ELP/MPP02 Lunar Ephemeris

Yuk Tung Liu

2018-08-14, last modified: 2024-12-03

ELP/MPP02 is a semi-analytic solution for the lunar motion developed by J. Chapront and G. Francou in 2002. It is an improvement of the ELP2000-82B lunar theory. The major paper about the ELP/MPP02 theory is *The lunar theory ELP revisited. Introduction of new planetary perturbations* by J. Chapront and G. Francou, Astronomy and Astrophysics, v.404, p.735-742 (2003), which you can also find references to the ELP2000-82B theory. The authors provide data files, a FORTRAN code and a pdf documentation on this ftp site. ELP/MPP02 theory provides two sets of parameters adjusted to fit either the lunar laser ranging (LLR) observation data or JPL's DE405/DE406 ephemerides.

I have written C++ functions to compute the lunar positions based on the information on that ftp site. The full ELP/MPP02 theory contains series involving 35901 terms. High accuracy of lunar positions may not be necessary for some applications. It is therefore useful to create a truncated series to speed up computation. I have written routines that create a truncated series using 4 parameters. I also wrote a function to estimate the accuracy of the truncated series.

JavaScript is convenient for HTML-based applications, such as my local star charts and equatorial star charts pages. I wrote C++ routines that generate JavaScript functions to compute a truncated ELP/MPP02 series.

This document describes my implementation of the ELP/MPP02 series, the creation of a truncated series, and the estimation of the accuracy of the truncated series. I do not explain the ELP/MPP02 theory in this document. Readers who are interested in the theory can read the paper and the pdf document on the ftp site mentioned above.

1 Implementation of ELP/MPP02

The equations for computing the ELP/MPP02 are given in the pdf document on this ftp site. In this section, I rewrite the equations in a form that is more convenient for code development.

1.1 Parameters Adjusted to Fit LLR or DE405/DE406

The following are values of the parameters adjusted to fit LLR or DE405/DE406. I use the same notation as in the pdf document on this ftp site.

Variable	Name in the code	LLR	DE405/DE406
$\Delta W_1^{(0)}$	Dw1_0	$-0.10525''$	$-0.07008''$
$\Delta W_2^{(0)}$	Dw2_0	$+0.16826''$	$+0.20794''$
$\Delta W_3^{(0)}$	Dw3_0	$-0.10760''$	$-0.07215''$
$\Delta W_1^{(1)}$	Dw1_1	$-0.32311''/\mathrm{cy}$	$-0.35106''/\mathrm{cy}$
$\Delta W_2^{(1)}$	Dw2_1	$+0.08017''/\mathrm{cy}$	$+0.08017''/\mathrm{cy}$
$\Delta W_3^{(1)}$	Dw3_1	$-0.04317''/\mathrm{cy}$	$-0.04317''/\mathrm{cy}$
$\Delta W_1^{(2)}$	Dw1_2	$-0.03794''/\mathrm{cy}^2$	$-0.03743''/\mathrm{cy}^2$
$\Delta\Gamma$	Dgam	$+0.00069''$	$+0.00085''$
ΔE	De	$+0.00005''$	$-0.00006''$
$\Delta T^{(0)}$	Deart_0	$-0.04012''$	$-0.00033''$
$\Delta T^{(1)}$	Deart_1	$+0.01442''/\mathrm{cy}$	$+0.00732''/\mathrm{cy}$
$\Delta\varpi'^{(0)}$	Dperi	$-0.04854''$	$-0.00749''$
$\Delta e'$	Dep	$+0.00226''$	$+0.00224''$
$\Delta W_1^{(3)}$	Dw1_3	0	$-0.00018865''/\mathrm{cy}^3$
$\Delta W_1^{(4)}$	Dw1_4	0	$-0.00001024''/\mathrm{cy}^4$
$\Delta W_2^{(2)}$	Dw2_2	0	$+0.00470602''/\mathrm{cy}^2$
$\Delta W_2^{(3)}$	Dw2_3	0	$-0.00025213''/\mathrm{cy}^3$
$\Delta W_3^{(2)}$	Dw3_2	0	$-0.00261070''/\mathrm{cy}^2$
$\Delta W_3^{(3)}$	Dw3_3	0	$-0.00010712''/\mathrm{cy}^3$

Here cy denotes Julian century. Note that the last 6 variables in the DE405/DE406 column are used to make ELP/MPP02 approaches closely the JPL Ephemeris DE406 on a long range (a few seconds over 6 millennia), whereas the rest are fitted to DE405 only.

