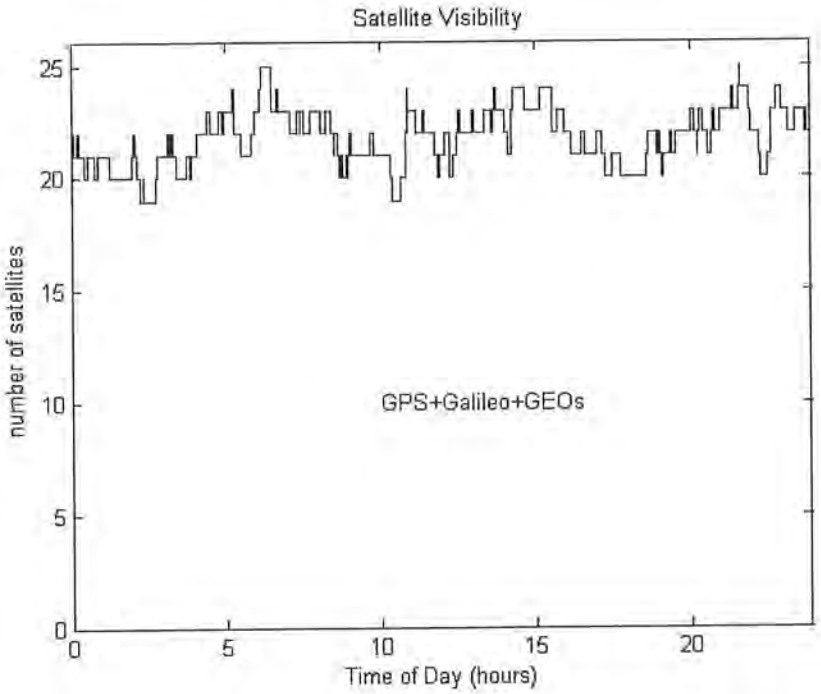
However, the SBAS satellites typically provide ionospheric correction information. If we assume this reduces the ionospheric error by 75%, the results are given in `sndemo18`:



Tropospheric error generally is reduced through a standard correction model. If we assume this model reduces the tropospheric error by 90% (typical performance), the results are given in `sndemo19`:



Clearly the error reduction techniques are extremely effective. It is interesting to consider how many satellites are visible over the course of an entire day. This is done in `sndemo20`:



Differential GPS (Range Corrections)

Virtually from the start of civilian use of GPS, one of the ways in which high accuracy was achieved was through the use of a differential architecture. The most straightforward implementation involves the use of range corrections. The reference station (at a surveyed, i.e., known location) computes the true ranges to the satellites and subtracts this from the measured pseudoranges. The resulting 'corrections' are applied to the user's measurements and then the user state estimate is formed in the traditional manner. An example is given in `sndemo21`:

```
%  sndemo21.m       DGPS (Range corrections)
```

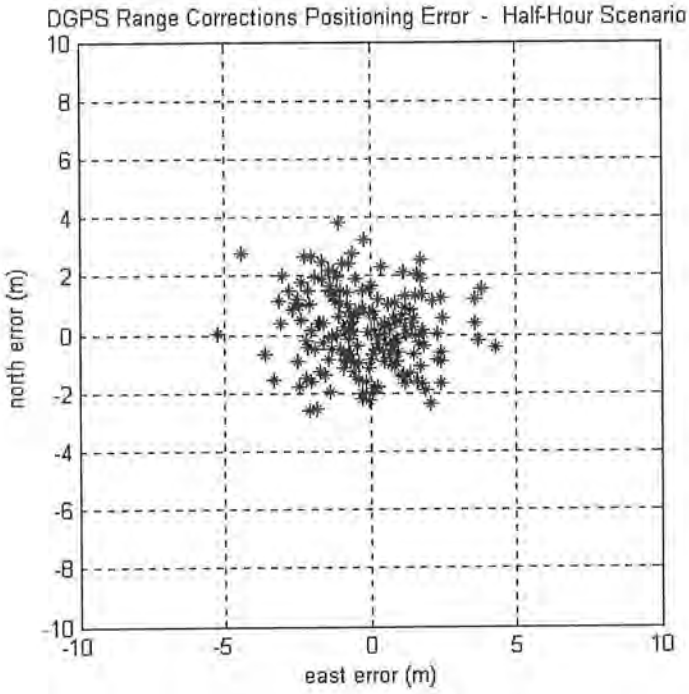
```
%                    No Carrier-Smoothing
clear all
close all
%
mpmatusr=mpgen(24,3600,1,22222);
mpmatref=mpgen(24,3600,1,33333);
refllh = [0*pi/180 0*pi/180 0];
refxyz = llh2xyz(refllh);
usrllh = [0.1*pi/180 0.1*pi/180 0];
usrxyz = llh2xyz(usrllh);
loadgps
i=0;
for t = 1000:10:2800,
    i=i+1;
%%%Reference Station
    clear svxyzmat svid prvec adrvec
    [svxyzmat,svid] = gensv(refxyz,t);
    prvec = genrng(1,refxyz,svxyzmat,svid,t,...
                   [1 1 1 0.1 1],samat,mpmatref);
    prc=9999*ones(1,24);
    for k = 1:max(size(svid)),
        true_range = ...
                  norm([svxyzmat(k,:) - refxyz]);
        prc(svid(k)) = prvec(k) - true_range;
    end
%%%User
    clear svid svxyzmat prvec adrvec
    [svxyzmat,svid] = gensv(usrxyz,t);
    prvec = genrng(2,usrxyz,svxyzmat,svid,t,...
                   [1 1 1 0.1 1],samat,mpmatusr);
    j=0;
    for k = 1:max(size(svid)),
        if prc(svid(k)) ~= 9999,
           j = j + 1;
           prvec_cr(j) = prvec(k) - prc(svid(k));
           svmat_cr(j,:) = svxyzmat(k,:);
        end
```

```
    end
    estusr = olspos(prvec_cr,svmat_cr);
    enuerr(i,:) = ...
                ( xyz2enu(estusr(1:3),usrxyz) )';
    terr(i) = estusr(4);  % true clk bias is zero
    clear prvec_cr svmat_cr j k
end
plot(enuerr(:,1),enuerr(:,2),'*')
axis([-10 10 -10 10])
axis('square')
axis('equal')
grid
title('DGPS {Range Corrections} Positioning Error
 -  Half-Hour Scenario')
ylabel('north error (m)')
xlabel('east error (m)')
```

Note that we are including multipath in our simulation of the pseudoranges. This is critical in a DGPS simulation since multipath does not get canceled in the differential correction. Note that we have guaranteed that the multipath at each receiver is independent by separate calls to `mpgen`.

Besides the calculation of the pseudorange corrections, the only other item of note has to do with application of the corrections. We must be wary of the fact that receivers in different locations do not always see the same satellites. For example, the reference station could be generating corrections for satellites that are not yet visible to the user. As a result, the satellite identification is paired with the correction and matched with the user's measurements.

The final results (as compared to those of `sndemo05`) show the DGPS eliminates the position bias. However, noise is actually increased since we are combining the measurements of two receivers:



Carrier-Smoothing

The accuracy achieved through DGPS (namely, elimination of the bias) can be improved further if carrier-smoothing is applied to the pseudoranges. In addition to multipath, thermal noise is independent between receivers and thus is not canceled in the differential correction. However, unlike multipath, thermal noise is a wide band phenomenon and thus can be reduced through filtering.

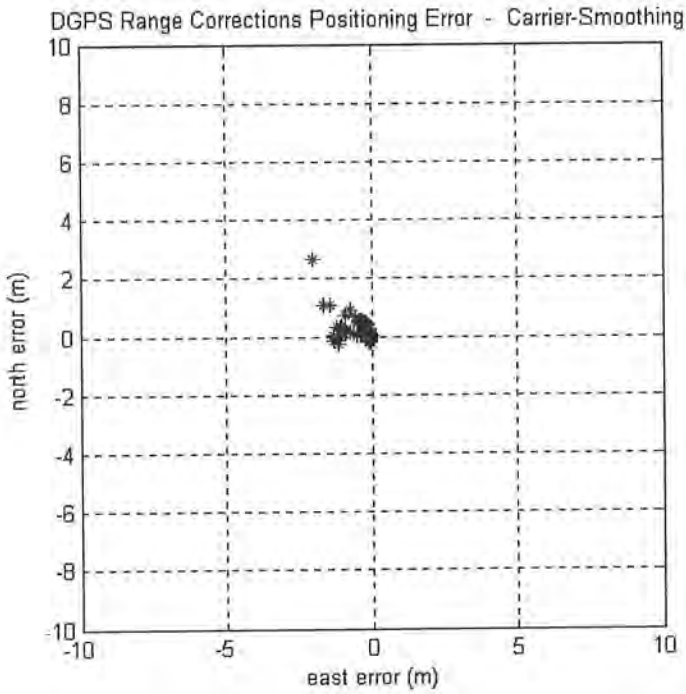
This filtering is accomplished through a process known as carrier-smoothing. The main idea is to estimate the bias (a.k.a. range ambiguity)

in the integrated Doppler measurement in order to convert it into an absolute measurement of range. The Satellite Navigation toolbox provides two functions for carrier-smoothing of the pseudorange measurements.

`hatch` implements the so-called Hatch filter. The Hatch filter estimates the aforementioned bias by a moving average of the difference between the pseudorange and integrated Doppler measurements for each satellite.

`compkalm` performs carrier-smoothing on each satellite using a single-state complementary Kalman filter. The filter uses the previous estimate plus the change in integrated Doppler as its prediction of the range for the current epoch. The pseudorange measurement is then blended with this prediction to form the next estimate.

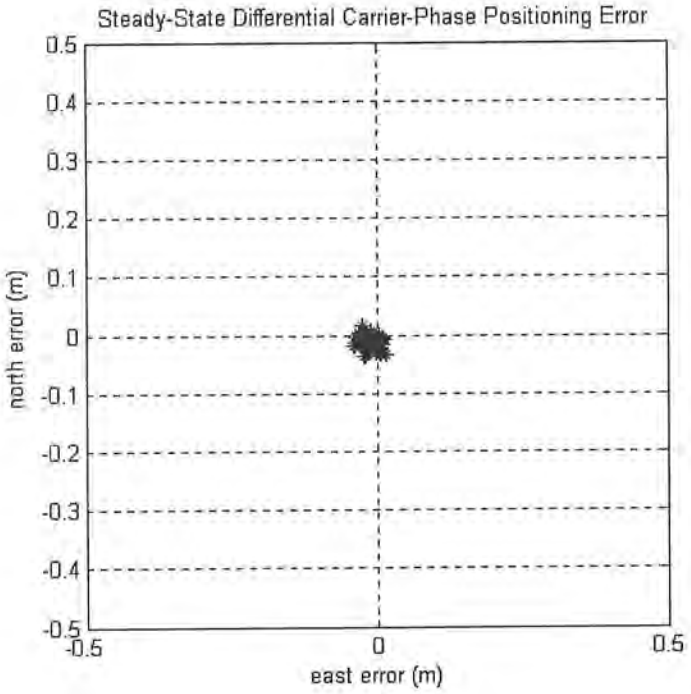
Let's see how the previous DGPS results are improved by using carrier-smoothing. The code is given in `sndemo22` where a Hatch filter is used:



The solutions are tightly clustered near zero with the exception of a few rogue data points. Although not readily apparent in this plot, these points occur at the beginning of the run while the filter is still converging.

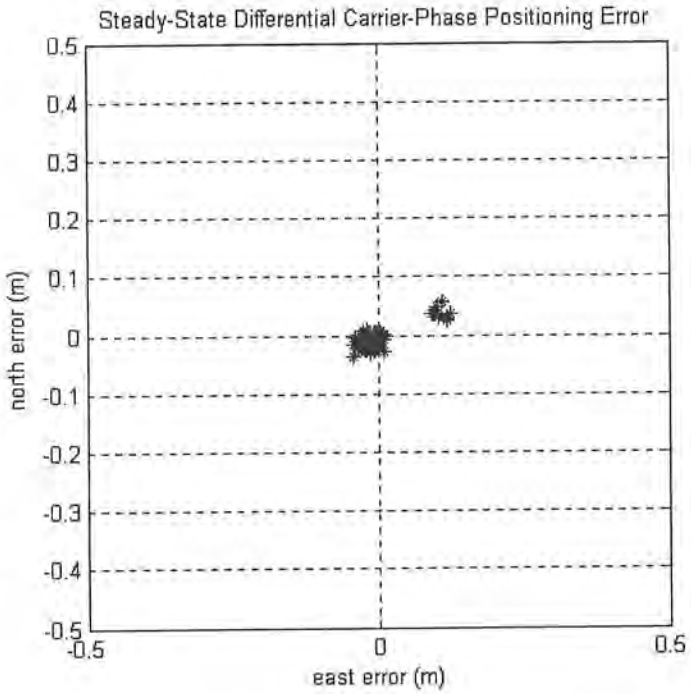
Steady-State Differential Carrier-Phase

Positioning accuracy on the order of one meter or less is not always acceptable. For the highest accuracy, we must use a carrier-phase only solution. The theory behind differential carrier-phase positioning is well beyond the scope of this tutorial and thus some familiarity on the reader's part is assumed (see references [2] and [4] for more details). The Toolbox supports these applications, however, as is shown in `sndemo23`:



The usual double-differences have been formed but the ambiguity resolution process has been sidestepped. Since `genrng` can generate measurements with or without errors, we are able to compute the true ambiguities and apply them to the measurements (see the next section for more on ambiguity resolution). As expected, the accuracy is now down to centimeters.

