How to compute planetary positions

By **Paul Schlyter, Stockholm, Sweden**
email: pausch@stjarnhimlen.se or WWW: http://stjarnhimlen.se/

Break out of a frame

Tutorial with numerical test cases
Computing rise and set times

0. Foreword

Below is a description of how to compute the positions for the Sun and Moon and the major planets, as well as for comets and minor planets, from a set of orbital elements.

The algorithms have been simplified as much as possible while still keeping a fairly good accuracy. The accuracy of the computed positions is a fraction of an arc minute for the Sun and the inner planets, about one arc minute for the outer planets, and 1-2 arc minutes for the Moon. If we limit our accuracy demands to this level, one can simplify further by e.g. ignoring the difference between mean, true and apparent positions.

The positions computed below are for the 'equinox of the day', which is suitable for computing rise/set times, but not for plotting the position on a star map drawn for a fixed epoch. In the latter case, correction for precession must be applied, which is most simply performed as a rotation along the ecliptic.

These algortihms were developed by myself back in 1979, based on a preprint from T. van Flandern's and K. Pulkkinen's paper "Low precision formulae for planetary positions", published in the Astrophysical Journal Supplement Series, 1980. It's basically a simplification of these algorithms, while keeping a reasonable accuracy. They were first implemented on a HP-41C programmable pocket calculator, in 1979, and ran in less than 2 KBytes of RAM! Nowadays considerable more accurate algorithms are available of course, as well as more powerful computers. Nevertheless I've retained these algorithms as what I believe is the simplest way to compute solar/lunar positions with an accuracy of 1-2 arc minutes.

1. Introduction

The text below describes how to compute the positions in the sky of the Sun, Moon and the major planets out to Neptune. The algorithm for Pluto is taken from a fourier fit to Pluto's position as computed by numerical integration at JPL. Positions of other celestial bodies as well (i.e. comets and asteroids) can also be computed, if their orbital elements are available.

These formulae may seem complicated, but I believe this is the simplest method to compute planetary positions with the fairly good accuracy of about one arc minute (=1/60 degree). Any further simplifications will yield lower accuracy, but of course that may be ok, depending on the application.

2. A few words about accuracy

The accuracy requirements are modest: a final position with an error of no more than 1-2 arc minutes (one arc minute = 1/60 degree). This accuracy is in one respect quite optimal: it is the highest accuracy one can strive for, while still being able to do many simplifications. The simplifications made here are:

1: Nutation and aberration are both ignored.
2: Planetary aberration (i.e. light travel time) is ignored.
3: The difference between Terrestial Time/Ephemeris Time (TT/ET), and Universal Time (UT) is ignored.
4: Precession is computed in a simplified way, by a simple addition to the ecliptic longitude.
5: Higher-order terms in the planetary orbital elements are ignored. This will give an additional error of up to 2 arc min in 1000 years from now. For the Moon, the error will be larger: 7 arc min 1000 years from now. This error will grow as the square of the time from the present.
6: Most planetary perturbations are ignored. Only the major perturbation terms for the Moon, Jupiter, Saturn, and Uranus, are included. If still lower accuracy is acceptable, these perturbations can be ignored as well.
7: The largest Uranus-Neptune perturbation is accounted for in the orbital elements of these planets. Therefore, the orbital elements of Uranus and Neptune are less accurace, especially in the distant past and future. The elements for these planets should therefore only be used for at most a few centuries into the past and the future.

3. The time scale

The time scale in these formulae are counted in days. Hours, minutes, seconds are expressed as fractions of a day. Day 0.0 occurs at 2000 Jan 0.0 UT (or 1999 Dec 31, 0:00 UT). This "day number" d is computed as follows (y=year, m=month, D=date, UT=UT in hours+decimals):

```
d = 367*y - 7 * ( y + (m+9)/12 ) / 4 + 275*m/9 + D - 730530
```

Note that ALL divisions here should be INTEGER divisions. In Pascal, use "div" instead of "/", in MS-Basic, use "\" instead of "/". In Fortran, C and C++ "/" can be used if both y and m are integers. Finally, include the time of the day, by adding:

```
d = d + UT/24.0        (this is a floating-point division)
```

4. The orbital elements

The primary orbital elements are here denoted as:

```
N = longitude of the ascending node
i = inclination to the ecliptic (plane of the Earth's orbit)
w = argument of perihelion
a = semi-major axis, or mean distance from Sun
e = eccentricity (0=circle, 0-1=ellipse, 1=parabola)
M = mean anomaly (0 at perihelion; increases uniformly with time)
```

Related orbital elements are:

```
w1 = N + w   = longitude of perihelion
L  = M + w1  = mean longitude
q  = a*(1-e) = perihelion distance
Q  = a*(1+e) = aphelion distance
P  = a ^ 1.5 = orbital period (years if a is in AU, astronomical units)
T  = Epoch_of_M - (M(deg)/360_deg) / P  = time of perihelion
v  = true anomaly (angle between position and perihelion)
E  = eccentric anomaly
```

One *Astronomical Unit (AU)* is the Earth's mean distance to the Sun, or 149.6 million km. When closest to the Sun, a planet is in *perihelion*, and when most distant from the Sun it's in *aphelion*. For the Moon, an artificial satellite, or any other body orbiting the Earth, one says *perigee* and *apogee* instead, for the points in orbit least and most distant from Earth.

To describe the position in the orbit, we use three angles: Mean Anomaly, True Anomaly, and Eccentric Anomaly. They are all zero when the planet is in perihelion:
Mean Anomaly (M): This angle increases uniformly over time, by 360 degrees per orbital period. It's zero at perihelion. It's easily computed from the orbital period and the time since last perihelion.
True Anomaly (v): This is the actual angle between the planet and the perihelion, as seen from the central body (in this case the Sun). It increases non-uniformly with time, changing most rapidly at perihelion.
Eccentric Anomaly (E): This is an auxiliary angle used in Kepler's Equation, when computing the True Anomaly from the Mean Anomaly and the orbital eccentricity.
Note that for a circular orbit (eccentricity=0), these three angles are all equal to each other.

Another quantity we will need is ecl, the *obliquity of the ecliptic*, i.e. the "tilt" of the Earth's axis of rotation (currently 23.4 degrees and slowly decreasing). First, compute the "d" of the moment of interest (section 3).

Then, compute the obliquity of the ecliptic:

```
ecl = 23.4393 - 3.563E-7 * d
```

Now compute the orbital elements of the planet of interest. If you want the position of the Sun or the Moon, you only need to compute the orbital elements for the Sun or the Moon. If you want the position of any other planet, you must compute the orbital elements for that planet *and* for the Sun (of course the orbital elements for the Sun are really the orbital elements for the Earth; however it's customary to here pretend that the Sun orbits the Earth). This is necessary to be able to compute the geocentric position of the planet.

Please note that a, the semi-major axis, is given in Earth radii for the Moon, but in Astronomical Units for the Sun and all the planets.

When computing M (and, for the Moon, when computing N and w as well), one will quite often get a result that is larger than 360 degrees, or negative (all angles are here computed in degrees). If negative, add 360 degrees until positive. If larger than 360 degrees, subtract 360 degrees until the value is less than 360 degrees. Note that, in most programming languages, one must then multiply these angles with pi/180 to convert them to radians, before taking the sine or cosine of them.

Orbital elements of the Sun:

```
N = 0.0
i = 0.0
w = 282.9404 + 4.70935E-5 * d
a = 1.000000  (AU)
e = 0.016709 - 1.151E-9 * d
M = 356.0470 + 0.9856002585 * d
```

Orbital elements of the Moon:

```
N = 125.1228 - 0.0529538083 * d
i = 5.1454
w = 318.0634 + 0.1643573223 * d
a = 60.2666  (Earth radii)
e = 0.054900
M = 115.3654 + 13.0649929509 * d
```

Orbital elements of Mercury:

```
N =  48.3313 + 3.24587E-5 * d
i = 7.0047 + 5.00E-8 * d
w =  29.1241 + 1.01444E-5 * d
a = 0.387098  (AU)
e = 0.205635 + 5.59E-10 * d
M = 168.6562 + 4.0923344368 * d
```

Orbital elements of Venus:

```
N =  76.6799 + 2.46590E-5 * d
i = 3.3946 + 2.75E-8 * d
w =  54.8910 + 1.38374E-5 * d
a = 0.723330  (AU)
e = 0.006773 - 1.302E-9 * d
M =  48.0052 + 1.6021302244 * d
```

Orbital elements of Mars:

```
N =  49.5574 + 2.11081E-5 * d
i = 1.8497 - 1.78E-8 * d
w = 286.5016 + 2.92961E-5 * d
a = 1.523688  (AU)
e = 0.093405 + 2.516E-9 * d
M =  18.6021 + 0.5240207766 * d
```

Orbital elements of Jupiter:

```
N = 100.4542 + 2.76854E-5 * d
i = 1.3030 - 1.557E-7 * d
w = 273.8777 + 1.64505E-5 * d
a = 5.20256  (AU)
e = 0.048498 + 4.469E-9 * d
M =  19.8950 + 0.0830853001 * d
```

Orbital elements of Saturn:

```
N = 113.6634 + 2.38980E-5 * d
i = 2.4886 - 1.081E-7 * d
w = 339.3939 + 2.97661E-5 * d
a = 9.55475  (AU)
e = 0.055546 - 9.499E-9 * d
M = 316.9670 + 0.0334442282 * d
```

Orbital elements of Uranus:

```
N =  74.0005 + 1.3978E-5 * d
i = 0.7733 + 1.9E-8 * d
w =  96.6612 + 3.0565E-5 * d
a = 19.18171 - 1.55E-8 * d  (AU)
e = 0.047318 + 7.45E-9 * d
M = 142.5905 + 0.011725806 * d
```

Orbital elements of Neptune:

```
N = 131.7806 + 3.0173E-5 * d
i = 1.7700 - 2.55E-7 * d
w = 272.8461 - 6.027E-6 * d
a = 30.05826 + 3.313E-8 * d  (AU)
e = 0.008606 + 2.15E-9 * d
M = 260.2471 + 0.005995147 * d
```

Please note than the orbital elements of Uranus and Neptune as given here are somewhat less accurate. They include a long period perturbation between Uranus and Neptune. The period of the perturbation is about 4200 years. Therefore, these elements should not be expected to give results within the stated accuracy for more than a few centuries in the past and into the future.

5. The position of the Sun

The position of the Sun is computed just like the position of any other planet, but since the Sun always is moving in the ecliptic, and since the eccentricity of the orbit is quite small, a few simplifications can be made. Therefore, a separate presentation for the Sun is given.

Of course, we're here really computing the position of the Earth in its orbit around the Sun, but since we're viewing the sky from an Earth-centered perspective, we'll pretend that the Sun is in orbit around the Earth instead.

First, compute the eccentric anomaly E from the mean anomaly M and from the eccentricity e (E and M in degrees):

```
E = M + e*(180/pi) * sin(M) * ( 1.0 + e * cos(M) )
```

or (if E and M are expressed in radians):

```
E = M + e * sin(M) * ( 1.0 + e * cos(M) )
```

Note that the formulae for computing E are not exact; however they're accurate enough here.

Then compute the Sun's distance r and its true anomaly v from:

```
xv = r * cos(v) = cos(E) - e
yv = r * sin(v) = sqrt(1.0 - e*e) * sin(E)

v = atan2( yv, xv )
r = sqrt( xv*xv + yv*yv )
```

(note that the r computed here is later used as rs)

atan2() is a function that converts an x,y coordinate pair to the correct angle in all four quadrants. It is available as a library function in Fortran, C and C++. In other languages, one has to write one's own atan2() function. It's not that difficult:

```
atan2( y, x ) = atan(y/x)                  if x positive
atan2( y, x ) = atan(y/x) +- 180 degrees   if x negative
atan2( y, x ) = sign(y) * 90 degrees       if x zero
```

See these links for some code in Basic or Pascal. Fortran and C/C++ already has atan2() as a standard library function.

Now, compute the Sun's true longitude:

```
lonsun = v + w
```

Convert lonsun,r to ecliptic rectangular geocentric coordinates xs,ys:

```
xs = r * cos(lonsun)
ys = r * sin(lonsun)
```

(since the Sun always is in the ecliptic plane, zs is of course zero). xs,ys is the Sun's position in a coordinate system in the plane of the ecliptic. To convert this to equatorial, rectangular, geocentric coordinates, compute:

```
xe = xs
ye = ys * cos(ecl)
ze = ys * sin(ecl)
```

Finally, compute the Sun's Right Ascension (RA) and Declination (Dec):

```
RA  = atan2( ye, xe )
Dec = atan2( ze, sqrt(xe*xe+ye*ye) )
```

5b. The Sidereal Time

Quite often we need a quantity called Sidereal Time. The Local Sideral Time (LST) is simply the RA of your local meridian. The Greenwich Mean Sideral Time (GMST) is the LST at Greenwich. And, finally, the Greenwich Mean Sidereal Time at 0h UT (GMST0) is, as the name says, the GMST at Greenwich Midnight. However, we will here extend the concept of GMST0 a bit, by letting "our" GMST0 be the same as the conventional GMST0 at UT midnight but also let GMST0 be defined at any other time such that GMST0 will increase by 3m51s every 24 hours. Then this formula will be valid at any time:

```
GMST = GMST0 + UT
```

We also need the Sun's mean longitude, Ls, which can be computed from the Sun's v and w as follows:

```
Ls = v + w
```

The GMST0 is easily computed from Ls (divide by 15 if you want GMST0 in hours rather than degrees), GMST is then computed by adding the UT, and finally the LST is computed by adding your local longitude (east longitude is positive, west negative).

Note that "time" is given in hours while "angle" is given in degrees. The two are related to one another due to the Earth's rotation: one hour is here the same as 15 degrees. Before adding or subtracting a "time" and an "angle", be sure to convert them to the same unit, e.g. degrees by multiplying the hours by 15 before adding/subtracting:

```
GMST0 = Ls + 180_degrees
GMST = GMST0 + UT
LST  = GMST + local_longitude
```

The formulae above are written as if times are expressed in degrees. If we instead assume times are given in hours and angles in degrees, and if we explicitly write out the conversion factor of 15, we get:

```
GMST0 = 15 * (Ls + 180_degrees)
GMST = GMST0 + UT
LST  = GMST + local_longitude/15
```

6. The position of the Moon and of the planets

First, compute the eccentric anomaly, E, from M, the mean anomaly, and e, the eccentricity. As a first approximation, do (E and M in degrees):

```
E = M + e*(180/pi) * sin(M) * ( 1.0 + e * cos(M) )
```

or, if E and M are in radians:

```
E = M + e * sin(M) * ( 1.0 + e * cos(M) )
```

If e, the eccentricity, is less than about 0.05-0.06, this approximation is sufficiently accurate. If the eccentricity is larger, set E0=E and then use this iteration formula (E and M in degrees):

```
E1 = E0 - ( E0 - e*(180/pi) * sin(E0) - M ) / ( 1 - e * cos(E0) )
```

or (E and M in radians):

```
    E1 = E0 - ( E0 - e * sin(E0) - M ) / ( 1 - e * cos(E0) )
```

For each new iteration, replace E0 with E1. Iterate until E0 and E1 are sufficiently close together (about 0.001 degrees). For comet orbits with eccentricites close to one, a difference of less than 1E-4 or 1E-5 degrees should be required.

If this iteration formula won't converge, the eccentricity is probably too close to one. Then you should instead use the formulae for near-parabolic or parabolic orbits.