There are 8 more parameters needed. To calculate them, the following constants are calculated first.

j	$B'_{2,j}$	$B'_{3,j}$
1	0.311079095	-0.103837907
2	-0.004482398	0.000668287
3	-0.001102485	-0.001298072
4	0.001056062	-0.000178028
5	0.000050928	-0.000037342

$$
\begin{aligned}
m &= n'/\nu = 0.074801329,\\
\alpha &= a_0/a' = 0.002571881,\\
W_1^{(1)} &= 1732559343.73604''/\mathrm{cy} + \Delta W_1^{(1)},\\
W_2^{(1)} &= 14643420.3171''/\mathrm{cy} + \Delta W_2^{(1)},\\
W_3^{(1)} &= -6967919.5383''/\mathrm{cy} + \Delta W_3^{(1)},\\
\delta\nu &= 0.55604''/\mathrm{cy} + \Delta W_1^{(1)},\\
\delta\Gamma &= -0.08066'' + \Delta\Gamma,\\
\delta E &= 0.01789'' + \Delta E,\\
\delta e' &= -0.12879'' + \Delta e',\\
\delta n' &= -0.0642''/\mathrm{cy} + \Delta T^{(1)}.
\end{aligned}
$$

The following are the equations for the 8 remaining parameters.

$$\begin{aligned}
\delta W_2^{(1)} &= \left[\frac{W_2^{(1)}}{W_1^{(1)}} - m\left(B'_{2,1} + \frac{2\alpha}{3m}B'_{2,5}\right)\right]\Delta W_1^{(1)} + \left(B'_{2,1} + \frac{2\alpha}{3m}B'_{2,5}\right)\Delta T^{(1)} \\
&\quad + W_1^{(1)}(B'_{2,2}\Delta\Gamma + B'_{2,3}\Delta E + B'_{2,4}\Delta e') \\
\delta W_3^{(1)} &= \left[\frac{W_3^{(1)}}{W_1^{(1)}} - m\left(B'_{3,1} + \frac{2\alpha}{3m}B'_{3,5}\right)\right]\Delta W_1^{(1)} + \left(B'_{3,1} + \frac{2\alpha}{3m}B'_{3,5}\right)\Delta T^{(1)} \\
&\quad + W_1^{(1)}(B'_{3,2}\Delta\Gamma + B'_{3,3}\Delta E + B'_{3,4}\Delta e') \\
f_A &= 1 - \frac{2\delta\nu}{3W_1^{(1)}} \\
f_{B1} &= \frac{\delta n' - m\delta\nu}{W_1^{(1)}} \\
f_{B2} &= \delta\Gamma \quad \text{(in radians)} \\
f_{B3} &= \delta E \quad \text{(in radians)} \\
f_{B4} &= \delta e' \quad \text{(in radians)} \\
f_{B5} &= \frac{2\alpha}{3mW_1^{(1)}}(\delta n' - m\delta\nu).
\end{aligned}$$

