1-1-1992

Methods for calculating solar position and day length including computer programs and subroutines

M L. Roderick

ISSN 0729-3135
1992

Methods for Calculating Solar Position and Day Length including Computer Programs and Subroutines

Prepared by:

M.L. Roderick
Land Management
Western Australian Department of Agriculture
South Perth Western Australia 6151

Resource Management Technical Report No. 137

Disclaimer

The contents of this report were based on the best available information at the time of publication. It is based in part on various assumptions and predictions. Conditions may change over time and conclusions should be interpreted in the light of the latest information available.

Contents

Page

1.0 Introduction 1

2.0 Theoretical background 2
- 2.1 Earth-Sun geometry 2
- 2.2 Coordinate systems used in solar prediction 3
  - 2.2.1 Longitude/latitude 3
  - 2.2.2 Right ascension/declination coordinates 3
  - 2.2.3 Time 5
- 2.3 The Astronomic Triangle 6
  - 2.3.1 Solution of the Astronomic Triangle 7
- 2.4 Corrections 8
  - 2.4.1 Zenith 8
    - 2.4.1.1 Parallax 8
    - 2.4.1.2 Atmospheric refraction 9
  - 2.4.2 Azimuth 11
- 2.5 Solar ephemeris 11

3.0 Computing the length of day 13

4.0 Uncertainty in solar position 17

5.0 Computer routines 19

6.0 Summary 21

Acknowledgments 22

References 22

Bibliography 22

Appendices 23
- A. Test data for computing solar position 23
- B. Documented source code and instruction for using subroutines 23

List of Figures

		Page
Figure 1.	Orientation of Ecliptic and Equator (per Montenbruck, 1989, p3)	2
Figure 2.	Geographic Coordinate System	4
Figure 3.	Right Ascension/Declination Coordinate System	4
Figure 4.	Relationship between time based coordinate systems	6
Figure 5.	The Astronomical Triangle	7
Figure 6.	Parallax correction	9
Figure 7.	Refraction correction	10
Figure 8.	Grid/true bearings	11
Figure 9.	Solar zenith at sun rise/set	13
Figure 10.	Day length (hrs) as a function of latitude for 1992. Latitudes shown are 0 to 40 degrees	15

1.0 Introduction

Predicting the position of the sun is critical for any remote sensing studies in the visible and infra-red regions of the electromagnetic spectrum. In this part of the spectrum, the radiance recorded by a sensor is totally composed of reflected solar radiation. The sensor, target, sun geometry has an important effect on the nature of the recorded reflectances, according to the bidirectional reflectance of the target, and the position of shadows. In studies detecting change using multi-temporal imagery, modelling of the variable solar position is an important component in removing variation in radiance, not attributable to changes on the land surface. In addition to these remote sensing applications, agricultural scientists are interested in calculating the amount of solar radiation and day length as inputs to crop models.

This paper briefly describes the theory underlying the calculation of solar position and the length of daylight hours. A number of corrections are described, as well as methods of integrating the data into a Geographic Information System (GIS).

Traditional methods of calculating solar position use specific sets of tables such as The Star Almanac for Land Surveyors. These tables list solar ephemeris data for each day of the year (often at 6 hourly intervals). The use of such tables is not convenient for large numbers of calculations and a computer based method is preferred. A permanent almanac has been developed from formula given in The Astronomical Almanac. This almanac will generate predictions of solar ephemeris data, for any time in the period 1950 to 2050. Using the predictions of solar ephemeris, the solar azimuth and zenith angles may be computed with an approximate accuracy of 1' of arc.

Techniques for estimating the uncertainty in solar azimuth and zenith predictions, given uncertainty in the input data are derived using propagation of variance techniques.

A method to calculate the length of day is developed. This is designed for the specific use of Agricultural scientists, in studies of the influence of day length on plant development.

The theory has been implemented in a number of computer subroutines. The source code and documentation for these routines are included in a separate document (Appendix B)

They are written in ANSI standard C and have been compiled on a number of platforms without any problems. In addition, a number of compiled programs are available for IBM PCs.

2.0 Theoretical Background

2.1 Earth-Sun geometry

The plane of the Earth's orbit around the Sun is described as the ecliptic and is shown in Figure 1.

The projection of the Earth's equatorial plane (to infinity) is described as the celestial equator. The line of intersection of this plane with the ecliptic defines the equinoxes. The earth's position at the vernal and autumnal equinoxes is shown in figure 1. The celestial equator remains practically fixed in position, although for very precise work such as geodetic Surveying, the variations in the Earth's orbit and gravitational effects of other celestial objects need to be considered. The angle between the ecliptic and equatorial planes is termed the obliquity of the ecliptic.

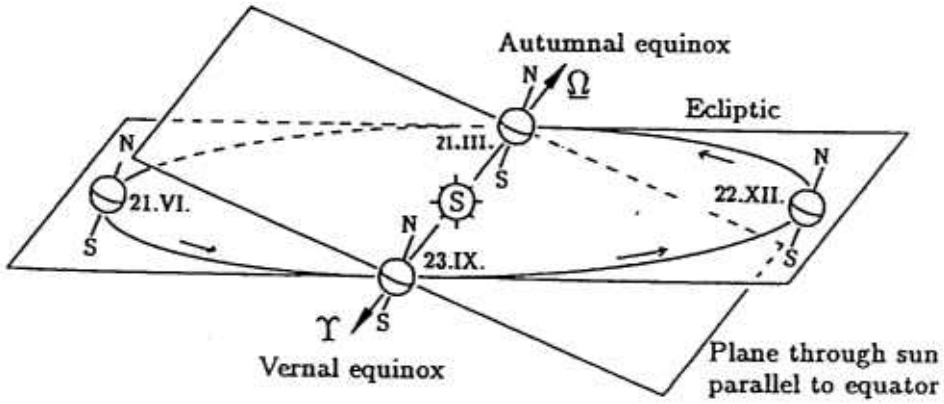

Figure 1. Orientation of Ecliptic and Equator (per Montenbruck, 1989, p3).

In the southern hemisphere, the vernal equinox marks the point where the next season will be summer (i.e. Sun crosses the equator in a southerly direction), and the autumnal equinox indicates winter will soon arrive. These equinoxes occur on approximately 23rd September (vernal) and 21st March (autumnal) each year, although the exact time will vary slightly from year to year. These variations are caused by perturbations in the earth's orbit and gravitational effects of the moon and other planets. It is common in precise applications, to actually define the datum year for the equinoxes.

While the Earth orbits the Sun, in positional astronomy, it is convenient to consider the Earth as fixed in space, and all celestial bodies rotate around the earth. This is also how celestial bodies appear to move to an observer on the Earth. The centre of this (imaginary) fixed Earth, defines the centre of the celestial sphere. The orientation of the sphere is defined by the North-South axis of the Earth (see Figure 3).

The celestial sphere has an infinite radius, and positions on the sphere are reckoned using (2) angles only. The intersection of the Earth's North-South axis with the celestial sphere defines the North and South celestial poles. In the northern hemisphere, this is also the (approximate) position in the sky of the so called Pole (or North) Star. In the southern hemisphere, a faint star called Sigma Octantis is very close to the South celestial pole.

2.2 Coordinate systems used in solar prediction

Three basic coordinate systems are used, right ascension/declination for the positions of celestial objects, longitude/latitude for positions on the earth, and time. These are discussed below.

2.2.1 Longitude/latitude

These are the familiar geographic coordinates used on the earth. They are a system of polar coordinates as shown in figure 2. Longitude is measured from the Greenwich meridian, and latitude is measured from the equator.

Thus any position on the Earth's surface can be uniquely specified by a longitude and latitude. For basic astronomy, it is sufficient to consider the earth as a perfect sphere. However, for precise applications (e.g. geodetic surveying), the earth is modeled as an ellipsoid (i.e. like an egg)

Circles of longitude are called meridians and are also great circles. Since circles of latitude do not cut the centre of the earth (except the equator), they are not great circles. The exception is the equator, which is a great circle. The relationship between meridians and time will be discussed in the section titled Time.

2.2.2 Right ascension/declination coordinates

The right ascension/declination system is used to reckon the position of celestial objects and is similar in concept to the longitude/latitude system discussed above. The right ascension is reckoned clockwise from the vernal equinox, and declination is reckoned as the angle from the celestial equator to the object(positive north), subtended at the centre of the celestial sphere. These are shown in figure 3.

Since the earth actually moves in space, the RA will vary according to the earth's orbital position. However, for stars, the distance to the stars is so great in comparison to the earth sun distance, that this introduces very small (negligible) errors for all but the most precise applications. This is not the case for the sun, as the distance is small (finite) in comparison to the stars. Thus the right ascension/declination of the sun will vary on a daily basis.

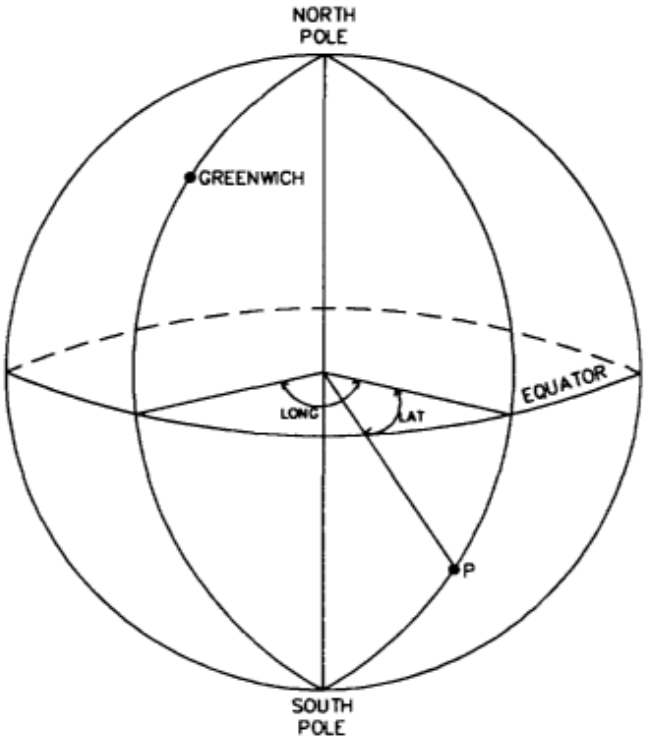

Figure 2. Geographic coordinate system

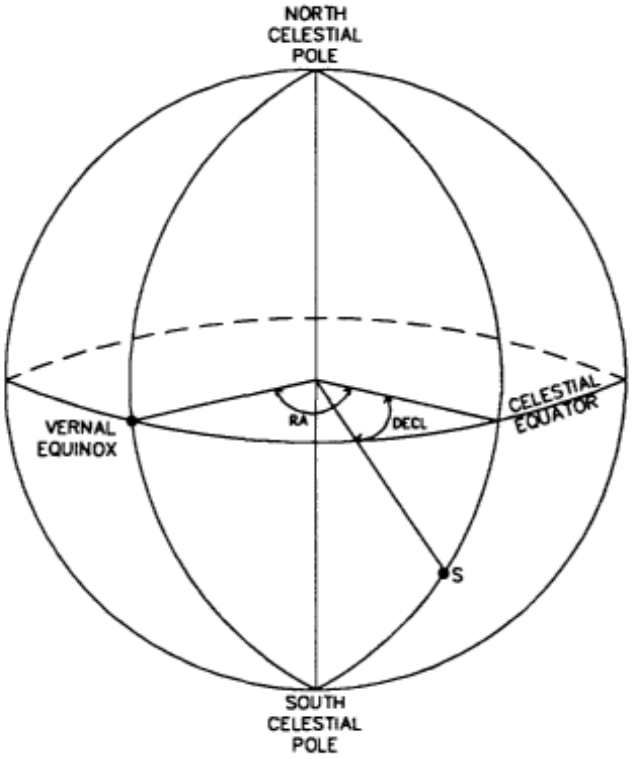

Figure 3. Right ascension/declination coordinate system

2.2.3 Time

Time is the third major coordinate system of prime importance in astronomy. To an observer on the earth, the line through the observer's zenith (i.e. vertically overhead) and to the North Pole is called the observer's meridian. This is a great circle following a circle of longitude.

Two systems of time are commonly used, local (or solar) time and civil time. Noon by local time is reckoned when the sun is on the observer's meridian. Obviously, every line of longitude will have its own unique solar time. Due to the need to standardize time for civil purposes (originally railway schedules), time zones were introduced (i.e. civil time). In Western Australia, we normally keep WST (Western Standard Time) which is 8 hours east (i.e. ahead) of Greenwich (Longitude = 0).

The earth's orbit around the sun is slightly elliptical, so the length of a solar day (i.e. time between successive crossings of the sun on the observers meridian) will not be constant throughout the year. Due to the civil requirements mentioned above, a fictitious sun (called the mean sun) has been invented. The length of a mean day is designed to be an average of all the days in the year. The difference between the true sun and this fictitious sun is called the equation of time. The equation of time varies between -16 and +16 minutes (of time) throughout the year.

Universal time (UT) is an internationally agreed method where time is reckoned by the fictitious mean sun and is referenced to the Greenwich meridian.

Thus in WA: UT = WST - 8 hrs.

Time is usually expressed in a 24 hour clock. For trigonometric calculations, time must be expressed as an angle. Since 24 hours is one complete revolution of the earth, and this equals 360 degrees of arc, then the relationship is given by:

Time (arc) = Time (hrs) x 15

From the above, we can see that WST is based on a meridian at 120 degrees (i.e. 15 x 8). Thus for locations east of this meridian, the sun will be at solar noon before our watches are at 1200 hrs. The converse will be true for locations west of the meridian (120 degrees longitude). During periods of daylight saving, one hour is added the time zone correction to get UT.

We also note, that very approximately, the sun travels 15 degrees of arc every hour of time. This is useful as a guide for later discussions on predicting the length of day.

The hour angle is the time since the celestial body was last on the upper branch (i.e. overhead) of the observer's meridian reckoned positive west. Thus, if the sun passed the observer's meridian at 1200 hrs, and the time was currently 1400 hrs, then the hour angle is 1400-1200, or 2 hours. Similarly, if the sun was due to cross the observer's meridian at 1200 hrs, and the current time was 1000 hrs, then the hour angle 0100-1200, or -2 hours.

Since right ascension is effectively a time based measure also (i.e. analogous to longitude), we can combine it with time in a diagram below to demonstrate the relationships discussed above.

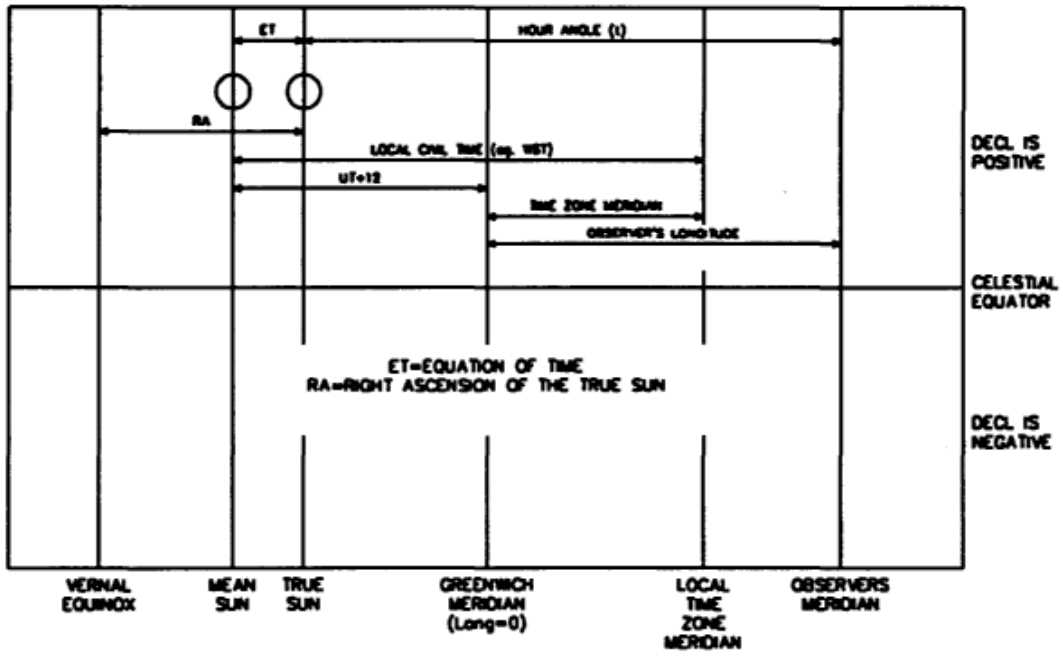

Figure 4. Relationship between time based coordinate system.

From Figure 4, the hour angle (t) is given by:

T = UT + Longitude + Equation Time - 180

2.4 The astronomic triangle

The astronomic triangle is a spherical triangle which forms the basis from which the solar position can be solved. It is the central tool in predicting solar and other celestial positions and is shown in Figure 5.