Now compute the planet's distance and true anomaly:

```
    xv = r * cos(v) = a * ( cos(E) - e )
    yv = r * sin(v) = a * ( sqrt(1.0 - e*e) * sin(E) )

    v = atan2( yv, xv )
    r = sqrt( xv*xv + yv*yv )
```

7. The position in space

Compute the planet's position in 3-dimensional space:

```
    xh = r * ( cos(N) * cos(v+w) - sin(N) * sin(v+w) * cos(i) )
    yh = r * ( sin(N) * cos(v+w) + cos(N) * sin(v+w) * cos(i) )
    zh = r * ( sin(v+w) * sin(i) )
```

For the Moon, this is the geocentric (Earth-centered) position in the ecliptic coordinate system. For the planets, this is the heliocentric (Sun-centered) position, also in the ecliptic coordinate system. If one wishes, one can compute the ecliptic longitude and latitude (this must be done if one wishes to correct for perturbations, or if one wants to precess the position to a standard epoch):

```
    lonecl = atan2( yh, xh )
    latecl = atan2( zh, sqrt(xh*xh+yh*yh) )
```

As a check one can compute sqrt(xh*xh+yh*yh+zh*zh), which of course should equal r (except for small round-off errors).

8. Precession

If one wishes to compute the planet's position for some standard epoch, such as 1950.0 or 2000.0 (e.g. to be able to plot the position on a star atlas), one must add the correction below to lonecl. If a planet's and not the Moon's position is computed, one must also add the same correction to lonsun, the Sun's longitude. The desired Epoch is expressed as the year, possibly with a fraction.

```
    lon_corr = 3.82394E-5 * ( 365.2422 * ( Epoch - 2000.0 ) - d )
```

If one wishes the position for today's epoch (useful when computing rising/setting times and the like), no corrections need to be done.

9. Perturbations of the Moon

If the position of the Moon is computed, and one wishes a better accuracy than about 2 degrees, the most important perturbations has to be taken into account. If one wishes 2 arc minute accuracy, all the following terms should be accounted for. If less accuracy is needed, some of the smaller terms can be omitted.

First compute:

```
Ms, Mm              Mean Anomaly of the Sun and the Moon
Nm                  Longitude of the Moon's node
ws, wm              Argument of perihelion for the Sun and the Moon
Ls = Ms + ws        Mean Longitude of the Sun  (Ns=0)
Lm = Mm + wm + Nm   Mean longitude of the Moon
D = Lm - Ls         Mean elongation of the Moon
F = Lm - Nm         Argument of latitude for the Moon
```

Add these terms to the Moon's longitude (degrees):

```
-1.274 * sin(Mm - 2*D)          (the Evection)
+0.658 * sin(2*D)               (the Variation)
-0.186 * sin(Ms)                (the Yearly Equation)
-0.059 * sin(2*Mm - 2*D)
-0.057 * sin(Mm - 2*D + Ms)
+0.053 * sin(Mm + 2*D)
+0.046 * sin(2*D - Ms)
+0.041 * sin(Mm - Ms)
-0.035 * sin(D)                 (the Parallactic Equation)
-0.031 * sin(Mm + Ms)
-0.015 * sin(2*F - 2*D)
+0.011 * sin(Mm - 4*D)
```

Add these terms to the Moon's latitude (degrees):

```
-0.173 * sin(F - 2*D)
-0.055 * sin(Mm - F - 2*D)
-0.046 * sin(Mm + F - 2*D)
+0.033 * sin(F + 2*D)
+0.017 * sin(2*Mm + F)
```

Add these terms to the Moon's distance (Earth radii):

```
-0.58 * cos(Mm - 2*D)
-0.46 * cos(2*D)
```

All perturbation terms that are smaller than 0.01 degrees in longitude or latitude and smaller than 0.1 Earth radii in distance have been omitted here. A few of the largest perturbation terms even have their own names! The Evection (the largest perturbation) was discovered already by Ptolemy a few thousand years ago (the Evection was one of Ptolemy's epicycles). The Variation and the Yearly Equation were both discovered by Tycho Brahe in the 16'th century.

The computations can be simplified by omitting the smaller perturbation terms. The error introduced by this seldom exceeds the sum of the amplitudes of the 4-5 largest omitted terms. If one only computes the three largest perturbation terms in longitude and the largest term in latitude, the error in longitude will rarley exceed 0.25 degrees, and in latitude 0.15 degrees.

10. Perturbations of Jupiter, Saturn and Uranus

The only planets having perturbations larger than 0.01 degrees are Jupiter, Saturn and Uranus. First compute:

```
Mj     Mean anomaly of Jupiter
Ms     Mean anomaly of Saturn
Mu     Mean anomaly of Uranus (needed for Uranus only)
```

Perturbations for Jupiter. Add these terms to the longitude:

```
-0.332 * sin(2*Mj - 5*Ms - 67.6 degrees)
-0.056 * sin(2*Mj - 2*Ms + 21 degrees)
+0.042 * sin(3*Mj - 5*Ms + 21 degrees)
-0.036 * sin(Mj - 2*Ms)
+0.022 * cos(Mj - Ms)
+0.023 * sin(2*Mj - 3*Ms + 52 degrees)
-0.016 * sin(Mj - 5*Ms - 69 degrees)
```

Perturbations for Saturn. Add these terms to the longitude:

```
+0.812 * sin(2*Mj - 5*Ms - 67.6 degrees)
-0.229 * cos(2*Mj - 4*Ms - 2 degrees)
+0.119 * sin(Mj - 2*Ms - 3 degrees)
+0.046 * sin(2*Mj - 6*Ms - 69 degrees)
+0.014 * sin(Mj - 3*Ms + 32 degrees)
```

For Saturn: *also* add these terms to the latitude:

```
-0.020 * cos(2*Mj - 4*Ms - 2 degrees)
+0.018 * sin(2*Mj - 6*Ms - 49 degrees)
```

Perturbations for Uranus: Add these terms to the longitude:

```
+0.040 * sin(Ms - 2*Mu + 6 degrees)
+0.035 * sin(Ms - 3*Mu + 33 degrees)
-0.015 * sin(Mj - Mu + 20 degrees)
```

The "great Jupiter-Saturn term" is the largest perturbation for both Jupiter and Saturn. Its period is 918 years, and its amplitude is 0.332 degrees for Jupiter and 0.812 degrees for Saturn. These is also a "great Saturn-Uranus term", period 560 years, amplitude 0.035 degrees for Uranus, less than 0.01 degrees for Saturn (and therefore omitted). The other perturbations have periods between 14 and 100 years. One should also mention the "great Uranus-Neptune term", which has a period of 4220 years and an amplitude of about one degree. It is not included here, instead it is included in the orbital elements of Uranus and Neptune.

For Mercury, Venus and Mars we can ignore all perturbations. For Neptune the only significant perturbation is already included in the orbital elements, as mentioned above, and therefore no further perturbation terms need to be accounted for.

11. Geocentric (Earth-centered) coordinates

Now we have computed the heliocentric (Sun-centered) coordinate of the planet, and we have included the most important perturbations. We want to compute the geocentric (Earth-centerd) position. We should convert the perturbed lonecl, latecl, r to (perturbed) xh, yh, zh:

```
xh = r * cos(lonecl) * cos(latecl)
yh = r * sin(lonecl) * cos(latecl)
zh = r               * sin(latecl)
```

If we are computing the Moon's position, this is already the geocentric position, and thus we simply set xg=xh, yg=yh, zg=zh. Otherwise we must also compute the Sun's position: convert lonsun, rs (where rs is the r computed here) to xs, ys:

```
xs = rs * cos(lonsun)
ys = rs * sin(lonsun)
```

(Of course, any correction for precession should be added to lonecl *and* lonsun *before* converting to xh,yh,zh and xs,ys).

Now convert from heliocentric to geocentric position:

```
xg = xh + xs
yg = yh + ys
zg = zh
```

We now have the planet's geocentric (Earth centered) position in rectangular, ecliptic coordinates.

12. Equatorial coordinates

Let's convert our rectangular, ecliptic coordinates to rectangular, equatorial coordinates: simply rotate the y-z-plane by ecl, the angle of the obliquity of the ecliptic:

```
xe = xg
ye = yg * cos(ecl) - zg * sin(ecl)
ze = yg * sin(ecl) + zg * cos(ecl)
```

Finally, compute the planet's Right Ascension (RA) and Declination (Dec):

```
RA  = atan2( ye, xe )
Dec = atan2( ze, sqrt(xe*xe+ye*ye) )
```

Compute the geocentric distance:

```
rg = sqrt(xg*xg+yg*yg+zg*zg) = sqrt(xe*xe+ye*ye+ze*ze)
```

Thie completes our computation of the equatorial coordinates.

12b. Azimuthal coordinates

To find the azimuthal coordinates (azimuth and altitude) we proceed by computing the HA (Hour Angle) of the object. But first we must compute the LST (Local Sidereal Time), which we do as described in 5b above. When we know LST, we can easily compute HA from:

```
HA = LST - RA
```

HA is usually given in the interval -12 to +12 hours, or -180 to +180 degrees. If HA is zero, the object can be seen directly to the south. If HA is negative, the object is to the east of south, and if HA is positive, the object is to the west of south. IF your computed HA should fall outside this interval, add or subtract 24 hours (or 360 degrees) until HA falls within this interval.

Now it's time to convert our objects HA and Decl to local azimuth and altitude. To do that, we also must know lat, our local latitude. Then we proceed as follows:

```
x = cos(HA) * cos(Decl)
y = sin(HA) * cos(Decl)
z = sin(Decl)

xhor = x * sin(lat) - z * cos(lat)
yhor = y
zhor = x * cos(lat) + z * sin(lat)

az  = atan2( yhor, xhor ) + 180_degrees
alt = asin( zhor ) = atan2( zhor, sqrt(xhor*xhor+yhor*yhor) )
```

This completes our calculation of the local azimuth and altitude. Note that azimuth is 0 at North, 90 deg at East, 180 deg at South and 270 deg at West. Altitude is of course 0 at the (mathematical) horizon, 90 deg at zenith, and negative below the horizon.

13. The Moon's topocentric position

The Moon's position, as computed earlier, is geocentric, i.e. as seen by an imaginary observer at the center of the Earth. Real observers dwell on the surface of the Earth, though, and they will see a different position - the topocentric position. This position can differ by more than one degree from the geocentric position. To compute the topocentric positions, we must add a correction to the geocentric position.

Let's start by computing the Moon's parallax, i.e. the apparent size of the (equatorial) radius of the Earth, as seen from the Moon:

```
mpar = asin( 1/r )
```

where r is the Moon's distance in Earth radii. It's simplest to apply the correction in horizontal coordinates (azimuth and altitude): within our accuracy aim of 1-2 arc minutes, no correction need to be applied to the azimuth. One need only apply a correction to the altitude above the horizon:

```
alt_topoc = alt_geoc - mpar * cos(alt_geoc)
```

Sometimes one need to correct for topocentric position directly in equatorial coordinates though, e.g. if one wants to draw on a star map how the Moon passes in front of the Pleiades, as seen from some specific location. Then we need to know the Moon's geocentric Right Ascension and Declination (RA, Decl), the Local Sidereal Time (LST), and our latitude (lat).

Our astronomical latitude (lat) must first be converted to a geocentric latitude (gclat), and distance from the center of the Earth (rho) in Earth equatorial radii. If we only want an approximate topocentric position, it's simplest to pretend that the Earth is a perfect sphere, and simply set:

```
gclat = lat,  rho = 1.0
```

However, if we do wish to account for the flattening of the Earth, we instead compute:

```
gclat = lat - 0.1924_deg * sin(2*lat)
rho   = 0.99833 + 0.00167 * cos(2*lat)
```

Next we compute the Moon's geocentric Hour Angle (HA) from the Moon's geocentric RA. First we must compute LST as described in 5b above, then we compute HA as:

```
HA = LST - RA
```

We also need an auxiliary angle, g:

```
g = atan( tan(gclat) / cos(HA) )
```

Now we're ready to convert the geocentric Right Ascention and Declination (RA, Decl) to their topocentric values (topRA, topDecl):

```
topRA   = RA  - mpar * rho * cos(gclat) * sin(HA) / cos(Decl)
topDecl = Decl - mpar * rho * sin(gclat) * sin(g - Decl) / sin(g)
```

(Note that if decl is exactly 90 deg, cos(Decl) becomes zero and we get a division by zero when computing topRA, but that formula breaks down only very close to the celestial poles anyway and we never see the Moon there. Also if gclat is precisely zero, g becomes zero too, and we get a division by zero when computing topDecl. In that case, replace the formula for topDecl with

```
topDecl = Decl - mpar * rho * sin(-Decl) * cos(HA)
```

which is valid for gclat equal to zero; it can also be used for gclat extremely close to zero).

This correction to topocentric position can also be applied to the Sun and the planets. But since they're much farther away, the correction becomes much smaller. It's largest for Venus at inferior conjunction, when Venus' parallax is somewhat larger than 32 arc seconds. Within our aim of obtaining a final accuracy of 1-2 arc minutes, it might barely be justified to correct to topocentric position when Venus is close to inferior conjunction, and perhaps also when Mars is at a favourable opposition. But in all other cases this correction can safely be ignored within our accuracy aim. We only need to worry about the Moon in this case.

If you want to compute topocentric coordinates for the planets too, you do it the same way as for the Moon, with one exception: the Moon's parallax is replaced by the parallax of the planet (ppar), as computed from this formula:

```
ppar = (8.794/3600)_deg / r
```

where r is the distance of the planet from the Earth, in astronomical units.

14. The position of Pluto

No analytical theory has ever been constructed for the planet Pluto. Our most accurate representation of the motion of this planet is from numerical integrations. Yet, a "curve fit" may be performed to these numerical integrations, and the result will be the formulae below, valid from about 1800 to about 2100. Compute d, our day number, as usual (section 3). Then compute these angles:

```
S  =   50.03  +  0.033459652 * d
P  =  238.95  +  0.003968789 * d
```

Next compute the heliocentric ecliptic longitude and latitude (degrees), and distance (a.u.):

```
lonecl = 238.9508  +  0.00400703 * d
        - 19.799 * sin(P)     + 19.848 * cos(P)
         + 0.897 * sin(2*P)    - 4.956 * cos(2*P)
         + 0.610 * sin(3*P)    + 1.211 * cos(3*P)
         - 0.341 * sin(4*P)    - 0.190 * cos(4*P)
         + 0.128 * sin(5*P)    - 0.034 * cos(5*P)
         - 0.038 * sin(6*P)    + 0.031 * cos(6*P)
         + 0.020 * sin(S-P)    - 0.010 * cos(S-P)
```

```
    latecl =  -3.9082
             - 5.453 * sin(P)     - 14.975 * cos(P)
             + 3.527 * sin(2*P)    + 1.673 * cos(2*P)
             - 1.051 * sin(3*P)    + 0.328 * cos(3*P)
             + 0.179 * sin(4*P)    - 0.292 * cos(4*P)
             + 0.019 * sin(5*P)    + 0.100 * cos(5*P)
             - 0.031 * sin(6*P)    - 0.026 * cos(6*P)
                                   + 0.011 * cos(S-P)

    r      =  40.72
           + 6.68 * sin(P)       + 6.90 * cos(P)
           - 1.18 * sin(2*P)     - 0.03 * cos(2*P)
           + 0.15 * sin(3*P)     - 0.14 * cos(3*P)
```

Now we know the heliocentric distance and ecliptic longitude/latitude for Pluto. To convert to geocentric coordinates, do as for the other planets.

15. The elongation and physical ephemerides of the planets

When we finally have completed our computation of the heliocentric and geocentric coordinates of the planets, it could also be interesting to know what the planet will look like. How large will it appear? What's its phase and magnitude (brightness)? These computations are much simpler than the computations of the positions.

Let's start by computing the apparent diameter of the planet:

```
    d = d0 / R
```

R is the planet's geocentric distance in astronomical units, and d is the planet's apparent diameter at a distance of 1 astronomical unit. d0 is of course different for each planet. The values below are given in seconds of arc. Some planets have different equatorial and polar diameters:

```
    Mercury     6.74"
    Venus      16.92"
    Earth      17.59" equ    17.53" pol
    Mars        9.36" equ     9.28" pol
    Jupiter   196.94" equ   185.08" pol
    Saturn    165.6"  equ   150.8"  pol
    Uranus     65.8"  equ    62.1"  pol
    Neptune    62.2"  equ    60.9"  pol
```

The Sun's apparent diameter at 1 astronomical unit is 1919.26". The Moon's apparent diameter is:

```
    d = 1873.7" * 60 / r
```

where r is the Moon's distance in Earth radii.

