太陽位置の計算

この文書は，拡張アメダス気象データの操作プログラム EA DataNavi 8，さらには，グラフィックス表示関連ツール EA Graphic Tools 2022 に含まれる SolMap や SkyMap で用いられる，太陽位置の計算方法について解説したものである。採用されている太陽位置 (太陽高度角と太陽方位角) の計算方法自体は，一般的なものであるが，その計算に必要となる太陽視赤緯と均時差の計算法として，「**松本の方法（2022 年改）**」(2 章以降参照，[1], [2]) がデフォルトとして採用されている。

ここでは特に，この方法を詳しく解説し，付録 A では計算精度の比較結果についても触れるが，最初の 1 章では，やや教科書的ではあるものの，太陽位置の計算方法について概説する。

1　太陽位置の計算

太陽の位置は，天文学的には，その「視赤緯 δ [°]」と「時角 t [°]」の 2 つのパラメータ（**図 1** 参照）で表せば事足りる。時角 t [°] は次式で計算するが，E_t [°] で与える均時差の計算が肝要である。視赤緯 δ と均時差 E_t の計算方法については，我々の使う時刻と位置天文学で計算に用いる時刻との差である，補正時差項 ΔT について次章で論じた後，3 章で詳述する。

$$t = 15(T_m - 12) + (L - L_0) + E_t \tag{1}$$

ここに，T_m: 計算対象とする標準時 [時]，　L_0: 標準時を代表する地点の経度 [°]，
L : 計算対象地点の経度 [°]，　E_t: 計算対象時の均時差 [°]　である。

L_0, L は東経を正，西経を負とする。例えば，日本の場合，標準時を代表する地点は東経 135° の明石であるから，$L_0 = 135$ となるが，仮に西経 135° であれば $L_0 = -135$ とすればよい。

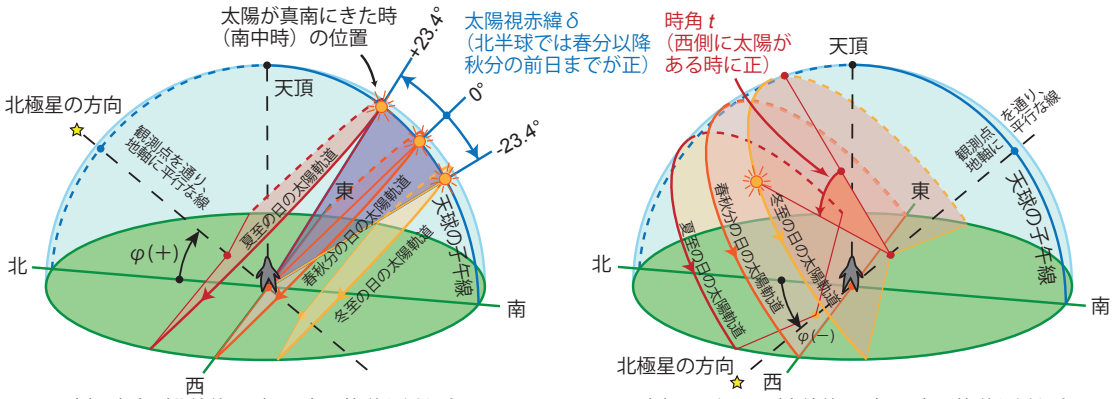

図 1　太陽の視赤緯 δ と時角 t の定義（季節は北半球の概念による）

建築環境工学分野では，太陽位置は，計算対象地点の緯度 φ [°] も指定して，その地における高度角 h [°] と方位角 A [°] で表す (**図 2**)。δ，t と h，A の関係は，球面三角法の公式により次のように表される（例えば，文献[11], [12])。

$$\sin h = \sin\varphi \sin\delta + \cos\phi \cos\delta \cos t \quad (2)$$

$$\sin A = \frac{\cos\delta \sin t}{\cos h} \quad (3)$$

$$\cos A = \frac{\sin h \sin\varphi - \sin\delta}{\cos h \cos\varphi} \quad (4)$$

式 (2) の $\sin h$ が正であれば昼間であるが，負であれば日の出前あるいは日没後である。太陽方位角 A は，真南を 0° とし，西回りを正，東回りを負としている [注1]。A(-180°≦ A ≦ 180°) の値は $\sin A$ の符号に注意して決めるとよい。$\sin A$ は，式 (3) の通りであるから，結局 t の符号，すなわち午前の太陽か，午後の太陽かに注意すればよい。

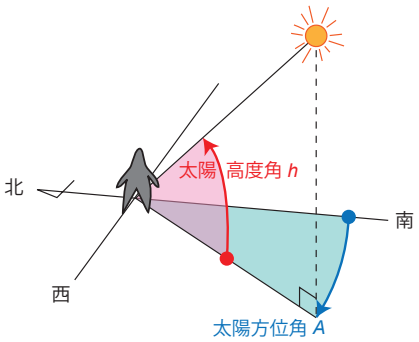

図 2　太陽の高度角 h と方位角 A の定義

すなわち，　$\sin A > 0$ $(t > 0)$　のとき　$A = 90 - \tan^{-1}\left(\dfrac{\sin A}{\cos A}\right)$ (5)

$\sin A < 0$ $(t < 0)$　のとき　$A = -90 - \tan^{-1}\left(\dfrac{\sin A}{\cos A}\right)$ (6)

次章以降に，**松本の式（2022 年改）**（文献[1], [2]）による太陽視赤緯 δ と均時差 E_t の計算方法を詳述する [注2]。

2　補正時差項 ΔT とユリウス世紀数 T の計算方法

2.1　「松本の方法（2022 年改）」開発の経緯 [注3]

松本は，文献[3]～[6]で，海上保安庁海洋情報部（開発開始当時，水路部）による計算方式[19]を援用した，太陽視赤緯 δ [°] と均時差 E_t [°]（T_e [min.]）の計算法（「**松本の式**」と称する）を提案した。これは，IAU（国際天文学連合）によって，2000 年までに定義や再定義のなされた，力学時 TD [h] に替わる地球時 TT [h] および地心座標時 TCG [h] を採用したもので，現在の位置天文学理論の共通基盤に基づいている。これは既に，旧版の拡張アメダス気象データのナビゲーション・プログラム（～EA DataNavi 7）における太陽位置の計算方法として用いられてきた。

このたび，以下の経緯と理由により，これまでの提案を改訂・改良してまとめ直すことになった。なお，この資料における計算方法を今後，（太陽視赤緯と均時差計算に関する）「**松本の方法（2022 年改）**」と称する。

[注1] 国際的には，太陽方位角は真北を 0° とし，西回りを正，東回りを負としている。式 (2)～式 (4) による A の計算値を，国際基準の太陽方位角 A' に変換するには次のようにする。

$A' = 180 - A$　$(0 \leqq A \leqq 180)$,　$A' = -180 - A$　$(-180 \leqq A \leqq 0)$

[注2] 付録には，比較的著名な「**山崎の式**」（文献[12], [13]）および「**赤坂の式**」（文献[14]～[17]）の解説も加えている。

[注3] 単に計算方法だけを理解したい読者は，この節を読み飛ばしても構わない。

(a) 惑星位置（軌道）の計算法としては，VSOP 87（Variations Séculaires des Orbites Planétairs, 1987）[20]が有名で，400 項に及ぶユリウス世紀数 T を変数とする三角関数係数群によって位置座標を決定するという大規模で難解なものであるが，計算プログラムが一般公開されていて利用しやすい。そのため，最新の位置天文学ではデファクト・スタンダードの計算方法と言えるようである。このことに関する理解が深まった。

(b) 欧米の建築環境工学の分野では，アメリカ・エネルギー省（DOE）の EnergyPlus™ 用の気象データファイル EPW で採用されている太陽高度角，太陽方位角データの計算アルゴリズム[21]が比較的有名で，プログラムのソース・コードも公開されている。この方法は，ベルギーの天文学者 Meeus が VSOP87 の大規模な数式を実用精度の範囲に簡素化した提案式[22]に基づいている。にも関わらず，計算精度に影響を及ぼす時差補正項 ΔT（後述）の扱い方を明示しておらず，改訂内容の公開された 2008 年以降も適切に精度よく使用できるものなのか不明である。

(c) 松本の方法の出典である海上保安庁海洋情報部の計算方式[19]や，かつて比較対象とした同庁ホームページに毎年掲げられる計算方法[23]（「海保 HP の方法」と呼ぶ）も，計算式の構成から見て，VSOP87 の派生型と見てよいと思われる。後述する ΔT の予測は一般に難しいとされている[19]から，毎年，最新の ΔT の確定値から誤差の少ない予測を立てて，VSOP87 または文献[19]の方法に基づいて，1 年の範囲で通用するように整理したものが，海保 HP の方法と考えられる。

(d) (a)〜(c) の経緯（考察）から，筆者が折に触れて，ΔT の推定式をアップデートしながら，最新の理科年表と松本の方法による計算結果を比較してきたのは，妥当なことであった（例えば，文献[6]）。ΔT の推定式のアップデートは，松本の方法が実装される「拡張アメダス気象データの操作プログラム」（EA DataNavi 6 や 7）の開発時期とリンクしていた。

(e) 2022 年春，新しい拡張アメダス気象データの操作プログラム（EA DataNavi 8）がリリースされ，扱われる気象データには 2086 年という将来の年のものも含まれることになった。そのため，50 年以上先の太陽位置も精度よく計算できるか否か，検討する必要に迫られることになった。別の言い方をすれば，ここで計算方法を見直すことはよい機会である。

特に今回は，2086 年などといった将来標準年気象データが開発中であるから，100 年程度将来にわたって使える太陽位置計算方法であることを意識した改良を考えるべきである。

(f) 松本の方法には，UT1 [h] と協定世界時 UTC [h] の差を無視する立場をとりながら，地球時 TT と地心座標時 TCG の差を考慮するようなアンバランスさがあった。Meeus が著書[22]で述べているように，かつての暦表時 ET [h] と力学時 TD，現行の地球時 TT は，本質的に同じ天文学上の不変の時間体系と考えても，実用上問題ないと考え直すに至った。そのための修正を施すべきである。

(g) Espenak と Meeus が (e) に関連する ΔT の推定式（「NASA の式」と呼ぶ）を提案しており[24]，例えば 2150 年といった将来まで，使えそうである。筆者の従来の方法に替えて採用することが妥当である。

次節で述べているのは，主に (f) と (g) に関する事項である。
なお，(f) に関して，ここでは，UT1 と UTC を区別せず，UT と表すことにする。

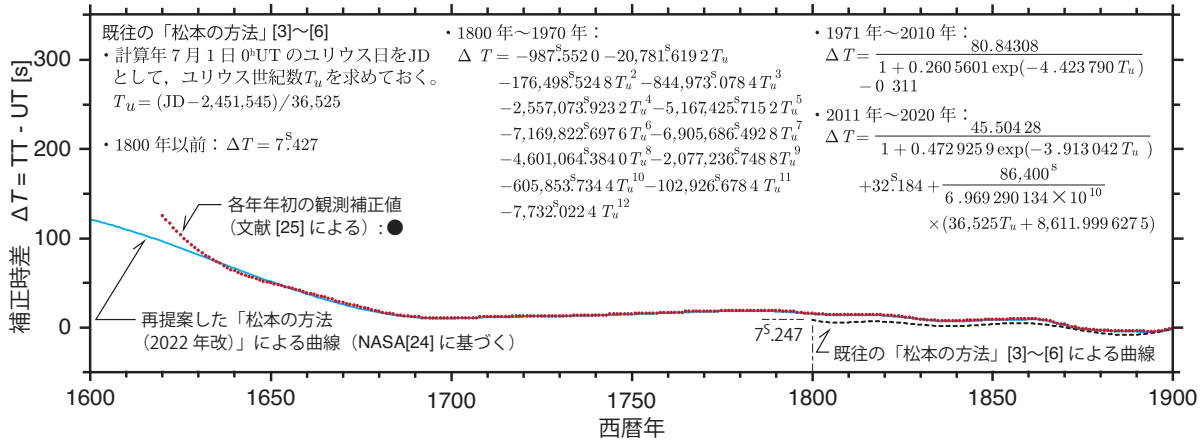

図 3　再提案した補正時差項 ΔT の経時変化（1/2）（1600 年〜1900 年）

2.2　補正時差項 ΔT の定義

地球時 TT（Terrestrial Time），力学時 TD（Dynamical Time），または暦表時 ET（Ephemeris Time）と世界時 UT（Universal Time）[h] の差を秒単位で表したものを補正時差項 ΔT と定義する[注4]。

$$\begin{aligned}\Delta T &= 3{,}600^{\mathrm{s}} \times (\mathrm{TT} - \mathrm{UT}) \\ &= 3{,}600^{\mathrm{s}} \times (\mathrm{TD} - \mathrm{UT}) = 3{,}600 \times (\mathrm{ET} - \mathrm{UT})\end{aligned} \tag{7}$$

ΔT は時間に応じて連続的に変化するものであるが，1 年の間に大きく変化することはない。そこで，本計算法では，太陽位置を計算する年の 7 月 1 日 0^{h}UT の値で 1 年間を通して変化しないものとして扱う。

2.3　補正時差項 ΔT の計算式（再定義）

2.1 節の箇条書き (e)〜(g) に述べた通り，文献[24]を参照し，計算対象の西暦年 YYYY（YYYY= $1600, \ldots, 2150$）ごとに以下に示す式を採用する[注5]。

まず，以下の式でパラメータ y（正の実数の年）を求める。端数の 0.5 は 7 月 1 日 0^{h}UT を表している。

$$y = \mathrm{YYYY} + 0.5 \tag{8}$$

y をある元期からの差 t に変更した上で，t に関する以下のような形式の多項式で ΔT を求める。

$$\begin{aligned}\Delta T &= a_0 + a_1\,t + a_2\,t^2 + \cdots + a_n\,t^n \\ &= a_0 + t\,(a_1 + t\,(a_2 + \cdots t\,(a_{n-1} + t\,a_n)\cdots))\end{aligned} \tag{9}$$

[注4] 文献[3]〜[6]において，ΔT_1 と記したものと同義と考える。ET は 1984 年以前，TD は 1984 年〜1990 年，TT は 1991 年から現在に至るまで，天文学分野で用いられている。なお，よく知られているように日本標準時 JST と UT とは，$\mathrm{UT} = \mathrm{JST} - 9^{\mathrm{h}}$ の関係にある。

[注5] 文献[24]では，紀元前 1999 年〜紀元 3000 年を対象とする推定式が提案されているが，工学的実用性の観点から，ユリウス暦（現行のグレゴリオ暦採用以前）の範囲や遠い将来の範囲は除き，1600 年〜2150 年を対象とすることにした。

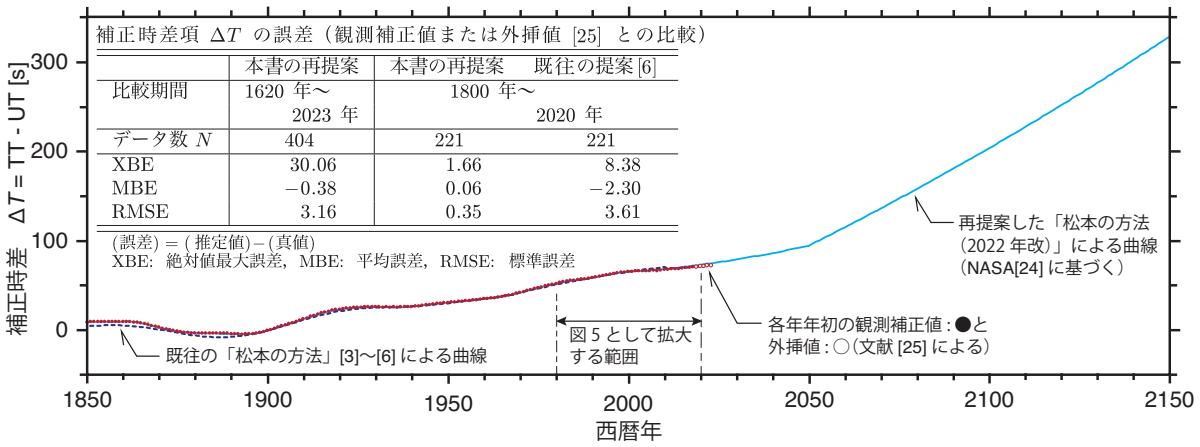

図 4　再提案した補正時差項 ΔT の経時変化（2/2）（1850 年～2150 年）

元期からの差 t，次数 n と係数 $a_i\,(i = 1, 2, \ldots, n)$ は，1600 年～2150 年の範囲を以下に示す 11 の年代区分別に与えられる [注6]。

- 1600 年～1700 年: $t = y - 1600,\ n = 3$

$$a_0 = 120 \quad a_1 = -0.980\,8 \quad a_2 = -0.015\,32 \quad a_3 = 1/7,129$$

- 1701 年～1800 年: $t = y - 1700,\ n = 4$

$$a_0 = 8.83 \qquad a_1 = 0.160\,3 \quad a_2 = -0.005\,928\,5 \quad a_3 = 0.000\,133\,36$$
$$a_4 = -1/1,174,000$$

- 1801 年～1860 年: $t = y - 1800,\ n = 7$

$$a_0 = 13.72 \qquad a_1 = -0.332\,447 \qquad a_2 = 0.006\,861\,2$$
$$a_3 = 0.004\,111\,6 \qquad a_4 = -0.000\,374\,36 \qquad a_5 = 0.000\,012\,127\,2$$
$$a_6 = -0.000\,000\,169\,9 \quad a_7 = 0.000\,000\,000\,875$$

- 1861 年～1900 年: $t = y - 1860,\ n = 5$

$$a_0 = 7.62 \qquad a_1 = 0.573\,7 \qquad a_2 = -0.251\,754 \quad a_3 = 0.016\,806\,68$$
$$a_4 = -0.000\,447\,362\,4 \quad a_5 = 1/233\,174$$

- 1901 年～1920 年: $t = y - 1900,\ n = 4$

$$a_0 = -2.79 \qquad a_1 = 1.494\,119 \quad a_2 = -0.059\,893\,9 \quad a_3 = 0.006\,196\,6$$
$$a_4 = -0.000\,197\,362\,4$$

