```
'   A series of astronomical functions which may be
'   useful. These are all 'user defined functions'.
'   This means that you can paste them into spreadsheets
'   just like the normal functions - see Insert|function,
'   you can even use the function wizard.

'   The disadvantage
'   of Excel's 'user defined functions' is that they
'   can only return a single value, and the function cannot alter
'   the properties of the worksheet. Arguments you pass to
'   the VBA functions you define are passed 'by value'.

'   However, VBA defaults to 'passing arguments by reference'
'   when a function is called from another VBA function! This
'   can lead to a function giving a different answer when
'   called in the VBA module compared with when called in the
'   spreadsheet. Use the ByVal keyword to tag arguments you
'   change later in functions. See smoon() for an example.

'   define some numerical constants - these are not
'   accessible in the spreadsheet.

Public Const pi As Double = 3.14159265358979
Public Const tpi As Double = 6.28318530717958
Public Const degs  As Double = 57.2957795130823
Public Const rads As Double = 1.74532925199433E-02

'   The trig formulas working in degrees. This just
'   makes the spreadsheet formulas a bit easier to
'   read. DegAtan2() has had the arguments swapped
'   from the Excel order, so the order matches most
'   textbooks

Function DegSin(x As Double) As Double
    DegSin = Sin(rads * x)
End Function

Function DegCos(x As Double) As Double
    DegCos = Cos(rads * x)
End Function

Function DegTan(x As Double) As Double
    DegTan = Tan(rads * x)
End Function

Function DegArcsin(x As Double) As Double
    DegArcsin = degs * Application.Asin(x)
End Function

Function DegArccos(x As Double) As Double
     DegArccos = degs * Application.Acos(x)
End Function

Function DegArctan(x As Double) As Double
    DegArctan = degs * Atn(x)
End Function

Function DegAtan2(y As Double, x As Double) As Double
'   this returns the angle in the range 0 to 360
'   instead of -180 to 180 - and swaps the arguments
'   This format matches Meeus and Duffett-Smith
    DegAtan2 = degs * Application.Atan2(x, y)
    If DegAtan2 < 0 Then DegAtan2 = DegAtan2 + 360
End Function

Private Function range2pi(x)
'
```

```
'   returns an angle x in the range 0 to two pi rads
'   This function is not available in the spreadsheet
'
range2pi = x - tpi * Int(x / tpi)
End Function

Private Function range360(x)
'
'   returns an angle x in the range 0 to 360
'   used to prevent the huge values of degrees
'   that you get from mean longitude formulas
'
'   this function is private to this module,
'   you won't find it in the Function Wizard,
'   and you can't use it on a spreadsheet.
'   If you want it on the spreadsheet, just remove
'   the 'private' keyword above.
'
range360 = x - 360 * Int(x / 360)
End Function

Function degdecimal(d, m, s)
'   converts from dms format to ddd format
    degdecimal = d + m / 60 + s / 3600
End Function

'
'   calander functions. jday and jcentury work on the Julian day numbers.
'   day2000 and century2000 work on the days to J2000 to reduce the
'   number of significant figures needed
'

Function jday(year As Integer, month As Integer, day As Integer, hour As Integer, _
 min As Integer, sec As Double, Optional greg) As Double
 '  returns julian day number given date in gregorian calender (greg=1)
 '  or julian calendar (greg = 0). If greg ommited, then Gregorian is assumed.