Careful examination of `sndemo23` would reveal that we have set the satellite mask angle to 15 degrees. The reason for this is as follows. Consider the results of `sndemo24`. `It` is the same as `sndemo23` except the mask angle has been lowered to 5 degrees:

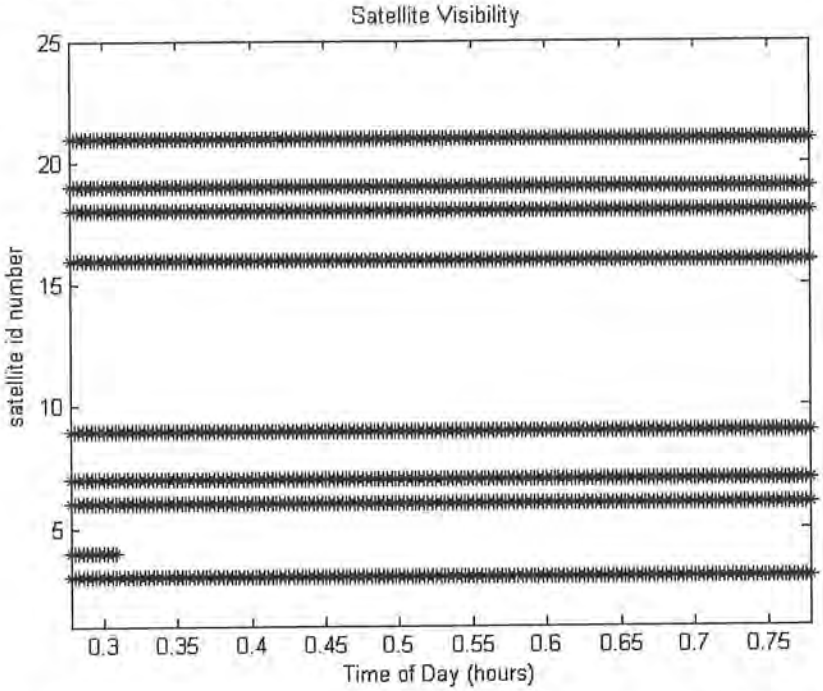


The cluster of data points is similar to the previous case with the exception, again, of a rogue cluster of points. We are not doing any filtering in this case so a transient is not possible. We did, however, lower the mask angle. Let's look at the satellite visibility over the duration of the run:

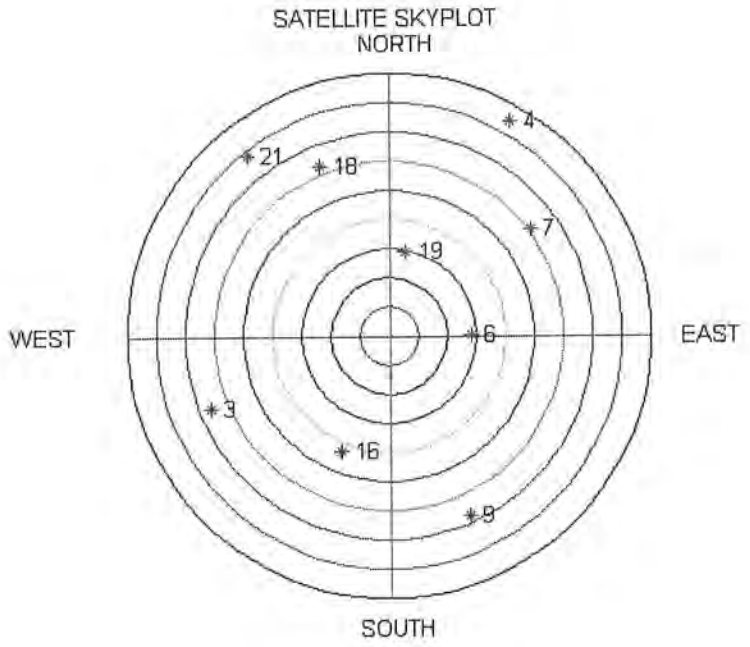
```
%  sndemo25.m    Satellite Visibility
%
clear all
close all
%
usrllh = [0.1*pi/180 0.1*pi/180 0];
usrxyz = llh2xyz(usrllh);
loadgps
tstart = 1000;  tstop = 2800;
```

```
tinc = 10;  sysflg = 1;
plotflg = 2;  maskang = 5;
vismat = satvis(usrxyz,tstart,tstop,tinc,...
                sysflg,plotflg,maskang);
```

Note that we have introduced the Toolbox function `satvis` which produces the following results:



We see that satellite 4 must be setting since it disappears shortly after the run starts. We can confirm this by running a skyplot (see the code listed in `sndemo19`):



Thus we see that satellite 4 is in the northeast quadrant at an elevation angle below 10 degrees. Although space does not permit it here, the problem can be traced to the atmospheric delay at low elevation angles (for example, try running a modification of `sndemo24`, but set the ionosphere and troposphere flags to zero in the two calls to `genrng`).

Although there is a loss of availability, high mask angles are needed in ultra-high accuracy systems.

Ambiguity Resolution

The Toolbox supports not only steady state processing, but initialization as well. In most real-time or near real-time systems (a.k.a., on-the-fly

ambiguity resolution; real-time kinematic), initial estimates of the double-difference ambiguities are formed using carrier-smoothed pseudoranges. The goal, then, is to solve for the difference between the true ambiguities and the initial estimates.

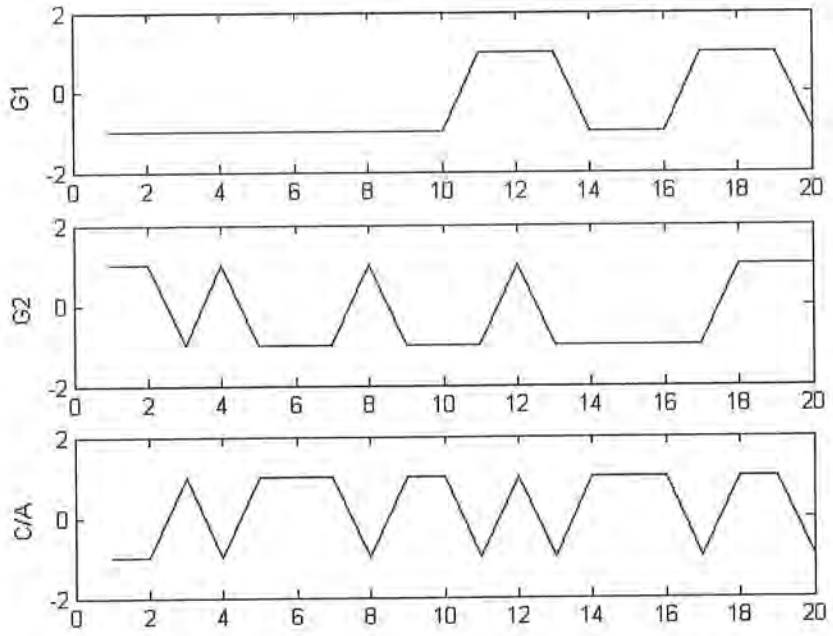
A great many ambiguity resolution techniques have been discussed in the literature. One of these (based on the discussion in reference [2]) relies on the same parity space technique described earlier for fault detection. Simply stated, the integrated Doppler double-differences, after being corrected with the true ambiguities, will yield parity vectors whose magnitude have the lowest mean over time as compared to those of the incorrect ambiguities.

See `sndemo27` for a detailed example of ambiguity resolution. References [2] and [4] contain further discussion of ambiguity resolution.

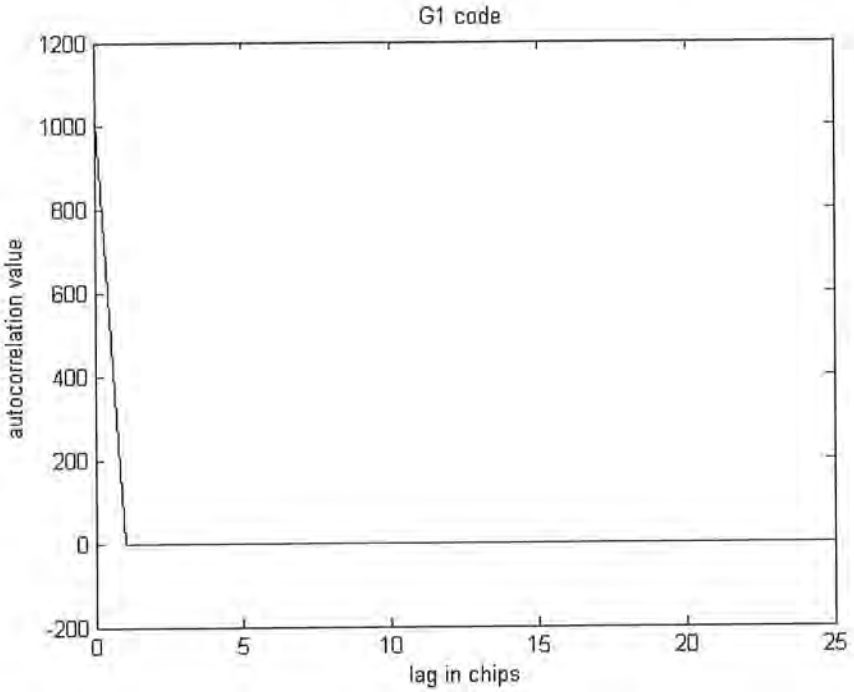
Pseudorandom Noise (PRN) Codes

At the heart of the GPS signal structure is the pseudorandom noise (PRN) code. The GPS C/A-codes are formed as the product of two maximal-length codes known as G1 and G2 [2-3]. As described in the GPS Interface Control Document [3], different C/A-codes are generated by changing the delay of the G2 code relative to the G1 code prior to forming their product. C/A PRN 10, for example, is formed by delaying G2 by 251 chips and then multiplying it with G1 (note that G1, G2 and the resulting C/A-codes are all 1023 bits (or chips) long).

The Toolbox function `prncode` can be used to generate the 37 codes in the C/A family ([3]). If the user is interested, all 1023 possible codes from this family can be generated using the Toolbox function `cacode`. Let's look at the first 20 chips of G1, G2 (shifted by 251 chips), and the resulting C/A PRN 10:

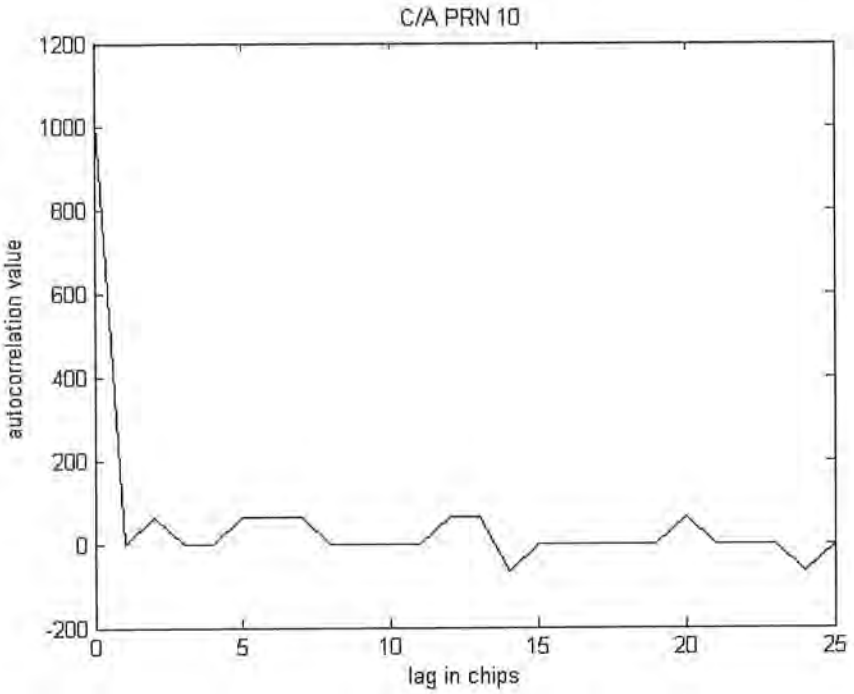


For the user's reference, this figure was generated by `sndemo28`. The multiple access properties associated with the C/A-codes lie in the nature of their autocorrelation and cross-correlation properties. Since the codes are periodic, it is possible to compute exact correlation values numerically using circular correlation. This is implemented in the Toolbox function `gpscor`. As an example, `sndemo29` computes and plots the first 25 lags of the G1 autocorrelation function:

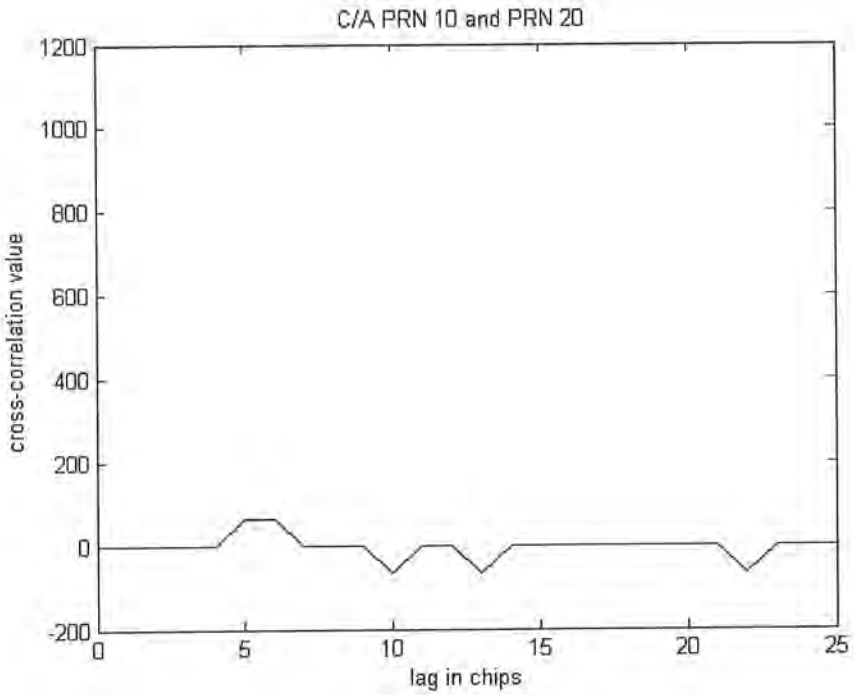


As the theory would predict, it starts off with a peak of 1023 at lag 0 and then drops to -1 (you can run `sndemo29` and then use the MATLAB axis command to zoom in and convince yourself that the function does indeed rest at -1). The function stays at -1 until lag 1023 and then the function repeats. Although G2 is a different code, it is still a 1023 length maximal length code (m-sequence) and thus has exactly the same autocorrelation function as G1.