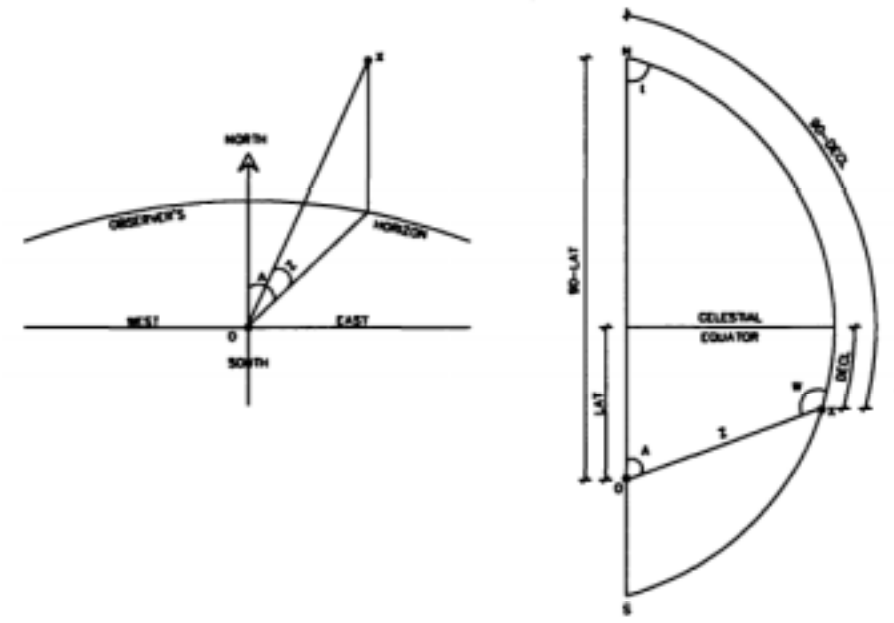

Figure 5. The astronomical triangle.

In the spherical triangle shown in Figure 5:

N = North pole
S = South pole
O = observer
X = sun or other celestial object

where:

The spherical angle at N is t= hour angle
The spherical angle at 0 is A = azimuth
The spherical angle at X is w = parallactic angle

Spherical distance NX = 90 – decl
Spherical distance NO = 90 – lat
Spherical distance OX = zenith angle

All elements of the astronomic triangle may be observed excluding the parallactic angle.

The declination may be found from solar ephemeris data, and the observer can find his geographic coordinates from a map.

2.4.1 Solution of the astronomic triangle

Spherical trigonometry is used to solve the spherical triangle. These formula can be found in any mathematics, surveying or astronomy textbook. The solution for azimuth and zenith is given by (refer to Figure 5);

$$\tan A = \frac{-\sin t}{\tan(\text{decl})\cdot\cos(\text{lat}) - \sin(\text{lat})\cdot\cos(t)}$$

$$\text{coz } Z = \text{Sin(lat)*sin(decl) + cos(lat)*cos(decl)*cos(t)}$$

where A = azimuth
Z = zenith angle

2.5 Corrections

This section discusses a number of corrections that may be applied to the azimuth and zenith computed per 2.4.1 above. The decision to apply the corrections will depend on the particular application.

2.5.1 Zenith

The zenith angle computed above, gives the angle from the centre of the earth to the sun. For certain applications we require the angle, incident on the earth's surface. Corrections are required for parallax and atmospheric refraction. In essence, we wish to estimate the angle that a surveyor would have observed, as this is the actual zenith of the incident solar radiation.

2.5.1.1 Parallax

The zenith angle computed above is from the centre of the earth to the sun. The correction for parallax is given by (see Figure 6).

$$\text{Sin } P = \frac{R * \sin(Z_{calc})}{D}$$

Where P = parallax correction
R = radius of earth
D = earth sun distance
Z = calculated zenith distance from centre of earth

and Z corr = Z calc + P

Typical values are:

R = 6378 km
D = 149597870 km

Thus: sin P = 0.00004263 * sin (Z)

The correction will never exceed 9" of arc. It may normally be neglected, except for precise applications.

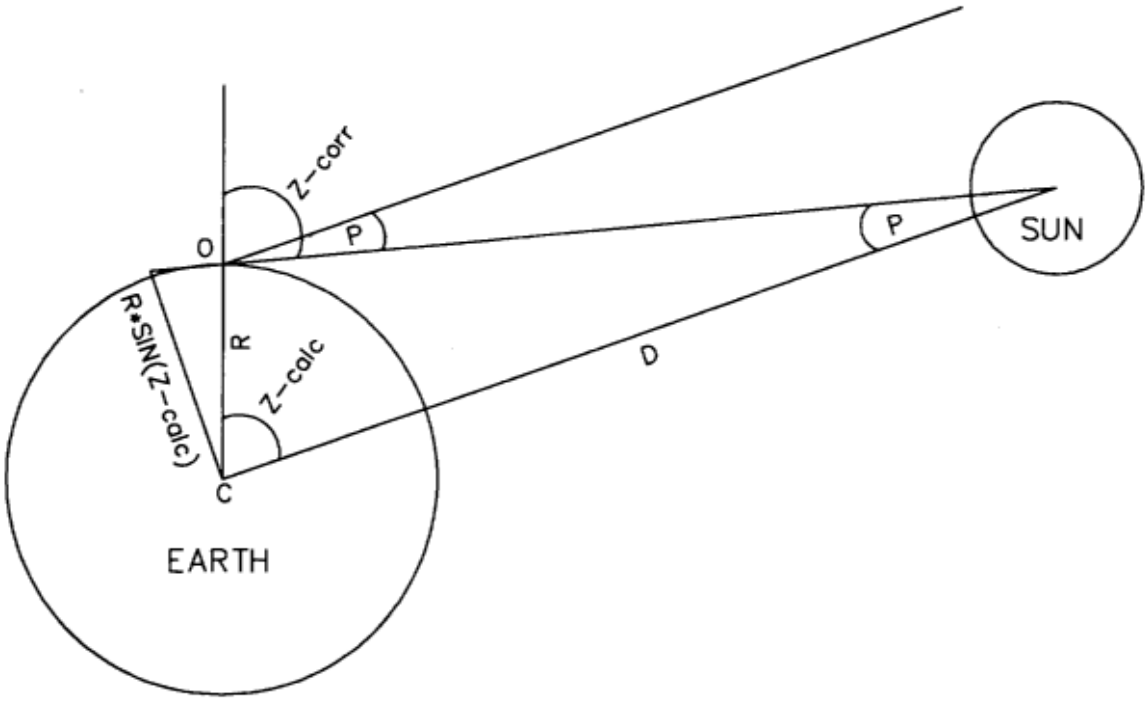

Figure 6. Parallax correction.

2.5.1.2 Atmospheric refraction

Light is refracted through the atmosphere on it's way to the earth's surface. This causes the sun to appear higher in the sky than it really is as shown in Figure 7.

Thus for studies that require a precise estimate for the actual incident angle of the solar radiation, we need to include a term for the refraction.

The methods of correction are given depending on the solar zenith angle Z as follows:

Method 1. Z < 70 degrees

$$R = 0.00452 * \tan(z) * [P / (273+T)]$$

where R = refraction
Z = zenith
P = pressure in millibars
T = temperature in degrees celsius

This is the method normally used by surveyors to correct observed zenith distances. For high zenith angles (i.e. near the horizon), refraction becomes very uncertain, due to the increased path length of light through the atmosphere. In the above formula, tan(Z) also becomes uncertain close to 90 degrees. An alternative formulation is given by *The Astronomical Almanac* (page B59) as follows for higher zenith angles:

Method 2. Z > 70 degrees

$$R = \frac{[P / (273+T)]}{} * \frac{(0.1594 + 0.0196*a + 0.00002*a*a)}{1 + 0.505*a + 0.0845*a*a}$$

where a = altitude in decimal degrees
= 90 – Z

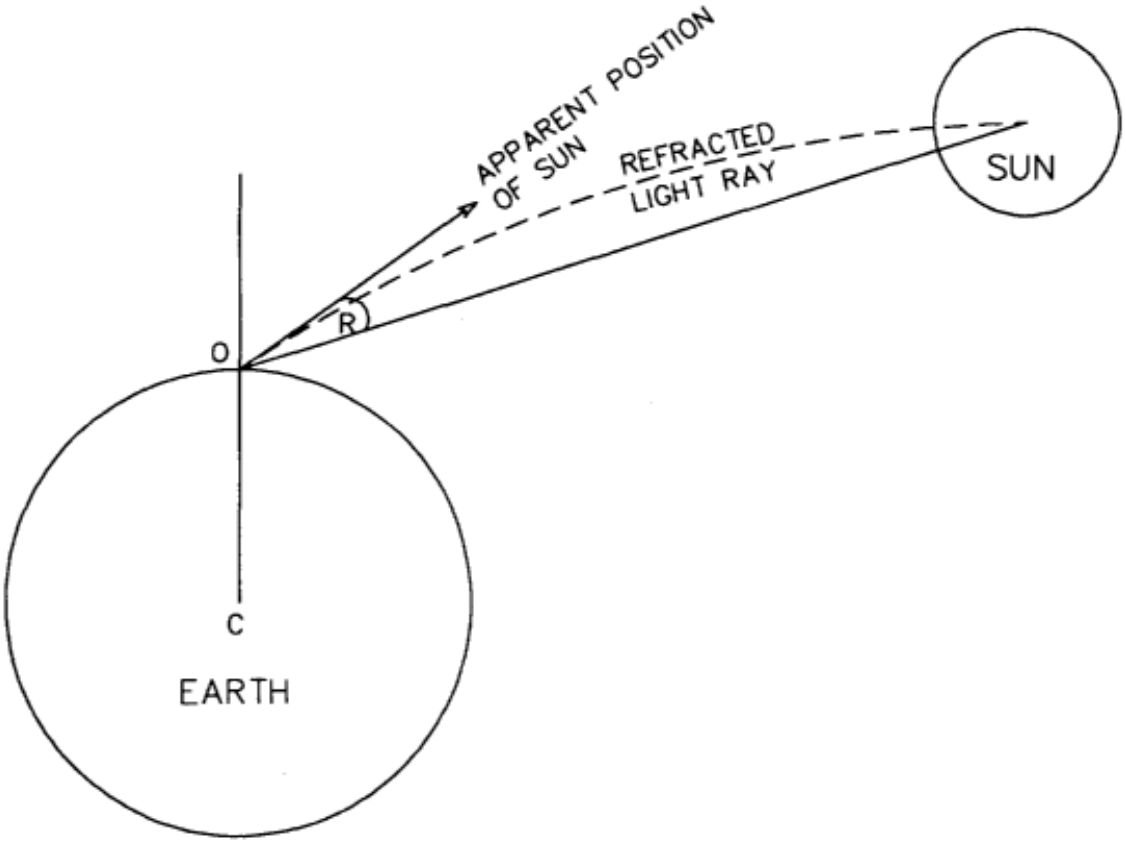

Figure 7. Refraction correction.

As noted, refraction becomes very uncertain for zenith angles greater than 70 degrees. For precise applications, observations are not made below 70 degrees. Refraction is mainly affected by temperature in the vicinity of the observer for zenith angles less than 75 degrees (Garfinkel, 1967). For an advanced discussion on refraction see Garfinkel, 1967.

Typical values for refraction are:

Zenith (degrees)	Refraction correction (minutes)
90	34’
80	5’
45	1
< 45	< 1’

For most (if not all) remote sensing studies, refraction will not be significant. It is included here for completeness.

2.5.2 Azimuth

The azimuth calculated is the so called true azimuth (or true bearing). It is the clockwise angle to the sun measured from the observer's meridian. For applying corrections to remotely sensed data, this will be preferred, as the satellite ephemeris is also computed using true bearings. However, in a GIS, using the Australian Map Grid, all bearings are known as grid bearings. The difference between grid and true bearings is known as grid convergence and is shown in figure 8 below:

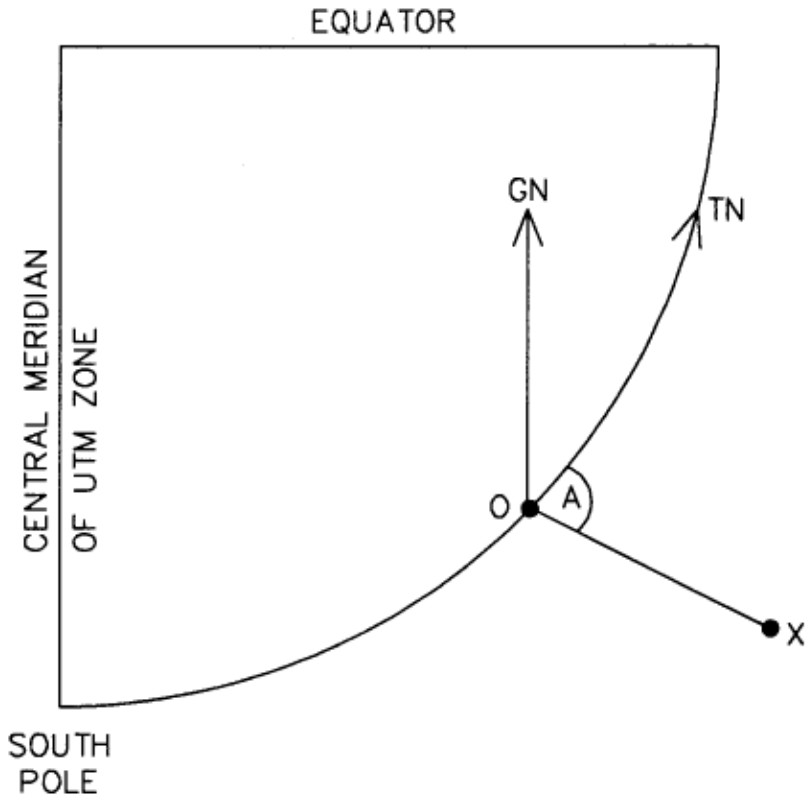

Figure 8. Grid/true bearings.

Grid convergence is given by:

Grid_Conv = -sin(lat) * tan(long – long_cen_mer)

Where long = observer's longitude
long_cen_mer = longitude of the central meridian of the AMG zone
lat = observer's latitude
Grid bearing = True bearing + Grid_conv.

2.6 Solar Ephemeris

Solar ephemeris data is required for the following elements of the astronomical triangle to solve for azimuth and zenith angles:

ET = equation time
Deci = solar declination

Other ephemeris elements which may be of interest include the earth sun distance and the sun's semi diameter.

These can be extracted from solar tables such as *The Star Almanac for Land Surveyors*, or *The Astronomical Almanac*. These tables are generated using very precise calculations for the earth's orbit and generate very accurate predictions of solar position (e.g.. 2" arc). Alternatively, we can make some approximations, and ignore small orbital effects, to derive relatively simple methods of predicting the above ephemeris elements.

A routine has been included in the software, that will predict the solar position to approximately 1 ' (arc) for any time during the period 1950 to 2050. The source code for the routine is included in appendix B, and is fully documented. This method has been adapted using formula given in *The Astronomical Almanac* on page C24.

3.0 Computing the Length of Day

Agricultural scientists modeling plant phenology can estimate the day length for each day of the year based on solar orbital geometry. It is normal to measure the day length, by specifying the solar zenith angle at sunrise and sunset. The solar zenith at sunrise/set is taken as 90 degrees 50 minutes. This is made up of 34' refraction and 16' for the sun's semi-diameter as shown in Figure 9.

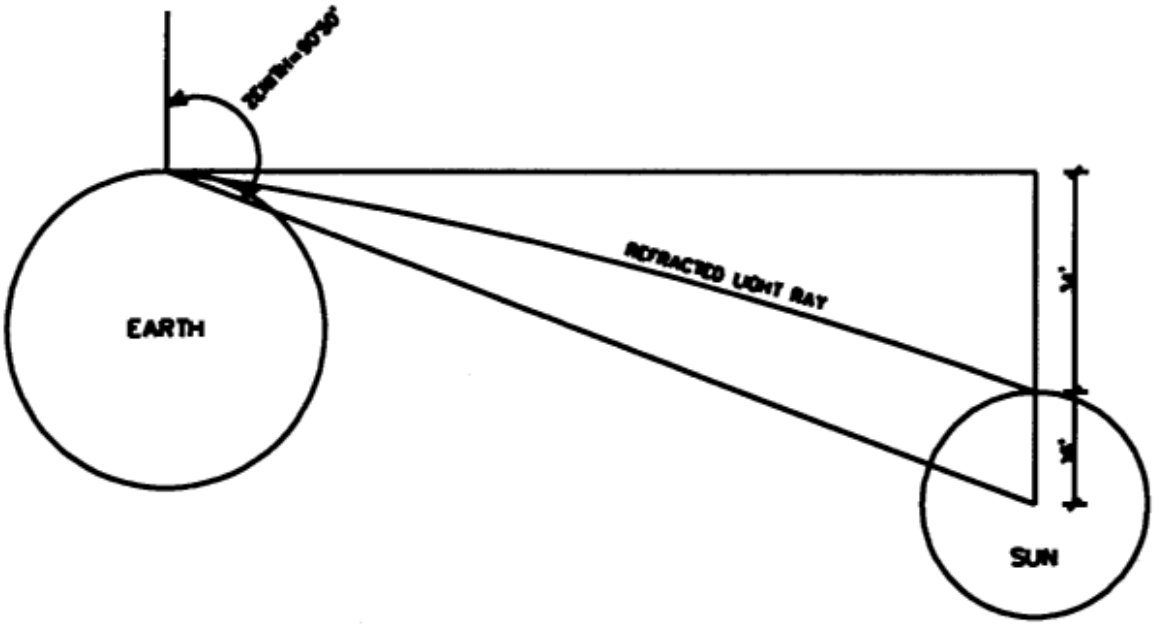

Figure 9. Solar zenith at sunrise/set.

Alternatively, in an agricultural application, it is possible to use experimental data to estimate the solar zenith angle when a certain amount of solar radiation becomes available for plant growth. For an example, see Keisling (1982).

Typical solar zenith angles are:

Sun rise/set:

When the upper limb of the sun is visible on horizon. Solar zenith = 90.833 degrees (i.e.90 deg. 50 mm)

Civil twilight:

Time between sun rise/set and when the solar zenith is 96 degrees.