    Dim a As Double
    Dim b As Integer
    a = 10000# * year + 100# * month + day
    If (a < -47120101) Then MsgBox "Warning: date too early for algorithm"
    If (IsMissing(greg)) Then greg = 1
    If (month <= 2) Then
        month = month + 12
        year = year - 1
    End If
    If (greg = 0) Then
        b = -2 + Fix((year + 4716) / 4) - 1179
    Else
        b = Fix(year / 400) - Fix(year / 100) + Fix(year / 4)
    End If
    a = 365# * year + 1720996.5
    jday = a + b + Fix(30.6001 * (month + 1)) + day + (hour + min / 60 + sec / 3600) / 24
End Function

Function jcentury(jd As Double) As Double
'   finds how many julian centuries since J2000 given
'    the julian day number. Not used below, I just add
'   a line into the subroutines which then take days
'   before J2000 as the time argument
    jcentury = (jd - 2451545) / 36525
End Function

Function day2000(year As Integer, month As Integer, day As Integer, hour As Integer, _
 min As Integer, sec As Double, Optional greg) As Double
 '  returns days before J2000.0 given date in gregorian calender (greg=1)
 '  or julian calendar (greg = 0). If you don't provide a value for greg,
 '  then assumed Gregorian calender
    Dim a As Double
```

```
    Dim b As Integer
    If (IsMissing(greg)) Then greg = 1
    a = 10000# * year + 100# * month + day
    If (month <= 2) Then
        month = month + 12
        year = year - 1
    End If
    If (greg = 0) Then
        b = -2 + Fix((year + 4716) / 4) - 1179
    Else
        b = Fix(year / 400) - Fix(year / 100) + Fix(year / 4)
    End If
    a = 365# * year - 730548.5
    day2000 = a + b + Fix(30.6001 * (month + 1)) + day + (hour + min / 60 + sec / 3600) / 24
End Function

Function century2000(day2000 As Double) As Double
'   finds how many julian centuries since J2000 given
'   the days before J2000
    century2000 = day2000 / 36525
End Function

'
'   Conversion to and from rectangular and polar coordinates.
'   X,Y,Z form a left handed set of axes, and r is the radius vector
'   of the point from the origin. Theta is the elevation angle of
'   r with the XY plane, and phi is the angle anti-clockwise from the
'   X axis and the projection of r in the X,Y plane.
'
'   in astronomical coordinate systems,
'
'   item    equatorial          ecliptic (helio or geo centric)
'   z       celestial pole      ecliptic pole
'   x,y     equatorial plane    ecliptic
'   theta   declination         latitude
'   phi     right ascension     longitude
'

Function rectangular(r As Double, theta As Double, phi As Double, _
 index As Integer) As Double
 '  takes spherical coordinates in degrees and returns the rectangular
 '  coordinate shown by index, 1 = x, 2 = y, 3 = z
 '
 '  x = r.cos(theta).cos(phi)
 '  y = r.cos(theta).sin(phi)
 '  z = r.sin(theta)
 '
    Dim r_cos_theta As Double
    r_cos_theta = r * DegCos(theta)
    Select Case index
        Case 1
            rectangular = r_cos_theta * DegCos(phi) 'returns x coord
        Case 2
            rectangular = r_cos_theta * DegSin(phi) 'returns y coord
        Case 3
            rectangular = r * DegSin(theta)         'returns z coord
    End Select
End Function

Function rlength(x As Double, y As Double, z As Double) As Double
'   returns radius vector given the rectangular coords
    rlength = Sqr(x * x + y * y + z * z)
End Function

Function spherical(x As Double, y As Double, z As Double, index As Integer) As Double
'
'   Takes the rectangular coordinates and returns the shperical
'   coordinate selected by index - 1 = r, 2 = theta, 3 = phi
```

```
'
'   r = sqrt(x*x + y*y + z*z)
'   tan(phi) = y/x - use atan2 to get in correct quadrant
'   tan(theta) = z/sqrt(x*x + y*y) - likewise
'
    Dim rho As Double
    rho = x * x + y * y
        Select Case index
        Case 1
            spherical = Sqr(rho + z * z)    'returns r
        Case 2
            rho = Sqr(rho)
            spherical = DegArctan(z / rho)    'returns theta
        Case 3
            rho = Sqr(rho)
            spherical = DegAtan2(y, x)      'returns phi
    End Select
End Function

'
'   returns the obliquity of the ecliptic in degrees given the number
'   of julian centuries from J2000
'
'   Most textbooks will give the IAU formula for the obliquity of
'   the ecliptic below;
'
'   obliquity = 23.43929111 - 46.8150"t - 0.00059"t^2 + 0.001813*t^3
'
'   as explained in Meeus or Numerical Recipes, it is more efficient and
'   accurate to use the nested brackets shown in the function. If you
'   multiply the brackets out, they come to the same.
'
Function obliquity(d As Double) As Double
    Dim t As Double
    t = d / 36525   'julian centuries since J2000.0
    obliquity = 23.43929111 - (46.815 + (0.00059 - 0.001813 * t) * t) * t / 3600#
End Function

'
'   functions for converting between equatorial and ecliptic
'   geocentric coordinates, both polar and rectangular coords
'


'
'   Converts geocentric ecliptic coordinates into geocentric equatorial
'   coordinates. Expects rectangular coordinates.