Maximal length codes do not provide multiple access capability, however, (e.g., Glonass satellites all use the same m-sequence but all transmit on different frequencies) and thus GPS uses the C/A-codes (or P-codes for authorized users). We can inspect the behavior of the C/A-code autocorrelation functions in the same way we did for G1. `sndemo30` provides the results for C/A PRN 10:

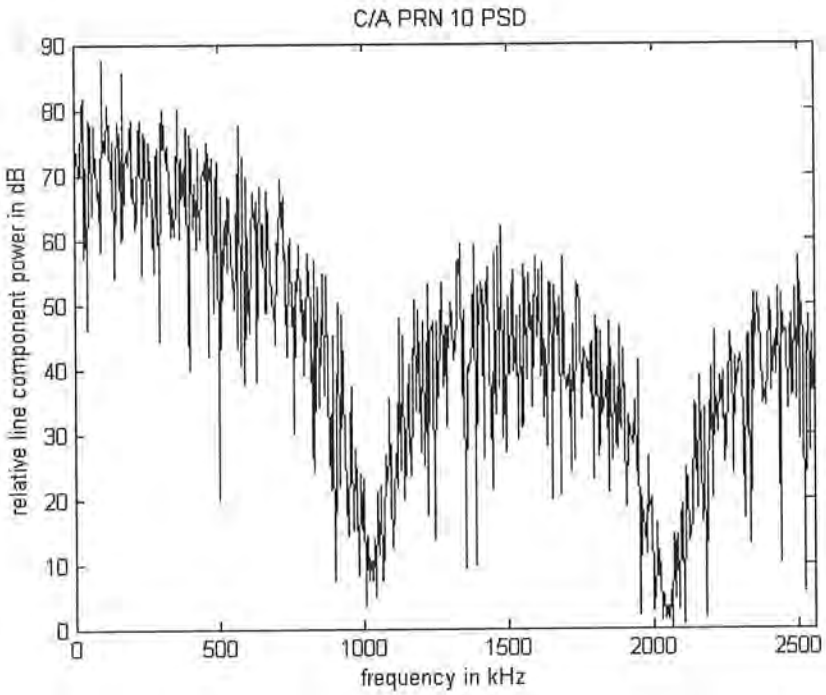


The sidelobes in the autocorrelation function make the process of code acquisition a bit more difficult for the receiver. This, however, is the price paid for the multiple access properties. The latter can be shown by computing the cross-correlation between two different C/A-codes (as is done in `sndemo31`):



Note that there are sidelobes but no main lobe. Admittedly, we have not plotted out all 1023 possible lags, but the user can verify that they are no different.

To close out this section, we consider the problem of computing the power spectral density of a C/A-code. Communications theory tells us the power spectral density (PSD) is the Fourier Transform of the autocorrelation function and thus our previous example gets us halfway there. Well, almost. We sped things up a bit by only computing the first 25 lags. In order to do the PSD properly, we will need to allow `gpscor` to compute all 1022 lags. The results, as given by `sndemo32`, are satisfying:



As expected, we see nulls at integer multiples of the C/A-code chipping rate (1.023 MHz).

Multipath Analysis

As was indicated in earlier sections, multipath and noise are the two error sources that do not cancel in DGPS. Since pseudorange noise can be filtered through carrier-smoothing, multipath remains as the primary error source in DGPS.

Often one desires to quantify the level of multipath influence in a given data set. The standard technique takes advantage of the fact that the

influence of multipath on the integrated Doppler (centimeters or less) is negligible compared to its influence on the pseudorange (meters or greater).

Error sources such as Selective Availability, tropospheric delay, and satellite and receiver clock offsets are all common to the pseudorange and integrated Doppler. By differencing these two observables, the common error sources cancel. What remains is multipath, noise, ionospheric divergence (the ionospheric delay is equal in magnitude but opposite in sign on the pseudorange and integrated Doppler), and the range bias in the integrated Doppler.

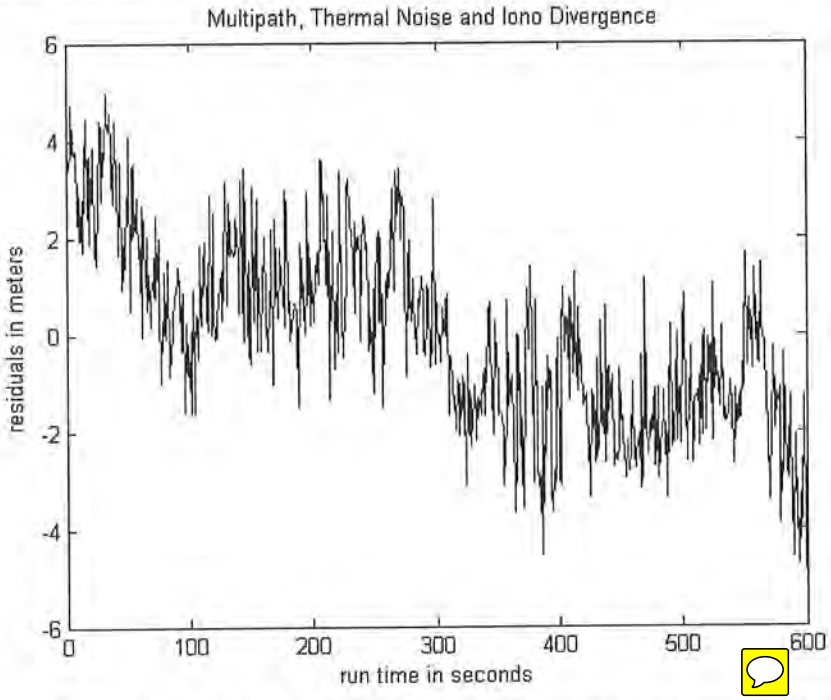
At this point we must accept the fact that the bias components in these errors are indistinguishable from one another. The mean of the pseudorange/integrated Doppler difference can thus be removed as long as the aforementioned is remembered. What remains is thus the noise, the variations in the multipath, and the ionospheric divergence. Sometimes this is referred to as the residuals.

Under quiet ionospheric conditions, the divergence shows up as a slow trend in the residuals. For data segments on the order of a fraction of an hour, the divergence can be estimated as a first or second order polynomial and then removed from the residuals. Noise and multipath then remain.

To the extent that the multipath variations are slow with respect to the noise, the two error sources can be distinguished. Care must be taken in the entire process, however. If the data segment is too short (i.e., a minute or two), a slow multipath variation could be removed with the other biases. Even for longer data segments, multipath could also be inadvertently removed along with the ionosphere.

With these cautions in mind, let's use the Toolbox to illustrate the basic procedure. `Sndemo33` sets a user at an equatorial location (the

intersection of the equator and the prime meridian) and collects pseudorange and integrated Doppler data for ten minutes (at a 1 Hz data rate). The two observables for satellite 8 are used in this differencing procedure. After removing the bias, the following remain:



We clearly see the noise as the high frequency component, the multipath as the medium frequency component, and the ionospheric divergence as the low frequency trend. We decide that the ionospheric divergence can be removed with a straight-line fit. This can be done as follows (note that resi is the variable which was plotted in the previous figure):

```
x = 1:600;
p = polyfit(x,resi,1);
yfit = polyval(p,x);
plot(x,resi,x,yfit)
title('Residuals and Straight-Line Fit')
```

We plotted the residuals and the fit just to satisfy ourselves that the curve fit has been successful. We can now remove it from the data:

```
resi_fit = resi - yfit;
plot(x,resi_fit)
title('Multipath and Noise with Iono Removed')
ylabel('residuals in meters')
xlabel('run time in seconds')
```

`sndemo26` simulates a run later in the day (just before local noon). This was done on purpose to demonstrate the effects of a more substantial ionosphere.

Support for Broadcast Almanac

Although perfectly circular Keplerian orbits are acceptable for most simulation purposes, the ability to emulate the actual GPS orbits on a given day is required in certain analysis applications. The toolbox supports this by allowing the user to input Yuma-formatted broadcast almanac data files. Yuma-formatted almanac files are available in a variety of places on the internet. One very convenient site is operated by the U.S. Coast Guard Navigation Center:

http://www.navcen.uscg.mil/gps/almanacs/default.htm

The toolbox function `loadyuma` accepts the Yuma-formatted data files and stores the parameters in global memory. The toolbox functions `gensvalm` and `svposalm` use the almanac parameters to determine the positions of the visible satellites.

It should be noted that around the time of the GPS week roll-over, the YUMA format was changed. `loadyuma` can decode both formats. Two

example files, `yuma1.txt` and `yuma1023.txt`, are included on the diskette.

Demonstration programs `sndemo34.m`, `sndemo35.m` and `sndemo36.m` help to illustrate the use of these new functions for static, dynamic and differential GPS applications. `sndemo37` gives an example of how to determine satellite visibility using Yuma-format almanac data.

P-Code and GPS Modernization (L2, L5 and more)

Full dual-frequency P-code measurements also can be emulated with the SatNav Toolbox. In addition, C/A-code measurements on L1 (1575.42 MHz), L2 (1227.6 MHz) and L5 (1176.45 MHz) can be simulated. This has been achieved by modifying the functions `ionogen` and `genrng`.

`ionogen` has been updated to support any arbitrary carrier frequency. In addition, the user has the option to specify the maximum and minimum ranges of total electron content (TEC).

Along with `ionogen`, `genrng` has been updated to accommodate a user-specified carrier-frequency. Additional information has been provided in the reference section covering `genrng` that will allow the user to exploit the function to emulate P-code measurements on dual-frequencies.

It should be pointed out that `genrng` calls `ionogen`. Hence, the carrier-frequency-dependent ionospheric delays are passed on to the function that emulates pseudorange and carrier-phase measurements.

User-defined Satellite Constellations

Toolbox users are not constrained to the constellation models provided on the CD. Constellations may be defined in ASCII text files, then loaded into MATLAB and saved as MAT-files. One can then modify the functions such as `loadgps` in order to utilize them with the other toolbox functions.

The best way to see how to construct the ASCII files is to examine the ones used for the toolbox's pre-defined constellations. For example, the file `gpskep.dat` is the ASCII equivalent of `gpskep.mat`. These files have six columns of data:

Column 1: satellite identification numbers
Column 2: mean anomalies (in degrees) at the reference time
Column 3: longitudes of the ascending node (in degrees) at the weekly epoch
Column 4: orbital radii (in meters)
Column 5: inclination angle (in degrees) of the orbital plane
Column 6: reference time for the orbital parameters

References

[1] Global Positioning System - Theory and Applications, Vol. I, Parkinson, B. and J. Spilker, Jr., editors, American Institute of Aeronautics and Astronautics, Washington, D.C., 1996.

[2] Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

[3] ICD-GPS-200, NAVSTAR GPS Space Segment/Navigation User Interfaces (Public Release Version), ARINC Research Corporation, 11770 Warner Ave., Suite 210, Foutain Valley, CA 92708, July 3, 1991.

[4] Global Positioning System: Signals, Measurements, and Performance, Pratap Misra and Per Enge, Ganga-Jamuna Press, Lincoln, MA, 2001.