While the sun's semi-diameter changes throughout the year from 15' to 17', given the uncertainty in refraction, a value of 16' can be adopted for daylength studies. The difference is only minor in time. For example, an error of 5' in the solar zenith angle, will produce an (approximate) error of 20 seconds in time.

To predict the sun rise/set time for a particular solar zenith, we need to solve the astronomical triangle for the hour angle t, given lat, deci and Z. However, declination

is a function of time, so iterative methods must be employed to find a solution. This has been programmed in a routine (called sunprezd) included in the software package.

The algorithm for the solution is as follows:

1. Assume an initial value for t = +- 6 hours on the day in question. The sign will depend on whether we are considering sun rise or set. Positive hour angles indicate sun set, and vice versa for sun rise.

2. Use the estimate of t from step 1, to get the declination from ephemeris data, and calculate the zenith at this hour angle.

3. The zenith in step 2, Z[l], will not equal the required zenith, so calculate a correction to t based on this difference. It is sufficient to assume the sun moves 15 degrees (arc) in an hour, so the correction is given by:

   correction to initial t = (Requ_Zenith - Z[1])/l5

4. Update t, and go to step 2. Continue the iteration until the correction is very small (say less than 0.00001).

5. From the final value of t, we can compute the local standard time using the equation for hour angle as follows:

   UT = t - Long - ET + 180 and
   Local Standard Time = UT + Time Zone

By repeating this procedure for sunrise and sunset we can estimate daylength by subtracting these times. The length of day is a function of latitude (longitude only affects the actual time of sun rise/set, but not the length of day), and time of the year.

The day length for various latitudes, throughout 1992, for Western Australia is shown in Figure 10.

From Figure 10, the day length at the equinoxes (21 March and 23 September) is equal, regardless of latitude. Thus in higher latitudes it can be seen that summer days are longer, and winter days are shorter. The tropics have less variation in day length throughout the year than higher latitudes.

Figure 10 was prepared using data from the wasunup program, which is part of this package.

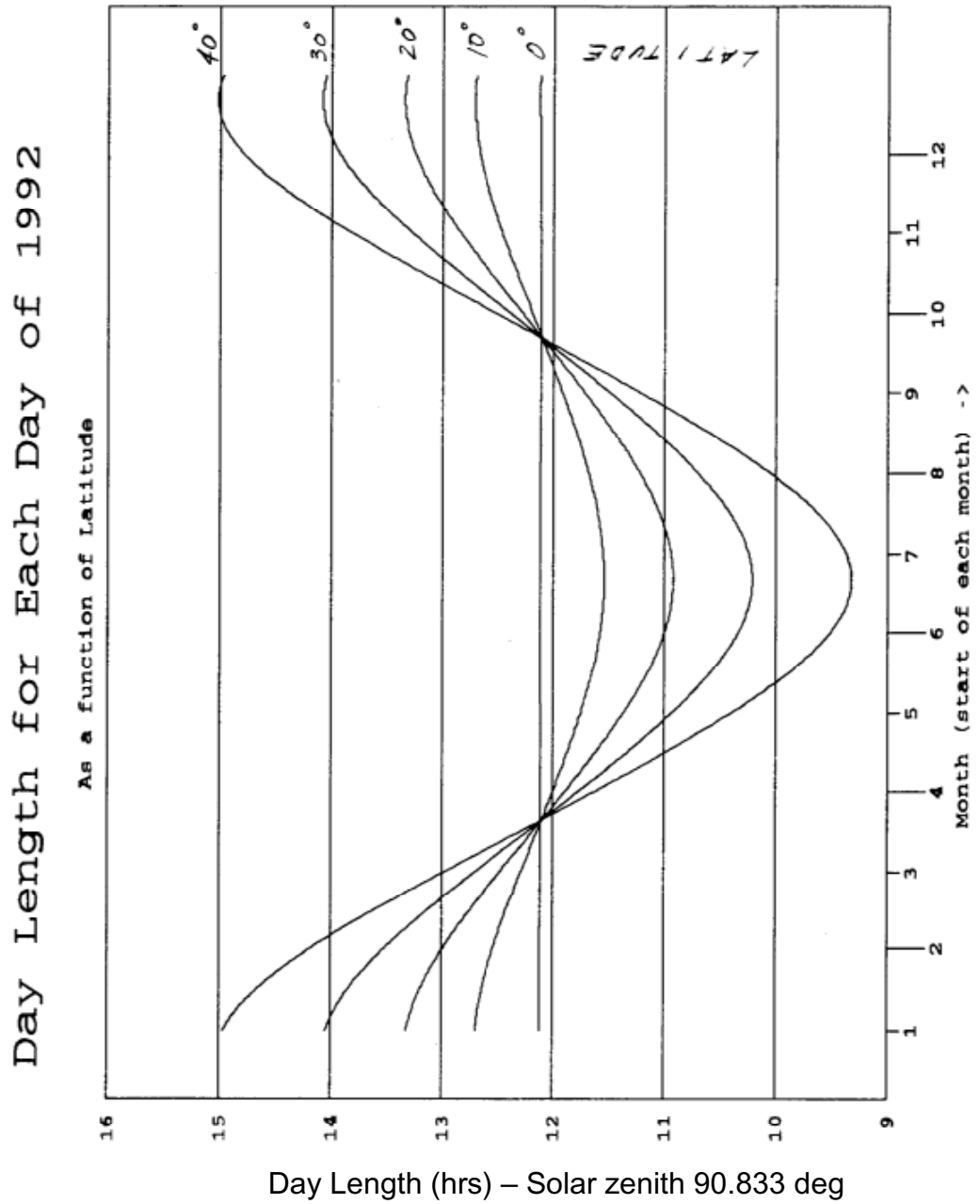

Figure 10. Day length (hrs) as a function of latitude for 1992. Latitude's shown are 0 to 40 degrees.

The estimates for length of daylight hours assume a perfectly spherical earth with no terrain variations. Terrain altitude will affect the visibility to the sun. Also, local shadowing will also have an impact, caused by adjacent objects such as trees, buildings, etc. In general, the effect of terrain and shadowing will reduce the length of daylight. The exception is sites right on the summit of hills, which can potentially get extra daylight hours both in the morning and afternoon. For the rest, such as sites on a hill side, they may get extra sunlight in the morning, and less in the afternoon depending on the particular circumstances.

Digital terrain models could be used to model terrain effects on solar radiation availability. In the Western Australian wheat belt, where the terrain is relatively flat and uniform, no extra benefit would be gained, as local shadowing (e.g. trees, shrubs, etc.) will have a comparable (or greater) effect on solar radiation availability than terrain. These shadowing effects would be extremely difficult and costly (? impossible) to model.

4.0 Uncertainty in Solar Position

It is important to generate an estimate of the uncertainty in the solar azimuth and zenith angles, given uncertainty in the input data. The uncertainty in solar position will vary according to the particular geometry.

Propagation of variance techniques have been used to develop expressions to estimate the uncertainty in solar position, and are discussed below. They are incorporated in a computer subroutine (sunerror) in the software package.

Considering Azimuth first:

$$\tan A = \frac{-\sin t}{\tan\delta \cos\phi - \sin\phi \cos t}$$

where A = azimuth
t = hour angle
δ = declination
ϕ = latitude

Using the law of propagation of variance, and assuming only random uncorrelated errors will occur, we get the following expression:

$$\sigma^2_A = \left(\frac{\partial A}{\partial t}\right)^2 \sigma^2_t + \left(\frac{\partial A}{\partial \phi}\right)^2 \sigma^2_\phi + \left(\frac{\partial A}{\partial \delta}\right)^2 \sigma^2_\delta$$

where

$$\frac{\partial A}{\partial t} = \frac{\text{Cos } w \ \sin A}{\sin t}$$

$$\frac{\partial A}{\partial \phi} = \sin A \ \cot Z$$

$$\frac{\partial A}{\partial \delta} = \frac{\cos \phi \ \sin t}{\text{Sin}^2 Z}$$

and w is the parallactic angle. This may be computed using any valid method for spherical triangles such as:

$$\sin w = \frac{\cos \phi \ \sin t}{\sin Z}$$

or

$$\cos w = \frac{\sin \phi - \sin \delta \cos Z}{\cos \delta \sin Z}$$

Similarly for the Zenith angle, we get the following:

$$\cos Z = \sin \phi \sin \delta + \cos \phi \cos \delta \cos t$$

where Z = zenith angle

Again, using the law of the propagation of variance:

$$\sigma^2_Z = \left(\frac{\partial Z}{\partial t}\right)^2 \sigma^2_t + \left(\frac{\partial Z}{\partial \phi}\right)^2 \sigma^2_\phi + \left(\frac{\partial Z}{\partial \delta}\right)^2 \sigma^2_\delta$$

where

$$\frac{\partial Z}{\partial t} = \cos\phi \sin A$$

$$\frac{\partial Z}{\partial \phi} = -\cos A$$

$$\frac{\partial Z}{\partial \delta} = \text{Cos } w$$

To estimate the variance in solar azimuth and zenith in decimal degrees, the uncertainties should also be in decimal degrees. For example, the permanent ephemeris subroutine (part of this package) from *The Astronomical Almanac* claims accuracies of:

Position	=	0.01	degrees (arc)
Equation of time	=	0.1	minutes (time)
	=	0.025	degrees (arc)

In the above example, this gives:

σ_t = 0.025 decimal degrees (arc)

σ_δ = 0.010 decimal degrees (arc)

σ_ϕ = 0.0 decimal degrees (arc)

The above partial differential equations have been programmed in the sunerror routine. No assumption about the distribution (e.g. normal) is required to propagate the variance. To derive confidence intervals, will require an assumption of normality (or some other distribution).

5.0 Computer Routines

The theory discussed above has been incorporated into a number of subroutines included in appendix B to this report. These routines and brief descriptions of their function are given below.

Routine name

sunephem ()

Purpose

To calculate ephemeris data for the sun, given the observer's position and date/time. The routine uses a permanent ephemeris for the period 1950 to 2050, as per page C24 of *The Astronomical Almanac*. Claimed accuracies are 0.1 minutes (time) for the equation of time, and 0.01 degrees for declination.

Routine name

sunazzd ()

Purpose

To calculate the azimuth and zenith angles to the sun given declination, hour angle and latitude. The zenith angle is from the centre of the earth. For some precise applications, the routine sunzdobs should be used to correct the zenith for parallax and refraction.

The azimuth returned is the so-called astronomic azimuth (i.e. clockwise angle from the observer's meridian). For some applications, a correction will be required to get grid bearing (ANG).

Routine name

sunprezd ()

Purpose

To calculate the local standard time that a particular solar zenith distance occurs. The user is able to set either the morning or afternoon time. This is useful for computing sun rise/set times and also length of daylight for agricultural applications.

Routine name

sunzdobs ()

Purpose

To correct the calculated zenith angle for parallax and refraction. The calculated zenith angle is returned from the sunazzd routine (part of this package). This is useful, if it is required to calculate the actual zenith angle that would have been observed by a surveyor. In (surveying) practice it is normal to go the other way, i.e. observe a zenith angle and correct it for parallax and refraction to the centre of the earth.

Routine name

calc_grid_conv ()

Purpose

To compute the grid convergence at a particular point given, Lat, Long and the longitude of the central meridian of the zone (Um).

Routine name

sunerror ()

Purpose

To compute a confidence interval for the predicted solar azimuth and zenith angles, given uncertainty in the input data. The confidence interval computed, is that of the input uncertainty data.

The routines are fully documented, and appendix B includes example programs of their use. They are written in ANSI standard C, and have been compiled on both IBM PCs and UNIX workstations without problems. Two useful programs for IBM PCs have been compiled called wasunup.exe and solarpos.exe

Their functions are as follows:

Program

solarpos.exe

Platform

IBM PC

Purpose

Calculates the solar azimuth and zenith given time, and the observers position. The program contains all appropriate prompts for user input, and output is listed to the screen.

Program

wasunup.exe

Platform

IBM PC

Purpose

Prepare an ASCII file (in tabulated format) of the length of day for each day of a user nominated year, at given latitudes. The output file is suitable for input into spreadsheets, or the preparation of tables.

6.0 Summary

The theory of calculating solar position has been described. Using this theory, a number of computer routines have been prepared in the C programming language. These routines are suitable for users who program in C, and have an ANSI compatible C compiler.

These routines are fully documented, with both the underlying theory, and code particulars, and may be used as a stand alone reference. The source code has also been published in appendix B of this report.

The routines are designed for any application that requires the position of the sun. This is essential, for understanding remotely sensed imagery in the visible and near infra-red portions of the electromagnetic spectrum. In particular, the current interest in change detection using multi temporal imagery requires that the effects of sun angle be considered, and modeled if relevant. These routines will be useful in that application.

The methodology to compute day length has been developed for agricultural applications. In particular, the methodology described by Keisling (1982) may be adopted, using these routines, for both field experimentation, and models of solar radiation availability for plant growth.

The incorporation of a permanent almanac for computing solar position, means the routines can operate as a stand alone data source on solar position. This almanac will generate ephemeris data which may be used to predict solar position within approximately 1' of arc. The precision of the estimate will vary depending on the particular geometric configuration. A methodology and routine have been developed that will estimate the precision of the solar position estimate.

The permanent ephemeris is suitable for the period 1950 to 2050. After the year 2050, the sunephem routine will need to be updated. Instructions for this updating are included in the source code.

Acknowledgments

Dr. D. Arbrecht pointed out the possibility of incorporating the routines into models of radiation availability for plant growth.

The contribution of the review team improved this paper. They included Dr. Doug Arbrecht and Dr. Ian Foster (Division of Plant Industry), Mr Alan McDonaugh (Division of Extension and Regional Operations) and Mr Graham Wright (Curtin University, Department of Surveying and Land Information). Mr Jim Dixon (Division of Extension and Regional Operations) coordinated all aspects of the review and his contribution is appreciated.

References

Anon. 1984. *The Astronomical Almanac*, US Government Printing Office and Her Majesty's Stationery Office.

Anon. 1973. *The Star Almanac for Land Surveyors*, Her Majesty's Stationery Office.

Garfinkel, B. 1967. Astronomical Refraction in a Polytropic Atmosphere, *The Astronomical Journal*, 72(3) :235-254.

Keisling,T.C. 1982. Calculation of the Length of Day, Agronomy Journal, 74:758-759.

Montenbruck, O. 1989. Practical Ephemeris Calculations, Springer Verlag, Berlin, 1989.

Walraven,R. 1978. Calculating the Position of the Sun, Solar Energy, 30:393-397.

Bibliography

For further reading on calculating solar position, any general textbook on Surveying or Astronomy may be consulted. Some recommended titles include:

Allan, A.L., Hollwey, J.R. and Mayne, J.H.B. 1968. Practical Field Surveying and Computations, Heinemann, London.

Davis, R.E., Foote, F.S., Anderson, J.M. and Mikhail, E.M. 1981. Surveying, Theory and Practice, McGraw-Hill.

Duffett-Smith, P. 1988. Practical Astronomy With Your Calculator, Cambridge University Press, Cambridge University.

APPENDIX A
TEST DATA FOR COMPUTING SOLAR POSITION

APPENDIX A. TEST DATA FOR COMPUTING SOLAR POSITION

Two example data sets are given for checking purposes.

Example 1

Test data source

From observations by the author during a surveying project in Brisbane, Queensland in 1984.

Input data

Date	=	2-2-1984
Local Std Time	=	17.610500
Time Zone	=	10.000000
Latitude	=	- 27.441389
Longitude	=	152.984444
LongCen_Mer	=	153.000000
Pressure	=	1013 mb
Temperature	=	25 deg celsius

Solar ephemeris data

JD	=	2445742.817104	Julian date
n	=	-5802.182896	No. Julian days since J2000.0
L	=	321.553514	Mean Long. of Sun(corr. for aberration)
g	=	38.894797	Mean anomaly of the sun
Lambda	=	322.775476	Ecliptic longitude
e	=	23.441321	Obliquity of the ecliptic
ra	=	325.122915	Right ascension of the Sun
decl	=	-13.924964	Declination
equation_time	=	356.430599	
hour_angle	=	83.572544	
Sun_dist_au	=	0.987105	
Sun_semi_diam	=	0.270488	

Uncertainty data

d_Latitude	=	0.000000
d_Longitude	=	0.000000
d_UT	=	0.000000
d_Equation_Time	=	0.025000
d_Declination	=	0.010000
Parallactic angle	=	115.644969

Result summary

Date	=	12-2-1984
Local Time/Time Zone	=	17.610/10.000
Latitude/Longitude	=	-27.441/152.984
True Azimuth	=	260.379
Grid Azimuth	=	260.372
Cen. Meridian	=	153.000
Grid Cony	=	-0.0072

Zenith (calc)	=		78.034	
Zenith (obs)	=		77.966	

68 percent confidence intervals

Az confidence	=	+-	0.0141	[decimal degrees]
ZD confidence	=	+-	0.0223	[decimal degrees]

where:

True Azimuth	=	true bearing
Grid Azimuth	=	AMG bearing
Zenith (calc)	=	zenith angle from centre of earth to centre of the sun
Zenith (obs)	=	zenith angle to centre of the sun as would be seen by an observer on the earth

Example 2

Test data source

Allan, Hollwey and Maynes (1968), page 678.

Note: This is a Northern Hemisphere example, so latitude is positive.