'
Function requatorial(x As Double, y As Double, z As Double, d As Double, _
 index As Integer) As Double
    Dim obl As Double
    obl = obliquity(d)
    Select Case index
        Case 1
            requatorial = x
        Case 2
            requatorial = y * DegCos(obl) - z * DegSin(obl)
        Case 3
            requatorial = y * DegSin(obl) + z * DegCos(obl)
    End Select
End Function
'
'   converts geocentric equatorial coordinates into geocentric ecliptic
'   coordinates. Expects rectangular coordinates.
'
Function recliptic(x As Double, y As Double, z As Double, d As Double, _
 index As Integer) As Double
    Dim obl As Double
    obl = obliquity(d)
```

```
    Select Case index
        Case 1
            recliptic = x
        Case 2
            recliptic = y * DegCos(obl) + z * DegSin(obl)
        Case 3
            recliptic = -y * DegSin(obl) + z * DegCos(obl)
    End Select
End Function

'
'   Converts geocentric ecliptic coordinates into geocentric equatorial
'   coordinates. Expects spherical coordinates.
'
Function sequatorial(r As Double, theta As Double, phi As Double, d As Double, _
 index As Integer) As Double
    Dim x As Double, y As Double, z As Double
    x = rectangular(r, theta, phi, 1)
    y = rectangular(r, theta, phi, 2)
    z = rectangular(r, theta, phi, 3)
    sequatorial = spherical(requatorial(x, y, z, d, 1), requatorial(x, y, z, d, 2), _
     requatorial(x, y, z, d, 3), index)
End Function

'   Converts geocentric equatorial coordinates into geocentric ecliptic
'   coordinates. Expects spherical coordinates.
'
Function secliptic(r As Double, theta As Double, phi As Double, d As Double, _
 index As Integer) As Double
    Dim x As Double, y As Double, z As Double
    x = rectangular(r, theta, phi, 1)
    y = rectangular(r, theta, phi, 2)
    z = rectangular(r, theta, phi, 3)
    secliptic = spherical(recliptic(x, y, z, d, 1), recliptic(x, y, z, d, 2), _
    recliptic(x, y, z, d, 3), index)
End Function
'
'   precession (approximate formula) from Meeus Algorithms p124
'   d1 is the epoch to precess from, d2 is the epoch to precess
'   to, and index selects ra or dec. The function takes optional
'   arguments dra and ddec to represent the proper motion of a
'   star in seconds of arc per year.
'
'   ra and dec must BOTH be in decimal degrees. This formula is
'   different to the one elsewhere on the Web site!
'
Function precess(d1 As Double, d2 As Double, dec As Double, _
                    ra As Double, index As Integer, _
                    Optional ddec, Optional dra) As Double
Dim m As Double, n As Double, t As Double
If (IsMissing(dra)) Then dra = 0
If (IsMissing(ddec)) Then ddec = 0
t = d1 / 36525          'years since J2000
m = 0.01281233333333 + 0.00000775 * t
n = 0.005567527777778 - 2.361111111111E-06 * t
t = (d2 - d1) / 365.25   'difference in julian _years_, not centuries!
Select Case index
    Case 1      'dec
        precess = dec + (n * DegCos(ra) + ddec / 3600) * t
    Case 2      'ra
        precess = ra + (m + n * DegSin(ra) * DegTan(dec) + dra / 3600) * t
End Select
End Function
'
'   The function below returns the geocentric ecliptic coordinates of the sun
'   to an accuracy corresponding to about 0.01 degree over
'   100 years either side of J2000. Coordinates returned
'   in spherical form. From page C24 of the 1996 Astronomical
```

```
'   Alamanac. Comparing accuracy with Planeph, a DOS ephemeris
'   by Chapront, we get;
'
'   Sun error
'                        RA sec    DEC arcsec
'   Max within 3 year     0.6        8.9
'   Min within 3 year    -2.1       -8.2
'   Max within 10 year    0.6       10.9
'   Min within 10 year   -2.6      -12.5
'   Max within 50 year    1.0       16.8
'   Min within 50 year   -2.9      -16.1
'
'   Error = C24 low precision method - Planeph
'
'   Note: Planeph was set to give output referred to mean
'         ecliptic and equinox of date.
'
'   The accuracy of this routine is good enough for sunrise
'   and shadow direction calculations, and for referring
'   low precision planetary and comet positions to the Earth,
'   but is no good for accurate coordinate conversion or
'   for eclipse or occultation use.