3 What's New in Version 3.0

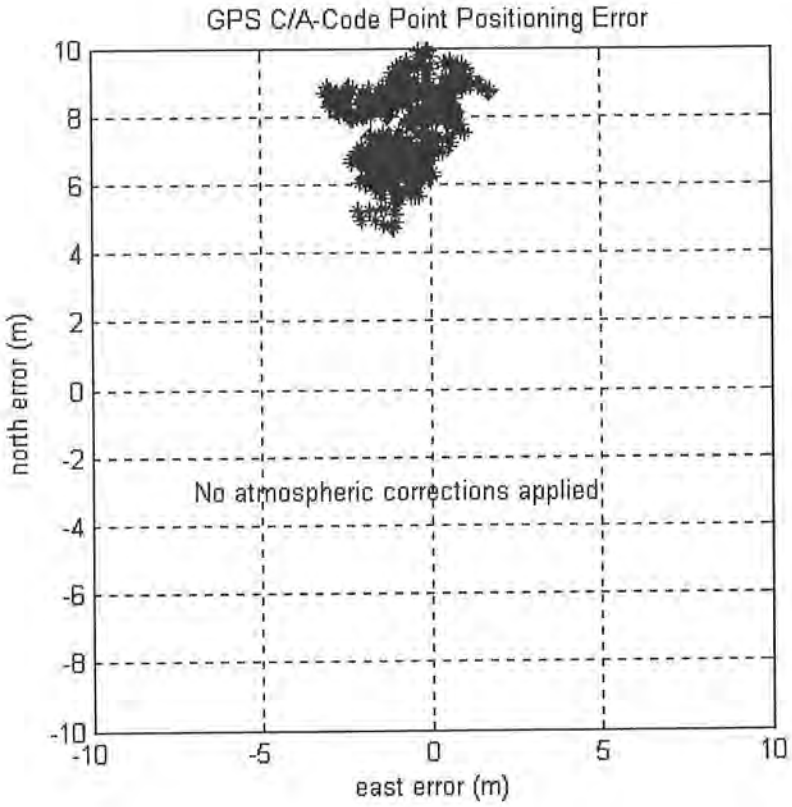
Real Data Processing: RINEX2 Support

With the release of Version 3.0, the SatNav Toolbox now provides support for real data files in the Receiver Independent Exchange format version 2 (RINEX2). The toolbox provides functions to interpret both ephemeris files (known as Navigation Data files: NAV files) and raw measurement files (observation files).

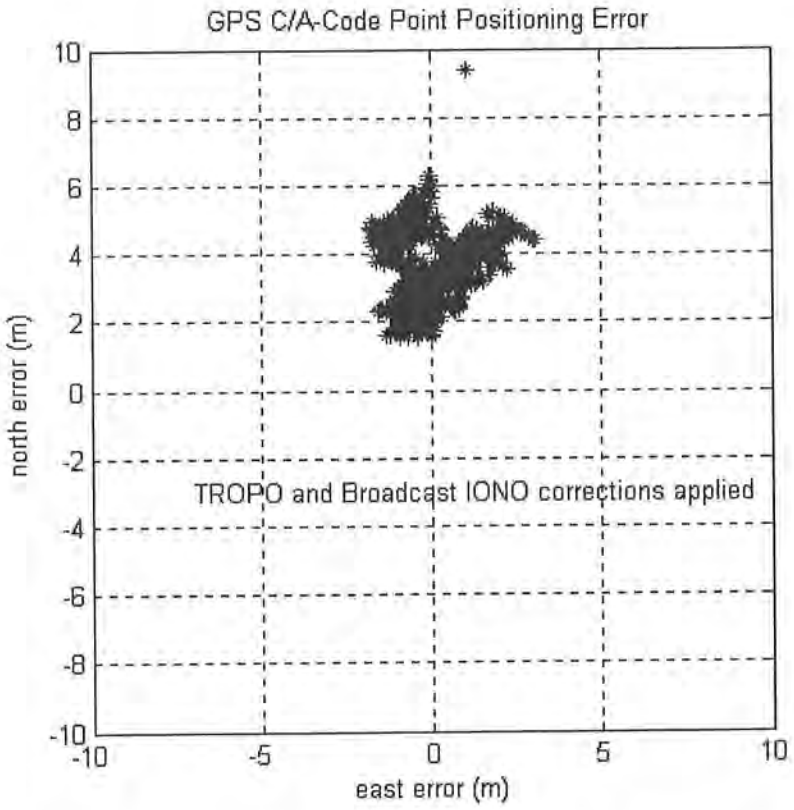
The RINEX files are ASCII files formatted with a Fortran compiler in mind. To facilitate the usage of the data in MATLAB, the functions `storerinnav` and `storerinob` have been provided to read and interpret the data files and store the data in a MAT-file for ease of use later. The user must be cautioned that these interpretation/translation routines tend to run quite slowly, even on machines with clock rates above 1 GHz. `Sndemo50` and `sndemo51` have been provided to illustrate how the RINEX files may be interpreted and stored along with an example point-positioning routine to process the data files.

`sndemo38` shows an example of the determination of satellite visibility using a RINEX nav file.

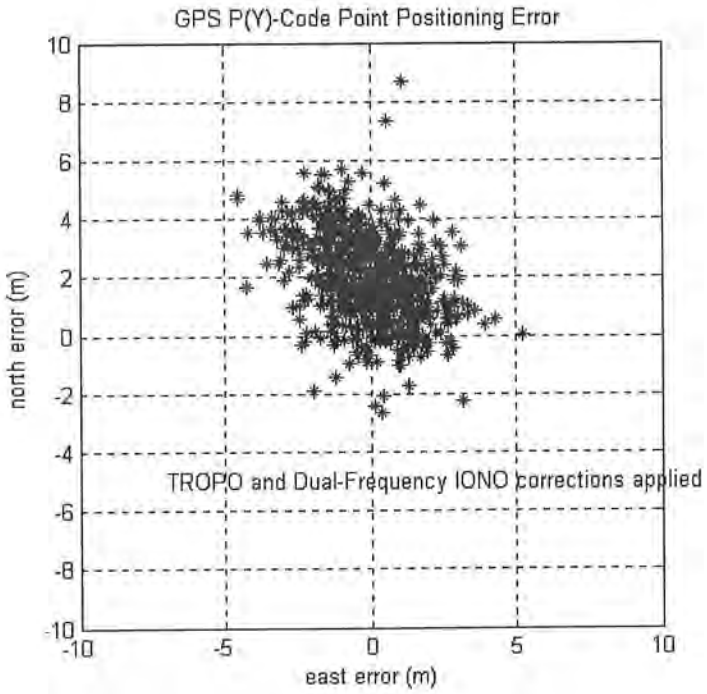
`sndemo39` shows an example of the determination of user position given C/A-code pseudoranges without corrections:



`sndemo40` extends demo 39 by incorporating the standard tropospheric correction and the broadcast ionospheric correction. A substantial reduction in bias is achieved:



`sndemo41` performs point-positioning with P-code pseudoranges and incorporates dual-frequency ionospheric corrections along with the standard tropospheric. The position bias is reduced but an increase in noise clearly is present. This is the result of the dual-frequency correction:



The noise could be reduced through carrier-smoothing of the pseudorange measurements as was demonstrated in earlier sections in this manual.

When reading through demos 39 through 41, the user will note the presence of an earth rotation correction. This is done to account for the fact that the satellite positions are calculated at their respective times-of-transmission, but these positions need to be expressed in the earth coordinate frame at the users' time-of-reception. Since the earth is rotating during the signal travel time, the earth-fixed coordinate system is also rotating. If this earth rotation correction is not made, a latitude-dependent position error will result. This error typically is between 20 and 25 meters for users in mid-latitudes.

Galileo and Geostationary Satellites

The ability to simulate the Galileo system has been incorporated into Version 3.0. See the earlier tutorial section for more details about Galileo and geostationary satellite simulations.

4 Reference

Introduction

This section contains a listing of all the M-files in the Satellite Navigation Toolbox. Details regarding input and output parameters are provided. Note that much of this information is also available through the online Help facility.

cacode

Purpose

Generate C/A-code sequence.

Synopsis

```
[ca,g1,g2]=cacode(g2shift)
```

Description

`g1` and `g2` are the maximal length sequences used to form the C/A-codes. Although all C/A-codes are given by the product of G1 and G2, each C/A-code is uniquely determined by the shift of the G2 sequence prior to the forming of the product.

Since the sequences are 1023 bits (chips) long, `g2shift` can uniquely range from 0 to 1022.

Examples

See tutorial section on pseudorandom noise codes.

Algorithm

Although digitally speaking, the sequences are composed of 0's and 1's and are combined

using the exclusive-or function, cacode uses +1's and -1's and combines with multiplication.

See Also

`gpscor`

References

ICD-GPS-200, NAVSTAR GPS Space Segment/Navigation User Interfaces (Public Release Version), ARINC Research Corporation, 11770 Warner Ave., Suite 210, Foutain Valley, CA 92708, July 3, 1991.

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

compkalm

Purpose

Perform carrier-smoothing of the pseudorange measurements via complementary Kalman filtering.

Synopsis

```
[prsm,Pvec,prevadr]=...
  compkalm(pr,adr,svid,P,Q,R,Pvec,prevsm,prevadr)
```

Description

A single-state complementary Kalman filter uses the integrated Doppler (carrier-phase) data in **adr** to filter the pseudorange data in **pr** with the resulting smoothed pseudoranges output in **prsm**.

svid is a vector of satellite identification numbers (corresponding to the measurement vectors) which is output from the function **gensv**.

P, Q, and R are the usual Kalman filter variables:

P = initial value for estimation error variance (e.g., 10 m^2)
Q = Process noise variance (e.g., 0.001 m^2)
R = measured Pseudorange thermal noise variance (e.g., 4 m^2)

Pvec = vector of prediction error variances for each measurement.

`prevsm` = vector of previous smoothed pseudoranges for all satellites.
`prevadr` = vector of previous integrated Doppler measurements.

When `compkalm` is first called, `Pvec`, `prevsm`, and `prevadr` must be set to [].

Algorithm

The complementary nature of the Kalman filter mechanization stems from the use of the integrated Doppler measurements to form the prediction. Specifically, the predicted pseudorange for the current epoch is the sum of the estimated (i.e., smoothed) pseudorange from the previous epoch plus the difference in integrated Doppler values of the current and previous epochs.

See Also

`hatch`

References

Van Graas, F., and M. Braasch, "GPS Interferometric Attitude and Heading Determination: Initial Flight Test Results," NAVIGATION, Journal of the Institute of Navigation, Vol. 38, No. 4, Winter, 1991-1992.

Cosentino, R., and D. Diggle, "Pseudorange (Code) Smoothing," Section 8.3.1.3, Understanding GPS: Principles and Applications, E. Kaplan, Editor, Artech House Publishers, Boston, 1996.

dayofweek

Purpose

Calculate the day of the week given the year, month and day of the month.

Synopsis

```
daynum = dayofweek(year,monthnum,dayofmonth)
```

Description

INPUTS

year = year number; only valid for the Gregorian calendar (the calendar the entire world now uses); this function is not valid for the Julian calendar which was in use until the 1500's (even later in some countries)

monthnum = month number (1=Jan, 2=Feb, etc)

dayofmonth = day of the month (i.e., 1 through 31)

OUTPUTS

daywknum = day-of-the-week number (0 = Sunday, 1 = Monday, ..., 6 = Saturday)

NOTE: algorithm is only good for the years 1700 through 2299 of the Gregorian calendar.

References

http://www.whichday.com/

dops

Purpose

Compute Dilution's-Of-Precision.

Synopsis

```
dopvec = dops(svxyzmat,usrxyz)
```

Description

The direction cosine matrix is formed from the satellite locations given in `svxyzmat` and the user location given in `usrxyz`. The user location must be specified in three coordinates. `svxyzmat`(i,1:3) = x,y,z coordinates for satellite i (see `gensv`).

`dopvec(1)` = X-axis DOP
`dopvec(2)` = Y-axis DOP
`dopvec(3)` = Z-axis DOP (VDOP if `svxyzmat` is given in ENU relative to user)
`dopvec(4)` = X & Y DOP (HDOP if `svxyzmat` is given in ENU relative to user)
`dopvec(5)` = TDOP
`dopvec(6)` = PDOP
`dopvec(7)` = GDOP

`dopvec` consists of -1's if less than four satellite positions are provided.

Examples

Compute dilution's-of-precision for the GPS constellation at midnight relative to a user at the intersection of the equator and the prime meridian:

```
t = 0;
usrllh = [0*pi/180 0*pi/180 0];
usrxyz = llh2xyz(usrllh)
loadgps
svxyzmat = gensv(usrxyz,t);
dopvec = dops(svxyzmat,usrxyz)
```

In the example above, the DOPS are computed relative to the ECEF coordinate frame. If local-level-tangent values are desired (i.e., for HDOP and VDOP):

```
t = 0;
usrllh = [0*pi/180 0*pi/180 0];
usrxyz = llh2xyz(usrllh)
loadgps
svxyzmat = gensv(usrxyz,t);
[m,n]=size(svxyzmat);
for i = 1:m,
    svenumat(i,:) = ...
 xyz2enu(svxyzmat(i,:),usrxyz)';
end
dopvec = dops(svenumat,[0 0 0])
```

Algorithm

`hmat` is called to form the direction cosine matrix which is then used to form the user state error covariance matrix. The DOPs are functions of the diagonal elements of the

covariance matrix. For example, the square root of the 1,1 element is XDOP and the square root of the 4,4 element is TDOP.

