ASE 372N Laboratory Exercise Set #2: Calculating Satellite Position

Posting Date: Sept. 8, 2016
Due Date: Sept. 21, 2016

Reading

- Misra & Enge, Ch. 4
- GPS Interface Specification IS-GPS-200 (the latest revision thereof), available online at

  `http://www.gps.gov/technical/icwg/#is-gps-200`.

  With this document alone, one can build an entire GPS receiver – no other text is needed. It's dry reading, but beautifully correct: the GPS Interface Specification has been checked and revised so many times over the last 30 years that it's probably one of the most error-free technical documents of its size in existence.

  For this lab exercise, you'll want to reference sections 20.3.3.4.1 through 20.3.3.4.3.2.

Objectives

A prerequisite for calculating a GNSS position fix is knowledge of the point of origin of each transmitted signal. A GPS receiver typically determines the position of each satellite from information provided in the so-called navigation message, which is a 50-Hz data stream modulated onto the GPS carrier. In this lab exercise you'll calculate satellite positions from the orbital elements in the broadcast ephemeris.

This lab exercise is primarily a coding exercise. **You'll hand in Matlab code instead of a lab report**. You may discuss your solution approach with other class members **but do not exchange code**.

Calculating Satellite Position and Velocity from Broadcast Ephemeris

The GPS navigation message is a 50-Hz data stream that modulates each GPS satellite's ranging code and the underlying carrier to communicate information about the satellite's position and clock offset from GPS system time. Each satellite broadcasts its own 15-element ephemeris called the broadcast ephemeris. Each satellite also broadcasts a set of less accurate 8-element ephemerides (plural of ephemeris, pronounced eff-em-EHR-ah-deeze), one for each GPS satellite. This complete set of low-accuracy ephemerides is called the GPS almanac. A GPS receiver decodes and stores these data. It uses the almanac to aid in acquisition of additional GPS satellites and uses the more accurate broadcast ephemeris to calculate position solutions.

The following steps will guide you through the satellite position and velocity calculation.

Step 1

Write two Matlab functions, one for converting from UTC to GPS time and one for converting from GPS time to UTC. The functions should adhere to the following interface:

```
function [gpsWeek, gpsSec] = utc2gps(n)
% utc2gps : Convert UTC time to GPS time expressed in GPS week and GPS
%           second of week.
%
%
% INPUTS
%
% n -------------- The UTC time and date expressed as a Matlab datenum.  Use
%                  the Matlab function datenum() to generate n; use datestr()
%                  to render n in a standard format.
%
%
% OUTPUTS
%
% gpsWeek -------- The unambiguous GPS week number where zero corresponds to
%                  midnight on the evening of 5 January/morning of 6 January,
%                  1980.  By unambiguous is meant the full week count since
%                  the 1980 reference time (no rollover at 1024 weeks).
%
% gpsSec --------- The GPS time of week expressed as GPS seconds from midnight
%                  on Saturday.
%
%+-------------------------------------------------------------------------------+
% References:
%
%
% Author:
%+===============================================================================+



function [n] = gps2utc(gpsWeek,gpsSec)
% gps2utc : Convert GPS time expressed in GPS week and GPS second of week to
%           UTC.
%
%
% INPUTS
%
% gpsWeek -------- The unambiguous GPS week number where zero corresponds to
%                  midnight on the evening of 5 January/morning of 6 January,
%                  1980.  By unambiguous is meant the full week count since
%                  the 1980 reference time (no rollover at 1024 weeks).
%
% gpsSec --------- The GPS time of week expressed as GPS seconds from midnight
%                  on Saturday.
%
%
% OUTPUTS
%
% n -------------- The UTC time and date expressed as a Matlab datenum.  Use
```

```
%                   the Matlab function datestr() to read n in a standard
%                   format.
%
%+------------------------------------------------------------------------------+
% References:
%
%
% Author:
%+==============================================================================+
```

Note that each of these functions will need some way of internally determining the current number of leap seconds separating UTC from GPS time. A simple constant will do, but you may wish to think of something more clever so that your code doesn't become incorrect with the next adjustment of the leap second or when operating on historical data. You may wish to use this handy time scale comparison written by Tom Van Baak to check your functions:

`http://leapsecond.com/java/gpsclock.htm`

Step 2

Download the `brdc` GPS navigation data file for September 1, 2013, from NOAAs archive of such data in RINEX format:

`ftp://www.ngs.noaa.gov/cors/rinex`

The site is organized into a series of directories: by year, by day of year, and then by operating station site. In the day of year directory you'll find a file name that starts with `brdc` and ends with `n.gz`. For example, for the 85th day of the year in 2013, you'll find the file `brdc0850.13n.gz` in the `rinex/2013/085` directory. It's easy to calculate the day of year for September 1, 2013 using Matlab's `datenum()` function.