[注6] 式 (9) 第 2 式の Müller の多項式表記は，一番内側の括弧から計算を進めるアルゴリズムを表しており，誤差を抑えるコーディング技法として推奨されるものである。

なお，係数の桁の表現から明らかなように，提案式は必ずしも 1 ms のオーダーの精度があるとは言えないが，後のユリウス世紀数 T の計算に及ぼす影響などを考慮して，（従来の通り）1 ms のオーダーまで計算するものとする。

- 1921 年〜1940 年: $t = y - 1920,\ n = 3$

  $a_0 = 21.20 \quad a_1 = 0.844\,93 \quad a_2 = -0.076\,100 \quad a_3 = 0.002\,093\,6$

- 1941 年〜1960 年: $t = y - 1950,\ n = 3$

  $a_0 = 29.07 \quad a_1 = 0.407 \quad a_2 = -1/233 \quad a_3 = 1/2,547$

- 1961 年〜1985 年: $t = y - 1975,\ n = 3$

  $a_0 = 45.45 \quad a_1 = 1.067 \quad a_2 = -1/260 \quad a_3 = -1/718$

- 1986 年〜2005 年: $t = y - 2000,\ n = 5$

  $a_0 = 63.86 \quad a_1 = 0.334\,5 \quad a_2 = -0.060\,374 \quad a_3 = 0.001\,727\,5$
  $a_4 = 0.000\,651\,814 \quad a_5 = 0.000\,023\,735\,99$

- 2006 年〜2050 年: $t = y - 2000,\ n = 2$

  $a_0 = 62.92 \quad a_1 = 0.322\,17 \quad a_2 = 0.005\,589$

- 2051 年〜2150 年: $t = y - 1820,\ n = 2$

  $a_0 = -205.724 \quad a_1 = 0.562\,8 \quad a_2 = 0.003\,2$

図 3 と図 4 に dT の経時変化を示す。ただしここでは，●で示した観測補正値，◯で示した外挿値[25]と比較するため，$y =$ YYYY，すなわち毎年 1 月 1 日 0^{h}UT における値を計算している。図 3 の余白には既往の提案式を記し，その変化（1800 年〜2020 年の範囲）も破線で示した。

図 3 によれば，1640 年以前では観測補正値との乖離が大きく，約 30 s に達している。しかし，観測精度や補正精度の高さが期待できない遠い過去であることと，工学的な利用価値の両面を鑑みれば，大きな問題とすべきではなかろう。また，図 4 から，2090 年頃に ΔT が 180 s，すなわち 3 分に達すると推定されることに注意したい。

2.4 補正時差項 ΔT の誤差評価

図 4 の余白に誤差評価結果も記した。また，拡張アメダス気象データ（年別ファイル）開発に関連して注目する直近の 40 年間における ΔT の経時変化を図 5 に示す。再提案によって推定精度が向上しており，拡張アメダス気象データの操作プログラムに実装するに相応しいと考えられる。

1981 年〜2010 年においては，太陽位置の計算に用いるプログラム（EA DataNavi 7 など）によって，太陽位置の計算結果に差が生じることは確実であるが，最終的な太陽高度角 h と方位角 A の収録精度である 0.1° には，影響しないことは確認済みである。

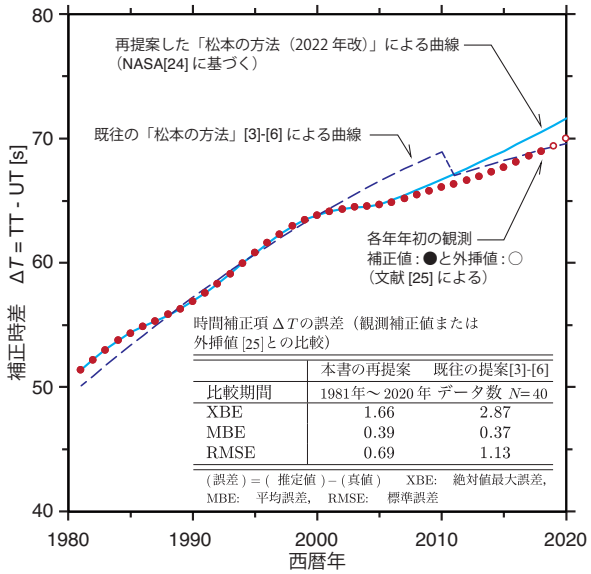

図 5 補正時差項 ΔT の変化（1981 年〜2020 年）

2.5 ユリウス世紀数 T の計算方法

ある年月日のユリウス通日（整数）を $\overline{\mathrm{JD}}$ とすると，その日の 0^{h}UT のユリウス日（実数）JD は，$\mathrm{JD} = \overline{\mathrm{JD}} - 0.5$ であることに注意する（天文学上の日付変更は正午のため）。よって，その日の日本標準時 H^{h}JST のユリウス日は $\mathrm{JD} = \overline{\mathrm{JD}} + (H - 9^{\mathrm{h}})/24 - 0.5$ である。本報の計算方法は，市民時ではなく，天文学で用いられる暦表系のユリウス日 JDE（Julian Ephemeris Day）に基づくものである。

つまり，先述の式 (9) によって得られる ΔT を用いて，

$$\begin{aligned} \mathrm{JDE} &= \mathrm{JD} + \Delta T/3,600 \\ &= \overline{\mathrm{JD}} + (H - 9^{\mathrm{h}})/24 - 0.5 + \Delta T/3,600 \end{aligned} \tag{10}$$

と表現する。さらに，以下に展開する各種計算に用いる時刻パラメータとして，J2000.0 元期のユリウス世紀数 T を用いる。

$$T = (\mathrm{JDE} - 2\,451\,545.0)\,/36\,525 \tag{11}$$

なお，ユリウス通日 $\overline{\mathrm{JD}}$ の求め方，特に Microsoft® エクセルでの計算方法などについては，ネットワーク上で多くの記事の検索結果が得られるので，参考にするとよいだろう。プログラムは付録 B を参照されたい。

3 太陽視赤緯 δ と均時差 E_t の計算手順

緯度 $\varphi\,[^\circ]$（北緯が正），経度 $L\,[^\circ]$（東経が正）の地点において，西暦 YYYY 年 MM 月 DD 日の日本標準時（JST）hh:mm:ss に太陽高度角 $h\,[^\circ]$ と太陽方位角 $A\,[^\circ]$ を求めるという基本的な問題を考え，その手順を説明する。

なお，以下の記述において，三角関数の引数は全て ° 単位で取り扱う。逆三角関数値も同様である。

まず，JST の hh:mm:ss を [h] 単位で表した上で，UT [h] に変更しておく。UT は負の値でも構わない。

$$\mathrm{UT} = \mathtt{hh} + \mathtt{mm}/60 + \mathtt{ss}/3,600 - 9.0000000 \tag{12}$$

3.1 計算の各ステップ

Step 1: 補正時差項 ΔT の算定

2.3 節の年代区分に注意して，式 (8) と式 (9) によって，$\Delta T\,[\mathrm{s}]$ を小数点以下 3 桁まで求める。$\Delta T\,[\mathrm{s}]$ は YYYY 年に固有の定数とみなす。

Step 2: ユリウス世紀数 T と T_u の算定

YYYY 年 MM 月 DD 日のユリウス通日 $\overline{\mathrm{JD}}$ を求め [注7]，式 (10) と式 (11) を用いて算定する。

[注7] O（ゼロ）UT を前提とすると，ユリウス日 JDE には必ず端数 0.5 がつく。1900 年 1 月 1 日以降であれば，この日を起日（1）とする通し日数に 2 415 019.5 を加えればよい。

さらに，ユリウス日 JD を $\mathrm{JD} = \overline{\mathrm{JD}} + \mathrm{UT}/24 - 0.5$ として求め，次の T_u も算定する。

$$T_u = (\mathrm{JD} - 2\,451\,545.0)\,/36\,525 \tag{13}$$

Step 3: 平均太陽の赤経 α_M の計算

Step: 2 で求めた T_u（T ではない）を用いて，平均太陽の赤経 α_M [°] を次の 2 式で算定する。

$$\begin{aligned}\alpha_m = &\Big(18^{\mathrm{h}}41^{\mathrm{m}}50^{\mathrm{s}}.548\,41 + 8,640,184^{\mathrm{s}}.812\,866\,T_u \\ &+ 0^{\mathrm{s}}.093\,104\,T_u^2 - 0^{\mathrm{s}}.000\,006\,2\,T_u^3\Big) \bmod 24 \end{aligned} \tag{14}$$

$$\alpha_M = 15\,\alpha_m \tag{15}$$

式 (14) は国際天文連合（IAU）の標準式で，単位は [h] であることに注意されたい。式 (15) では，$360^\circ/24^{\mathrm{h}} = 15$ を乗じて，これを [°] 単位に直している。

Step 4: 黄道傾斜角 ε の計算

T をパラメータとして，表 1 の係数 V_i（$i = 0, \cdots, 3$），X_j, Y_j, Z_j（$j = 1, 2$）を用いて，次式で ε [°] を計算する。

$$\varepsilon = \left(V_0 + \sum_{i=1}^{3} V_i T^i\right)/3,600S + \sum_{j=1}^{2} X_j \cos(Y_j T + Z_j) \tag{16}$$

式 (16) の余弦関数の変数（括弧内）は [°] の単位であり，適宜 360° の剰余（modulo）として計算する。

表 1　黄道傾斜角 ε 計算用の係数

i	V_i [″]	j	X_j [°]	Y_j [°]	Z_j [°]
0	−84,381.448	–			
1	+46.815	1	−0.002 56	1,934	235
2	+0.000 590	2	−0.000 15	72,002	201
3	−0.001 813	–			

Step 5: 視黄経 Ψ の計算

表 2 の係数 P_i, Q_i, R_i（$i = 1, \cdots, 18$）を用いて，次式で Ψ [°] を計算する。パラメータは T である。

なお，本書付録（B）には，その計算のためのソース・コードも含まれている。

$$\Psi = \left[\sum_{i=1}^{17} P_i \cos(Q_i T + R_i) + P_{18} T \cos(Q_{18} T + R_{18}) + 36,000.769\,5\, T + 280.460\,2 \right] \bmod 360 \tag{17}$$

この式においても，余弦関数の変数（括弧内）は [°] 単位である。適宜 360° の剰余（modulo）として計算する。

なお，式 (16) や式 (17) における総和 $\sum$ を計算する順番は，数式的にはここに記した通りであるが，桁落ちを防ぐため，値の小さいものから順に計算して加算すべきである。

表 2　視黄経 Ψ，均時差 T_e 計算用の係数

i	P_i [°]	Q_i [°]	R_i [°]	i	P_i [°]	Q_i [°]	R_i [°]
1	+1.914 7	35,999.05	267.52	10	+0.000 7	9,038	64
2	+0.020 0	71,998.1	265.1	11	+0.000 6	33,718	316
3	+0.020 0	32,964	158	12	+0.000 5	155	118
4	+0.001 8	19	159	13	+0.000 5	2,281	221
5	+0.001 8	445,267	208	14	+0.000 4	29,930	48
6	+0.001 5	45,038	254	15	+0.000 4	31,557	161
7	+0.001 3	22,519	352	16	+0.004 8	1,934	145
8	+0.000 7	65,929	45	17	−0.000 4	72,002	111
9	+0.000 7	3,035	110	18	−0.004 8	35,999	268

Step 6: 視赤緯 δ と均時差 E_t の計算

Step: 5 までに得られた平均太陽の赤経 α_M，黄道傾斜角 ε，視黄経 Ψ と，再び表 2 の係数の一部 P_i, Q_i, R_i（$i = 16, 17$）を用いて，次のように計算する。

$$\delta = \arctan\left(\frac{\sin\Psi \sin\varepsilon}{\sqrt{1 - \sin^2\Psi \sin^2\varepsilon}} \right) \tag{18}$$

$$E_t = \left[\sum_{i=16}^{17} P_i \cos(Q_i T + R_i) - 0.005\,7 \right] \cos\varepsilon + \arctan\left(\frac{\tan\alpha_M - \tan\Psi \cos\varepsilon}{1 + \tan\alpha_M \tan\Psi \cos\varepsilon} \right) \tag{19}$$

$$T_e = \frac{1}{15} E_t \tag{20}$$

均時差 E_t は式 (19) で [°] の単位で計算される。また，式 (20) 右辺の項「1/15」から分かるように，均時差 T_e は [h] の単位で計算される。なお，これら 2 式は，本報の計算式の元となった文献[19]や[23]が示してているものではなく，筆者のオリジナルである [注8]。

[注8] 海上保安庁海洋情報部（旧水路部）は，δ と T_e をフーリエ余弦級数で近似する式を提案している。筆者はこの別式を用いても，僅かではあるが煩雑さが回避でき，そこそこの精度が得られると考えて，独自に展開した。

3.2 最終 Step: 太陽高度角 h と方位角 A の計算

教科書通りに時角 $t\,[^\circ]$ に均時差 E_t または T_e を反映させ，視赤緯 δ とともに球面三角法の公式に適用すればよい。すなわち，既出の式 (1)〜式 (6) に代入して求める [注9]。

3.3 その他の計算—地心距離 r

地球の中心から太陽の中心までの真距離（天文単位 AU，ここでいう地心距離）r は，大気圏外法線面日射量 $I_0\,[\mathrm{W/m^2}]$ を太陽定数 $\overline{I_0} = 1367\,[\mathrm{W/m^2}]$ から時刻別 [注10] に設定する場合（式 (21)）などに，重要なパラメータである。

$$I_0 = \overline{I_0}/r^2 \tag{21}$$

r は，ユリウス世紀数 T の関数として，以下の式で計算する [19]。

$$r = \sum_{i=1}^{8} S_i \cos(U_i T + W_i) + S_9 T \cos(U_9 T + W_9) \tag{22}$$

式中の係数 S_i，U_i，W_i は下表の通りである。

表 3　地心距離 r 計算用の係数

i	S_i [–]	U_i [°]	W_i [°]
1	1.000 140	0.0	0.0
2	0.016 706	35,999.05	177.53
3	0.000 139	71,998	175
4	0.000 031	445,267	298
5	0.000 016	32,964	68
6	0.000 016	45,038	164
7	0.000 005	22,519	233
8	0.000 005	33,718	226
9	−0.000 042	35,999	178

この r の計算式のプログラム・ソース・コード（FORTRAN，C/C++）と検証用の計算例（データ）も付録 D に含まれている。

[注9] チップス的なコンピュータ技法であるが，日中の太陽方位角 A の計算精度を上げるには，$\sin A = \cos\delta \sin t / \cos h$ を用いたり，$\tan A$ として計算することも一考に値する。また，δ や E_t の値は 1 日を通して大きく変化するものではないので，例えば，E ①理科年表の日別値を引用するような調子で，計算対象日の $0^{\mathrm{h}}\mathrm{UT}$ の計算値で代表させる，②計算対象地域のタイム・ゾーンにおける正午の値で代表させることが考えられる。しかしながら，後述する計算例（表 4）が示すように，精度確保の観点からは，時刻ごとに計算するか，②の方法が好ましい。

[注10] 日別程度に考えるのが現実的であろう。

4 計算例

4.1 簡易な例

東京（緯度 $\varphi = +35°41'06''$，経度 $L = 139°45'36''$）における 2020 年 7 月 24 日 15^{h}JST（6^{h}UT）における太陽視赤緯 δ [°]，均時差 T_e [s]，太陽高度角 h [°] および方位角 A [°] を求めるものとする。

計算結果は表 4 に一括して記した。0^{h}UT と 6^{h}UT の行の違いは，δ と T_e を計算した時刻の違いである。（15^{h}JST は 6^{h}UT に相当する。）

表 4　東京 2020 年 7 月 24 日 15^{h}JST に対する計算例

緯度 $\varphi = +35°41'06''$，経度 $L = +139°45'36''$

方法	δ [°]	T_e [s]	h [°]	A [°]
理科年表 0^{h}UT	19.808 9	−392.1	45.053 0	82.593 0
理科年表 6^{h}UT*	19.755 9	−392.4	45.026 6	82.528 0
本報の方法 0^{h}UT	19.808 7	-392.0	45.052 7	82.592 9
本報の方法 6^{h}UT	19.755 6	-392.3	45.026 1	82.528 0
赤坂の方法	19.807 8	-392.0	45.052 0	82.592 1

* 前後の日付の理科年表[18]のデータから 2 次式で内挿した。

理科年表：文献[18]，赤坂の方法：文献[14]を参照した。

4.2 プログラミング・テスト・ベッド用の例

2020 年用の補正時差項 ΔT を求めた上で，3 月 30 日（通日 90 日目），6 月 28 日（通日 180 日目），9 月 26 日（通日 270 日目）の太陽赤緯（`Sol.Decl.`），均時差（`Eq.Time`）を時別（1 時〜24 時）に算出する。さらに，東京（北緯 $\varphi = 35.692$，東経 $L = 139.750$ とする）と鹿児島（北緯 $\varphi = 31.555$，東経 $L = 130.541$ とする）の 2 地点における時角（`Hr.Angle`），太陽高度角（`Altitude`），太陽方位角（`Azimuth`）を計算する。

同様の計算を，2086 年の 3 月 31 日（通日 90 日目），6 月 29 日（通日 180 日目），9 月 27 日（通日 270 日目）についても行う。

計算結果であるが，先ず ΔT は以下の通りである。

- 2020 年（7 月 1 日 0^{h}UT）　$\Delta T = 71.873$ [s]
- 2086 年（7 月 1 日 0^{h}UT）　$\Delta T = 171.532$ [s]

他の計算結果は，全て度 [°] の単位で示すことにした。太陽赤緯（`Sol.Decl.`）と均時差（`Eq.Time`）は，小数点以下 5 桁まで（6 桁目を四捨悟入）とし，その他は，小数点以下 4 桁まで（5 桁目を四捨悟入）とした。

結果の一覧を表 5 と表 6 にまとめて掲げた。これらの結果は，ユーザーがプログラム・コードを書き，その出力を検証する場合などに有用であろう。

表 5　2020 年の 3 日間における東京・鹿児島に対する計算例

Matsumoto Rev. Method (2022) ... Year:2020 Delta T: 71.873 sec. [All Units: deg. of angle]

