Need for Precise Orbits

1. proper focus

2. transform from geographic radar coordinates

3. image alignment

4. flattening interferogram

Brute force algorithm given precise orbit subroutine

pick target point

fly satellite along orbit and identify point of closest approach

(This is inefficient but computers are fast so don’t waste time coding.)
low and high orbits

Figure 4. Opposite and above: polar and geostationary orbits for NOAA satellites. Note that the polar orbit rotates one degree per day; this is to make it synchronous with the sun. The geostationary satellite stays continuously above one spot on Earth.
geometry of orbit plane

- S: satellite
- E: Earth
- A: apogee
- P: perigee
- a: semimajor axis
- b: semiminor axis
- e: eccentricity

\[ e^2 = \frac{a^2 - b^2}{a^2} \]
elliptical orbit motion

apogee

perigee
Orbital Geometry

The ideal elliptical orbit is described by 6 Keplerian elements:

- $\alpha$ - true anomaly (instantaneous angle from satellite to perigee)
- $\omega$ - argument of perigee
- $\Omega$ - longitude of node
- $a$ - semimajor axis
- $e$ - eccentricity
- $i$ - inclination
Two ways to describe an orbit with 6 numbers

Kepler elements
lower accuracy > 10 m

Used for making ground tracks for planning and data management

Cartesian state vector
Can be high accuracy ~3 cm for Sentinel-1

The ideal elliptical orbit is described by 6 Keplerian elements:
\[ \alpha \] - true anomaly (instantaneous angle from satellite to perigee)
\[ \omega \] - argument of perigee
\[ \Omega \] - longitude of node
\[ a \] - semimajor axis
\[ e \] - eccentricity
\[ i \] - inclination

Orbital elements are based on precise GPS tracking and orbit dynamic modeling.
low and high orbits

Figure 4. Opposite and above: polar and geostationary orbits for NOAA satellites. Note that the polar orbit rotates one degree per day; this is to make it synchronous with the sun. The geostationary satellite stays continuously above one spot on Earth.
geostationary orbit
orbit period = 1 day
4) Sun-synchronous Orbit

For many remote sensing applications it is important to have the ascending node pass over the equator at the same local time. To create a sun-synchronous orbit, the plane of the orbit must precess in a prograde direction with a period of exactly 1 year.

\[ \omega_n = \frac{2\pi}{(365.25 \times 86400)} = 1.991 \times 10^{-7} \text{ s}^{-1} \]

Remember \( \frac{\omega_n}{\omega_s} = -\frac{3J_2}{2} \left( \frac{a_e}{a} \right)^2 \cos i \left( \frac{1-e^2}{1-e^2} \right)^2 \) so prograde precession requires \( i > 90^\circ \). Thus the orbital inclination is dictated by the orbital altitude. For example, a sun-synchronous orbit with an orbital radius of \( a = 7878 \text{ km} \) (altitude 1500 km) must have an inclination of 102°.

![Graph showing relationship between semimajor axis, altitude, period, and inclination.](image)
low and high orbits

Figure 4. Opposite and above: polar and geostationary orbits for NOAA satellites. Note that the polar orbit rotates one degree per day; this is to make it synchronous with the sun. The geostationary satellite stays continuously above one spot on Earth.
sun synchronous orbit
How to compute precise orbits from elements

Hermite polynomial interpolation

Orbital elements are based on precise GPS tracking and orbit dynamic modeling.

In GMTSAR this is called a *.LED file

---

**PRECISE STATE VECTORS FOR ENVISAT SPACED AT 60 SECOND INTERVALS**

<table>
<thead>
<tr>
<th>TIME</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>DX/DT</th>
<th>DY/DT</th>
<th>DZ/DT</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:55:28.00</td>
<td>+4925775.296</td>
<td>-0672489.270</td>
<td>+5149599.921</td>
<td>-5648.045305</td>
<td>-1230.624128</td>
<td>+5056.89226</td>
</tr>
<tr>
<td>21:56:28.00</td>
<td>+4588023.337</td>
<td>-0743509.493</td>
<td>+5442786.786</td>
<td>-5786.560443</td>
<td>-1135.486760</td>
<td>+4711.22472</td>
</tr>
<tr>
<td>21:58:28.00</td>
<td>+3858712.626</td>
<td>-0867371.390</td>
<td>+5964606.354</td>
<td>-6352.228261</td>
<td>-924.559337</td>
<td>+3966.18750</td>
</tr>
<tr>
<td>21:59:28.00</td>
<td>+3470102.184</td>
<td>-0919436.302</td>
<td>+6192012.706</td>
<td>-6597.087114</td>
<td>-810.020339</td>
<td>+3569.74320</td>
</tr>
<tr>
<td>22:00:28.00</td>
<td>+3067595.664</td>
<td>-0964470.578</td>
<td>+6392152.503</td>
<td>-6815.275024</td>
<td>-690.319610</td>
<td>+3159.39884</td>
</tr>
<tr>
<td>22:01:28.00</td>
<td>+2652820.268</td>
<td>-1002185.442</td>
<td>+6569094.621</td>
<td>-7005.912367</td>
<td>-566.159017</td>
<td>+2736.75872</td>
</tr>
<tr>
<td>22:02:28.00</td>
<td>+2227452.596</td>
<td>-1032334.971</td>
<td>+6720350.620</td>
<td>-7168.232800</td>
<td>-438.264632</td>
<td>+2303.47344</td>
</tr>
<tr>
<td>22:03:28.00</td>
<td>+1793211.746</td>
<td>-1054717.441</td>
<td>+6845332.311</td>
<td>-7301.586442</td>
<td>-3037.382928</td>
<td>+1861.23307</td>
</tr>
<tr>
<td>22:04:28.00</td>
<td>+1351852.262</td>
<td>-1089176.424</td>
<td>+6943553.921</td>
<td>-7405.442085</td>
<td>-174.276681</td>
<td>+1411.76019</td>
</tr>
<tr>
<td>22:05:28.00</td>
<td>+905156.961</td>
<td>-1075601.647</td>
<td>+7014633.841</td>
<td>-7479.388840</td>
<td>-1097.529020</td>
<td>+956.80327</td>
</tr>
</tbody>
</table>