See Also

Tutorial section on dilution of precision

`hmat, gensv`

References

Understanding GPS: Principles and Applications, E. Kaplan, Editor, Artech House Publishers, Boston, 1996.

enu2xyz

Purpose

Convert from rectangular local-level-tangent ('East'-'North'-Up) coordinates to WGS-84 ECEF Cartesian coordinates.

Synopsis

```
xyz = enu2xyz(enu,orgxyz)
```

Description

`enu(1)` = 'East'-coordinate relative to local origin (meters)
`enu(2)` = 'North'-coordinate relative to local origin (meters)
`enu(3)` = Up-coordinate relative to local origin (meters)

`orgxyz(1)` = ECEF x-coordinate of local origin in meters
`orgxyz(2)` = ECEF y-coordinate of local origin in meters
`orgxyz(3)` = ECEF z-coordinate of local origin in meters

`xyz(1,1)` = ECEF x-coordinate in meters
`xyz(2,1)` = ECEF y-coordinate in meters
`xyz(3,1)` = ECEF z-coordinate in meters

Note that ENU coordinates are truly from a local-level-tangent Cartesian coordinate system. In this local system, the x-axis points east from the local origin and the y-axis

points north, et cetera. Since 'east' actually follows a line of latitude (which is actually a circle), the local Cartesian coordinates approximately follow the geodetic directions for low and mid-latitudes only. When operating in the Earth's polar regions, the direction 'east' is not well approximated by a straight line.

Examples

Define a receiver location in geodetic coordinates (latitude: 39 degrees, longitude: -82 degrees, height: 150 meters). Now define the location of a second receiver to be 1 kilometer 'east' and 2 kilometers 'north' of the first receiver. Compute the ECEF coordinates of the second receiver:

```
rx1llh = [39*pi/180 -82*pi/180 150];
rx1xyz = llh2xyz(rx1llh)
rx2enu = [1000 2000 0];
rx2xyz = enu2xyz(rx2enu,rx1xyz)
```

Algorithm

A call to **xyz2llh** provides the geodetic coordinates of the local origin. This allows for the coordinate system rotation and translation required to convert from ENU to ECEF.

See Also

xyz2enu, xyz2llh, llh2xyz

References

Alfred Leick, GPS Satellite Surveying, 2nd ed., Wiley-Interscience, John Wiley & Sons, New York, 1995.

erotcorr

Purpose

Apply earth rotation correction to satellite position coordinates.

Synopsis

```
svxyzr = erotcorr(svxyz,pr)
```

Description

svxyz = satellite position at time-of-transmission expressed in the ECEF cartesian coordinate frame at the time of transmission

pr = measured pseudorange for the given satellite corrected for satellite clock offset (meters)

svxyzr = satellite position at time-of-transmission expressed in the ECEF Cartesian coordinate frame at time-of-reception (returned as a column vector)

genrng

Purpose

Generate vector of 'measured' pseudoranges and accumulated delta ranges (i.e., integrated Doppler, a.k.a. carrier-phase) to all visible satellites.

Synopsis

```
[prvec,adrvec] = genrng(rxid,usrxyz,svmat,...
                 svidvec,time,esf,samat,mpmat,...
                 freqc,TECmax,TECmin)
```

Description

`rxid` = receiver identification number (must be a positive integer)
For single receiver (stand-alone) simulations, `rxid` can be set to 1.

`usrxyz(1:3)` = true user position in ECEF Cartesian coordinates.

`svmat(i,1:3)` = position of satellite i in ECEF Cartesian coordinates.

`svid` = vector of identification numbers for the visible satellites corresponding to svmat

`time` = time of week in seconds

`esf` = error flags and scaling. If set to zero, error-free pseudoranges are computed. If

`esf` is any other scalar, all error sources are added (with default scaling) to the true ranges to produce 'measured' pseudoranges. If `esf` is a vector, specific error sources can be turned on or off:

`esf(1)`: Thermal Noise scaling factor
`esf(2)`: Tropospheric error scaling factor
`esf(3)`: SA scaling factor
`esf(4)`: Multipath error scaling factor
`esf(5)`: Ionospheric error scaling factor

Example: `esf`=[1 0 0 0 0] would cause only thermal noise to be added

The scaling factors have a valid range between 0 and 1 and the default value is 1. For thermal noise, the default one sigma ranging error is 1 meter for pseudorange and 1 centimeter for integrated Doppler. For tropospheric delay, the default is determined by the function **`tropgen`** and results in (roughly) a delay of 3 meters for a satellite at zenith and increases to a value of 25 meters for a satellite at 5 degrees elevation. For selective availability, the ranging error level is set by the routine which created **`samat`**. For multipath, the ranging error level is set by the routine which created **`mpmat`**. Ionospheric delay is set by the function **`ionogen`**.

`samat` = Optional matrix of Selective Availability error. `samat(t,i)` is the SA error for satellite i at t seconds after the current GPS hour. See **`sagen`**.

`mpmat` = Optional matrix of multipath error. `mpmat(t,i)` is the zero-elevation angle equivalent pseudorange multipath error for satellite i at t seconds after the GPS hour. The zero-angle error is scaled by the cosine of the true satellite elevation angle before it is applied to the range measurement. The carrier-phase multipath error is given by multiplying the pseudorange multipath error by (lambda*0.005). Since lambda (the carrier wavelength in meters) is approximately 0.2 (depending upon the carrier frequency), the carrier-phase multipath error is approximately a factor of 1000 less than the pseudorange multipath error. This is typical with the pseudorange multipath error being on the meter level and the carrier-phase multipath error being on the millimeter-

level. For DGPS applications, a different mpmat should be used for each receiver. Note also that a different mpmat should be used for each simulated code (i.e., C/A and P) and each carrier frequency (i.e., the multipath on L1 is not the same as the multipath on L2).

freqc = optional GPS carrier frequency in MHz; default is 1575.42; for L2 use 1227.6; the third civil frequency is planned for 1176.45; if Glonass is being simulated, user should input the corresponding GPS frequency. For example, input 1227.6 if you are trying to simulate Glonass L2. Note that Glonass satellites are identified by satellite id numbers which are 50 or above. Although any carrier frequency for GPS can be modeled; simulation of Glonass is limited to its L1 or L2 bands.

TECmax = optional argument equal to maximum daytime value of total election content (default is 1.6*1e18).

TECmin = optional argument equal to minimum daytime value of total election content (default is 4*1e17).

prvec = vector of 'measured' pseudoranges for satellites specified in svmat.

adrvec = vector of 'measured' accumulated delta ranges for satellites specified in svmat.

If the input parameters freqc, TECmax and TECmin are to be used, the matrices samat and mpmat must be input as well. These matrices may be set equal to the empty set, [], if the user does not want these errors to be included.

The scaling factors normally have a valid range between 0 and 1 and the default value is 1. For thermal noise, the default one sigma ranging error is 1 meter for pseudorange and (0.05*lambda) meters for the integrated Doppler (where lambda is the carrier wavelength in meters). For tropospheric delay, the default is determined by the function tropgen and results in (roughly) a delay of 3 meters for a satellite at zenith and increases to a

value of 25 meters for a satellite at 5 degrees elevation.

For selective availability, the ranging error level is set by the function `sagen` which is used to create `samat`. For multipath, the ranging error level is set by the function `mpgen` which is used to create `mpmat`. Ionospheric delay is set by the function `ionogen`.

The following should be noted regarding simulation of P-code, dual-frequency, third civil frequency, et cetera. The primary differences between C/A-code pseudoranges measured on L1 and C/A-code pseudoranges on, say, the third civil frequency are ionospheric delay and multipath error. In addition, the noise and multipath on the respective carrier-phase measurements will be different.

The function `ionogen` handles the differences in ionospheric delay as a function of carrier frequency. Note also that the ionospheric delay on the C/A-code on L1 is the same as the ionospheric delay on the P-code on L1. However, the P-code (on L1) noise is significantly less than the noise on the C/A-code (on L1). In order to simulate both C/A and P-code measurements, the user must call `genrng` once for each code.

When simulating C/A-code, the noise scaling flag, esf(1), might be set equal to 1. When calling `genrng` again to simulate simultaneous P-code measurements, `esf(1)` should then be set equal to 0.32 to account for the 1/sqrt(10) noise reduction in P-code over the C/A-code. If simulating both P-code on L1 and P-code on L2, `genrng` must be called separately for each to ensure that the noise is independent for each one and to account for the different carrier frequencies. The situation is the same for multiple civilian frequencies.

Regarding multipath, consider the typical case where the reflecting obstacles are relatively close to the receiver. The pseudorange multipath error level is independent of whether P-code or C/A-code is used. In addition, the multipath level for the P-code on L1 is the same as the multipath level for the P-code on L2. At any instant in time, however, the two codes will not have equal multipath error. Thus, `mpgen` must be used

separately to generate the multipath error for each code.

As is implied by the input parameter set, multipath and selective availability are created 'in batch.' This is done to speed runtime and, in the case of selective availability, is required in order to achieve the proper correlation properties (vital for DGPS applications). **genrng** only recognizes the first hour (3600 seconds) of **mpmat** and **samat**. Simulations which surpass an hour will result in error discontinuities resulting from the recycling of **mpmat** and **samat**. If the user desires to run longer simulations, this can be accomplished by setting multipath and SA to zero in **genrng** and then adding it to the measurements externally (see **mpgen** and **sagen**).

In order to accommodate simultaneous GPS and Glonass operation, **genrng** only adds SA if the satellite identification number is less than 25. Note that Glonass signals do not have SA. Note also that **loadgg** results in satellite orbital parameters loaded for GPS (SV ID's 1 - 24) and for Glonass (SV ID's 51 - 74).

Note that `genrng` does not emulate the effects of Earth rotation. The satellite and user positions at a particular time are used to calculate range.

Examples

Specify a receiver at North 39 degrees, West 82 degrees and height 0 meters. Compute pseudoranges and integrated Doppler values (including all error sources) for all GPS satellites in view above 15 degrees at 2000 seconds past midnight:

```
samat=sagen(24,3600);
mpmat=mpgen(24,3600);
rxllh = [39*pi/180 -82*pi/180 0];
rxxyz = llh2xyz(rxllh);
t = 2000;
```

```
loadgps
[svmat,svid] = gensv(rxxyz,t,15);
[pr,adr] = genrng(1,refxyz,svmat,...
                  svid,t,[1 1 1 1 1],samat,mpmat);
```

Algorithm

`genrng` calculates the true range from each satellite to the user at the requested time (Earth rotation effects are ignored). 'Measured' pseudoranges are computed by adding errors to the true ranges.

genrng interpolates between multipath and selective availability data points if the requested time is not an even number of seconds.

Ionospheric delay is added to the pseudoranges but subtracted from the integrated Dopplers to emulate the dispersive nature of the medium.

Carrier cycle ambiguities are emulated on each integrated Doppler measurement by adding the following:

10*rxid*rxid*svid*lambda

where svid is the identification number of the particular satellite and lambda is the carrier-wavelength (in meters). Note that for Glonass (svid's greater than 50), lambda is different for each satellite (frequency division multiple access). The pairing of frequencies on Glonass satellites is not currently emulated by the TOOLBOX.

See Also

`loadgps, loadglo, loadgg, loadyuma, gensv, gensvalm, sagen, mpgen, ionogen, tropgen`

gensv

Purpose

Generate matrix of positions of satellites (space vehicles: sv) which are visible at a given location.

Synopsis

```
svxyzmat = gensv(usrxyz,time,maskang,orgxyz)
     or
[svxyzmat,svid] = ...
           gensv(usrxyz,time,maskang,orgxyz)
```

Description

Generate matrix of positions of satellites which are visible to the user (or any general location) at location `usrxyz`. If `orgxyz` is not specified, then satellite positions are computed in ECEF coordinates. If `orgxyz` is specified, it is taken to be the origin of a locally-level, East-North-UP (ENU) coordinate system and satellite positions are given accordingly.

`usrxyz`(1:3) = user position in Cartesian ECEF coordinates if `orgxyz` is not provided. If `orgxyz` is provided, `usrxyz` is given in Cartesian ENU coordinates relative to `orgxyz`.

`time` = time of week in seconds

maskang = satellite visibility mask angle (in degrees). This argument is optional and has a default value of 5 degrees

orgxyz(1:3) = optional argument. Cartesian ECEF coordinates of locally-level ENU coordinate origin

svxyzmat = matrix of visible satellite positions in Cartesian coordinates.
svxyzmat(i,1:3) = x,y,z coordinates for satellite i

svid = vector of identification numbers for the visible satellites corresponding to svxyzmat

NOTE: gensv expects satellite orbital parameters to be loaded and maintained as global variables. This is accomplished with loadgps, loadglo or loadgg.

Examples

Specify a receiver at North 39 degrees, West 82 degrees and height 0 meters. Compute positions of all GPS satellites in view above 15 degrees at 2000 seconds past midnight:

```
rxllh = [39*pi/180 -82*pi/180 0];
rxxyz = llh2xyz(rxllh);
t = 2000;
loadgps
[svmat,svid] = gensv(rxxyz,t,15);
```

Algorithm

gensv calculates the position of all satellites (as given by the parameters loaded by

`loadgps`, `loadglo` or `loadgg`). Satellite elevation angle relative to `usrxyz` is computed. Those positions of the satellites above the specified mask angle are stored in `svxyzmat` and their identification numbers are stored in `svid`.