		Tokyo				Kagoshima			
	Latitude, Longitude:	35.692	139.750			31.555	130.547		
YYYY/MM/DD	HHh:	Sol.Decl.	Eq.Time	Hr.Angle	Altitude	Azimuth	Hr.Angle	Altitude	Azimuth
2020/03/30	01h:	3.73383	-1.14699	-161.3870	-46.8974	-152.2178	-170.6003	-53.5941	-164.0620
2020/03/30	02h:	3.75002	-1.14387	-146.3839	-39.5538	-134.2338	-155.5972	-47.7419	-142.1890
2020/03/30	03h:	3.76621	-1.14075	-131.3807	-29.8322	-120.3355	-140.5941	-38.5080	-125.9543
2020/03/30	04h:	3.78240	-1.13763	-116.3776	-18.7596	-109.2503	-125.5910	-27.4089	-113.9322
2020/03/30	05h:	3.79859	-1.13451	-101.3745	-6.9610	-99.7793	-110.5878	-15.3266	-104.4138
2020/03/30	06h:	3.81477	-1.13139	-86.3714	5.1694	-90.9897	-95.5847	-2.7471	-96.1789
2020/03/30	07h:	3.83095	-1.12827	-71.3683	17.3310	-82.0712	-80.5816	10.0262	-88.3408
2020/03/30	08h:	3.84713	-1.12515	-56.3652	29.2106	-72.1270	-65.5785	22.7451	-80.0910
2020/03/30	09h:	3.86331	-1.12203	-41.3620	40.3564	-59.9061	-50.5754	35.1130	-70.4176
2020/03/30	10h:	3.87949	-1.11891	-26.3589	49.9593	-43.5166	-35.5722	46.6299	-57.6918
2020/03/30	11h:	3.89566	-1.11580	-11.3558	56.5250	-20.8643	-20.5691	56.2556	-39.1256
2020/03/30	12h:	3.91183	-1.11268	3.6473	58.0487	6.8880	-5.5660	61.8658	-11.8421
2020/03/30	13h:	3.92800	-1.10956	18.6504	53.8753	32.7631	9.4371	60.9834	19.7085
2020/03/30	14h:	3.94417	-1.10645	33.6536	45.6130	52.2191	24.4402	54.0900	44.7293
2020/03/30	15h:	3.96033	-1.10333	48.6567	35.1378	66.3298	39.4433	43.8393	61.4867
2020/03/30	16h:	3.97650	-1.10022	63.6598	23.5766	77.2757	54.4464	32.0450	73.2373
2020/03/30	17h:	3.99266	-1.09710	78.6629	11.5311	86.6159	69.4496	19.5628	82.4429
2020/03/30	18h:	4.00882	-1.09399	93.6660	-0.6318	95.3935	84.4527	6.8204	90.5256
2020/03/30	19h:	4.02497	-1.09088	108.6691	-12.6158	104.4358	99.4558	-5.9072	98.4156
2020/03/30	20h:	4.04113	-1.08776	123.6722	-24.0861	114.5876	114.4589	-18.3646	106.9184
2020/03/30	21h:	4.05728	-1.08465	138.6753	-34.5533	126.8942	129.4620	-30.2124	116.9805
2020/03/30	22h:	4.07343	-1.08154	153.6785	-43.2158	142.6352	144.4651	-40.8827	129.9338
2020/03/30	23h:	4.08957	-1.07843	168.6816	-48.8345	162.6983	159.4682	-49.3458	147.5222
2020/03/30	24h:	4.10572	-1.07532	183.6847	-50.0596	-174.2697	174.4714	-53.9525	170.6014
2020/06/28	01h:	23.27937	-0.80563	-161.0456	-28.3668	-160.1787	-170.2590	-34.3784	-169.1456
2020/06/28	02h:	23.27743	-0.80777	-146.0478	-22.8543	-146.1688	-155.2611	-30.2736	-153.5688
2020/06/28	03h:	23.27548	-0.80990	-131.0499	-15.0400	-134.1651	-140.2632	-23.2763	-140.2629
2020/06/28	04h:	23.27351	-0.81203	-116.0520	-5.5786	-123.9813	-125.2654	-14.1929	-129.3159
2020/06/28	05h:	23.27154	-0.81417	-101.0542	5.0141	-115.1684	-110.2675	-3.6931	-120.2815
2020/06/28	06h:	23.26955	-0.81630	-86.0563	16.3657	-107.2186	-95.2696	7.7496	-112.6007
2020/06/28	07h:	23.26756	-0.81843	-71.0584	28.2059	-99.5959	-80.2718	19.8161	-105.7508
2020/06/28	08h:	23.26555	-0.82055	-56.0606	40.3173	-91.6255	-65.2739	32.2867	-99.2229
2020/06/28	09h:	23.26352	-0.82268	-41.0627	52.4687	-82.1442	-50.2760	44.9918	-92.3749
2020/06/28	10h:	23.26149	-0.82480	-26.0648	64.2493	-68.3009	-35.2781	57.7550	-83.9802
2020/06/28	11h:	23.25945	-0.82692	-11.0669	74.3016	-40.6760	-20.2803	70.2210	-70.2273
2020/06/28	12h:	23.25739	-0.82905	3.9310	77.1133	16.4040	-5.2824	80.4727	-30.7320
2020/06/28	13h:	23.25532	-0.83116	18.9288	69.4238	57.9967	9.7155	78.0415	48.4415
2020/06/28	14h:	23.25324	-0.83328	33.9267	58.1578	76.4026	24.7134	66.6086	75.3589
2020/06/28	15h:	23.25115	-0.83540	48.9246	46.1052	87.3953	39.7113	53.9860	86.7352
2020/06/28	16h:	23.24904	-0.83751	63.9225	33.9391	95.8814	54.7092	41.2157	94.4663
2020/06/28	17h:	23.24693	-0.83962	78.9204	21.9411	103.5667	69.7070	28.5611	101.1319
2020/06/28	18h:	23.24480	-0.84174	93.9183	10.3243	111.2883	84.7049	16.1879	107.6970
2020/06/28	19h:	23.24266	-0.84385	108.9162	-0.6731	119.6251	99.7028	4.2755	114.7387
2020/06/28	20h:	23.24051	-0.84595	123.9140	-10.7332	129.0938	114.7007	-6.9325	122.7613
2020/06/28	21h:	23.23835	-0.84806	138.9119	-19.4121	140.1859	129.6986	-17.0775	132.3023
2020/06/28	22h:	23.23617	-0.85016	153.9098	-26.1145	153.2527	144.6965	-25.6303	143.9150
2020/06/28	23h:	23.23399	-0.85227	168.9077	-30.1527	168.2026	159.6944	-31.8656	157.9465
2020/06/28	24h:	23.23179	-0.85437	183.9056	-30.9673	-175.8140	174.6923	-34.9781	174.0453
2020/09/26	01h:	-1.20818	2.13946	-158.1005	-49.9744	-144.5625	-167.3139	-57.3716	-155.9705
2020/09/26	02h:	-1.22440	2.14305	-143.0970	-41.4391	-126.7937	-152.3103	-49.9567	-133.7715
2020/09/26	03h:	-1.24062	2.14663	-128.0934	-30.9051	-113.5056	-137.3067	-39.6057	-118.3704
2020/09/26	04h:	-1.25685	2.15021	-113.0898	-19.3450	-102.9155	-122.3031	-27.8236	-107.1580
2020/09/26	05h:	-1.27307	2.15379	-98.0862	-7.3069	-93.6988	-107.2995	-15.3647	-98.1469
2020/09/26	06h:	-1.28929	2.15736	-83.0826	4.8573	-84.9036	-92.2960	-2.6310	-90.1024
2020/09/26	07h:	-1.30552	2.16094	-68.0791	16.8496	-75.7107	-77.2924	10.1065	-82.1476
2020/09/26	08h:	-1.32174	2.16451	-53.0755	28.3187	-65.2121	-62.2888	22.5867	-73.4540
2020/09/26	09h:	-1.33796	2.16809	-38.0719	38.7283	-52.2078	-47.2852	34.4488	-62.9675
2020/09/26	10h:	-1.35418	2.17166	-23.0683	47.1633	-35.1790	-32.2817	45.0608	-49.1039
2020/09/26	11h:	-1.37040	2.17523	-8.0648	52.1876	-13.2245	-17.2781	53.2196	-29.7297
2020/09/26	12h:	-1.38662	2.17880	6.9388	52.3667	11.4080	-2.2745	56.9877	-4.1762
2020/09/26	13h:	-1.40285	2.18237	21.9424	47.6371	33.6687	12.7290	54.8985	22.5239
2020/09/26	14h:	-1.41907	2.18594	36.9459	39.3827	51.0237	27.7326	47.8219	43.8558
2020/09/26	15h:	-1.43529	2.18950	51.9495	29.0687	64.2465	42.7362	37.7763	59.1266
2020/09/26	16h:	-1.45151	2.19307	66.9531	17.6439	74.8617	57.7397	26.1958	70.4117
2020/09/26	17h:	-1.46773	2.19663	81.9566	5.6627	84.0938	72.7433	13.8454	79.4953
2020/09/26	18h:	-1.48395	2.20019	96.9602	-6.5171	92.8632	87.7469	1.1424	87.5560
2020/09/26	19h:	-1.50017	2.20375	111.9638	-18.6000	101.9867	102.7504	-11.6368	95.4574
2020/09/26	20h:	-1.51638	2.20731	126.9673	-30.2458	112.3984	117.7540	-24.2381	104.0301
2020/09/26	21h:	-1.53260	2.21087	141.9709	-40.9325	125.3964	132.7575	-36.3207	114.3622
2020/09/26	22h:	-1.54882	2.21443	156.9744	-49.7311	142.7777	147.7611	-47.2773	128.1876
2020/09/26	23h:	-1.56504	2.21798	171.9780	-55.0768	165.8962	162.7647	-55.8798	148.1277
2020/09/26	24h:	-1.58126	2.22154	186.9815	-55.2859	-167.6808	177.7682	-59.9522	175.5412

表 6　2086 年の 3 日間における東京・鹿児島に対する計算例

Matsumoto Rev. Method (2022) ... Year:2086 Delta T: 171.532 sec. [All Units: deg. of angle]

	Tokyo					Kagoshima			
Latitude, Longitude:	35.692	139.750				31.555	130.547		
YYYY/MM/DD HHh:	Sol.Decl.	Eq.Time	Hr.Angle	Altitude	Azimuth	Hr.Angle	Altitude	Azimuth
2086/03/31 01h:	4.12483	-1.05896	-161.2990	-46.5023	-152.3153	-170.5123	-53.1934	-164.0726
2086/03/31 02h:	4.14096	-1.05587	-146.2959	-39.1848	-134.4359	-155.5092	-47.3627	-142.3808
2086/03/31 03h:	4.15710	-1.05277	-131.2928	-29.4921	-120.5792	-140.5061	-38.1646	-126.2174
2086/03/31 04h:	4.17323	-1.04967	-116.2897	-18.4419	-109.5077	-125.5030	-27.0959	-114.2133
2086/03/31 05h:	4.18936	-1.04657	-101.2866	-6.6572	-100.0422	-110.4999	-15.0343	-104.6974
2086/03/31 06h:	4.20549	-1.04348	-86.2835	5.4677	-91.2580	-95.4968	-2.4662	-96.4646
2086/03/31 07h:	4.22162	-1.04038	-71.2804	17.6325	-82.3473	-80.4937	10.3044	-88.6334
2086/03/31 08h:	4.23774	-1.03728	-56.2773	29.5251	-72.4111	-65.4906	23.0296	-80.3966
2086/03/31 09h:	4.25387	-1.03419	-41.2742	40.6950	-60.1869	-50.4875	35.4149	-70.7394
2086/03/31 10h:	4.26999	-1.03110	-26.2711	50.3315	-43.7459	-35.4844	46.9629	-58.0160
2086/03/31 11h:	4.28610	-1.02800	-11.2680	56.9241	-20.9182	-20.4813	56.6314	-39.3741
2086/03/31 12h:	4.30222	-1.02491	3.7351	58.4284	7.1272	-5.4782	62.2655	-11.8040
2086/03/31 13h:	4.31833	-1.02182	18.7382	54.1864	33.1912	9.5248	61.3314	20.1175
2086/03/31 14h:	4.33444	-1.01873	33.7413	45.8545	52.6758	24.5279	54.3482	45.2518
2086/03/31 15h:	4.35055	-1.01564	48.7444	35.3319	66.7579	39.5310	44.0305	61.9745
2086/03/31 16h:	4.36666	-1.01255	63.7475	23.7437	77.6723	54.5341	32.1972	73.6739
2086/03/31 17h:	4.38276	-1.00946	78.7505	11.6871	86.9917	69.5372	19.6958	82.8432
2086/03/31 18h:	4.39886	-1.00637	93.7536	-0.4742	95.7609	84.5403	6.9485	90.9070
2086/03/31 19h:	4.41496	-1.00329	108.7567	-12.4446	104.8057	99.5434	-5.7723	98.7937
2086/03/31 20h:	4.43106	-1.00020	123.7598	-23.8882	114.9676	114.5465	-18.2109	107.3068
2086/03/31 21h:	4.44716	-0.99711	138.7629	-34.3136	127.2818	129.5496	-30.0254	117.3887
2086/03/31 22h:	4.46325	-0.99403	153.7660	-42.9194	143.0018	144.5526	-40.6445	130.3572
2086/03/31 23h:	4.47934	-0.99095	168.7691	-48.4771	162.9683	159.5557	-49.0389	147.9131
2086/03/31 24h:	4.49543	-0.98786	183.7721	-49.6640	-174.1844	174.5588	-53.5785	170.8384
2086/06/29 01h:	23.21697	-0.89536	-161.1354	-28.4511	-160.2469	-170.3487	-34.4542	-169.2308
2086/06/29 02h:	23.21473	-0.89745	-146.1374	-22.9494	-146.2132	-155.3508	-30.3647	-153.6258
2086/06/29 03h:	23.21248	-0.89953	-131.1395	-15.1410	-134.1897	-140.3529	-23.3758	-140.2945
2086/06/29 04h:	23.21022	-0.90162	-116.1416	-5.6820	-123.9910	-125.3550	-14.2960	-129.3283
2086/06/29 05h:	23.20794	-0.90370	-101.1437	4.9102	-115.1668	-110.3570	-3.7970	-120.2798
2086/06/29 06h:	23.20566	-0.90578	-86.1458	16.2620	-107.2083	-95.3591	7.6462	-112.5884
2086/06/29 07h:	23.20336	-0.90786	-71.1479	28.1028	-99.5781	-80.3612	19.7138	-105.7293
2086/06/29 08h:	23.20105	-0.90994	-56.1499	40.2145	-91.6013	-65.3633	32.1855	-99.1922
2086/06/29 09h:	23.19873	-0.91202	-41.1520	52.3655	-82.1162	-50.3654	44.8914	-92.3338
2086/06/29 10h:	23.19640	-0.91410	-26.1541	64.1447	-68.2857	-35.3674	57.6542	-83.9273
2086/06/29 11h:	23.19406	-0.91617	-11.1562	74.2009	-40.7839	-20.3695	70.1175	-70.1773
2086/06/29 12h:	23.19170	-0.91824	3.8418	77.0699	15.9759	-5.3716	80.3760	-30.9777
2086/06/29 13h:	23.18933	-0.92031	18.8397	69.4413	57.7014	9.6264	78.0505	47.9355
2086/06/29 14h:	23.18695	-0.92238	33.8376	58.1941	76.2151	24.6243	66.6527	75.1176
2086/06/29 15h:	23.18456	-0.92445	48.8355	46.1461	87.2551	39.6222	54.0366	86.5776
2086/06/29 16h:	23.18216	-0.92652	63.8335	33.9792	95.7635	54.6201	41.2658	94.3426
2086/06/29 17h:	23.17974	-0.92858	78.8314	21.9770	103.4594	69.6181	28.6076	101.0236
2086/06/29 18h:	23.17732	-0.93065	93.8294	10.3533	111.1847	84.6160	16.2285	107.5951
2086/06/29 19h:	23.17488	-0.93271	108.8273	-0.6537	119.5207	99.6140	4.3078	114.6374
2086/06/29 20h:	23.17243	-0.93477	123.8252	-10.7267	128.9858	114.6119	-6.9114	122.6564
2086/06/29 21h:	23.16997	-0.93683	138.8232	-19.4223	140.0736	129.6098	-17.0713	132.1911
2086/06/29 22h:	23.16750	-0.93888	153.8211	-26.1450	153.1390	144.6078	-25.6432	143.7976
2086/06/29 23h:	23.16501	-0.94094	168.8191	-30.2058	168.0954	159.6057	-31.9018	157.8282
2086/06/29 24h:	23.16251	-0.94299	183.8170	-31.0416	-175.9036	174.6037	-35.0391	173.9381
2086/09/27 01h:	-1.61061	2.21709	-158.0229	-50.2921	-144.1586	-167.2362	-57.7214	-155.5727
2086/09/27 02h:	-1.62681	2.22067	-143.0193	-41.6934	-126.3642	-152.2327	-50.2251	-133.2888
2086/09/27 03h:	-1.64300	2.22424	-128.0158	-31.1150	-113.0950	-137.2291	-39.8127	-117.9107
2086/09/27 04h:	-1.65920	2.22782	-113.0122	-19.5289	-102.5272	-122.2255	-27.9932	-106.7359
2086/09/27 05h:	-1.67539	2.23139	-98.0086	-7.4793	-93.3240	-107.2219	-15.5149	-97.7514
2086/09/27 06h:	-1.69159	2.23496	-83.0050	4.6836	-84.5312	-92.2184	-2.7753	-89.7185
2086/09/27 07h:	-1.70779	2.23853	-68.0015	16.6623	-75.3302	-77.2148	9.9562	-81.7605
2086/09/27 08h:	-1.72398	2.24210	-52.9979	28.1037	-64.8164	-62.2112	22.4176	-73.0501
2086/09/27 09h:	-1.74018	2.24567	-37.9943	38.4690	-51.8020	-47.2077	34.2452	-62.5370
2086/09/27 10h:	-1.75637	2.24923	-22.9908	46.8435	-34.8036	-32.2041	44.8023	-48.6551
2086/09/27 11h:	-1.77257	2.25279	-7.9872	51.8066	-12.9804	-17.2005	52.8873	-29.3315
2086/09/27 12h:	-1.78876	2.25636	7.0164	51.9574	11.4273	-2.1970	56.5910	-3.9904
2086/09/27 13h:	-1.80495	2.25992	22.0199	47.2430	33.5039	12.8066	54.4930	22.4239
2086/09/27 14h:	-1.82115	2.26348	37.0235	39.0217	50.7741	27.8101	47.4513	43.5972
2086/09/27 15h:	-1.83734	2.26703	52.0270	28.7377	63.9712	42.8137	37.4451	58.8226
2086/09/27 16h:	-1.85353	2.27059	67.0306	17.3335	74.5817	57.8173	25.8937	70.1042
2086/09/27 17h:	-1.86972	2.27414	82.0341	5.3633	83.8133	72.8208	13.5610	79.1922
2086/09/27 18h:	-1.88592	2.27769	97.0377	-6.8148	92.5803	87.8244	0.8658	87.2542
2086/09/27 19h:	-1.90211	2.28124	112.0412	-18.9053	101.6987	102.8279	-11.9150	95.1506
2086/09/27 20h:	-1.91830	2.28479	127.0448	-30.5690	112.1071	117.8315	-24.5278	103.7126
2086/09/27 21h:	-1.93449	2.28834	142.0483	-41.2845	125.1194	132.8350	-36.6336	114.0341
2086/09/27 22h:	-1.95068	2.29188	157.0519	-50.1189	142.5747	147.8385	-47.6271	127.8733
2086/09/27 23h:	-1.96687	2.29543	172.0554	-55.4860	165.8896	162.8421	-56.2733	147.9265
2086/09/27 24h:	-1.98306	2.29897	187.0590	-55.6679	-167.4222	177.8456	-60.3582	175.6434