'
'   Coordinates are referred to the ecliptic and mean equinox of date
'
Function ssun(d As Double, index As Integer) As Double
    Dim g As Double
    Dim l As Double
    g = range360(357.528 + 0.9856003 * d)
    l = range360(280.461 + 0.9856474 * d)
    Select Case index
        Case 1
            ' radius vector of Sun
            ssun = 1.00014 - 0.01671 * DegCos(g) - 0.00014 * DegCos(2 * g)
        Case 2
            ssun = 0    'ecliptic latitude of Sun is zero to very good accuracy
        Case 3
            'longitude of Sun
            ssun = range360(l + 1.915 * DegSin(g) + 0.02 * DegSin(2 * g))
    End Select
End Function

'   returns the geocentric ecliptic coordinates of the sun
'   to an accuracy corresponding to about 0.01 degree over
'   100 years either side of J2000. Assumes ssun() exists.
'
'   rectangular form is easier for converting positions from helio-
'   centric to geocentric, but beware low accuracy (roughly 0.01 degree)
'   of values
'
Function rsun(d As Double, index As Integer) As Double
    Dim x As Double
    Dim y As Double
    Dim z As Double
    rsun = rectangular(ssun(d, 1), ssun(d, 2), ssun(d, 3), index)
End Function
'
'   sun() returns the geocentric ra and dec and radius vector
'   of the moon - calls smoon three times, and sequatorial
'   three times - sequatorial calls rectangular three times
'   each!
'
Function sun(d As Double, index As Integer) As Double
    sun = sequatorial(ssun(d, 1), ssun(d, 2), ssun(d, 3), d, index)
End Function
'
'   The function below implements Paul Schlyter's simplification
'   of van Flandern and Pulkkinen's method for finding the geocentric
```

```
'   ecliptic positions of the Moon to an accuracy of about 1 to 4 arcmin.
'
'   I can probably reduce the number of variables, and there must
'   be a quicker way of declaring variables!
'
'   The VBA trig functions have been used throughout for speed,
'   note how the atan function returns values in domain -pi to pi
'
Function smoon(ByVal d As Double, index As Integer) As Double
    Dim Nm As Double, im As Double, wm As Double, am As Double, ecm As Double, _
    Mm As Double, em As Double, Ms As Double, ws As Double, xv As Double, _
    yv As Double, vm As Double, rm As Double, x As Double, y As Double, _
    z As Double, lon As Double, lat As Double, ls As Double, lm As Double, _
    dm As Double, F As Double, dlong As Double, dlat As Double
    '   Paul's routine uses a slightly different definition of
    '   the day number - I adjust for it below. Remember that VBA
    '   defaults to 'pass by reference' so this change in d
    '   will be visible to other functions unless you set d to 'ByVal'
    '   to force it to be passed by value!
    d = d + 1.5
    '   moon elements
    Nm = range360(125.1228 - 0.0529538083 * d) * rads
    im = 5.1454 * rads
    wm = range360(318.0634 + 0.1643573223 * d) * rads
    am = 60.2666  '(Earth radii)
    ecm = 0.0549
    Mm = range360(115.3654 + 13.0649929509 * d) * rads
    '   position of Moon
    em = Mm + ecm * Sin(Mm) * (1# + ecm * Cos(Mm))
    xv = am * (Cos(em) - ecm)
    yv = am * (Sqr(1# - ecm * ecm) * Sin(em))
    vm = Application.Atan2(xv, yv)
    '   If vm < 0 Then vm = tpi + vm
    rm = Sqr(xv * xv + yv * yv)
    x = rm * (Cos(Nm) * Cos(vm + wm) - Sin(Nm) * Sin(vm + wm) * Cos(im))
    y = rm * (Sin(Nm) * Cos(vm + wm) + Cos(Nm) * Sin(vm + wm) * Cos(im))
    z = rm * (Sin(vm + wm) * Sin(im))
    '   moons geocentric long and lat
    lon = Application.Atan2(x, y)
    If lon < 0 Then lon = tpi + lon
    lat = Atn(z / Sqr(x * x + y * y))
    '   mean longitude of sun
    ws = range360(282.9404 + 0.0000470935 * d) * rads
    Ms = range360(356.047 + 0.9856002585 * d) * rads
    '   perturbations
    '   first calculate arguments below,
    '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
    dm = lm - ls        'Mean elongation of the Moon
    F = lm - Nm        'Argument of latitude for the Moon
    ' then add the following terms to the longitude
    ' note amplitudes are in degrees, convert at end
    Select Case index
        Case 1  '   distance terms earth radii
            rm = rm - 0.58 * Cos(Mm - 2 * dm)