The first 21 parameters $\Delta W_1^{(0)}$, $\Delta W_2^{(0)}$, $\Delta W_3^{(0)}$, $\Delta W_1^{(1)}$, $\Delta W_2^{(1)}$, $\Delta W_3^{(1)}$, $\Delta W_1^{(2)}$, $\Delta\Gamma$, ΔE, $\Delta T^{(0)}$, $\Delta T^{(1)}$, $\Delta\varpi'^{(0)}$, $\Delta e'$, $\Delta W_1^{(3)}$, $\Delta W_1^{(4)}$, $\Delta W_2^{(2)}$, $\Delta W_2^{(3)}$, $\Delta W_3^{(2)}$, $\Delta W_3^{(3)}$, $\delta W_2^{(1)}$, and $\delta W_3^{(1)}$ are grouped together in the struct `Elp_paras` in the C++ file `ElpMpp02.cpp`. The last 6 parameters f_A, f_{B1}, f_{B2}, f_{B3}, f_{B4}, and f_{B5} are grouped in the struct `Elp_facs` in `ElpMpp02.cpp`. These 27 parameters are fixed once LLR or DE405/DE406 are specified. They only need to be computed once.

These parameters are calculated in the subroutine
`setup_parameters(int corr, Elp_paras &paras, Elp_facs &facs)`
Parameters fitted to LLR are computed if `corr` is 0 and parameters fitted to DE405/DE406 are computed if `corr` is 1. These groupings are convenient if one wants to construct other sets of parameters fitted to other data (e.g. DE431).

1.2 Coefficients of the ELP/MPP02 Series

The coefficients are stored in the 14 data files:
`elp_main.long`, `elp_main.lat`, `elp_main.dist`,
`elp_pert.longT0`, `elp_pert.longT1`, `elp_pert.longT2`, `elp_pert.longT3`,
`elp_pert.latT0`, `elp_pert.latT1`, `elp_pert.latT2`,
`elp_pert.distT0`, `elp_pert.distT1`, `elp_pert.distT2`, `elp_pert.distT3`.

They are constructed from the data files on this ftp site and put in a format more convenient for C++ implementation. In each file, the first line is an integer indicating the number of terms in the series.

The first three files `elp_main.long`, `elp_main.lat`, and `elp_main.dist` are the Fourier series of the longitude, latitude and distance. They contain 11 columns containing

$$\{i_1, i_2, i_3, i_4, A, B_1, B_2, B_3, B_4, B_5, B_6\}.$$

The series is calculated by the equation

$$\sum_{\{i\}} \tilde{A}_{\{i\}} \left\{ \begin{array}{c} \sin \\ \cos \end{array} \right\} (i_1 D + i_2 F + i_3 l + i_4 l') \quad \{i\} = \{i_1, i_2, i_3, i_4\}. \tag{1}$$

Longitude and latitude are sine series and distance is cosine series. The 4 variables D, F, l and l' are the Delaunay arguments (see the next subsection). The coefficients $\tilde{A}_{\{i\}}$ are calculated according to

$$\tilde{A}_{\{i\}} = \left\{ \begin{array}{ll} A_{\{i\}} + \sum\limits_{j=1}^{5} f_{Bj} B_{j\{i\}} & \text{for longitude and latitude} \\ \\ f_A A_{\{i\}} + \sum\limits_{j=1}^{5} f_{Bj} B_{j\{i\}} & \text{for distance} \end{array} \right. , \tag{2}$$

where f_A and f_{Bj} $(j = 1, 2, 3, 4, 5)$ are the paremeters described in the previous subsection. They are grouped together in the struct `Elp_facs` in `ElpMpp02.cpp`. Note that $B_{6\{i\}}$ are not used in the calculation. In addition, the amplitudes $A_{\{i\}}$ for longitude and latitude are in arcseconds in the original data files. I converted them to radians in the files `elp_main.long` and `elp_main.lat`.

The remaining 11 files contain the Poisson series for perturbations. Each file has 15 columns containing i_1, i_2, ..., i_{13}, A and ϕ_0. The series is summed according to the equation

$$\sum_{\{i\}} A_{\{i\}} \sin \phi \quad \{i\} = \{i_1, i_2, ..., i_{13}\}, \tag{3}$$

The phase ϕ is given by

$$\phi = \phi_0 + i_1 D + i_2 F + i_3 l + i_4 l' + i_5 Me + i_6 Ve + i_7 EM + i_8 Ma + i_9 Ju + i_{10} Sa + i_{11} Ur + i_{12} Ne + i_{13} \zeta, \tag{4}$$

where the 13 arguments D, F, ... will be described in the next subsection. In the longitude and latitude files, the amplitudes $A_{\{i\}}$ are in radians/cy^n. In all perturbation files, The initial phase $\phi_{0\{i\}}$ are in radians.