付録

A　4つの計算方法の精度の比較

A.1　比較する計算方法の概説

我が国の環境工学分野では，古くから教科書にも取り上げられてきた，(a) 山崎の方法（文献[12], [13]）と (b) 赤坂の方法（文献[14]–[17]）がよく知られている。また，米国エネルギー省のEnergyPlus用の気象データ（EPW）の開発に用いられている (c) NRELの方法[21]もよく知られている。そこで，本書で説明した (d)「**松本の方法（2020年改）**」と合わせ，4つの方法を比較する。なお，赤坂の方法は，最近改訂されており，ここでは文献[17]に記載された最新の方法を採用する。

表 7　太陽位置計算の **4** つの方法の概要

方法の名称	山崎の方法 [13]	赤坂の方法（改） [17]
開発者 公開年	山崎　均 (1980)	赤坂　裕 (1992, Rev.2022)
基礎とする理論 時刻系	S. Newcomb (1906) [10] 暦 表 時 , ET	
基礎理論の簡易化	山崎　均	赤坂　裕
座標系	地心座標系	
座標変換	—	
ΔT の指定	N/A	
備考／特記事項	黄道傾斜角の長期 変化を考慮	山崎の方法(1980)を 簡易化

方法の名称	(DOE) **NREL** [21]	松本の方法（改） [1]
開発者 公開年	I. Reda & A. Andreas (2003, Rev.2008)	松本　真一 (Rev.2022)
基礎とする理論 時刻系	VSOP87 by P. Bretagnon & G. Francou (1988) [20] 地 球 時 , TT	
基礎理論の簡易化	J. Meeus [22]	海上保安庁海洋情報部[19]
座標系	黄道座標系	
座標変換	Meeusの方法による	松本の方法による
ΔT の指定	AA [25]のデータを引用?	数式で指定
備考／特記事項	ΔT の指定に関して 何の示唆もなし。	ΔT はNASA [24]の式。 1600年から2150年まで 適用可能。

それぞれの特徴を整理して，**表 7** にまとめて示す。教科書で有名な2つの方法は，古い Newcomb の理論に基づいており，後述の通り，現在の位置天文学における国際的共有事項から外れ，もはや教育的にも推奨できない。

A.1.1　Newcomb の計算理論に基づく 2 つの方法

Newcomb の計算理論[10]は，現在では国際天文連合の規格外の暦表時（Ephemeris Time, ET）に基づいており，また赤道座標系の基準となる春分点の定義も古いために推奨できないが，計算が容易で実用上精度が十分と言える場合もあるため，今なお使われている。4種類の中では，山崎の方法と赤坂の方法がこれに準拠している。ETは，現在位置天文学で用いられるTTと同等と言ってよいとされるが[22]，市民時UTとの差が吟味されていないため，将来に亘って適用できるか疑問である。

位置天文学分野の常識的な計算パラメータである地球時TTは，暦表時ET，力学時TDを経て，定着したものである。TTは，我々が日常使う市民時UTとは科学的に全く異なるものであるから，その差 ΔT を重要視することは当然であろう。

A.1.2　VSOP87 に基づく 2 つの方法

VSOP87は，数千項にも及ぶ余弦関数を用いて黄道座標系で惑星位置を表現する高度で複雑な方法であるが，そのプログラムが無料公開されているためか広く使われている。位置天文学分野ではデファクト・スタンダードと言える[20]。しかし，あまりにも計算量が多いため，やや簡単に整理したものも用いられている。その代表格がMeeusによる簡略化であり[22]，NRELの方法として全面的に採用されている[21]。NRELの方法は精度が高いとされるが，(i) 実は ΔT をどう

設定するかが明示されておらず，ブラックボックス的な安直な使い方によって精度が落ちるリスクが高い。(ii) また，我々が太陽位置計算で用いたい赤道座標系への変換も厳密過ぎて，煩雑・難解とも言える。計算プログラムは無料で入手可能であるが，ソース・コードを精読した上で利用すべきである [注11]。

我が国でも簡略化の検討がなされた模様で，「松本の方法」や「松本の方法（2022 年改）」が準拠する海上保安庁海洋情報部（研究開発当時，水路部）の式[19]は，その代表的なものである。

A.2 視赤緯 δ と均時差 T_e の計算精度の比較

A.2.1 比較の方法

前章でレビューした 4 種類の計算方法が，批判した通りの性能を持つかどうかを検討する。おそらく厳密に VSOP87 に準拠した計算結果を公開しているであろう，理科年表の視赤緯 δ[°] と均時差 T_e[min.] を真の値と考え，計算誤差を XBE（絶対値最大誤差），MBE（平均誤差），RMSE（平均二乗誤差）の 3 つの指標で評価する。

真値とした元データは，1981 年から 2020 年までの日別値（$N = 14,610$）である[18]。元データは自前で調整して用いた [注12]。また計算は，筆者が作成した実行形式プログラムを使って行った。なお，NREL の提供するソース・コードは参照にするに留め，ほぼ自前でコーディングしたが，計算結果を文献[21]で検証してある。

A.2.2 計算結果の比較と考察

計算結果を分析した結果を表 8 と図 6 に示す。表によれば，Newcomb の計算理論に基づく山崎や赤坂の方法よりも，VSOP87 に基づく，松本や NREL の方法の精度が高いことが分かる。また，NREL の方法に比べて，相対的にかなり簡易な数式表現に基づき，計算プログラム上の演算ステップ数も少ない松本の方法（2022 年改）は，NREL の方法に引けを取らない精度を持つ方法と言えよう。

このようにして松本の方法（2022 年改）の精度が確認され，2086 年という遠い将来の気象データにも対応できるものとして実装が進められたのである。

表 8 4 つの計算方法の計算精度の分析

(a) 太陽視赤緯 δ [″(arcsec)] の誤差

指標 ∗	山崎の方法	赤坂の方法	**NREL** の方法	松本の方法（改）
XBE	19.0	18.0	2.0	3.0
MBE	−0.2	−0.7	0.0	0.0
RMSE	7.0	6.5	0.7	0.9

(b) 均時差 T_e [s] の誤差

指標 ∗	山崎の方法	赤坂の方法	**NREL** の方法	松本の方法（改）
XBE	2.50	3.00	0.40	0.40
MBE	−0.35	−0.35	0.19	0.07
RMSE	0.74	0.83	0.19	0.15

∗: (Error)=(Estimated Value)−(True Value)

[注11] VSOP87 のソース・コードにおいてすら，桁落ちを防ぐ技法が用いられておらず，精度よく計算するためには，利用する側の工夫が必要である。例えば，式 (9) の最下段の Müller の多項式表記などは，古典的だが大切な事項である。

[注12] 1 ヶ所の誤植を内挿によって修正した。また，視赤緯は arcsec 単位の整数，均時差は dsec 単位の整数として整備した。なお，1981 年から 1984 年までは 0UT ではなく，0ET の値が示されているが，これは計算側で出力を調整した。

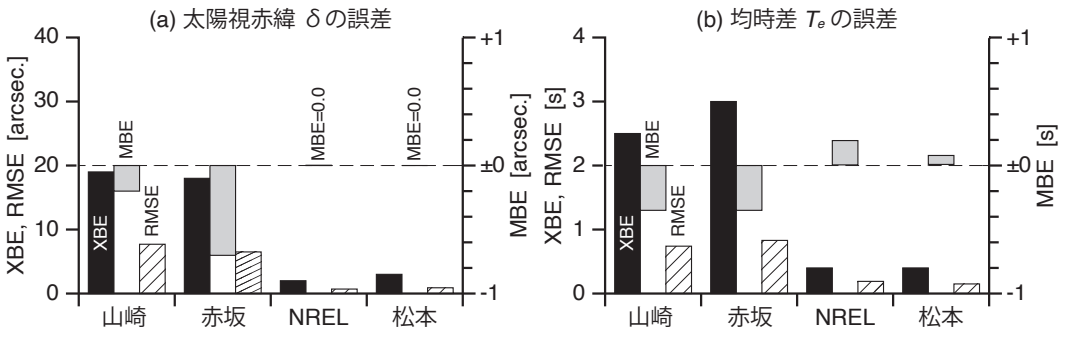

図 6　4 つの計算方法の計算精度の比較

B　ユリウス通日 $\overline{\mathrm{JD}}$ のプログラム・ソース・コード

Meeus の方法[22]によるコード（FORTRAN90 と C/C++）を示す。

B.1　FORTRAN

```
! ------------------------------------------------
!   Calculation of the Julian Day as long integer
!   "Meeus Method"  (ユリウス通日計算関数, Meeus)
!     YYYY/MM/DD (0 < YYYY, 1 <= MM <= 12,
!                 1 <= DD <= 28, 29, 30, 31)
!   Note:
!     This function does not check the ranges of
!     args.: YYYY, MM, DD
!   Ref.
!     J. Meeus: Astronomical Algorithms (2nd Ed.),
!       pp.59-66, Willmann-Bell, Richmond, 1998.
! --+----*----+----*----+----*----+----*----+----*
function JD(YYYY,MM,DD)
  implicit none
  integer   YYYY,MM,DD
  integer*8 JD, GREG0, jul, GREG1
  integer   ja, jm, jy
!
! Gregorian Calendar adopted in Oct. 15, 1582.
  GREG0 = GREG(1582,10,15)
!
  jy = YYYY
  GREG1 = GREG(YYYY,MM,DD)
  if ( 0 > jy ) then
    jy = jy + 1
  end if
  if ( 2 < MM ) then
    jm = MM + 1
  else
    jy = jy - 1
    jm = MM + 13
  end if
  jul = floor(365.25*jy) + floor(30.6001*jm) &
      + DD + 1720995
  if ( GREG0 <= GREG1 ) then
    ja  = int(0.01*jy)
    jul = jul + 2 - ja + int(0.25*ja);
  end if
  JD = jul
!
  contains
    function GREG(Y,M,D)
      integer   Y, M, D
      integer*8 GREG
      GREG = D+31*(M+12*Y)
    end function ! GREG
end function
```

B.2　C/C++

```
// ------------------------------------------------
//      Reference day number macro for comparison
#define LGREG(Y,M,D) ((D)+31L*((M)+12L*(Y)))
#define GREG0 LGREG(1582,10,15)
//      Gregorian Calendar's adopted date in 16C
typedef unsigned int   uint;
typedef unsigned long ulong;
// ------------------------------------------------
ulong JD0( uint YYYY, uint MM, uint DD )
//  Calculation of the Julian Day as long integer
//  "Meeus Method"
//   (ユリウス通日を求める関数, Meeus による)
//    YYYY/MM/DD (0 < YYYY, 1 <= MM <= 12,
//                1 <= DD <= 28, 29, 30, 31)
//  Note:
//    This function does not check the ranges of
//    args.: YYYY, MM, DD
//  Ref.
//    J. Meeus: Astronomical Algorithms (2nd Ed.),
//      pp.59-66, Willmann-Bell, Richmond, 1998.
{
  int   ja, jm, jy = int(YYYY);
  long  jul;
  long GREG1 =
    LGREG( long(YYYY), long(MM), long(DD) );
  if ( 0 > jy ) ++jy;
  if ( 2 < int(MM) )  jm = int(MM) + 1;
  else {
    --jy; jm = int(MM) + 13;
  }
  jul = long(floor(365.25*jy)
           + floor(30.6001*jm) + aDate.DD)
      + 1720995L;
  if ( GREG0 <= GREG1 ) {
    ja   = int(0.01*jy);
    jul += 2 - ja + int(0.25*ja);
  }
  return ( ulong(jul) );
}
```

B.3 参考値

プログラム検証のためのデータを示す。なお，$*$ 印のあるものは $\overline{\mathrm{JD}}$（整数の通日）ではなく，0^{h}UT 時の JD あるいは，0^{h}TT 時の JDE などと解釈すべき実数である。

表 **9** ユリウス日の例

2000	Jan.	1	2 451 545	1600	Dec.	31.0$*$	2 305 812.5
1999	Jan.	1.0$*$	2 451 179.5	837	Apr.	10.3$*$	2 026 871.8
1987	Jan.	27.0$*$	2 446 822.5	-123	Dec.	31.0$*$	1 676 496.5
1987	June	19	2 446 966	-122	Jan.	1.0$*$	1 676 497.5
1988	Jan.	27.0$*$	2 447 187.5	-1000	July	12	1 356 001
1988	June	19	2 447 332	-1000	Feb.	29.0$*$	1 355 866.5
1900	Jan.	1.0$*$	2 415 020.5	-1001	Aug.	17.9$*$	1 355 671.4
1600	Jan.	1.0$*$	2 305 447.5	-4712	Jan.	1	0

$*$: With consideration of time UT or TT

C 古典的な視赤緯 δ と均時差 E_t のプログラム・ソース・コード

山崎の式（文献[12], [13]）と赤坂の式（文献[**?**]〜[16]）によるコード（FORTRAN90）を示す。なお，C/C++によるコードは，時刻の取り扱いがさらに精密なものであるが，付録 D に含まれている。なお，前の付録 A においては，C/C++のコードを使用した。

C.1 山崎の式に対する FORTRAN コード

```
! ------------------------------------------------
!   Calculation of the Solar Declination
!   and the Equation of Time "Yamazaki Method"
!    (赤緯と均時差の計算サブルーチン，山崎による)
!   (output)
!    DELTA: Solar Declination [rad.]
!       ET: Equation of Time [rad.]
!   (input)
!       YY: Year, MM: Month, DD: Day
!       TT: Hour in ZST
!   LONGIT: Reference Longitude for ZST in [deg.]
!   Note:
!     This subroutine does not check the ranges
!     of the inputarguments.
!   Ref.
!     Hitoshi Yamazaki: Fundamental Formulae for
!       Daylighting   IV (Precise Program to
!       Calculate Solar Declination   and Equation
!       of Time) (in Japanese), Proc. of AIJ
!       Annual Meeting 1980, pp.407--408, 1980.9.
!     山崎　均：日照環境の基礎計算式 IV (対象
!       地域の太陽視赤緯及び均時差を正確に計算
!       するプログラム)，日本建築学会大会学術
!       講演梗概集，計画系，pp.407--408，1980.9.
! --+----*----+----*----+----*----+----*----+----*
subroutine SUN(DELTA,ET,YY,MM,DD,TT,LONGIT)
  implicit none
  integer YY, MM, DD
  real DELTA, ET, LONGIT, TT
  real M, dM, A, D, D2, D3, ET1, ET2
  real t1, t2, t3, DELTA0, E, E0, EPS
  real SD, CD, V, VEPS2, YN
  real SinM, Sin2M, Sin3M
  integer M1,i
  integer :: MONTH(12) = &
    (/ 31,28,31,30,31,30,31,31,30,31,30,31 /)
  real :: RAD  = 1.7453292e-2
  real :: DEL0 = 23.4522
!
  YN = float(YY) - 1900.0 ! Elasped yr after 1900
  D2 = aint((YN - 1.)/4.)
  D3 = aint(YN/4.)
  MONTH(2) = 28 + ifix(D3 - D2)
  D = float(DD) + (YN - 30.) * 1.1574e-5
  D = D + (TT - 12.) / 24. - LONGIT / 15. / 24.
  M1 = MM - 1
  if ( M1 >= 1 ) then
    do i=1, M1
      D = D + float(MONTH(i))
    end do
  end if
!
! Calculation of Julian Century from JD1900.0
  t1 = (365. * YN + D2 + D) / 36525.
  t2 = t1 * t1
  t3 = t2 * t1
!
! Calculation of the Zodiac Tilt Angle DELTA0
!  黄道傾斜角
  DELTA0 = -9.44e-5 + 1.30125e-2 * t1 &
         + 1.64e-6 * t2 + 5.0e-7 * t3
  DELTA0 = DELTA0 - 23.4522
  DELTA0 = DELTA0 * RAD
! Altanative DELTA0 for year 2000
  DELTA0 = -DEL0 * RAD
!
! Calculation of Eccentricity (E) 離心率
  E = 1.04e-6 - 4.18e-5 * t1 - 1.26e-7 * t2 &
    + 0.01675
!
! Angle (rad.) bwetween Perihelion and
! Winter Solstice (EPS)
  EPS = 0.719175 * t1 + 0.000453 * t2
  EPS = EPS + 11.220833 + t1
  EPS = EPS * RAD
!
```

```
! Calculation of Mean Perigee Elongation (M)
! 平均近点離角
  M = 6.00267e-4 * (D2 + D) &
    - 9.02579e-4 * YN &
    - 0.00015 * t2 - 1.667e-4
  M = M - 1.524 - 0.255 * YN + 0.985 * D2
  M = M + 0.985 * D
  M = M * RAD
  SinM   = sin(M)
  Sin2M  = sin(2.0 * M)
  Sin3M  = sin(3.0 * M)
!
  dM = 9.93502e-5 * &
    (1.0 - E * (cos(M) - 2.0 * E * SinM * SinM))
  M = M - dM
!
  V = (2.0 - 0.25 * E * E) * E * SinM
  V = V + 1.25 * E * E * Sin2M
  V = V + 13.0 / 12.0 * E * E * E * Sin3M
  v = V + M
!
  SD = cos(EPS + V) * sin( DELTA0 )
  CD = sqrt(1.0 - SD * SD)
  VEPS2 = 2.0 * (EPS + V)
  A = (1.0 - cos(DELTA0)) / (1.0 + cos(DELTA0))
!
! Solar Declination 視赤緯
  DELTA = atan( SD / CD )
!
! Equation of Time 均時差
  ET1 = M - V;
  ET2 = -atan( (A * sin(VEPS2)) &
             / (1.0 - A * cos(VEPS2)) )
  ET = ET1 + ET2;
!
  return
end subroutine
```