Input data

Date	=	24-9-1964	
Local Std Time	=	14.378833	
Time Zone	=	0.000000	
Latitude	=	51.591667	
Longitude	=	359.989583	
Long_Cen_Mer	=	0	
Pressure	=	1013 mb	
Temperature	=	20 deg celsius	

Solar ephemeris data

JD	=	2438663.099118	Julian Date
n	=	- 12881.900882	No. Julian Days since J2000.0
L	=	183.447889	Mean Long. of Sun(corr. for aberration)
g	=	261.122626	Mean anomaly of the sun
Lambda	=	181.561928	Ecliptic Longitude
e	=	23.444153	Obliquity of the ecliptic
ra	=	181.433044	Right Ascension of the Sun
decl	=	- 0.621356	Declination
equation_time	=	2.014845	
hour_angle	=	37.686928	
Sun_dist_au	=	1.002852	
Sun_semi_diam	=	0.266241	

Uncertainty data

d_Latitude	=	0.000000
d_Longitude	=	0.000000
d_UT	=	0.000000
d_Equation_Time	=	0.025000
d_Declination	=	0.010000
Parallactic angle	=	25.708240

Solar position results

Date	=	24-9-1964
Local Time/Time Zone	=	14.379 / 0.000
Latitude/Longitude	=	51.592 / 359.990
True Azimuth	=	224.283
Grid Azimuth	=	224.291
Cen. Meridian	=	0.000
Grid Cony	=	0.0082
Zenith (calc)	=	61.111
Zenith (obs)	=	61.085

68 percent confidence intervals

Az confidence	=	+-	0.0262	[decimal degrees]
ZD confidence	=	+-	0.0141	[decimal degrees]

where:

True Azimuth	=	true bearing
Grid Azimuth	=	AMG bearing
Zenith (calc)	=	zenith angle from centre of earth to centre of the sun
Zenith (obs)	=	zenith angle to centre of the sun as would be seen by an observer on the earth

APPENDIX B
SOURCE CODE FOR SUBROUTINES

APPENDIX B. SOURCE CODE FOR SUBROUTINES

Software developed by

Western Australian Department of Agriculture
Division of Resource Management
GIS Group - © 1992
Author: M. Roderick

As part of a general package for use in
predicting solar position.

A SUMMARY OF THE AVAILABLE ROUTINES IN THE 'SUN' PACKAGE

Computer Language - ANSI Standard C

Discussion

A series of subroutines have been prepared for use in predicting the solar azimuth and zenith angles, given the observers position and the time. These routines are useful for Remote Sensing studies, and may be of interest to Agricultural Scientists and others who need to compute the solar position. In addition, a routine has been programmed (sunprezd) which allows Agricultural Scientists to compute the length of daylight.

The routines use a permanent ephemeris for the sun (per *The Astronomical Almanac*), and will predict the solar position at any time during the period 1950 to 2050 inclusive. A typical accuracy is 1' (i.e. minute) of arc. The accuracy varies according to the particular geometry, and estimates of the uncertainty in solar azimuth and zenith are also computed.

All routines are fully documented in the source code. Please refer to this code for further details. The code is written in ANSI Standard C, and has been kept as simple as possible.

All angle input is in decimal degrees. For example, an angle of 90 deg 34 mm (arc) is 90.833 degrees. Units and other details are given in the remarks for each routine.

Please pay particular attention to the use of pointers in the routines. If you are not familiar with the C programming language and the use of pointers, now is not the time to learn! There are numerous good text books on the C programming language.

Software availability

The software is available as a self extracting archive from the Western Australian Department of Agriculture - GIS Group. Files include:

- readme.2 [instructions]
- sunarc2.exe [self extracting archive]

All compiled programs and source code are available in the self extracting archive.

Support/warranty

No support or warranty on the software is provided.

Routine name

sunephem ()

File

suriephem.c

Purpose

To calculate ephemeris data for the sun, given the observer's position and date/time.

Syntax

```
void sunephem ( int year, int month, int day, double lstdt_dec, double time_zone_dec, double stn_lat_dec,
                double stn_long_dec, int printdataswitch, double *dec1, double *equation time, double *hour
                angle,
                double *sun_dist_au, double *sun_semi_diam );
```

Remarks

Variable usage is as follows

```
int      year,              /*  <=  input year              (e.g. 1986)                                  */
int      month,             /*  <=  input month             (e.g. 3)                                     */
int      day,               /*  <=  input day               (e.g. 12)                                    */
double   lstdt_dec,         /*  <=  input local std time in hrs                                          (e.g.
14.2341) */
                                        (decimal hours)
double   time_zone..dec,    /*  <=  input time zone         (e.g. Perth WST = + 8.00)                    */
double   stn_lat_dec,       /*  <=  input latitude of stn   (e.g.  - 27.127)
                                        South is -ve, North is +ve. (decimal deg)                        */
double   stn_long_dec,      /*  <=  input longitude of stn  (e.g. 120.141)
                                        East is +ve, West is -ve (decimal deg)                           */
int      printdataswitch,   /*  <=  input causes the ephemeris data to be printed.
                                        1 for printing, =0 for no printing.                              */
double   *decl,             /*  =>  returns a pointer to decl (decimal deg)                              */
double   *equation_time,    /*  =>  returns a pointer to equation_time (decimal deg)                     */
double   *hour_angle,       /*  =>  returns a pointer to hour_angle (decimal deg)                        */
double   *sun_dist_au,      /*  =>  returns a pointer to earth sun distance in AU
                                       (astronomical units)                                              */
double   *sun_semi_diam     /*  =>  returns a pointer to sun semi-diameter in decimal degrees            */
```

Demo program/s

solarpos.c

Routine name

sunazzd ()

File

sunazzd.c

Purpose

To calculate the azimuth and zenith angles to the sun given declination, hour angle and latitude. The zenith angle is from the centre of the earth. For some precise applications, the routine sunzdobs should be used to correct the zenith for parallax and refraction.

See the notes in the sunzdobs routine.

The azimuth returned is the so-called astronomic azimuth) i.e. clockwise angle from the observer's meridian). For some applications, a correction will be required to get grid bearing (AMG) This is discussed in the technical report published by the Western Australian Department of Agriculture.

Syntax

```
void sunazzd    ( double hour_angle, double sun_decl, double stn_lat_dec, int printdataswitch, double
                  *azimuth,
                  double *zenith_calc);
```

Remarks

Variable usage is as follows:

```
double   hour_angle,      /*  <=   input hour_angle (decimal degrees)
                                    +ve west, -ve east                          */
double   decl,            /*  <=   input the declination to sun (decimal degrees)
                                    +ve North, -ve South                        */
double   stn_lat_dec,     /*  <=   input latitude of stn      (e.g.  - 27.127)
                                     South is -ve, North is +ve. (decimal deg)  */
int      printdataswitch, /*  <=   input causes the ephemeris data to be printed.
                                     1 for printing, =0 for no printing.        */
double   *azimuth,        /*  =>   returns a pointer to azimuth (decimal deg)   */
double   *zenith_calc,    /*  =>   returns a pointer to zenith_calc (the zenith angle to the
                                     sun from the centre of the earth)          */
                                     in decimal degrees                         */
```

Demo program/s

solarpos.c

Routine name

sunprezd ()

File

sunprezd.c

Purpose

To calculate the local standard time that a particular solar zenith distance occurs. The user is able to set either the morning or afternoon time. This is useful for computing sun rise/set times and also length of daylight for agricultural applications.

Syntax

```
void sunprezd    ( int year, int month, int day, double time_zone_dec, double stn_lat_dec, double stn_long_dec,
                   int printdataswitch, int sun_pos, double requ_zenith, double *lstdt_dec, double *sun_dist_au);
```

Remarks

Variable usage is as follows:

```
int      year,             /*  <=  input year  (e.g. 1986)                                  */
int      month,            /*  <=  input month (e.g. 3)                                     */
int      day,              /*  <=  input day (e.g. 12)                                      */
double   time_zone_dec,    /*  <=  input time zone (e.g. Perth WST = +8.00)                 */
double   stn_lat_dec,      /*  <=  input latitude of stn       (e.g.  - 27.127)
                                       South is -ve, North is +ve. (decimal deg)            */
double   stn_long_dec,     /*  <=  input longitude of stn     (e.g.  120.341)
                                       East is +ve, West is –ve     (decimal deg)           */
int      printdataswitch,  /*  <=  input causes the output data to be printed.
                                       = 1 for printing, =0 for no printing.                */
int      sun_pos,          /*  <=  input sun position.  Sun rise  =  -1
                                       Sun set  =  1.                                       */
double   requ_zenith,      /*  <=  input zenith angle for which the local
                                     standard time is required (decimal degrees)            */
double   *lstdt_dec,       /*  =>  returns a pointer to local std time in hrs
                                     (e.g. 5.781, 17.812 in decimal hrs)
                                     that the requ_zenith occurs                            */
double   *sun_dist_au,     /*  =>  returns a pointer to distance to the sun for the day
                                       in question.
                                       (In AU where 1 AU = 149 597 870 000 metres.)         */
```

Demo program/s

wasunup.c

Routine name

sunzdobs ()

File

sunzdobs.c

Purpose

To correct the calculated zenith angle for parallax and refraction. The calculated zenith angle is returned from the sunazzd routine (part of this package). This is useful, if it is required to calculate the actual zenith angle that would have been observed by a surveyor. In (surveying) practice it is normal to go the other way, i.e. observe a zenith angle and correct it for parallax and refraction to the centre of the earth.

Syntax

```
void sunzdobs    ( double zenith_calc, double pressure, double temperature, int printdataswitch, double
                   *zenith_obs);
```

Remarks

Variable usage is as follows:

```
double   zenith_calc,      /*  <=   input the zenith_calc from the sunazzd routine
                                        (in decimal degrees)                                  */
double   pressure,         /*  <=   input the pressure at the earth's surface (in milli-bars)   */
double   temperature,      /*  <=   input the temperature at the earth's surface (in degrees celsius)   */
int      printdataswitch,  /*  <=   input causes the output data to be printed.
                                        = 1 for printing, =0 for no printing.                 */
double   zenith_obs,       /*  =>   returns a pointer to zenith_obs, the corrected
                                       zenith in decimal degrees                              */
```

Demo program/s

solarpos.c

Routine name

Calc_grid_conv ()

File

suntools.c

Purpose

To compute the grid convergence at a particular point given, Lat, Long and the longitude of the central meridian of the zone (UTM).

Syntax

```
void calc_grid_conv             (double stn_lat_dec, double stn_long_dec, double cen_mer_long, int
                    printdataswitch,
                            double *grid_conv);
```

Remarks

Variable usage is as follows:

```
double   stn_lat_dec,      /*  <=   input latitude of stn       (e.g.  - 27.127)
                                    South is -ve, North is +ve. (decimal degrees)          */
double   stn_long_dec,     /*  <=   input longitude of stn     (e.g.  120.341)
                                    East is +ve, West is –ve     (decimal degrees)         */
double   cen_mer_long,     /*  <=   the longitude of the central meridian (decimal degrees)  */
int      printdataswitch,  /*  <=   input causes the output data to be printed.
                                    = 1 for printing, =0 for no printing.                  */
double   *grid_conv,       /*  =>   returns a pointer to the grid convergence (decimal degrees) */
```

Demo program/s

solarpos.c

Routine name

sunerror ()

File

sunerror.c

Purpose

To compute a confidence interval for the predicted solar azimuth and zenith angles, given uncertainty in the input data. The confidence interval computed, is that of the input uncertainty data.

Syntax

```
void sunerror        (double hour_angle, double decl, double stn_lat_dec, double azimuth, double
                      senith_calc,
                       double d_lat, double d_long, double d_UT, double d_equation_time, double d_decl,
                        int printdataswitch, double *az_ci, double *zd_ci );
```

Remarks

Variable usage is as follows:

```
double   hour_angle        /*  <=  input hour_angle (decimal degrees)
                                    +ve west, -ve east                                  */
double   decl,             /*  <=  input the declination to sub (decimal degrees)
                                    +ve North, -ve South                                */
double   stn_lat_dec,      /*  <=  input latitude of stn       (e.g.  - 27.127)
                                    South is -ve, North is +ve. (decimal degrees)       */
double   azimuth,          /*  <=  input azimuth (decimal degrees)                     */
double   zenith,           /*  <=  input zenith (decimal degrees)                      */
double   d_lat,            /*  <=  input estimated error in latitude (decimal degrees) */
double   d_long,           /*  <=  input estimated error in longitude (decimal degrees)*/
double   d_UT,             /*  <=  input estimated error in Universal Time
                                    (in decimal degrees of arc, where arc  =  time*15)  */
double   d_equation_time,  /*  <=  input estimated error in equation time
                                    (in decimal degrees of arc, where arc  =  time*15)  */
double   decl,             /*  <=  input estimated error in declination
                                    (in decimal degrees of arc, where arc  =  time*15)  */
int      printdataswitch,  /*  <=  input causes the output data to be printed.
                                    = 1 for printing, =0 for no printing.               */
double   *az_ci,           /*  =>  returns a pointer to azimuth confidence interval
                                    (in decimal degrees of arc)                         */
double   *zd_ci,           /*  =>  returns a pointer to zenith confidence interval
                                    (in decimal degrees of arc)                         */
```

Demo program/s

solarpos.c
errtest.c

demonstration programs

ERRTEST.C

SOLARPOS.C

WASUNUP.C

ZENTEST.C

```
/*      Program name    :   errtest.c
        Revisions       :
```

Version	Date	By	Details
1.0	15-02-1992	mlr	The original subroutine

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include “sunazzd.c”
#include “sunerror.c”
#include “suntools.c”

*/ prototype definitions, i.e. ANSI C */
void errtestinfo(void);

void main()
{

    int         printdataswitch;
    double      stn_lat_dec;
    double      decl, equation_time, hour_angle;
    double      azimuth, zenith_calc;
    double      d_lat, d_long, d_UT, d_equation_time, d_decl;
    double      az_ci, zd_ci;

    /* set up the screen */
    my_slow_clear_screen();
    wada_gis_screen();
    errtestinfo();

printf(“\n\Enter Station Latitude                    [e.g.   -32.784 in decimal]  >  “);
    scanf(“%lf”, &stn_lat_dec);
    printfl“Enter Declination                       [e.g.   116.921 in decimal]  >  “);
    scanf(“%lf”, &decl);
    printf(“\nEnter Hour Angle)                     [e.g.    32.143 in decimal]  >  “);
    scanf(“%lf”, &hour_angle);

    /* enter uncertainty data */
    printf(“\nEnter uncertainty in Lat              [e.g.      0.01 in decimal]  >  “);
    scanf(“%lf”, &d_lat);
    printf(“\nEnter uncertainty in Long             [e.g.      0.01 in decimal]  >  “);
    scanf(“%lf”, &d_long);
    printf(“\nEnter uncertainty in UT               [e.g.dec degrees arc]        >  “);
    scanf(“%lf”, &d_UT);
    printf(“\nEnter uncertainty in Equ Time         [e.g.dec degrees arc]        >  “);
    scanf(“%lf”, &d_equation_time);
    printf(“\nEnter uncertainty in decl             [e.g.dec degrees arc]        >  “);
    scanf(“%lf”, &d_decl);

    printf(“\nDo you want all the data echoed to the screen  [1=yes 0=no]       >  “);
    scanf(“%d”, &printdataswitch);
```

```
sunazzd  (hour_angle, decl, stn_lat_dec, printdataswitch,
            &azimuth, &zenith_calc);

sunerror  (hour_angle, decl, stn_lat_dec, azimuth, zenith_calc, d_lat,
            d_long, d_UT, d_equation_time, de_decl, printdatawitch,
            &az_ci, &zd_ci );

/* results */
errtestinfo();
printf("\n\nResult Summary:\n");
printf("---------------\n");
printf("Latitude              =  %7.3lf\n",   stn_lat_dec);
printf("Declination           =  %7.3lf\n",   decl);
printf("Hour Angle            =  %7.3lf\n",   hour_angle);
printf("True Azimuth          =  %7.3lf\n",   azimuth);
printf("Zenith (calc)         =  %7.3lf\n",   zenith_calc);
printf("Confidence Intervals.\n");
printf("----------------------------\n");
printf("Az confidence         =  +-  %7.4lf\n",   az_ci);
printf("ZD confidence         =  +-  %7.4lf\n",   zd_ci);
printf("where  :  \n");
printf("True Azimuth          =  true bearing\n");
printf("Zenith (calc)         =  zenith angle from centre of earth\n");
printf("                         to centre of the sun\n\n");

}

/*-------------------------------------------------------------------------------------------------*/
void errtestinfo(void)
{
 printf("\nerrtest          v1.0   -   A program to test propogation of variance\n");
    printf("                                    techniques for estimating the error in solar\n");
    printf("                                    is that of the input data.\n");
}
```