Two other quantities we'd like to know are the phase angle and the elongation.

The phase angle tells us the phase: if it's zero the planet appears "full", if it's 90 degrees it appears "half", and if it's 180 degrees it appears "new". Only the Moon and the inferior planets (i.e. Mercury and Venus) can have phase angles exceeding about 50 degrees.

The elongation is the apparent angular distance of the planet from the Sun. If the elongation is smaller than about 20 degrees, the planet is hard to observe, and if it's smaller than about 10 degrees it's usually not possible to observe the planet.

To compute phase angle and elongation we need to know the planet's heliocentric distance, r, its geocentric distance, R, and the distance to the Sun, s. Now we can compute the phase angle, FV, and the elongation, elong:

```
elong = acos( ( s*s + R*R - r*r ) / (2*s*R) )

FV    = acos( ( r*r + R*R - s*s ) / (2*r*R) )
```

When we know the phase angle, we can easily compute the phase:

```
phase  =  ( 1 + cos(FV) ) / 2  =  hav(180_deg - FV)
```

hav is the "haversine" function. The "haversine" (or "half versine") is an old and now obsolete trigonometric function. It's defined as:

```
hav(x)  =  ( 1 - cos(x) ) / 2   =   sin^2 (x/2)
```

As usual we must use a different procedure for the Moon. Since the Moon is so close to the Earth, the procedure above would introduce too big errors. Instead we use the Moon's ecliptic longitude and latitude, mlon and mlat, and the Sun's ecliptic longitude, mlon, to compute first the elongation, then the phase angle, of the Moon:

```
elong = acos( cos(slon - mlon) * cos(mlat) )

FV = 180_deg - elong
```

Finally we'll compute the magnitude (or brightness) of the planets. Here we need to use a formula that's different for each planet. FV is the phase angle (in degrees), r is the heliocentric and R the geocentric distance (both in AU):

```
Mercury:   -0.36 + 5*log10(r*R) + 0.027 * FV + 2.2E-13 * FV**6
Venus:     -4.34 + 5*log10(r*R) + 0.013 * FV + 4.2E-7  * FV**3
Mars:      -1.51 + 5*log10(r*R) + 0.016 * FV
Jupiter:   -9.25 + 5*log10(r*R) + 0.014 * FV
Saturn:    -9.0  + 5*log10(r*R) + 0.044 * FV + ring_magn
Uranus:    -7.15 + 5*log10(r*R) + 0.001 * FV
Neptune:   -6.90 + 5*log10(r*R) + 0.001 * FV

Moon:      +0.23 + 5*log10(r*R) + 0.026 * FV + 4.0E-9 * FV**4
```

** is the power operator, thus FV**6 is the phase angle (in degrees) raised to the sixth power. If FV is 150 degrees then FV**6 becomes ca 1.14E+13, which is a quite large number.

For the Moon, we also need the heliocentric distance, r, and geocentric distance, R, in AU (astronomical units). Here r can be set equal to the Sun's geocentric distance in AU. The Moon's geocentric distance, R, previously computed i Earth radii, must be converted to AU's - we do this by multiplying by sin(17.59"/2) = 1/23450. Or we could modify the magnitude formula for the Moon so it uses r in AU's and R in Earth radii:

```
Moon:     -21.62 + 5*log10(r*R) + 0.026 * FV + 4.0E-9 * FV**4
```

Saturn needs special treatment due to its rings: when Saturn's rings are "open" then Saturn will appear much brighter than when we view Saturn's rings edgewise. We'll compute ring_mang like this:

```
ring_magn = -2.6 * sin(abs(B)) + 1.2 * (sin(B))**2
```

Here B is the tilt of Saturn's rings which we also need to compute. Then we start with Saturn's geocentric ecliptic longitude and latitude (los, las) which we've already computed. We also need the tilt of the rings to the ecliptic, ir, and the "ascending node" of the plane of the rings, Nr:

```
ir = 28.06_deg
Nr = 169.51_deg + 3.82E-5_deg * d
```

Here d is our "day number" which we've used so many times before. Now let's compute the tilt of the rings:

```
B = asin( sin(las) * cos(ir) - cos(las) * sin(ir) * sin(los-Nr) )
```

This concludes our computation of the magnitudes of the planets.

16. Positions of asteroids

For asteroids, the orbital elements are often given as: N,i,w,a,e,M, where N,i,w are valid for a specific epoch (nowadays usually 2000.0). In our simplified computational scheme, the only significant changes with the epoch occurs in N. To convert N_Epoch to the N (today's epoch) we want to use, simply add a correction for precession:

```
N = N_Epoch + 0.013967 * ( 2000.0 - Epoch ) + 3.82394E-5 * d
```

where Epoch is expressed as a year with fractions, e.g. 1950.0 or 2000.0

Most often M, the mean anomaly, is given for another day than the day we want to compute the asteroid's position for. If the daily motion, n, is given, simply add n * (time difference in days) to M. If n is not given, but the period P (in days) is given, then n = 360.0/P. If P is not given, it can be computed from:

```
P = 365.2568984 * a**1.5 (days) = 1.00004024 * a**1.5   (years)
```

** is the power-of operator. a**1.5 is the same as sqrt(a*a*a).

When all orbital elements has been computed, proceed as with the other planets (section 6).

17. Position of comets.

For comets having elliptical orbits, M is usually not given. Instead T, the time of perihelion, is given. At perihelion M is zero. To compute M for any other moment, first compute the "day number" d of T (section 3), let's call this dT. Then compute the "day number" d of the moment for which you want to compute a position, let's call this d. Then M, the mean anomaly, is computed like:

```
M = 360.0 * (d-dT)/P    (degrees)
```

where P is given in days, and d-dT of course is the time since last perihelion, also in days.

Also, a, the semi-major axis, is usually not given. Instead q, the perihelion distance, is given. a can easily be computed from q and e:

```
a = q / (1.0 - e)
```

Then proceed as with an asteroid (section 16).

18. Parabolic orbits

If the comet has a parabolic orbit, a different method has to be used. Then the orbital period of the comet is infinite, and M (the mean anomaly) is always zero. The eccentricity, e, is always exactly 1. Since the semi-major axis, a, is infinite, we must instead directly use the perihelion distance, q. To compute a parabolic orbit, we proceed like this:

Compute the "day number", d, for T, the moment of perihelion, call this dT. Compute d for the moment we want to compute a position, call it d (section 3). The constant k is the Gaussian gravitational constant: *k = 0.01720209895 exactly!*

Then compute:

```
H = (d-dT) * (k/sqrt(2)) / q**1.5
```

where q**1.5 is the same as sqrt(q*q*q). Also compute:

```
h = 1.5 * H
g = sqrt( 1.0 + h*h )
s = cbrt( g + h ) - cbrt( g - h )
```

cbrt() is the cube root function: cbrt(x) = x**(1.0/3.0). The formulae has been devised so that both g+h and g-h always are positive. Therefore one can here safely compute cbrt(x) as exp(log(x)/3.0) . In general, cbrt(-x) = -cbrt(x) and of course cbrt(0) = 0.

Instead of trying to compute some eccentric anomaly, we compute the true anomaly and the heliocentric distance directly:

```
v = 2.0 * atan(s)
r = q * ( 1.0 + s*s )
```

When we know the true anomaly and the heliocentric distance, we can proceed by computing the position in space (section 7).

19. Near-parabolic orbits.

The most common case for a newly discovered comet is that the orbit isn't an exact parabola, but very nearly so. It's eccentricity is slightly below, or slightly above, one. The algorithm presented here can be used for eccentricities between about 0.98 and 1.02. If the eccentricity is smaller than 0.98 the elliptic algorithm (Kepler's equation/etc) should be used instead. No known comet has an eccentricity exceeding 1.02.

As for the purely parabolic orbit, we start by computing the time since perihelion in days, d-dT, and the perihelion distance, q. We also need to know the eccentricity, e. The constant k is the Gaussian gravitational constant: *k = 0.01720209895 exactly!*

Then we can proceed as:

```
a = 0.75 * (d-dT) * k * sqrt( (1 + e) / (q*q*q) )
```

```
b = sqrt( 1 + a*a )
W = cbrt(b + a) - cbrt(b - a)
f = (1 - e) / (1 + e)

a1 = (2/3) + (2/5) * W*W
a2 = (7/5) + (33/35) * W*W + (37/175) * W**4
a3 = W*W * ( (432/175) + (956/1125) * W*W + (84/1575) * W**4 )

C = W*W / (1 + W*W)
g = f * C*C
w = W * ( 1 + f * C * ( a1 + a2*g + a3*g*g ) )

v = 2 * atan(w)
r = q * ( 1 + w*w ) / ( 1 + w*w * f )
```

This algorithm yields the true anomaly, v, and the heliocentric distance, r, for a nearly-parabolic orbit. Note that this algorithm will fail very far from the perihelion; however the accuracy is sufficient for all comets closer than Pluto.

20. Rise and set times.

(this subject has received a document of its own)

21. Validity of orbital elements.

Due to perturbations from mainly the giant planets, like Jupiter and Saturn, the orbital elements of celestial bodies are constantly changing. The orbital elements for the Sun, Moon and the major planets, as given here, are valid for a long time period. However, orbital elemets given for a comet or an asteroid are valid only for a limited time. How long they are valid is hard to say generally. It depends on many factors, such as the accuracy you need, and the magnitude of the perturbations the comet or asteroid is subjected to from, say, Jupiter. A comet might travel in roughly the same orbit several orbital periods, experiencing only slight perturbations, but suddenly it might pass very close to Jupiter and get its orbit changed drastically. To compute this in a reliable way is quite complicated and completely out of scope for this description. As a rule of thumb, though, one can assume that an asteriod, if one uses the orbital elements for a specific epoch, one or a few revolutions away from that moment will have an error in its computed position of at least one or a few arc minutes, and possibly more. The errors will accumulate with time.

22. Links to other sites.

Astronomical Calculations by Keith Burnett: http://www.xylem.f2s.com/kepler/

Free BASIC programs can be found at ftp://seds.lpl.arizona.edu/pub/software/pc/general/ in: *ast.exe* (needs GWBASIC!) and *duff2ed.exe* (Pete Duffett-Smiths programs)

Books from **Willmann-Bell** about Math and Celestial Mechanics: http://www.willbell.com/math/index.htm

John Walker's freeware program **Home Planet + other stuff**: http://www.fourmilab.ch/

Elwood Downey's **Xephem** and **Ephem** programs, with C source code: http://www.clearskyinstitute.com/xephem/.
Ephem can also be found at ftp://seds.lpl.arizona.edu/pub/software/pc/general/ as *ephem421.zip*

Steven Moshier: Astronomy and numerical software source codes: http://www.moshier.net/
Dan Bruton's astronomical software links: http://www.physics.sfasu.edu/astro/software.html

Mel Bartel's software (much ATM stuff): http://www.efn.org/~mbartels/tm/software.html

Almanac data from **USNO**: http://aa.usno.navy.mil/data/

Asteroid orbital elements from **Lowell Observatory**: http://asteroid.lowell.edu/

SAC downloads: http://www.saguaroastro.org/content/downloads.htm

Earth Satellite software from **AMSAT**: http://www.amsat.org/amsat/ftpsoft.html

IMCCE (formerly Bureau des Longitudes): http://www.imcce.fr/
VSOP87: ftp://ftp.imcce.fr/pub/ephem/planets/vsop87/

DE403/404/410/414 at JPL: ftp://ssd.jpl.nasa.gov/pub/eph/export/
SSEphem at NRAO: ftp://ftp.cv.nrao.edu/NRAO-staff/rfisher/SSEphem/

Some catalogues at **CDS, Strasbourg, France** - high accuracy orbital theories:
Overview: http://cdsweb.u-strasbg.fr/cgi-bin/qcat?VI/
Precession & mean orbital elements: http://cdsweb.u-strasbg.fr/cgi-bin/qcat?VI/66/
ELP2000-82 (orbital theory of Moon): http://cdsweb.u-strasbg.fr/cgi-bin/qcat?VI/79/
VSOP87 (orbital theories of planets): http://cdsweb.u-strasbg.fr/cgi-bin/qcat?VI/81/

Astronomical Data Center http://adc.gsfc.nasa.gov/adc.html has lots of catalogs. Some of them are:
Asteroid orbital elements 1998: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1245/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1245/
JPL ephemeris DE118/LE62: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1093A/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1093A/
JPL ephemeris DE200/LE200: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1094A/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1094A/
USNO ZZCAT: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1157/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1157/
XZ catalog of zodiacal stars: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1201/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1201/
Tycho Reference Catalog: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1250/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1250/
Tycho 2 catalog: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1259/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1259/
USNO A2.0 catalog (very large): ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1252/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1252/
HST Guide Star catalog 1.2: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1254/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1254/
HST Guide Star catalog 1.3: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1255/

ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1255/
Tycho 2 catalog: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1259/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1259/
AC 2000.2 catalog: ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1275/
ftp://adc.astro.umd.edu/pub/adc/archives/catalogs/1/1275/
Other similar services are available at: CDS (France) - ADAC (Japan) - CAD (Russia)

The original ZC (Zodiacal Catalogue):
http://stjarnhimlen.se/zc/
http://web.archive.org/web/20030604102426/sorry.vse.cz/~ludek/zakryty/pub.phtml#zc
http://web.archive.org/web/20030728014108/http://sorry.vse.cz/~ludek/zakryty/pub/

Computing planetary positions - a tutorial with worked examples

By **Paul Schlyter, Stockholm, Sweden**
email: pausch@stjarnhimlen.se or WWW: http://stjarnhimlen.se/

Break out of a frame

How to compute planetary positions
Computing rise and set times

Today it's not really that difficult to compute a planet's position from its orbital elements. The only thing you'll need is a computer and a suitable program. If you want to write such a program yourself, this text contains the formulae you need. The aim here is to obtain the planetary positions at any date during the 20'th and 21'st century with an error of one or at the most two arc minutes, and to compute the position of an asteroid or a comet from its orbital elements.

No programs are given, because different computers and calculators are programmed in different languages. It is easier to convert formulae to your favourite language than to translate a program from one programming language to another. Therefore formulae are presented instead. Each formula is accompanied with one numeric example - this will enable you to check your implementation of these formulae. Just remember that all numerical quantities contain rounding errors, therefore you may get slightly different results in your program compared to the numerical results presented here. When I checked these numerical values I used a HP-48SX pocket calculator, which uses 12 digits of accuracy.

1. Fundamentals

A celestial body usually orbits the sun in an elliptical orbit. Perturbations from other planets causes small deviations from this elliptical orbit, but an unperturbed elliptical orbit can be used as a first approximation, and sometimes as the final approximation. If the distance from the Sun to the planet always is the same, then the planet follows a circular orbit. No planet does this, but the orbits of Venus and Neptune are very close to circles. Among the planets, Mercury and Pluto have orbits that deviate the most from a circle, i.e. are the most eccentric. Many asteroids have even more eccentric orbits, but the most eccentric orbits are to be found among the comets. Halley's comet, for instance, is closer to the Sun than Venus at perihelion, but farther away from the Sun than Neptune at aphelion. Some comets have even more eccentric orbits that are best approximated as a parabola. These orbits are not closed - a comet following a parabolic orbit passes the Sun only once, never to return. In reality these orbits are extremely elongated ellipses though, and these comets will eventually return, sometimes after many millennia.

The **perihelion** and **aphelion** are the points in the orbit when the planet is closest to and most distant from the Sun. A parabolic orbit only has a perihemium of course.