            rm = rm - 0.46 * Cos(2 * dm)
            smoon = rm
        Case 2  '   latitude terms
            dlat = -0.173 * Sin(F - 2 * dm)
            dlat = dlat - 0.055 * Sin(Mm - F - 2 * dm)
            dlat = dlat - 0.046 * Sin(Mm + F - 2 * dm)
            dlat = dlat + 0.033 * Sin(F + 2 * dm)
            dlat = dlat + 0.017 * Sin(2 * Mm + F)
            smoon = lat * degs + dlat
        Case 3  '   longitude terms
```

```
            dlon = -1.274 * Sin(Mm - 2 * dm)        '(the Evection)
            dlon = dlon + 0.658 * Sin(2 * dm)       '(the Variation)
            dlon = dlon - 0.186 * Sin(Ms)           '(the Yearly Equation)
            dlon = dlon - 0.059 * Sin(2 * Mm - 2 * dm)
            dlon = dlon - 0.057 * Sin(Mm - 2 * dm + Ms)
            dlon = dlon + 0.053 * Sin(Mm + 2 * dm)
            dlon = dlon + 0.046 * Sin(2 * dm - Ms)
            dlon = dlon + 0.041 * Sin(Mm - Ms)
            dlon = dlon - 0.035 * Sin(dm)           '(the Parallactic Equation)
            dlon = dlon - 0.031 * Sin(Mm + Ms)
            dlon = dlon - 0.015 * Sin(2 * F - 2 * dm)
            dlon = dlon + 0.011 * Sin(Mm - 4 * dm)
            smoon = lon * degs + dlon
    End Select
End Function
'
'   rmoon uses smoon to return the geocentric ecliptic rectangular coordinates
'   of the moon to lowish accuracy.
'
'
Function rmoon(d As Double, index As Integer) As Double
    rmoon = rectangular(smoon(d, 1), smoon(d, 2), smoon(d, 3), index)
End Function
'
'   moon() returns the geocentric ra and dec and radius vector
'   of the moon - calls smoon three times, and sequatorial
'   three times - sequatorial calls rectangular three times
'   each!
'
Function moon(d As Double, index As Integer) As Double
Dim nigel As Double
    If index = 4 Then
        nigel = smoon(d, 2)
        moon = d
    Else
        moon = sequatorial(smoon(d, 1), smoon(d, 2), smoon(d, 3), d, index)
    End If
End Function
'
'   Solutions of the kepler equation using a Newton's method approach.
'   See Meeus or Duffett-Smith (calculator)
'
Function kepler(m As Double, ecc As Double, Optional eps)
'   solves the equation e - ecc*sin(e) = m for e given an m
'   returns the value of the 'true anomaly' in rads
'   m  -  the 'mean anomaly' in rads
'   ecc - the eccentricity of the orbit
'   eps - the precision parameter - solution will be
'         within 10^-eps of the true value.
'         don't set eps above 14, as convergence
'         can't be guaranteed. If not specified, then
'         taken as 10^-8 or 10 nano radians!
'
    Dim delta As Double, e As Double, v As Double

    e = m               'first guess
    delta = 0.05        'set delta equal to a dummy value
    If (IsMissing(eps)) Then eps = 8                'if no eps then assume 10^-8
    Do While Abs(delta) >= 10 ^ -eps                'converged?
        delta = e - ecc * Sin(e) - m                'new error
        e = e - delta / (1 - ecc * Cos(e))          'corrected guess
    Loop
    v = 2 * Atn(((1 + ecc) / (1 - ecc)) ^ 0.5 * Tan(0.5 * e))
    If v < 0 Then v = v + tpi
    kepler = v
End Function
'
'   The functions below return the heliocentric ecliptic coordinates
```

```
'   of the planets to an accuracy of a few minutes of arc. The coordinates
'   are referred to the equinox of J2000.0
'
'   The functions use a simple Kepler ellipse, but with
'   mean elements which change slightly with the time since
'   J2000. See 'Explanatory supplement to the Astronomical
'   Almanac' 1992, page 316, table 5.8.1. Worst case errors
'   over the period 1800 - 2050 AD in arcsec are below;
'
'                    Ra      Dec
'   Mercury          20"      5"
'   Venus            20"      5"
'   Earth            20"      5"
'   Mars             25"     30"
'   Jupiter         300"    100"
'   Saturn          600"    200"
'   Uranus           60"     25"
'   Neptune          40"     20"
'   Pluto            40"     10"
'

'
'   The rplanet() function returns the ecliptic heliocentric coordinates
'   of each of the major planets. You select the planet you want using
'   pnumber, and the coordinate you want with index as usual.