The amplitudes $\tilde{A}_{\{i\}}$ in the main problem are fixed once the adjustable parameters fitted to LLR or DE405/DE406 are determined. The amplitudes $A_{\{i\}}$ in the Poisson series for perturbations are the same for the LLR and DE405/DE406 parameters. Hence, all coefficients are determined once LLR or DE405/DE406 are specified. The coefficients are calculated after the parameters in the previous subsection are calculated. They are grouped together in the struct `Elp_coefs` in `ElpMpp02.cpp`.

The coefficients are calculated in the subroutine
`setup_Elp_coefs(Elp_coefs &coefs, Elp_facs facs)`

1.3 Lunar and Planetary Arguments

The Moon and Earth-Moon arguments are given by

$$\begin{aligned} W_1 &= (218^\circ 18' 59.95571'' + \Delta W_1^{(0)}) + (1732559343.73604''/\text{cy} + \Delta W_1^{(1)})T \\ &\quad + (-6.8084''/\text{cy}^2 + \Delta W_1^{(2)})T^2 + (0.006604''/\text{cy}^3 + \Delta W_1^{(3)})T^3 \\ &\quad + (-0.00003169''/\text{cy}^4 + \Delta W_1^{(4)})T^4 \\ W_2 &= (83^\circ 21' 11.67475'' + \Delta W_2^{(0)}) + (14643420.3171''/\text{cy} + \Delta W_2^{(1)} + \delta W_2^{(1)})T \end{aligned} \tag{5}$$

$$
\begin{aligned}
& +(-38.2631''/\mathrm{cy}^2 + \Delta W_2^{(2)})T^2 + (-0.045047''/\mathrm{cy}^3 + \Delta W_2^{(3)})T^3 \\
& +(0.00021301''/\mathrm{cy}^4)T^4 \qquad (6)\\
W_3 &= (125^\circ 2' 40.39816'' + \Delta W_3^{(0)}) + (-6967919.5383''/\mathrm{cy} + \Delta W_3^{(1)} + \delta W_3^{(1)})T \\
& +(6.359''/\mathrm{cy}^2 + \Delta W_3^{(2)})T^2 + (0.007625''/\mathrm{cy}^3 + \Delta W_3^{(3)})T^3 \\
& +(-0.00003586''/\mathrm{cy}^4)T^4 \qquad (7)\\
Ea &= (100^\circ 27' 59.13885'' + \Delta T^{(0)}) + (129597742.293''/\mathrm{cy} + \Delta T^{(1)})T \\
& +(-0.0202''/\mathrm{cy}^2)T^2 + (9'' \times 10^{-6}/\mathrm{cy}^3)T^3 \\
& +(1.5'' \times 10^{-7}/\mathrm{cy}^4)T^4 \qquad (8)\\
\varpi' &= (102^\circ 56' 14.45766'' + \Delta \varpi'^{(0)}) + (1161.24342''/\mathrm{cy})T + (0.529265''/\mathrm{cy}^2)T^2 \\
& -(1.1814'' \times 10^{-4}/\mathrm{cy}^3)T^3 + (1.1379'' \times 10^{-5}/\mathrm{cy}^4)T^4, \qquad (9)
\end{aligned}
$$

where $T = (\mathrm{JD} - 2451545)/36525$ is the barycentric dynamical time (TDB) in Julian centuries from J2000.0

Delaunay arguments D, F, l and l' are given by

$$
\begin{aligned}
D &= W_1 - Ea + 180^\circ \qquad (10)\\
F &= W_1 - W_3 \qquad (11)\\
l &= W_1 - W_2 \qquad (12)\\
l' &= Ea - \varpi'. \qquad (13)
\end{aligned}
$$

