GPS Modeling and Analysis

Summary of Research: GPS Satellite Axial Ratio Predictions

Principal Investigator: Penina Axelrad
Graduate Research Assistant: Lisa Reeh

Period Covered: 1/15/2002 – 1/14/2003

Colorado Center for Astrodynamics Research
University of Colorado, UCB 431
Boulder, CO 80309-0431

Grant No: NAG5-11421

Report No: PA-02-145
Date: September 23, 2002

Abstract

This report outlines the algorithms developed at the Colorado Center for Astrodynamics Research to model yaw and predict the axial ratio as measured from a ground station. The algorithms are implemented in a collection of Matlab functions and scripts that read certain user input, such as ground station coordinates, the UTC time, and the desired GPS satellites, and compute the above-mentioned parameters. The position information for the GPS satellites is obtained from Yuma almanac files corresponding to the prescribed date. The results are displayed graphically through time histories and azimuth-elevation plots.

Table of Contents

List of Figures

1.0 Introduction

1.1 Purpose

This report outlines the algorithms developed at the Colorado Center for Astrodynamics Research to model yaw and predict the axial ratio as measured from a ground station. The algorithms are implemented in a collection of Matlab functions and scripts that read certain user input, such as ground station coordinates, the UTC time, and the desired GPS satellites, and compute the above-mentioned parameters. The position information for the GPS satellites is obtained from Yuma almanac files corresponding to the prescribed date. The results are displayed graphically through time histories and azimuth-elevation plots.

1.2 Document Organization

Section 2 describes the calculations used to determine the visibility of the GPS satellites. This includes defining the input parameters, loading the appropriate almanac data, and calculating the position and velocity of each satellite.

Section 3 illustrates the coordinate systems used throughout the algorithms, and also defines the transformations between coordinate systems. Calculation of GPS time is described, as well as the calculation of the sun vector.

Section 4 presents the algorithms for ideal yaw angle determination and the axial ratio modeling. Section 5 contains examples of the calculations performed in this package. Azimuth/Elevation plots showing satellite paths and ideal yaw angles are presented, as well as sample time histories of HDOP, VDOP, and GDOP. Axial ratio variations are displayed as a function of time for each GPS satellite. Section 6 lists references utilized during this project.

2.0 Visibility Calculations

2.1 Creating and Loading Input

The script *almanac* is essentially the executable that runs the entire set of functions. The first operation performed by *almanac* is calling the script, *load_input*, which loads input parameters from *user_input*. This input file contains the following information:

- latitude (degrees), longitude (degrees), and altitude (meters) of the reference ground station;
- year, month, date, hour, minute, and second defining the start time of all calculations;
- duration, in hours and minutes, over which calculations are to be performed;
- PRN numbers of the satellites to be included in the calculations;
- mask table of azimuths and corresponding elevations, used to determine satellite visibility;
- table of axial ratios (dB) as a function of the off-boresite angle and the azimuth of the satellite antenna with respect to the line of sight vector from satellite to ground station.

The user may specify an azimuth-dependent elevation mask by choosing a vector of azimuths in increments no smaller than one degree. The user then must choose a corresponding vector of cut-off elevations. For example, setting the azimuth vector to [90, 270, 360] with a corresponding elevation cut-off vector of [10, 40, 50] would set a 10-degree elevation cut-off for azimuths between 0 and 90 degrees, a 40-degree elevation cut-off for azimuths between 91 and 270 degrees, and a 50-degree elevation cut-off for azimuths between 271 and 360 degrees.

Once the parameters have been loaded, *load_input* converts the UTC time to GPS time in weeks and seconds. It also converts the latitude, longitude, altitude coordinates of the reference station to Earth-Centered, Earth-Fixed (ECEF) coordinates, which will be discussed in Section 3.

2.2 Loading almanac data

The script *load_almanac* opens the almanac file corresponding to the GPS week computed from the user-specified time. It reads the following parameters and stores the information in the structure array, *alm_data*:

- PRN,
- eccentricity,
- toa (almanac time of applicability) in seconds,
- inclination in degrees,
- right ascension of the ascending node (RAAN) in radians,
- rate of change of RAAN in radians per second,
- semi-major axis in meters,
- argument of perigee in radians,
- mean anomaly in radians.

2.3 Calculating position, velocity, and visibility

Most of the calculations in this collection of functions and scripts are carried out by the script, *alm_pos_az_el*. For each minute of the user-specified time duration, several time-dependent parameters are computed. The Julian Date is computed as shown below [2]. Please note that (1) is only valid for dates between March 1, 1900 and February 28, 2100.

$$JD = 367 \cdot year - INT\left\{\frac{7 \cdot \left[year + INT\left(\frac{month+9}{12}\right)\right]}{4}\right\} + \ldots$$
$$\ldots + INT\left(\frac{275 \cdot month}{9}\right) + day + 17210135 + \left(\frac{\frac{\frac{sec}{60} + min}{60} + hour}{24}\right) \quad (1)$$

The adjusted Julian Date is determined by adding the one-minute time increment to the Julian Date of the previous time step. The Greenwich Sidereal Time is computed by

$$\theta_{GST} = 67{,}310.54841^{s} + \left(876{,}600^{h} + 8{,}640{,}184.812866^{s}\right)T + 0.093104T^{2} - 6.2 \cdot 10^{-6}T^{3} \quad (2)$$

where the *h* and *s* superscripts denote units of hours and seconds, respectively, and T is the number of Julian centuries elapsed since the J2000 epoch [2].

$$T = \frac{JD - 2{,}451{,}545}{36525} \quad (3)$$

Assuming latitude and longitude have been converted to radians, the local up (LU) vector at the reference station is calculated using

$$\overrightarrow{LU} = [\cos(\phi)\cos(\lambda) \quad \cos(\phi)\sin(\lambda) \quad \sin(\phi)] \quad (4)$$

where ϕ is geodetic latitude and λ is longitude. Finally, both the position vector of the reference station and the local up vector are rotated into the inertial frame. Coordinate transformations will be addressed in Section 3.

The function *alm_pos_vel* is called to compute the inertial satellite position and velocity. Please refer to [1] for the complete algorithm used. The line of sight vector ($\vec{e}$) is calculated by subtracting the inertial position vector of the reference station from the inertial satellite position vector. Using knowledge of $\vec{e}$ and the local up vector, the elevation angle of the satellite can be computed from the dot product of the two vectors.

$$elevation\ angle = \sin^{-1}\left[\frac{\vec{e} \bullet \overrightarrow{LU}}{\|\vec{e}\| \cdot \left\|\overrightarrow{LU}\right\|}\right] \quad (5)$$

The azimuth of the satellite, as seen by the user at the reference station, is the angle between north and the projection of $\vec{e}$ onto the local horizonal plane. First $\vec{e}$ is rotated into the north, east, down (NED) frame (see Section 3), and the "down" component is removed. This projects $\vec{e}$ into the north/east plane.

$$\vec{e}_{proj}^{\,NED} = \begin{bmatrix} e_1^{NED} & e_2^{NED} & 0 \end{bmatrix} \tag{6}$$

The azimuth angle is then computed as the angle between this projected $\vec{e}$ and the north unit vector.

$$azimuth\ angle = \cos^{-1}\left[\frac{e_1^{NED}}{\left\| \vec{e}_{proj}^{\,NED} \right\|}\right] \tag{7}$$

The sign of the azimuth angle should be the same as the sign of the "east" component of the projected line of sight vector; positive for clockwise from north, negative for counterclockwise. After the sign has been correctly determined, a check is performed to place the azimuth angle between 0 and 360 degrees.

The visibility of each satellite can be determined using information in the user-specified mask table, and the computed satellite elevation and azimuth. For a particular azimuth, if the elevation of the satellite is greater than the cut-off elevation, the satellite is considered visible. The line of sight unit vector of each visible satellite is added to an "H" matrix as shown below. The numbered subscripts are used to differentiate between satellites. For example, e_{1x}^{NED} refers to the x-component of the line of sight vector for satellite 1 in the NED frame.

$$H = \begin{bmatrix} e_{1x}^{NED} & e_{1y}^{NED} & e_{1z}^{NED} & 1 \\ e_{2x}^{NED} & e_{2y}^{NED} & e_{2z}^{NED} & 1 \\ e_{3x}^{NED} & e_{3y}^{NED} & e_{3z}^{NED} & 1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{8}$$

From the H matrix, "P" can be computed by

$$P = \left(H^T H\right)^{-1}. \tag{9}$$

The following dilution of precision (DOP) values can then be calculated:

$$\begin{aligned} GDOP &= \sqrt{trace(P)} \\ HDOP &= \sqrt{P(1,1) + P(2,2)} \\ VDOP &= \sqrt{P(3,3)} \\ PDOP &= \sqrt{P(1,1) + P(2,2) + P(3,3)} \\ TDOP &= \sqrt{P(4,4)} \end{aligned} \tag{10}$$

Time histories of HDOP, VDOP, and GDOP are presented graphically.

2.4 Display satellite positions and visibility

Results for each satellite at each time step are presented on an azimuth-elevation plot. The paths of the visible satellites are plotted as a function of their azimuths and elevations. The elevation cut-off line, as determined by the mask table, is plotted on the same graph to show when satellites enter and leave the field of view. Examples of az-el plots are presented in Section 5.

3.0 Coordinate Systems, Transformations, and Time Systems

This section describes the various coordinate systems and time systems used throughout the algorithms, and transformations between these coordinate systems.

3.1 Coordinate Systems

3.1.1 Earth-Centered, Earth-Fixed Coordinate System

The Earth-Centered, Earth-Fixed (ECEF) coordinate system has its origin at the center of the earth and its principal direction extending towards the point where the Greenwich meridian intersects the equator. Using the World Geodetic System (WGS)-84 reference frame, the *X*-axis is the principal direction and the *Z*-axis points towards the "adopted geographic pole", which is the location of the spin axis pole at a particular epoch, accounting for polar motion. The *Y*-axis is mutually perpendicular to these axes, forming a right-handed coordinate system.

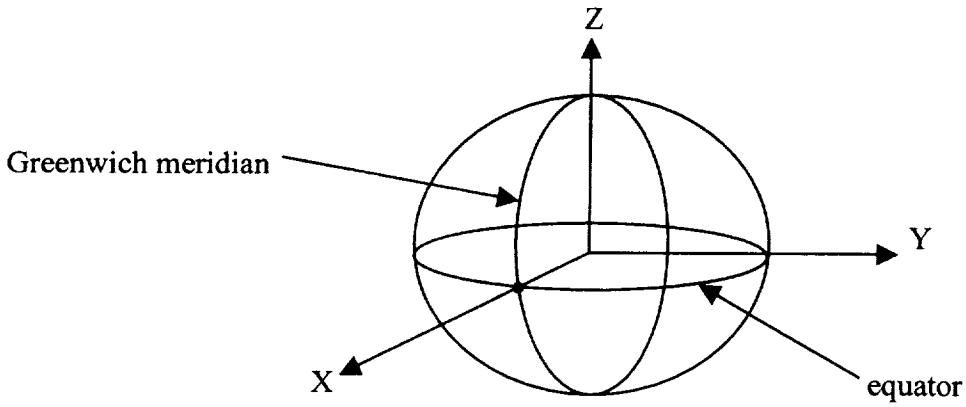

Figure 1. Earth-Centered, Earth-Fixed Coordinate System

3.1.2 Earth-Centered Inertial Coordinate System

The Earth-Centered Inertial (ECI) coordinate system has its origin at the center of the earth, and its principal direction extending towards the first point in Aries (the vernal equinox of J2000). The *X*-axis is the principal direction and the *Y*-axis is 90 degrees to the east in the earth's equatorial plane. The *Z*-axis points towards the North Pole.

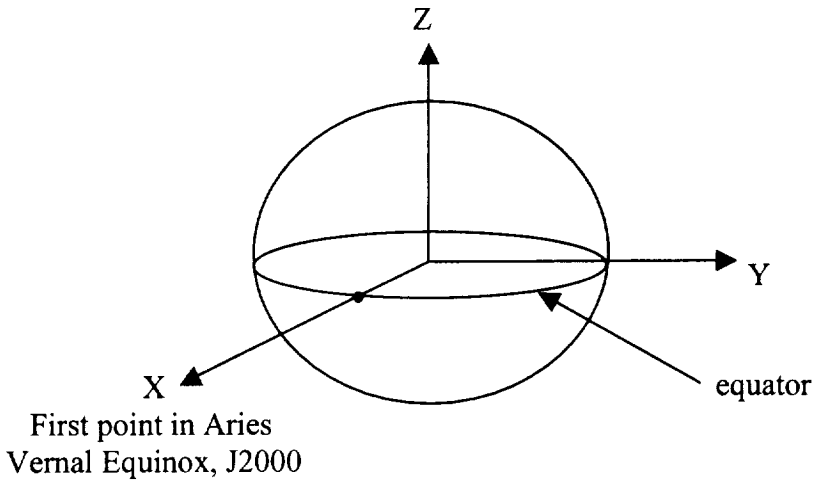

Figure 2. Earth-Centered Inertial Coordinate System

3.1.3 North, East, Down Coordinate System

The North, East, Down (NED) coordinate system is centered at some specified location on the earth's surface. The *N*-axis points due north and the *E*-axis points due east, creating a reference plane on the local horizon. The *D*-axis points radially inward, perpendicular to the surface of the earth.

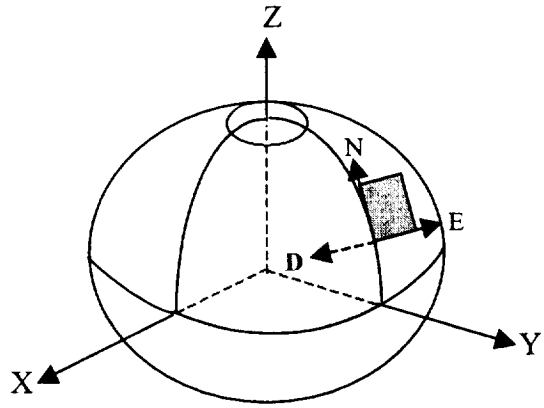

Figure 3. North, East, Down Coordinate System

3.1.4 Radial, In-Track, Cross-Track Coordinate System

The Radial, In-Track, Cross-Track (RIC) coordinate system moves with the spacecraft in its orbit. *R* points away from the spacecraft from the center of the earth and *I* is perpendicular to the *R* vector in the orbit plane. Finally, the *C* vector points in the direction of the angular momentum of the spacecraft, perpendicular to the orbit plane.