'
Function rplanet(d As Double, pnumber As Integer, index As Integer) As Double
    Dim x As Double, y As Double, z As Double, v As Double, m As Double, _
    i As Double, o As Double, p As Double, a As Double, e As Double, _
    l As Double, r As Double
    '
    '   get elements of the planet
    '
    element i, o, p, a, e, l, d, pnumber
    '
    '   position of planet in its orbit
    '
    m = range2pi(l - p)
    v = kepler(m, e, 8)
    r = a * (1 - e * e) / (1 + e * Cos(v))
    '
    '   heliocentric rectangular coordinates of planet
    '
    Select Case index
    Case 1      'x coordinate
        rplanet = r * (Cos(o) * Cos(v + p - o) - Sin(o) * Sin(v + p - o) * Cos(i))
    Case 2      'y coordinate
        rplanet = r * (Sin(o) * Cos(v + p - o) + Cos(o) * Sin(v + p - o) * Cos(i))
    Case 3      'z coordinate
        rplanet = r * (Sin(v + p - o) * Sin(i))
    End Select
End Function
'
'
'   The planet() function returns the equatorial geocentric coordinates
'   of each of the major planets. You select the planet you want using
'   pnumber, and the coordinate you want with index as usual. Code is
'   duplicated from rplanet() to reduce the number of calls to kepler()
'
'
Function planet(d As Double, pnumber As Integer, index As Integer) As Double
    Dim x As Double, y As Double, z As Double, v As Double, m As Double, _
    i As Double, o As Double, p As Double, a As Double, e As Double, _
    l As Double, r As Double, xe As Double, ye As Double, ze As Double, _
    s1 As Double, si As Double, so As Double, c1 As Double, ci As Double, _
    co As Double
    '
    '   elements of planet - select from the values
```

```
    '
    element i, o, p, a, e, l, d, pnumber
    '
    '   position of planet in its orbit
    '
    m = range2pi(l - p)
    v = kepler(m, e, 8)
    r = a * (1 - e * e) / (1 + e * Cos(v))
    '
    '   heliocentric rectangular coordinates of planet
    '
    s1 = Sin(v + p - o)
    si = Sin(i)
    so = Sin(o)
    c1 = Cos(v + p - o)
    ci = Cos(i)
    co = Cos(o)
    x = r * (co * c1 - so * s1 * ci)
    y = r * (so * c1 + co * s1 * ci)
    z = r * (s1 * si)
    '
    '   elements of earth (reusing variables)
    '
    element i, o, p, a, e, l, d, 3
    '
    '   position of earth in its orbit
    '
    m = range2pi(l - p)
    v = kepler(m, e, 8)
    r = a * (1 - e * e) / (1 + e * Cos(v))
    '
    '   heliocentric rectangular coordinates of earth
    '
    s1 = Sin(v + p - o)
    si = Sin(i)
    so = Sin(o)
    c1 = Cos(v + p - o)
    ci = Cos(i)
    co = Cos(o)
    xe = r * (co * c1 - so * s1 * ci)
    ye = r * (so * c1 + co * s1 * ci)
    ze = r * (s1 * si)
    '
    '   convert to geocentric rectangular coordinates
    '
    x = x - xe
    y = y - ye
    '   z = z
    '
    '   rotate around x axis from ecliptic to equatorial coords
    '
    ecl = 23.429292 * rads      'value for J2000.0 frame
    xe = x
    ye = y * Cos(ecl) - z * Sin(ecl)
    ze = y * Sin(ecl) + z * Cos(ecl)
    '
    '   find the RA and DEC from the rectangular equatorial coords
    '
    Select Case index
    Case 3
        ' RA in degrees
        planet = Application.Atan2(xe, ye) * degs
        If planet < 0 Then planet = 360 + planet
    Case 2
        ' DEC in degrees
        planet = Atn(ze / Sqr(xe * xe + ye * ye)) * degs
    Case 1
        ' Radius vector in au
```

```
        planet = Sqr(xe * xe + ye * ye + ze * ze)
    End Select
End Function
'
'   The subroutine below replaces the values of i,o,p,a,e,L
'   with the values for the planet selected by pnum. You could
'   always add planet like objects, but watch the value of
'   the inclination i. The method used in planet is only
'   good for orbits 'near' the ecliptic.