The **perigee** and **apogee** are points in the Moon's orbit (or the orbit of an artificial Earth satellite) which are closest to and most distant from the Earth.

The **celestial sphere** is an imaginary sphere around the observer, at an arbitrary distance.

The **celestial equator** is the Earth's equatorial plane projected on the celestial sphere.

The **ecliptic** is the plane of the Earth's orbit. This is also the plane of the Sun's yearly apparent motion. The ecliptic is inclined by approximately 23.4_deg to the celestial equator. The ecliptic intersects the celestial equator at two points: The Vernal Point (or "the first point of Aries"), and the Autumnal Point. The Vernal Point is the point of origin for two different commonly used celestial coordinates: equatorial coordinates and ecliptic coordinates.

Right Ascension and **Declination** are equatorial coordinates using the celestial equator as a fundamental plane. At the Vernal Point both the Right Ascension and the Declination are zero. The Right Ascension is usually measured in hours and minutes, where one revolution is 24 hours (which means 1 hour equals 15 degrees). It's counted countersunwise along the celestial equator. The Declination goes from +90 to -90 degrees, and it's positive north of, and negative south of, the celestial equator.

Longitude and **Latitude** are ecliptic coordinates, which use the ecliptic as a fundamental plane. Both are measured in degrees, and these coorinates too are both zero at the Vernal Point. The Longitude is counted countersunwise along the ecliptic. The Latitude is positive north of the ecliptic. Of course longitude and latitude are also used as terrestial coordinates, to measure a position of a point on the surface of the Earth.

Heliocentric, **Geocentric**, **Topocentric**. A position relative to the Sun is heliocentric. If the position is relative to the center of the Earth, then it's geocentric. A topocentric position is relative to an observer on the surface of the Earth. Within the aim of our accuracy of 1-2 arc minutes, the difference between geocentric and topocentric position is negligible for all celestial bodies except the Moon (and some occasional asteroid which happens to pass very close to the Earth).

The **orbital elements** consist of 6 quantities which completely define a circular, elliptic, parabolic or hyperbolic orbit. Three of these quantities describe the shape and size of the orbit, and the position of the planet in the orbit:

```
a  Mean distance, or semi-major axis
e  Eccentricity
T  Time at perihelion
```

A cirular orbit has zero eccentricity. An elliptical orbit has an eccentricity between zero and one. A parabolic orbit has an eccentricity of exactly one. Finally, a hyperbolic orbit has an eccentricity larger than one. A parabolic orbit has an infinite semi-major axis, a, therefore one instead gives the perihelion distance, q, for a parabolic orbit:

```
q  Perihelion distance  = a * (1 - e)
```

It is customary to give q instead of a also for hyperbolic orbit, and for elliptical orbits with eccentricity close to one.

The three remaining orbital elements define the orientation of the orbit in space:

```
i  Inclination, i.e. the "tilt" of the orbit relative to the
   ecliptic.  The inclination varies from 0 to 180 degrees. If
   the inclination is larger than 90 degrees, the planet is in
   a retrogade orbit, i.e. it moves "backwards".  The most
   well-known celestial body with retrogade motion is Comet Halley.

N  (usually written as "Capital Omega") Longitude of Ascending
   Node. This is the angle, along the ecliptic, from the Vernal
   Point to the Ascending Node, which is the intersection between
   the orbit and the ecliptic, where the planet moves from south
   of to north of the ecliptic, i.e. from negative to positive
   latitudes.

w  (usually written as "small Omega") The angle from the Ascending
   node to the Perihelion, along the orbit.
```

These are the primary orbital elements. From these many secondary orbital elements can be computed:

```
q  Perihelion distance  = a * (1 - e)

Q  Aphelion distance    = a * (1 + e)

P  Orbital period       = 365.256898326 * a**1.5/sqrt(1+m) days,
   where m = the mass of the planet in solar masses (0 for
   comets and asteroids). sqrt() is the square root function.

n  Daily motion         = 360_deg / P    degrees/day

t  Some epoch as a day count, e.g. Julian Day Number. The Time
   at Perihelion, T, should then be expressed as the same day count.

t - T   Time since Perihelion, usually in days

M  Mean Anomaly         = n * (t - T)  =  (t - T) * 360_deg / P
   Mean Anomaly is 0 at perihelion and 180 degrees at aphelion

L  Mean Longitude       = M + w + N

E  Eccentric anomaly, defined by Kepler's equation:   M = E - e * sin(E)
```

```
       An auxiliary angle to compute the position in an elliptic orbit

    v  True anomaly: the angle from perihelion to the planet, as seen
       from the Sun

    r  Heliocentric distance: the planet's distance from the Sun.
```

This relation is valid for an elliptic orbit:

```
        r * cos(v) = a * (cos(E) - e)
        r * sin(v) = a * sqrt(1 - e*e) * sin(E)

    x,y,z  Rectangular coordinates. Used e.g. when a heliocentric
           position (seen from the Sun) should be converted to a
           corresponding geocentric position (seen from the Earth).
```

2. Some useful functions

When doing these orbital computations it's useful to have access to several utility functions. Some of them are in the standard library of the programming language, but others must be added by the programmer. Pocket calculators are often better equipped: they usually have sin/cos/tan and their inverses in both radians and degrees. Often one also finds functions to directly convert between rectangular and polar coordinates.

On microcomputers the situation is worse. Let's start with the programming language Basic: there we can count on having sin/cos/tan and atn (=arctan) in radians, but nothing more. arcsin and arccos are missing, and the trig functions don't work in degrees. However one can add one's own function library, like e.g. this:

```
 95 rem Constants
100 pi = 3.14159265359
110 radeg = 180/pi

115 rem arcsin and arccos
120 def fnasin(x) = atn(x/sqr(1-x*x))
130 def fnacos(x) = pi/2 - fnasin(x)

135 rem Trig. functions in degrees
140 def fnsind(x) = sin(x/radeg)
150 def fncosd(x) = cos(x/radeg)
160 def fntand(x) = tan(x/radeg)
170 def fnasind(x) = radeg*atn(x/sqr(1-x*x))
180 def fnacosd(x) = 90 - fnasind(x)
190 def fnatand(x) = radeg*atn(x)

195 rem arctan in all four quadrants
200 def fnatan2(y,x) = atn(y/x) - pi*(x<0)
210 def fnatan2d(y,x) = radeg*atn(y/x) - 180*(x<0)

215 rem Normalize an angle between 0 and 360 degrees
217 rem Use Double Precision, if possible
220 def fnrev#(x#) = x# - int(x#/360#)*360#

225 rem Cube Root (needed for parabolic orbits)
230 def fncbrt(x) = exp( log(x)/3 )
```

The code above follows the convensions of traditional Microsoft Basic (MBASIC/BASICA/GWBASIC). If you use some other Basic dialect, you may want to modify this code.

The code above gives you sin/cos/tan and their inverses, in radians and degrees. The functions fnatan2() and fnatan2d() may need some explanation: they take two arguments, x and y, and they compute arctan(y/x) but puts the angle in the correct quadrant from -180 to +180 degrees. This is really part of a conversion from rectangular to polar coordinates, where the angle is computed. The distance is then computed by:

```
sqrt( x*x + y*y )
```

The function fnrev# normalizes an angle to between 0 and 360 degrees, by adding or subtracting even mutiples of 360 degrees until the final value falls between 0 and 360. The # sign means that the function and its argument use double precision. It is essential that this function uses more than single precision. More about this later.

One warning: some of these functions may divide by zero. If one tries to compute fnasin(1.0), it's computed as: atn(1/sqrt(1-1.0*1.0)) = atn(1/sqr(0)) = atn(1/0). This is not such a big problem, since in practice one rarely tries to compute the arc sine of exactly 1.0. Also, some dialects of Microsoft Basic then just print the warning message "Overflow", compute 1/0 as the highest possible floating-point number, and then continue the program. When computing the arctan of this very large number, one will get pi/2 (or 90 degrees), which is the correct result. However, if you have access to a more modern, structured, Basic, with the ability to define multi-line function, then by all means use this to write a better version of arcsin, which treats arcsin(1.0) as a special case.

There's a similar problem with the fncbrt() function - it only works for positive values. With a multi-line function definition it can be rewritten to work for negative values and zero as well, if one follows these simple rules: the Cube Root of zero is of course zero. The Cube Root of a negative number is computed by making the number positive, taking the Cube Root of that positive number, and then negating the result.

Two other popular programming language are C and C++. The standard library of these languages are better equipped with trigonometric functions. You'll find sin/cos/tan and their inverses, and even an atan2() function among the standard functions. All you need to do is to define some macros to get the trig functions in degrees (include *all* parentheses in these macro definitions), and to define a rev() function which reduces an angle to between 0 and 360 degrees, and a cbrt() function which computes the cube root.

```
#define PI          3.14159265358979323846
#define RADEG       (180.0/PI)
#define DEGRAD      (PI/180.0)
#define sind(x)     sin((x)*DEGRAD)
#define cosd(x)     cos((x)*DEGRAD)
#define tand(x)     tan((x)*DEGRAD)
#define asind(x)    (RADEG*asin(x))
#define acosd(x)    (RADEG*acos(x))
#define atand(x)    (RADEG*atan(x))
#define atan2d(y,x) (RADEG*atan2((y),(x)))


double rev( double x )
{
    return  x - floor(x/360.0)*360.0;
}
```

```
double cbrt( double x )
{
    if ( x > 0.0 )
        return exp( log(x) / 3.0 );
    else if ( x < 0.0 )
        return -cbrt(-x);
    else /* x == 0.0 */
        return 0.0;
}
```

In C++ the macros could preferably be defined as inline functions instead - this enables better type checking and also makes overloading of these function names possible.

The good ol' programming langauge FORTRAN is also well equipped with standard library trig functions: we find sin/cos/tan + inverses, and also an atan2 but only for radians. Therefore we need several function definitions to get the trig functions in degrees too. Below I give code only for SIND, ATAND, ATAN2D, plus REV and CBRT. The remaining functions COSD, TAND, ASIND and ACOSD are written in a similar way:

```
      FUNCTION SIND(X)
      PARAMETER(RADEG=57.2957795130823)
      SIND = SIN( X * (1.0/RADEG) )
      END

      FUNCTION ATAND(X)
      PARAMETER(RADEG=57.2957795130823)
      ATAND = RADEG * ATAN(X)
      END

      FUNCTION ATAN2D(Y,X)
      PARAMETER(RADEG=57.2957795130823)
      ATAN2D = RADEG * ATAN2(Y,X)
      END

      FUNCTION REV(X)
      REV = X - AINT(X/360.0)*360.0
      IF (REV.LT.0.0)  REV = REV + 360.0
      END

      FUNCTION CBRT(X)
      IF (X.GE.0.0) THEN
        CBRT = X ** (1.0/3.0)
      ELSE
        CBRT = -((-X)**(1.0/3.0))
      ENDIF
      END
```

The programming language Pascal is not as well equipped with trig functions. We have sin, cos, tan and arctan but nothing more. Therefore we need to write our own arcsin, arccos and arctan2, plus all the trig functions in degrees, and also the functions rev and cbrt. The trig functions in degrees are trivial when the others are defined, therefore I only define arcsin, arccos, arctan2, rev and cbrt:

```
const pi      = 3.14159265358979323846;
      half_pi = 1.57079632679489661923;
```

```
function arcsin( x : real ) : real;
begin
  if x = 1.0 then
    arcsin := half_pi
  else if x = -1.0 then
    arcsin := -half_pi
  else
    arcsin := arctan( x / sqrt( 1.0 - x*x ) )
end;

function arccos( x : real ) : real;
begin
  arccos := half_pi - arcsin(x);
end;

function arctan2( y, x : real ) : real;
begin
  if x = 0.0 then
    begin
      if y = 0.0 then
        (* Error! Give error message and stop program *)
      else if y > 0.0 then
        arctan2 := half_pi
      else
        arctan2 := -half_pi
    end
  else
    begin
      if x > 0.0 then
          arctan2 := arctan( y / x )
      else if x < 0.0 then
        begin
          if y >= 0.0 then
            arctan2 := arctan( y / x ) + pi
          else
            arctan2 := arctan( y / x ) - pi
        end;
    end;
end;

function rev( x : real ) : real;
var rv : real;
begin
  rv := x - trunc(x/360.0)*360.0;
  if rv < 0.0 then
    rv := rv + 360.0;
  rev := rv;
end;

function cbrt( x : real ) : real;
begin
  if x > 0.0 then
    cbrt := exp ( ln(x) / 3.0 )
  else if x < 0.0 then
    cbrt := -cbrt(-x)
  else
    cbrt := 0.0
end;
```

It's well worth the effort to ensure that all these functions are available. Then you don't need to worry about these details which really don't have much to do with the problem of computing a planetary position.

3. Rectangular and spherical coordinates

The position of a planet can be given in one of several ways. Two different ways that we'll use are rectangular and spherical coordinates.

Suppose a planet is situated at some RA, Decl and r, where RA is the Right Ascension, Decl the declination, and r the distance in some length unit. If r is unknown or irrelevant, set r = 1. Let's convert this to rectangular coordinates, x,y,z:

```
x = r * cos(RA) * cos(Decl)
y = r * sin(RA) * cos(Decl)
z = r * sin(Decl)
```

(before we compute the sine/cosine of RA, we must first convert RA from hours/minutes/seconds to hours + decimals. Then the hours are converted to degrees by multiplying by 15)

If we know the rectangular coordinates, we can convert to spherical coordinates by the formulae below:

```
r    = sqrt( x*x + y*y + z*z )
RA   = atan2( y, x )
Decl = asin( z / r ) = atan2( z, sqrt( x*x + y*y ) )
```

At the north and south celestial poles, both x and y are zero. Since atan2(0,0) is undefined, the RA is undefined too at the celestial poles. The simplest way to handle this is to assign RA some arbitrary value, e.g. zero. Close to the celestial poles the formula asin(z/r) to compute the declination becomes sensitive to round-off errors - here the formula atan2(z,sqrt(x*x+y*y)) is preferable.

Not only equatorial coordinates can be converted between spherical and rectangular. These conversions can also be applied to ecliptic and horizontal coordinates. Just exchange RA,Decl with long,lat (ecliptic coordinates) or azimuth,altitude (horizontal coordinates).

A coordinate system can be rotated. If a rectangular coordinate system is rotated around, say, the X axis, one can easily compute the new x,y,z coordinates. As an example, let's consider rotating an ecliptic x,y,z system to an equatorial x,y,z system. This rotation is done around the X axis (which points to the Vernal Point, the common point of origin in ecliptic and equatorial coordinates), through an angle of oblecl (the obliquity of the ecliptic, which is approximately 23.4 degrees):

```
xequat = xeclip
yequat = yeclip * cos(oblecl) - zeclip * sin(oblecl)
zequat = yeclip * sin(oblecl) + zeclip * cos(oblecl)
```

Now the x,y,z system is equatorial. It's easily rotated back to ecliptic coordinates by simply switching sign on oblecl:

```
xeclip = xequat
yeclip = yequat * cos(-oblecl) - zequat * sin(-oblecl)
```

```
    zeclip = yequat * sin(-oblecl) + zequat * cos(-oblecl)
```

When computing sin and cos of -oblecl, one can use the identities:

```
    cos(-x) = cos(x), sin(-x) = -sin(x)
```

Now let's put this together to convert directly from spherical ecliptic coordinates (long, lat) to spherical equatorial coordinates (RA, Decl). Since the distance r is irrelevant in this case, let's set r=1 for simplicity.