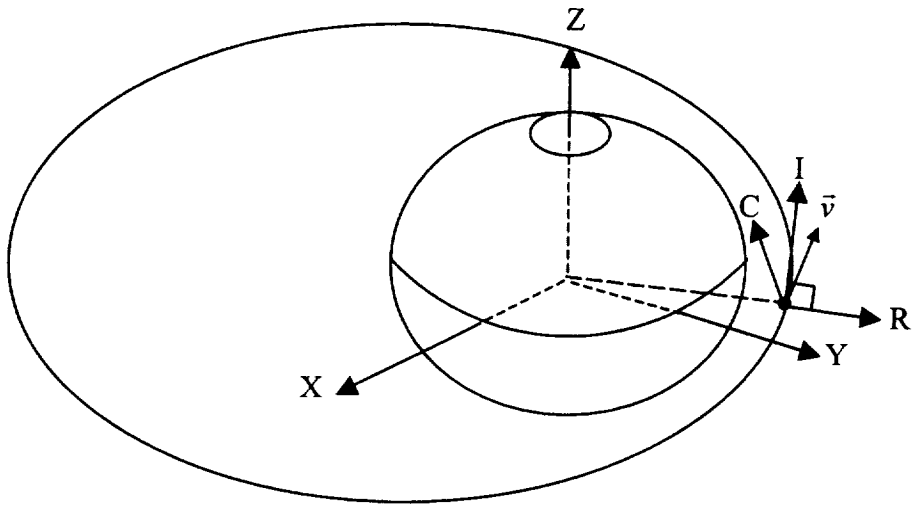

Figure 4. Radial, In-Track, Cross-Track Coordinate System

3.1.5 Local Antenna Coordinate System

The Local Antenna coordinate system is also centered on the spacecraft in orbit. It is defined by a rotation about the Radial axis of the RIC system described above. The local antenna frame is designated by x_A, y_A, and z_A, where the x_A vector is parallel to the Radial vector of the RIC frame. The y_A and z_A axes are rotated in a clockwise sense through the yaw angle about the R axis. Thus, the y_A and z_A vectors are parallel to the I and C vectors, respectively, when the ideal yaw angle of the satellite is zero.

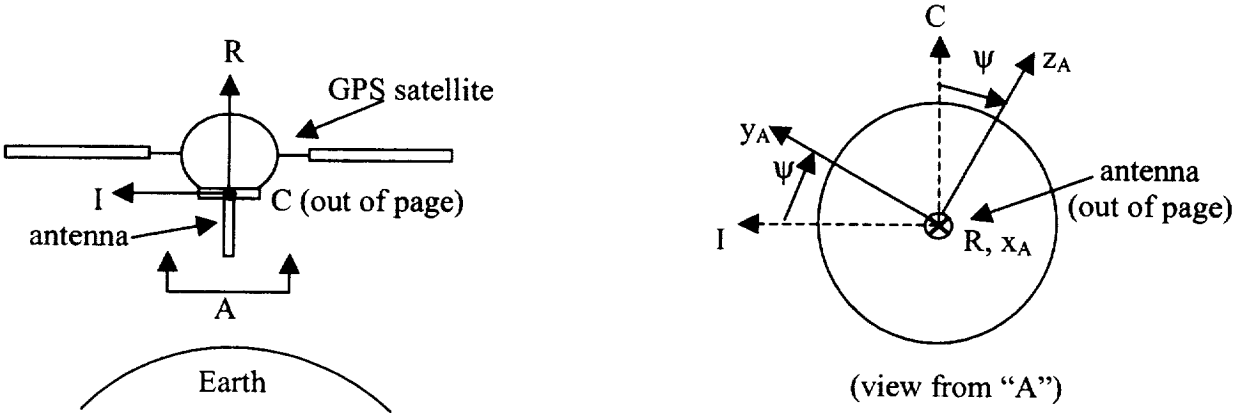

Figure 5. Local Antenna Coordinate System

In the above figure, ψ represents the ideal yaw angle. In the view on the right, the earth-pointing GPS antenna points out of the page, whereas the R and x_A vectors point into the page.

3.2 Coordinate Transformations

3.2.1 Latitude, Longitude, Altitude to Earth-Centered, Earth-Fixed

The purpose of the function, *wgslla2xyz*, is to convert the input location of the reference station, described by latitude, longitude, and altitude, to *x*, *y*, *z* coordinates in the ECEF frame. In this algorithm, the following parameters are first defined [3]:

a_E = mean Earth radius = 6378137 meters
f = flattening coefficient = 1/298.257223563

$$r_n = \frac{a_E}{\sqrt{1-\sin^2(\phi)[f(2-f)]}}$$

Then the expressions for *x*, *y*, and *z* are as follows [3]:

$$\begin{aligned} x_{ECEF} &= (r_n + altitude)\cos(\phi)\cos(\lambda) \\ y_{ECEF} &= (r_n + altitude)\cos(\phi)\sin(\lambda) \\ z_{ECEF} &= \{r_n[1-f(2-f)] + altitude\}\sin(\phi) \end{aligned} \tag{11}$$

3.2.2 Earth-Centered, Earth-Fixed to Earth-Centered Inertial

Since the algorithm obtained from [1] computes satellite position and velocity in the inertial frame, it is necessary to convert the ECEF station coordinates to inertial coordinates so the line of sight vector can be determined. This transformation involves an R_3 rotation about the negative of the Greenwich Sidereal Time angle [2].

$$R_3(-\theta_{GST}) = \begin{bmatrix} \cos(-\theta_{GST}) & \sin(-\theta_{GST}) & 0 \\ -\sin(-\theta_{GST}) & \cos(-\theta_{GST}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Using this rotation matrix, the reference station position vector can be transformed from ECEF to ECI as follows:

$$\begin{bmatrix} x_{ECI} \\ y_{ECI} \\ z_{ECI} \end{bmatrix} = R_3(-\theta_{GST}) \begin{bmatrix} x_{ECEF} \\ y_{ECEF} \\ z_{ECEF} \end{bmatrix} \tag{13}$$

3.2.3 Earth-Centered Inertial to Earth-Centered, Earth-Fixed

This transformation is performed on the line of sight vector so that it can subsequently be rotated from ECEF coordinates to NED coordinates. Once in NED coordinates, azimuth can be determined, as previously discussed. The ECEF to ECI transformation involves an R_3 rotation

about the Greenwich Sidereal Time angle [2].

$$R_3(\theta_{GST}) = \begin{bmatrix} \cos(\theta_{GST}) & \sin(\theta_{GST}) & 0 \\ -\sin(\theta_{GST}) & \cos(\theta_{GST}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

The vector expressed in the ECEF frame is

$$\begin{bmatrix} x_{ECEF} \\ y_{ECEF} \\ z_{ECEF} \end{bmatrix} = R_3(\theta_{GST}) \begin{bmatrix} x_{ECI} \\ y_{ECI} \\ z_{ECI} \end{bmatrix}. \tag{15}$$

3.2.4 Earth-Centered, Earth-Fixed to North, East, Down

This coordinate transformation is used to rotate the line of sight vector from the ECEF system to the NED system so the azimuth of the satellite can easily be determined. The rotation matrix for this conversion utilizes the latitude and longitude of the reference station, which is the origin of the line of sight vector. The rotation matrix is shown below:

$$R_{NED}^{ECEF} = \begin{bmatrix} -\cos(\lambda)\sin(\phi) & -\sin(\lambda)\sin(\phi) & \cos(\phi) \\ -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\lambda)\cos(\phi) & -\sin(\lambda)\cos(\phi) & -\sin(\phi) \end{bmatrix} \tag{16}$$

The vector expressed in the NED frame is

$$\begin{bmatrix} x_{NED} \\ y_{NED} \\ z_{NED} \end{bmatrix} = R_{NED}^{ECEF} \begin{bmatrix} x_{ECEF} \\ y_{ECEF} \\ z_{ECEF} \end{bmatrix}. \tag{17}$$

3.2.5 Earth-Centered Inertial to Local Antenna Coordinates

In order to determine the azimuth of the satellite antenna with respect to the line of sight vector, the line of sight vector must be represented in a local coordinate system fixed to the satellite. This is accomplished in two steps: first, $\vec{e}$ is rotated from inertial coordinates into a Radial, In-Track, Cross-Track (RIC) coordinate system, and second, into the local antenna coordinate system. The *R*, *I*, and *C* vectors that comprise the rows of the rotation matrix for the ECI to RIC transformation are computed as shown in (18). The *r* and *v* vectors are the position and velocity, respectively, of a particular satellite expressed in the inertial frame [2].

$$\begin{aligned} \left[\hat{R}\right]^{ECI} &= \frac{\vec{r}_{sat}^{\,ECI}}{\left\|\vec{r}_{sat}^{\,ECI}\right\|} \\ \left[\hat{C}\right]^{ECI} &= \frac{\vec{r}_{sat}^{\,ECI} \times \vec{v}_{sat}^{\,ECI}}{\left\|\vec{r}_{sat}^{\,ECI} \times \vec{v}_{sat}^{\,ECI}\right\|} \\ \left[\hat{I}\right]^{ECI} &= \hat{C} \times \hat{R} \end{aligned} \tag{18}$$

The rotation matrix is [2]:

$$R_{RIC}^{ECI} = \begin{bmatrix} \hat{R}_x^{ECI} & \hat{R}_y^{ECI} & \hat{R}_z^{ECI} \\ \hat{I}_x^{ECI} & \hat{I}_y^{ECI} & \hat{I}_z^{ECI} \\ \hat{C}_x^{ECI} & \hat{C}_y^{ECI} & \hat{C}_z^{ECI} \end{bmatrix} \tag{19}$$

Next, an R_1 rotation about the ideal yaw angle is performed to place the line of sight vector in the local antenna coordinate system using the rotation matrix in (20) [2].

$$R_1(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{bmatrix} \tag{20}$$

The transformation from ECI to the local antenna frame (A) is

$$\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix} = R_A^{RIC} R_{RIC}^{ECI} \begin{bmatrix} x_{ECI} \\ y_{ECI} \\ z_{ECI} \end{bmatrix}. \tag{21}$$

3.3 Time Systems

3.3.1 Converting UTC time to GPS time

The following algorithm involves calculation of the Julian Date, illustrated by (1). The GPS week number can then be determined as follows:

$$GPS\ week = INT\left(\frac{JD - 2444244.5}{7}\right) \tag{22}$$

To account for the GPS rollover every 1,024th week, the *modulo* function is used to ensure that a week number between 0 and 1,023 is obtained. The GPS seconds of the week are determined using (24).

$$GPS\ seconds = round\left[\left(\frac{JD - 2444244.5}{7} - GPS\ week\right) \cdot 7 \cdot 24 \cdot 3600\right] \tag{23}$$

The "round" function is used to round the quantity in brackets to the nearest integer.

3.4 Computing the Sun vector

In order to model the ideal yaw angle of the satellite, which will be discussed in the next section, the unit vector that points from the center of the earth to the center of the sun must be computed. This calculation involves knowledge of the number of elapsed Julian centuries, T, and the Greenwich Hour Angle, θ_{GST}. The following algorithm is used [2]:

Determine the mean longitude of the sun in degrees:

$$\lambda_{sun} = 280.4606184 + 36000.77005361 \cdot T \tag{24}$$

Determine the mean anomaly of the Sun in degrees:

$$M_{sun} = 357.5277233 + 35999.05034 \cdot T \tag{25}$$

Make sure that both λ_{sun} and M_{sun} are between 0 and 360 degrees. Calculate the ecliptic longitude:

$$\lambda_{ecl} = \lambda_{sun} + 1.914666471 \sin(M_{sun}) + 0.019994643 \sin(2M_{sun}) \tag{26}$$

Compute the obliquity of the ecliptic in degrees:

$$\varepsilon = 23.439291 - 0.0130042 \cdot T \tag{27}$$

The sun position unit vector is then calculated in inertial coordinates using (28).

$$\vec{r}_{sun} = \begin{bmatrix} \cos(\lambda_{ecl}) \\ \cos(\varepsilon)\sin(\lambda_{ecl}) \\ \sin(\varepsilon)\sin(\lambda_{ecl}) \end{bmatrix} \tag{28}$$

4.0 Yaw Angle and Axial Ratio Modeling

This section describes the calculation of the ideal yaw angles of the GPS satellites, and explains how the axial ratios are modeled.

4.1 Determine Beta, Alpha, and Yaw Angles

The figure below describes the geometry of the beta and alpha angles.

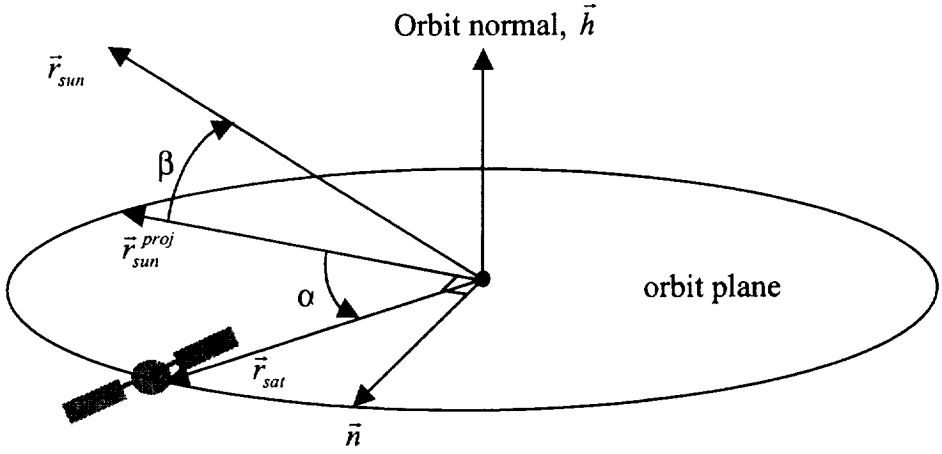

Figure 6. Orbit Plane Geometry

From Figure 6, it is clear that β can be found in much the same way that the elevation angles of the satellites were determined using (5). β is simply the complement of the angle between the sun unit vector and the orbit normal vector, $\vec{h}$.