The data type is navigation data, which explains the 'n' in `.13n`. The `.gz` extension means the file has been "g-zipped" (compressed). You'll have to "unzip" (uncompress) it. In Linux, just use, e.g., `gzip -d brdc0850.13n.gz`. In Windows 7, you can use 7-zip. On either platform, you can also use Matlab's `gunzip` function.

Take a look at the contents of the file. RINEX stands for Receiver Independent Exchange format, a format for sharing navigation data and GPS observables without regard for the particular type of receiver that collected the data. This was a noble idea when it was first introduced, but by now RINEX is just an archaic outdated outmoded cryptic byzantine data format that is best left back in the 20th century. Close the file and never look back.

Step 3

Download the following Matlab functions and scripts from the course website and put them in your working directory:

`navConstants.m`
`retrieveNavigationData.m`

`getephem.m`
`Rinex2p1Script.m`

Take a look at each of these files. The script `navConstants.m` contains universal constants and WGS-84 constants useful for GPS navigation. The function `retrieveNavigationData.m`, which calls the subroutines `getephem.m` and `Rinex2p1Script.m`, reaches out and retrieves the appropriate RINEX file from the NOAA ftp server, parses it, and then packages up all the relevant ephemeris information in an easy-to-use Matlab structure.

Using your `utc2gps` function, figure out the right GPS week and GPS second of week for 12:00:00 (noon) UTC time on September 1, 2013. Plug these into `retrieveNavigationData.m` along with the other arguments to obtain the vector `satdata` and the structure `ionodata` that correspond to this epoch. `satdata` is a vector of 32 structures, one for each of 32 possible GPS satellites. Each structure's fields and corresponding units are defined in the comments within `retrieveNavigationData.m`. Index through the `satdata` vector and make sure most of its structures are full (empty structures correspond to non-existant satellites).

Note that `retrieveNavigationData.m` may fail to download the target RINEX file if your network connection is over a wireless link: the timeout within the Matlab function `urlwrite` doesn't like the higher latency of wireless links. Remedy this problem in one of three ways: (1) Operate on a machine (such as a Mac or Linux box, or Windows with Cygwin) that has the utility `wget` installed (the `retrieveNavigationData.m` function will then use `wget` instead of `urlwrite`), (2) use a wired network connection, (3) download the RINEX file manually, as instructed by the error message.

Step 4

Write a Matlab function for determining the position and velocity of a GPS satellite in the ECEF reference frame from the broadcast ephemeris. Your function should adhere to the following interface:

```
function [rSvEcef, vSvEcef] = satloc(gpsWeek, gpsSec, sd)
% satloc : Return satellite location and velocity expressed in and relative to
%          the ECEF reference frame.
%
%
% INPUTS
%
% gpsWeek ---- Week of true time at which SV location and velocity are
%              desired.
%
% gpsSec ----- Seconds of week of true time at which SV location and velocity
%              are desired.
%
% sd --------- Ephemeris structure array for a single SV.  Let ii be the
%              numerical identifier (PRN identifier) for the SV whose location
%              is sought.  Then sd = satdata(ii).  sd has the following
%              fields:
%
%       SVID - satellite number
%       health - satellite health flag (0 = healthy; otherwise unhealthy)
%       we - week of ephemeris epoch (GPS week, unambiguous)
%       te - time of ephemeris epoch (GPS seconds of week)
%       wc - week of clock epoch (GPS week)
```

```
%       tc - time of clock epoch (GPS seconds of week)
%       e - eccentricity (unitless)
%       sqrta - sqrt of orbit semi-major axis (m^1/2)
%       omega0 - argument of perigee (rad.)
%       M0 - mean anomaly at epoch (rad.)
%       L0 - longitude of ascending node at beginning of week (rad.)
%       i0 - inclination angle at epoch (rad.)
%       dOdt - longitude rate (rad / sec.)
%       dn - mean motion difference (rad / sec.)
%       didt - inclination rate (rad / sec.)
%       Cuc - cosine correction to argument of perigee (rad.)
%       Cus - sine correction to argument of perigee (rad.)
%       Crc - cosine correction to orbital radius (m)
%       Crs - sine correction to orbital radius (m)
%       Cic - cosine correction to inclination (rad.)
%       Cis - sine correction to inclination (rad.)
%       af0 - 0th order satellite clock correction (s)
%       af1 - 1st order satellite clock correction (s / s)
%       af2 - 2nd order satellite clock correction (s / s^2)
%       TGD - group delay time for the satellite (s)
%
%
% OUTPUTS
%
% rSvEcef ---- Location of SV at desired time expressed in the ECEF reference
%              frame (m).
%
% vSvEcef ---- Velocity of SV at desired time relative to the ECEF reference
%              frame and expressed in the ECEF reference frame (m/s). NOTE:
%              vSvEcef is NOT inertial velocity, e.g., for geostationary SVs
%              vSvEcef = 0.
%
%+-------------------------------------------------------------------------+
% References:
%
%
%
%
% Author:
%+=========================================================================+
```