Example: At the Summer Solstice the Sun's ecliptic longitude is 90 degrees. The Sun's ecliptic latitude is always very nearly zero. Suppose the obliquity of the ecliptic is 23.4 degrees:

```
    xeclip = cos(90_deg) * cos(0_deg) = 0.0000
    yeclip = sin(90_deg) * cos(0_deg) = 1.0000
    zeclip = sin(0_deg)               = 0.0000
```

Rotate through oblecl = 23.4_deg:

```
    xequat = 0.0000
    yequat = 1.0000 * cos(23.4_deg) - 0.0000 * sin(23.4_deg)
    zequat = 1.0000 * sin(23.4_deg) + 0.0000 * cos(23.4_deg)
```

Our equatorial rectangular coordinates become:

```
    x = 0
    y = cos(23.4_deg) = 0.9178
    z = sin(23.4_deg) = 0.3971
```

The "distance", r, becomes: sqrt(0.8423 + 0.1577) = 1.0000 i.e. unchanged

```
    RA   = atan2( 0.9178, 0 ) = 90_deg
    Decl = asin( 0.3971 / 1.0000 ) = 23.40_deg
```

Alternatively:

```
    Decl = atan2( 0.3971, sqrt( 0.8423 + 0.0000 ) ) = 23.40_deg
```

Here we immediately see how simple it is to compute RA, thanks to the atan2() function: no need to consider in which quadrant it falls, the atan2() function handles this.

4. The time scale. A test date.

The time scale used here is a "day number" from 2000 Jan 0.0 TDT, which is the same as 1999 Dec 31.0 TDT, i.e. precisely at midnight TDT at the start of the last day of this century. With the modest accuracy we strive for here, one can usually disregard the difference between TDT (formerly canned ET) and UT.

We call our day number d. It can be computed from a JD (Julian Day Number) or MJD (Modified Julian Day Number) like this:

```
d  =  JD - 2451543.5  =  MJD - 51543.0
```

We can also compute d directly from the calendar date like this:

```
d = 367*Y - (7*(Y + ((M+9)/12)))/4 + (275*M)/9 + D - 730530
```

Y is the year (all 4 digits!), M the month (1-12) and D the date. In this formula all divisions should be INTEGER divisions. Use "div" instead of "/" in Pascal, and "\" instead of "/" in Microsoft Basic. In C/C++ and FORTRAN it's sufficient to ensure that both operands to "/" are integers.

This formula yields d as an integer, which is valid at the start (at midnight), in UT or TDT, of that calendar date. If you want d for some other time, add UT/24.0 (here the division is a floating-point division!) to the d obtained above.

Example: compute d for 19 april 1990, at 0:00 UT :

We can look up, or compute the JD for this moment, and we'll get: JD = 2448000.5 which yields d = -3543.0

Or we can compute d directly from the calendar date:

```
d = 367*1990 - (7*(1990 + ((4+9)/12)))/4 + (275*4)/9 + 19 - 730530

d = 730330 - (7*(1990 + (13/12)))/4 + 1100/9 + 19 - 730530

d = 730330 - (7*(1990 + 1))/4 + 122 + 19 - 730530

d = 730330 - (7*1991)/4 + 122 + 19 - 730530

d = 730330 - 13937/4 + 122 + 19 - 730530

d = 730330 - 3484 + 122 + 19 - 730530  =  -3543
```

This moment, 1990 april 19, 0:00 UT/TDT, will be our test date for most numerical examples below. d is negative since our test date, 19 april 1990, is earlier than the "point of origin" of our day number, 31 dec 1999.

5. The Sun's position.

Today most people know that the Earth orbits the Sun and not the other way around. But below we'll pretend as if it was the other way around. These orbital elements are thus valid for the Sun's (apparent) orbit around the Earth. All angular values are expressed in degrees:

```
w = 282.9404_deg + 4.70935E-5_deg   * d    (longitude of perihelion)
a = 1.000000                               (mean distance, a.u.)
e = 0.016709 - 1.151E-9             * d    (eccentricity)
M = 356.0470_deg + 0.9856002585_deg * d    (mean anomaly)
```

We also need the obliquity of the ecliptic, oblecl:

```
oblecl = 23.4393_deg - 3.563E-7_deg * d
```

and the Sun's mean longitude, L:

```
L = w + M
```

By definition the Sun is (apparently) moving in the plane of the ecliptic. The inclination, i, is therefore zero, and the longitude of the ascending node, N, becomes undefined. For simplicity we'll assign the value zero to N, which means that w, the angle between acending node and perihelion, becomes equal to the longitude of the perihelion.

Now let's compute the Sun's position for our test date 19 april 1990. Earlier we've computed d = -3543.0 which yields:

```
w = 282.7735_deg
a = 1.000000
e = 0.016713
M = -3135.9347_deg
```

We immediately notice that the mean anomaly, M, will get a large negative value. We use our function rev() to reduce this value to between 0 and 360 degrees. To do this, rev() will need to add 9*360 = 3240 degrees to this angle:

```
M = 104.0653_deg
```

We also compute:

```
L = w + M = 386.8388_deg = 26.8388_deg

oblecl = 23.4406_deg
```

Let's go on computing an auxiliary angle, the eccentric anomaly. Since the eccentricity of the Sun's (i.e. the Earth's) orbit is so small, 0.017, a first approximation of E will be accurate enough. Below E and M are in degrees:

```
E = M + (180/pi) * e * sin(M) * (1 + e * cos(M))
```

When we plug in M and e, we get:

```
E = 104.9904_deg
```

Now we compute the Sun's rectangular coordinates in the plane of the ecliptic, where the X axis points towards the perihelion:

```
x = r * cos(v) = cos(E) - e
y = r * sin(v) = sin(E) * sqrt(1 - e*e)
```

We plug in E and get:

```
x = -0.275370
y = +0.965834
```

Convert to distance and true anomaly:

```
r = sqrt(x*x + y*y)
v = arctan2( y, x )
```

Numerically we get:

```
r = 1.004323
v = 105.9134_deg
```

Now we can compute the longitude of the Sun:

```
lon = v + w

lon = 105.9134_deg + 282.7735_deg = 388.6869_deg = 28.6869_deg
```

We're done!

How close did we get to the correct values? Let's compare with the Astronomical Almanac:

```
        Our results    Astron. Almanac      Difference

lon     28.6869_deg      28.6813_deg        +0.0056_deg = 20"
r        1.004323         1.004311          +0.000012
```

The error in the Sun's longitude was 20 arc seconds, which is well below our aim of an accuracy of one arc minute. The error in the distance was about 1/3 Earth radius. Not bad!

Finally we'll compute the Sun's ecliptic rectangular coordinates, rotate these to equatorial coordinates, and then compute the Sun's RA and Decl:

```
x = r * cos(lon)
y = r * sin(lon)
z = 0.0
```

We plug in our longitude:

```
x = 0.881048
y = 0.482098
z = 0.0
```

We use oblecl = 23.4406 degrees, and rotate these coordinates:

```
xequat = 0.881048
yequat = 0.482098 * cos(23.4406_deg) - 0.0 * sin(23.4406_deg)
zequat = 0.482098 * sin(23.4406_deg) + 0.0 * cos(23.4406_deg)
```

which yields:

```
xequat = 0.881048
yequat = 0.442312
zequat = 0.191778
```

Convert to RA and Decl:

```
r    =   1.004323  (unchanged)
RA   =  26.6580_deg = 26.6580/15 h = 1.77720 h = 1h 46m 37.9s
Decl = +11.0084_deg = +11_deg 0' 30"
```

The Astronomical Almanac says:

```
RA  = 1h 46m 36.0s     Decl = +11_deg 0' 22"
```

6. Sidereal time and hour angle. Altitude and azimuth

The Sidereal Time tells the Right Ascension of the part of the sky that's precisely south, i.e. in the meridian. Sidereal Time is a local time, which can be computed from:

```
SIDTIME = GMST0 + UT + LON/15
```

where SIDTIME, GMST0 and UT are given in hours + decimals. GMST0 is the Sidereal Time at the Greenwich meridian at 00:00 right now, and UT is the same as Greenwich time. LON is the terrestial longitude in degrees (western longitude is negative, eastern positive). To "convert" the longitude from degrees to hours we divide it by 15. If the Sidereal Time becomes negative, we add 24 hours, if it exceeds 24 hours we subtract 24 hours.

Now, how do we compute GMST0? Simple - we add (or subtract) 180 degrees to (from) L, the Sun's mean longitude, which we've already computed earlier. Then we normalise the result to between 0 and 360 degrees, by applying the rev() function. Finally we divide by 15 to convert degrees to hours:

```
GMST0 = ( L + 180_deg ) / 15 = L/15 + 12h
```

We've already computed L = 26.8388_deg, which yields:

```
GMST0 = 26.8388_deg/15 + 12h = 13.78925 hours
```

Now let's compute the local Sidereal Time for the time meridian of Central Europe (at 15 deg east longitude = +15 degrees long) on 19 april 1990 at 00:00 UT:

```
SIDTIME = GMST0 + UT + LON/15 = 13.78925h + 0 + 15_deg/15 = 14.78925 hours

SIDTIME = 14h 47m 21.3s
```

To compute the altitude and azimuth we also need to know the Hour Angle, HA. The Hour Angle is zero when the clestial body is in the meridian i.e. in the south (or, from the southern heimsphere, in the north) - this is the moment when the celestial body is at its highest above the horizon.

The Hour Angle increases with time (unless the object is moving faster than the Earth rotates; this is the case for most artificial satellites). It is computed from:

```
HA = SIDTIME - RA
```

Here SIDTIME and RA must be expressed in the same unit, hours or degrees. We choose hours:

```
HA = 14.78925h - 1.77720h = 13.01205h = 195.1808_deg
```

If the Hour Angle is 180_deg the celestial body can be seen (or not be seen, if it's below the horizon) in the north (or in the south, from the southern hemisphere). We get HA = 195_deg for the Sun, which seems OK since it's around 01:00 local time.

Now we'll convert the Sun's HA = 195.1808_deg and Decl = +11.0084_deg to a rectangular (x,y,z) coordinate system where the X axis points to the celestial equator in the south, the Y axis to the horizon in the west, and the Z axis to the north celestial pole: The distance, r, is here irrelevant so we set r=1 for simplicity:

```
x = cos(HA) * cos(Decl) = -0.947346
y = sin(HA) * cos(Decl) = -0.257047
z = sin(Decl)           = +0.190953
```

Now we'll rotate this x,y,z system along an axis going east-west, i.e. the Y axis, in such a way that the Z axis will point to the zenith. At the North Pole the angle of rotation will be zero since there the north celestial pole already is in the zenith. At other latitudes the angle of rotation becomes 90_deg - latitude. This yields:

```
xhor = x * cos(90_deg - lat) - z * sin(90_deg - lat)
yhor = y
zhor = x * sin(90_deg - lat) + z * cos(90_deg - lat)
```

Since sin(90_deg-lat) = cos(lat) (and reverse) we'll get:

```
xhor = x * sin(lat) - z * cos(lat)
yhor = y
zhor = x * cos(lat) + z * sin(lat)
```

Finally we compute our azimuth and altitude:

```
azimuth  = atan2( yhor, xhor ) + 180_deg
altitude = asin( zhor ) = atan2( zhor, sqrt(xhor*xhor+yhor*yhor) )
```

Why did we add 180_deg to the azimuth? To adapt to the most common way to specify azimuth: from North (0_deg) through East (90_deg), South (180_deg), West (270_deg) and back to North. If we didn't add 180_deg the azimuth would be counted from South through West/etc instead. If you want to use that kind of azimuth, then don't add 180_deg above.

We select some place in central Scandinavia: the longitude is as before +15_deg (15_deg East), and the latitude is +60_deg (60_deg N):

```
xhor = -0.947346 * sin(60_deg) - (+0.190953) * cos(60_deg) = -0.915902
yhor = -0.257047                                            = -0.257047
zhor = -0.947346 * cos(60_deg) + (+0.190953) * sin(60_deg) = -0.308303
```

Now we've computed the horizontal coordinates in rectangular form. To get azimuth and altitude we convert to spherical coordinates (r=1):

```
azimuth  = atan2(-0.257047,-0.915902) + 180_deg = 375.6767_deg = 15.6767_deg
altitude = asin( -0.308303 ) = -17.9570_deg
```

Let's round the final result to two decimals:

```
azimuth = 15.68_deg, altitude = -17.96_deg.
```

The Sun is thus 17.96_deg below the horizon at this moment and place. This is very close to astronomical twilight (18_deg below the horizon).

7. The Moon's position

Let's continue by computing the position of the Moon. The computatons will become more complicated, since the Moon doesn't move in the plane of the ecliptic, but in a plane inclined somewhat more than 5 degrees to the ecliptic. Also, the Sun perturbs the Moon's motion significantly, an effect we must account for.

The orbital elements of the Moon are:

```
N = 125.1228_deg - 0.0529538083_deg  * d    (Long asc. node)
i =   5.1454_deg                            (Inclination)
w = 318.0634_deg + 0.1643573223_deg  * d    (Arg. of perigee)
a =  60.2666                                (Mean distance)
e = 0.054900                                (Eccentricity)
M = 115.3654_deg + 13.0649929509_deg * d    (Mean anomaly)
```

The Moon's ascending node is moving in a retrogade ("backwards") direction one revolution in about 18.6 years. The Moon's perigee (the point of the orbit closest to the Earth) moves in a "forwards" direction one revolution in about 8.8 years. The Moon itself moves one revolution in aboout 27.5 days. The mean distance, or semi-major axis, is expressed in Earth equatorial radii).

Let's compute numerical values for the Moon's orbital elements on our test date 19 april 1990 (d = -3543):

```
N =  312.7381_deg
i =    5.1454_deg
w = -264.2546_deg
a =   60.2666         (Earth equatorial radii)
e =    0.054900
M = -46173.9046_deg
```

Now the need for sufficient numerical accuracy becomes obvious. If we would compute M with normal single precision, i.e. only 7 decimal digits of accuracy, then the error in M would here be about 0.01 degrees. Had we selected a date around 1901 or 2099 then the error in M would have been about 0.1 degrees, which is definitely worse than our aim of a maximum error of one or two arc minutes. Therefore, when computing the

Moon's mean anomaly, M, it's important to use at least 9 or 10 digits of accuracy.

(This was a real problem around 1980, when microcomputers were a novelty. Around then, pocket calculators usually offered better precision than microcomputers. But these days are long gone: nowadays microcomputers routinely offer double precision (14-16 digits of accuracy) support in hardware; all you need to do is to select a compiler which really suports this.)

All angular elements should be normalized to within 0-360 degrees, by calling the rev() function. We get:

```
N = 312.7381_deg
i =   5.1454_deg
w =  95.7454_deg
a =  60.2666          (Earth equatorial radii)
e =   0.054900
M = 266.0954_deg
```

To normalize M we had to add 129*360 = 46440 degrees.

Next, we compute E, the eccentric anomaly. We start with a first approximation (E0 and M in degrees):

```
E0 = M + (180_deg/pi) * e * sin(M) * (1 + e * cos(M))
```

The eccentricity of the Moon's orbit is larger than of the Earth's orbit. This means that our first approximation will have a bigger error - it'll be close to the limit of what we can tolerate within our accuracy aim. If you want to be careful, you should therefore use the iteration formula below: set E0 to our first approximation, compute E1, then set E0 to E1 and compute a new E1, until the difference between E0 and E1 becomes small enough, i.e. smaller than about 0.005 degrees. Then use the last E1 as the final value. In the formula below, E0, E1 and M are in degrees:

```
E1 = E0 - (E0 - (180_deg/pi) * e * sin(E0) - M) / (1 - e * cos(E0))
```

On our test date, the first approximation of E becomes: E=262.9689_deg The iterations then yield: E = 262.9735_deg, 262.9735_deg ......