Determining α is more complicated. α is the angle in the orbit plane between the projection of the sun vector onto the orbit plane and the satellite position vector. It can be computed through a series of cross products. Crossing the orbit normal with the sun unit vector results in the vector $\vec{n}$, which is perpendicular to both vectors and, thus, lies in the orbit plane.

$$\vec{n} = \vec{h} \times \vec{r}_{sun} \tag{29}$$

Crossing $\vec{n}$ with $\vec{h}$ will produce the projection of the sun unit vector in the orbit plane.

$$\vec{r}_{sun}^{\,proj} = \vec{n} \times \vec{h} \tag{30}$$

Figure 6 describes the geometry of (29) and (30). α can then be determined as follows:

$$\alpha = \tan^{-1}\left[\frac{\vec{r}_{sat} \cdot \vec{n}}{\vec{r}_{sat} \cdot \vec{r}_{sun}^{\,proj}}\right] \tag{31}$$

The following model [4] is used to obtain the ideal yaw angle, ψ, for a given α and β.

$$\psi_0 = \tan^{-1}\left[\frac{\tan(\beta)}{\sin(\alpha)}\right] \tag{32}$$

This equation has the following conditions for determining ψ:

$$\begin{aligned}
&\textit{for } \beta > 0: \\
&\quad \textit{if } 0 < \alpha < 180 \Rightarrow \psi = \psi_0 \\
&\quad \textit{if } 180 < \alpha \leq 360 \Rightarrow \psi = \psi_0 + 180 \\
&\textit{for } \beta \leq 0: \\
&\quad \textit{if } 180 < \alpha < 360 \Rightarrow \psi = \psi_0 + 180 \\
&\quad \textit{otherwise } \psi = \psi_0
\end{aligned}$$

Figure 7 gives a graphical representation of the model described in (31) and (32) [4, 5].

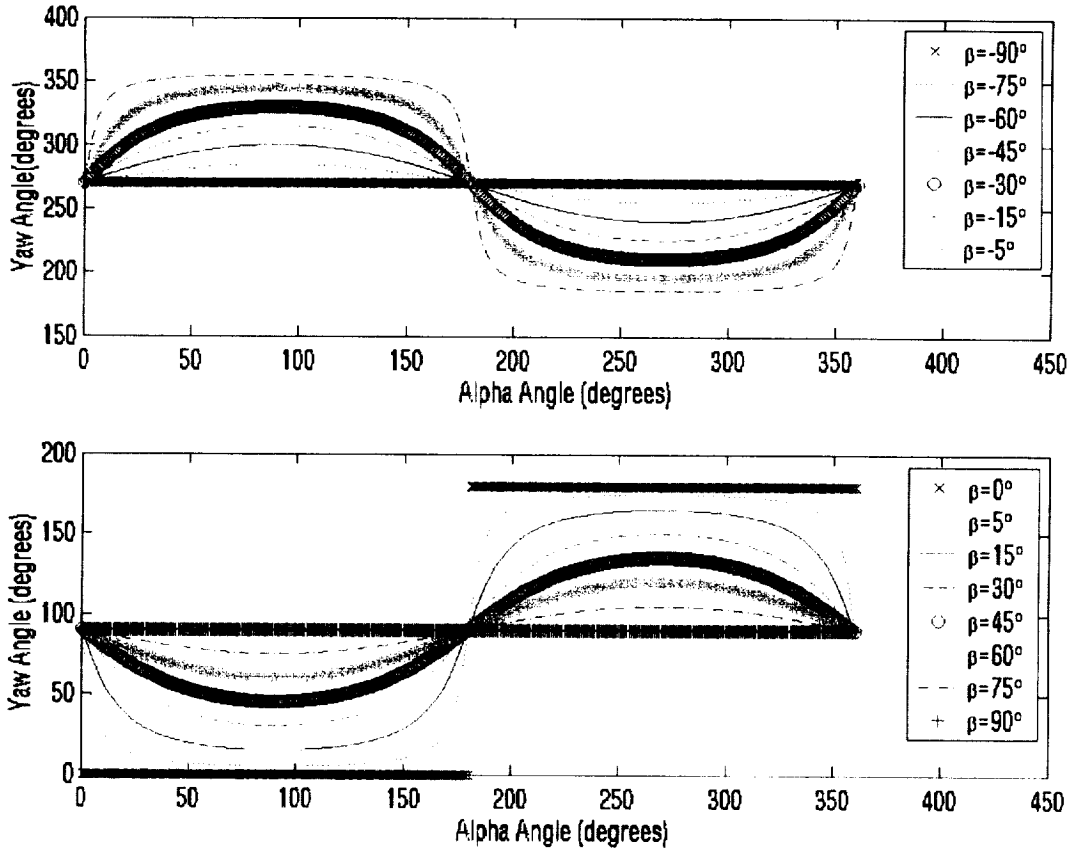

Figure 7. Graphical Representation of Ideal Yaw Model

4.2 Display Ideal Yaw Angles

As described in Section 2.4, the satellite paths are displayed on an Az-El plot for the specified time duration. In addition, the ideal yaw angles for these satellites are also illustrated on the same plot. While the satellite positions are plotted every minute, ideal yaw angles are only plotted every 30 minutes. The yaw angles are represented by arrows that are measured positive clockwise from the top of the page. An example of this type of plot is included in Section 5.

4.3 Determining the Axial Ratio

Section 3.2.5 described how the line of sight vector is rotated into the local antenna coordinate system. Using (33), the azimuth of the GPS satellite's antenna with respect to the line of sight vector can be calculated as

$$AZ_{ant} = \tan^{-1}\left[\frac{\vec{e}_z^{\,A}}{\vec{e}_y^{\,A}}\right] \tag{33}$$

where the subscripted letters refer to vector components, and the superscript A denotes that the line of sight vector is in the local antenna coordinate system. Next, the off-boresite angle, γ, is determined. This is the angle between the satellite position vector and the line of sight vector, as illustrated in Figure 8.

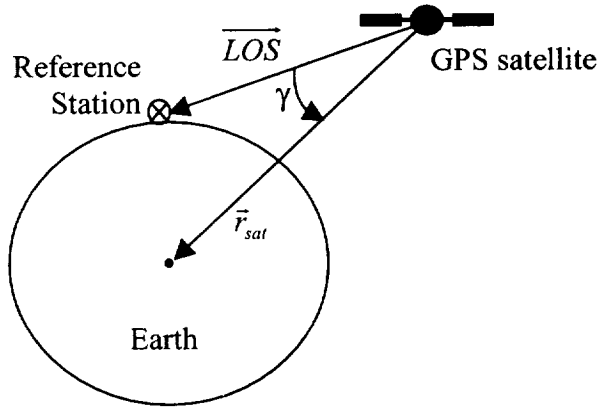

Figure 8. Geometry for Off-Boresite Angle (γ) Calculation

The off-boresite angle is computed as follows:

$$\gamma = \cos^{-1}\left[\frac{\vec{r}_{sat}\cdot\vec{e}}{\|\vec{r}_{sat}\|\cdot\|\vec{e}\|}\right] \tag{34}$$

Once the azimuth of the satellite antenna with respect to the line of sight vector and off-boresite angle are known, the axial ratio can be pulled from a table that gives axial ratio as a function of these two parameters. The antenna azimuths range from 0 to 360 degrees, and the off-boresite angles range from 0 to 15 degrees. The axial ratio values are reported in dB, and the

table is arranged as shown below in Figure 9. The first column of axial ratios applies to azimuths greater than or equal to zero, but less than ten degrees. The second column applies to azimuths greater than or equal to ten degrees, but less than twenty, and so on. Similarly, the first row applies to off-boresite angles greater than or equal to zero, but less than one degree.

Satellite Antenna Azimuths →

Off-Boresite Angles ↓

	0	**10**	**20**	. . .	**350**
0	1.2359	1.2359	1.2359	. . .	1.2359
1	1.2388	1.2388	1.2359	. . .	1.2388
2	1.2388	1.2388	1.2388	. . .	1.2417
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
15	1.2274	1.2331	1.2331	. . .	1.2218

Figure 9. Organization of Axial Ratio Table

4.4 Display Time Histories of Axial Ratios

Axial ratios are determined for each PRN at each time in the specified duration. The script *plot_axial_ratio* creates a time history of the axial ratios of each PRN. A separate figure is created for each satellite. An example of this plot is shown in Section 5.

5.0 Examples

5.1 Az-El Plots

The figure below is an example of an Az-El plot. It shows the satellite paths in black, and an associated PRN number at the beginning of each trace. The red arcs represent the mask table. Notice that there are no satellite paths at elevations below those "cut off" by the red arcs. Finally, the superimposed blue arrows represent the ideal yaw angles of each satellite. The yaw angles are to be measured positive clockwise from the top of the page. For example, PRN 4 has almost zero yaw angle for the duration of its pass while PRN 5 has a yaw angle of about –90 degrees over its path.

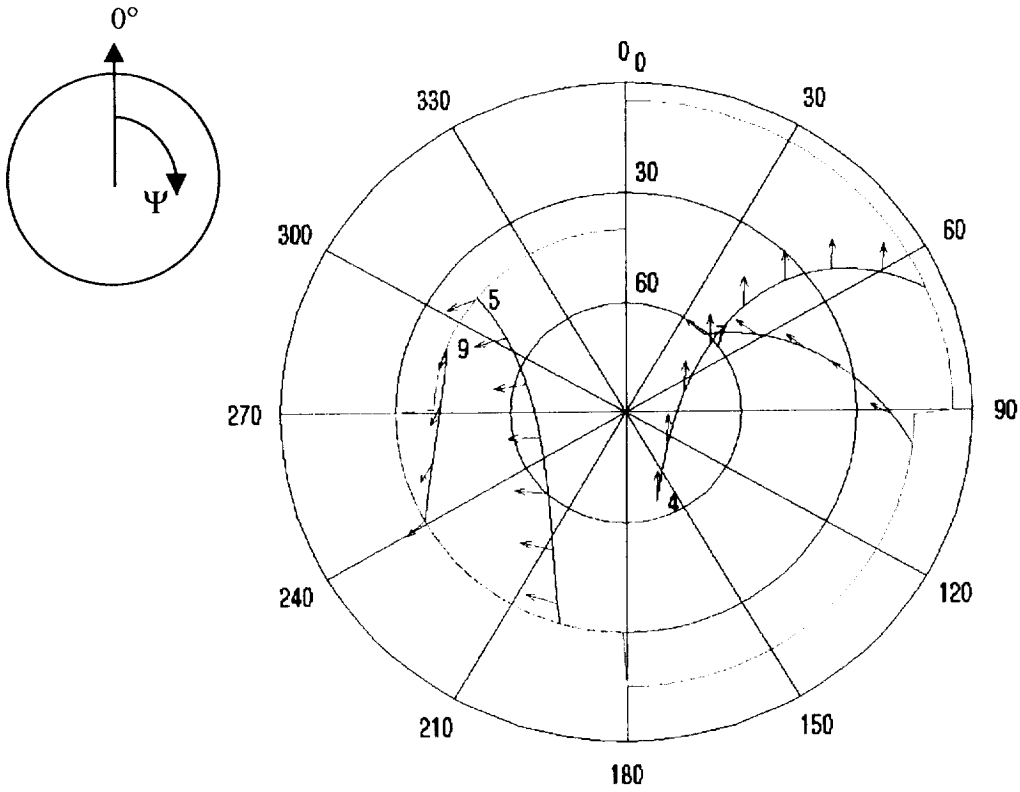

Figure 10. Azimuth-Elevation Plot of Visible Satellites: January 31, 2002 20:42:00 – 26:42:00

5.2 Time Histories of HDOP, VDOP, and GDOP

Figure 11 is an example of a plot that shows time histories of HDOP, VDOP, and GDOP.

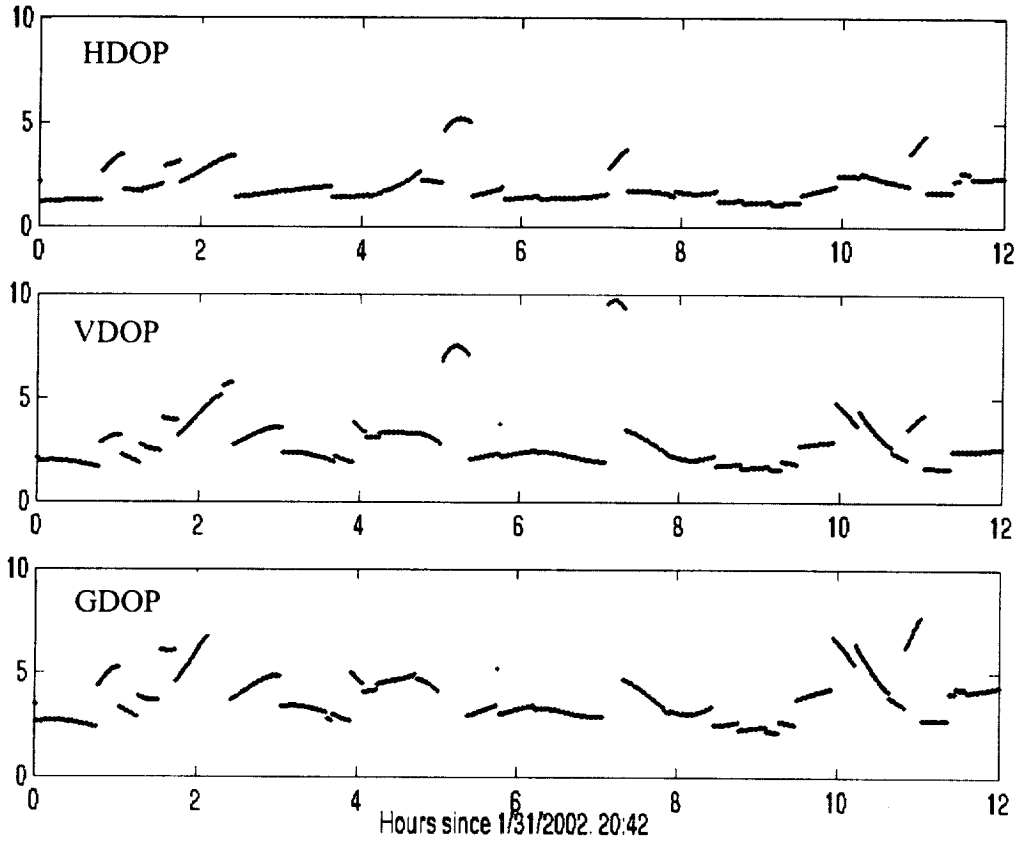

Figure 11. Variation of HDOP, VDOP, and GDOP for Visible Satellites

5.3 Time Histories of Axial Ratios

Figure 12 is an example of a plot that shows the variation of axial ratio, in dB, for PRN 5 over a 12-hour period.