These are the 4 arguments appearing in the ELP/MPP02 series for the main problem, and also the first 4 arguments appearing in the ELP/MPP02 series for perturbations. The remaining 9 arguments in the ELP/MPP02 series for perturbations are given by

$$
\begin{aligned}
Me &= 252^\circ 15' 3.216919'' + (538101628.66888''/\mathrm{cy})T \qquad (14)\\
Ve &= 181^\circ 58' 44.758419'' + (210664136.45777''/\mathrm{cy})T \qquad (15)\\
EM &= 100^\circ 27' 59.13885'' + (129597742.293''/\mathrm{cy})T \qquad (16)\\
Ma &= 355^\circ 26' 3.642778'' + (68905077.65936''/\mathrm{cy})T \qquad (17)\\
Ju &= 34^\circ 21' 5.379392'' + (10925660.57335''/\mathrm{cy})T \qquad (18)\\
Sa &= 50^\circ 4' 38.902495'' + (4399609.33632''/\mathrm{cy})T \qquad (19)\\
Ur &= 314^\circ 3' 4.354234'' + (1542482.57845''/\mathrm{cy})T \qquad (20)\\
Ne &= 304^\circ 20' 56.808371'' + (786547.897''/\mathrm{cy})T \qquad (21)\\
\zeta &= W_1 + (5028.79695''/\mathrm{cy})T. \qquad (22)
\end{aligned}
$$

These 13 arguments are grouped together in the struct `Elp_args` in `ElpMpp02.cpp`. They are calculated in the subroutine
`compute_Elp_arguments(double T, Elp_paras paras, Elp_args &args)`

1.4 Position of the Moon

The natural coordinate system in ELP/MPP02 theory is based on the mean ecliptic of date. In this coordinate system, the geocentric ecliptic longitude V, latitude U and distance r of the Moon are given by

$$V = W_1 + \mathrm{series}(\texttt{elp\_main.long}) + \mathrm{series}(\texttt{elp\_pert.longT0}) + \mathrm{series}(\texttt{elp\_pert.longT1}) \cdot T$$

$$+\text{series}(\texttt{elp\_pert.longT2})\cdot T^2+\text{series}(\texttt{elp\_pert.longT3})\cdot T^3 \quad (23)$$

$$U = \text{series}(\texttt{elp\_main.lat})+\text{series}(\texttt{elp\_pert.latT0})+\text{series}(\texttt{elp\_pert.latT1})\cdot T +\text{series}(\texttt{elp\_pert.latT2})\cdot T^2 \quad (24)$$

$$r = r_{a0}\cdot[\text{series}(\texttt{elp\_main.dist})+\text{series}(\texttt{elp\_pert.distT0})+\text{series}(\texttt{elp\_pert.distT1})\cdot T +\text{series}(\texttt{elp\_pert.distT2})\cdot T^2+\text{series}(\texttt{elp\_pert.distT3})\cdot T^3], \quad (25)$$

where $r_{a0}=a_0(\text{DE405})/a_0(\text{ELP})=384747.961370173/384747.980674318$. The rectrangular coordinates X, Y, Z with respect to the mean ecliptic and equinox of J2000.0 are given by

$$\begin{pmatrix} X\\ Y\\ Z\end{pmatrix}=\begin{pmatrix} 1-2P^2 & 2PQ & 2P\sqrt{1-P^2-Q^2}\\ 2PQ & 1-2Q^2 & -2Q\sqrt{1-P^2-Q^2}\\ -2P\sqrt{1-P^2-Q^2} & 2Q\sqrt{1-P^2-Q^2} & 1-2P^2-2Q^2\end{pmatrix}\begin{pmatrix} r\cos V\cos U\\ r\sin V\cos U\\ r\sin U\end{pmatrix}, \quad (26)$$

where

$$P = 0.10180391\times10^{-4}T+0.47020439\times10^{-6}T^2-0.5417367\times10^{-9}T^3 -0.2507948\times10^{-11}T^4+0.463486\times10^{-14}T^5 \quad (27)$$

$$Q = -0.113469002\times10^{-3}T+0.12372674\times10^{-6}T^2+0.1265417\times10^{-8}T^3 -0.1371808\times10^{-11}T^4-0.320334\times10^{-14}T^5. \quad (28)$$