Now we've computed E - the next step is to compute the Moon's distance and true anomaly. First we compute rectangular (x,y) coordinates in the plane of the lunar orbit:

```
x = r * cos(v) = a * (cos(E) - e)
y = r * sin(v) = a * sqrt(1 - e*e) * sin(E)
```

Our test date yields:

```
x = -10.68095
y = -59.72377
```

Then we convert this to distance and true anonaly:

```
r = sqrt( x*x + y*y ) = 60.67134 Earth radii
v = atan2( y, x )     = 259.8605_deg
```

Now we know the Moon's position in the plane of the lunar orbit. To compute the Moon's position in ecliptic coordinates, we apply these formulae:

```
xeclip = r * ( cos(N) * cos(v+w) - sin(N) * sin(v+w) * cos(i) )
yeclip = r * ( sin(N) * cos(v+w) + cos(N) * sin(v+w) * cos(i) )
zeclip = r * sin(v+w) * sin(i)
```

Our test date yields:

```
xeclip = +37.65311
yeclip = -47.57180
zeclip =  -0.41687
```

Convert to ecliptic longitude, latitude, and distance:

```
long = 308.3616_deg
lat  =  -0.3937_deg
r    =  60.6713
```

According to the Astronomical Almanac, the Moon's position at this moment is 306.94_deg, and the latitude is -0.55_deg. This differs from our figures by 1.42_deg in longitude and 0.16_deg in latitude!!! This difference is much larger than our aim of an error of max 1-2 arc minutes. Why is this so?

8. The Moon's position with higher accuracy. Perturbations

The big error in our computed lunar position is because we completely ignored the perturbations on the Moon. Below we'll compute the most important perturbation terms, and then add these as corrections to our previous figures. This will cut down the error to 1-2 arc minutes, or less.

First we need several fundamental arguments:

```
Sun's  mean longitude:        Ls      (already computed)
Moon's mean longitude:        Lm   =  N + w + M (for the Moon)
Sun's  mean anomaly:          Ms      (already computed)
Moon's mean anomaly:          Mm      (already computed)
Moon's mean elongation:       D    =  Lm - Ls
Moon's argument of latitude:  F    =  Lm - N
```

Let's plug in the figures for our test date:

```
Ms = 104.0653_deg
Mm = 266.0954_deg
Ls =  26.8388_deg
Lm = 312.7381_deg + 95.7454_deg + 266.0954_deg = 674.5789_deg
      = 314.5789_deg
D  = 314.5789_deg -  26.8388_deg = 287.7401_deg
F  = 314.5789_deg - 312.7381_deg = 1.8408_deg
```

Now it's time to compute and add up the 12 largest perturbation terms in longitude, the 5 largest in latitude, and the 2 largest in distance. These are all the perturbation terms with an amplitude larger than 0.01_deg in longitude resp latitude. In the lunar distance, only the perturbation terms larger than 0.1 Earth radii has been included:

Perturbations in longitude (degrees):

```
-1.274_deg * sin(Mm - 2*D)      (Evection)
+0.658_deg * sin(2*D)           (Variation)
-0.186_deg * sin(Ms)            (Yearly equation)
-0.059_deg * sin(2*Mm - 2*D)
-0.057_deg * sin(Mm - 2*D + Ms)
+0.053_deg * sin(Mm + 2*D)
+0.046_deg * sin(2*D - Ms)
+0.041_deg * sin(Mm - Ms)
-0.035_deg * sin(D)              (Parallactic equation)
-0.031_deg * sin(Mm + Ms)
-0.015_deg * sin(2*F - 2*D)
+0.011_deg * sin(Mm - 4*D)
```

Perturbations in latitude (degrees):

```
-0.173_deg * sin(F - 2*D)
-0.055_deg * sin(Mm - F - 2*D)
-0.046_deg * sin(Mm + F - 2*D)
+0.033_deg * sin(F + 2*D)
+0.017_deg * sin(2*Mm + F)
```

Perturbations in lunar distance (Earth radii):

```
-0.58 * cos(Mm - 2*D)
-0.46 * cos(2*D)
```

Some of the largest perturbation terms in longitude even have received individual names! The largest perturbation, the Evection, was discovered already by Ptolemy (he made it one of the epicycles in his theory for the Moon's motion). The two next largest perturbations, the Variation and the Yearly equation, were discovered by Tycho Brahe.

If you don't need 1-2 arcmin accuracy, you don't need to compute all these perturbation terms. If you only include the two largest terms in longitude and the largest in latitude, the error in longitude rarely becomes larger than 0.25_deg, and in latitude rarely larger than 0.15_deg.

Let's compute these perturbation terms for our test date:

```
longitude: -0.9847 - 0.3819 - 0.1804 + 0.0405 - 0.0244 + 0.0452 +
            0.0428 + 0.0126 - 0.0333 - 0.0055 - 0.0079 - 0.0029
           = -1.4132_deg

latitude: -0.0958 - 0.0414 - 0.0365 - 0.0200 + 0.0018 = -0.1919_deg

distance: -0.3680 + 0.3745 = +0.0066 Earth radii
```

Add this to the ecliptic positions we earlier computed:

```
long = 308.3616_deg - 1.4132_deg  =  306.9484_deg
lat  =  -0.3937_deg - 0.1919_deg  =   -0.5856_deg
dist =  60.6713  + 0.0066         =   60.6779 Earth radii
```

Let's compare with the Astronomical Almanac:

```
longitude 306.94_deg,  latitude -0.55_deg,  distance 60.793 Earth radii
```

Now the agreement is much better, right?

Let's continue by converting these ecliptic coordinates to Right Ascension and Declination. We do as described earlier: convert the ecliptic longitude/latitude to rectangular (x,y,z) coordinates, rotate this x,y,z, system through an angle corresponding to the obliquity of the ecliptic, then convert back to spherical coordinates. The Moon's distance doesn't matter here, and one can therefore set r=1.0. We get:

```
RA   = 309.5011_deg
Decl = -19.1032_deg
```

According to the Astronomical Almanac:

```
RA = 309.4881_deg
Decl = -19.0741_deg
```

9. The Moon's topocentric position.

The Moon's position, as computed earlier, is geocentric, i.e. as seen by an imaginary observer at the center of the Earth. Real observers dwell on the surface of the Earth, though, and they will see a different position - the topocentric position. This position can differ by more than one degree from the geocentric position. To compute the topocentric positions, we must add a correction to the geocentric position.

Let's start by computing the Moon's parallax, i.e. the apparent size of the (equatorial) radius of the Earth, as seen from the Moon:

```
mpar = asin( 1/r )
```

where r is the Moon's distance in Earth radii. It's simplest to apply the correction in horizontal coordinates (azimuth and altitude): within our accuracy aim of 1-2 arc minutes, no correction need to be applied to the azimuth. One need only apply a correction to the altitude above the horizon:

```
alt_topoc = alt_geoc - mpar * cos(alt_geoc)
```

Sometimes one needs to correct for topocentric position directly in equatorial coordinates though, e.g. if one wants to draw on a star map how the Moon passes in front of the Pleiades, as seen from some specific location. Then we need to know the Moon's geocentric Right Ascension and Declination (RA, Decl), the Local Sidereal Time (LST), and our latitude (lat).

Our astronomical latitude (lat) must first be converted to a geocentric latitude (gclat) and distance from the center of the Earth (rho) in Earth equatorial radii. If we only want an approximate topocentric position, it's simplest to pretend that the Earth is a perfect sphere, and simply set:

```
gclat = lat,  rho = 1.0
```

However, if we do wish to account for the flattening of the Earth, we instead compute:

```
gclat = lat - 0.1924_deg * sin(2*lat)
rho   = 0.99833 + 0.00167 * cos(2*lat)
```

Next we compute the Moon's geocentric Hour Angle (HA):

```
HA = LST - RA
```

We also need an auxiliary angle, g:

```
g = atan( tan(gclat) / cos(HA) )
```

Now we're ready to convert the geocentric Right Ascention and Declination (RA, Decl) to their topocentric values (topRA, topDecl):

```
topRA   = RA  - mpar * rho * cos(gclat) * sin(HA) / cos(Decl)
topDecl = Decl - mpar * rho * sin(gclat) * sin(g - Decl) / sin(g)
```

Let's do this correction for our test date and for the geographical position 15 deg E longitude (= +15_deg) and 60 deg N latitude (= +60_deg). Earlier we computed the Local Sidereal Time as LST = SIDTIME = 14.78925 hours. Multiply by 15 to get degrees: LST = 221.8388_deg

The Moon's Hour Angle becomes:

```
HA = LST - RA = -87.6623_deg = 272.3377_deg
```

Our latitude +60_deg yields the following geocentric latitude and distance from the Earth's center:

```
gclat = 59.83_deg  rho   = 0.9975
```

We've already computed the Moon's distance as 60.6779 Earth radii, which means the Moon's parallax is:

```
mpar = 0.9443_deg
```

The auxiliary angle g becomes:

```
g = 88.642
```

And finally the Moon's topocentric position becomes:

```
topRA   = 309.5011_deg - (-0.5006_deg) = 310.0017_deg
topDecl = -19.1032_deg - (+0.7758_deg) = -19.8790_deg
```

This correction to topocentric position can also be applied to the Sun and the planets. But since they're much farther away, the correction becomes much smaller. It's largest for Venus at inferior conjunction, when Venus' parallax is somewhat larger than 32 arc seconds. Within our aim of obtaining a final accuracy of 1-2 arc minutes, it might barely be justified to correct to topocentric position when Venus is close to inferior conjunction, and perhaps also when Mars is at a favourable opposition. But in all other cases this correction can safely be ignored within our accuracy aim. We only need to worry about the Moon in this case.

If you want to compute topocentric coordinates for the planets anyway, you do it the same way as for the Moon, with one exception: the parallax of the planet (ppar) is computed from this formula:

```
ppar = (8.794/3600)_deg / r
```

where r is the distance of the planet from the Earth, in astronomical units.

10. The orbital elements of the planets

To compute the positions of the major planets, we first must compute their orbital elements:

Mercury:

```
N =  48.3313_deg + 3.24587E-5_deg   * d    (Long of asc. node)
i =   7.0047_deg + 5.00E-8_deg      * d    (Inclination)
w =  29.1241_deg + 1.01444E-5_deg   * d    (Argument of perihelion)
a = 0.387098                               (Semi-major axis)
e = 0.205635     + 5.59E-10         * d    (Eccentricity)
M = 168.6562_deg + 4.0923344368_deg * d    (Mean anonaly)
```

Venus:

```
N =  76.6799_deg + 2.46590E-5_deg   * d
i =   3.3946_deg + 2.75E-8_deg      * d
w =  54.8910_deg + 1.38374E-5_deg   * d
a = 0.723330
e = 0.006773     - 1.302E-9         * d
M =  48.0052_deg + 1.6021302244_deg * d
```

Mars:

```
N =  49.5574_deg + 2.11081E-5_deg   * d
i =   1.8497_deg - 1.78E-8_deg      * d
w = 286.5016_deg + 2.92961E-5_deg   * d
a = 1.523688
e = 0.093405     + 2.516E-9         * d
M =  18.6021_deg + 0.5240207766_deg * d
```

Jupiter:

```
N = 100.4542_deg + 2.76854E-5_deg   * d
i =   1.3030_deg - 1.557E-7_deg     * d
w = 273.8777_deg + 1.64505E-5_deg   * d
a = 5.20256
```

```
e = 0.048498     + 4.469E-9          * d
M =  19.8950_deg + 0.0830853001_deg * d
```

Saturn:

```
N = 113.6634_deg + 2.38980E-5_deg   * d
i =   2.4886_deg - 1.081E-7_deg     * d
w = 339.3939_deg + 2.97661E-5_deg   * d
a = 9.55475
e = 0.055546     - 9.499E-9         * d
M = 316.9670_deg + 0.0334442282_deg * d
```

Uranus:

```
N =  74.0005_deg + 1.3978E-5_deg    * d
i =   0.7733_deg + 1.9E-8_deg       * d
w =  96.6612_deg + 3.0565E-5_deg    * d
a = 19.18171     - 1.55E-8          * d
e = 0.047318     + 7.45E-9          * d
M = 142.5905_deg + 0.011725806_deg  * d
```

Neptune:

```
N = 131.7806_deg + 3.0173E-5_deg    * d
i =   1.7700_deg - 2.55E-7_deg      * d
w = 272.8461_deg - 6.027E-6_deg     * d
a = 30.05826     + 3.313E-8         * d
e = 0.008606     + 2.15E-9          * d
M = 260.2471_deg + 0.005995147_deg  * d
```

Let's compute all these elements for our test date, 19 april 1990 0h UT:

```
             N        i        w         a         e         M
            deg      deg      deg       a.e.                deg

Mercury   48.2163  7.0045   29.0882   0.387098  0.205633   69.5153
Venus     76.5925  3.3945   54.8420   0.723330  0.006778  131.6578
Mars      49.4826  1.8498  286.3978   1.523688  0.093396  321.9965

Jupiter  100.3561  1.3036  273.8194   5.20256   0.048482   85.5238
Saturn   113.5787  2.4890  339.2884   9.55475   0.055580  198.4741
Uranus    73.9510  0.7732   96.5529  19.18176   0.047292  101.0460
Neptune  131.6737  1.7709  272.8675  30.05814   0.008598  239.0063
```

11. The heliocentric positions of the planets

The heliocentric ecliptic coordinates of the planets are computed in the same way as we computed the geocentric ecliptic coordinates of the Moon: first we compute E, the eccentric anomaly. Several planetary orbits have quite high eccentricities, which means we must use the iteration formula to obtain an accurate value of E. When we know E, we compute, as earlier, the distance r ("radius vector") and the true anomaly, v. Then we compute ecliptic rectangular (x,y,z) coordinates as we did for the Moon. Since the Moon orbits the Earth, this position of the Moon was geocentric. The planets though orbit the Sun, which means we get heliocentric positions instead. Also the semi-major axis, a, and the distance, r, which was given in Earth radii for the Moon, are given in astronomical units for the planets, where one astronomical unit is 149.6 million kilometers.

Let's do this for Mercury on our test date: the first approximation of E is 81.3464_deg, and following iterations yield 81.1572_deg, 81.1572_deg .... From this we find:

```
r = 0.374862
v = 93.0727_deg
```

Mercury's heliocentric ecliptic rectangular coordinates become:

```
x = -0.367821
y = +0.061084
z = +0.038699
```

Convert to spherical coordinates:

```
lon = 170.5709_deg
lat = +5.9255_deg
r   = 0.374862
```

The Astronomical Almanac gives these figures:

```
lon = 170.5701_deg
lat = +5.9258_deg
r   = 0.374856
```

The agreement is almost perfect! The discrepancy is only a few arc seconds. This is because it's quite easy to get an accurate position for Mercury: it's close to the Sun where the Sun's gravitational field is strongest, and therefore perturbations from the other planets will be smallest for Mercury.

If we compute the heliocentric longitude, latitude and distance for the other planets from their orbital elements, we get:

	Heliocentric		
	longitude lon	latitude lat	distance r
Mercury	170.5709_deg	+5.9255_deg	0.374862
Venus	263.6570_deg	-0.4180_deg	0.726607
Mars	290.6297_deg	-1.6203_deg	1.417194
Jupiter	105.2543_deg	+0.1113_deg	5.19508
Saturn	289.4523_deg	+0.1792_deg	10.06118
Uranus	276.7999_deg	-0.3003_deg	19.39628
Neptune	282.7192_deg	+0.8575_deg	30.19284

For e.g. Saturn, the Astronomical Almanac says:

```
lon = 289.3864_deg
lat = +0.1816_deg
r   = 10.01850
```

The difference is here much larger! For Mercury our discrepancy was only a few arc seconds, but for Saturn it's up to four arc minutes! And still we've been lucky, since sometimes the discrepancy can be up to one full degree for Saturn. This is the planet that's perturbed most severely, mostly by Jupiter.

12. Higher accuracy - perturbations

To reach our aim of a final accuracy of 1-2 arc minutes, we must compute Jupiter's and Saturn's perturbations on each other, and their perturbations on Uranus. The perturbations on, and by, other planets can be ignored, with our aim for 1-2 arcmin accuracy.