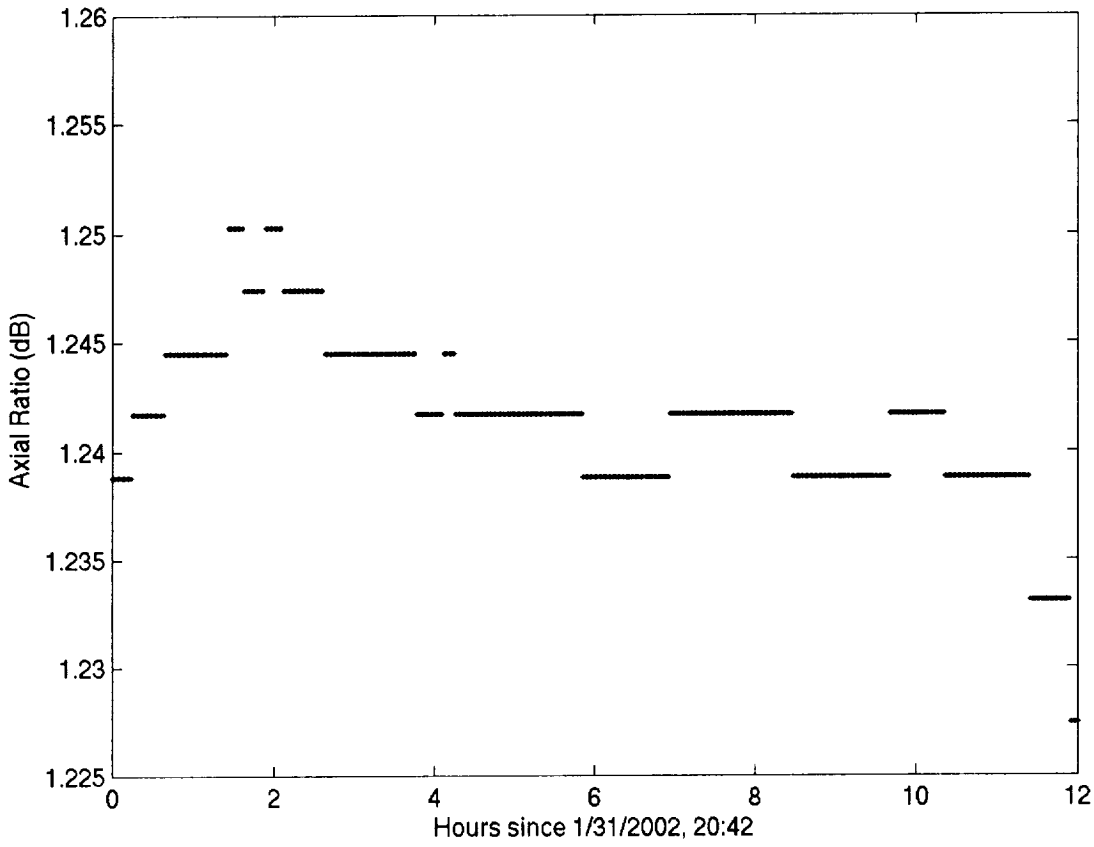

Figure 12. Axial Ratio vs. Time for PRN 5

6.0 References

1. ICD-GPS-200 (1997). *Navstar GPS Space Segment/Navigation User Interfaces*. U.S. Air Force. [(CD) Documents\ICD200C.pdf].

2. Vallado, David A. *Fundamentals of Astrodynamics and Applications, 2nd Edition.* Microcosm Press, El Segundo, California, 2001.

3. Misra, Pratap, and Enge, Per. *Global Positioning System: Signals, Measurements, and Performance.* Ganga-Jamuna Press, Lincoln, Massachusetts, 2001.

4. Anselmi, Joseph A. "GPS Block IIR Yaw Steering." The Aerospace Corporation, July 11, 2001.

5. Bar-Sever, Yoaz E., Bertiger, William I., Davis, Edgar S., and Anselmi, Joseph A. "Fixing the GPS Bad Attitude: Modeling GPS Satellite Yaw During Eclipse Seasons." *Journal of the Institute of Navigation*, vol. 43, no. 1. Spring 1996, pp. 25-39.

```
%*************************************************************************************************************
% Title: yaw_readme.txt
% Author: Lisa Reeh
% Date written: 8 Mar. 2002
% Date last modified: 23 Sept. 2002
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: the purpose of this readme file is to explain how to use the almanac functions and scripts.
%*************************************************************************************************************

This group of functions is used to:
    - read in the YUMA almanac file for the GPS week corresponding to the UTC time entered by the user;
    - compute the ECI positions (in meters) and velocities (im meters/sec) of the selected GPS
      satellite(s);
    - compute the azimuths and elevations of the selected satellite(s) in degrees;
    - determine satellite visibility from a user-defined mask table of cut-off elevations as a function
      of azimuth;
    - calculate the beta and ideal yaw angles for each satellite at each time;
    - compute GDOP, HDOP, VDOP, PDOP, TDOP for all visible satellites, and plot a time history of HDOP,
      VDOP,
      and GDOP;
    - create an AZ-EL plot of the visible satellites, illustrating the cut-off elevations, satellite
      ground
      tracks, and ideal yaw angles.
    - model the axial ratios of the GPS satellites and plot them as a function of time.

Below is a list of the functions/scripts called:

almanac.m:
This script is essentially the 'executable' that runs the entire collection of functions. Simply type
'almanac' at the command prompt to execute. It contains the following subroutines:

    1) load_input.m (script):
        - reads in all input parameters contained in the script 'user_input.m', described below.
        - calculates the GPS time in weeks and seconds corresponding to the UTC time specified in
            'user_input.m'.
        - converts the latitude, longitude, and altitude specified in 'user_input.m' into ECEF coordinates.
        - computes the local up vector at the reference station.
        - computes the Julian Date for the initial time given.
        - calls the script 'user_input' and the functions 'GPS_week' and 'wgslla2xyz', all described below:

        1.a) user_input.m (script):
            Contains the following input parameters:
              Ref_Lat: latitude of the reference station in degrees (North = positive, South = negative)
              Ref_Long: longitude of the reference station in degrees (East = positive, West = negative)
              Ref_Alt: altitude of the reference station above the WGS84 ellipsoid in meters
              Year: The four-digit year
              Month: the number of the month, 1-12
              Date: 1-31
              Hour: hour in military time, 0-23
              Min: 0-59
              Sec: 0-59
              Duration: vector containing the duration of time over which the satellite positions are to
                be calculated, [hour min]
              PRN: vector containing the PRN numbers of the satellites to be used. Entering a '0'
                signifies that all PRNs are to be computed
              max_DOP: maximum DOP value to be plotted in the time history
              AR_table: table listing axial ratios (in dB) as a function of the off-boresite angle and
                the azimuth of the satellite antenna with respect to the line of sight vector

              Elevation mask table -
              The user may specify an azimuth dependent elevation mask as follows:
                mask.az: vector of azimuths in increments no smaller than 1 degree
                mask.el: vector of corresponding cut-off elevations

              Example mask table:
                mask.az = [90, 180, 270, 360]
                mask.el = [10, 30, 40, 50]

                This table sets an elevation cutoff of 10 degrees for azimuths from 0 to 90 degrees.
                For azimuths greater than 90 degrees up to 180 degrees the elevation cut-off i 30 degrees;
                for azimuths greater than 180 up to 270 degrees the elevation cut-off is 40 degrees; and
                for azimuths greater than 270 up to 360 degrees, the elevation cut-off is 50 degrees.

                The azimuth intervals can be irregular at any desired integer degree values.  If the final
                value is not 360, the last value of the azimuth is reset to 360 degrees and the last
```

elevation cut-off is applied up to an azimuth of 360 degrees.

1.b) GPS_week (function):
Converts UTC time to GPS time
Input: Year, Month, Date, Hour, Min, Sec
Output: GPS_wk (1 < GPS_wk < 1024), GPS_sec

1.c) wgslla2xyz (function)
Converts latitude, longitude, and altitude to ECEF x, y, z coordinates
Input: Ref_Lat, Ref_Long, Ref_Alt (degrees, degrees, meters)
Output: Ref_ECEF (ECEF position of reference station in meters)

2) load_almanac.m (script)
- determines the correct YUMA file to open based on the GPS week computed
- reads in the following parameters from the almanac file and stores them in a structure called 'alm_data'
  * alm_data.prn: satellite PRN number
  * alm_data.e: eccentricity
  * alm_data.toa: almanac time of applicability, seconds
  * alm_data.i: orbital inclination, radians
  * alm_data.OMEGAo: right ascension of the ascending node (RAAN), radians
  * alm_data.OMEGAdot: rate of change of RAAN, radians/sec
  * alm_data.a: semi-major axis, meters
  * alm_data.omega: argument of perigee, radians
  * alm_data.M: mean anomaly, radians

3) alm_pos_az_el.m (script)
- This script loops over each selected satellite over the time period specified by 'Duration' in 'user_input.m'. For each satellite at each time, the following parameters are computed:
  * Adjusted Julian Date;
  * Greenwich Sidereal Time (radians);
  * Local up vector at reference station transformed from ECEF to ECI;
  * satellite ECI position (m) and velocity (m/s);
  * Line of Sight (LOS) vector;
  * azimuth angle, in degrees;
  * elevation angle, in degrees;
  * visibility based on a mask table of cut-off elevations as a function of azimuth;
  * beta angle and ideal yaw angle (degrees);
  * GDOP, HDOP, VDOP, PDOP, TDOP.
- Creates a time history plot of GDOP, HDOP, and VDOP.
- Calls the following subroutines, each described below: ecef2eci, alm_pos_vel, el_angle, eci2ecef, az_angle, el_cut_off, beta_yaw_angle

3.a) ecef2eci (function)
Transforms Earth-Centered, Earth-Fixed coordinates to Earth-Centered Inertial coordinates.
Input: ecef_pos (position vector in ECEF coordinates), thetaGST (Greenwich Sidereal Time, radians)
Output: eci_pos (position vector in ECI coordinates)

3.b) alm_pos_vel (function)
Solves Kepler's Equation from the given orbital elements and computes the ECI satellite position in meters and velocity in meters/second.
Input: alm_data (structure containing the fields mentioned above in (2)), time (time at which calculations are to be made, in GPS seconds), Theta (Greenwich Sidereal Time, radians)
Output: ECI position and velocity vectors of satellite, in meters and meters/sec

3.c) el_angle (function)
Calculates the elevation angle of the satellite in degrees
Input: LOS unit vector, Local up vector
Output: elevation angle in degrees

3.d) eci2ecef (function)
Transforms Earth-Centered Inertial coordinates to Earth-Centered, Earth-Fixed coordinates.
Input: eci_pos (position vector in ECEF coordinates), thetaGST (Greenwich Sidereal Time, radians)
Output: ecef_pos (position vector in ECI coordinates)

3.e) ECEF2NED (function)
Converts the ECEF Line of Sight unit vector to NED (North, East, Down) coordinates
Input: Ref_LLA (vector containing the latitude, longitude, altitude of the reference station), LOS_unit_ECEF (LOS unit vector in ECEF coordinates)
Output: los_ned (the LOS vector in NED coordinates)

3.f) az_angle (function)
Computes the azimuth angle of the satellite in degrees

Input: los_ned (LOS unit vector in NED frame)
Output: azimuth angle in degrees

3.g) el_cut_off (function)
Determines the satellite visibility based on the mask table defined by the user in 'user_input.m'. The satellite is visible if its elevation is greater than the cut-off elevation for a particular azimuth.
Input: az (satellite's azimuth, degrees), el (satellite's elevation, degrees), mask (the user-defined mask table)
Output: vis = 1 if satellite is visible, vis = 0 if satellite is not visible.