Rectangular coordinates X, Y and Z are computed in `ElpMpp02.cpp` in the subroutine `getX2000(double T, Elp_paras &paras, Elp_coefs &coefs, double &X, double &Y, double &Z)`

1.5 Velocity of the Moon

If rectangular components of velocity $\dot{X}$, $\dot{Y}$ and $\dot{Z}$ are also required, call this subroutine instead: `getX2000_Xdot2000(double T, Elp_paras &paras, Elp_coefs &coefs, double (&Xvec)[6])`. The first three components of the array `Xvec` stores X, Y and Z. The last three components store $\dot{X}$, $\dot{Y}$ and $\dot{Z}$. Note that the computation of velocity doubles the computation time and should be avoided if velocity is not needed.

1.6 Test Cases

Test cases are provided in the pdf document on this ftp site. Here I provide another sets of cases for code test.

Lunar geocentric rectangular coordinates (ecliptic and equinox of J2000.0) using parameters fitted to LLR.

TDB JD	TDB Date, Time (YYYY-MM-DD, HH:MM:SS)	X (km)	Y (km)	Z (km)
2444269.5	1980-01-31, 00:00:00	-186813.01288	349310.13512	-19003.33883
2446269.7	1985-07-23, 04:48:00	-367970.07950	-45234.88375	20221.87153
2448269.9	1991-01-13, 09:36:00	-38942.82455	-403238.94206	-20800.77410
2450270.1	1996-07-05, 14:24:00	357372.04971	-89978.49535	14501.18831
2452270.3	2001-12-26, 19:12:00	252208.00739	294433.40162	-21940.36333

Lunar geocentric rectangular coordinates (ecliptic and equinox of J2000.0) using parameters fitted to DE405/DE406.

TDB JD	TDB Date, Time (YYYY-MM-DD, HH:MM:SS)	X (km)	Y (km)	Z (km)
2521835.67	2192-06-13, 04:04:48	-184108.21468	345893.25529	30395.06868
2265621.33	1490-12-07, 19:55:12	-298024.37832	-213909.67132	-23263.21426
2009406.99	0789-06-16, 11:45:36	350041.24745	-201093.28987	1643.33539
1753192.65	0087-12-25, 03:36:00	90272.39894	351997.39617	13417.01712
1496978.31	-0614-07-03, 19:26:24	-403018.01560	-2639.93889	-28463.89733

More test cases can be generated by modifying the C++ code `example.cpp` or using this JavaScript calculator (or this JavaScript calculator if velocity is also needed).

2 Truncated ELP/MPP02 Series

2.1 Implementation

The ELP/MPP02 series can be written in the form

$$V(T) = W_1(T) + \sum_{i=0}^{3} T^i \sum_j A_{ij}^{(V)} \sin\left[\phi_{ij}^{(V)}(T)\right] \quad (29)$$

$$U(T) = \sum_{i=0}^{2} T^i \sum_j A_{ij}^{(U)} \sin\left[\phi_{ij}^{(U)}(T)\right] \quad (30)$$

$$r(T) = \sum_{i=0}^{3} T^i \sum_j A_{ij}^{(r)} \sin\left[\phi_{ij}^{(r)}(T)\right], \quad (31)$$

where $A_{ij}^{(U)}$, $A_{ij}^{(V)}$, $A_{ij}^{(r)}$ are constant amplitudes once the adjustable parameters are chosen. The phase angles $\phi_{ij}^{(U)}$, $\phi_{ij}^{(V)}$, $\phi_{ij}^{(r)}$ are linear functions of 13 arguments (see Section 1.2).

I construct a truncated series by the following procedure:

- Choose four parameters $A_{\rm th}^{(U)}$, $A_{\rm th}^{(V)}$, $A_{\rm th}^{(r)}$ and τ.
- Drop the terms in the series with $A_{ij}^{(U)} < A_{\rm th}^{(U)}/\tau^i$, $A_{ij}^{(V)} < A_{\rm th}^{(V)}/\tau^i$, and $A_{ij}^{(r)} < A_{\rm th}^{(r)}/\tau^i$.