First we need three fundamental arguments:

```
Jupiters mean anomaly:   Mj
Saturn   mean anomaly:   Ms
Uranus   mean anomaly:   Mu
```

Then these terms should be added to Jupiter's heliocentric longitude:

```
-0.332_deg * sin(2*Mj - 5*Ms - 67.6_deg)
-0.056_deg * sin(2*Mj - 2*Ms + 21_deg)
+0.042_deg * sin(3*Mj - 5*Ms + 21_deg)
-0.036_deg * sin(Mj - 2*Ms)
+0.022_deg * cos(Mj - Ms)
+0.023_deg * sin(2*Mj - 3*Ms + 52_deg)
-0.016_deg * sin(Mj - 5*Ms - 69_deg)
```

For Saturn we must correct both the longitude and latitude. Add this to Saturn's heliocentric longitude:

```
+0.812_deg * sin(2*Mj - 5*Ms - 67.6_deg)
-0.229_deg * cos(2*Mj - 4*Ms - 2_deg)
+0.119_deg * sin(Mj - 2*Ms - 3_deg)
+0.046_deg * sin(2*Mj - 6*Ms - 69_deg)
+0.014_deg * sin(Mj - 3*Ms + 32_deg)
```

and to Saturn's heliocentric latitude these terms should be added:

```
-0.020_deg * cos(2*Mj - 4*Ms - 2_deg)
+0.018_deg * sin(2*Mj - 6*Ms - 49_deg)
```

Finally, add this to Uranus heliocentric longitude:

```
+0.040_deg * sin(Ms - 2*Mu + 6_deg)
+0.035_deg * sin(Ms - 3*Mu + 33_deg)
-0.015_deg * sin(Mj - Mu + 20_deg)
```

The perturbation terms above are all terms having an amplitude of 0.01 degrees or more. We ignore all perturbations in the distances of the planets, since a modest perturbation in distance won't affect the apparent position very much.

The largest perturbation term, "the grand Jupiter-Saturn term" is the perturbation in longitude with the largest amplitude in both Jupiter and Saturn. Its period is 918 years, and its amplitude is a large part of a degree for both Jupiter and Saturn. There is also a "grand Saturn-Uranus term", which has a period of 560 years and an amplitude of 0.035 degrees for Uranus, but less than 0.01 degrees for Saturn. Other included terms have periods between 14 and 100 years. Finally we should mention the "grand Uranus-Neptune term", which has a period if 4200 years and an amplitude of almost one degree. It's not included in our perturbation terms here, instead its effects have been included in the orbital elements for Uranus and Neptune. This is why the mean distances of Uranus and Neptune are varying.

If we compute the perturbations for our test date, we get:

```
    Mj = 85.5238_deg     Ms = 198.4741_deg     Mu = 101.0460:
```

Perturbations in Jupiter's longitude:

```
    + 0.0637_deg - 0.0236_deg + 0.0038_deg - 0.0270_deg - 0.0086_deg
    - 0.0049_deg - 0.0155_deg  =  -0.0120_deg

Jupiter's heliocentric longitude, with perturbations:    105.2423_deg
The Astronomical Almanac says:                           105.2603_deg
```

Perturbations in Saturn's longitude:

```
    -0.1560_deg + 0.0206_deg + 0.0850_deg - 0.0070_deg - 0.0124_deg
    = - 0.0699_deg
```

Perturbations in Saturn's latitude:

```
    +0.0018_deg + 0.0035_deg = +0.0053_deg

Saturns position, with perturbations:      289.3824_deg  +0.1845_deg
The Astronomical Almanac says:             289.3864_deg  +0.1816_deg
```

Perturbations in Uranus' longitude:

```
    +0.0017_deg - 0.0332_deg - 0.0012_deg = -0.0327_deg

Uranus heliocentric longitude, with perturbations:       276.7672_deg
The Astronomical Almanac says:                           276.7706_deg
```

13. Precession

The planetary positions computed here are for "the epoch of the day", i.e. relative to the celestial equator and ecliptic at the moment. Sometimes you need to use some other epoch, e.g. some standard epoch like 1950.0 or 2000.0. Due to our modest accuracy requirement of 1-2 arc minutes, we need not distinguish J2000.0 from B2000.0, it's enough to simply use 2000.0.

We will simplify the precession correction further by doing it in eliptic coordinates: the correction is simply done by adding

```
    3.82394E-5_deg * ( 365.2422 * ( epoch - 2000.0 ) - d )
```

to the ecliptic longitude. We ignore precession in ecliptic latitude. "epoch" is the epoch we wish to precess to, and "d" is the "day number" we used when computing our planetary positions.

Example: if we wish to precess computations done at our test date 19 April 1990, when d = -3543, we add the quantity below (degrees) to the ecliptic longitude:

```
    3.82394E-5_deg * ( 365.2422 * ( 2000.0 - 2000.0 ) - (-3543) ) =
    = 0.1355_deg
```

So we simply add 0.1355_deg to our ecliptic longitude to get the position at 2000.0.

14. Geocentric positions of the planets

To convert the planets' heliocentric positions to geocentric positions, we simply add the Sun's rectangular (x,y,z) coordinates to the rectangular (x,y,z) heliocentric coordinates of the planet:

Let's do this for Mercury on our test date - we add the x, y and z coordinates separately:

```
    xsun  = +0.881048   ysun  = +0.482098   zsun  = 0.0
    xplan = -0.367821   yplan = +0.061084   zplan = +0.038699
-----------------------------------------------------------------
    xgeoc = +0.513227   ygeoc = +0.543182   zgeoc = +0.038699
```

Now we have rectangular geocentric coordinates of Mercury. If we wish, we can convert this to spherical coordinates - then we get geocentric ecliptic longitude and latitude. This is useful if we want to precess the position to some other epoch: we then simply add the approproate precessional value to the longitude. Then we can convert back to rectangular coordinates.

But for the moment we want the "epoch of the day": let's rotate the x,y,z, coordinates around the X axis, as described earlier. Then we'll get equatorial rectangular geocentric (whew!) coordinates:

```
    xequat = +0.513227  yequat = +0.482961  zequat = 0.251582
```

We can convert these coordinates to spherical coordinates, and then we'll (finally!) get geocentric Right Ascension, Declination and distance for Mercury:

```
    RA = 43.2598_deg   Decl = +19.6460_deg   R = 0.748296
```

Note that the distance now is different. This is quite natural since the distance now is from the Earth and not, as earlier, from the Sun.

Let's conclude by checking the values given by the Astronomical Almanac:

```
    RA = 43.2535_deg   Decl = +19.6458_deg   R = 0.748262
```

15. The elongation and physical ephemerides of the planets

When we finally have completed our computation of the heliocentric and geocentric coordinates of the planets, it could also be interesting to know what the planet will look like. How large will it appear? What are its phase and magnitude (brightness)? These computations are much simpler than the computations of the positions.

Let's start by computing the apparent diameter of the planet:

```
d = d0 / R
```

R is the planet's geocentric distance in astronomical units, and d is the planet's apparent diameter at a distance of 1 astronomical unit. d0 is of course different for each planet. The values below are given in seconds of arc. Some planets have different equatorial and polar diameters:

```
Mercury     6.74"
Venus      16.92"
Earth      17.59" equu    17.53" pol
Mars        9.36" equ      9.28" pol
Jupiter   196.94" equ    185.08" pol
Saturn    165.6"  equ    150.8"  pol
Uranus     65.8"  equ     62.1"  pol
Neptune    62.2"  equ     60.9"  pol
```

The Sun's apparent diameter at 1 astronomical unit is 1919.26". The Moon's apparent diameter is:

```
d = 1873.7" * 60 / r
```

where r is the Moon's distance in Earth radii.

Two other quantities we'd like to know are the phase angle and the elongation.

The phase angle tells us the phase: if it's zero the planet appears "full", if it's 90 degrees it appears "half", and if it's 180 degrees it appears "new". Only the Moon and the inferior planets (i.e. Mercury and Venus) can have phase angles exceeding about 50 degrees.

The elongation is the apparent angular distance of the planet from the Sun. If the elongation is smaller than about 20 degrees, the planet is hard to observe, and if it's smaller than about 10 degrees it's usually not possible to observe the planet.

To compute phase angle and elongation we need to know the planet's heliocentric distance, r, its geocentric distance, R, and the distance to the Sun, s. Now we can compute the phase angle, FV, and the elongation, elong:

```
elong = acos( ( s*s + R*R - r*r ) / (2*s*R) )

FV    = acos( ( r*r + R*R - s*s ) / (2*r*R) )
```

When we know the phase angle, we can easily compute the phase:

```
phase  =  ( 1 + cos(FV) ) / 2  =  hav(180_deg - FV)
```

hav is the "haversine" function. The "haversine" (or "half versine") is an old and now obsolete trigonometric function. It's defined as:

```
hav(x)  =  ( 1 - cos(x) ) / 2  =  sin^2 (x/2)
```

As usual we must use a different procedure for the Moon. Since the Moon is so close to the Earth, the procedure above would introduce too big errors. Instead we use the Moon's ecliptic longitude and latitude, mlon and mlat, and the Sun's ecliptic longitude, mlon, to compute first the elongation, then the phase angle, of the Moon:

```
elong = acos( cos(slon - mlon) * cos(mlat) )

FV = 180_deg - elong
```

Finally we'll compute the magnitude (or brightness) of the planets. Here we need to use a formula that's different for each planet. The phase angle, FV, is in degrees:

```
Mercury:   -0.36 + 5*log10(r*R) + 0.027 * FV + 2.2E-13 * FV**6
Venus:     -4.34 + 5*log10(r*R) + 0.013 * FV + 4.2E-7  * FV**3
Mars:      -1.51 + 5*log10(r*R) + 0.016 * FV
Jupiter:   -9.25 + 5*log10(r*R) + 0.014 * FV
Saturn:    -9.0  + 5*log10(r*R) + 0.044 * FV + ring_magn
Uranus:    -7.15 + 5*log10(r*R) + 0.001 * FV
Neptune:   -6.90 + 5*log10(r*R) + 0.001 * FV
```

** is the power operator, thus FV**6 is the phase angle (in degrees) raised to the sixth power. If FV is 150 degrees, then FV**6 becomes ca 1.14E+13, which is a quite large number.

Saturn needs special treatment due to its rings: when Saturn's rings are "open" then Saturn will appear much brighter than when we view Saturn's rings edgewise. We'll compute ring_mang like this:

```
ring_magn = -2.6 * sin(abs(B)) + 1.2 * (sin(B))**2
```

Here B is the tilt of Saturn's rings which we also need to compute. Then we start with Saturn's geocentric ecliptic longitude and latitude (los, las) which we've already computed. We also need the tilt of the rings to the ecliptic, ir, and the "ascending node" of the plane of the rings, Nr:

```
ir = 28.06_deg
Nr = 169.51_deg + 3.82E-5_deg * d
```

Here d is our "day number" which we've used so many times before. For our test date d = -3543. Now let's compute the tilt of the rings:

```
B = asin( sin(las) * cos(ir) - cos(las) * sin(ir) * sin(los-Nr) )
```

This concludes our computation of the magnitudes of the planets.

16. The positions of comets. Comet Encke and Levy.

If you want to compute the position of a comet or an asteroid, you must have access to orbital elements that still are valid. One set of orbital elements isn't valid forever. For instance if you try to use the 1986 orbital elements of comet Halley to compute its appearance in either 1910 or 2061, you'll get very large errors in your computed positions - sometimes the errors will be 90 degrees or more.

Comets will usually have a new set of orbital elements computed for each perihelion. The comets are perturbed most severely when they're close to aphelion, far away from the gravity of the Sun but maybe much closer to Jupiter, Saturn, Uranus or Neptune. When the comet is passing through the inner solar system, the perturbations are usually so small that the same set of orbital elements can be used for the entire apparition.

Orbital elements for an asteroid should preferably not be more than about one year old. If your accuracy requirements are lower, you can of course use older elements. If you use orbital elements that are five years old for a main-belt asteroid, then your computed positions can be several degrees in error. If the orbital elements are less than one year old, the errors usually stay below approximately one arc minute, for a main-belt asteroid.

If you have access to valid orbital elements for a comet or an asteroid, proceed as below to compute its position at some date:

1. If necessary, precess the angular elements N,w,i to the epoch of today. The simples way to do this is to add the precession angle to N, the longitude of the ascending node. This method is approximate, but it's good enough for our accuracy aim of 1-2 arc minutes.

2. Compute the day number for the time or perihelion, call it D. Then compute the number of days since perihelion, d - D (before perihelion this number is of course negative).

3. If the orbit is elliptical, compute the Mean Anomaly, M. Then compute r, the heliocentric distance, and v, the true anomaly.

4. If the orbit is a parabola, or close to a parabola (the eccentricity is 1.0 or nearly 1.0), then the algorithms for elliptical orbits will break down. Then use another algorithm, presented below, to compute r, the heliocentric distance, and v, the true anomaly, for near-parabolic orbits.

5. When you know r and v, proceed as with the planets: compute first the heliocentric, then the geocentric, position.

6. If needed, precess the final position to the desired epoch, e.g. 2000.0

A quantity we'll encounter here is Gauss' gravitational constant, k. This constant links the Sun's mass with our time unit (the day) and the length unit (the astronomical unit). The EXACT value of Gauss' gravitational constant k is:

```
    k = 0.01720209895   (exactly!)
```

If the orbit is elliptical, and if the perihelion distance, q, is given instead of the mean distance, a, we start by computing the mean distance a from the perihelion distance q and the eccentricity e:

```
a = q / (1 - e)
```

Now we compute the Mean Anomaly, M:

```
M = (180_deg/pi) * (d - D) * k / (a ** 1.5)

a ** 1.5 is most easily computed as:  sqrt(a*a*a)
```

Now we know the Mean Anomaly, M. We proceed as for a planetary orbit by computing E, the eccentric anomaly. Since comet and asteroid orbits often have high eccentricities, we must use the iteration formula given earlier, and be sure to iterate until we get convergence for the value of E.

The orbital period for a comet or an asteroid in elliptic orbit is (P in days):

```
P = 2 * pi * (a ** 1.5) / k
```

If the comet's orbit is a parabola, the algorithm for elliptic orbits will break down: the semi-major axis and the orbital period will be infinite, and the Mean Anomaly will be zero. Then we must proceed in a different way. For a parabolic orbit we start by computing the quantities a, b and w (where a is not at all related to a for an elliptic orbit):

```
a = 1.5 * (d - D) * k / sqrt(2 * q*q*q)

b = sqrt( 1 + a*a )

w = cbrt(b + a) - cbrt(b - a)
```

cbrt is the Cubic Root function. Finally we compute the true anomaly, v, and the heliocentric distiance, r:

```
v = 2 * atan(w)
r = q * ( 1 + w*w )
```

From here we can proceed as usual.

Finally we have the case that's most common for newly discovered comets: the orbit isn't an exact parabola, but very nearly so. It's eccentricity is slightly below, or slightly above, one. The algorithm presented here can be used for eccentricities between about 0.98 and 1.02. If the eccentricity is smaller than 0.98 the elliptic algorithm should be used instead. No known comet has an eccentricity exceeding 1.02.

As for the purely parabolic orbit, we start by computing the time since perihelion in days, d - D, and the perihelion distance, q. We also need to know the eccentricity, e. Then we can proceed as:

```
a = 0.75 * (d - D) * k * sqrt( (1 + e) / (q*q*q) )
b = sqrt( 1 + a*a )
W = cbrt(b + a) - cbrt(b - a)
f = (1 - e) / (1 + e)

a1 = (2/3) + (2/5) * W*W
a2 = (7/5) + (33/35) * W*W + (37/175) * W**4
```

```
a3 = W*W * ( (432/175) + (956/1125) * W*W + (84/1575) * W**4 )

C = W*W / (1 + W*W)
g = f * C*C
w = W * ( 1 + f * C * ( a1 + a2*g + a3*g*g ) )

v = 2 * atan(w)
r = q * ( 1 + w*w ) / ( 1 + w*w * f )
```

This algorithm yields the true anomaly, v, and the heliocentric distance, r, for a nearly-parabolic orbit.