3.h) beta_yaw_angle (function)
- calculates r_sun_unit (unit vector from the center of the earth to the center of the sun) in ECI;
- computes the orbit normal vector;
- computes beta as the angle between r_sun_unit and the orbit normal;
- computes alpha (angle between r_sun_unit and the satellite position vector, measured in the orbital plane;
- calculates the ideal yaw angle from the alpha and beta angles
- calls the following subroutines: sun_vector, el_angle, get_ideal_yaw

Input: pos_vect (ECI satellite position vector, meters), vel_vect (ECI satellite velocity vector, meters/sec), T (number of elapsed Julian centuries since J2000 epoch), Theta (Greenwich Sidereal Time, radians)
Output: beta and yaw angles, in degrees

3.h.1) sun_vector (function)
Computes the ECI unit sun vector
Input: T (number of elapsed Julian centuries since J2000 epoch), Theta (Greenwich Sidereal Time, radians)
Output: r_sun_unit (sun unit vector in ECI frame)

3.h.2) get_ideal_yaw (function)
Computes the ideal yaw angle of the satellite in degrees
Input: beta (beta angle in radians), alpha (alpha angle in radians)
Output: yaw (ideal yaw angle in degrees)

3.i) eci2ant (function)
converts the ECI line of sight vector to local coordinates with respect to the satellite antenna
Input: yaw (ideal yaw angle in degrees), sat_pos (ECI satellite position in meters), sat_vel (ECI satellite velocity in meters/second), LOS_unit (Line of sight unit vector in the ECI frame in meters)
Output: LOS_ant (Line of sight unit vector in the local antenna frame in meters)

3.j) get_axial_ratio (function)
determines the satellite axial ratio as a function of the azimuth of the satellite antenna with respect to the line of sight vector and the off-boresite angle of the satellite.
Input: sat_az (azimuth of the satellite in degrees), OB_angle (off-boresite angle of the satellite in degrees), sat_az_array (range of possible satellite antenna azimuths, 0 to 359 degrees), OB_angle_array (range of possible off-boresite angles, 0 to 15 degrees), AR_table (table of axial ratios corresponding to each possible combination of satellite antenna azimuths and off-boresite angles)
Output: AR (axial ratio of the satellite)

4) plot_azel (function)
Creates an AZ-EL plot of the visible satellites
Input: az (vector of satellite azimuths), el (vector of satellite elevations), svs (vector of corresponding satellite PRN numbers)
Output: hpol (handle to polar plot)

5) plot_yaw (function)
- When used in conjunction with 'plot_azel', creates a quiver plot of yaw angles for each satellite in 30-minute intervals. Quiver plot is created on top of the existing AZ-EL plot of satellite positions.
- Initially calls 'plot_azel' to create a blank polar plot.

Input: azimuth (vector of satellite azimuths), elevation (vector of satellite elevations), yaw_angle (vector of corresponding yaw angles in degrees
Output: hquiv (handle to quiver plot)
- NOTE: the arrows representing yaw angles in the quiver plot are to be measured positive clockwise from top of page.

5) plot_mask (function)
Creates an AZ-EL plot of the user-defined mask table. Initially calls 'plot_azel' to create a blank polar plot.
Input: az (vector of azimuths), el (vector of cut-off elevations)
Output: hpol (handle to polar plot)