See Also

`genrng`, `loadgps`, `loadglo`, `loadgg`, `dops`

gensvalm

Purpose

Broadcast almanac version of `gensv`. Generate matrix of positions of satellites (space vehicles: sv) which are visible at a given location.

Synopsis

```
svxyzmat = gensvalm(usrxyz,time,maskang,orgxyz)
          or
[svxyzmat,svid] = ...
        gensvalm(usrxyz,time,maskang,orgxyz)
```

Description

Generate matrix of positions of satellites which are visible to the user (or any general location) at location `usrxyz`. If `orgxyz` is not specified, then satellite positions are computed in ECEF coordinates. If `orgxyz` is specified, it is taken to be the origin of a locally-level, East-North-UP (ENU) coordinate system and satellite positions are given accordingly.

`usrxyz(1:3)` = user position in Cartesian ECEF coordinates if `orgxyz` is not provided. If `orgxyz` is provided, `usrxyz` is given in Cartesian ENU coordinates relative to `orgxyz`.

`time` = time of week in seconds

`maskang` = satellite visibility mask angle (in degrees). This argument is optional and has a default value of 5 degrees

`orgxyz(1:3)` = optional argument. Cartesian ECEF coordinates of locally-level ENU coordinate origin

`svxyzmat` = matrix of visible satellite positions in Cartesian coordinates.
`svxyzmat(i,1:3)` = x,y,z coordinates for satellite i

`svid` = vector of identification numbers for the visible satellites corresponding to `svxyzmat`

NOTE: `gensvalm` expects satellite orbital parameters to be loaded and maintained as global variables. This is accomplished with `loadyuma`.

Examples

Specify a receiver at North 39 degrees, West 82 degrees and height 0 meters. Compute positions of all GPS satellites in view above 15 degrees at 2000 seconds past midnight:

```
rxllh = [39*pi/180 -82*pi/180 0];
rxxyz = llh2xyz(rxllh);
t = 2000;
loadyuma('yuma1.txt')
[svmat,svid] = gensvalm(rxxyz,t,15);
```

Algorithm

`gensvalm` calculates the position of all satellites (as given by the parameters loaded by

`loadyuma`). Satellite elevation angle relative to `usrxyz` is computed. Those positions of the satellites above the specified mask angle are stored in `svxyzmat` and their identification numbers are stored in `svid`.

See Also

`genrng, loadyuma, dops`

gpscor

Purpose

Limited circular cross-correlation function for C/A-codes.

Synopsis

```
[lag,r] = gpscor(x1,x2,numlag)
```

Description

Unlike standard cross-correlation functions (see the MATLAB function `xcorr`, for example), `gpscor` takes into account the fact that the C/A-codes (and the maximum-length sequences used to build C/A-codes) are periodic sequences. `gpscor`, therefore, performs a circular correlation function on `x1` and `x2` and yields `r` for the lags output in `lag`.

Correlation is a computationally intensive process and often times the full correlation function is not needed. In order to facilitate this, `gpscor` performs cross-correlation on `x1` and `x2` from lag 0 up through `numlag`.

Note that, because `gpscor` was designed to work with C/A-codes, it expects `x1` and `x2` to be 1023 length, periodic sequences.

Examples

Generate a C/A-code. Compute and plot the first twenty-five lags of the autocorrelation function of the C/A-code and the two maximum-length sequences used to make the code:

```
[ca,g1,g2] = cacode(5);
[lagg1,rg1] = gpscor(g1,g1,25);
plot(lagg1,rg1)
[lagg2,rg2] = gpscor(g2,g2,25);
plot(lagg2,rg2)
[lagca,rca] = gpscor(ca,ca,25);
plot(lagca,rca)
```

Now generate two different C/A-codes, compute the first twenty-five lags of their cross-correlation function and plot:

```
[ca5,g1,g2] = cacode(5);
[ca6,g1,g2] = cacode(6);
[lagca56,rca56] = gpscor(ca5,ca6,25);
plot(lagca56,rca56)
axis([0 25 -200 1200])
```

See Also

`cacode`

Tutorial section on pseudorandom noise codes

hatch

Purpose

Perform carrier-smoothing of the pseudorange measurements via the Hatch filter.

Synopsis

```
[prsm,prmat,adrmat]=...
        hatch(pr,adr,svid,smint,prmat,adrmat)
```

Description

A Hatch filter processes pseudorange and integrated Doppler measurements to produce a smoothed pseudorange observable.

`pr` = vector of measured pseudoranges for the current epoch

`adr` = vector of integrated Doppler measurements for the current epoch

`svid` = vector of satellite id's corresponding to the measurements in `pr` and `adr`

`smint` = smoothing interval in seconds (must be an integer)

`prmat` = matrix of previous pseudorange values for all satellites.

`adrmat` = matrix of previous integrated Doppler values for all satellites

When `hatch` is first called, `prmat` and `adrmat` are set to []. These matrices are updated by `hatch` and output to the user who must input them at the next epoch

`prsm` = vector of carrier-smoothed pseudoranges

`prmat` = updated matrix of previous pseudoranges

`adrmat` = updated matrix of previous integrated Doppler

Examples

Specify a static user and generate satellite positions and pseudoranges. Generate pseudoranges which only have noise (and no other error sources). Carrier-smooth the pseudoranges and compute position with the raw and smoothed measurements. Plot the error in both cases:

```
usrllh = [39*pi/180 -82*pi/180 150];
usrxyz = llh2xyz(usrllh);
loadgps
prmat=[]; adrmat=[]; i=0;
for t = 1:200,
    i=i+1;
    fprintf(1,' time = %i   Stop time = 200\n',t)
    [svxyzmat,svid] = gensv(usrxyz,t);
    [prvec,adrvec] = genrng(1,usrxyz,svxyzmat,...
                            svid,t,[1 0 0 0 0]);
    [prsm,prmat,adrmat] = hatch(prvec,adrvec,...
                          svid,25,prmat,adrmat);
    estusr = olspos(prvec,svxyzmat);
    smestusr = olspos(prsm,svxyzmat);
    err(i,:) = estusr - [usrxyz 0];
    smerr(i,:) = smestusr - [usrxyz 0];
```

```
    time(i) = t;
end
plot(time,err(:,1),'r',time,smerr(:,1),'g')
```

Algorithm

The heart of carrier-smoothing lies in the estimation of the difference (ambiguity) between the pseudorange and integrated Doppler. The estimated difference is added to the integrated Doppler measurement to produce a low noise range observable. A moving average of `smint` seconds is used for the estimation. Note that the size of the smoothing interval is limited by the code/carrier divergence caused by the ionosphere.

See Also

`compkal, loadgps, loadglo, loadgg, gensv, genrng`

Reference

"The Synergism of GPS Code and Carrier Measurements," by Ron Hatch, Proceedings of the Third International Geodetic Symposium on Satellite Doppler Positioning, Las Cruces, NM, February 1982.

hmat

Purpose

Compute direction cosine matrix.

Synopsis

```
h = hmat(svmat,usrpos)
```

Description

The direction cosine matrix corresponding to a user at `usrpos` and satellites located at `svmat` is computed.

`svmat` = matrix of satellite positions in user defined Cartesian coordinates. `svmat(i,1:3)` = x,y,z coordinates for satellite i

`usrpos` = estimated user position in user defined Cartesian coordinates

`h` = direction cosine matrix for GPS positioning

Reference

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

ionocorr

Purpose

Compute ionospheric correction from broadcast model parameters.

Synopsis

```
ionodel = ionocorr(systime,svxyz,usrxyz)
```

Description

systime = time at which iono delay is to be
computed; note this is expressed as
system time (i.e., GPS time of week in seconds)
svxyz = satellite position expressed in ECEF cartesian coordinates
usrxyz = user position expressed in ECEF cartesian coordinates

NOTE: This function assumes the iono broadcast model
parameters (ALPHA and BETA vectors) have been loaded
into global memory (such as by the use of `loadrinexn`)

Reference

ICD-GPS-200, NAVSTAR GPS Space Segment/Navigation User Interfaces (Public Release Version), ARINC Research Corporation, 11770 Warner Ave., Suite 210, Foutain Valley, CA 92708, July 3, 1991.

ionogen

Purpose

Generate ionospheric delay in meters.

Synopsis

```
delta = ...
ionogen(usrxyz,svxyz,time,freqc,TECmax,TECmin)
```

Description

`delta` is the ionospheric delay associated with a signal propagating from a navigation satellite (located at `svxyz`) to a user (located at `usrxyz`) at a particular time.

`usrxyz`(1:3) = true user position in ECEF Cartesian coordinates.

`svxyz`(1:3) = position of satellite in ECEF Cartesian coordinates.

`time` = GPS time of week in seconds.

`freqc` = optional argument equal to carrier frequency in MHz (default is 1575.42).

`TECmax` = optional argument equal to maximum daytime value of total election content (default is 1.6*1e18)

`TECmin` = optional argument equal to minimum daytime value of total election content (default is 4*1e17)

`delta` = ionospheric delay in meters.

Algorithm

The model is composed of the traditional raised half-cosine profile for zenith delay [1]. This delay is scaled by the FAA Wide Area Augmentation System (WAAS) obliquity factor described in [2] to account for the actual elevation angle of the satellite.

Note that `ionogen` can handle any arbitrary satellite navigation carrier frequency (i.e., L1, L2, L5, etc) but it does not emulate scintillation.

References

[1] Global Positioning System - Theory and Applications, Vol. I, Parkinson, B. and J. Spilker, Jr., editors, American Institute of Aeronautics and Astronautics, Washington, D.C., 1996.

[2] Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

llh2xyz

Purpose

Convert from WGS-84 geodetic coordinates (latitude, longitude, height) to WGS-84 ECEF Cartesian coordinates.

Synopsis

```
xyz = llh2xyz(llh)
```

Description

`xyz(1)` = ECEF x-coordinate in meters
`xyz(2)` = ECEF y-coordinate in meters
`xyz(3)` = ECEF z-coordinate in meters

`llh(1)` = latitude in radians
`llh(2)` = longitude in radians
`llh(3)` = height above ellipsoid in meters

Examples

Define a receiver location in geodetic coordinates (latitude: 39 degrees, longitude: -82 degrees, height: 150 meters) and compute its ECEF Cartesian coordinates:

```
usrllh = [39*pi/180 -82*pi/180 150];
usrxyz = llh2xyz(usrllh)
```

See Also

`enu2xyz, xyz2enu, xyz2llh`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

loadgalileo, loadgeo, loadglo, loadgps

Purpose

Load Galileo, Geostationary, Glonass or GPS orbital parameters into global memory (for simulation purposes only).

Synopsis

```
loadgalileo
```

or

```
loadgeo
```

or

```
loadglo
```

or

```
loadgps
```

Description

The following Kepler parameters for ideal circular orbits are loaded and maintained as global variables:

SVIDV = vector of satellite identification numbers

MV = vector of Mean anomalies for the satellites in SVIDV (at reference time) in degrees

RV = vector orbit radii for the satellites in SVIDV(semi-major axis) in meters

TOEV = vector of reference times for the Kepler parameters for the satellites in SVIDV (time of ephemeris) in seconds

OMGV = vector of longitudes of the ascending nodes for the satellites in SVIDV (at weekly epoch) in degrees

INCLV = vector of inclination angles of orbital planes of the satellites in SVIDV (in degrees)

`loadgps` loads GPS parameters only. `loadgalileo` loads Galileo parameters only, et cetera. Each type of satellite is assigned a unique range of satellite identification numbers (GPS: 1-40; Glonass: 51-74; Geo's: 101-150; Galileo: 201-230)

Examples

See the tutorial section for usage of these functions. If it is desired to simulate more than one system simultaneously, merely call each load function in sequence. If you are interested in simulating both GPS and Galileo, simply enter the following:

```
loadgps
loadgalileo
```

See Also

gensv, genrng

loadrinexn

Purpose

Satellite ephemeris (Navigation message) data in RINEX2 format. Load the data from a user-specified ASCII text file in RINEX2 format. The data are maintained as global variables.

Synopsis

```
loadrinexn(filename)
```

where: `filename` is the name of the ASCII text file containing the RINEX2-formatted Navigation message data also known as the satellite ephemeris (and related data)

NOTE: make sure to put the name in single quotation marks. For example:

loadrinexn('stkr2581.02n')

loadrinexob

Purpose

Load raw GPS measurement (i.e., observations) data from a RINEX2 formatted file. NOTE: This routine only processes RINEX files with GPS measurements (no Glonass, GEOs, et cetera).

Synopsis

```
loadrinexob(filename,decimate_factor)
```

where: `filename` is the name of the ASCII text file containing the RINEX2-formatted observation data

NOTE: make sure to put the name in single quotation marks. For example:

loadrinexn('stkr2581.02o')

decimate_factor = an integer which indicates the desired decimation of the data. If it is equal to '1', then every data point is read in and stored. If it is equal to '2', then every second data point is stored. If '3', then every third data point is stored, et cetera. This is an optional parameter, the default is '1'.