It is clear that the smaller the parameters $A_{\rm th}^{(U)}$, $A_{\rm th}^{(V)}$, and $A_{\rm th}^{(r)}$, the closer the truncated series is to the original series. The parameter τ has a unit of time and should be chosen to cover the time span of interest.

The truncation is implemented in the file `ElpMpp_trim.cpp` by the subroutine
`trim_Elp_coefs(Elp_coefs &coefs, Elp_coefs &coefs_trim, double AthU, double AthV, double AthR, double tau)`
where `coefs` is a struct containing the coefficients of the full series. Coefficients of the truncated series will be written to the struct `coefs_trim`. This new struct can be passed to the function `getX2000()` to compute the Moon's position using the truncated series.

2.2 Accuracy of Truncated Series

One simple way to estimate the accuracy of a truncated series is to calculate the sum of the absolute value of the amplitudes of the dropped terms. This gives the maximum deviation the

truncated series could be from the full series. Suppose the truncated series is to be used in the time span $T_1 < T < T_2$, the maximum possible deviations between the full and truncated series are

$$\Delta V_{\rm max} = \sum_{i=0}^{3} T_{\rm max}^i \sum_{\rm dropped\ A_{ij}^{(V)}} |A_{ij}^{(V)}| \tag{32}$$

$$\Delta U_{\rm max} = \sum_{i=0}^{2} T_{\rm max}^i \sum_{\rm dropped\ A_{ij}^{(U)}} |A_{ij}^{(U)}| \tag{33}$$

$$\Delta r_{\rm max} = \sum_{i=0}^{3} T_{\rm max}^i \sum_{\rm dropped\ A_{ij}^{(r)}} |A_{ij}^{(r)}|, \tag{34}$$

where $T_{\rm max} = \max(|T_1|, |T_2|)$. These estimated deviations are very conservative because in general the phases in the terms are different and the terms do not have the same sign. Root mean square deviations may provide better estimates of typical deviations at any given time between T_1 and T_2:

$$rms(\Delta q) = \sqrt{\langle \Delta q^2(T) \rangle} = \sqrt{\left\langle \sum_{i=0}^{3}\sum_{j=0}^{3} T^{i+j} \sum_{\rm dropped\ A_{ik}^{(q)}} \sum_{\rm dropped\ A_{jm}^{(q)}} A_{ik}^{(q)} A_{jm}^{(q)} \sin\phi_{ik}^{(q)}(T) \sin\phi_{jm}^{(q)}(T) \right\rangle}, \tag{35}$$

where $q = U, V, r$ and $\langle\rangle$ denotes time average. Assume that the time average of the cross terms are small, which might not be true, the expression may be simplified to

$$rms(\Delta q) \approx \sqrt{\sum_{i=0}^{3} \frac{T_2^{2i+1} - T_1^{2i+1}}{2(2i+1)(T_2 - T_1)} \sum_{\rm dropped\ A_{ij}^{(q)}} \left(A_{ij}^{(q)}\right)^2}, \tag{36}$$

where I have used the approximation that

$$\left\langle T^{2i} \sin^2\phi_{ij}^{(q)} \right\rangle \approx \left\langle T^{2i} \right\rangle \left\langle \sin^2\phi_{ij}^{(q)}(T) \right\rangle \approx \frac{T_2^{2i+1} - T_1^{2i+1}}{2(2i+1)(T_2 - T_1)}. \tag{37}$$

The validity of this approximation is also uncertain. There is also an implicit assumption that any time between T_1 and T_2 is equally likely to be chosen. Whether or not this assumption is valid depends on specific applications. For example, one may want to use a truncated series to calculate lunar positions between 1900 and 2100 ($T_1 = -1$, $T_2 = 1$) most of the time, but occasionally may want to calculate the positions in ancient times. If this is the case, one should choose multiple pairs of (T_1, T_2) and estimate the accuracy of the truncated series in those time intervals.