6) plot_axial_ratio (script)
Creates a time history of axial ratios for each PRN at each time. Axial ratios are reported in dB, and a separate plot is produced for each PRN

```
%*******************************************************************************
% Title: almanac
% Author: Lisa Reeh
% Date written: 1 Feb. 2002
% Date modified: 10 Mar. 2002
% Copyright 2002 University of Colorado, Boulder
%
% This program takes input information regarding user position and
% time (UTC), and calculates the ECI positions and velocities for the
% designated satellite(s).
%
% The script 'load_input' reads in the following parameters from the
% input file, 'user_input':
%   latitude (deg), longitude (deg), altitude (m);
%   Year, Month, Date, Hour, Minute, Second, Duration (hours, mins);
%   PRN numbers of satellites to be used.
%   mask table of cut-off elevations as a function of azimuth. Used to
%       check visibility
%   table of axial ratios as a function of off-boresite angle and the azimuth
%       of the satellite antenna with respect to the line of sight vector
%
% 'load_input' then computes the GPS week and GPS seconds corresponding
% to the UTC time given in the input file, and converts the latitude, longitude,
% and altitude into ECEF x, y, z coordinates.
%
% The script 'load_almanac' reads in the data from the designated almanac
% file based on the GPS week number computed
%
% The script 'alm_pos_az_el' is run for each PRN at each time step in the
% specified time duration. It calculates the satellite position and velocity based
% on the orbital data in the almanac file. It also computes the satellite's azimuth
% and elevation angles. Based on information in 'mask.m', satellite visibility
% is determined as a function of azimuth. The satellite's beta and yaw angles are
% also computed.
%
% The function 'plot_azel' creates an az-el plot of the visible satellites
%
% The function 'plot_yaw' creates a quiver plot of each visible satellite's yaw
% angle at 30-minute intervals
%
% The function 'plot_mask' creates an az-el plot of the cut-off elevation angles
% contained in the mask table in the input file
%
% The function 'plot_axial_ratio' creates a time history of axial ratios for each
% PRN at each time.  Axial ratios are reported in dB, and a separate plot is
% produced for each PRN.
%*******************************************************************************

clear all
close all
disp('Loading user input')
load_input

disp('Loading almanac data')
load_almanac

stop_time = GPS_sec + Duration(1) * 3600 + Duration(2) * 60;
dt = 60;     % increment time: 60 seconds
time = GPS_sec:dt:stop_time;

% initialize at zero the matrices that will contain the positions,
% LOS vectors, elevation angles, azimuth angles, and visibilities
% of all satellites for the selected time duration
sat_alm_pos = zeros(length(alm_data),length(time));
LOS_alm = zeros(3*length(alm_data),length(time));
elev_angle_alm = zeros(length(alm_data),length(time));
azimuth_angle_alm = zeros(length(alm_data),length(time));
visible = zeros(length(alm_data),length(time));

disp('Computing satellite azimuth and elevations')
alm_pos_az_el

figure(2);
disp('Creating Az-El plot of visible satellites');
% Create az-el plot and yaw angle plot of selected satellites for all times
for w = 1:size(visible,1)        % 'w' loops over all PRNs (w is the number of rows in 'visible')
```

```
    i = find(visible(w,:));        % 'i' is a vector of indices
    if ~isempty(i)
        plot_azel(azimuth_angle_alm(w,i), elev_angle_alm(w,i), PRN(w));
        hold on;
        plot_yaw(azimuth_angle_alm(w,i), elev_angle_alm(w,i), yaw(w,i));
        hold on;
    end
end

% plot cut-off elevations from mask table
plot_mask(mask.az, mask.el);

% Add plot title containing date, time, and duration
plot_title = sprintf('Az-El Plot of Visible Satellites, %d/%d/%d, %02d:%02d - %02d:%02d UTC\n Yaw Angles ar ↙
e Designated by Arrow Directions', Month, Date, Year,...
                     Hour,Min,Hour + Duration(1),Min + Duration(2));
title(plot_title);
hold off;

% plot axial ratio vs. time for each PRN
plot_axial_ratio;
```

```
%*************************************************************************
% Title: load_input
% Author: Lisa Reeh
% Date written: 1 Feb. 2002
% Date last modified: 10 Mar. 2002
% Copyright 2002 University of Colorado, Boulder
%
% This script reads in the user input from user_input.m and
% converts the time given in UTC to GPS weeks and seconds.
%
% The function 'wgslla2xyz' converts latitude, longitude, and altitude
% to ECEF x, y, z coordinates
%*************************************************************************

user_input    % reads input file

% Check for invalid inputs
if Ref_Lat < -90 | Ref_Lat > 90
    error('Error: Reference Latitude out of range');
elseif Ref_Long < -180 | Ref_Long > 180
    error('Error: Reference Longitude out of range');
elseif Year < 100
    error('Error: Reference Year must contain 4 digits');
elseif Month < 1 | Month > 12
    error('Error: Invalid entry for Reference Month');
elseif Date < 1 | Date > 31
    error('Error: Invalid entry for Reference Date');
elseif Hour < 0 | Hour > 23
    error('Error: Invalid entry for Reference Month');
elseif Min < 0 | Min > 59
    error('Error: Invalid entry for Reference Minute');
elseif Sec < 0 | Sec > 59
    error('Error: Invalid entry for Reference Second');
elseif Duration(1) < 0
    error('Error: Invalid entry for hours of Duration');
elseif Duration(2) < 0
    error('Error: Invalid entry for minutes of Duration');
end

[GPS_wk GPS_sec] = GPS_week(Year, Month, Date, Hour, Min, Sec);

Ref_LLA = [Ref_Lat; Ref_Long; Ref_Alt];     % deg, m
Ref_ECEF= wgslla2xyz(Ref_LLA);      % Ref_ECEF is a column vector, meters

% Find local up (unit vector) at reference station.
Ref_ECEF_up = [cos(Ref_LLA(1)*pi/180)*cos(Ref_LLA(2)*pi/180); ...
      cos(Ref_LLA(1)*pi/180)*sin(Ref_LLA(2)*pi/180); sin(Ref_LLA(1)*pi/180)];

% calculate the Julian Date from the reference starting time
% Reference: Vallado, pg.67
JD_init = 367*Year - floor(7*(Year + floor((Month+9)/12))/4) +...
    floor(275*Month/9) + Date + 1721013.5 + (((Sec/60 + Min)/60) + Hour)/24;
```

```
%*********************************************************************
% Title: user_input
% Author: Lisa Reeh
% Date written: 1 Feb. 2002
% Date last modified: 19 Sept. 2002
% Copyright 2002 University of Colorado, Boulder
%
% This file contains the input parameters for the function 'load_input.m'
% Boulder, CO: 40N, 105W, 1631 m altitude
% Time is given in UTC
%*********************************************************************

Ref_Lat = 40;           % latitude in degrees (North - positive; South - negative)
Ref_Long = -105;          % longitude in degrees (East - positive; West - negative)
Ref_Alt = 1631;          % altitude above the WGS 84 ellipsoid in meters
Year = 2002;
Month = 1;              % 1 - 12
Date = 31;               % 1 - 31
Hour = 20;             % hours in military time, 0 - 23
Min = 42;                % 0 - 59
Sec = 0;                % 0 - 59
Duration = [12,0];      % duration of time over which positions of satellites is to be determined: [hours mi ↙
nutes]
PRN = [4, 5, 7, 9];         % satellite PRNs used; enter 0 to calculate for all PRN's

% Set up 'mask' table as a structure array with components 'az' and 'el'
% To set up the mask table:
%   Azimuths must range from 0 to 360 degrees
%   Elevations must be within the range from -90 to 90 degrees
%   The first azimuth element, e.g. 10, covers azimuths from 0 - 10 degrees
%   The second azimuth element, e.g. 20, covers azimuths > 10 and <= 20 degrees
%   etc....
%   Therefore, the cut-off elevation corresponding to the first azimuth element
%   is constant for that range of azimuths

mask.az = [90, 180, 270, 360];
mask.el= [5, 5, 5, 5]

% Final mask angle must be 360 deg
mask.az(length(mask.az))=360;

% Specify maximum value of DOPs to plot
max_DOP = 10;

% Set up table of axial ratios as a function of off-boresite angle and satellite antenna azimuth
% Satellite azimuths (increasing from left to right) range from 0 to 359 degrees.
% Off-boresite angles (increasing from top to bottom) range from 0 to 15 degrees.
sat_az_array = 0:10:350;
OB_angle_array = 0:1:15;

AR_table = [1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359...
        1.2359   1.2359   1.2359  1.2359   1.2359    1.2359   1.2359   1.2359   1.2359   1.2359...
        1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359...
        1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359;
1.2388   1.2388   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359   1.2359...
    1.2359   1.2359  1.2359   1.2359    1.2359   1.2359   1.2359   1.2359   1.2388   1.2388...
    1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388...
    1.2388   1.2388   1.2388   1.2388   1.2388;
1.2388   1.2388   1.2388   1.2388   1.2359   1.2359   1.2359   1.2331   1.2331   1.2331   1.2331...
    1.2331   1.2359  1.2359   1.2359    1.2388   1.2388   1.2388   1.2388   1.2417   1.2388...
    1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388   1.2388...
    1.2417   1.2417   1.2417   1.2417   1.2417;
1.2417   1.2417   1.2388   1.2388   1.2359   1.2359   1.2331   1.2331   1.2331   1.2331   1.2331...
    1.2331   1.2359  1.2359   1.2388    1.2417   1.2445   1.2445   1.2445   1.2445   1.2445...
    1.2445   1.2417   1.2417   1.2388   1.2388   1.2388   1.2388   1.2388   1.2417   1.2417...
    1.2417   1.2417   1.2445   1.2445   1.2417;
1.2445   1.2417   1.2417   1.2388   1.2359   1.2359   1.2331   1.2331   1.2303   1.2303   1.2331...
    1.2331   1.2359  1.2388   1.2417    1.2445   1.2474   1.2503   1.2503   1.2503   1.2474...
    1.2474   1.2445   1.2417   1.2388   1.2388   1.2388   1.2388   1.2388   1.2417   1.2417...
    1.2445   1.2445   1.2445   1.2445   1.2445;
1.2445   1.2417   1.2417   1.2388   1.2359   1.2331   1.2331   1.2303   1.2303   1.2303   1.2303...
    1.2331   1.2359  1.2388   1.2445    1.2503   1.2531   1.2531   1.2503   1.2503   1.2503...
    1.2503   1.2474   1.2417   1.2388   1.2388   1.2359   1.2388   1.2388   1.2417   1.2445...
    1.2474   1.2474   1.2474   1.2474   1.2445;
1.2445   1.2445   1.2417   1.2388   1.2359   1.2331   1.2303   1.2303   1.2303   1.2303   1.2303...
    1.2331   1.2359  1.2388   1.2474    1.2531   1.2531   1.2474   1.2445   1.2445   1.2474...
```

```
    1.2503   1.2503   1.2445   1.2388   1.2359   1.2359   1.2359   1.2388   1.2417   1.2474...
    1.2503   1.2503   1.2503   1.2474   1.2474;
1.2474   1.2445   1.2417   1.2388   1.2331   1.2331   1.2303   1.2303   1.2303   1.2303   1.2303...
    1.2331   1.2359  1.2388   1.2474    1.2560   1.2503   1.2445   1.2388   1.2388   1.2445...
    1.2503   1.2503   1.2445   1.2388   1.2359   1.2359   1.2359   1.2388   1.2445   1.2474...
    1.2531   1.2531   1.2503   1.2474   1.2474;
1.2474   1.2445   1.2417   1.2388   1.2331   1.2303   1.2303   1.2303   1.2303   1.2303   1.2331...
    1.2331   1.2331  1.2388   1.2474    1.2560   1.2474   1.2388   1.2359   1.2359   1.2417...
    1.2474   1.2531   1.2445   1.2388   1.2331   1.2331   1.2359   1.2388   1.2445   1.2503...
    1.2560   1.2560   1.2503   1.2474   1.2474;
1.2445   1.2445   1.2417   1.2388   1.2331   1.2303   1.2303   1.2303   1.2303   1.2331   1.2331...
    1.2303   1.2331  1.2359   1.2474    1.2560   1.2445   1.2359   1.2331   1.2331   1.2388...
    1.2474   1.2531   1.2445   1.2359   1.2331   1.2331   1.2359   1.2388   1.2445   1.2503...
    1.2560   1.2531   1.2503   1.2474   1.2445;
1.2445   1.2445   1.2417   1.2388   1.2331   1.2303   1.2274   1.2274   1.2303   1.2331   1.2331...
    1.2303   1.2303  1.2359   1.2445    1.2531   1.2417   1.2331   1.2303   1.2331   1.2388...
    1.2503   1.2560   1.2417   1.2331   1.2303   1.2303   1.2359   1.2388   1.2445   1.2503...
    1.2531   1.2503   1.2474   1.2445   1.2445;
1.2445   1.2445   1.2417   1.2388   1.2331   1.2303   1.2274   1.2274   1.2303   1.2331   1.2331...
    1.2303   1.2303  1.2359   1.2445    1.2503   1.2388   1.2331   1.2331   1.2331   1.2388...
    1.2503   1.2531   1.2417   1.2331   1.2274   1.2303   1.2331   1.2388   1.2445   1.2474...
    1.2503   1.2474   1.2445   1.2445   1.2445;
1.2417   1.2417   1.2417   1.2388   1.2331   1.2274   1.2246   1.2246   1.2274   1.2303   1.2303...
    1.2303   1.2303  1.2359   1.2445    1.2474   1.2388   1.2331   1.2331   1.2359   1.2417...
    1.2503   1.2531   1.2388   1.2303   1.2274   1.2274   1.2331   1.2359   1.2417   1.2445...
    1.2474   1.2445   1.2417   1.2417   1.2417;
1.2388   1.2417   1.2388   1.2388   1.2331   1.2274   1.2218   1.2190   1.2246   1.2303   1.2303...
    1.2274   1.2274  1.2359   1.2445    1.2445   1.2388   1.2331   1.2359   1.2388   1.2417...
    1.2474   1.2474   1.2359   1.2246   1.2246   1.2274   1.2303   1.2359   1.2388   1.2417...
    1.2445   1.2417   1.2388   1.2359   1.2388;
1.2359   1.2359   1.2359   1.2359   1.2359   1.2274   1.2162   1.2162   1.2218   1.2274   1.2274...
    1.2246   1.2274  1.2331   1.2445    1.2445   1.2388   1.2359   1.2388   1.2388   1.2388...
    1.2417   1.2445   1.2331   1.2218   1.2190   1.2246   1.2274   1.2303   1.2331   1.2388...
    1.2417   1.2359   1.2303   1.2274   1.2303;
1.2274   1.2331   1.2331   1.2359   1.2359   1.2246   1.2106   1.2106   1.2162   1.2218   1.2218...
    1.2218   1.2246  1.2331   1.2445    1.2445   1.2359   1.2359   1.2388   1.2359   1.2331...
    1.2388   1.2417   1.2274   1.2190   1.2162   1.2190   1.2218   1.2246   1.2303   1.2359...
    1.2359   1.2303   1.2218   1.2190   1.2218];
```

```
%*******************************************************************************************
% Title: GPS_week
% Originally distributed by Kristine Larson with no guarantees
% Date written: ?
% Date last modified: 5 Feb. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% This function finds GPS week and GPS second of the week.
% Inputs are: Year (4 digits), mo, day, hrs (in military time), minutes, seconds.
% Outputs are: GPS week, and GPS seconds of the week.
%
% NOTE: This function uses the "mod" function in Matlab so that the GPS week number
% never exceeds 1023; this is required for reading in the correct almanac file.
%*******************************************************************************************

function [GPS_wk, GPS_sec_wk] = GPS_week(Y,M,D,H,min,sec)

format long g;
if nargin==4
    min=0;
    sec=0;
end;

UT = H+(min/60)+(sec/3600);
if M > 2
    y=Y;
    m=M;
else
    y=Y-1;
    m=M+12;
end;

JD = fix(365.25*y) + fix(30.6001*(m+1)) + D + (UT/24) + 1720981.5;
GPS_wk = fix((JD-2444244.5)/7);
GPS_sec_wk = round( ( ((JD-2444244.5)/7)-GPS_wk)*7*24*3600);

% Ensure that 1 < GPS_wk < 1024
GPS_wk = mod(fix((JD-2444244.5)/7), 1024);
```

```
%*********************************************************************
% Title: wgslla2xyz
% Source: Misra, Pratap, and Enge, Per. "Global Positioning System:
%       Signals, Measurements, and Performance,"
%       cdrom: GPS Book/Navigation_utilities
% Reference: Decker, B.L., World Geodetic System, 1984,
%       Defense Mapping Agency Aerospace Center.
% Date last modified: 5 Feb. 2002 by Lisa Reeh
%
% Returns the equivalent WGS84 XYZ coordinates (in meters) for a
% given geodetic latitude "lat" (degrees), longitude "lon"
% (degrees), and altitude above the WGS84 ellipsoid in meters.
%
% Note: N latitude is positive, S latitude is negative,
%       E longitude is positive, W longitude is negative.
%
% NOTE: wgs_xyz and LLA are both column vectors
%*********************************************************************

function wgs_xyz = wgslla2xyz(LLA);

wlat = LLA(1);
wlon = LLA(2);
walt = LLA(3);

A_EARTH = 6378137;
flattening = 1/298.257223563;
NAV_E2 = (2-flattening)*flattening;   % also e^2
deg2rad = pi/180;

slat = sin(wlat*deg2rad);
clat = cos(wlat*deg2rad);
r_n = A_EARTH/sqrt(1 - NAV_E2*slat*slat);

x = (r_n + walt)*clat*cos(wlon*deg2rad);
y = (r_n + walt)*clat*sin(wlon*deg2rad);
z = (r_n*(1 - NAV_E2) + walt)*slat;
wgs_xyz = [x; y; z];

if ((wlat < -90.0) | (wlat > +90.0) | (wlon < -180.0) | (wlon > +360.0))
    error('WGS lat or WGS lon out of range');
end
return
```

```
%*********************************************************************
% Title: load_almanac
% Authors: Lisa Reeh & Andria Bilich
% Date written: 1 Oct. 2001
% Date last modified: 7 Feb. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% This script opens the correct YUMA almanac file based on