ALL OUTPUTS ARE GIVEN AS GLOBAL VARIABLES

SVID_MAT = matrix of satellite measurement availability;
if measurements have been made for a given satellite
(i.e., prn #N) at the k-th epoch of time, then:
SVID_MAT(N,k) = 1;
TOWSEC = vector of times-of-reception given in GPS time-of-week
in seconds
PHASE1,PHASE2 = carrier-phase measurements made on L1 and L2
in units of carrier cycles (wavelengths)
C1 = C/A-code pseudorange measurement made on L1 in units of
meters
P1,P2 = P(Y)-code pseudorange measurements made on L1 and L2
in units of meters
D1,D2 = Doppler measurements on L1 and L2 in units of Hz
S1,S2 = Raw signal strength measurements

MARKER_XYZ = approximate position of geodetic marker (WGS-84
cartesian coordinates)
ANTDELTA: ANTDELTA(1) = height of bottom surface of antenna
above the marker (in meters)
ANTDELTA(2:3) = east and north eccentricities of antenna
center relative to the marker (in meters)
OBSINT = observation interval in seconds
CLOCKOFFSET = vector of receiver clock offsets as determined by
the receiver itself (units of seconds)

ASSUMPTIONS: 1) receiver tracks 12 or fewer satellites
2) wavelength factors are equal to 1, namely
full cycle ambiguities
3) number of different types of observations is 9 or less
4) only GPS observations! no Glonass, GEO's, etc

NOTE: Since the RINEX time-of-clock gives the year in a two digit
format, the conversion to time-of-week in this routine is only valid

for the years 1971 through 2070

REFERENCE: Gurtner, W., RINEX user manual, Version 2.10,
http://www.ngs.noaa.gov/CORS/rinex210.txt

loadyuma

Purpose

Load GPS broadcast almanac orbital parameters into global memory.

Synopsis

```
loadyuma(filename)
```

where: `filename` is the name of the ASCII text file containing the YUMA-formatted almanac data (NOTE: make sure to put the name in single quotation marks. For example:

loadyuma('yuma1.txt')

Description

The following broadcast almanac orbital parameters are loaded and maintained as global variables:

SVIDV = vector of satellite identification numbers

MV = vector of Mean anomalies for the satellites in SVIDV (at reference time) in degrees

RV = vector orbit radii for the satellites in SVIDV(semi-major axis) in meters

TOEV = vector of reference times for the Kepler parameters

for the satellites in SVIDV (time of ephemeris) in seconds

OMGV = vector of longitudes of the ascending nodes for the satellites in SVIDV (at weekly epoch) in degrees

INCLV = vector of inclination angles of orbital planes of the satellites in SVIDV (in degrees)

HEALTHV = vector of health words for the satellites in SVID (000 = satellite is good)

ECCENV = vector of eccentricities for the satellites in SVID (dimensionless)

OMGDOTV = vector of Rates of Right Ascension for the satellites in SVID (radians/sec)

ARGPERIV = vector of Arguments of Perigee for the satellites in SVID (radians)

AF0V = vector of satellite clock offsets for the satellites in SVID (seconds)

AF1V = vector of satellite clock drift coefficients for the satellites in SVID (sec/sec)

WEEKV = vector of week numbers for the almanac data for the satellites in SVID

Obtaining YUMA-formatted Almanac Files

Yuma-formatted almanac files (along with a description of the format) may be obtained at the U.S. Coast Guard Navigation Center web site:

http://www.navcen.uscg.mil/gps/almanacs/default.htm

Note that `loadyuma` is able to decode both the old and the new Yuma formats.

Examples

Load GPS satellite parameters and calculate all satellites in view for a given receiver:

```
usrllh = [39*pi/180 -82*pi/180 150];
usrxyz = llh2xyz(usrllh);
loadyuma('yuma1.txt')
t = 0;
[svxyzmat,svid] = gensvalm(usrxyz,t);
skyplot(svxyzmat,svid,usrxyz)
```

See Also

`gensvalm, genrng`

mpgen

Purpose

Create pseudorange multipath error.

Synopsis

```
delta = mpgen(numsat,duration,model,mpseed)
```

Description

`delta` is a matrix of pseudorange multipath errors corresponding to the specified number of satellites (`numsat`: number of columns in `delta`) and the total number of seconds (`duration`: number of rows in `delta`).

`model` specifies one of two multipath models. `model` = 1 produces pseudorange multipath with a standard deviation of approximately 1.6 meters and a time constant of approximately 2 minutes. `model` = 2 produces multipath with a standard deviation of approximately 0.4 meters and a time constant of approximately 15 seconds.

`mpseed` allows the user to set the random number generator seed (usually done for purposes of repeatability). See the MATLAB functions `rand` and `randn` for more information on MATLAB's random number

generator.

Examples

Generate multipath for 24 satellites for a one hour period using the long time constant model. Plot out the multipath error for one of the satellites:

```
delta = mpgen(24,3600,1);
plot(delta(:,1))
```

Algorithm

`mpgen` creates multipath by passing white noise through a first-order Butterworth low-pass filter. Note that in many applications multipath is well-modeled by this kind of simple colored noise.

See Also

`genrng, sagen, ionogen, tropgen`

nmatgen

Purpose

Generate matrix of candidate ambiguity sets for the case of 5 satellites (4 double-differences) with uncertainty less than ±1 meter (±5 wavelengths).

Synopsis

```
Nmat = nmatgen
```

Description

`Nmat` = candidate ambiguity set matrix. Rows in `Nmat` range from [-5 -5 -5 -5] up to [5 5 5 5].

Since this routine takes awhile to run, this has been done for you and the result may be loaded directly from `nmat.mat`.

olspos

Purpose

Ordinary least squares solution for user state given satellite position matrix and pseudorange measurement vector.

Synopsis

```
estusr = olspos(prvec,svxyzmat,initpos,tol)
```

Description

`prvec` = vector of 'measured' pseudoranges for satellites specified in `svxyzmat`.

`svxyzmat(i,1:3)` = position of satellite i in user defined Cartesian coordinates.

`initpos` = optional argument. Initial 'estimate' of user state: three-dimensional position and clock offset (in user defined coordinates). Used to speed up iterative solution. Initial clock offset is optional with default value = 0.

`tol` = optional argument. Tolerance value used to determine convergence of iterative solution. Default value = 1e-3

`estusr(1:3)` = estimated user x, y, and z coordinates. `estusr(4)` = estimated user clock offset. Note: all four elements of estusr are in the same units as those used in `prvec`.

Examples

See tutorial section on stand-alone positioning.

Algorithm

The initial guess for the user state is used to linearize the pseudorange observation. The goal is to solve for the offset between the initial guess and the actual user state. This procedure iterates until the norm of the offset vector is less than `tol`. To prevent infinite looping, a maximum of ten iterations are allowed. Note that even with the initial guess set to all zeros, the solution for points on or near the earth will converge in less than ten steps.

See Also

`genrng, gensv`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

olsposgg

Purpose

Ordinary least squares solution for user state given GPS and Galileo satellite position matrix and corresponding pseudorange measurement vector. User state includes three-dimensional position, receiver clock offset and GPS/Galileo time offset. This routine also works for GPS and Glonass, or Glonass and Galileo, et cetera.

Synopsis

```
estusr = olsposgg(prvec,svxyzmat,svid,initpos,tol)
```

Description

`prvec` = vector of 'measured' pseudoranges for satellites specified in `svxyzmat`.

`svxyzmat(i,1:3)` = position of satellite i in user defined Cartesian coordinates.

`svid` = vector of satellite identification numbers corresponding to svxyzmat.

`initpos` = optional argument. Initial 'estimate' of user state: three-dimensional position, receiver clock offset and GPS/Glonass system time offset (in user defined coordinates). Used to speed up iterative

solution. Initial clock offset and time offset are optional with default values = 0.

`tol` = optional argument. Tolerance value used to determine convergence of iterative solution. Default value = 1e-3

`estusr(1:3)` = estimated user x, y, and z coordinates. `estusr(4)` = estimated user clock offset. `estusr(5)` = estimated GPS/Glonass system time offset. Note: all five elements of estusr are in the same units as those used in `prvec`.

Examples

See tutorial section.

Algorithm

The initial guess for the user state is used to linearize the pseudorange observation. The goal is to solve for the offset between the initial guess and the actual user state. This procedure iterates until the norm of the offset vector is less than tol. To prevent infinite looping, a maximum of ten iterations are allowed. Note that even with the initial guess set to all zeros, the solution for points on or near the earth will converge in less than ten steps.

Since there are five variables to solve for (3-d position, receiver offset, GPS/Galileo time offset), a minimum of five measurements are required with at least one from each system.

See Also

`olspos, genrng, gensv`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

parityvec

Purpose

Compute the parity vector associated with the overdetermined user position solution. This is a new name for the function RAIM in previous versions of the toolbox.

Synopsis

```
parvec = parityvec(prvec,svxyzmat,initpos,tol)
```

Description

`prvec` = vector of 'measured' pseudoranges for satellites specified in svxyzmat

`svxyzmat(i,1:3)` = position of satellite i in user defined Cartesian coordinates.

`initpos` = optional argument. Initial 'estimate' of user state: three-dimensional position and clock offset (in user defined coordinates). Used to speed up iterative solution. Initial clock offset is optional with default value = 0.

`tol` = optional argument. Tolerance value used to determine convergence of iterative solution. Default value = 1e-3.

`parvec` = parity vector with n-4 elements where n is the total number of pseudorange measurements.

Examples

See tutorial section.

Algorithm

The QR decomposition (see the MATLAB function qr) is used as an aid in separating the solution space (which yields the ordinary least-squares state solution) from the error space. The error space yields a measure of the inconsistency of the redundant measurements. It is also called the parity space. The measure of inconsistency is given by the norm of the parity vector.

See Also

`olspos, genrng, gensv`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

pathgen

Purpose

Vehicle path (trajectory) generator. Path consists of an arbitrary number of user-defined straight segments joined by constant-radius turns.

Synopsis

```
path = pathgen(initpos,initvel,segparam,deltat)
```

Description

`initpos` = initial position of vehicle (ENU Cartesian coordinates). Units are meters.

`initvel` = initial velocity vector (East and North Cartesian coordinates only). Units are meters per second. Note that the magnitude of the linear velocity is constant throughout the trajectory.

`segparam` = segment and turn parameters:

- `segparam(i,1)` = duration (in seconds) of i-th straight segment
- `segparam(i,2)` = direction and amount of i-th turn (set to zero if no turn is desired) (degrees). Positive is a left turn.
- `segparam(i,3)` = centripetal acceleration of i-th turn (in m/s^2)

`deltat` = time increment in seconds

`path` = ENU path generated

Note that `pathgen` only supports horizontal paths. If a height is specified in `initpos`, it will be held constant throughout the generation of `path`.

Examples

Generate a vehicle path which starts at the local origin. Initial velocity is 50 meters per second in the north direction. After one minute, make a 90 degree right-hand 1-g turn, followed by a straight segment for one minute followed by a 45 degree right-hand 1-g turn followed by a one minute straight segment:

```
initpos = [0 0 0];
initvel = [0 50];
segparam = [60 -90 9.8; 60 -45 9.8; 60 0 0];
deltat = 1;
path = pathgen(initpos,initvel,segparam,deltat);
plot(path(:,1),path(:,2))
```

Algorithm

Newton's laws applied to circular motion state that the turn radius is given by the square of the tangential velocity divided by the centripetal acceleration. Angular velocity is then given by the tangential velocity divided by the turn radius.
At the end of a turn, the prior velocity vector is rotated for generation of the next straight segment.

prncode

Purpose

GPS C/A pseudorandom noise code generator.

Synopsis

[ca,g1,g2] = prncode(prnnum)

Description

prnnum = GPS PRN signal number (must be an integer in the range 1:37)

ca = C/A-code sequence realized with +1's and -1's (digital 0's and 1's).
ca is the product of G1 and G2 (after G2 has been shifted)
g1, g2 = maximal length sequences with g2 shifted according to the specification in GPS-ICD-200.

raim

Purpose

Receiver Autonomous Integrity Monitoring. `raim` computes the parity vector associated with the overdetermined user position solution.

Synopsis

```
parvec = raim(prvec,svxyzmat,initpos,tol)
```

Description

`prvec` = vector of 'measured' pseudoranges for satellites specified in svxyzmat

`svxyzmat(i,1:3)` = position of satellite i in user defined Cartesian coordinates.

`initpos` = optional argument. Initial 'estimate' of user state: three-dimensional position and clock offset (in user defined coordinates). Used to speed up iterative solution. Initial clock offset is optional with default value = 0.

`tol` = optional argument. Tolerance value used to determine convergence of iterative solution. Default value = 1e-3.
`parvec` = parity vector with n-4 elements where n is the total number of

pseudorange measurements.

Examples

See tutorial section on RAIM.

Algorithm

The QR decomposition (see the MATLAB function qr) is used as an aid in separating the solution space (which yields the ordinary least-squares state solution) from the error space. The error space yields a measure of the inconsistency of the redundant measurements. It is also called the parity space. The measure of inconsistency is given by the norm of the parity vector.

See Also

`olspos, genrng, gensv`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

sagen

Purpose

Non-"Realtime" generation of Selective Availability error.

Synopsis

```
delta = sagen(numsat,duration,model,saseed)
```

Description

`numsat` = number of SA error traces (i.e., number of satellites for which SA is to be generated)

`duration` = duration of error traces in seconds (must be an integer). For model= 4 or 5, duration must not be less than twenty minutes (1200 seconds).