```
/*      Program name    : solarpos.c
        Purpose         : To compute the solar position at a given instant in time given the local std time, time
                          zone, date,
                          latitude and longitude.  A 68 per cent confidence interval is also estimated using the
                          sunerror routine.  This program is an example of how to use the routines in your own
                          programs.
        Author          : M.L. Roderick
                          Western Australian Department of Agriculture
                          GIS Group.
        Date            : 06-02-1992
        Language        : ANSI Standard C
        Compilation     :
        Limitations     : a)  The ephemeris data (see routine sunephem documentation) can only predict solar
                              position between 1950 and 2050.
                          b)  See the routines  -  sunephem
                                                -  sunazzd
                                                -  grid_conv
                              for further details.
        Revisions       :
```

Version	Date	By	Details
1.0	06-02-1992	mlr	The original subroutine
1.1	15-02-1992	mlr	Update for changes in sunazzd routine and addition of sunerror routine.

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include “sunazzd.c”
#include “sunephem.c”
#include “sunzdobs.c”
#include “sunerror.c”
#include “suntools.c”

*/ prototype definitions, i.e. ANSI C */
void solarposinfo(void);

void main()
{
        int     year, month, day, printdataswitch;
        double      lstdt_dec, time_zone)_dec, stn_lat_dec; stn_long_dec;
        double      decl, equation_time, hour_angle;
        double      azimuth, zenith_calc;
        double      sun_dest_au, sun_semi_diam;
        double      grid_conv, cen_mer_long;
        double      pressure, temperature, zenith_obs;
        double      d_lat, d_long, d_UT, d_equation_time, d_decl;
        double      az_ci, zd_ci;

    /* set up the screen */
    my_slow_clear_screen();
    wada_gis_screen();
    solarposinfo();
```

```
printf("\n\Enter date                          [dd mm yyyy]          > ");
scanf("%d %d %d", &day, &month, &year);
printfl"\nEnter Local Time                    [decimal]              > ");
scanf("%lf", &lstdt_dec);
printf("\nEnter Time Zone                     [e.g.  +8 for Perth WST, +9 daylight saving    > ");
scanf("%lf", &time_zone_dec);

printf("\n\Enter Station Latitude             [e.g.  -32.784 in decimal]     > ");
scanf("%lf", &stn_lat_dec);
printf("\nEnter Station Longitude)            [e.g.  116.921 in decimal]     > ");
scanf("%lf", &stn_long_dec);

printf("\nCen Mer  =  117, Zone 50 ;Cen Mer  =  123, Zone 51");
printf("\nCen Mer  =  129, Zone 52 ;… etc …\n");
printf("\nEnter central meridian for AMG zone  [e.g. 117]                    > ");
scanf("%lf", &cen_mer_long);

printf("\n\Enter Pressure in mb               [e.g.  1013]                   > ");
scanf("%lf", &pressure);
printf("\nEnter Temperature in deg celcius)   [e.g.  25]                     > ");
scanf("%lf", &temperature);

printf("\n\Do you want all the data echoed to the screen          [1=Yes 0=No]                    > ");
scanf("%f", &printdataswitch);

/* set up variables for calculation of uncertainty */
/* consider no error in time, or position.  Thus the only uncertainty is that due to ephemeris errors.  For a
Surveying     application, the user could be prompted for entry of the estimated errors in time and position.

d_lat          =  0;
d_long         =  0;
d_UT           =  0:
/* see notes in routine sunerror.c for details of the following data.
   They are assumed to represent a 68% confidence internal */
d_equation_time   =                                                    0;
d_decl         =  0:

sunephem       (year, month, day, lstdt_dec, time_zone_dec, stn_lat_dec,
                 stn_long_dec, printdatawitch,
                 &decl, &equation_time, &hour_angle, &sun_dist_au,
                 &sun_semi_diam );

sunazzd        (hour_angle, decl, stn_lat_dec, printdatawitch,
                 &azimuth, &zenith_calc);

calc_grid_conv (stn_lat_dec, stn_long_dec, cen_mer_long, printdatawitch,
                 &grid_conv);

sunzdobs       (zenith_calc, pressure, temperature, printdatawitch,, &zenith_obs);

sunerror       (hour_angle, decl, stn_lat_dec, azimuth, zenith_calc, d_lat,
                 d_long, d_UT, d_equation_time, de_decl, printdatawitch,
                 &az_ci, &zd_ci );

/* results */
solarposinfoo();
printf("Result Summary:\n");
printf("--------------------\n");
printf("Date                      =      %2d-%2d-%4d\n", day, month, year);
printf("Local Time / Time Zone    =      %7.3lf / %7.3lf\n", lstdt_dec, time_zone_dec);
printf("Latitude / Longitude      =      %7.3lf / %7.31f\n", stn_lat_dec, stn_long_dec);
```

```
    printf("True Azimuth   =  %7.31f\n", azimuth);
    printf("Grid Azimuth   =  %7.3lf\n", azimuth + grid_cony);
    printf("Cen. Meridian  =  %7.3lf\n", cen_mer_long);
    printf("Grid Conv      =  %7.4lf\n", grid_cony);
    printf("Zenith (calc)  =  %7.3lf\n", zenith_calc);
    printf("Zenith (obs)   =   Z7.3lf\n", zenith_obs);
    printf("68 percent Confidence Intervals.\n");
    printf("Az confidence  =  +- %7.4Lf\n", az_ci);
    printf("ZD confidence  =  +- %7.41f\n", zd_ci);
    printf("where :\n");
    printf("True Azimuth   =  true bearing\n");
    printf("Grid Azimuth   =  AMG bearing\n");
    printf("Zenith (calc)  =  zenith angle from centre of earth\n");
    printf("                  to centre of the sun\n");
    printf("Zenith (obs)   =  zenith angle to centre of the sun as would\n");
    printf("                  be seen by an observer on the earth\n");
}
/*-------------------------------------------------------------------------------------------------------------*/
void solarposinfo(void)
{

printf("\nsolarpos vl.1     -  A program for computing the solar position\n");
   printf("                     during the period 1950 to 2050.\n");
}

^Z
```

```
/*      Program name    : wasunup.c
        Purpose         :  To prepare an ASCII file of the day length as referenced to a particular solar zenith
                           angle.  This is designed for use by agricultural scientists interested in modelling the
                           amount of PAR (photosynthetically active radiation) available to a plant canopy.
        Author          :  M.L. Roderick
                           Western Australian Department of Agriculture
                           GIS Group.
        Date            :  05-02-1992
        Language        :  ANSI Standard C
        Compilation     :
        Limitations     :  None known
        References      :  a)  For the rule to determine leap years, Montenbruck, O. (1989).  Practical Ephemeris
                               Calculations, Springer-Verlag, Berlin, 1989 on page 33.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include “sunprezd.c”
#include “suntools.c”

*/ prototype definitions, ie ANSI C */
void wasunupinfo(void);

void main()
{
        int         year, month, day, printdataswitch;
        double      time_zone_dec, stn_lat_dec, stn_long_dec;
        double      requ_zenith;
        double      morn_sun_dist_au, afternoon_sun_dist_au;
        int         sun_pos;    /*  =1 for sun_set, =-1 for sun_rise */
        double      morning_time, afternoon_time;
        FILE        *outflptr;
        char        outputfile[80];
        int         days_per_month [13];
        double      max_lat, min_lat, increment_lat, curr_lat;

        /* set up array for     days in each month */
        days_per_month [1] =31;
        days_per_month [2] =28;
        days_per_month [3] =31;
        days_per_month [4] =30;
        days_per_month [5] =31;
        days_per_month [6] =30;
        days_per_month [7] =31;
        days_per_month [8] =31;
        days_per_month [9] =30;
        days_per_month [10]    =31;
        days_per_month [11]    =30;
        days_per_month [12]    =31;
```

```
/*      Daylength can be predicted by the time that the hour_angle of the sun reaches a certain zenith distance.
        Because the hour_angle is a function of the equation_time, we need to estimate the equation_time from
        the UT. Thus the routine needs the time_zone to get the UT.  The longitude won't actually affect anything
        so we can set the longitude at a default as shown below.

        While the time zone will change with daylight saving, this will not affect the calculations for daylength.
*/

/* declare some default settings */
stn_long_dec=117;
printdataswitch=0;

/* set up the screen */
my_slow_clear_screen();
wada_gis_screen();
wasunupinfo();

printf("\nEnter year for this program run          [e.g. 1992]          > ");
scanf ("%d", &year);

printf("\nEnter Time Zone (eg.+8 for Perth WST, +9 daylight saving )    > ")'
scanf ("%lf", &time_zone_dec);

/* check for leap year and add 1 to February if it is */
/* per Montenbruck, p33. "Leap year is every year whose yearly number
   is divisible by four, but not by a hundred, or is divisble by 400". */
{
   int year_4, year_100, year_400, leap_year;

   year_4     =   year/4;
   year_100 =   year/100;
   year_400 =   year/400;

   leap_year =   0;
   if    (!year_4*4 – year))
        {
          leap_year  =       1;
          if (!(year_100*100  -  year))
             {
               leap year = 0;
               if (!(year_400*400  -  year))
                  {
                    leap_year=1;
                  }
             }
        }
   if(leap_year)
   {
     days_per_month[2]   =29;
   }
}

printf("Enter solar zenith angle that defines the time \n");
printf("of sun rise-set in decimal degrees [e.g. 90.833] > ");
scanf("%lf", &requ_zenith);

printf("The program calculates the data for number of latitudes.\n");
printf("Enter lowest latitude for calculation [e.g.  -36]    > ");
scanf("%lf", &min_lat);
```

```
printf("Enter highest latitude for calculation [eg.  -26]    > ")
scanf("%lf", &max_Lat);
printf("Enter Latitude increment [e.g. 2 ]                   > ")
scanf("%lf", &increment_lat);

/* get filename and open it in text mode for output */
printf("Enter output file name > ");
scanf("%s", outputfile);

if ((outflptr=fopen(outputfile,"wt"))==NULL)
    {
    printf("Error opening output file %s\n", outputfile);
    printf("... aborting …\n\n");
    exit(1);
    }
/* print up header data in the output file */
fprintf(outflptr, "Day Length calculations for Solar Zenith = %7.31f\n", requ_zenith);
/* set loop variable for below */
curr_lat = min_lat;

fprintf(outflptr,"                    \tLATITUDE\n");
fprintf(outflptr,"DAY MONTH YEAR \t");

  do
  {
   fprintf(outflptr," %5.2lf ", curr_lat);
   curr_lat = increment_lat + curr_lat;
  } while (curr_lat < max_lat);
fprmntf(outflptr," Mean Earth_Sun Dist (AU)\n");

/* now do the calculations, and print to the file */
for(month=1; month < 13; month++)
  {
   for(day=1; day < days_per_month [month] + 1 ; day++)
      {
       fprmntf(outflptr,"\n%3d %5d %4d \t", day, month, year);
       /* set parameters for Loop below */
       curr_lat = min_lat;

      do
          {
           sun_pos = -1;
           sunprezd (       year, month, day, tirne_zone_dec, curr_lat, stn_long_dec,
           printdataswitch, sun_pos, requ_zenith, &morning_time,
                &morn_sun_di st_au);

           sun_pos = 1;
           sunprezd (       year, month, day, time_zone_dec, curr_lat, stn_long_dec,
           printdataswitch, sun_pos, requ_zenith, &afternoon_time,
           &afternoon sun_di st_au);

           printf(".");
           fprintf(outftptr," %5.2Lf ", afternoon_time - morning_time);

           /* increment the latitude */
           curr_lat = increment_lat + curr_lat;
```

```
            } while (curr_lat <= max_lat);

            fprintf(outflptr," %6.4lf ", (morn_sun_dist_au + afternoon_sun_dist_au)/2);
        }

        printf("\nCompleted Month No. %d\n", month);
    }

}

/*-------------------------------------------------------------------------------------------------*/
void wasunupinfo(void)
 {
        printf(uu\n*** wasunup V1.0 ***\n");
        printf("\nA program to compute the length of daylight hours.\n");
            printf("This program computes the day Light hours for every day of a user\n");
            printf("nominated year. The output is an ASCII file which may be input into\n");
            printf("spreadsheets or other modelling programs.\n");
            printf("The program is designed specifically for the use of Agricultural\n");
            printf("Scientists in estimating solar radiation. The mean Earth_Sun\n");
            printf("distance is also calculated. This may be prove useful in radiation\n");
            printf("studies. The output file is in the form of a table.\n");

}
```

```
/*      Program name    : zentest.c
        Purpose         :   To test the refraction/parallax corrections to
                            zenith distance in routine subzdobs.
        Author          :   M.L. Roderick
                            Western Australian Department of Agriculture
                            GIS Group.
        Date            :   06-02-1992
        Language        :   ANSI Standard C
        Compilation     :
        References      :   a)  Refraction corrections are uncertain near the
                                horizon (zenith = 90 degrees).  This is a general
                                problem in astronomy and surveying, and cannot
                                (easily) for overcome.

        Revisions       :
```

Version	Date	By	Details
1.0	06-02-1992	mlr	The original subroutine

```
*/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include “sunzdobs.c”
#include “suntools.c”


/* prototype eclarations here i.e. ANSI C */

void zentestinfo(void);

void main()

{
    in          printdataswi tch;
    double      zenith_calc, zenith_obs;
    double      pressure, temperature;
    int         icntr;

    /* don’t want all the data echoed to the screen */
    printdataswitch = 0;

    /* set up the screen */
    my_slow_clear_screen();
    wada_gis_screen();
    zentestinfo();

    printf(“\nEnter Pressure [mb]                 > “);
    scanf (“%lf”, &pressure);
```

```
    printf("\nEnter Temperature [deg C]                > ");
    scanf("%lf", &temperature);

    printf("\n\nP\tT\tZD(c)\tZD(obs)\t\tDiff(\")\n");

    for(icntr=0; icntr < 100; icntr = icntr+15)
        {
          zenith_calc = icntr;
          sunzdobs(zenith_calc, pressure, temperature, printdataswitch, &zenith_obs);
          printf("%5.1lf\t%3.1lf\t%4.4lf\t%4.4lf\t\t%5.2lf\n", pressure, temperature, zenith_calc,
            zenith_obs, (zenith_calc~zenith_obs)*3600);
        {

        printf("\n\nwhere ZD(c)    =   zenith angle calculated to centre of true sun\n");
          printf("       ZD(obs)    =   zenith angle corrected for refraction and\n");
          printf("                      parallax.\n");
          printf("       Diff(\")       =   ZD(c) - ZD(obs)      in seconds of arc\n");
}


/*-------------------------------------------------------------------------------------------------------*/
void zentestinfo(void)
{
    printf("\n\nzentest vi.0          -   test refraction/parallax routines\n");
          printf("                      in the sunzdobs module\n\n");
}
/*-------------------------------------------------------------------------------------------------------*/
```

SOURCE CODE LISTINGS FOR CORE MODULES

SUNAZZD.C

SUNEPHEM.C

SUNERROR.C

SUNPREZD.C

SUNTOOLS.C

SUNZDOBS.C

/* Program name : sunazzd.c
Purpose : To calculate the azimuth and zenith angles to the sun given declination, hour angle and latitude. The zenith angle is from the centre of the earth. For some precise applications, the routine sunzdobs should be used to correct the zenith for parallax and refraction. See the notes in the sunzdobs routine. The azimuth returned is the so-called astronomic azimuth (i.e. clockwise angle from the observer's meridian). For some applications, a correction will be required to get grid bearing (AIIG). This is discussed in the technical report published by the Western Australian Department of Agriculture.

Author : M.L.Roderick
Western Australian Department of Agriculture
GIS Group

Date : 28-1-92

Language : ANSI Standard C

Compilation :

Limitations : None known

References :

General astronomy principles

Glassock, J.T.C. (1983). Lecture Notes - Land Surveying VI - Astronomy. Queensland Institute of Technology, Brisbane, 1983.

Davis, R.E., Foote, F.E., Anderson, J.M. and Mikhail,E.M. (1981). Surveying, Theory and Practice. McGraw-Hill, New York, 1981.

or any other numerical astronomy or surveying textbook.

General theory

Azimuth:

$$\tan(\text{Azimuth}) = \frac{-\sin(\text{hour\_angle})}{\tan(\text{decl})\cos(\text{lat}) - \sin(\text{lat})\cos(\text{hour\_angle})}$$

where hour_angle = time since sun was on the observer's meridian
= UT + Long + equation_time - 180 (decimal degrees)
long = observer's longitude
equation_time = true sun - mean sun
decl = declination of the sun
Lat = latitude of station

The hour_angle and declination are computed by the sunephemn routine, which is part of this package.

The above formula is a general solution for Azimuth of the Astronomical Triangle.

To solve for Azimuth it is best to use the atan2 function available in all ANSI standard C compilers.

Zenith distance

Note: Zenith Distance = 90 - elevation

cos(ZD) = sin(lat)*sin(decl) + cos(lat)*cos(decl)*cos(hour_angle)

This computes the "theoretical[1]" Zenith Distance (zenith_calc) to the centre of the Sun from the centre of the earth. For some applications, it will be necessary to correct the ZD (zenith_calc) to a value that would have been observed on the earth's surface, and affected by refraction. This is done using the sunzdobs routine.

The routine sunzdobs (part of this package) can be used to correct the zenith distance calculated in this routine, to derive an actual observed zenith distance.

The suns rays are refracted in the atmosphere. The actual zenith distance, is smaller than the 'theoretical' version. Atmospheric refraction, is a function of temperature, pressure, and the zenith distance. It is a maximun at the horizon, (i.e. zenith = 90 deg). Typical values are:
34' at Zenith of 90 deg.
5' at Zenith of 80 deg.
1' at Zenith of 45 deg.
< 1' at Zenith less than 45 deg.

Whether you will want the theoretical (zenith_calc) or actual zenith will depend on the application. For remote sensing studies, either will probably be sufficient, although you should use the actual value. The difference will be minor for solar zenith angles < 70 degrees (say 2').

For determining the sun rise/set times, the theoretical value will be appropriate. Typical zenith distances for sun rise/set are given as 90 deg 50' (i.e. you can see the sun 50' below the horizon). This is made up of 34' refraction, and 16' for the half-width of the sun (i.e. semi_diam_sun). The increased refraction near the horizon,
is caused principally by the increased path length of solar radiation through the atmosphere, and refraction of light rays through atmospheric boundary layers.

Revisions :

Version	Date	By	Details
1.0	28-1-1992	mlr	The original subroutine
1.1	15-02-92	mlr	Removed uncertainty calc's and put them in a separate routine called sunerror.