% the GPS week calculated from the user-defined UTC time.
% The data is read in and stored in the structure array 'alm_data'
%*********************************************************************

string = ['YUMA', num2str(GPS_wk), '.txt'];
fid = fopen(string);
if fid == -1
    error('Error: Cannot open almanac file');
end

% read first line of the file
line = fgets(fid);

j = 0;
% alm_data.prn = [];

while (line ~= -1)
    lead = line(1);
    switch (lead)
    case 'I'      % PRN number
        temp = sscanf(line,'%*27c%d')';
        j = j+1;
        alm_data(j).prn = temp;
    case 'E'      % eccentricity
        temp = sscanf(line,'%*27c%f')';
        alm_data(j).e = temp;
    case 'T'        % time of applicability, seconds
        temp = sscanf(line,'%*27c%f')';
        alm_data(j).toa = temp;
    case 'O'      % inclination
        temp = sscanf(line,'%*27c%f')';
        alm_data(j).i = temp;
    case 'R'
        if line(2) == 'i'  % Right Ascension of the Ascending Node (RAAN), in radians
            temp2 = sscanf(line,'%*27c%f')';
            alm_data(j).OMEGAo = temp2;
        elseif line(2) == 'a'           % Rate of RAAN, radians per second
            temp = sscanf(line,'%*27c%f')';
            alm_data(j).OMEGAdot = temp;
        end
    case 'S'      % semimajor axis, meters
        temp = sscanf(line,'%*27c%f')';
        alm_data(j).a = temp^2;
    case 'A'      % argument of perigee
        if line(2) == 'r'
            temp = sscanf(line,'%*27c%f')';
            alm_data(j).omega = temp;
        end
    case 'M'      % mean anomaly
        temp = sscanf(line,'%*27c%f')';
        alm_data(j).M = temp;
    end       % end case/switch
    line = fgets(fid);
end      % end while loop
```

```
%*******************************************************************************
% Title: alm_pos_az_el
% Author: Lisa Reeh
% Date written: 1 Feb. 2002
% Date modified: 19 Sept. 2002
%   Revisions made:
% Copyright 2002 University of Colorado, Boulder
%
% This script calculates satellite position, velocity, azimuth angle, and elevation
% angle for the user-defined time duration for the specified PRNs. The azimuths and
% elevations are then compared to a mask table to check satellite visibility. All
% DOPs for all visible satellites are also calculated at each time.
% This script calls the following functions/scripts:
%
% The function 'ecef2eci' is called to transform a vector in the ECEF frame to a
% vector in the ECI frame
%   input: vector in ECEF coordinates, Greenwich Sidereal Time in radians
%   output: vector in ECI coordinates
%
% The function 'alm_pos_vel' is called to solve Kepler's equation and
% compute the satellite's position and velocity
%   input: orbital elements from almanac file, current time, Greenwich Sideral time
%   output: 'sat_pos', the satellite's position vector in ECI (meters),
%           'sat_vel', the satellite's velocity vector in ECI (meters/sec)
%
% The function 'el_angle' is called to calculate the elevation angle of the
% satellite.
%   input: Local up vector at reference station, Line of Sight (LOS) unit vector
%   output: elevation angle of satellite in degrees
%
% The function 'eci2ecef' is called to transform a vector in the ECI frame to a
% vector in the ECEF frame
%   input: vector in ECI coordinates, Greenwich Sidereal Time in radians
%   output: vector in ECEF coordinates
%
% The function 'ECEF2NED' is calle dto transform the LOS unit vector from ECEF
% coordinates to NED (North, East, Down) coordinates.
%   input: Ref_LLA (lat., lon., alt. of reference station), LOS_unit (LOS unit vector)
%   output: LOS vector in NED coordinates
%
% The function 'az_angle' is called to calculate the azimuth angle of the satellite.
%   input: LOS_unit_NED (the line of sight unit vector from the reference station
%                to the satellite in the NED frame)
%   output: azimuth angle in degrees
%
% The function 'el_cut_off' is called to compare the satellite's elevation angle
% to a table of cut-off elevations to determine if the satellite is visible based on
% its azimuth.
%   input: az_angle (azimuth angle of the satellite, 0 to 359 degrees)
%          el_angle (elevation angle of the satellite, -90 to 90 degrees)
%          mask (structure containing cut-off elevations as a function of azimuth)
%   output: visible = 1 if satellite is visible (i.e., sat elev > cut-off elev.)
%           visible = 0 if satellite is not visible
%
% The function 'beta_yaw_angle' calculates the ideal yaw angle of the satellite
% based on the beta and alpha angles. Beta is the angle between the sun vector and the
% orbit plane; alpha is the angle between the sun vector and satellite position vector
% measured in the orbit plane.
%    input: sat_pos (ECI position vector of satellite in meters)
%           sat_vel (ECI velocity vector of satellite in meters/sec)
%           T (number of Julian centuries elapsed since J2000 epoch)
%           Theta_GST (Greenwich Sidereal Time, in radians)
%    output: beta and yaw angles, in degrees
```

```
%
% The function 'eci2ant' converts the ECI line of sight vector to local coordinates
% with respect to the satellite antenna
%    input: yaw - ideal yaw angle, degrees
%           sat_pos (ECI satellite position, in meters)
%           sat_vel (ECI satellite velocity, in meters/second)
%           LOS_unit (Line of sight unit vector, ECI frame, in meters)
%    output: LOS_ant (Line of sight unit vector, local antenna frame, in meters)
%
% The function 'get_axial_ratio' determines the satellite axial ratio as a function
% of the azimuth of the satellite antenna with respect to the line of sight vector
% and the off-boresite angle of the satellite.
%    input: sat_az (azimuth of the satellite, degrees)
%           OB_angle (off-boresite angle of the satellite, degrees)
%           sat_az_array (range of possible satellite azimuths, 0 to 359 degrees)
%           OB_angle_array (range of possible off-boresite angles, 0 to 15 degrees)
%           AR_table (table of axial ratios corresponding to each possible combination
%               of satellite antenna azimuths and off-boresite angles)
%    output: AR (axial ratio of the satellite)
%
% Finally, GDOP, HDOP, VDOP, PDOP, and TDOP are calculated for the visible satellites
% at each time, and a time history is plotted.
%*******************************************************************************

OMEGAdot_earth = 7.2921151467e-5;     % rad/sec

% A '0' in the variable 'PRN' indicates that all PRNs in the almanac file
% are to be calculated
if PRN == 0
    for i = 1:length(alm_data)
        PRN(i) = alm_data(i).prn;
    end
end

PRN_az_el = [ ];

GDOP = NaN * ones(length(time),1);
HDOP = NaN * ones(length(time),1);
VDOP = NaN * ones(length(time),1);
PDOP = NaN * ones(length(time),1);
TDOP = NaN * ones(length(time),1);

JD_adj = JD_init;

for j = 1:length(time)
    m = 1;
    H = [ ];

    % Calculate the adjusted Julian Date
    JD_adj = JD_adj + ( ((dt/60)/60)/24 );

    % Calculate the number of Julian centuries elapsed from the epoch J2000
    T = (JD_adj - 2451545)/36525;

    % Update the Greenwich Sidereal Time in radians
    Theta_GST = rem((67310.54841 + (876600*3600+8640184.812866)*T + 0.093104*T^2 -...
       6.2e-6*T^3), 86400)/240 *pi/180;

    % Quadrant check
    Theta_GST = mod(Theta_GST,2*pi);

    % Convert reference station position from ECEF to ECI frame
    % Ref_ECI is a unit vector in meters
```

```
    Ref_ECI = ecef2eci(Ref_ECEF, Theta_GST);

    % Convert the local up vector to ECI
    Ref_ECI_up = ecef2eci(Ref_ECEF_up, Theta_GST);

    for k = 1:length(PRN)
        sat = find([alm_data.prn] == PRN(k));

        % call function 'alm_pos_vel' to calculate the (ECI) satellite x, y, z - position
        % (m) from reduced set of Keplerian elements
        % 'sat_pos' is a column vector
        [sat_pos, sat_vel] = alm_pos_vel(alm_data(sat),time(j), Theta_GST);

        % first three rows of 'sat_alm_pos' correspond to the x, y, z coordinates
        % of the first PRN at each time during the defined duration.
        % The second PRN will be contained in rows 4-6, etc.
        % Time thus propogates horizontally in 'sat_alm_pos'.
        sat_alm_pos(m:3*k,j) = sat_pos;

        % Calculate the Line of Sight vector (m) from the user-defined reference
        % station to the user-defined satellite and put in unit vector form
        % LOS and LOS_alm are column vectors
        LOS = sat_pos - Ref_ECI;
        LOS_unit = LOS./sqrt(LOS(1)^2 + LOS(2)^2 + LOS(3)^2);
        LOS_alm(m:3*k,j) = LOS_unit;

        % Calculate the satellite's elevation angle.
        elev_angle = el_angle(LOS_unit, Ref_ECI_up);
        elev_angle_alm(k,j) = elev_angle;

        % Convert LOS unit vector to ECEF from ECI so it can be converted to NED
        LOS_unit_ECEF = eci2ecef(LOS_unit, Theta_GST);

        % Calculate the satellite's azimuth in degrees (0<azimuth<360)
        LOS_unit_NED = ECEF2NED(Ref_LLA, LOS_unit_ECEF);
        azimuth_angle = az_angle(LOS_unit_NED);
        azimuth_angle_alm(k,j) = azimuth_angle;

        % check visibility of satellite
        visible(k,j) = el_cut_off(azimuth_angle, elev_angle, mask);

        % calculate the beta angle and ideal yaw angle of the satellite in degrees
        [beta(k,j), yaw(k,j)] = beta_yaw_angle(sat_pos, sat_vel, T, Theta_GST);

        % calculate unit vector of satellite position
        sat_pos_unit = sat_pos/norm(sat_pos);

        % Transform the LOS vector to local coordinates with respect to the satellite antenna.
        [LOS_ant] = eci2ant(yaw(k,j), sat_pos, sat_vel, LOS_unit);

        % Compute the azimuth of the satellite antenna in degrees from 0 to 359
        sat_az(k,j) = (atan2(LOS_ant(3), LOS_ant(2))) * 180/pi;
        if sat_az(k,j) < 0
            sat_az(k,j) = sat_az(k,j) + 360;
        end

        % reverse directions of sat_pos_unit and LOS_unit so that they point away
        % from the satellite towards the earth
        sat_pos_unit = (-1).*sat_pos_unit;
        LOS_unit = (-1).*LOS_unit;

        % determine the off-boresite angle in degrees
        OB_angle(k,j) = (acos(dot(sat_pos_unit, LOS_unit))) * 180/pi;
```

```
        % Determine axial ratio
        axial_ratio(k,j) = get_axial_ratio(sat_az(k,j), OB_angle(k,j), sat_az_array, OB_angle_array
, AR_table);

        % If satellite is visible, add its LOS unit vector to the H matrix
        if visible(k,j) == 1
           LOS_unit_one = [LOS_unit_NED; 1];
           H = [H; LOS_unit_one'];

           % Create matrix containing PRNs, azimuth angles, and elevation angles
           % of visible satellites
           PRN_az_el = [PRN_az_el; PRN(k), azimuth_angle, elev_angle];
        end

          m = m + 3;
      end     % next PRN

      if size(H,1) >= 4
          P = inv(H'*H);
          GDOP(j) = sqrt(trace(P));
          HDOP(j) = sqrt(P(1,1) + P(2,2));
          VDOP(j) = sqrt(P(3,3));
          PDOP(j) = sqrt(P(1,1) + P(2,2) + P(3,3));
          TDOP(j) = sqrt(P(4,4));
      end

end      % next time

% Set up x-axis to plot in hours since start time
plot_time = ( time - GPS_sec*ones(1,length(time)) )./60./60;;

% Plot time history of HDOP, VDOP, and GDOP
x_axis_label = sprintf('Hours since %d/%d/%d, %02d:%02d', Month, Date, Year, Hour, Min);

% Plot time history of HDOP, VDOP, and GDOP
figure(1);
subplot(311);
plot(plot_time, HDOP, '.');
ylim([0 max_DOP]);
title('Variation of HDOP Over Time for Visible Satellites');
xlabel(x_axis_label);

subplot(312);
plot(plot_time, VDOP, '.');
ylim([0 max_DOP]);
title('Variation of VDOP Over Time for Visible Satellites');
xlabel(x_axis_label);

subplot(313);
plot(plot_time, GDOP, '.');
ylim([0 max_DOP]);
title('Variation of GDOP Over Time for Visible Satellites');
xlabel(x_axis_label);
```

```
%*************************************************************
% Title: ecef2eci
% Author: Lisa Reeh
% Date written: 26 Feb. 2002
% Date last modified: 5 Mar. 2002
% Copyright 2002 University of Colorado, Boulder
%
% Reference: Vallado, Fundamentals of Astrodynamics, pp. 49,53
%
% Purpose: translates earth-centered, earth-fixed coordinates
% into earth-centered inertial position given Greenwich
% Siderial Time
%*************************************************************

function [eci_pos] = ecef2eci(ecef_pos, thetaGST)

rot3_Ntheta = [cos(-thetaGST), sin(-thetaGST), 0; ...
               -sin(-thetaGST), cos(-thetaGST), 0; ...
               0,               0,              1];
eci_pos = rot3_Ntheta*ecef_pos;
```

```
%*************************************************************************
% Title: alm_pos_vel
% Authors: Lisa Reeh & Andria Bilich
% Date written: 1 Oct. 2001
% Date last modified: 10 Mar. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: to compute ECI satellite position and velocity when given
% Keplerian orbital elements from almanac file.
% Assumes all angles are in radians, and all distances are in meters
%
% 'alm_data' is a 1x1 struct with the follwing fields:
%       prn, e (eccentricity), toa (almanac time of applicability),
%       i (inclination), OMEGAo (right ascension of ascending node - RAAN)
%       OMEGAdot (rate of change of RAAN), a (semi-major axis), omega
%       (argument of perigee), M (mean anomaly).
%*************************************************************************

function [position,velocity] = alm_pos_vel(alm_data, time, Theta)

mu = 3.986005e14;            % m3/sec2
OMEGAdot_earth = 7.2921151467e-5;     % rad/sec

n = sqrt(mu/alm_data.a^3);
t = time - alm_data.toa;
M_new = alm_data.M + n*t;

% make sure M is in the correct quadrant
M_new = mod(M_new,2*pi);

Eold = M_new;
correction = 1;
tolerance = 10e-12;
while correction > tolerance
    % Solve for the eccentric anomaly using Newton's iteration scheme
    Enew = Eold - (Eold - M_new - alm_data.e*sin(Eold))/(1-alm_data.e*cos(Eold));
    correction = abs(Enew - Eold);
    Eold = Enew;
end
E = Eold;

% find true anomaly
nu = atan2( ((sqrt(1-alm_data.e^2)*sin(E))/(1-alm_data.e*cos(E))),...
    (cos(E)-alm_data.e)/(1-alm_data.e*cos(E)) );
if nu < 0
   nu = nu + 2*pi;
elseif nu > 2*pi
   nu = nu - 2*pi;
end

PHI = nu + alm_data.omega;  % argument of latitude
OMEGA = mod(alm_data.OMEGAo + (alm_data.OMEGAdot - OMEGAdot_earth) * t...
    - OMEGAdot_earth * alm_data.toa,2*pi);

p = alm_data.a*(1-alm_data.e^2);
r = p/(1+alm_data.e*cos(nu));
h = sqrt(mu*p);

% position in the orbital plane:
x_prime = r*cos(PHI);
y_prime = r*sin(PHI);

OMEGA = OMEGA + Theta;

% satellite position in ECI coordinates (in meters)
x = (x_prime*cos(OMEGA) - y_prime*cos(alm_data.i)*sin(OMEGA));
y = (x_prime*sin(OMEGA) + y_prime*cos(alm_data.i)*cos(OMEGA));
z = (y_prime*sin(alm_data.i));

position = [x; y; z];

% Compute the ECI velocity, from Prussing & Conway, pg. 54
vx = -mu/h * (cos(OMEGA) * (sin(PHI) + alm_data.e*sin(alm_data.omega)) +...
    sin(OMEGA) * cos(alm_data.i) * (cos(PHI) + alm_data.e*cos(alm_data.omega)));
vy = -mu/h * (sin(OMEGA) * (sin(PHI) + alm_data.e*sin(alm_data.omega)) -...
    cos(OMEGA) * cos(alm_data.i) * (cos(PHI) + alm_data.e*cos(alm_data.omega)));
```

```
vz = mu/h * sin(alm_data.i) * (cos(PHI) + alm_data.e*cos(alm_data.omega));

velocity= [vx; vy; vz];
```

```
%*************************************************************************
% Title: el_angle
% Author: Andria Bilich
% Date written: 30 Sept. 2001
% Date last modified: 5 Feb. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: To calculate the complement of the angle between two unit
% vectors via dot product:  a.b = ||a||*||b||*cos(angle between vectors)
%*************************************************************************

function [elev_angle] = el_angle(vectA, vectB)

mag_vectA = sqrt(vectA(1)^2 + vectA(2)^2 + vectA(3)^2);
mag_vectB = sqrt(vectB(1)^2 + vectB(2)^2 + vectB(3)^2);

AdotB = dot(vectA,vectB);
elev_angle = asin(AdotB/(mag_vectA*mag_vectB))*180/pi;

return;
```

```
%*****************************************************************
% Title: eci2ecef
% Author: Lisa Reeh
% Date written: Feb. 11, 2002
% Date last modified: Feb. 11, 2002
% Copyright 2002 University of Colorado, Boulder
%
% Reference: Vallado, Fundamentals of Astrodynamics, pp. 49,53
%
% Converts vector in ECI frame to vector in ECEF frame,
% given Greenwich Sidereal Time in degrees
%*****************************************************************

function [ecef_pos] = eci2ecef(eci_pos, thetaGST)

rot_matrix = [cos(thetaGST), sin(thetaGST), 0; -sin(thetaGST), cos(thetaGST), 0; 0,0,1];