C.2 赤坂 (2022) の式に対する FORTRAN コード

```
! ------------------------------------------------
!  Calculation of the Solar Declination
!  and the Equation of Time "Akasaka(2022) Method"
!   (赤緯と均時差を求めるサブルーチン，赤坂 (2022)
!    による)
!  (output)
!    SINDLT: Sine of Solar Declination [-]
!    COSDLT: Cosine of Solar Declination [-]
!        ET: Equation of Time [deg.]
!  (input)
!    YEAR: Year
!    NDAY: Serial day number of the YEAR
!  Note:
!    This subroutine does not check the ranges
!    of the input arguments.
!  Ref.
!    H. Akasaka: Simplified Calculation Method of
!     the Sun Position with Secular Changes,
!     General Technical Report on the EA Weather
!     Data. MetDS HP. 2022.8
!    赤坂　裕：年差を考慮した太陽位置の簡易計算，
!      EA 気象データ技術解説一般，MetDS HP, 2022.8.
!
! --+----*----+----*----+----*----+----*----+----*
subroutine SUNLD(YEAR,NDAY,SINDLT,COSDLT,ET)
  implicit none
  integer YEAR, NDAY
  real SINDLT, COSDLT, ET
  real D0, M, M1, N
  real V, RAD, DLT0, EPS, VEPS, VE2
!
  RAD  = 3.141592 / 180.
!
  N = float(YEAR) - 1968.
!
  DLT0 = (-23.4393 + 0.00013 * &
         (FLOAT(YEAR) - 2000.)) * RAD
  D0 = 3.71 + 0.2596 * N - int((N + 3.) / 4.)
!
  M = 0.9856 * (NDAY - D0)
  EPS = 12.3901 + 0.0172 * (N + M / 360.)
  V = M + 1.914 * SIN(M * RAD) &
        + 0.02 * SIN(2.* M * RAD)
  VEPS = (V+EPS) * RAD
  VE2  = 2. * VEPS
!
  ET  = (M - V) &
      - ATAN(0.043 * SIN(VE2) &
      / (1.-0.043 * COS(VE2))) / RAD
!
      SINDLT = COS(VEPS) * SIN(DLT0)
      COSDLT = SQRT(ABS(1. - SINDLT * SINDLT))
!
  return
end subroutine
```

D 松本の式（2022 年改）による視赤緯 δ と均時差 E_t のプログラム・ソース・コード

D.1 FORTRAN コード

このソース・コードは D.2 節に示した C/C++のコードを翻訳したものである。FORTRAN のコードとしては，やや C/C++に由来する構造化に偏ったもので，非効率で生産性が低く，なおかつ危険なものかも知れないが，読者諸賢の多少の参考にはなるであろう。

```
! ------------------------------------------------
!   SolPos.f90
!     Calculation Subroutines and functions
!     for Sun Position Parameters:
!       Solar Declination 太陽赤緯 [deg.] and
!       Equation of Time  均時差    [deg.]
!
!   Coded by Shin-ichi Matsumoto, 2022
!
!   Proofed by gfortran
!
! (c) 2022, Shin-ichi Matsumoto, Akita Pref. Univ.
! --+----*----+----*----+----*----+----*----+----*
module mGrobal
  implicit none
  integer, parameter :: YEAR_GREG = 1582
!             Gregorian calendar adopted in 1582
  integer*8, parameter ::  JD2000 = 2451545
!             Julian day of Jan. 1 12TT, 2000
  double precision, parameter :: pi=3.14159265d0
  integer, parameter :: DAYS_OF_MONTH(12) = &
    (/ 31,28,31,30,31,30,31,31,30,31,30,31 /)
!
  logical, parameter :: T_ = .true.
  logical, parameter :: F_ = .false.
!
  type TC
    logical :: T
    double precision :: A, B, C
  end type TC
```

```
!
  type TJday
    integer*8 :: Value
  end type TJday
!
  type TTday
    integer :: Value
  end type TTday
!
  type TIntDate
    integer :: Year, Month, Day
  end type TIntDate
!
  integer :: C_MAX = 18
  type(TC), save :: CLC(18) = (/ &
  TC(T_, 36000.7695d0, 0.0000000d0, 0.0000000d0),&
  TC(F_, 280.465900d0, 0.0000000d0, 0.0000000d0),&
  TC(F_, 1.91470000d0, 35999.050d0, 267.52000d0),&
  TC(F_, 0.02000000d0, 71998.100d0, 265.10000d0),&
  TC(T_,-0.00480000d0, 35999.000d0, 268.00000d0),&
  TC(F_, 0.00200000d0, 32964.000d0, 158.00000d0),&
  TC(F_, 0.00180000d0, 19.000000d0, 159.00000d0),&
  TC(F_, 0.00180000d0, 445267.00d0, 208.00000d0),&
  TC(F_, 0.00150000d0, 45038.000d0, 254.00000d0),&
  TC(F_, 0.00130000d0, 22519.000d0, 352.00000d0),&
  TC(F_, 0.00070000d0, 65929.000d0, 45.000000d0),&
  TC(F_, 0.00070000d0, 3035.0000d0, 110.00000d0),&
  TC(F_, 0.00070000d0, 9038.0000d0, 64.000000d0),&
  TC(F_, 0.00060000d0, 33718.000d0, 316.00000d0),&
  TC(F_, 0.00050000d0, 155.00000d0, 118.00000d0),&
  TC(F_, 0.00050000d0, 2281.0000d0, 221.00000d0),&
  TC(F_, 0.00040000d0, 29930.000d0, 48.000000d0),&
  TC(F_, 0.00040000d0, 31557.000d0, 161.00000d0) &
  /)
!
  integer, save :: D_MAX = 9
  type(TC), save :: SDC(9) = (/ &
  TC(F_, 1.00014000d0, 0.0000000d0, 0.0000000d0),&
  TC(F_, 0.01670600d0, 35999.050d0, 177.53000d0),&
  TC(F_, 0.00013900d0, 71998.000d0, 175.00000d0),&
  TC(T_,-0.00004200d0, 35999.000d0, 178.00000d0),&
  TC(F_, 0.00003100d0, 445267.00d0, 298.00000d0),&
  TC(F_, 0.00001600d0, 32964.000d0, 68.000000d0),&
  TC(F_, 0.00001600d0, 45038.000d0, 164.00000d0),&
  TC(F_, 0.00000500d0, 22519.000d0, 233.00000d0),&
  TC(F_, 0.00000500d0, 33718.000d0, 226.00000d0) &
  /)
!
contains
  function DegToRad(D)
    implicit none
    double precision D, DegToRad
    DegToRad = D * (pi / 180.d0)
  end function DegToRad
!
  function RadToDeg(R)
    implicit none
    double precision R, RadToDeg
    RadToDeg = R * (180.d0 / pi)
  end function RadToDeg
!
  function JC2000(JDE)
    implicit none
    integer*8 JDE
    double precision JC2000
    JC2000 = dble(JDE - JD2000) / 36525.d0
  end function JC2000
end module mGrobal
! ------------------------------------------------
! Calendar functions and subroutines
function Is_Leap(YY)
  use mGrobal
  implicit none
  logical Is_Leap, Ret
  integer YY
!
  if ( YY <= YEAR_GREG ) then
    Ret = .false.
  else
    if ( 0 == mod(YY,400) ) then
      Ret = .true.
    else
      if ( 0 == mod(YY,100) ) then
        Ret = .false.
      else
        if ( 0 == mod(YY,4) ) then
          Ret = .true.
        else
          Ret = .false.
        end if
      end if
    end if
  end if
!
  Is_Leap = Ret
!
end function
! ------------------------------------------------
subroutine CorrectDate(YY,MM,DD,Done)
  use mGrobal
  implicit none
  integer YY, MM, DD
  integer m, DOM(12)
  logical Done, chk, leap, Is_Leap
!
  chk = .true.
  if ( 0 == YY ) then
    YY =  1; chk = .false.
  end if
  if (  1 > MM ) then
    MM =  1; chk = .false.
  end if
  if ( 12 < MM ) then
    MM = 12; chk = .false.
  end if
  if (  1 > DD ) then
    DD =  1; chk = .false.
  end if
  leap = Is_Leap( YY );
  do m=1, 12
    DOM(m) = DAYS_OF_MONTH(m)
  end do
  if ( leap ) then
    DOM(2) = DOM(2) + 1
  end if
  if ( DD > DOM(MM) ) then
    DD = DOM(MM); chk = .false.
  end if
!
  Done = chk
  return
end subroutine
! ------------------------------------------------
subroutine TdayToIntDate(TDay,YY,IDate)
  use mGrobal
  implicit none
  type(TTday) :: TDay, t_day
  type(TIntDate) :: IDate
  integer YY
  integer m, day
  integer days(12)
  logical leap, Is_Leap
!
  leap = Is_Leap(YY);
  day = 0
  do m=1, 12
    day = day + DAYS_OF_MONTH(m);
    if ( leap .and. 2 == m ) then
      day = day + 1
    end if
    days(m) = day
  end do
  t_day%Value = TDay%Value
  if ( 1 > TDay%Value ) then
    t_day%Value = 1
  else
    if ( 365 < TDay%Value ) then
      if ( leap ) then
        t_day%Value = 366
      else
        t_day%Value = 365
      endif
    end if
  end if
!
  IDate%Year = YY
  do m=1, 12
    if ( t_day%Value <= days(m) ) then
      IDate%Month = m
      exit
    end if
  end do
  IDate%Day = t_day%Value
```

```
  if ( 1 /= IDate%Month ) then
    IDate%Day = IDate%Day &
              - days(IDate%Month - 1)
  end if
!
  return
end subroutine
! ------------------------------------------------
subroutine IntDateToTday(IDate, TDay)
  use mGrobal
  implicit none
  type(TIntDate) :: IDate
  type(TTday) ::    TDay, days
  integer m, DOM(12)
  logical leap, Is_Leap
!
  if ( 1 == IDate%month ) then
    TDay%Value = IDate%Day
  else
    leap = Is_Leap(IDate%year)
    do m=1, 12
      DOM(m) = DAYS_OF_MONTH(m)
    end do
    if ( leap ) then
      DOM(2) = DOM(2) + 1
    end if
    days%Value = 0
    do m=1, IDate%Month - 1
      days%Value = days%Value + DOM(m)
    end do
    TDay%Value = IDate%Day + days%Value
  end if
!
  return
end subroutine
! vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
!   This part is a little bit different
!   from code described above in the
!   technical report
! ------------------------------------------------
!   Calculation of the Julian Day as long integer
!   "Meeus Method" （ユリウス通日計算関数, Meeus）
!   using TIntDate argument
!   SUBROUTINE STYLE
!   Note:
!     This function does not check the ranges of
!     args.: YYYY, MM, DD
!   Ref.
!     J. Meeus: Astronomical Algorithms (2nd Ed.),
!       pp.59-66, Willmann-Bell, Richmond, 1998.
! --+----*----+----*----+----*----+----*----+----*
subroutine IntDateToJDay(IDate,JDay)
  use mGrobal
  implicit none
  type(TIntDate) :: IDate
  type(TJday) ::    JDay
  integer   YY,MM,DD
  integer*8 JD, GREG0, jul, GREG1
  integer   ja, jm, jy
!
! Gregorian Calendar adopted in Oct. 15, 1582.
  GREG0 = GREG(1582,10,15)
!
  jy = IDate%Year
  YY = IDate%Year
  MM = IDate%Month
  DD = IDate%Day
  GREG1 = GREG( YY, MM, DD )
  if ( 0 > jy ) then
    jy = jy + 1
  end if
  if ( 2 < MM ) then
    jm = MM + 1
  else
    jy = jy - 1
    jm = MM + 13
  end if
  jul = floor(365.25d0*jy) + floor(30.6001d0*jm) &
      + DD + 1720995
  if ( GREG0 <= GREG1 ) then
    ja  = int(0.01d0*jy)
    jul = jul + 2 - ja + int(0.25d0*ja)
  end if
  JDay%Value = jul
!
  contains
    function GREG(Y,M,D)
      integer   Y, M, D
      integer*8 GREG
      GREG = D+31*(M+12*Y)
    end function ! GREG
end subroutine
! ------------------------------------------------
!   Calculation of the Julian Day as long integer
!   "Meeus Method" （ユリウス通日計算関数, Meeus）
!   using TIntDate argument
!   FUNCTION STYLE
!   Note:
!     This function does not check the ranges of
!     args.: YYYY, MM, DD
!   Ref.
!     J. Meeus: Astronomical Algorithms (2nd Ed.),
!       pp.59-66, Willmann-Bell, Richmond, 1998.
! --+----*----+----*----+----*----+----*----+----*
function JD_of( IDate )
  use mGrobal
  implicit none
  type(TIntDate) :: IDate
  type(TJday) ::    JDay
  integer   ja, jm, jy
  integer*8 jul, GREG0, JD_of
!
  call IntDateToJDay( IDate, JDay )
!
  JD_of = JDay%Value
!
  return
end function
! ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
! MAIN REVISION 2022   !
! .............................................
! Time Difference dT in [msec.] integer
!  "NASA Method"
!   YY: year (1600-2150)
! Ref.
!   Fred~Espenak and Jean~Meeus: Five
!     Millennium Canon of Solar Eclipses:
!     -1999 to +3000, The NASA Technical
!     Publication (NASA/TP--2006--214141),
!     NASA, Greenbelt, 2006.10.
! --+----*----+----*----+----*----+----*----+----*
function DeltaTime(YY) ! in [msec.]
  use mGrobal
  implicit none
  type(TIntDate) :: UTDate
  integer  YY, DeltaTime, dt
  UTDate%Year  = YY
  UTDate%Month = 7
  UTDate%Day   = 1
!
  DeltaTime = DeltaTimeEx(UTDate)
!
  contains
    function RoundTo(V)
      implicit none
      double precision V
      integer IV, RoundTo
      logical :: minus = .false.
!
      if ( 0.d0 > V ) minus = .true.
      IV = nint(abs(V) * 1000.d0)
      if ( minus ) then
        IV = -IV
      end if
      RoundTo = IV
!
      return
    end function
!
    function DeltaTimeEx(IDate)
      use mGrobal
      implicit none
      type(TIntDate) :: IDate
      double precision y, t, S
      integer DeltaTimeEx
      integer YY, MM, DD
!
      YY = IDate%Year
      MM = IDate%Month
      DD = IDate%Day
      y = dble(YY) + (dble(MM) - 0.5) / 12.0
      if ( 7 == MM .and. 1 == DD ) then
        y = dble(YY) + 0.5d0
```

```
      end if
      if ( 1 == MM .and. 1 == DD ) then
        y = dble(YY)
      end if
!
      if ( 1600 > YY ) then
        DeltaTimeEx = 0
        return
      end if
      if ( 1701 > YY ) then
        t = y - 1600d0
        S =  ((1. / 7129d0 * t  - 0.01532d0) &
          * t - 0.9808d0) * t + 120d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1801 > YY ) then
        t = y - 1700d0
        S = (((-1d0 / 1174000d0 * t &
          + 0.00013336d0) * t - 0.0059285d0) &
          * t  + 0.1603d0) * t + 8.83d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1861 > YY ) then
        t = y - 1800d0
        S = ((((((0.000000000875d0 * t &
          - 0.0000001699d0) * t + 0.0000121272d0) &
          * t - 0.00037436d0) * t + 0.0041116d0) &
          * t + 0.0068612d0) * t - 0.332447d0) &
          * t + 13.72d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1901 > YY ) then
        t = y - 1860d0
        S =  ((((1d0 / 233174d0 * t &
          - 0.0004473624d0) * t + 0.01680668d0) &
          * t - 0.251754d0) * t + 0.5737d0) &
          * t + 7.62d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1921 > YY ) then
        t = y - 1900d0
        S = (((-0.000197d0 * t + 0.0061966d0) &
          * t - 0.0598939d0) * t + 1.494119d0) &
          * t - 2.79d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1941 > YY ) then
        t = y - 1920d0
        S = ((0.0020936d0 * t - 0.076100d0) &
          * t + 0.84493d0) * t + 21.20d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1961 > YY ) then
        t = y - 1950d0
        S = ((1d0 / 2547d0 * t  - 1d0/233d0) &
          * t + 0.407d0) * t + 29.07d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 1986 > YY ) then
        t = y - 1975d0
        S = ((-1d0/718d0 * t  - 1d0/260d0) &
          * t  + 1.067d0) * t + 45.45d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 2006 > YY ) then
        t = y - 2000d0
        S = ((((0.00002373599d0 * t &
          + 0.000651814d0) * t + 0.0017275d0) &
          * t - 0.060374d0) &
          * t + 0.3345d0) * t + 63.86d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 2051 > YY ) then
        t = y - 2000d0
        S = (0.005589d0 * t + 0.32217d0) &
          * t + 62.92d0
        DeltaTimeEx = RoundTo(S)
        return
      end if
      if ( 2151 > YY ) then
        t = y - 1820d0
        S = (0.0032d0 * t + 0.5628d0) &
          * t - 205.724d0
      else
        S = 328.48d0
      end if
      DeltaTimeEx = RoundTo(S)
      return
    end function DeltaTimeEx
end function DeltaTime
! ..............................................
! ----------------------------------------------
! Calculation of the obliquity of the ecliptic
! for a given Julian century JC based on J2000.0
! (黄道傾角)
!   Note:
!     On consideration of the nutation, IsPrec
!     is a switch.
!       = true (default): with consideration
!       = false: without consideration
!     Return value is in deg.
!     Hereafter, abbr. JC2000 for a Julian century
!     based on the J2000.0.
! --+----*----+----*----+----*----+----*----+----*
function Obliquity( JC, IsPrec )
  use mGrobal
  implicit none
  double precision JC, Obliquity, Eps, T1, T2
  logical IsPrec
!
  Eps = (( -5.0361111d-7  * JC &
         + 1.6388889d-7) * JC &
         + 1.3004167d-2) * JC - 2.343929d+1
  if ( IsPrec ) then
    T1 = DegToRad( &
         dmod( 1934d0 * JC + 235d0, 360d0) )
    T2 = DegToRad( &
         dmod(72002d0 * JC + 201d0, 360d0) )
    Eps = Eps - 0.00256d0 * cos( T1 ) &
              - 0.00015d0 * cos( T2 )
  end if
  Obliquity = Eps
!
  return
end function
! ----------------------------------------------
! Calculation of the solar celestial longitude
! for a given JC2000 (視黄経または平均黄経)
!   Note:
!    If IsApp = .false., then the mean value is
!    returned. Return value is in deg.
! --+----*----+----*----+----*----+----*----+----*
function CLongit( JC, IsApp )
  use mGrobal
  implicit none
  double precision    JC, CLongit
  logical IsApp
  double precision    cl, A, BT_C
  integer i
!
  cl = 0.0d0
  do i=C_MAX, 3, -1
    ! DON'T CHANGE the loop INDEX ORDER
    ! to avoid information loss!
    if ( CLC(i)%T ) then
      A = CLC(i)%A * JC
    else
      A = CLC(i)%A
    end if
    BT_C = CLC(i)%B * JC + CLC(i)%C
    BT_C = DegToRad( dmod(BT_C, 360d0) )
    cl = cl + A * cos( BT_C );
  end do
  cl = cl + CLC(2)%A + CLC(1)%A * JC
  if ( IsApp ) then
    A    = 0.0048d0
    BT_C = 1934d0 * JC + 145d0
    BT_C = DegToRad( dmod(BT_C, 360d0) )
    cl = cl + A * cos( BT_C )
    A    = -0.0004d0
    BT_C = 72002d0 * JC + 111d0
    BT_C = DegToRad( dmod(BT_C, 360d0) )
    cl = cl + A * cos( BT_C ) - 0.0057d0
  end if
  cl = dmod(cl, 360d0);
  if ( 0.0 <= cl ) then
    CLongit = cl
  else
```

```
    CLongit = cl + 360d0
  end if
!
  return
end function
! ------------------------------------------------
! Calculation of the mean solar right ascension
! for a given JC2000 (UT). (平均太陽の赤経)
!   Note:
!     Julian century UTJC should be calculated
!     with UTC. Return value is in [h].
! --+----*----+----*----+----*----+----*----+----*
function MAscens( UTJC )
  use mGrobal
  implicit none
  double precision UTJC, MAscens, Sec, Hour
!
  Sec = ((-0.0000062d0*UTJC + 0.093104d0)*UTJC + &
         8640184.812866d0)*UTJC + 67310.54841d0
  Hour = Sec / 3600d0
  Hour = dmod(Hour, 24d0)
  if ( 0.0 <= Hour ) then
    MAscens = Hour
  else
    MAscens = Hour + 24d0
  end if
!
  return
end function
! ------------------------------------------------
! Calculation of the equation of equinoxes for a
! for a given JC2000 (JC). (分点差)
!   Note:
!     Return value is in [h]. Based on the eq.:
!       dL * cos(eps) / 15.
!       where dL is the nutation of the
!       celestial longitude[deg.] and eps is the
!       apparent obliquity of the ecliptic.
! --+----*----+----*----+----*----+----*----+----*
function Eq( JC )
  use mGrobal
  implicit none
  double precision JC, Eq, X, Obliquity
!
  X = 0.0048 * sin( DegToRad( &
      dmod(1934d0 * JC + 235d0, 360d0) ) ) &
    - 0.0004 * sin( DegToRad( &
      dmod(72002d0 * JC + 201d0, 360d0) ) )
  X = X * cos( DegToRad(Obliquity(JC,.true.)) )
  Eq = X / 15d0
!
  return
end function
! ------------------------------------------------
! Calculation of the geocentric distance to
! the sun for a given JC2000 (JC). (地心距離)
!   Note:
!     Return value is in AU.
! --+----*----+----*----+----*----+----*----+----*
function SolarDistance( JC )
  use mGrobal
  implicit none
  double precision JC, SolarDistance, sd, A, BT_C
  integer i
!
  sd = 0.0d0
  do i=D_MAX, 2, -1
    ! DON'T CHANGE the loop INDEX ORDER
    ! to avoid information loss!
    if ( SDC(i)%T ) then
      A = SDC(i)%A * JC
    else
      A = SDC(i)%A
    end if
    BT_C = SDC(i)%B * JC + SDC(i)%C
    BT_C = DegToRad( dmod(BT_C, 360d0) )
    sd = sd + A * cos( BT_C );
  end do
  sd = sd + SDC(1)%A
  SolarDistance = sd
!
  return
end function
! ------------------------------------------------
! Calculation of the solar declination and the
! equation of time (solar parameters)
! "Matsumoto Method (Rev. 2022)" (視赤緯と均時差)
! SUBROUTINE STYLE
! (input)
!  YYYY: a given year (1600-2150)
!  SDAY: serial day number of YYYY (1-365, 366)
!    UT: a given time in UTC.
! (output)
!         D: the solar declination in deg.
! SinD, CosD: sin(D), cos(D)
!         R: the solar distance in AU.
!        ET: the equation of time in deg.
! --+----*----+----*----+----*----+----*----+----*
subroutine SolarParams( YYYY, SDAY, UT, &
                        SinD, CosD, D, R, ET )
  use mGrobal
  implicit none
  integer YYYY, SDAY, DeltaTime
  double precision    UT, SinD, CosD, D, R, ET
  double precision    Hr, T, Tu, JC, Del, L,
  double precision    Sd, Cd, Dec, Ta, Tam, ET0
  double precision, save :: dHr = 0.0d0
  integer, save :: pYear = 0
  type(TIntDate) theDate
  type(TTDay) TDay
  integer*8 JD_of, JD
  double precision Obliquity, Clongit
  double precision MAscens, Eq, SolarDistance
!
  TDay%Value = SDAY
  call TdayToIntDate( TDay, YYYY, theDate )
!
  Hr = UT
  if ( pYear /= YYYY ) then
    dHr = DeltaTime( YYYY ) * 0.001d0 / 3600d0
    pYear = YYYY
  end if
  JD = JD_of( theDate )
  JC = JC2000( JD )
  T  = JC &
    + (Hr + dHr - 12d0) / (24d0*36525d0)
  Tu = JC &
    + (Hr - 12d0) / (24d0*36525d0)
  Del = Obliquity( T, .true. )
  L   = CLongit( T, .true. )
  Del = DegToRad( Del )
  L   = DegToRad( L )
  Sd  = -sin( L ) * sin( Del )
  Cd  = sqrt( 1.0 - Sd * Sd )
  Dec = atan( Sd / Cd ) ! in RAD.
  Ta  = tan( L ) * cos( Del )
  Tam = tan( DegToRad( 15d0*MAscens( Tu )) )
  ET0 = RadToDeg( &
    atan( (Tam - Ta) / (1d0 + Tam * Ta) ) )
  ET  = ET0 + Eq( T ) * 15d0
  SinD = sin( Dec )
  CosD = cos( Dec )
  D = RadToDeg( Dec )
  R = SolarDistance( T )
!
  return
end subroutine
! ------------------------------------------------
! Calculation of the sun position
! "Matsumoto Method (Rev. 2022)" (太陽位置)
! SUBROUTINE STYLE (JAPAN ONLY)
! (input)
!   Phi: Latitude of a given location in deg.
!   Ell: Longitude of a given location in deg.
!   JST: Hour in JST
!    ET: Eauation of time calculated in deg.
!  SinD: Sin of the solar declination calculated
!  CosD: Cosine of the solar declination calculated
! (output)
!  SinH, CosH: Sin and Cosine of the solar altitude
!  SinA, CosA: Sin and Cosine of the solar azimuth
! --+----*----+----*----+----*----+----*----+----*
subroutine SolarPosition( Phi, Ell, JST, ET, &
             SinD, CosD, SinH, CosH, SinA, CosA )
  ! This function is just for JAPAN.
  ! You can change 135.0 in the following
  ! code to be a lontitude value for
  ! reference one in your time zone.
  use mGrobal
  implicit none
  double precision Phi, Ell, JST, ET, SinD, CosD
  double precision SinH, CosH, SinA, CosA, T, P
```

```
!
  T = DegToRad(15d0 * (JST - 12d0) &
    + (Ell - 135d0) + ET);
  P = DegToRad(Phi);
  SinH = sin( P ) * SinD &
    + cos( P ) * CosD * cos( T )
  CosH = sqrt( abs(1.0 - SinH*SinH) )
  SinA = CosD * sin( T ) / CosH;
  CosA = (SinH * sin( P ) - SinD) &
    / (CosH * cos( P ));
end subroutine
!
! ------------------------------------------------
! THE CODES DESCRIBED IN THE ABOVE CAN BE USE,
! COPIED, AND MODIFIED WITHOUT ANY ACCEPTANCE OF
! THE AUTHOR: Dr. Shin-ichi Matsumoto, Prof.,
! Akita Prefectural University, Japan.
!
! HOWEVER, PROMISE THAT YOU NEVER CHANGE THE
! COPYRIGHT DESCRITIONS, AND IMPORTANT COEFFICIENT
! VALUES IN THE CODES.
! IN ADDITION, PLEASE REFER THIS PDF, IF YOU MAKE
! YOUR CALCULATED RESULT(S) OPEN TO THE PUBLIC.
!
! THE AUTHOR NEVER ACCEPT ANY RESPONSIBILITY TO
! CALCULATED RESULT(S) YOU GOT VIA THESE CODES.
! ------------------------------------------------
program SolPos
  use mGrobal
  implicit none
  type(TIntDate) Date
  type(TTday)    TDay
  type(TJday)    JDay
  integer   YY, SDAY, dT, DeltaTime
  integer*8 JDE12, JD_of
  double precision UT, SD, CD, D, R, ET, JC
!
  YY = 2020
  Date%Year  = YY
  Date%Month = 7
  Date%Day   = 24

  call IntDateToTday( Date, TDay )
  call IntDateToJday( Date, JDay )

  SDAY = TDay%Value
  JDE12 = JD_of( Date )
  JC = JC2000( JDE12 )
  dT = DeltaTime( YY )
  print *, "2020/07/24 =", SDAY
  print *, "J.Day@12TT =", JDE12
  print *, "Jul. Cent. =", JC
  print *, "Time Diff. =", dT * 0.001

  UT = 6d0
  call SolarParams( YY,SDAY,UT, SD,CD,D,R,ET )
  print *, "Declination (deg)", D
  print *, "Eq. of time (s)", ET * 240d0

  stop
end program SolPos
```