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/* Constants rounded for 21 decimals. */
#define MY_PI      3.14159265358979323846

/* prototype definition  -  i.e.  ANSI C declaration */
void sunazzd (       double hour_angle, double sun_decl, double stn_lat_dec,
                     int printdataswitch, double *azimuth, double *zenith_calc);

/* the subroutine */

void sunazzd
```

```
(
    double    hour_angle,        /*  <=   input hour_angle ( decimal degrees)
                                          +ve west, -ve east                    */
    double    decl,              /*  <=   input the declination to sun (decimal degrees)
                                          +ve North, -ve South                  */
    double    stn_lat_dec,       /*  <=   input latitude of stn (e.g. –27.127)
                                          South is –ve, North is +ve. (decimal degrees)  */
    int       printdataswitch,   /*  <=   input causes the ephemeris data to be printed.
                                          =1 for printing, =0 for no printing   */
    double    *azimuth,          /*  =>   returns a pointer to azimuth (decimal degrees)  */
    double    *zenith_calc       /*  =>   returns a pointer to zenith_calc (the zenith)
                                          angle to the sun from the centre of the earth)
                                          in decimal degrees                    */
)

{
    double    deg2rad,   rad2deg;

    /* some useful constants */
    deg2rad = MY_PI / 180;
    rad2deg = 180 / MY_PI ;

    /* convert required quantities to radians */
    hour_angle = hour_angle * deg2rad;
    decl = decl * deg2rad;
    stn_lat_dec = stn_lat_dec * deg2rad;

    /* compute azimuth */
{
    double numerator, denominator;

    numerator    =   -1*sin(hour_angle);
    denominator=   (tan(decl)*cos(stn_lat_dec))-(sin(stn_lat_dec)*cos(hour_angle));

    if (denominator == 0)
    {
      if (numerator > 0)
         {
           *azimuth = (MY_PI/2);
         }
    else if (numerator < 0)
         {
           *azimuth = (MY_PI * 3)/2;
         }
    else if (numerator == 0)
         {
           *azimuth = 0;
         }
    }
else
    {
      *azimuth = atan2(numerator, denominator);
     jf (*azimuth < 0)
         {
           *azimuth = *azimuth + (2*MY_PI);
         }
    }
}
```

```
/* compute zenith distance */
{
       double cl;

       ci = (smn(stn_lat_dec)*sin(decl))+(cos(stn_lat_dec)*cos(decl)*cos(hour_angle));
       *zenithcalc = acos(ci);

}


/* convert azimuth and zenith to degrees */
*azimuth    =   *azimuth * rad2deg;
*zenithcalc =   *zenith_calc * rad2deg;


/* convert original data back to degrees for printing out */
 hour_angle = hour_angle * rad2deg;

     decl = decl * rad2deg;
     stn_lat_dec = stn_lat_~dec * rad2deg;


    /* Print out summary *1
    if (printdataswitch)
    {
      printf("\n\nSummary of Calculations\n");
      printf("                                    \n");
      printf("Input Data :\n");
      printf("                  \n");
      printf("Latitude          =%lf\n",      stn_lat_dec);
      printf("Hour Angle (ha)   =   %lf\n", hour_angle);
      printf("Declination       =   %lf\n", decl);
      printf("Calculated Data   :\n");
      printf("-------------------------\n");
printf("Azimuth                 =   %lf\nuu, *azimuth);
      printf("Zenith_calc     =   %lf\n", *zenithcalc);
    }

}
```

/* Program name : sunephem.c
Purpose : To calculate ephemeris data for the sun, given the observer's position and date/time.
Author : M.L. Roderick
Western Australian Department of Agriculture
GIS Group.
Date : 28-1-92
Language : ANSI Standard C
Compilation :
Limitations :
a) The Julian Date algorithm used (Montenbruck, p33-34) is suitable for all dates after 15-10-1582.
b) The ephemeris data for the sun is taken from "The Astronomical Almanac". These formula are approximate only for the period 1950-2050. After the year 2050, please consult the almanac for updated formula.
c) The ephemeris data will predict the for the sun is taken from "The equation_time to +- 0.1 minutes declination to +- 0.01 degrees (The Astronomic Almanac). The accuracy of azimuth/zenith obtainable from these is discussed in a technical bulletin by the WA Department of Agriculture.

References :

For the algorithm to compute Julian Dates;

Montenbruck,0. (1989). Practical Ephemeris Calculations Springer-Verlag, Berlin, 1989.

For the ephemeris formula;

Anon., The Astronomical Almanac , Nautical Almanac Office United States Naval Observatory and Her Majesty's Nautical Almanac Office Royal Greenwich Observatory, US Government Printing Office and Her Majesty's Stationery Office, (for the year 1984).

For data to cheek the ephemeris data;

Anon., The Star Almanac For Land Surveyors, Her Majesty's Nautical Almanac Office, Her Majesty's Stationery Office, (for the year 1973).

General Astronomy Principles

Glassock, J.T.C. (1983). Lecture Notes - Land Surveying VI -Astronomy. Queensland Institute of Technology, Brisbane, 1983.

Davis, R.E., Foote, F.E., Anderson, J.M. and Mikhail, E.M. (1981). Surveying, Theory and Practice. McGraw-Hill, New York, 1981.

or any other numerical astronomy or surveying textbook.

Revisions :

Version	Date	By	Details
1.0	28-1-1992	mlr	The original subroutine

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
/* Constants rounded for 21 decimals. */
#define MY_PI               3.14159265358979323846

/* prototype definition  -  i.e. ANSI C declaration */
void sunephem(     int year, int month, int day, double lstdt_dec, double time_zone_dec, double stn_lat_dec,
                   double stn_long_dec, int printdataswitch, double *decl, double *equation_time, double
                   *hour_angle, double *sun_dist_au, double *sun_semi_diam );


/* the subroutine */

void sunephem
(
    int         year,               /*  <= input year (e.g. 1986)                                   */
    int         month,              /*  <=  input month (e.g. 3)                                    */
    int         day,                /*  <= input day (e.g. 12)                                      */
    double      lstdt_dec,          /*  <= input local std time in hrs (e.g. 14.2341)
                                        (decimal hrs)                                               */
double          time_zone_dec,      /*  <= input time zone (e.g. Perth WST = +8.00)
    double      tn_Lat_dec,         /*  <= input latitude of stn (e.g. -27.127)
                                        South is -ve, North is +ve. (decimal deg)                   */
    double      tn_long_dec,        /*  <= input longitude of stn (e.g. 120.341)
                                        East is +ve, West is -ve (decimal deg)
    int         printdataswitch,    /*  <= input causes the ephemeris data to be printed.
                                        = 1 for printing , =0 for no printing.          *1
    double      *decl,              /*  => returns a pointer to decl (decimal deg)                  */
    double      *equation_time,     /*  => returns a pointer to equation_time (decimal deg)             */
    double      *hour_angle,        /*  => returns a pointer to hour_angle (decimal deg)                */
    double      *sun_dist_au,       /*  => returns a pointer to earth sun distance in
                                        AU (astronomical units)
    double      *sun_semi_diam      /*  => returns a pointer to sun semi-diameter
                                        in decimal degrees

)

{
    double      ut_dec, julian_day_dec;
    double      deg2rad, rad2deg;
    double      n,L,g;
    double      lambda,e,ra;

    /* some useful constants */
    deg2rad = MY_PI / 180;
    rad2deg = 180 / MY_PI ;

    /*   check the year  -  if after 2050, then the almanac data will need
         to be updated from the Astronomical Almanac */
    if   ((year < 1950) || (year > 2050))
    {
    printf("\n\n*** ERROR ***\n");
    printf("This program is only good between 1950 and 2O5O\n\n");
    printf("You need to update the almanac for years outside\n");
    printf("that range\n");
    printf("... Aborting …\n\n");
    exit (i);
    }
```

```
/*  calculate the julian day - use algorithm given by
    Montenbruck p33-34.  Only consider dates on or after 15-10-1582 */
    {
        long int litem1, litem2, litem3, jdyear, jdmonth;
        double diteml, ditem2;

    /*  calculate the universal time */
    ut_dec = lstdt_dec - tirne_zone_dec;

    /*  check that date is after 15-10-1582  -  I realise the test for between 1950 -> 2050 above will also get this,
        but for completeness sake it is included */
    if  (year <= 1582)
        {
          if(month <= 10)
          {
            if((day+(ut_dec/24)) < 15)
            {
              printf("*** Error ***   can't correctly calculate dates before\n");
              printf("                        15-10-1582\n");
              printf("...  aborting …\n\n\n");
              exi t( 1);
            }
        }
    }
jdyear = year;
jdonth = month;
if (month <= 2)
    {
      jdyear = year - 1;
      jdmonth = month + 12;
    }

ditem1    =  (365.25 * jdyear);
litemi    =  (long int) ditem1;
ditem2    =  30.6001 * (jc~month + 1);
litem2    =  (long int) ditem2;
litem1    =  ((long int) (jdyear/400)) - ((long int) (jdyear/100));

julian_day_dec = litemi + litem2 + litem3 + 1720996.5  +  day  +  (ut_dec/24);
}

/*
    Calculate ephemeris data using formula given on page C24 of "The AstronomicaL Almanac ". These formula
    have a quoted accuracy of 0.01 degrees for declination and 0.1 minutes (time) for the equation of time. For a
    full discussion on uncertainty, see the sunerror.c module. They are only to be used for the period 1950-2050
    inclusive.

    n  =  Current JuL ian Day - Jul ian Day on J2000.0
    L  =  mean Longitude of the sun corrected for aberration
    g  =  mean anomaLy of the sun
    lambda   =   ecliptic Longitude
    e  =  obliquity of the ecliptic
    ra =  right ascension
    decl   =   declination
    equation_time   =   true sun - mean sun
                        =   L - ra

    To update the formula after 2050, you just update the formulas for n, L, g, Lantda, e, set out below. The rest
    should still be the same (unLess there is some cataclysm - then you won't care anyway).
```

```
*/


{

    long int num_revs_360;

    n   =   julian_day_dec - 2451545.0;
    L   =   280.460 + (0.9856474 * n);
    G   =   357.528 + (0.9856003 *
    E   =   23.439 - (0.0000004*n);


    /1* put L and g in range 0->360 degrees */

    /* L first */
    num_reys_360 = (long int) (L/360);
    if (L<0)
        {
          num_revs_360  =  num_revs_360  -  1;
        }
    L   =   L  -  (num_revs_360 * 360);

    /* Now g */
    num_revs_360  =  (long int) (g/360);
    if (g<0)
        {
      num_revs_360  =  num_revs_360  -  1;
        }
    g   =   g   -   (num_revs_360 * 360);


    lambda  =  L  +  (1.915 * sin(g*deg2rad))  +  (0.020 * sin(2*g*deg2rad));
    /* put lambda in interval O->36O degrees */
    num_revs_360  =  (long int) (lambda/360);
    if (lambda<0)
        {
          num_revs_360 = num_revs_360  -  1;
        }
    lambda  =  lambda - (nun_revs_360 * 360);
    ra      =   rad2deg * (atan(cos(e*deg2rad)*tan(lambda*deg2rad));
    *decl   =   rad2deg * (asin(smn(e*deg2rad)*smn(lambda*deg2rad));

    /* put ra in same quadrant as lambda */
    /* first put ra in 0->360 degrees */
    num_revs_360  =  (long int) (ra/360);
    if (rac<0)
        {
          num_revs_36O  =  nun_revs_360  -  1;
        }
    ra  =  ra  -  (num_revs_360 * 360);

    /* Now put ra in same quadrant as lambda */
    {
      long int quadlambda, quadra;
```

```
    quadlambda =  (long int) lantda/90;
    quadra                          =  (long int) ra/90;

    ra  =  ra + (quadlambda-quadra) * 90;
    }

    *equation time = L - ra;
    /* put equation_time in 0->360 degrees */
    num_revs_36O = (long int) (*equation_time/360);
    if (*equation_time<0)
        {
          num_revs_360 = num_revs_360  -  1;
        }
    *equation time = *equation_time - (num_revs_360 * 360);

    *hour angle         =   (15 * ut_dec)  +  stn_long_dec  +  *equation_time  -  180;
    *sun_dist_au        =   1.00014  -  (0.01671*cos(g*deg2rad))  -  (0.00014*cos(2*g*deg2rad));
    *sun_semi_diem  =   0.267/(*sun_dist_au);

    /* put *hour_angle in interval 0->360 degrees */
    num_revs_360  =  (long int) (*hour_angle/360);
    if (*hour_angle<0)
        {
          num_revs_360  =  num_revs_360  -  1;
        }
    *hour_angle  =  *hour_angle  -  (num_revs_360 * 360);

}


    /* Print out summary */
    if (printdataswitch)
    {
    printf("\n\nSumnary of Calculations\n");
    printf("-------------------------------------------------------\n");
    printf("Input Data  :\n");
    printf("                    \n");
    printf("Date                        =   %2d-%2d-%4d\n", day, month, year);
    printf("Local Std Time          =   %lf\n", lstdt_dec);
    printf("Time Zone               =   %lf\n", time_zone_dec);
    printf('Latitude                    =   %lf\n", stn_lat_dec);
    printf("Longitude               =   %lf\n", stn_long_dec);
    printf("Calculated Data             :\n");
    printf("-----------------------------------\n");
    printf("JD                          =   %lf\n",julian_day_dec);
    printf("n                           =   %lf\n",n);
    printf("    L                       =   %lf\n",L);
    printf("g                           =   %lf\n",g);
    printf("Lambda                      =   %lf\n",Lambda);
    printf("e                           =   %lf\n",e);
    printf("ra                          =   %lf\n",ra);
    printf("decl                        =   %lf\n",*decl);
    printf("equation_time           =   %lf\n",*equation_time);
    printf("hour_angle              =   %lf\n",*hour_angle);
    printf("Sun_dist_au             =   %lf\n",*sun_dist_au);
    printf("Sun_semi_diam           =   %lf\n",*sun_semi_diam);
    }
}
```

/* Program name : sunerror.c

Purpose : To compute a confidence interval for the predicted solar azimuth and zenith angles, given uncertainty in the input data. The confidence interval computed, is that of the input uncertainty data.

The methodology has been developed by the author using calculus and propogation of variance techniques.

Author : M.L. Roderick
Western Australian Department of Agriculture
GIS Group.

Date : 15-02-92

Language : ANSI Standard C

Compilation :

Limitations : None known

References :

General astronomy principles

Glassock, J.T.C. (1983). Lecture Notes - Land Surveying VI - Astronomy. Queensland Institute of Technology, Brisbane, 1983.

Davis, R.E., Foote, F.E., Anderson, J.M. and Mikhail, E.M. (1981). Surveying, Theory and Practice. McGraw-Hill, New York, 1981.

or any other numericaL astronomy or surveying textbook.

General theory

The solar azimth is given by:

$$\tan(\text{Azimuth}) = \frac{-\sin(\text{hour\_angle})}{\tan(\text{decl})*\cos(\text{lat}) - \sin(\text{lat})*\cos(\text{hour\_angle})}$$

where hour_angle = time since sun was on the observer's meridian
= UT + Long + equation_time - 180 (decimal degrees)
long = observer's longitude
equation_time = true sun - mean sun
decl = declination of the sun
lat = latitude of station

The above formula is a general solution for Azimuth of the Astronomical Triangle.

Zenith distance

Note : Zenith Distance = 90 – elevation

cos(ZD) = sin(lat)*sin(decl) + cos(lat)*cos(decl)*cos(hour_angle)

The routine sunzdobs (part of this package) can be used to correct the zenith distance calculated in this routine, to derive an actual observed zenith distance.

Dealing with uncertainty

An important part of the routines is an estimate of the precision of the solar azimuth and zenith angles predicted using estimates of the uncertainty in the input data (including the ephemaris data). This is

done, using propogation of variance techniques. The expressions for solar azimuth and zenith are given by:

$$\tan(A) = \frac{-\sin(\text{hour\_angle})}{\tan(\text{decl})\cos(\text{lat}) - \sin(\text{lat})\cos(\text{hour\_angle})}$$

$$\cos(ZD) = \sin(\text{lat})\sin(\text{decl}) + \cos(\text{lat})\cos(\text{decl})\cos(\text{hour\_angle})$$

Considering Azimuth

Differentiating we get;

$$d(A) = \frac{\sin(A)}{\tan(ZD)}\cos(W)\,d(\text{lat}) + \frac{\sin(A)\cos(W)}{\sin(\text{hour\_angle})}\,d(\text{hour\_angle}) + \frac{\sin(\text{hour\_angle})\cos(\text{lat})}{\sin(ZD)\sin(ZD)}\,d(\text{decl})$$

where w = parallactic angle , and may be computed by:

$$\cos(W) = \frac{\sin(\text{lat}) - \sin(\text{decl})\cos(ZD)}{\cos(\text{decl})\sin(ZD)}$$

The hour_angle is given by:

hour_angle = UT + Long + equation_time - 180 (decimal degrees)

The uncertainty in hour_angle is a function of uncertainty in UT, Long and equation_time. Assuming the errors are random and not correlated, then an expression may be derived for d(hour_angle) as follows:

d(hour_angle) = sqrt [d(UT)^2 + d(Long)^2 + d(equation_tirme)^2]

The above expression for dA may be used in propogation of variance. By assuming that all errors are random and not correlated (which gives a diagonal variance-covariance matrix), a general expression for the uncertainty in the predicted Azimuth is:

$$d(A) = \text{sqrt}\left[\left(\frac{d(A)}{d(\text{decl})}\right)^2 d(\text{decl})^2 + \left(\frac{d(A)}{d(\text{hour\_angle})}\right)^2 d(\text{hour\_angle})^2 + \left(\frac{d(A)}{d(\text{lat})}\right)^2 d(\text{lat})^2\right]$$

where $\frac{d(A)}{d(\text{lat})} = \frac{\sin(\text{hour\_angle})}{\cos(\text{lat})}$

$$\frac{d(decl)}{} = \frac{\sin(ZD)\ \sin(ZD)}{}$$

$$\frac{d(A)}{d(hour\_angle)} = \frac{\sin(A)\ \cos(w)}{\sin(hour\_angle)}$$

$$\frac{d(A)}{d(lat)} = \frac{-\sin(A)}{\tan(ZD)}$$

Considering Zenith

Similar reasoning can be used to show that

$$d(ZD) = -\cos(A)\ d(lat) + \sin(A)\ \cos(lat)\ d(hour\_angle) + -\cos(w)\ d(decl)$$

$$d(ZD) = sqrt\left[\left(\frac{d(ZD)}{d(decl)}\right)^2 * d(decl)^2 + \left(\frac{d(ZD)}{d(hour\_angle)}\right)^2 * d(hour\_angle)^2 + \left(\frac{d(ZD)}{d(lat)}\right)^2 * d(lat)^2 \right]$$

where $\frac{d(ZD)}{d(decl)} = -\cos(w)$

and $\frac{d(ZD)}{d(hour\_angle)} = \cos(lat)\ \sin(A)$

$$\frac{d(ZD)}{d(lat)} = -\cos(A)$$

and the parallatic angle w is given above.