ecef_pos = rot_matrix * eci_pos;
```

```
%*************************************************************************
% Author: Michael Armatys
% Date written: 14 Sept. 1998
% Date last modified: 5 Feb. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: Transforms line-of-sight vector from ECEF frame to NED frame.
%
% Inputs:
%    Ref_LLA - Reference Position in lat, lon, alt (deg, m)
%    los - Line of sight vector to convert
%
% Outputs:
%    los_NED - Vector in NED frame
%*************************************************************************
function [los_NED] = ECEF2NED(Ref_LLA,los);

clat = cos(Ref_LLA(1)*pi/180);
slat = sin(Ref_LLA(1)*pi/180);
clon = cos(Ref_LLA(2)*pi/180);
slon = sin(Ref_LLA(2)*pi/180);

% Transformation matrix for ECEF to NED
C = [-clon*slat, -slon*slat, clat; -slon, clon, 0; -clon*clat, -slon*clat, -slat];

los_NED = C*los;
```

```
%*************************************************************************
% Title: az_angle
% Author: Michael Moreau
% Date written: ?
% Date last modified: 12 Mar. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% Computes azimuth of GPS satellites with respect to local
% North direction in a NED frame at the current ECEF position
%
% Inputs:
%    los_ned: Line of Sight unit vector, NED frame
%
% Output:
%    azimuth: 0 < azimuth < 360 degrees
%*************************************************************************

function [azimuth] = az_angle(los_ned)

% projection of los in local horizonal plane (north and east)
los_ned_proj = [los_ned(1);los_ned(2);0];
los_proj_mag = sqrt(los_ned_proj(1)^2 + los_ned_proj(2)^2 + los_ned_proj(3)^2);

% Compute azimuth as angle between los_ned_proj and north unit vector
az_mag = acos( (dot (los_ned_proj,[1;0;0]))/los_proj_mag );

% Sign of azimuth should be equal to the sign of the east component
% Positive for clockwise from north, negative for counter-clockwise from north
% azimuth is in degrees
azimuth = sign(los_ned_proj(2)) * az_mag * 180/pi;

% Convert azimuth angle to 0 < az < 360 degrees
if azimuth < 0
    azimuth = azimuth + 360;
end
```

```
%*************************************************************************************************
% Title: el_cut_off
% Author: Lisa Reeh
% Date written: 1 Feb. 2002
% Date last modified: 7 Feb. 2002
% Copyright 2002 University of Colorado, Boulder
%
% Determines satellite visibility based on cut-off elevation angles according to azimuth angle.
% 'mask.az' contains azimuth angles 0 - 360 degrees
% 'mask.el' contains the elevation below which terrain would block satellite visibility
%
% Inputs:
%       az: azimuth angle of satellite, in degrees
%       el: elevation angle of satellite, in degrees
% Output:
%       'vis' = 1 if satellite is visible
%       'vis' = 0 if satellite is not visible
%*************************************************************************************************

function vis = el_cut_off(az, el, mask)

% Determine the index of 'mask.az' whose value is greater than the input azimuth
index = min(find(mask.az >= az));

vis = el > mask.el(index);
return;
```

```
%*************************************************************************
% Title: beta_yaw_angle
% Author: Lisa Reeh
% Date written: 8 Feb. 2002
% Date last modified: 5 Mar. 2002
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: calculates the the vector from the center of the earth to the
% sun; the beta angle, which is the angle between the orbit plane of the
% satellite and the sun vector; and the ideal yaw angle
%
% Input:
%       pos_vect: position vector of the satellite (km, ECI)
%       vel_vect: velocity vector of the satellite (km/s, ECI)
%       T: number of elapsed Julian centuries (J2000 epoch)
%       Theta: Greenwich Hour Angle in radians
%
% Output:
%       beta: angle in degrees
%       yaw: ideal yaw angle in degrees
%*************************************************************************

function [beta, yaw] = beta_yaw_angle(pos_vect, vel_vect, T, Theta)

d2r = pi/180;       % convert from degrees to radians

% Compute the earth-sun unit vector (in ECI frame)
r_sun_unit = sun_vector(T, Theta);

% Compute the orbit normal unit vector (in ECI frame)
orb_normal = cross(pos_vect, vel_vect);
orb_norm_unit = orb_normal/norm(orb_normal);

% Compute the beta angle in degrees
beta = el_angle(orb_norm_unit, r_sun_unit);

%*************************************************************************
% compute ideal yaw angle:

% determine alpha, the angle in the satellite's orbit plane between noon and
% the position of the satellite
% "noon" is defined as the direction of the unit sun vector

% convert satellite position vector to a unit vector
pos_unit = pos_vect./norm(pos_vect);

% determine the projection of the sun unit vector onto the orbit plane
h_cross_rsun = cross(orb_norm_unit, r_sun_unit);
rsun_in_plane = cross(h_cross_rsun, orb_norm_unit);

% compute alpha in radians
alpha = atan2( dot(pos_unit,h_cross_rsun), dot(pos_unit,rsun_in_plane) );

% quadrant check: make sure 0 < alpha < 2*pi
if alpha < 0
    alpha = alpha + 2*pi;
end

% calculate yaw angle in degrees
yaw = get_ideal_yaw(alpha, beta*d2r);
```

```
%*****************************************************************
% Title: sun_vector
% Author: Lisa Reeh
% Date written: 14 Feb. 2002
% Date last modified: 14 Feb. 2002
% Copyright 2002 Univesity of Colorado, Boulder
%
% Purpose: computes the unit sun vector in ECI given the number of
% elapsed Julian centuries (J2000 epoch), and the Greenwich hour
% angle
%
% Reference: Vallado, p 183
%*****************************************************************

function r_sun_unit = sun_vector(T, Theta)

d2r = pi/180;
AU = 1.495978e11;     % meters

% determine the mean longitude of the sun in degrees
lambda = mod( (280.4606184 + 36000.77005361 * T), 360 );

% determine the mean anomaly of the sun in degrees
M = mod( (357.5277233 + 35999.05034 * T), 360 );

% Calculate the ecliptic longitude in degrees
lambda_ecl = mod(lambda + 1.914666471 * sin(M*d2r) + 0.019994643 * sin(2*M*d2r), 360);

% approximate the obliquity of the ecliptic
ob_ecl = mod(23.439291 - 0.0130042 * T, 360);

% Compute sun unit vector in ECI frame
r_sun_unit = [cos(lambda_ecl*d2r); cos(ob_ecl*d2r)*sin(lambda_ecl*d2r);...
        sin(ob_ecl*d2r)*sin(lambda_ecl*d2r)];
```

```
%*******************************************************************************
% Title: get_ideal_yaw
% Author: Lisa Reeh
% Date written: 8 March 2002
% Date last modified: 8 March 2002
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: computes ideal yaw angle of satellite in degrees
% Inputs: beta angle- angle between sun vector and orbit plane,measured in radians
%         alpha angle- angle between satellite position vector and sun vector
%           measured in the orbital plane in radians
% Output: yaw angle, in degrees
%*******************************************************************************

function yaw = get_ideal_yaw(alpha, beta)

r2d = 180/pi;

if beta > 0
    yaw = (atan( tan(beta)/sin(alpha) ))*r2d;
    if alpha > pi & alpha <= 2*pi
        yaw = yaw + 180;
    end
elseif beta <= 0
    yaw = mod( (atan( tan(beta)/sin(alpha) ))*r2d, 360);
    if alpha > pi
        yaw = yaw + 180;
    end
end
```

```
%*******************************************************************************
% Title: eci2ant
% Author: Lisa Reeh
% Date written: 16 May 2002
% Date last modified: 16 May 2002
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: convert the LOS ECI vector to local coordinates with
% respect to the satellite antenna.
%
% Input:    yaw - ideal yaw angle, degrees
%           sat_pos - ECI satellite position, in meters
%           sat_vel - ECI satellite velocity, in meters/second
%           LOS_unit - Line of sight unit vector, ECI frame, in meters
% Output:   LOS_ant - Line of sight unit vector, local antenna frame, in meters
%*******************************************************************************

function [LOS_ant] = eci2ant(yaw, sat_pos, sat_vel, LOS_unit)

yaw_rad = yaw*pi/180;

ROT1 = [1, 0, 0; 0, cos(yaw_rad), sin(yaw_rad); 0, -sin(yaw_rad), cos(yaw_rad)];

R_hat = sat_pos./norm(sat_pos);
C_hat = cross(sat_pos, sat_vel)./norm(cross(sat_pos, sat_vel));
I_hat = cross(C_hat, R_hat);

RIC_matrix = [R_hat, I_hat, C_hat];

LOS_ant = ROT1 * RIC_matrix' * LOS_unit;
```

```
%*************************************************************************************************
% Title: get_axial_ratio
% Author: Lisa Reeh
% Date written: 17 May 2002
% Date last modified: 17 May 2002
% Copyright 2002 University of Colorado, Boulder
%
% Purpose: determine the satellite axial ratio as a function of the azimuth of the satellite
% antenna with respect to the line of sight vector and the off-boresite angle of the satellite.
%
% Inputs:
%       sat_az - azimuth of the satellite, degrees
%       OB_angle - off-boresite angle of the satellite, degrees
%       sat_az_array - range of possible satellite azimuths, 0 to 359 degrees
%       OB_angle_array - range of possible off-boresite angles, 0 to 15 degrees
%       AR_table - table of axial ratios corresponding to each possible combination of
%           satellite antenna azimuths and off-boresite angles.
%
% Output:
%       AR - axial ratio of the satellite
%*************************************************************************************************

function AR = get_axial_ratio(sat_az, OB_angle, sat_az_array, OB_angle_array, AR_table)

% Determine the column index of 'AR_table' whose value is greater than the input satellite azimuth
col_index = max(find(sat_az_array <= sat_az));
row_index = max(find(OB_angle_array <= OB_angle));

AR = AR_table(row_index, col_index);
return;
```

```
%**********************************************************************
% Title: plot_azel
% Author:  P. Axelrad
% Date written:
% Date last modified: 5 Feb. 2002 by Lisa Reeh
%
% Creates an az-el plot of satellites
% Inputs:
%       az: vector of azimuth angles, in degrees
%       el: vector of elevation angles, in degrees
%       svs: vector of satellite PRN numbers
% Outputs:
%       hpol: handle to polar plot
%
% NOTE: To avoid printing PRN numbers on the plot, make 'svs' a vector of zeros
%**********************************************************************

function hpol = plot_azel(az,el,svs)

line_style = 'auto';

if nargin < 1
    error('Requires 3 input arguments.')
end

if isstr(az) | isstr(el)
    error('Input arguments must be numeric.');
end
if any(size(az) ~= size(el))
    error('AZ and EL must be the same size.');
end

% get hold state
cax = newplot;
next = lower(get(cax,'NextPlot'));
hold_state = ishold;

% get x-axis text color so grid is in same color
tc = get(cax,'xcolor');

% Hold on to current Text defaults, reset them to the
% Axes' font attributes so tick marks use them.
fAngle  = get(cax, 'DefaultTextFontAngle');
fName   = get(cax, 'DefaultTextFontName');
fSize   = get(cax, 'DefaultTextFontSize');
fWeight = get(cax, 'DefaultTextFontWeight');
set(cax, 'DefaultTextFontAngle',  get(cax, 'FontAngle'), ...
    'DefaultTextFontName',   get(cax, 'FontName'), ...
    'DefaultTextFontSize',   get(cax, 'FontSize'), ...
    'DefaultTextFontWeight', get(cax, 'FontWeight') )

% only do grids if hold is off
if ~hold_state

    % make a radial grid
    hold on;
    hhh=plot([0 2*pi],[0 90],'-','linewidth',0.5);
    v = [get(cax,'xlim') get(cax,'ylim')];
    ticks = length(get(cax,'ytick'));
    delete(hhh);

    % check radial limits and ticks
    rmin = 0; rmax = v(4); rticks = ticks-1;

    if rticks > 5   % see if we can reduce the number
        if rem(rticks,2) == 0
            rticks = rticks/2;
        elseif rem(rticks,3) == 0
            rticks = rticks/3;
        end
    end

    % define a circle
    th = 0:pi/50:2*pi;
    xunit = cos(th);
    yunit = sin(th);
```

```
    % now really force points on x/y axes to lie on them exactly
    inds = [1:(length(th)-1)/4:length(th)];
    xunits(inds(2:2:4)) = zeros(2,1);
    yunits(inds(1:2:5)) = zeros(3,1);

    rinc = (rmax-rmin)/rticks;
    for i=(rmin+rinc):rinc:rmax
        plot(yunit*i,xunit*i,'-','color',tc,'linewidth',0.5);
        text(0,i+rinc/20,[' ' num2str(90-i)],'verticalalignment','bottom' );
    end

    % plot spokes
    th = (1:6)*2*pi/12;
    cst = cos(th); snt = sin(th);
    cs = [cst; -cst];
    sn = [snt; -snt];
    plot(rmax*sn,rmax*cs,'-','color',tc,'linewidth',0.5);

    % annotate spokes in degrees
    rt = 1.1*rmax;
    for i = 1:max(size(th))
        text(rt*snt(i),rt*cst(i),int2str(i*30),'horizontalalignment','center' );
        if i == max(size(th))
            loc = int2str(0);
        else
            loc = int2str(180+i*30);
        end
        text(-rt*snt(i),-rt*cst(i),loc,'horizontalalignment','center' );
    end

    % set viewto 2-D
    view(0,90);
    % set axis limits
    axis(rmax*[-1 1 -1.1 1.1]);
end

% Reset defaults.
set(cax, 'DefaultTextFontAngle', fAngle , ...
    'DefaultTextFontName',   fName , ...
    'DefaultTextFontSize',   fSize, ...
    'DefaultTextFontWeight', fWeight );

% transform data to Cartesian coordinates.
yy = (90-el).*cos(az*pi/180);
xx = (90-el).*sin(az*pi/180);

% plot data on top of grid
q = plot(xx,yy,'*k','MarkerSize',2);

% Place satellite PRN numbers with satellite position
for i = 1:length(svs)
    if(svs(i)~=0)
    text(xx(i)+3,yy(i),int2str(svs(i)));
    end
end

if nargout > 0
    hpol = q;
end

if ~hold_state
    axis('equal');axis('off');
end

% reset hold state
if ~hold_state, set(cax,'NextPlot',next);
end
```

```
%*****************************************************************
% Title: plot_yaw
% Author: Lisa Reeh
% Date written: 5 Mar. 2002
% Date last modified: 10 Mar. 2002
% Copyright 2002 University of Colorado, Boulder
%
% When used in conjunction with 'plot_azel', creates a quiver
% plot of yaw angles for each satellite at each position in
% 30 - minute intervals
%
% Inputs:
%       azimuth: vector of azimuth angles, in degrees
%       elevation: vector of elevation angles, in degrees
%       yaw_angle: vector of corresponding yaw angles in degrees
%
% Outputs:
%       hquiv: handle to quiver plot
%
% NOTE: in the plot, the arrows representing yaw angles are
%       to be measured positive clockwise from top of page
%*****************************************************************

function hquiv = plot_yaw(azimuth, elevation, yaw_angle)

plot_azel(0,90,0), hold on

% only plot yaw angles every 30 minutes
k = 1;
for j = 1:30:length(azimuth)
    az(k) = azimuth(j);
    el(k) = elevation(j);
    yaw(k) = yaw_angle(j);
    k = k + 1;
end

% transform data to Cartesian coordinates.
yy = (90-el).*cos(az*pi/180);
xx = (90-el).*sin(az*pi/180);

% plot yaw angles on top of grid
% scale arrows by factor of 0.3 to make them smaller
hquiv = quiver(xx,yy,sin(yaw*pi/180),cos(yaw*pi/180),0.3);
```

```
%*************************************************************
% Title: plot_mask
% Author: P. Axelrad
% Date written:
% Date last modified: 18 Feb. 2002 by Lisa Reeh
% Copyright 2002 University of Colorado, Boulder
%
% Creates an az-el plot of cut-off elevation angles specified
% in 'mask' struct defined in input file 'user_input'
% Inputs:
%       az: vector of azimuth angles, in degrees
%       el: vector of elevation angles, in degrees
% Outputs:
%       hpol: handle to polar plot
%*************************************************************

function hpol = plot_mask(az,el)

plot_azel(0,90,0),hold on
big_az=[0:360]';
big_el=zeros(361,1);
im=length(az);
ideg=1;big_el(1)=el(1);
for i=1:im
    while (ideg<=az(i))
        ideg=ideg+1;
        big_el(ideg)=el(i);
    end
end

r=90-big_el;
th=-big_az*pi/180+pi/2;
hpol = polar(th,r,'r-');
```

```
%*******************************************************************************************
% Title: plot_axial_ratio
% Author: Lisa Reeh
% Date written: 19 Sept. 2002
% Date modified: 19 Sept. 2002
%   Revisions made:
% Copyright 2002 University of Colorado, Boulder
%
% This script plots the variation of axial ratio (in dB) for each satellite as a
% function of time.
%*******************************************************************************************

for i = 1:length(PRN)
    figure(i+2);
    plot(plot_time, axial_ratio(i,:), '.');
    plot_title = sprintf(['Axial Ratio vs. Time for PRN ', num2str(PRN(i))]);
    title(plot_title);
    xlabel(x_axis_label);
    ylabel('Axial Ratio (dB)');
end
```