D.2 C/C++コード

C++言語による計算プログラムを以下に示す。太陽位置は，関数 SolPos(...) を呼び出して求めることができる。

```
// ================================================
//  main.cpp (example) メイン関数の例
// ================================================
#include <tchar.h>
#include <stdio.h>
#include <conio.h>
#include <System.Math.hpp>
// For following functions, Round(...),
// DegToRad(...), RadToDeg(...) and so on
// are local for a specified compiler.
// You may find/make equivalent ones.
#pragma hdrstop
#pragma argsused
// Declaration of important functions
#include "SolPos.h"
//
int _tmain( void )
{
  TIntDate  Date;
  double    Dlt, Et;
  double    SinD, CosD, D, R, CosV,
            SinH, CosH, SinA, CosA, h, A;
  //
  // Calculation for Tokyo in 15h JST,
  // July 24, 2020
  //
  int  Year = 2020 Date.year = Year;
  Date.month = 7; Date.day = 24;
  int  Jst = 15;
  double Phi =  35.0 + 41.1 / 60.0;
  double Ell = 139.0 + 45.6 / 60.0;
  //
  // By Rika Nempyo
  // (Chronological Scientific Tables)
  //
  Dlt = DegToRad( 19.7559 );
  SinD  = sin( Dlt ); CosD = cos( Dlt );
  Et = 15.0 / 3600.0 * (-392.4);
  SolarPosition( Phi, Ell, Jst, Et,
    SinD, CosD, SinH, CosH, SinA, CosA );
  h = RadToDeg( asin( SinH ) );
  A = RadToDeg( acos( CosA ) )
    * (SinA >= 0.0 ? 1.0 : -1.0);
  wprintf( L"Rika 6UT Intp.: h = %8.4lf, "
    "A =%8.4lf\n", h , A );
  //
  //  Matsumoto Method (Rev. 2022)
  //
  unsigned Sday =
    unsigned(IntDateToTday( Date ));
  Et = SolarParams( unsigned(Year), Sday,
    6.0, SinD, CosD, D, R );
  wprintf( L"Matsu Rev. 6UT: d = %8.4lf, "
    "Te=%8.1lf\n", D , Et * 240.0 );
  SolarPosition( Phi, Ell, Jst, Et,
    SinD, CosD, SinH, CosH, SinA, CosA );
  h = RadToDeg( asin( SinH ) );
  A = RadToDeg( acos( CosA ) )
    * (SinA >= 0.0 ? 1.0 : -1.0);
  wprintf( L"Rika 6UT Intp.: h = %8.4lf, "
    "A =%8.4lf\n", h , A );
  //
  //  Akasaka Method afer Yamazaki Method
  //
  Et = SolarParamsAk2( unsigned(Year),
    Sday, SinD, CosD, CosV );
  D = RadToDeg( asin( SinD ) );
  wprintf( L"Akasaka Method: d = %8.4lf, "
    "Te=%8.1lf\n", D , Et * 240.0 );
  SolarPosition( Phi, Ell, Jst, Et,
    SinD, CosD, SinH, CosH, SinA, CosA );
  h = RadToDeg( asin( SinH ) );
  A = RadToDeg( acos( CosA ) )
    * (SinA >= 0.0 ? 1.0 : -1.0);
  wprintf( L"Rika 6UT Intp.: h = %8.4lf, "
    "A =%8.4lf\n", h , A );

  printf( "Hit any key..." );
  getch();

  return ( 0 );
}
// ================================================
```

```
//  SolPos.h (header file)
// ================================================
#ifndef SolPosH
#define SolPosH
// ------------------------------------------------
// SolPos for the building env. eng'g.
// SolPos.h & SolPos.cpp
// - Functions related on calc. of
//   the solar position
// (c) 2014-2020, Shin-ichi Matsumoto
// ------------------------------------------------
typedef unsigned long ulong;
typedef long TJday; // Julian day
typedef int  TTday; // Serial day no.
extern const ulong JD2000;
#pragma pack(push,1)
typedef struct {
  int year; int month; int day;
} TIntDate;
#pragma pack(pop)
// ------------------------------------------------
extern void SolarPosition(
  const double Phi,  const double Ell,
  const double Jst,  const double Et,
  const double SinD, const double CosD,
  double& SinH, double& CosH,
  double& SinA, double& CosA );
  /* Solar position calculation func.
      specified for JAPAN
     INPUT
       Phi: Latit.(+deg.),
       Ell: Longit.(+deg.),
       Jst: Hour in JST(0-24),
        Et: Eq. of time (deg.),
       SinD, CosD: for the declination
     OUTPUT
       SinH, CosH: for the solar altit.
       SinA, CosA: for the solar azimuth
   NOTE
     The input parameters of Et, SinD,
     and CosD must be calculated by
     function SolarParams0(...), or
     SolarParams1(...).
  */
// ------------------------------------------------
extern bool Is_Leap( int aYYYY );
  /* Judge whether the given year
     (aYYYY) is leap or not.
    When it is a leap year, then returns
    true.
  */
// ------------------------------------------------
extern bool CorrectDate( int& aYear,
  int& aMonth, int& aDay );
  /* Corrects and Returns data of aYear,
     aMonth, and aDay on a date if it was
     wrong.
  */
// ------------------------------------------------
extern void TdayToIntDate( const TTday aTday,
  const int aYYYY, TIntDate& aIntDate );
  /* Serial day number (aTday) in a
     year (aYYYY) is converted into
     the TIntDate Struct (aIntDate).
  */
// ------------------------------------------------
extern TTday IntDateToTday(
  const TIntDate& aIntD );
  /* A given TIntDate Struct (aIntD)
     is converted into the serial day
     number as a return value.
  */
// ------------------------------------------------
extern TJday IntDateToJday(
  const TIntDate& aIntDate );
  /* Julian day for the given Struct
     TIntDate (aIntDate) is returned.
  */
// ------------------------------------------------
inline double JC2000( const double aJD )
{ /* Returns the Julian century based on
     J2000.0 for a given Julina day aJD.
     NOTE
      aJD is NOT INTEGER!
  */
  return ( (aJD - JD2000) / 36525.0 );
}
// ------------------------------------------------
extern ulong JD_of( const TIntDate& aDate );
  /* Returns a Julian day number after
     calculation based on the given
     Struct TIntDate arg.( aDate )
    NOTE
     Meeus Method was implemented.
     This function does not check the
     validation of the given arg.
     Execution of the correction func.,
     CorrectDate(...) is recommended
     before using this function.
     DON'T GIVE a date before Jan. 1,
     4713B.C.
     The day separation of the Julian
     day system is at noon, thus
     returned ulong value means
     the Julian day at noon.
     Ref:  W.H.Press et al.,
       Numerical Recipes in C,
       Cambridge Univ. Press, pp.28-29,
       1988.
  */
// ------------------------------------------------
extern long DeltaTime( const int aYear );
  /* Returns a time difference between TT
     and UT for OUT on July 1 in a given
     year.
     Matsumoto Method (Rev. 2022) after
     Espenak and Meeus (NASA TP, 2006)
    NOTE
     Return value is in MILI SECOND.
  */
// ------------------------------------------------
extern long DeltaTimeEx(
  const TIntDate& aUTDate );
  /* Returns a time difference between TT
     and UT1 for OUT on a given date.
     Matsumoto Method (Rev. 2022) after
     Espenak and Meeus (NASA TP, 2006)
    NOTE
     Return value is in MILI SECOND.
  */
// ------------------------------------------------
extern double Obliquity( const double aJC,
  const bool IsPrecise = true );
  /* (黄道傾角)
     Returns the obliquity of the
     ecliptic for a given Julian century
     based on the J2000.0 point (aJC).
    NOTE
     On consideration of the nutation,
     IsPrecise is switch.
     = true (default): with consideration
     = false: w/o consideration
     Return value is in DEG.
     Hwreafter, abbr. JC2000 for
     a Julian day based on the J2000.0.
  */
// ------------------------------------------------
extern double CLongit( const double aJC,
  const bool IsApparent = true );
  /* (視黄経または平均黄経)
     Returns the solar celestial
     longitude for a given JC2000.
    NOTE
     If IsApparent = false, then the
     mean value is retuerned.
     Return value is in DEG.
  */
// ------------------------------------------------
extern double MAscens( const double aUTJC );
  /* (平均太陽の赤経)
     Returns the mean solar right
     ascension for a given JC2000 (aUTJC).
    NOTE
     aUTJC should be calculated
     with UTC.
     Return value is in DEG.
  */
// ------------------------------------------------
```

```
extern double Eq( const double aJC );
  /* (分点差)
     Returns the equation of equinoxes
     for a given JC2000 (aJC).
    NOTE
     Return value is in HOUR.
     Based on the equation:
       dL * cos(eps) / 15.
     Where dL is the nutation of the
     celestial longitude[deg.] and
     eps is he apparent obliquity of
     the ecliptic.
  */
// ------------------------------------------------
extern double SolarDistance( const double aJC );
  /* (地心距離)
     Returns the geocentric distance to
     the sun for a given JC2000 (aJC).
    NOTE
     Return value is in AU.
  */
// ------------------------------------------------
// For calc. of the solar declination
// and the equation of time (solar
// parameters) (視赤緯と均時差)
//
// INPUT args.
//    aYear: a given year,
//  aSerDay: a given serial day of aYear,
//     aUTC: a given JC2000 value at
//           designated UTC.
// OUTPUT args.
//  D: the solar declination in DEG.
//     [SinD, CosD: sin(D), cos(D)],
//  R: the solar distance in AU.
// NOTE
//  Following all functions return
//  the equation of time in DEG.
//
// ------------------------------------------------
extern double SolarParams(
  const unsigned aYear,
  const unsigned aSerDay,
  const double aUTC,
  double& SinD, double& CosD,
  double&D, double& R );
  /* Matsumoto Method (Rev. 2020)
     (松本の方法 2020 年改)
  */
// ------------------------------------------------
extern double SolarParamsAk2(
  const unsigned aYear,
  double& SinD, double& CosD,
  double& CosV );
  /* Akasaka(2022) Method (赤坂 (2022) の方法)
    NOTE
     For output arg. CosV, see
     the following reference.
    H. Akasaka: Simplified Calculation Method of
      the Sun Position with Secular Changes,
      General Technical Report on the EA Weather
      Data. MetDS HP. 2022.8
     赤坂　裕：年差を考慮した太陽位置の簡易計算，
       EA 気象データ技術解説一般，MetDS HP，2022.8.
  */
// ------------------------------------------------
extern void Yamazaki(
  const SDate* aDate,
  const STime* aUT,
  double& Decl, double& Et );
  /* Yamazaki Method (山崎の方法)
    Ref. H. Yamazaki: Fundamental Formulae for
      Daylighting   IV (Precise Program to
      Calculate Solar Declination   and Equation of
      Time) (in Japanese), Proc. of AIJ Annual
      Meeting 1980, pp.407--408, 1980.9.
      山崎　均：日照環境の基礎計算式 IV (対象地域
      の太陽視赤緯及び均時差を正確に計算するプロ
      グラム)，日本建築学会大会学術講演梗概集，
      計画系，pp.407--408，1980.9.
  */
// ------------------------------------------------
#endif // End flag for "SolPos.h"
//
// ================================================
//  SolPos.cpp (source code file)
// ================================================
#include <math.h>
#include <System.Math.hpp>  // Local!
#pragma hdrstop
#include "SolPos.h"
#pragma package(smart_init) // Local!
// ------------------------------------------------
// SolPos for the building env. eng'g.
// SolPos.h & SolPos.cpp
// - Functions related on calc. of
//   the solar position
// (c) 2014-2022, Shin-ichi Matsumoto
// ------------------------------------------------
#define YEAR_GREG (1582) // Adoped in 1582
const ulong JD2000 = 2451545UL;
//          Julian day of Jan. 1 0UT, 2000
//
const
  int DAYS_OF_MONTH[12] = { 31, 28, 31,
  30, 31, 30, 31, 31, 30, 31, 30, 31, };
int Days_of_Month[12]; // for temp.
#pragma pack(push,2)
typedef struct {
  bool T; double A; double B; double C;
} TAstroChart;
#pragma pack(pop)
// ------------------------------------------------
// Calendar functions
// ------------------------------------------------
bool Is_Leap( int aYYYY )
{
  if ( aYYYY  <= YEAR_GREG )
    return ( false );
  if ( 0 == (aYYYY % 400) )
    return ( true  );
  if ( 0 == (aYYYY % 100) )
    return ( false );
  if ( 0 == (aYYYY % 4) )
    return ( true  );
  else
    return ( false );
}
// ------------------------------------------------
bool CorrectDate(
  int& aYear, int& aMonth, int& aDay )
{
  bool check = true;
  if ( 0 == aYear ) {
    aYear = 1; check = false; }
  if (  1 > aMonth ) {
    aMonth =  1; check = false; }
  if ( 12 < aMonth ) {
    aMonth = 12; check = false; }
  if (  1 > aDay ) {
    aDay = 1; check = false; }
  int leap = Is_Leap( aYear );
  for ( int m = 0; m < 12; m++ )
    Days_of_Month[m] = DAYS_OF_MONTH[m];
  if ( leap )
    Days_of_Month[1]++;
  if ( aDay > Days_of_Month[aMonth - 1]) {
    aDay = Days_of_Month[aMonth - 1 ];
    check = false;
  }
  return ( check );
}
// ------------------------------------------------
void TdayToIntDate( const TTday aTday,
  const int aYYYY, TIntDate& aIntDate )
{
  int m, day;
  int days[12];
  int leap = Is_Leap( aYYYY );
  for ( m = 0, day = 0; m < 12; m++ ) {
    day += DAYS_OF_MONTH[m];
    if ( leap && (1 == m) ) day++;
    days[m] = day;
```

```
  }
  TTday tday = aTday;
  if (   1 > aTday ) tday = 1;
  if ( 365 < aTday )
    tday = leap ? 366 : 365;
  aIntDate.year = aYYYY;
  for ( m = 1; m <= 12; m++ ) {
    if ( tday <= days[m - 1] ) {
      aIntDate.month = m; break; }
  }
  if ( 1 == aIntDate.month )
    aIntDate.day = tday;
  else
    aIntDate.day = tday
      - days[aIntDate.month - 2];
}
// -----------------------------------------------
TTday IntDateToTday(
  const TIntDate& aIntD )
{
  int m;
  if ( 1 == aIntD.month )
    return( TTday(aIntD.day) );
  bool leap = Is_Leap( aIntD.year );
  for ( m = 0; m < 12; m++ )
    Days_of_Month[m] = DAYS_OF_MONTH[m];
  if ( leap ) Days_of_Month[1]++;
  TTday days = (TTday)0;
  for ( m = 0; m < aIntD.month - 1; m++ )
    days += TTday(Days_of_Month[m]);
  return ( days + (TTday)aIntD.day );
}
// -----------------------------------------------
#define LGREG(Y,M,D) ((D)+31L*((M)+12L*(Y)))
#define GREG0 LGREG(1582,10,15)
// The Greg. calendar was adopted in...
TJday IntDateToJday(
  const TIntDate& aIntDate )