'
'   This is an example of the Visual Basic default 'passing by
'   reference'
'
Sub element(i As Double, o As Double, p As Double, _
            a As Double, e As Double, l As Double, _
            ByVal d As Double, ByVal pnum As Integer)
    Select Case pnum
        Case 1          'mercury
            i = (7.00487 - 0.000000178797 * d) * rads
            o = (48.33167 - 0.0000033942 * d) * rads
            p = (77.45645 + 0.00000436208 * d) * rads
            a = 0.38709893 + 1.80698E-11 * d
            e = 0.20563069 + 0.000000000691855 * d
            l = range2pi(rads * (252.25084 + 4.092338796 * d))
        Case 2          'venus
            i = (3.39471 - 0.0000000217507 * d) * rads
            o = (76.68069 - 0.0000075815 * d) * rads
            p = (131.53298 - 0.000000827439 * d) * rads
            a = 0.72333199 + 2.51882E-11 * d
            e = 0.00677323 - 0.00000000135195 * d
            l = range2pi(rads * (181.97973 + 1.602130474 * d))
        Case 3          'earth
            i = (0.00005 - 0.000000356985 * d) * rads
            o = (-11.26064 - 0.00013863 * d) * rads
            p = (102.94719 + 0.00000911309 * d) * rads
            a = 1.00000011 - 1.36893E-12 * d
            e = 0.01671022 - 0.00000000104148 * d
            l = range2pi(rads * (100.46435 + 0.985609101 * d))
        Case 4          'mars
            i = (1.85061 - 0.000000193703 * d) * rads
            o = (49.57854 - 0.0000077587 * d) * rads
            p = (336.04084 + 0.00001187 * d) * rads
            a = 1.52366231 - 0.000000001977 * d
            e = 0.09341233 - 0.00000000325859 * d
            l = range2pi(rads * (355.45332 + 0.524033035 * d))
        Case 5          'jupiter
            i = (1.3053 - 0.0000000315613 * d) * rads
            o = (100.55615 + 0.00000925675 * d) * rads
            p = (14.75385 + 0.00000638779 * d) * rads
            a = 5.20336301 + 0.0000000166289 * d
            e = 0.04839266 - 0.00000000352635 * d
            l = range2pi(rads * (34.40438 + 0.083086762 * d))
        Case 6          'saturn
            i = (2.48446 + 0.0000000464674 * d) * rads
            o = (113.71504 - 0.0000121 * d) * rads
            p = (92.43194 - 0.0000148216 * d) * rads
            a = 9.53707032 - 0.0000000825544 * d
            e = 0.0541506 - 0.0000000100649 * d
            l = range2pi(rads * (49.94432 + 0.033470629 * d))
        Case 7          'uranus
            i = (0.76986 - 0.0000000158947 * d) * rads
            o = (74.22988 + 0.0000127873 * d) * rads
            p = (170.96424 + 0.0000099822 * d) * rads
            a = 19.19126393 + 0.0000000416222 * d
            e = 0.04716771 - 0.00000000524298 * d
            l = range2pi(rads * (313.23218 + 0.011731294 * d))
        Case 8          'neptune
            i = (1.76917 - 0.0000000276827 * d) * rads
```

```
            o = (131.72169 - 0.0000011503 * d) * rads
            p = (44.97135 - 0.00000642201 * d) * rads
            a = 30.06896348 - 0.0000000342768 * d
            e = 0.00858587 + 0.000000000688296 * d
            l = range2pi(rads * (304.88003 + 0.0059810572 * d))
        Case 9          'pluto
            i = (17.14175 + 0.0000000841889 * d) * rads
            o = (110.30347 - 0.0000002839 * d) * rads
            p = (224.06676 - 0.00000100578 * d) * rads
            a = 39.48168677 - 0.0000000210574 * d
            e = 0.24880766 + 0.00000000177002 * d
            l = range2pi(rads * (238.92881 + 3.97557152635181E-03 * d))
    End Select
End Sub
```