Implementation

The routine may be used to solve any uncertainty problem for predicting solar azimuth and zenith given estimates of the precision of the input data. For this specific case, (i.e. use of the sunephem routine) the logic is developed below. The routine is kept as general as possible, so the programmer will be required to set the variables as per the following discussion.

The routines will propogate uncertainty in random variables. If the input uncertainty related to a 95 per cent confidence interval, then the output would also. Thus the programmer will have to control this in applications.

Note: The input uncertainty are in degrees of arc. Thus for time (i.e. equation time and UT, multiply the uncertainty by 15 to get decimal degrees of arc).

For example:

The Astronomical Almanac quotes on page C24, “The following formulae give the apparent coordinates of the Sun to a precision of 0.01 degrees and the equation of time to a precision of 0.1 minutes between 1950 and 2050.”

d(equation_time) = 0.1 minutes (time)
= 0.025 degrees (arc)
d(decl) = 0.01 degrees (arc)

The confidence interval is not stated in The Astronomical Almanac. I have assumed that it is 1 standard deviation or 68 per cent.

All we need do is then estimate a 68 per cent confidence interval for the rest of the input data and then propogate the variance, to get a 68 per cent confidence interval for azimuth and zenith.

If we were to assume no errors in lat, long or UT then they would be zero. Thus we would only consider the effect of ephemeris errors on the result.

Alternatively, in an Agricultural experiment to measure solar radiation at a given location, we would have to consider uncertainty in both position and time. In this application, time would be recorded, and Later related to the solar position by calculating the solar position at that time. Typical values would be;

For 68 X confidence intervaLs we could estimate the following:

d(lat) = 0.00056 (i.e. 2 secs of lat = about 60 metres on the ground)
d(long) = 0.00056
d(UT) = 0.042 (i.e. 10 secs of time = 0.042 deg of arc)
(could easily get to 3 seconds by setting your watch by Telecom clock or the VNG short wave radio station.)

Obviously the input data will depend on the application.

It is acceptable to assume a normal distribution for these purposes, and the standard confidence intervals of the normal distribution are:

68.26 % +- 1 SD
90 % +- 1.645 SD
95 % +- 1.96 SD
95.45 % +- 2 SD
99 % +- 2.576 SD
99.73 % +- 3 SD
etc.

where SD = standard deviation.

Revisions :

Version	Date	By	Details
1.0	15-02-92	mlr	The original subroutine

```
*/

#include <stdio.h>
#incLude cstdlib.h>
#include cmath.h>


/* Constants rounded for 21 decimals. */
#define MY_PI      3.14159265358979323846

/* prototype definition  -  i.e. ANSI C declaration */
void sunerror (      double hour_angle, double decl, double stn_lat_dec, double azimuth, double zenith_calc,
                double d_lat, double d_long, double d_UT, double d_equation_time, double d_decl, mt
                printdataswitch, double *az_ci, double *zd_ci );

/* the subroutine */

void sunerror
(
    double    hour_angle,     /*  <= input hour_angle (decimal degrees)
                                  +ve west, -ve east                                  */
    double    decl,           /*  <= input the declination to sun (decimal degrees)
                                  +ve North, -ve South                                */
    double    stn_lat_dec,    /*  <=   input latitude of stn (eg. -27.127)
                                  South is -ve, North is +ve. (decimal degrees)       */
    double    azimuth,        /*  <    input azimuth (decimal degrees)                */
    double    zenith,         /*  <=   input zenith (decimal degrees)                 */
    double    d_lat,          /*  <=   input estimated error in latitude (decimal degrees)  */
    double    d_Long,         /*  <=   input estimated error in longitude (decimal degrees) */
    double    d_UT,           /*  <=   input estimated error in Universal Time
                                  (in decimal degrees of arc, where arc = time*15     */
    double    d_equation_time, /*    <                                                input estimated error
                                  in equation_time
                                  (in decimal degrees of arc, where arc = time*15     */
    double    d_decl,         /*  <=   input estimated error in declination
                                  (in decimal degrees of arc, .where arc = time*15    */
    int       printdataswitch, /*  <=   input causes the ephemeris data to be printed.
                                  =1 for printing , =0 for no printing.               */
    double    *az_ci,         /*  =>   returns a pointer to azimuth confidence interval
                                  (in decimal degrees of arc)                         */
    double    *zdci           /*  =>   returns a pointer to zenith confidence interval
                                  (in decimal degrees of arc)                         */

)


{
    double    deg2rad, rad2deg, w;
    double    dhour_angle, dA_dlat, dA_dhour_angle, dA_ddecl;
    double    dZ_dlat, dZ_dhour_angle, dZ_ddecl;


    /* some usefuL constants */
    deg2rad  =  MY_PI  /  180;
    rad2deg  =  180  /  MY_PI ;
```

```
/* convert required quantities to radians */
hour_angle      = hour_angle * deg2rad;
decl            = decl * deg2rad;
stn_lat_dec     = stn_lat_dec * deg2rad;
azimuth         = azimuth * deg2rad;
zenith          = zenith * deg2rad;

/* calculate parralactic angle w */
    {
    double tempnumerator, tenpdenominator;

    tempnumerator  =  sin(stn_lat_dec)  -  (sin(decl)*cos(zenith));
    tempdenominator  =  cos(decl)*sin(zenmth);
    w  =  acos(tempnumerator/tempdenominator);
    }

/*  calculate uncertainty in hour_angle */
    dhour_angle = sqrt(   (d_UT*d_UT) + (d_long*d_long)
                        +  (d_equation_ti,ne*d_equation_time) );

/*  calculate coefficients of the differential equations */
    dA_dlat  =  -1 * sin(azimuth)  /  tan(zenith);
    dA_dhour_angle  =  sin(azimuth)*cos(w)/sin(hour_angle);
    dA_ddecl  =  (sin(hour_angle)*cos(stn_Lat_dec))/(sin(zeni th)*smn(zenith));

    dZ_dlat  =  -1*cos(azimuth);
    dZ_dhour_angle  =  cos(stn_lat_dec)*sin(azimuth);
    dZ_ddecl  =  -1*cos(w);

/* calculate confidence intervals for azimuth and zenith */


    {
        double   tempdbl1,tempdbl2,tençdbl3;

        /* azimuth first */
        tempdbll    =   dA_dlat*dA_dlat*d_lat*d_lat;
        tempdbL2  =   dA_dhour_angle*dA_dhour_angle*dhour_angle*dhour_angle;
        tempdbl3   =   dA_ddecl*dA_ddecl*d_decl*d_decl;

        *azci        =   sqrt(tempdbl1+tempdbl2+tempdbl3);

        /* now zenith */
        tempdbl1   =   dZ_dlat*dZ_dlat*d_lat*d_lat;
        tempdbl2   =   dZ_dhour_angle*dZ_dhour_angle*dhour_angle*dhour_angle;
        tempdbll3  =   dZ_ddecl*dZ_ddecl*d_decl*d_decl;

        *zd_ci       =   sqrt(tempdbl1+tempdbl2+tempdbl3);
    }

/* convert original data back to degrees for printing out */
hour_angle  =   hour_angle * rad2deg;
decl            =   dccl * rad2deg;
stn_lat_dec  =   stn_lat_dec * rad2deg;
azimuth       =   azimuth * rad2deg;
zenith         =   zenith * rad2deg;
```

```
    /* convert parallactic angle w to degrees */
      w  =  w  *  rad2deg;


    /* Print out summary */
    if (printdataswitch)
    {
    printf("\n\nSumnary of Calculations\n");
    printf("----------------------------------\n");
    printf("Input Data  :\n");
    printf("---------------\n");
    printf("Hour Angle    =     %lf\n",      hour_angle);
    printf("DecLination   =     %lf\n",     dccl);
    printf("Latitude      =     %lf\n",     stn_lat_dec);
    printf("Azimuth       =     %lf\n",     azimuth);
    printf("Zenith        =     %lf\n",     zenith);
    printf("Uncertainty Data\n");
    printf("d_Latitude          =     %lf\n",     d_lat);
    printf("d_Longitude         =     %lf\n",     d_Long);
    printf("d_UT                =     %lf\n",     d_UT);
    printf("d_Equation_Time     =     %lf\n",     d_equation_time);
    printf("d_Declination       =     %lf\n",     d_decl);
    printf("Calculated Data  :\n");
    printf("-----------------------\n);
    printf("ParalLactic Angle  =     %lf\n",   w);
    printf("Confidence Intervals.\n");
    printf("Az confidence       =     %lf\n",   *az_ci);
    printf("ZD confidence       =     %lf\n",   *zd_ci);
    }

}



^Z
```

/* Program name : sunprezd.c

Purpose : To calculate the Local standard time that a particular solar zenith distance occurs. The user is able to set either the morning or afternoon time. This is useful for computing sun rise/set times and also length of daylight for agricultural applications.

Author : M.L. Roderick
Western Australian Department of Agriculture
GIS Group.

Date : 29-1-92

Language : ANSI Standard C

Compilation :

Limitations : a) The program calls the sunazzd and sunephem routines, which are part of this package. Limitations for these routines will apply.

References :

See notes in sunazzd.c and sunephem.c.

General astronomy principles

Glassock, J.T.C. (1983). Lecture Notes - Land Surveying VI -Astronomy. Queensland Institute of Technology, Brisbane, 1983.

Davis, R.E., Foote, F.E., Anderson, J.M. and Mikhail, E.M. (1981). Surveying, Theory and Practice. McGraw-HiLL, New York, 1981.

or any other numerical astronomy or surveying textbook.

General theory

This routine finds the time that a particular zenith distance occurrs. The particular zenith distance occurrs twice on each day (e.g. sun rise/set) and the user must nominate which solution is sought (using the sun_pos variable).

The formula to find zenith distance is given by:

cos(ZD) = sin(Lat)*sin(decl) + cos(lat)*cos(decl)*cos(hour_angLe)

Note: Zenith Distance = 90 - elevation
and lat = observer's latitude
dccl = sun's declination
hour_angle = hour angle of the sun (west of the meridian is +ve).
= UT + Long + ET - 180

where UT = universal time
= Local Standard Time - Time Zone
Long = observer's longitude
ET = equation of time
= true sun - mean sun

Obviously, we need to solve the above equation for hour_angle, and hence derive local standard time. However, both ET and declination are functions of time, so a direct solution is not possible.

Various mathematical methods (e.g. differentiation) can be used to solve for hour_angle with minimal computing effort. An easier method is to use 'grunt' and iterate using successively better approximations until a pre set tolerance is reached.

This is shown in the steps below:

1. Using initial estimate of local std time (i.e. 6am or 6pm) use the sunephem and sunazzd routines to derive a zenith distance (say ZD[1]) for that time.

2. ZD[1] will not equal the required zenith distance, and the difference can be used to make an estimate of the correct Local std time by the following:

   correction = (requ_zenith - ZD[1])*(sun_pos)/15

   where 15 is the (very) approximate number of degrees the sun moves in 1 hour. This gives a new local std time of:

   new_lstdt_est = correction + lstdt_dec

3. If the correction is Less than some tolerance then we have reached the result. Otherwise repeat step 1 using new_lstdt_est from step 2 above until the correction is smaller than the tolerance set.

The scheme could be improved using calculus, by partial differentiating the equation for cos(ZD) [see equation i above], and using this as a basis for updating each estimate. However, the solution described above and implemented is still correct, but it may require many more iterations to reach a solution. This is not considered a problem, as the processing is memory based.

The required solar zenith angle input will depend on the application. Some coninon examples include:

- Sun rise/set is considered to occurr when the true centre of the sun is at zenith 90.833 degrees (i.e. 90 deg 50'). The 50' is made up of refraction (34') and the sun's semi-diameter of 16'. The routine sunephem returns a value for the Sun's semi-diameter. This could be used instead of the approxiate value of 16' above. The difference between the values will not be significant for most applications. Thus at a predicted solar zenith of 90 deg 50', the top of the sun will appear on the horizon of a flat landscape. It is noted that topography can make a significant difference (i.e. shorter day length, as can abnormal atmospheric conditions (i.e. affect refraction). The effect of topography will vary, and the day length can be either shorter or longer depending on the particular circumstances. To model this effect would require a DTM (digital terrain model), with viewshed analysis capabilities, and the ability to calculate solar zenith (corrected for parallax/ refraction). This program could be used, although it may be slow if it can't be linked into the DTM package.

- Civil twilight is the interval between sun rise/set and when the true position of the sun is 6 degrees below the horizon (i.e. zenith = 96 degreees).

- For various agricultural applications, it may be desireable to use experimental data to define the zenith angle when a certain amount of solar radiation (i.e. energy) is available.

  For example, see:
  KeisLing,T.C. (1982). Calculation of the Length of the Day. Agronomy Journal 74, pp758-759.

  The techniques noted in this routine could be used to replace

  Keisling's calculations for day length. For example, if a certain solar radiation (important to agricultural applications) occurred at say zenith < 93 deg. Then the local standard time for a solar zenith of 93 degrees could be calculated for morning/afternoon and day length is given by local std time (afternoon - morning).

  It is important to note that all applications above consider the 'true position' of the sun. This is equivalent to the zenith_calc variable returned from the sunazzd.c routines.

Earth-Sun distance

The routine returns the mean distance to the sun (mean_earth_sun_au) in AU (astronomical units). 1 AU = i49 597 870 000 metres. This has been included for potential agricultural applications. It may be feasible in agricultural applications to use the earth sun distance to get an idea of variation in radiation by the sun. The earth sun

distance is computed in routine sunephem, and in this routine, the mean earth-sun distance is taken.

Revisions :

Version	Date	By	Details
1.0	29-1-1992	mlr	The original subroutine
1.1	15-2-1992	mlr	Update for changes in sunazzd and sunerror routines.