{ // General ver. of JD_of(...)
  TJday jul;
  int ja, jm, jy = aIntDate.year;
  if ( jy == 0 ) return( TJday(-1) );
  if ( jy < 0 )  jy++;
  if ( aIntDate.month > 2 )
    jm = aIntDate.month + 1;
  else {
    jy--; jm = aIntDate.month + 13;
  }
  jul  = TJday(floor( 365.25 * jy ));
  jul += TJday(floor( 30.6001 * jm ));
  jul += TJday(aIntDate.day)
    + TJday(1720995L);
  if ( GREG0 <= LGREG(aIntDate.year,
    long(aIntDate.month),
    long(aIntDate.day)) ) {
    // Use Gregorian Calendar
    ja   = int(0.01 * aIntDate.year);
    jul += TJday(2 - ja + int(0.25*ja));
  }
  return ( jul );
}
// -----------------------------------------------
ulong JD_of( const TIntDate& aDate )
{
  int ja, jm, jy = aDate.year;
  long jul;
  if ( 0 > jy ) ++jy;
  if ( 2 < aDate.month )
    jm = aDate.month + 1;
  else {
    --jy; jm = aDate.month + 13;
  }
  jul = long(floor( 365.25 * jy )
    + floor( 30.6001 * jm ) + aDate.day)
    + 1720995L;
  if ( GREG0 <= LGREG(aDate.year,
    long(aDate.month), long(aDate.day))) {
    ja   = int( 0.01 * jy );
    jul += 2 - ja +int( 0.25 * ja );
  }
  return ( (ulong)jul );
}
#undef GREG0
#undef LGREG(Y,M,D)
//
// vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
//  MAIN REVISION 2022 !
// -----------------------------------------------
extern long DeltaTime( const int aYear )
{
  TIntDate UTDate;
  UTDate.year  = aYear;
  UTDate.month = 7;
  UTDate.day   = 1;

  return ( DeltaTimeEx( UTDate ) );
}
// -----------------------------------------------
extern long DeltaTimeEx(
  const TIntDate& aUTDate )
{
  double y = aUTDate.year
    + (aUTDate.month - 0.5) / 12.0;
  if ( 7 == aUTDate.month &&
    1 == aUTDate.day )
    y = aUTDate.year + 0.5;
  if ( 1 == aUTDate.month &&
    1 == aUTDate.day ) y = aUTDate.year;

  double  t, S;
  long    R;

  if ( 1600 > aUTDate.year )  return (0L);
  else if ( 1701 > aUTDate.year ) {
    t = y - 1600.0;
    S =  ((1.0 / 7129.0 * t  - 0.01532)
      * t - 0.9808) * t + 120.0;
  }
  else if ( 1801 > aUTDate.year ) {
    t = y - 1700.0;
    S = (((-1.0 / 1174000.0 * t +
       0.00013336) * t - 0.0059285)
      * t  + 0.1603) * t + 8.83;
  }
  else if ( 1861 > aUTDate.year ) {
    t = y - 1800.0;
    S = ((((((0.000000000875 * t
      - 0.0000001699) * t + 0.0000121272)
      * t - 0.00037436) * t + 0.0041116)
      * t + 0.0068612) * t - 0.332447)
      * t + 13.72;
  }
  else if ( 1901 > aUTDate.year ) {
    t = y - 1860.0;
    S =  ((((1.0 / 233174.0 * t
      - 0.0004473624) * t + 0.01680668)
      * t - 0.251754) * t + 0.5737)
      * t + 7.62;
  }
  else if ( 1921 > aUTDate.year ) {
    t = y - 1900.0;
    S = (((-0.000197 * t + 0.0061966)
      * t - 0.0598939) * t + 1.494119)
      * t - 2.79;
  }
  else if ( 1941 > aUTDate.year ) {
    t = y - 1920.0;
    S = ((0.0020936 * t - 0.076100)
      * t + 0.84493) * t + 21.20;
  }
  else if ( 1961 > aUTDate.year ) {
    t = y - 1950.0;
    S = ((1.0 / 2547.0 * t  - 1.0 / 233.0)
      * t + 0.407) * t + 29.07;
  }
  else if ( 1986 > aUTDate.year ) {
    t = y - 1975.0;
    S = ((-1.0/718.0 * t  - 1.0 / 260.0)
      * t  + 1.067) * t + 45.45;
  }
  else if ( 2006 > aUTDate.year ) {
    t = y - 2000.0;
    S = ((((0.00002373599 * t
      + 0.000651814) * t + 0.0017275)
      * t - 0.060374)
```

```
      * t + 0.3345) * t + 63.86;
  }
  else if ( 2051 > aUTDate.year ) {
    t = y - 2000.0;
    S = (0.005589 * t + 0.32217)
    * t + 62.92;
  }
  else if ( 2151 > aUTDate.year ) {
    t = y - 1820.0;
    S = (0.0032 * t + 0.5628)
      * t - 205.724;
  }
  else S = 328.48;

  R = long( RoundTo( S, -3 ) * 1000.0 );

  return ( R );
}
// ------------------------------------------------
// ------------------------------------------------
double Obliquity( const double aJC,
  const bool IsPrecise )
{
  double epsilon = ((-5.036111e-7 * aJC
    + 1.638889e-7) * aJC + 1.300417e-2)
    * aJC - 2.343929e+1;
  if ( IsPrecise ) {
    double T1 = DegToRad(fmod(
       1934.0 * aJC + 235.0, 360.0 ));
    double T2 = DegToRad(fmod(
      72002.0 * aJC + 201.0, 360.0 ));
    epsilon  -= 0.00256 * cos( T1 )
      + 0.00015 * cos( T2 );
  }
  return ( epsilon );
}
// ------------------------------------------------
#define _T (true)
#define _F (false)
double CLongit( const double aJC,
  const bool IsApparent )
{
  static const int C_MAX = 18;
  static const TAstroChart Coef[C_MAX] = {
  {_T, 36000.7695,0.0,0.0},
  {_F, 280.465900,0.0,0.0},
  {_F, 1.91470000,35999.050,267.52000},
  {_F, 0.02000000,71998.100,265.10000},
  {_T,-0.00480000,35999.000,268.00000},
  {_F, 0.00200000,32964.000,158.00000},
  {_F, 0.00180000,19.000000,159.00000},
  {_F, 0.00180000,445267.00,208.00000},
  {_F, 0.00150000,45038.000,254.00000},
  {_F, 0.00130000,22519.000,352.00000},
  {_F, 0.00070000,65929.000,45.000000},
  {_F, 0.00070000,3035.0000,110.00000},
  {_F, 0.00070000,9038.0000,64.000000},
  {_F, 0.00060000,33718.000,316.00000},
  {_F, 0.00050000,155.00000,118.00000},
  {_F, 0.00050000,2281.0000,221.00000},
  {_F, 0.00040000,29930.000,48.000000},
  {_F, 0.00040000,31557.000,161.00000},
  };
  double cl = 0.0, A, BT_C;
  for ( int i = C_MAX - 1; i >= 2; i-- ) {
    // DON'T CHANGE the loop INDEX ORDER
    // to avoid information loss!
    A    = (Coef[i].T ?
      Coef[i].A * aJC : Coef[i].A);
    BT_C = Coef[i].B * aJC + Coef[i].C;
    BT_C = DegToRad( fmod(BT_C, 360.0) );
    cl += A * cos( BT_C );
  }
  cl += Coef[1].A + Coef[0].A * aJC;
  if ( IsApparent ) {
    A    = 0.0048;
    BT_C = 1934.0 * aJC + 145.0;
    BT_C = DegToRad( fmod(BT_C, 360.0) );
    cl += A * cos( BT_C );
    A    = -0.0004;
    BT_C = 72002.0 * aJC + 111.0;
    BT_C = DegToRad( fmod(BT_C, 360.0) );
    cl += A * cos( BT_C ) - 0.0057;
  }
  cl = fmod(cl, 360.0);
  return ( cl >= 0.0 ? cl : cl + 360.0 );
}
// ------------------------------------------------
double MAscens( const double aUTJC )
{
  static const double SecConst
    = 67310.54841;
  double Sec = -0.0000062
    * aUTJC * aUTJC * aUTJC;
  Sec += 0.093104 * aUTJC * aUTJC;
  Sec += 8640184.812866 * aUTJC;
  Sec += SecConst;
  double Hour = Sec / 3600.0;
  Hour = fmod(Hour, 24.0);
  return ( Hour < 0 ? Hour+24.0 : Hour );
}
// ------------------------------------------------
double Eq( const double aJC )
{
  double eq = 0.0048 * sin(
    DegToRad(fmod(1934.0 * aJC + 235.0,
    360.0)) ) - 0.0004 * sin( DegToRad(
    fmod(72002.0 * aJC + 201.0, 360.0)) );
  eq *= cos( DegToRad(
    Obliquity( aJC, true )) );
  return ( eq / 15.0 );
}
// ------------------------------------------------
double SolarDistance( const double aJC )
{
  static const int C_MAX = 9;
  static const TAstroChart Coef[C_MAX] = {
  {_F, 1.00014000,0.00000000,0.00000000},
  {_F, 0.01670600,35999.0500,177.530000},
  {_F, 0.00013900,71998.0000,175.000000},
  {_T,-0.00004200,35999.0000,178.000000},
  {_F, 0.00003100,445267.000,298.000000},
  {_F, 0.00001600,32964.0000,68.0000000},
  {_F, 0.00001600,45038.0000,164.000000},
  {_F, 0.00000500,22519.0000,233.000000},
  {_F, 0.00000500,33718.0000,226.000000},
  };
  double sd = 0.0, A, BT_C;
  for ( int i = TAC_MAX-1; i >= 1; i-- ) {
    // DON'T CHANGE the loop INDEX ORDER
    // to avoid information loss!
    A    = (Coef[i].T ?
      Coef[i].A * aJC : Coef[i].A);
    BT_C = Coef[i].B * aJC + Coef[i].C;
    BT_C = DegToRad(fmod(BT_C, 360.0));
    sd  += A * cos( BT_C );
  }
  sd += Coef[0].A;
  return ( sd );
}
#undef _T
#undef _F
// ------------------------------------------------
double SolarParamsAk2(
  const unsigned aYear, const unsigned aSerDay,
  const unsigned aDay,  const unsigned aHour,
  const unsigned aMin,  const unsigned aSec,
  const int aLongit0,
  double& SinD, double& CosD, double& CosV )
{
  //
  // Akasaka(2022) Method
  //
  int     n = int(aYear - 1968u);
  double d0 =-(n+3) /4; d0 += 3.71 + 0.2596 * n;
  double Delta0 = double(DegToRad(-23.4393 +
                  0.00013 * (aYear - 2000)));
  double  M = 0.9856 * (double(aSerDay) - d0);
  double MRad = DegToRad(M);
  double Eps  = 12.39 + 0.0172 *
                (double(n) + M / 360.0);
  double V    = M + 1.914 * sin(MRad)
              + 0.02 * sin(2.0 * MRad);
  double VEps = DegToRad(V + Eps);
  double VE2  = 2.0 * VEps;
  double Ang  = atan( (0.043 * sin( VE2 ))
```

```
              / (1.0 - 0.043 * cos( VE2 )) );
  double Et   = (M - V) - RadToDeg(Ang);
  SinD = cos( VEps ) * sin( Delta0 );
  CosD = sqrt( fabs( 1.0 - SinD*SinD ) );
  CosV = cos( DegToRad(V) );

  return ( Et ); // in DEG.! NOT in MIN.
}
// ------------------------------------------------
void Yamazaki(
  const SDate* aDate,
  const STime* aUT,
  double& Decl, double& Et )
{
  //
  // Yamazaki Method
  //
  double Hour = anUT->HH + (anUT->MM * 60.0
              + anUT->SS) / 3600.0;
  Hour = anUT->PM == sgnPlus ? Hour : -Hour;
  SDate Date = *aDate;
  int yn = Date.YY - 1900;
  int d2 = static_cast<int>((yn - 1) / 4.0);
  // temporary.....................
  double d = SerialDay_of( Date )
           + (yn - 30) * 1.1574e-5;
  //..............................
  // double d = SerialDay_of( Date );
  if ( yn >= 85 ) d += (yn - 30) * 1.1574e-5;
  d += (Hour - 12.0) / 24.0;
  double t1 = (365.0 * yn + d2 + d) / 36525.0;
  double t2 = t1 * t1;
  double t3 = t2 * t1;
//  a little bit bonehead but as it is original
  double delta0 = -9.44e-5 + 1.30125e-2 * t1
                + 1.64e-6 * t2 + 5.0e-7 * t3;
  delta0 -= 23.4522;
  delta0 = DegToRad( delta0 );
  double e = 1.04e-6 - 4.18e-5 * t1
           - 1.26e-7 * t2 + 0.01675;
  double eps = 0.719175 * t1 + 0.000453 * t2;
  eps += 0.220833 + t1 + 11.0;
  eps = DegToRad( eps );
  double m = 6.00267e-4 * (d2 + d)
           - 9.02579e-4 * yn
           - 0.00015 * t2 - 1.667e-4;
  m += -1.524 - 0.255 * yn + 0.985 * d2;
  m += 0.985 * d;
  m = DegToRad( m );

  double SinM   = sin( m );
  double Sin2M  = sin( 2.0 * m );
  double Sin3M  = sin( 3.0 * m );

  double dm = 9.93502e-5 * (1.0 - e * (cos( m )
            - 2.0 * e * SinM * SinM));
  m -= dm;

  double v = (2.0 - 0.25 * e * e) * e * SinM;
  v += 1.25 * e * e * Sin2M;
  v += 13.0 / 12.0 * e * e * e * Sin3M;
  v += m;

  double sd = cos( eps + v ) * sin( delta0 );
  double cd = sqrt( 1.0 - sd * sd );
  double v_eps2 = 2.0 * (eps + v);
  double a = (1.0 - cos( delta0 ))
           / (1.0 + cos( delta0 ));
  // Solar Declination in DEG. 視赤緯
  Decl = RadToDeg( atan( sd / cd ) )
  // Equation of time in min.  均時差
  double et1 = RadToDeg( m - v );
  double et2 = -atan( (a * sin( v_eps2 ))
             / (1.0 - a * cos( v_eps2 )) )
  et2 = RadToDeg( et2 );
  Et = (et1 + et2) * 4.0;
}
// ------------------------------------------------
double SolarParams(
    const unsigned aYear,
    const unsigned aSerDay,
    const double aUTC,
    double& SinD, double& CosD,
    double&D, double& R )
{
  //
  // Matsumoto Method (Rev. 2022)
  // with DeltaTime and DeltaTimeEx
  //
  static double   dHr(0.0);
  static unsigned pYear(0);
  TIntDate        theDate;
  TdayToIntDate(aSerDay, aYear, theDate);
  double Hr = aUTC; // UT
  if ( pYear != aYear ) {
    dHr = DeltaTime( aYear )
      * 0.001 / 3600.0;  // in Hour;
    // LOL!
    // This is only difference with
    // function SolarParams0(...)
    pYear = aYear;
  }
  double JC = JC2000( JD_of( theDate ) );
  double T  = JC
    + (Hr + dHr - 12.0) / (24.0*36525.0);
  double Tu = JC
    + (Hr - 12.0) / (24.0 * 36525.0);
  double Del =
    DegToRad(Obliquity( T, true ));
  double L   =
    DegToRad(CLongit( T, true ));
  double Sd  = -sin( L ) * sin( Del );
  double Cd  = sqrt( 1.0 - Sd * Sd );
  double Dec = atan( Sd / Cd ); // in RAD.
  double Ta  = tan( L ) * cos( Del );
  double Tam =
    tan( DegToRad( 15.0*MAscens( Tu )) );
  double et  = RadToDeg(
    atan( (Tam - Ta) / (1.0 + Tam*Ta) ));
    // in DEG.
  double Et  = et + Eq( T ) * 15.0;
  SinD = sin( Dec );
  CosD = cos( Dec );
  D = RadToDeg( Dec );
  R = SolarDistance( T );
  return ( Et ); // in DEG.! NOT in MIN.
}
// ------------------------------------------------
void SolarPosition(
    const double Phi,  // Latit. in deg.
    const double Ell,  // Longit. in deg.
    const double Jst,  // Hour in JST
    const double Et,   // Eqt. in deg.
                       // D: Declination
    const double SinD, // H: Solar altit.
    const double CosD, // A: Solar azim.
    double& SinH, double& CosH, //
    double& SinA, double& CosA )
{
  // This function is just for JAPAN.
  // You can change 135.0 in the following
  // code to be a lontitude value for
  // reference one in your time zone.
  double T = DegToRad(15.0 * (Jst - 12.0)
    + (Ell - 135.0) + Et);
  double P = DegToRad(Phi);
  SinH = sin( P ) * SinD
    + cos( P ) * CosD * cos( T );
  CosH = sqrt( fabs( 1.0 - SinH*SinH ) );
  SinA = CosD * sin( T ) / CosH;
  CosA = (SinH * sin( P ) - SinD)
    / (CosH * cos( P ));
}
// ------------------------------------------------
// THE CODES DESCRIBED IN THE ABOVE CAN BE USE,
// COPIED, AND MODIFIED WITHOUT ANY ACCEPTANCE OF
// THE AUTHOR: Dr. Shin-ichi Matsumoto, Prof.,
// Akita Prefectural University, Japan.
//
// HOWEVER, PROMISE THAT YOU NEVER CHANGE THE
// COPYRIGHT DESCRITIONS, AND IMPORTANT COEFFICIENT
// VALUES IN THE CODES.
// IN ADDITION, PLEASE REFER THIS PDF, IF YOU MAKE
// YOUR CALCULATED RESULT(S) OPEN TO THE PUBLIC.
//
// THE AUTHOR NEVER ACCEPT ANY RESPONSIBILITY TO
// CALCULATED RESULT(S) YOU GOT VIA THESE CODES.
// ------------------------------------------------
```