Now it's time for a practical example. Let's select two of the comets that were seen in the autumn of 1990: Comet Encke, a well-known periodic comet, and comet Levy, which was easily seen towards a dark sky in the autumn of 1990. When passing the inner solar system, the orbit of comet Levy was slightly hyperbolic.

According the the Handbook of the British Astronomical Association the orbital elements for comet Encke in 1990 are:

```
T = 1990 Oct 28.54502 TDT
e = 0.8502196
q = 0.3308858
w = 186.24444_deg
N = 334.04096_deg   1950.0
i =  11.93911_deg
```

The orbital elements for comet Levy are (BAA Circular 704):

```
T = 1990 Oct 24.6954 ET
e = 1.000270
q = 0.93858
w = 242.6797_deg
N = 138.6637_deg    1950.0
i = 131.5856_deg
```

Let's also choose another test date, when both these comets were visible: 1990 Aug 22, 0t UT, which yields a "day number" d = -3418.0

Now we compute the day numbers at perihelion for these two comets. We get for comet Encke:

```
D = -3350.45498     d - D = -67.54502
```

and for comet Levy:

```
D = -3354.3046      d - D = -63.6954
```

We'll continue by computing the Mean Anomaly for comet Encke:

```
M = -20.2751_deg = 339.7249_deg
```

The first approximation plus successive approximation for the Eccentric anomaly, E, becomes (degrees):

```
E = 309.3811  293.5105  295.8474  295.9061  295.9061_deg ....
```

Here we clearly see the great need for iteration: the initial approximation differs from the final value by 14 degrees. Finally we compute the true anomaly, v, and heliocentric distance, r, for comet Encke:

```
v = 228.8837_deg
r = 1.3885
```

Now it's time for comet Levy: we'll compute the true anomaly, v, and the heliocentric distance, r, for Levy in two different ways. First we'll pretend that the orbit of Levy is an exact parabola. We get:

```
a = -1.2780823   b = 1.6228045  w = -0.7250189

v = -71.8856_deg
r = 1.431947
```

Then we repeat the computation but accounts for the fact that Levy's orbit deviates slightly from a parabola. We get:

```
a = -1.2781686   b = 1.6228724   W = -0.7250566
c = 2.9022000    f = -1.3498E-4  g = -1.60258E-5
a1= 0.8769495    a2= 1.9540987   a3= 1.5403455
w = -0.7250270

v = -71.8863_deg
r = 1.432059
```

The difference is small in this case - only 0.0007 degrees or 2.5 arc seconds in true anomaly, and 0.000112 a.u. in heliocentric distance. Here it would have been sufficient to treat Levy's orbit as an exact parabola.

Now we know the true anomaly, v, and the heliocentric distance, r, for both Encke and Levy. Next we proceed by precessing N, the longitude of the ascending node, from 1950.0 to the "epoch of the day". Let's compute the precession angle from 1950.0 to 1990 Aug 22:

```
prec = 3.82394E-5_deg * ( 365.2422 * ( 1950.0 - 2000.0 ) - (-3418) )
prec = -0.5676_deg
```

To precess from 1990 Aug 22 to 1950.0, we should add this angle to N. But now we want to do the opposite: precess from 1950.0 to 1990 Aug 22, therefore we must instead subtract this angle:

For comet Encke we get:

```
N = 334.04096_deg - (-0.5676_deg) = 334.60856_deg
```

and for comet Levy we get:

```
N = 138.6637_deg - (-0.5676_deg) = 139.2313_deg
```

Using this modified value for N we proceed just like for the planets. I won't repeat the details, but merely state some intermediate and final results:

```
Sun's position:    x = -0.863890   y = +0.526123

Heliocentric:    Encke           Levy

x              +1.195087       +1.169908
y              +0.666455       -0.807922
z              +0.235663       +0.171375

Geoc., eclipt.:  Encke           Levy

x              +0.331197       +0.306018
y              +1.192579       -0.281799
z              +0.235663       +0.171375

Geoc., equat.:   Encke           Levy

x              +0.331197       +0.306018
y              +1.000414       -0.326716
z              +0.690619       +0.045133

RA              71.6824_deg     313.1264_deg
Decl           +33.2390_deg      +5.7572_deg
R                1.259950         0.449919
```

These positions are for the "epoch of the day". If you want positions for some standard epoch, e.g. 2000.0, these positions must be precessed to that epoch.

Finally some notes about computing the magnitude of a comet. To accurately predict a comet's magnitude is usually hard and sometimes impossible. It's fairly common that a magnitude prediction is off by 1-2 magnitudes or even more. For comet Levy the magnitude formula looked like this:

```
m = 4.0 * 5*log10(R) + 10*log10(r)
```

where R is the geocentric distance and r the heliocentric distance. The general case is:

```
m = G * 5*log10(R) + H*log10(r)
```

where H usually is around 10. If H is unknown, it's usually assumed to be 10. Each comet has it's own G and H.

Some comets have a different magnitude formula. One good example is comet Encke, where the magnitude formula looks like this:

```
m1 = 10.8 + 5*log10(R) + 3.55 * ( r**1.8 - 1 )
```

"m1" refers to the total magnitude of the comet. There is another cometary magnitude, "m2", which refers to the magnitude of the nucleus of the comet. The magnitude formula for Encke's m2 magnitude looks like this:

```
m2 = 14.5 + 5*log10(R) + 5*log10(r) + 0.03*FV
```

Here FV is the phase angle. This kind of magnitude formula looks very much like the magnitude formula of asteroids, for a very good reason: when a comet is far away form the Sun, no gases are evaporated from the surface of the comet. Then the comet has no tail (of course) and no coma, only a nucleus. Which means the comet then behaves much like an asteroid.

During the last few years it's become increasingly obvious that comets and asteroids often are similar kinds of solar-system objects. The asteroid (2060) Chiron has displayed cometary activity and is now also considered a comet. And in some cases comets that have "disappeared" have been re-discovered as asteroids! Apparently they "ran out of gas" and what remains of the former comet is only rock, i.e. an asteroid.

How to compute rise/set times and altitude above horizon

By **Paul Schlyter, Stockholm, Sweden**
email: pausch@stjarnhimlen.se or WWW: http://stjarnhimlen.se/

Break out of a frame

How to compute planetary positions
Tutorial with numerical test cases

1. Computing the Sun's altitude above the horizon

First we compute the Sun's RA and Decl for this moment, as outlined earlier. Now we need to know our Local Sidereal Time. We start by computing the sidereal time at Greenwich at 00:00 Universal Time, let's call this quantity GMST0:

```
GMST0 = L + 180
```

L is the Sun's mean longitude, which we compute as:

```
L = M + w
```

Note that we express GMST in degrees here to simplify the computations. 360 degrees of course corresponds to 24 hours, i.e. each hour corresponds to 15 degrees.

Now we can compute our Local Sidereal Time (LST):

```
LST = GMST0 + UT*15.0 + long
```

UT is the Universal Time, expressed in hours+decimals, the remaining quantities are expressed in degrees. To convert UT to degrees we must multiply it by 15 above. long is our local longitude in degrees, where east longitude counts as positive, and west longitude as negative. (this is according to the geographic standard, and the recent astronomical standard; if you prefer to use the older astronomical standard where west longitude counts as positive, then you must change the '+' in front of 'long' to a '-' above).

Next let's compute the Sun's Local Hour Angle (LHA), i.e. the angle the Earth has turned since the Sun last was in the south:

```
LHA = LST - RA
```

A negative hour angle means the Sun hasn't been in the south yet, this day. The angle -10 degrees is of course the same as 350 degrees, i.e. adding or subtracting even multiples of 360 degrees does not change the angle.

We also need to know our latitude (lat), where north latitude counts as poisitive and south latitude as negative. Now we can compute the Sun's altitude above the horizon:

```
sin(h) = sin(lat) * sin(Decl) + cos(lat) * cos(Decl) * cos(LHA)
```

We compute sin(h), and then take the arcsine of this to get h, the Sun's altitude above the horizon.

2. Computing the Sun's rise/set times.

This is really the inverse of the previous problem, where we computed the Sun's altitude at a specific moment. Now we want to know at which moment the Sun reaches a specific altitude.

First we must decide which altitude we're interested in:

h = 0 degrees: Center of Sun's disk touches a mathematical horizon
h = -0.25 degrees: Sun's upper limb touches a mathematical horizon
h = -0.583 degrees: Center of Sun's disk touches the horizon; atmospheric refraction accounted for
h = -0.833 degrees: Sun's upper limb touches the horizon; atmospheric refraction accounted for
h = -6 degrees: Civil twilight (one can no longer read outside without artificial illumination)
h = -12 degrees: Nautical twilight (navigation using a sea horizon no longer possible)
h = -15 degrees: Amateur astronomical twilight (the sky is dark enough for most astronomical observations)
h = -18 degrees: Astronomical twilight (the sky is completely dark)

As you can see, there are several altitides to choose among. In most countries an altitude of -0.833 degrees is used to compute sunrise/set times (Sun's upper limb touches the horizon; atmospheric refraction accounted for). One exception is the Swedish national alamancs, which use -0.583 degrees (Center of Sun's disk touches the horizon; atmospheric refraction accounted for) - however my own Swedish almanac Stjärnhimlen ("The Starry Sky") uses the international convention of -0.833 degrees.

When we've decided on some value for the altitude above the horizon, we start by computing the Sun's RA at noon local time. When the Local Sidereal Time equals the Sun's RA, then the Sun is in the south:

```
LST  =  RA
```

which yields:

```
GMST0 + UT*15.0 + long  = RA
```

Since we know GMST, long, and RA, it's now a simple matter to compute UT (GMST0 should also be computed at noon local time):

```
UT_Sun_in_south = ( RA - GMST0 - long ) / 15.0
```

Now we're going to compute the Sun's Local Hour Angle (LHA) at rise/set (or at twilight, if we've decided to compute the time of twilight). This is the angle the Earth must turn from sunrise to noon, or from noon to sunset:

```
                 sin(h) - sin(lat)*sin(Decl)
    cos(LHA) = -----------------------------
                    cos(lat) * cos(Decl)
```

If cos(LHA) is less than -1.0, then the Sun is always above our altitude limit. If we were computing rise/set times, the Sun is then aboute the horizon continuously; we have Midnight Sun. Or, if we computed a twilight, then the sky never gets dark (a good example is Stockholm, Sweden, at midsummer midnight: the Sun then only reaches about 7 degrees below the horizon: there will be civil twilight, but never nautical or astronomical twilight, at midsummer in Stockholm).

If cos(LHA) is greater than +1.0, then the Sun is always below our altitude limit. One example is midwinter in the arctics, when the Sun never gets above the horizon.

If cos(LHA) is between +1.0 and -1.0, then we take the arccos to find LHA. Convert from degrees to hours by dividing by 15.0

Now, if we add LHA to UT_Sun_in_south, we get the time of sunset. If we subtract LHA from UT_Sun_in_south, we get the time of sunrise.

Finally, we convert UT to our local time.

3. Higher accuracy: iteration

The method outlined abouve only gives an approximate value of the Sun's rise/set times. The error rarely exceeds one or two minutes, but at high latitudes, when the Midnight Sun soon will start or just has ended, the errors may be much larger. If you want higher accuracy, you must then iterate, and you must do a separate iteration for sunrise and sunset:

a) Compute sunrise or sunset as above, with one exception: to convert LHA from degrees to hours, divide by 15.04107 instead of 15.0 (this accounts for the difference between the solar day and the sidereal day. You should *only* use 15.04107 if you intend to iterate; if you don't want to iterate, use 15.0 as before since that will give an approximate correction for the Earth's orbital motion during the day).

b) Re-do the computation but compute the Sun's RA and Decl, and also GMST0, for the moment of sunrise or sunset last computed.

c) Iterate b) until the computed sunrise or sunset no longer changes significantly. Usually 2 iterations are enough, in rare cases 3 or 4 iterations may be needed.

d) Make sure you iterate towards the sunrise or sunset you want to compute, and not a sunrise or sunset one day earlier or later. If the computed rise or set time is, say, -0.5 hours local time, this means that it really happens at 23:30 *the day before*. If you get a value exceeding 24 hours local time, it means it happens *the day after*. If this is what you want, fine, otherwise add or subtract 24 hours. This only becomes a problem when there soon will be, or just has been, Midnight Sun.

e) Above the arctic circle, there are occasionally two sunrises, or two sunsets, during the same calendar day. Also there are days when the Sun only sets, or only rises, or neither rises nor sets. Pay attention to this if you don't want to miss any sunrise or sunset in your computations.

f) If you compute twilight instead of rise/set times, e) applies to the "twilight arctic circle". The normal arctic circle is situated at 66.7 deg latitude N and S (65.9 deg if one accounts for atmospheric refraction and the size of the solar disk). The "twilight arctic circle" is situated 6, 12, 15 or 18 deg closer to the equator, i.e. at latitude 60.7, 54.7, 51.7 or 48.7 degrees, depending on which twilight you're computing.

4. Computing the Moon's rise/set times.

This is really done the same way as the Sun's rise/set times, the only difference being that you should compute the RA and Decl for the Moon and not for the Sun. However, the Moon moves quickly and its rise/set times may change one or even two hours from one day to the next. If you don't iterate the Moon's rise/set times, you may get results which are in error by up to an hour, or more.

Another thing to consider is the lunar parallax, which affects the Moon's rise/set time by several minutes or more. One way to deal with the lunar parallax is to always use the Moon's topocentric RA and Decl. Another, simpler, way is to use the Moon's geocentric RA and Decl and instead adjust h, the rise/set altitude, by decreasing it by m_par, the lunar parallax. If you want to compute rise/set times for the Moon's upper limb rather than the center of the Moon's disk, you also need to compute m_sd, the semi-diameter or apparent radius of the Moon's disk in the sky. Note that the Moon's upper limb may for some lunar phases and circumstances be on the dark part of the Moon's disk

Thus you choose your h for Moon rise/set computation like this:

h = -m_par: Center of Moon's disk touches a mathematical horizon
h = -(m_par+m_sd): Moon's upper limb touches a mathematical horizon
h = -0.583 degrees - m_par: Center of Moon's disk touches the horizon; atmospheric refraction accounted for
h = -0.583 degrees - (m_par+m_sd): Moon's upper limb touches the horizon; atmospheric refraction accounted for

Yet another thing to consider: the Sun is always in the south near 12:00 local time, but the Moon may be in the south at any time of the day (or night). This means you must pay more attention that you're really iterating towards the rise or set time you want, and not some rise/set time one day earlier, or later.

Since the Moon rises and sets on the average 50 minutes later each day, there usually will be one day each month when the Moon never rises, and another day when it never sets. You must have this in mind when iterating your rise/set times, otherwise your program may easily get caught into an infinite loop when it tries to force e.g. a rise time between 00:00 and 24:00 local time on a day when the Moon never rises.

At high latitudes the Moon occasionally rises, or sets, twice on a single calendar day. This may happen above the "lunar arctic circle", which moves between 61.5 and 71.9 deg latitude during the 18-year period of the motion of the lunar nodes. You may want to pay attention to this.

Yes, computing the Moon's rise/set times is unfortunately messy, much due to its quick orbital motion around the Earth.

5. Computing rise/set times for other celestial bodies.

This is done the same way as for the Sun, with some differences:

a) Compute the RA and Decl for that body instead of for the Sun. If the body is a star, get its RA and Decl from a suitable star catalog.

b) GMST0 is still needed, so you should compute the Sun's mean longitude.

c) *Always* use 15.04107 instead of 15.0 when converting LHA from degrees to hours.

Since the planets move much slower than the Moon, and the stars hardly move at all, one need not iterate. If one wants high accuracy, one may find it worthwhile to iterate the rise/set times for Mercury, Venus and Mars (these are the planets that move most quickly).