Algorithms for the computation of the sun position from 2010 to 2110

The C++ header file `Sun_position_algorithms.h` contains the code of 5 new algorithms for the computation of the sun position, from 2010 to 2110.

Algorithms

The five algorithms have different accuracy and computational cost, and are suitable for different applications. The algorithms require the following quantities as input parameters:

- ◇ Universal time UT (Greenwich time), in hours;
- ◇ Day, Month, and Year;
- ◇ Difference TT-UT (terrestrial time - universal time), in seconds;
- ◇ Longitude and Latitude of the observer, in radians;
- ◇ Pressure in atm;
- ◇ Temperature in Celsius degrees.

In order to obtain UT, subtract from the local hour the difference due to the time zone with respect to Greenwich time (e.g., in Rome one must subtract an hour from the local time; in New York, one must add 5 hours). Minutes and seconds must be converted to decimal form (e.g., 12h30' must be written as 12.5).

Day, Month and Year are simply the integer numbers that appear in the date, with ranges, respectively, 1–31, 1–12, 2010–2110.

Among the required input parameters required, the difference TT-UT can be difficult to assess. However, errors up to 30 s (100 s if Algorithm 5 is excluded) are acceptable for this quantity. One can use the current value, or use the following extrapolation made on the last 100 years:

$$\Delta\tau = 96.4 + 0.567 * (\text{Year} - 2061). \tag{1}$$

No preferential geodesic system is given for Longitude and Latitude; since the two quantities are used to identify the direction of the local polar system (and *not* to identify a position on the earth surface), any preferred coordinate system can be employed (results for Zenith and Azimuth will be referred to the local system employed, of course). The standard system WSG84 is the most common.

The algorithms output five quantities. Two of them are the global sun coordinates in the equatorial system:

◇ Right ascension (in radians);
◇ Declination (in radians).

Right ascension is in the range $[0, 2\pi]$.

The local coordinates of output are:

◇ Hour angle (in radians);
◇ Zenith (in radians);
◇ Azimuth (in radians).

Hour angle and Azimuth are both 0 when the sun is toward south, increasing westwards, and they are given in the range $[-\pi, \pi]$. Zenith and Azimuth are the two angles of the standard local system with the polar axis along the vertical. They are the final output of the algorithms (and the coordinates on which the errors are computed). Hour angle can be associated with Declination to obtain a local equatorial system, with the polar axis aligned with the earth axis. Since Hour angle is computed before Zenith and Azimuth, the algorithms can be used in a short version if the Declination–Hour angle reference system is used, thus saving computational time (the conversion to Zenith–Azimuth is not made). However, the two systems are not strictly equivalent, since there are two corrections applied only in the Zenith–Azimuth system: the parallax correction, due to the fact that the observer is on the earth surface and not at the earth center, and the refraction correction due to the atmosphere. Short versions of the algorithms are recommended only for Algorithm 1 and 2, otherwise the errors increase significantly.

The algorithms follow two different approaches: in the first two algorithms, Right ascension and Declination are directly computed with a Fourier series, and then converted to local coordinates. In the other three algorithms, the ecliptic longitude is calculated, then converted to Right ascension and Declination, which are finally converted to local coordinates.

The correction for the refraction depends on the pressure and temperature. The included formula is independent of the main body of the algorithms and it can be replaced by any other formula that the user may prefer (the three parameters of the formula must be: Pressure, Temperature and sun elevation). For this reason, the formula is clearly highlighted in the code of the algorithms.

A detailed description of the algorithms is given in

R. Grena, Five new algorithms for the computation of sun position from 2010 to 2110 (in press).

The maximum errors of the algorithms (in the long versions), in absolute angular distance from the true position, obtained on 20 million trials, are:

Algorithm 1: 0.19 deg
Algorithm 2: 0.034 deg
Algorithm 3: 0.0094 deg
Algorithm 4: 0.0094 deg
Algorithm 5: 0.0027 deg.

In the short versions, the error of the first two algorithms is unchanged, the errors of Algorithm 3 and 4 increases by 30%, and the error of Algorithm 5 is almost doubled.

C++ header file

The header file `Sun_position_algorithms.h` defines a class (called `sunpos`) where all the input and output quantities are stored. Moreover, in this class five functions are defined that read the input data and calculate the output quantities, writing them in the class variables. In the following, instructions on how to perform sun position computations are given.

The first step is the inclusion of the header file in the program where position computations are to be performed. This is done inserting at the beginning of the program (in the include section) the line

```
#include "Sun_position_algorithms.h"
```

If the header is not in the same directory as the program, the path must be added before the file name. If the header is placed in the standard header directory, the symbols `<>` must be used instead of `""`.

To perform the calculations, a class of type `sunpos` must be created. The command is

```
sunpos Sun_Position;
```

With this command, a class of type `sunpos` named `Sun_Position` is created. The life span of the class should cover all the points in which calculations must be performed. Note that one class is sufficient to make all the computations required in the program (input parameters can be changed and the sun position recalculated in the same class at any moment).