**C-PROGRAM TO INTERPOLATE STATE VECTORS**

```c
#include <stdio.h>

void interpolate_ALOS_orbit(struct ALOS_ORB *orb, double *pt, double *p,
                             double *pv, double time, double *x, double *y, double *z, int *ir)
{
    /* ir; return code */
    /* time; seconds since Jan 1 */
    /* x, y, z; position */
    int k, nval, nd;

    nval = 6; /* number of points to use in interpolation */
    nd = orb->nd;

    if (debug) printf(stderr, "time %ld\n", time, nd);

    /* interpolate for each coordinate direction */
    /* #hermite.c version */
    for (k=0; k<nd; k++) {
        p[k] = orb->points[k].px;
        pv[k] = orb->points[k].vx;
    }

    hermite_c(pt, p, pv, nval, time, x, ir);

    for (k=0; k<nd; k++) {
        p[k] = orb->points[k].py;
        pv[k] = orb->points[k].vy;
    }

    hermite_c(pt, p, pv, nval, time, y, ir);

    for (k=0; k<nd; k++) {
        p[k] = orb->points[k].pz;
        pv[k] = orb->points[k].vz;
    }

    hermite_c(pt, p, pv, nval, time, z, ir);
}
```
2.4 m corner reflectors
D1, D2 installed 1996
A1 installed Nov, 2005
Back Projection

\[ R(s) = R_o + \dot{R}_o (s - s_o) + \frac{\ddot{R}_o}{2} (s - s_o)^2 + \ldots \]

Least-squares fit of range history for each point in DEM provides both the accurate position in range azimuth \([R_o, s_o]\) space and the Dopper centroid and rate parameters needed to focus the image. This analysis only needs to be applied to the master image.
Back Projection

back projection algorithm

① select a point on the DEM (lon, lat, height above WGS84 ellipsoid)
② convert lon, lat, height to x, y, z.
③ use the state vector to “fly” the satellite over the point
④ calculate the minimum distance (range) and time of minimum distance (azimuth)
⑤ do that for every point in the DEM
⑥ keep that transformation table

In GMTSAR the transformation file is called *trans.dat* in the /topo directory.

lon, lat, height, range azimuth
Precise orbit for image focussing and geolocation - ERS, Envisat

ERS 36 acquisitions

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>rms</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>-17.2 m</td>
<td>2.8 m</td>
</tr>
<tr>
<td>azimuth</td>
<td>14.8 m</td>
<td>10.4 m</td>
</tr>
</tbody>
</table>

Envisat 10 acquisitions

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>rms</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>80.6 m</td>
<td>1.9 m</td>
</tr>
<tr>
<td>azimuth</td>
<td>3.6 m</td>
<td>8.8 m</td>
</tr>
</tbody>
</table>
~18,000 man-made objects orbiting the Earth
Conclusions

- All SAR satellites are in low Earth orbit at an altitude of 600-800 km.
- The ground track speed is ~7000 m/s and cannot be adjusted.
- The orbits repeat exactly (Envisat 35 days; ALOS-1 42 days; Sentinel-1 12 days).
- Accurate orbits (< 1 m) greatly simplify InSAR processing. They are used for:
  - proper focus
  - transformation (lon, lat, height) to (range, azimuth) coordinates (back projection)
  - image alignment to sub-pixel accuracy
  - flattening the interferogram (i.e. removing the topographic phase)

In GMTSAR the orbits (state vectors) are stored in *.LED files. The transformation is stored in trans.dat file.
Altitudes of various remote sensing platforms

- 10,000,000 km: geostationary satellites
- 10,000 km: Van Allen belt
- 1,000 km: Ionosphere
- 100 km: Low Earth orbit satellites
- 10 km: Jet aircraft
- 1 km: Light aircraft
- 100 m: UAVs
- 10 m: Tethered balloons, kites
- 10 m: Towers
- 1 m: Hand-held sensors
NOTES ON BB

http://topex.ucsd.edu/rs/orbits.pdf