Definitions for the GPS ephemeris data elements can be found in the function comments above and in Table 20-II of the GPS IS. Units of the original broadcast ephemeris data are given in Table 20-III of the GPS IS. These include some strange units like "semicircles." The `getephem.m` function converts all units to meters, radians, and seconds before packaging the data into the `satdata` structure.

Use the step-by-step procedure you were given in lecture to develop the position component of your `satloc` function. If you wish, you can also refer to the official procedure for calculating satellite position, given in Table 20-IV of the GPS IS. Aside from some minor notational differences, the official procedure is essentially equivalent to the one given in lecture. Conversion from perifocal to ECEF coordinates is implemented in the final calculations shown at the bottom of Table 20-IV.

Pay close attention to ensure that your units are consistent throughout. The entire position calculation should only take you about 20 lines of Matlab code, excluding comments.

Calculation of the SV velocity relative to the ECEF frame and expressed in the ECEF frame is not explicitly discussed in the GPS IS (nor was it discussed in lecture). However, it can be derived via standard calculus techniques (derivatives, chain rule, etc). For help with these calculations, you may wish to consult the following paper:

Remondi, Benjamin W., "Computing satellite velocity using the broadcast ephemeris," GPS Solutions, vol. 8 no. 3, 2004.

Step 5

To test your `satloc` function, compare it against the following reference inputs and outputs of a correctly-implemented `satloc` function. The outputs are the ECEF position and velocity for GPS satellites with pseudorandom code number (PRN) identifiers 1, 5, and 31. The GPS week and seconds are given below. These correspond to a date in 2011. The satellite data `satdata(1)` is also given so that you can ensure you're using data corresponding to the appropriate ephemeris epoch.

```
>> gpsWeek
gpsWeek =
        1653
>> gpsSec
gpsSec =
          570957.338101566
>> satdata(1)
ans =
      SVID: 1
    health: 63
        we: 1653
        te: 576000
        wc: 1653
        tc: 575999.999996647
         e: 0.000301708350889
     sqrta: 5153.66840553
    omega0: 1.77176130104
        M0: -2.03728834211
        L0: -2.88543635184
        i0: 0.960714536657
      dOdt: -7.8396122656e-009
        dn: 4.39839749662e-009
      didt: -4.64305054455e-010
       Cuc: -5.40167093277e-007
       Cus: 9.78447496891e-006
       Crc: 192.0625
       Crs: -6.90625
       Cic: 5.40167093277e-008
       Cis: 1.49011611938e-008
       af0: -2.90526077151e-006
       af1: -1.5916157281e-012
       af2: 0
       TGD: 7.91624188423e-009
>> [rSvEcef, vSvEcef] = satloc(gpsWeek, gpsSec, satdata(1))
rSvEcef =
          5734379.67991244
        -18348853.9562115
        -18337913.5840076
vSvEcef =
          2075.74428570655
         -1061.65685114262
          1711.92358273592
>> [rSvEcef, vSvEcef] = satloc(gpsWeek, gpsSec, satdata(5))
rSvEcef =
        -16008969.0055597
          7217735.69592498
          19948401.7376856
vSvEcef =
         -2148.64330656461
         -1349.59716343779
         -1223.93898463887
>> [rSvEcef, vSvEcef] = satloc(gpsWeek, gpsSec, satdata(31))
rSvEcef =
          24022243.1967653
         -6336813.56153242
         -9489878.85576618
vSvEcef =
         -972.709470938258
          766.921515088891
          -2892.2628384092
```

Exercises

Calculate the ECEF position and velocity of the GPS satellites with PRN identifiers 2 and 5 for September 1, 2013 at 12:00:00 UTC. Repeat the calculations for one second later, or September 1, 2013 at 12:00:01 UTC. Show that the difference in position over the 1-second interval is consistent with the calculated velocity.

Wrap-up

No formal lab report is required for this lab. Hand in a copy of your `utc2gps.m`, `gps2utc.m`, and `satloc.m` functions. Also hand in answers to the exercise questions. Email your `satloc.m` function to the TA. You will be graded on your code style, so make your code is clear and well-documented. Neat and easy-to-understand Matlab code will be graded higher than sloppy and hard-to-understand code.