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include “sunazzd.c”
#include “sunephem.c”


/* Constants rounded for 21 decimals. */
#define MY_PI       3.14159265358979323846


/* prototype definition - ie. ANSI C declaration */
void sunprezd (int year, int month, int day, double time_zone_dec, double stn_lat_dec, double
                stn_long_dec,
                int printdataswitch, int sun_pos, double requ_zenith, double *lstdtdec, double
                *sundistau);


/* the subroutine */

void sunprezd
(
    int         year,               /* <=     input year (e.g. 1986)                                  */
    int         month,              /* <=     input month (e.g. 3)                                    */
    int         day,                /* <=     input day (e.g. 12)                                     */
    double      time_zone_dec,      /* <=     input time zone (e.g. Perth UST  =  +8.00)              */
    double      stn_lat_dec,        /* <=     input latitude of stn (e.g. -27.127)
                                               South is -ve, North is +ve.     (decimal deg)           */
    double      stn_long_dec,       /* <      input longitude of stn (e.g. 120.341)
                                               East is +ve, West is –ve      (decimal deg)           */
    int         printdataswitch,    /* <=     input causes the output data to be printed.
                                               = 1 for printing , =0 for no printing.                 */
    int         sun_pos,            /* <=     input sun position.
                                               Sun rise = 1; Sunset = 1                               */
    double      requ_zenith,        /* <=     input zenith angle for which the local
                                               standard time is required (decimal degrees)            */
    double      *lstdtdec,          /* =>     returns a pointer to local std time in hrs
                                               (e.g. 5.781, 17.812 in decimal hrs) that the
                                               requ_zenith occurrs.                                   */
    double      *sundistau          /* >      returns a pointer to distance to the sun
                                               for the day in question.
                                               (in AU where 1 AU = 149 597 870 000 metres.            */
)


{
    double      decl, equation_time, hour_angle, sun_semi_diem
```

```
    double      azimuth, zenith_calc;
    double      correction;
    double      new_kstdt_est;
    double      tolerance_cony;
    Long imt    icntr;

    /* check that sun_pos = either -1 or 1 */
    if (sun_pos != -1)
        {
          if (sun_pos != 1)
             {
               printf("*** Error ***  \n");
               printf("Variable sun_pos not specified correctly in call\n");
               printf("to sunprezd routine.\n");
               printf("... aborting ... \n");
               exit (1);
             }
        }



    /*  this gives a tolerance for the iteration */
    /*  the units are decimal degrees - so 0.00001 decimal degrees
        =  0.54 seconds of time.  Change if you want to.  As the tolerance gets smaller, the number of
           iterations required to complete the job gets larger. */

    tolerance_conv 0.00001;
    icntr = 0;


    /*  initial estimae of local standard time - if the user set sun_pos = -1 (i.e. sun rise) then the initial
        approximation is 6 hrs (local standard time).  Otherwise the user set sun_pos = 1 (i.e. sun set)
        so an initial guess is made of 18 hrs local time */

    *lstdt dec = 12  +  (sun_pos * 6);


    /*  set the temporary variable to be used in the loop below */
    new_lstdt_est = *lstdtdec;


    /*  the iteration loop  -  hold in loop until the corrections are less than the variable tolerance_conv
*/
    do
    {
      *lstdtdec = new_lstdt_est;

    /*  use local standard time estimate to get ephemeris data */
    /*  Note:  this routine returns a pointer to sun_dist_au.  Since sun_dist_au is already a pointer in
        the sunprezd declaration, it is not prefixed by &. */

    sunephem(  year, month, day, *lstdtdec, time_zone_dec, stn_lat_dec, stn_long_dec, printdataswi
                    tch, &decl, &equation_time, &hour_angle, sun_dist_au, &sun_semi_diam );

    /*  use ephemeris data to calculate zenith distance for that estimated local standard time */

    sunazzd (     hour_angle, decl, stn_lat_dec, printdataswitch, &azimuth, &zenith_calc );

    /*  the sun moves (very) approximately 180 deg in 12 hours, which is 15 deg per hour.  Thus we
        update the local standard time, by dividing the difference in zenith (i.e. required - estimated) by
        15.  Typcally, about 8 to 9 iterations will be required to get within a tolerance_cony = 0.00001.
        This is not a major problem as it is a memory process and not disk based.  The direction for the
```

```
        correction is per the sun_pos variable (i.e. sun rise/set) */


    correction = (requ_zenith - zenith_calc)*(sun_pos)/15;

    /* update the Local standard time with a new estimate */
    new_lstdt_est = correction + *Lstdtdec;


    /*   this is a work around to get the abs() value.  In many c compilers abs() takes an int only.  I have
         generally found the abs() function to be unsatisfactory in different C compilers. */
    if (correction c 0)
    {
      correction = .1*correction;
    }

    icntr++;

    }    while (correction > tolerance_conv );




    /* set local standard time to our Last estimate */
    *lstdt_dec  =  new_lstdt_est;


    /* Print out summary */
    if (printdataswitch)
    {
    printf("\n\nSummnary of Calculations\n");
    printf("---------------------------------\n");
    printf("Input Data :\n");
    printf("---------- \n");
    printf("Date                  =   %2d-%2d-%4d\n", day, month, year);
    printf("Time Zone             =   %lf\n", time_zone_dec);
    printf("Latitude              =   %lf\n", stn_lat_dec);
    printf("Longitude             =   %lf\n", stn_long_dec);
    printf("Sun pos               =   %d\n", sun_pos);
    printf("Requi red Zeni th   =   %l f\n", requ_zeni th);
    printf("Calculated Data    :\n");
    printf("----------------------\n");
    printf("Earth_Sun_dist (AU)  =   %Lf\n", *sun_dist_au);
    printf("Local Std time          =   %lf\n", *lstdt_dec);
    printf("This time is for ");
    if (sun_pos == -1)
        }
          printf("morning. \n\n");
        {
    else
        {
          printf("afternoon. \n\n");
        }
    }



}
```

```
/*      Program name    : suntools.c
        Purpose     :   Some general tools for use in writing programs using the other routines in this
                        package.
        Author          : M.L. Roderick
                          Western Australian Department of Agriculture
                          GIS Group.
        Language    :   ANSI Standard C
        Compilation :
        Limitations :
        References  :
        For the algorithm to compute grid convergence ;
        Any technical manual on the AMG (Australian Map Grid).


*/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Constants rounded for 21 decimals. */
#define MY_PI      3.14159265358979323846

/* prototype definitions  -  i.e. ANSI C declaration */

void calc_grid_conv(double stn_lat_dec, double stn_long_dec, double cen_mer_long, int
                        printdataswitch, double *grid_conv);

void wada_gis_screen (void);

void my_slow_clear_screen(void);


/*-------------------------------------------------------------------------------------------------------------------------*1
/*      Routine  :  grid_conv()
        Purpose  :  to compute the grid convergence at a particular point  given, Lat, Long,
                    Long_Cen_Meridian.
        Date     :  06-02-1992

        Revisions   :
```

Version	Date	By	Details
1.0	06-2-1992	mlr	The original subroutine

Discussion

The sunazzd routine returns the azimuth to the sun at a particular time (ie. hour_angle). This azimuth is called astronomic azimuth, and is the clockwise angLe measured from the observer's meridian. It is sometimes called TRUE NORTH. GRID NORTH is the clockwise angle measured from a line parallel to the central meridian of the AMG zone. It is more useful in a GIS context, when the other geographic data are referenced to the AMG projection.

The GRID CONVERGENCE is the difference between true north and grid north at the observers position.

GRID_CONV is given by:
tan (grid_conv) = - sin (Lat) * tan (long - cen_mer_long)

where lat = observers latitude (south -ve)
long = observers longitude
cen_mer_long = longitude of the central meridian

and GRID_BEARING = TRUE_BEARING + GRID_CONV

Each zone in the AMG is 6 deg (of longitude) wide, and for zone 50 (which covers the SW of WA) the central meridian = 117 deg. So cen_mer_long (Zone 5i) = 117 + 6 deg
= 123 deg
and so on for zones 52,53, ……

Whether you require GRID or TRUE bearings will depend on the application. For modelling sun, ground, satellite geometry, the TRUE bearing will generally be more useful, as trus bearings are used in ephemeris predictions. For day length studies, using a DTM referenced to the AMG, the GRID bearing will be essential, to relate the sun's azimuth to the AMG.

*/

```
void calc_grid_conv
(
double    stn_lat_dec,      /* <=  input latitude of stn (e.g. -27.127)
                                    South is -ve, North is +ve. (decimal deg)        */
double    stn_long_dec,     /* <=  input longitude of stn (e.g. 120.341)
                                    East is +ve, West is -ve (decimal deg)           */
double    cen_mer_long,     /* <=  the longitude of the central meridian  (decimal deg) */
int       printdataswitch,  /* <=  input causes the data to be printed.
                                    = 1 for printing , =0 for no printing.           */
double    *grid_conv        /* =>  returns a pointer to the grid convergence (decimal deg)  */
)

{
    double    lat_rads, long_rads, cen_mer_tong_rads;
    double    deg2rad, rad2deg;


/* some usefuL constants */
deg2rad  =  MY_PI / 180;
rad2deg  =  180 I MY_PI ;

/* convert quantities to radians */
lat_rads    =  deg2rad * stn_lat_dec;
long_rads  =  deg2rad * stn_long_dec;
cen_mer_long_rads   =      deg2rad * cen_mer_long;

*grid conv   =  atan(-i * sin(lat_rads) * tan(long_rads - cen_mer_long_rads));
*grid conv   =  *grid cony * rad2deg;

/* Print out summary */
if (printdataswitch)
{
printf("\n\nSummary of Calculations\n");
printf("---------------------------------\n");
printf("Input Data  :\n");
printf("---------------\n");
printf("Latitude      =  %lf\n",stn_lat_dec);
printf("Longitude    =  %lf\n",stn_long_dec);
printf("Longitude Cen Meridian    = %lf\n", cen_mer_long);
printf("Calculated Data :\n");
```

```
    printf("----------------------\n");
    printf("Grid Convergence  =  %lf\n",*grid_conv);
    {
    }
    /*  ------------------------------------------------------------------------------------------------------ */



    /*  ------------------------------------------------------------------------------------------------------ */
    /*  Routine  :   wada_gis_screen()
        Purpose  :   prints up a screen banner

    Discussion  :
    No maths this time !

    */

    void wada_gis_screen (void)
    {
    printf("                                                                                                n);
    printf("                          Software developed by                                                 \n);
    printf("                                                                                                \n);
    printf("                  Western Australian Department of Agriculture                                  \n);
    printf("                      Division of Resource Management                                           \n);
    printf("                              GIS Group  -  © 1992                                              \n);
    printf("                              Author:  M. Roderick                                              \n);
    printf("                                                                                                \n);
    printf("                  As part of a general package for use in                                       \n);
    printf("                      predicting solar position.                                                \n);
    printf("                                                                                                \n);

    }
    /*  ------------------------------------------------------------------------------------------------------ */


    /*  ------------------------------------------------------------------------------------------------------ */
    /*   Routine  :   my_slow_clear_screen()
        Purpose  :    clears the screen

    Discussion :
    While PC compilers include clear screen functions, they are not portable in the Unix environment
    and vice-versa.  This method is very slow, but does the trick.

    */

    void my_slow_clear_screen(void)
    {
      int icntr;

    for(icntr=0; icntr < 25; icntr++)
    {
    printf("\n");
    }
}
```

/* Program name : sunzdobs.c

Purpose : To correct the calculated zenith angle for parallax and refraction. The calculated zenith angle is returned from the sunazzd routine (part of this package). This is useful, if it is required to calculate the actual zenith angle that would have been observed by a surveyor. In practice it is normal to go the other way, i.e. observe a zenith angle and correct it for parallax and refraction to the centre of the earth.

Author : M.L. Roderick
Western Australian Department of Agriculture
GIS Group.

Date : 04-02-1992

Language : ANSI Standard C

Compilation :

Limitations : a) Refraction corrections are uncertain near the horizon (zenith = 90 degrees). This is a general problem in astronomy and surveying, and cannot (easily) be overcome.

References :

For the refraction corrections:

Anon., The Astronomical Almanac, Nautical Almanac Office United States Naval Observatory and Her Majesty's Nautical Almanac Office Royal Greenwich Observatory, US Government Printing Office and Her Majesty's Stationery Office, (for the year 1984).

The refraction correction formula used in this subroutine are given on page 859 of the Almanac.

For an advanced discussion on refraction:

Garfinke, L.B. (1967). "Astronomical Refraction in a Polytropic Atmosphere", The Astronomical Journal, 72(2), 235-254.

General astronomy principles

Glassock, J.T.C. (1983). Lecture Notes - Land Surveying VI - Astronomy. Queensland Institute of Technology, Brisbane, 1983.

Davis, R.E., Foote, F.E., Anderson, J.M. and Mikhail,E.M. (1981). Surveying, Theory and Practice. McGraw-Hill, New York, 1981.

or any other numerical astronomy or surveying textbook.

General theory

The sunazzd routine computes a theoretical zenith distance to the sun from the centre of the earth. For most applications this is adequate. Some applications, may require the actual incident zenith angle of solar radiation on the earth's surface. The best way to visualise this, is the angle a surveyor would have observed to the sun in the field. To derive this requires corrections for parallax and atmospheric refraction (sometimes termed astronomic refraction).

Parallax

Because the sun is not infinitely distant from the earth (as the other stars can be considered to be), a correction termed the parallax correction is required for precise applications. This correction is given by:

Parallax = asin (0.00004263 * sin Z(c))
where Z(c) = calculated zenith angle from the centre of the earth
= zenith_calc returned from sunazzd routine (in decimal degrees)

The parallax correction will not exceed 0.0025 decimal degrees (i.e. 9" of arc). As is seen, it is relatively minor.

The correction is added to zenith_calc in all cases.

Refraction

The sun's rays are refracted in the atmosphere, making the sun appear higher in the sky than it really is. The amount of refraction varies as a function, of solar zenith and atmospheric conditions. The traditional method for computing refraction used by surveyors is:

Method 1 (Z(c) < 70

Refraction = 0.00452 * tan Z(c) * [P/(273+t)
= result in decimal degrees.
where P = pressure at the ground in mb
t = temperature at the ground in deg celcius
Z(c) = defined above

The correction is within +- 0.00167 degrees (i.e. 6 " arc) for typical atmospheric conditions.

As can be seen, for a Z(c) of 0 deg, tan Z(c) = 0, so the correction = 0 degrees. However at high zenith distances (i.e. close to the horizon) refraction becomes uncertain, in the above equation (as it does in practice).

Method 2 (Z(c) > 70

For zenith angles > 70 deg, an alternative formulation of the refraction formula is:

$$\text{Refraction} = \left(\frac{P}{273+t}\right) * \left[\frac{0.1594 + 0.0196*a + 0.00002*a*a}{1 + 0.505*a + 0.0845*a*a}\right]$$

where a = altitude in decimal degrees
= 90 - Z(c)

This method is given by "The Astronomical Almanac on page B59, and is designed to avoid the tan function for angles approaching 90 degrees.

Note: The Astronomical Almanac recommends that method 1 be used for Z(c) < 75 and method 2 for the rest. By modelling the equations given, it was found that they are continuous at Z(c)=70.77 degrees. That is, at 70.77 degrees correction by method 1 = correction by method 2. Adopting the recommended value of 75 degrees leaves a 3" gap in the correction. For completeness sake, the cut-off was changed to 70 degrees (a nice even number). For either case, the difference is splitting hairs.

Discussion

In general, refraction is largely controlled by temperature at the earth's surface for Zenith angles < 75 deg (Garfinkel, 1967). For zenith angles > 75 degrees, accurate modelling requires a knowledge of the atmospheric parameters along the line of sight. This is generally not practical. For this reason, observations are restricted to solar zenith angles < 70 degrees in applications requiring precise observations of the solar zenith angle (such as surveying).

The typical magnitude of refraction corrections is shown in the table below:

34' at Zenith of 90 deg.
5' at Zenith of 80 deg.
1' at Zenith of 45 deg.
< 1' at Zenith less than 45 deg.

The actual zenith angle (i.e. angle observed by a surveyor at the earth's surface), is smaller than the "theoretical" version.

Whether you will want the theoretical (zenith_cab from the sunazzd routine) or actual zenith (zenith_obs) will depend on the application. For remote sensing studies, the actual value is required although either will probably be sufficient for practical appliations. The difference will

be minor for most solar zenith angles, and applications.

For determining the sun rise/set times, the theoretical value will be required. Sun rise/set times are calculated using a refraction value of 34' on the horizon, which is deemed to be the typical refraction. While refraction will vary throughout the year, the effect on sun rise/set times is relatively minor. For example, the sun approximately moves 15 deg / hour (i.e. 360 deg / 24 hours). Therefore, an uncertainty of say 0.17 deg in refraction will only make 0.17/15 = 0.011 hours (i.e. 41 seconds of time) difference in the calculation of sun rise/set times.

Revisions :

Version	Date	By	Details
1.0	04-02-1992	mlr	The original subroutine

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/* Constants rounded for 21 decimals. */
#define MY_PI       3.14159265358979323846

/* prototype definition – i.e. ANSI C declaration */
void sunzdobs (double zenith_calc, double pressure, double temperature, in printdataswitch, double
                *zenithobs);



/* the subroutine */

void sunzdobs
(
    double    zenith_calc,      /* <=    input the zenith_calc from the sunazzd routine
                                         (in decimal degrees)                          */
    double    pressure,         /* <=    input the pressure at the earth's surface
                                         (in mill-bars)                                */
    double    temperature,      /* <     input the temperature at the earth's surface
                                         (in degrees celcius)                          */
    int       printdataswitch,  /* <=    input causes the output data to be printed.
                                         = 1 for printing , =0 for no printing.        */
    double    *zenith_obs,      /* =>    returns a pointer to zenith_obs, the corrected
                                         zenith in decimal degrees                     */
)

{
    double      deg2rad, rad2deg;
    double      refraction, parallax, elevation;
    double      tempdbl1, tempdbl2, tempdbl3;

    /* some useful constants */
    deg2rad  =   MP_PI / 180;
    rad2deg  =   180 / MY_PI ;
```

```
/* check for zenith_calc in range 0->180 deg */
if (zenith_calc < 0 )
    {
      printf("Aenith angle must be greater than 0 degrees\n");
      printf("in module zunzdobs().\n");
      printf("…aborting …\n\n");
      exit (1);
    }
if (zenith_calc > 180)
    {
   (zenith_calc = 260 – zenith_calc ;
    }


/* set output value */
*zenith_obs = zenith_calc;

/* compute parallax and apply to zenith_calc */
tempdbll  =   0.00004263 * sin(zenith_calc*deg2rad);
parallax  =   asin(tempdbl1) * rad2deg;
if (parallax < 0)
    {
    parallax = ~1*parallax;             /* parallax is always +ve */
    }

*zenith_obs = *zenith_obs + parallax;



    1* compute refraction */
    if (*zenith_obs < 70)
    {
    tempdbll  =   pressure / (273+ temperature);
    refraction=   0.00452 * tan(*zenith_obs * deg2rad) * tempdbl1;
    }
else
    {
        tempdbll    =   pressure / (273+ temperature);
        elevation   =   90 - *zenith_obs;
        tempdbl2    =   (0.1594 + (0.0196 * elevation)  +  (0.00002 * elevation * elevation));
        tempdbl3    =   (1 + (0.505 * elevation)  +  (0.0845 * elevation * elevation));
        refraction  =   tempdbl1 * tempdbl2 / tempdbl3 ;
    }

*zenithobs    =   *zenith_obs - refraction;

/* Print out summary */
if (printdataswitch)
{
printf("\n\nSumnary of Calculations\n");
printf("----------------------------\n");
printf("Input Data :\n");
printf("---------------\n");
printf("Zenith (calc)     =   %lf\n", zenith_calc);
printf("Pressure (mb)  =   %lf\n", pressure);
printf("Temperature (c)     =   %lf\n", temperature);
printf("Calculated Data :\n");
printf("---------------------\n");
printf("Zenith (obs)      =   %lf\n", *zenith_obs);
}
```