Once the class has been declared, the input quantities can be set using a simple assignment to class members. The members representing inputs are named `UT`, `Day`, `Month`, `Year`, `Dt`, `Longitude`, `Latitude`, `Pressure`, `Temperature`. As an example, the complete assignment of the input variables in the class `Sun_Position` declared above will look like:

```
Sun_Position.UT = 11.0; // UT = Rome time - 1
Sun_Position.Day = 1;
Sun_Position.Month = 1;
Sun_Position.Year = 2010;
Sun_Position.Dt = 96.4 + 0.567*double(2010-2061);
Sun_Position.Longitude = 0.21787; // Rome longitude and latitude
Sun_Position.Latitude = 0.73117;
Sun_Position.Pressure = 1.0;
Sun_Position.Temperature = 20.0;
```

These input data are used for computing the solar position in Rome at midday (local hour 12.0), on January 1st, 2010.

An alternative, shorter way to assign the *initial* value of the input data is to define them directly in the class creation: the declaration of the class seen above can be replaced by the line

```
sunpos Sun_Position(11.0, 1, 1, 2010, 96.4 +
    0.567*double(2010-2061), 0.21787, 0.73117, 1.0, 20.0);
```

that creates the class and assigns directly all the input variables. However, no special protections or privacy conditions are given to the input values, which can be changed at any moment after the declaration with a simple assignment.

In this declaration, the arguments must be given in the same order in which they are listed above. In particular, the three arguments for the date date must be given in the order: day, month, year.

Once the input variables are assigned, the computation of the sun position can be performed by calling one of the five member functions representing the five algorithms. The functions are called `Algorithm1`, `Algorithm2`, `Algorithm3`, `Algorithm4` and `Algorithm5`. They require no arguments (input values are read from the class members); however, the optional argument `'s'` must be given if the user wishes to employ the short version of the algorithm. In this case, the computation will stop once the hour angle is found, without computing Zenith and Azimuth.

As an example, after defining `Sun_Position` and assigning the input data, with one of the two methods shown above, computation of sun position using Algorithm 3 can be performed with

```
Sun_Position.Algorithm3();
```

or, if the short algorithm is desired

```
Sun_Position.Algorithm3('s');
```

These functions do not give any output. Instead, they write directly the computed values in the class members where output data are stored. These members can then be read and used without restrictions. The class output members are named `RightAscension`, `Declination`, `HourAngle`, `Zenith`, `Azimuth`.

As an example, to print the computed value of the Zenith on the screen, the command

```
cout<<"Zenith = "<<Sun_Position.Zenith<<endl;
```

can be used.

At the end of the calculation, another computation with different input values and also a different algorithm can be performed, simply by changing the input values and then calling the algorithm function. The use of the class al-

lows to change only some of the input values, leaving the others unchanged. This is useful, since some input values are less likely than others to change between two computations (e.g., one usually needs to perform many computations at different times in the same location). As an example, after the calculations shown above, suppose that one desires to compute the sun position one hour later (local hour 13.0), using a more accurate algorithm (Algorithm 5), and printing the Zenith and Azimuth values. This can be done with

```
Sun_Position.UT = 12.0; // UT = Rome time - 1
Sun_Position.Algorithm5();
cout<<"Zenith = "<<Sun_Position.Zenith<<endl<<
    "Azimuth = "<<Sun_Position.Azimuth<<endl;
```

As a final complete example, a short program is shown below that computes the values at UT = 0, 1, 2, 3 ... to 23, on January 25th, 2020, in Rome, using the extrapolation formula for $\Delta\tau$ shown in Section 1, and keeping the pressure and temperature values fixed. The program prints a table of values on the screen.

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include"Sun_position_algorithms.h"

using namespace std;

int main () {
    sunpos Sun_Position(
        0.0, // initial hour (arbitrary - not used)
        25,1,2020, // date (day, month, year)
        96.4 + 0.567*double(2025-2061), //Dt
        0.21787, // Longitude in radians
        0.73117, // Latitude in radians
        1.0, 20.0); // Pressure (atm) and Temperature (Celsius)
    cout<<endl<<"UT\tZen\tAz"<<endl;
    for (Sun_Position.UT = 0.0; Sun_Position.UT < 23.1;
            Sun_Position.UT += 1.0) {
        Sun_Position.Algorithm5();
        cout<<Sun_Position.UT<<"\t"<<Sun_Position.Zenith<<"\t"
            <<Sun_Position.Azimuth<<endl;
    }
}
```

The screen output of this program is

```
UT       Zen      Az
0        2.72050  -2.75173
```

```
1       2.60415  -2.27183
2       2.43685  -1.95563
3       2.24918  -1.72944
4       2.05490  -1.54571
5       1.86133  -1.37913
6       1.67383  -1.21446
7       1.49471  -1.04050
8       1.33870  -0.847739
9       1.20778  -0.628351
10      1.11358  -0.378857
11      1.06666  -0.104799
12      1.07355  0.177227
13      1.13323  0.446305
14      1.23773  0.688385
15      1.37610  0.900630
16      1.53557  1.08803
17      1.72007  1.25900
18      1.90948  1.42343
19      2.10359  1.59325
20      2.29672  1.78559
21      2.48042  2.03000
22      2.63773  2.38154
23      2.73198  2.90807
```

Note that, despite the use of a C++ class, access to data is completely free (all the class variables are public), and no consistence checks are made. If the input data change, output data are not automatically updated: they are computed only when an algorithm function is explicitly called. One can also change output data with an assignment (it's useless, but possible). This choice of programming style is due to the fact that the library is aimed at numerical efficiency and generality of use; however, some caution is required by the user to avoid mistakes.