D.3 C/C++コードによる計算例

計算例として，2051 年の 1 月 0 日 0^{h}TT（すなわち，2050 年 12 月 31 日 0^{h}TT）から，5 日刻みで 1 年分，ユリウス日 JDE（JDE@0TT），太陽視赤緯 δ （Decl.）[°]，均時差 T_e（Te）[s]，太陽までの地心距離 r（r）[–, AU] を計算した結果を表 10 に示す。FORTRAN コードによる計算結果も同等である [注13]。

表 10 2051 年の太陽位置パラメータの C/C++コードによる計算例

Year: 2051, Delta T: 96.059sec. by Matsumoto Method (Rev. 2022)

MM/DD	JDE@0TT	Decl.	Te. (sec.)	r [AU]	MM/DD	JDE@0TT	Decl.	Te. (sec.)	r [AU]
01/00*	2470171.5	-23.09193	-165.31929	0.98334	07/04	2470356.5	22.88638	-264.64520	1.01670
01/05	2470176.5	-22.63395	-304.33107	0.98332	07/09	2470361.5	22.37526	-314.77421	1.01670
01/10	2470181.5	-21.98920	-433.22671	0.98343	07/14	2470366.5	21.70330	-354.61873	1.01655
01/15	2470186.5	-21.16544	-548.48732	0.98364	07/19	2470371.5	20.87733	-381.85381	1.01627
01/20	2470191.5	-20.17269	-646.98543	0.98394	07/24	2470376.5	19.90510	-395.06927	1.01589
01/25	2470196.5	-19.02258	-726.55231	0.98437	07/29	2470381.5	18.79491	-393.72087	1.01542
01/30	2470201.5	-17.72755	-786.20481	0.98495	08/03	2470386.5	17.55570	-377.67640	1.01486
02/04	2470206.5	-16.30053	-825.83925	0.98567	08/08	2470391.5	16.19736	-346.84386	1.01417
02/09	2470211.5	-14.75525	-845.71362	0.98649	08/13	2470396.5	14.73074	-301.24806	1.01335
02/14	2470216.5	-13.10632	-846.21164	0.98738	08/18	2470401.5	13.16702	-241.43677	1.01242
02/19	2470221.5	-11.36882	-828.05557	0.98835	08/23	2470406.5	11.51701	-168.75724	1.01142
02/24	2470226.5	-9.55748	-792.66398	0.98941	08/28	2470411.5	9.79110	-85.18456	1.01036
03/01	2470231.5	-7.68611	-742.21138	0.99057	09/02	2470416.5	7.99984	7.15827	1.00924
03/06	2470236.5	-5.76796	-679.26110	0.99183	09/07	2470421.5	6.15451	106.34449	1.00803
03/11	2470241.5	-3.81639	-606.27623	0.99315	09/12	2470426.5	4.26704	210.55636	1.00673
03/16	2470246.5	-1.84520	-525.43320	0.99448	09/17	2470431.5	2.34929	317.70756	1.00537
03/21	2470251.5	0.13186	-438.85737	0.99583	09/22	2470436.5	0.41250	425.20099	1.00399
03/26	2470256.5	2.10184	-348.98347	0.99722	09/27	2470441.5	-1.53242	530.10232	1.00261
03/31	2470261.5	4.05280	-258.62747	0.99865	10/02	2470446.5	-3.47408	629.58878	1.00122
04/05	2470266.5	5.97320	-170.65203	1.00012	10/07	2470451.5	-5.40006	721.25778	0.99979
04/10	2470271.5	7.85108	-87.51879	1.00157	10/12	2470456.5	-7.29715	803.02375	0.99833
04/15	2470276.5	9.67398	-11.13769	1.00298	10/17	2470461.5	-9.15216	872.74892	0.99687
04/20	2470281.5	11.42946	56.87268	1.00435	10/22	2470466.5	-10.95226	928.02983	0.99547
04/25	2470286.5	13.10585	114.81770	1.00569	10/27	2470471.5	-12.68466	966.41643	0.99412
04/30	2470291.5	14.69226	160.87469	1.00702	11/01	2470476.5	-14.33590	985.91275	0.99282
05/05	2470296.5	16.17804	193.46856	1.00832	11/06	2470481.5	-15.89164	985.32270	0.99154
05/10	2470301.5	17.55225	211.74972	1.00954	11/11	2470486.5	-17.33722	964.15641	0.99030
05/15	2470306.5	18.80392	215.72964	1.01066	11/16	2470491.5	-18.65853	922.26004	0.98913
05/20	2470311.5	19.92272	205.96466	1.01168	11/21	2470496.5	-19.84235	859.62444	0.98807
05/25	2470316.5	20.89953	183.13841	1.01263	11/26	2470501.5	-20.87619	776.65109	0.98712
05/30	2470321.5	21.72643	147.99991	1.01352	12/01	2470506.5	-21.74811	674.66748	0.98627
06/04	2470326.5	22.39640	101.73850	1.01433	12/06	2470511.5	-22.44716	556.18637	0.98550
06/09	2470331.5	22.90333	46.39453	1.01503	12/11	2470516.5	-22.96410	424.61963	0.98481
06/14	2470336.5	23.24239	-15.17115	1.01558	12/16	2470521.5	-23.29203	283.68523	0.98424
06/19	2470341.5	23.41058	-79.79708	1.01600	12/21	2470526.5	-23.42651	137.03591	0.98381
06/24	2470346.5	23.40676	-144.58355	1.01633	12/26	2470531.5	-23.36537	-11.62754	0.98354
06/29	2470351.5	23.23139	-207.01679	1.01657	12/31	2470536.5	-23.10885	-158.24134	0.98339

[注13] C/C++と FORTRAN で，不動小数点定数や使用する組込み関数の精度などが等しくなるような調整をしていないことなどの理由で，両者で僅かな違いが生じるが，実用上問題とならないと考える。特に，$T_e = E_t * 4\mathrm{min.} * 60\mathrm{sec./min.}$ であるため，小数点以下の比較的小さな位で違いが生じるが，均時差としてのミリ秒差と考えれば，深刻な問題ではなかろう。

参考文献

［1］ 松本真一：太陽視赤緯・均時差計算法の改良—松本の方法（2022 年改）, 日本建築学会東北支部研究報告集, 第 85 号, pp.27–34, 2022.6.

［2］ 松本真一：太陽視赤緯・均時差の代表的計算方法４種類の概説とそれらの計算精度の比較, 空気調和・衛生工学会令和４年度大会論文集, 第 5 巻, pp.000-999, 2022.9 (掲載予定).

［3］ 松本真一：海上保安庁海洋情報部の式の援用による太陽視赤緯と均時差の計算について, 空気調和・衛生工学会令和１年度大会論文集, 第 5 巻, pp.97-100, 2019.9.

［4］ 松本真一：太陽視赤緯・均時差計算法（松本の方法）の精度検証, 日本建築学会東北支部研究報告集, 計画系, 第 82 号, pp.7–10, 2019.6.

［5］ 松本真一：太陽視赤緯・均時差計算に関する筆者の方法の精度検証, 日本建築学会大会学術講演梗概集, D–2 (環境工学 II), pp.25–26, 2014.9.

［6］ 松本真一：太陽視赤緯と均時差の計算法に関する補遺, 日本建築学会東北支部研究報告集, 計画系, 第 77 号, pp.49–56, 2014.6.

［7］ 松本真一：太陽視赤緯と均時差の計算精度の検討, 日本建築学会大会学術講演梗概集, D–2 (環境工学 II), pp.7–8, 2006.8.

［8］ 松本真一：太陽視赤緯と均時差計算に関する一考察, 日本建築学会東北支部研究報告集, 計画系, 第 68 号, pp.89–96, 2005.6.

［9］ 長澤　工：天体の位置計算 増補版, 地人書館 (東京), 1985.

［10］ Simon Newcomb: A Compendium of Spherical Astronomy with its Applications to the Determination and Reduction of Positions of the Fixed Stars, Macmillan & Co., New York, 1906.

［11］ 例えば，田中俊六，武田　仁　他：日照・日射, 最新建築環境工学（改訂４版）, 田中俊六 編 第 3 章, pp.80–86, 井上書院（東京），2014.

［12］ 例えば，渡辺俊行，林　徹夫　他：日照と日射, 建築環境工学, 浦野良美・中村　洋 編 第 4 章, pp.134–139, 森北出版（東京），1996.

［13］ 山崎　均：日照環境の基礎計算式 IV(対象地域の太陽視赤緯及び均時差を正確に計算するプログラム), 日本建築学会大会学術講演梗概集, 計画系, , pp.407–408, 1980.9.

［14］ 赤坂　裕　他：拡張アメダス気象データ, 日本建築学会 (丸善, 東京), 2000.

［15］ Hiroshi Akasaka et al.: Expanded AMeDAS Weather Data, Architectural Institute of Japan (Maruzen, Tokyo), 2003.

［16］ 赤坂　裕　他：拡張アメダス気象データ 1981–2000, 日本建築学会 (鹿児島 TLO, 鹿児島), 2005.

［17］ 赤坂　裕：年差を考慮した太陽位置の簡易計算（“TE_Simplified_SP220804.pdf”）, MetDS (株) 気象データシステム ホームページ「拡張アメダス気象データ 技術解説一般」, https://www.metds.co.jp/, 2022.8.

［18］ 国立天文台 (編)：理科年表プレミアム（会員制電子ブック）, https://www.rikanenpyo.jp/（2022.5.31 最終アクセス）.

［19］ 暦計算研究会 (編)：新こよみ便利帳 天文現象・暦計算のすべて, 恒星社厚生閣（東京）, 1991.

［20］ Pierre Bretagnon and George Francou: Planetary theories in rectangular and spherical variables. VSOP87 solutions, Astronomy and Astrophysics, 202, 1988, pp.309–315.

［21］ Ibrahim Reda and Afsin Andreas: Solar Position Algorithm for Solar Radiation Applications, NREL Report (No. TP–560–34302), NREL, Golden, 2003 (Rev.2008.1).

［22］ Jean Meeus: Astronomical Algorithms (2nd Ed.), Willmann-Bells, Inc, Virginia, 1999.

［23］ 海上保安庁海洋情報部（HP）: https://www1.kaiho.mlit.go.jp/KOHO/ index.html （2022.3.6 最終アクセス）.

［24］ Fred Espenak and Jean Meeus: Five Millennium Canon of Solar Eclipses: −1999 to +3000, The NASA Technical Publication (NASA/TP–2006–214141), NASA, Greenbelt, 2006.10.

［25］ Nautical Almanac Offices of U.S. Naval Observatory and Her Majesty's Hydrographic Office U.K.: The Astronomical Almanac for the year 2020, U.S. Government Publishing Office, Washington D.C., pp. K8–K9, etc., 2019.