`model` = Optional model choice for generation of Selective Availability. Default is: model = 3

- 1 = second-order Gauss-Markov model
- 2 = second-order Gauss-Markov with velocity error
- 3 = Autoregressive model
- 4 = Rater analytic model
- 5 = Rater analytic model with velocity and acceleration errors

`saseed` = Optional seed for Gaussian random number generator. Default

setting is: `saseed = sum(100*clock).`

If `model` = 1, 3 or 4, delta is a matrix of Selective Availability ranging error. The columns in delta are the 'duration'-number of seconds of range error corresponding to the number of traces specified in saseed.

If `model` = 5, errors are generated as follows: For the satellite I,

`delta(:,3*i - 2)` = range error in meters;
`delta(:,3*i - 1)` = range-rate error in meters/second;
`delta(:,3*i)` = acceleration error in meters/second-squared.

For `model` = 2, delta is similar except acceleration errors are not generated.

NOTE: All error traces are generated at a rate of 1 Hz. It should also be noted that the SA errors generated by each model start at zero and can take several hundred seconds before reach nominal levels.

Examples

Use the autoregressive model to generate half an hour of SA error for one satellite and plot the results.

```
delta = sagen(1,1800,3,2345);
plot(delta)
```

See Also

`mpgen, ionogen, tropgen, genrng`

References

Global Positioning System - Theory and Applications, Volume I, B. Parkinson and J. Spilker, Jr., Senior Editors, Volume 163, Progress in Astronautics and Aeronautics, American Institute of Aeronautics and Astronautics, Washington, D.C., 1996.

satvis, satvis2, satvis3

Purpose

Generate satellite visibility plots for simulated constellations (satvis), for RINEX2-format ephemeris data (satvis2) and for Yuma-formatted almanac data (satvis3).

Synopsis

```
vismat =
satvis(usrxyz,tstart,tstop,tinc,sysflg,plotflg,mas
kang)

vismat =
satvis2(usrxyz,tstart,tstop,tinc,daynum,filename,p
lotflg,maskang)

vismat = satvis3(usrxyz,tstart,tstop,tinc,daynum,...
filename,plotflg,currweek,maskang)
```

Description

`usrxyz` = user position in Cartesian ECEF coordinates

`tstart,tstop,tinc` = start, stop and increment times for satellite visibility assessment (GPS time of day in seconds)

daynum = day number of the week (0=Sunday, 1=Monday, ..., 6=Saturday)

filename = name of the RINEX or Yuma file; note the files must be in RINEX2 or Yuma formats; the filename must be in quotes (e.g., 'stkr2581.o2n' or 'yuma19.txt')

`sysflg` = vector specifying which satellite systems are to be included: 1 for GPS, 2 for Glonass, 3 for GEOs and 4 for Galileo. For example, if you want both GPS and Galileo: sysflg = [1 4].

`plotflg` =
1 for total number of visible satellites versus time
2 for bar plot of visibility of each satellite versus time.

currweek = GPS week number for the day (daynum) and time (tstart, etc) specified

`maskang` = Visibility mask angle. Optional argument. Default is 5 degrees.

`vismat` = matrix of satellite visibility parameters.
If `plotflg` = 1, `vismat(:,1)` is the total number of visible satellites for the time points specified in `vismat(:,2)`.

If `plotflg` = 2, `vismat(:,25)` is the vector of time points at which visibility was evaluated. Columns 1 through 24 correspond to the 24 satellites in the constellation (either GPS or Glonass) and contain a zero if that satellite is not visible at the given time point.

Examples

See the Tutorial and What's New sections of this manual.

See Also

gensv, loadgps, loadglo, loadgg, skyplot

skyplot, skyplot2, skyplot3

Purpose

Generate satellite sky plots for simulated constellations (skyplot), for RINEX2-formatted ephemeris (skyplot2) or for Yuma-formatted almanac data (skyplot3).

Synopsis

```
skyplot(svxyzmat,svid,usrxyz,figflg,idflg)
```
or
skyplot2(filename,tow,usrxyz,maskangle,figflg,idflg)
or
skyplot3(filename,tow,usrxyz,currweek,maskangle,figflg,idflg)

Description

`svxyzmat` = matrix of satellite positions in Cartesian ECEF coordinates
`svxyzmat(i,1:3)` = x,y,z coordinates for satellite i

`svid` = vector of satellite identification numbers corresponding to the satellite locations given in `svxyzmat`.

`usrxyz` = user position in Cartesian ECEF coordinates

`svenumat` = matrix of satellite positions in East-North-Up coordinates relative to the user-determined skyplot origin (usually user location)

```
filename = name of the RINEX or YUMA file; note
the files must be either in RINEX or YUMA format;
the filename must be in quotes(e.g.,
'stkr2581.o2n' or 'yuma19.txt')

tow = GPS time of week in seconds

currweek = GPS week number for the time specified
(default is the week number specified for the
first satellite listed in the almanac file)
```

`figflg` = 1 if the MATLAB function `close` is to be executed before plotting; 0 if the `close` function is NOT to be executed before plotting. Default value is 1.

`idflg` = 1 if satellite identification numbers are desired on the plot, 0 otherwise. Default is 1.

See Also

`gensv, loadgps, loadglo, loadgg, satvis`

storerinnav

Purpose

Program to load a RINEX2 navigation (ephemeris) file and store the data in a MATLAB file format (MAT-file).

Synopsis

```
storerinnav(rinexfilename,matfilename)
```

Description

rinexfilename = Name of the ASCII text file containing the
RINEX2-formatted Navigation message data
matfilename = filename for the MAT-file which will be created
If matfilename='day101nav' then a MAT-file will
be created called: day101nav.mat

NOTE: make sure to put the names in single quotation marks (e.g., storerinnav('stkr2581.02n','day258nav')

storerinob

Purpose

Program to load a RINEX2 observation (measurement) file and store the data in a MATLAB file format (MAT-file).

Synopsis

```
storerinob(rinexfilename,matfilename)
```

Description

rinexfilename = Name of the ASCII text file containing the
RINEX2-formatted observation (measurement) data
matfilename = filename for the MAT-file which will be created
If matfilename='day258ob' then a MAT-file will
be created called: day258ob.mat

NOTE: make sure to put the names in single quotation marks (e.g., storerinob('stkr2581.02o','day258ob')

svclkcorr

Purpose

Compute satellite clock correction based on the parameters broadcast in the navigation message.

Synopsis

```
[clkcorr_m,grpdel] = svclkcorr(svid,towsec,Ek)
```

Description

svid = satellite identification number

towsec = GPS time of week in seconds

Ek = eccentric anomaly of the satellite in radians

clkcorr_m = satellite clock correction expressed in meters

grpdel = group delay of the satellite expressed in meters

NOTE: The group delay term, TGD, is NOT incorporated into the clock correction term. The single-frequency user must implement this externally to this routine. Dual frequency users do not need to apply TGD. See sections 20.3.3.3.3 through 20.3.3.3.3.3 of ICD-GPS-200 for more details.

svpos

Purpose

Compute satellite position given WGS-84 Kepler parameters corresponding to ideal circular orbits. (space vehicles: sv)

Synopsis

```
svxyz = svpos(r,toe,Mo,OMGo,incl,t)
```

Description

`r` = orbit radius (semi-major axis) in meters

`toe` = reference time for Kepler parameters (time of ephemeris) in seconds

`Mo` = Mean anomaly at reference time in degrees

`OMGo` = Longitude of the ascending node at weekly epoch in degrees

`incl` = inclination angle of orbital plane in degrees

`t` = time at which to evaluate satellite position (in seconds)

`svxyz(1,1)` = ECEF x-coordinate of satellite in meters
`svxyz(2,1)` = ECEF y-coordinate of satellite in meters

`svxyz(3,1)` = ECEF z-coordinate of satellite in meters

See Also

`loadgps, loadglo, loadgg, gensv, dops`

svposalm

Purpose

Compute satellite position given WGS-84 broadcast almanac orbital parameters. (space vehicles: sv)

Synopsis

```
svxyz = svposalm(r,toe,Mo,OMGo,incl,t,...
eccen,argperi,omgdot)
```

Description

`r` = orbit radius (semi-major axis) in meters

`toe` = reference time for Kepler parameters (time of applicability ephemeris) in seconds

`Mo` = Mean anomaly at reference time in degrees

`OMGo` = Longitude of the ascending node at weekly epoch in degrees

`incl` = inclination angle of orbital plane in degrees

`t` = time at which to evaluate satellite position (in seconds)

`svxyz(1,1)` = ECEF x-coordinate of satellite in meters

`svxyz(2,1)` = ECEF y-coordinate of satellite in meters
`svxyz(3,1)` = ECEF z-coordinate of satellite in meters

See Also

`loadyuma, gensvalm, dops`

svposeph

Purpose

Compute satellite position given WGS-84 broadcast ephemeris parameters. (space vehicles: sv)

Synopsis

```
[svxyz,Ek] = svposeph(svid,t)
```

Description

```
svid = identification number of satellite

t = time at which to evaluate satellite position
(in seconds)

svxyz(1,1) = ECEF x-coordinate of satellite in
meters

svxyz(2,1) = ECEF y-coordinate of satellite in
meters

svxyz(3,1) = ECEF z-coordinate of satellite in
meters

Ek = Eccentric anomaly

NOTE:  SVPOSEPH assumes the user has used the
function LOADRINEXN to load the ephemeris
```

parameters into global memory.

tropgen

Purpose

Generate tropospheric delay in meters.

Synopsis

```
delta = tropgen(usrxyz,svxyz,humid)
```

Description

`delta` is the tropospheric delay associated with a signal propagating from a navigation satellite (located at `svxyz`) to a user (located at `usrxyz`) through humidity specified by `humid`.

`usrxyz`(1:3) = true user position in ECEF Cartesian coordinates.

`svxyz`(1:3) = position of satellite in ECEF Cartesian coordinates.

`humid` = optional argument. Humidity in percentage.

`delta` = tropospheric delay in meters.

Algorithm

The model follows the modified Hopfield algorithm given in [1] as quoted

by [2].

References

[1] "A modified Hopfield tropospheric refraction correction model," C. C. Goad and L. Goodman, Presented at the Fall Annual Meeting of the American Geophysical Union, San Francisco, December 1974.

[2] Alfred Leick, GPS Satellite Surveying, 2nd ed., Wiley-Interscience, John Wiley & Sons, New York, 1995.

[3] Global Positioning System - Theory and Practice, 2nd ed., B. Hofmann-Wellenhof, H. Lichtenegger and J. Collins, Springer-Verlag, Wien, New York, 1992.

[4] Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.

tropocorr

Purpose

Compute tropospheric correction for the dry component using the Hopfield model.

Synopsis

```
tropodel = tropocorr(svxyz,usrxyz)
```

Description

```
svxyz = satellite position expressed in ECEF
cartesian coordinates

usrxyz = user position expressed in ECEF cartesian
coordinates

tropodel = estimate of tropospheric error(meters)
```

xyz2enu

Purpose

Convert from WGS-84 ECEF Cartesian coordinates to rectangular local-level-tangent ('East'-'North'-Up) coordinates.

Synopsis

```
enu = xyz2enu(xyz,orgxyz)
```

Description

`enu(1)` = 'East'-coordinate relative to local origin (meters)
`enu(2)` = 'North'-coordinate relative to local origin (meters)
`enu(3)` = Up-coordinate relative to local origin (meters)

`orgxyz(1)` = ECEF x-coordinate of local origin in meters
`orgxyz(2)` = ECEF y-coordinate of local origin in meters
`orgxyz(3)` = ECEF z-coordinate of local origin in meters

`xyz(1,1)` = ECEF x-coordinate in meters
`xyz(2,1)` = ECEF y-coordinate in meters
`xyz(3,1)` = ECEF z-coordinate in meters

Note that ENU coordinates are truly from a local-level-tangent Cartesian coordinate system. In this local system, the x-axis points east from the local origin and the y-axis points north, et cetera. Since 'east' actually follows a line of latitude (which is actually a circle), the local Cartesian

coordinates approximately follow the geodetic directions for low and mid-latitudes only. When operating in the Earth's polar regions, the direction 'east' is not well approximated by a straight line.

Algorithm

A call to `xyz2llh` provides the geodetic coordinates of the local origin. This allows for the coordinate system rotation and translation required to convert from ECEF to ENU.

See Also

`enu2xyz, xyz2llh, llh2xyz`

References

Alfred Leick, GPS Satellite Surveying, 2nd ed., Wiley-Interscience, John Wiley & Sons, New York, 1995.

xyz2llh

Purpose

Convert from WGS-84 ECEF Cartesian coordinates to WGS-84 geodetic coordinates (latitude, longitude, height).

Synopsis

```
llh = xyz2llh(xyz)
```

Description

`xyz(1)` = ECEF x-coordinate in meters
`xyz(2)` = ECEF y-coordinate in meters
`xyz(3)` = ECEF z-coordinate in meters

`llh(1)` = latitude in radians
`llh(2)` = longitude in radians
`llh(3)` = height above ellipsoid in meters

See Also

`enu2xyz, xyz2enu, llh2xyz`

References

Understanding GPS: Principles and Applications, Elliott D. Kaplan, Editor, Artech House Publishers, Boston, 1996.