A more reliable way of estimating the accuracy is to perform a Monte Carlo simulation, in which values of V, U, r are calculated from both the full series and truncated series at n randomly chosen times between T_1 and T_2. The deviations are then estimated according to the equations

$$\Delta q_{\rm max} \approx \max(|\Delta q_1|, |\Delta q_2|, \cdots, |\Delta q_n|) \tag{38}$$

$$rms(\Delta q) \approx \sqrt{\frac{1}{n}\sum_{i=0}^{n} (\Delta q_i)^2}, \tag{39}$$

where

$$\Delta q_i = q_{\text{full series}}(T_i) - q_{\text{truncated series}}(T_i) = \sum_{k=0}^{3} T_i^k \sum_{\text{dropped } A_{kj}^{(q)}} A_{kj}^{(q)} \sin \phi_{kj}^{(q)}(T_i). \tag{40}$$

The crude error estimates from equations (32), (33), (34) and (36) are implemented in `ElpMpp_trim.cpp` in subroutine

`trim_Elp_coefs_errorEst(Elp_coefs &coefs, Elp_coefs &coefs_trim, Elp_coefs &coefs_drop, double AthU, double AthV, double AthR, double tau, double T1, double T2, devStats &stats)`

where `coefs_drop` stores the dropped coefficients and `devStats` is a struct grouping the variables for $\Delta q_{\max}$ and $rms(\Delta q)$.

The error estimate based on a Monte Carlo (MC) simulation is implemented in `ElpMpp_trim.cpp` in subroutine

`error_est(Elp_coefs &coefs_drop, Elp_paras &paras, int n, double T1, double T2, devStats &errStats)`

I set $n = 10000$ for a typical run to obtain decent statistics. It may take a while to complete the simulation, depending on your processor speed. I use the `rand()` function in `<cstdlib>` to generate random numbers. In my Linux g++ compiler, `rand()` generates pseudo random integers between 0 and 2147483647. A seed is currently set using `srand(4852618)` in the code to ensure reproducibility. It is not clear to me that `rand()/2147483647` will produce random numbers uniformly distributed between 0 and 1, but the result I get from using `rand()` seems to agree with that obtained from pseudo random numbers generated by R's `runif()` function, which generates pseudo random numbers uniformly distributed between 0 and 1.

The following are examples of error estimates computed from the two subroutines.

In all cases, $\tau = 50$, $T_1 = -50$, $T_2 = 10$ and parameters fitted to DE405/DE406 are used. N is the number of terms in the truncated ELP/MPP02 series. $\Delta q_{\max}$ and $rms(\Delta q)$ $(q = V, U, r)$ are computed from equations (32), (33), (34) and (36).

$A_{\text{th}}^{(U)}$	$A_{\text{th}}^{(V)}$	$A_{\text{th}}^{(r)}$	N	$\Delta V_{\max}$	$rms(\Delta V)$	$\Delta U_{\max}$	$rms(\Delta U)$	$\Delta r_{\max}$	$rms(\Delta r)$
$30''$	$30''$	100 km	42	$422''$	$45.4''$	$209''$	$32.5''$	462 km	84.0 km
$10''$	$10''$	20 km	69	$242''$	$20.5''$	$137''$	$17.6''$	282 km	30.5 km
$1''$	$1''$	2 km	187	$74''$	$2.95''$	$38.6''$	$3.34''$	60.2 km	3.96 km
$0.001''$	$0.001''$	0.1 km	3759	$1.47''$	$0.016''$	$0.80''$	$0.015''$	16.3 km	0.42 km

Same as above, but $\Delta q_{\max}$ and $rms(\Delta q)$ are estimated by Monte Carlo simulations using equations (38) and (39) with $n = 10000$. All $\Delta q_{\max}$ and $rms(\Delta q)$ are rounded to two significant figures, which is about the accuracy of the estimates using $n = 10000$.

$A_{\text{th}}^{(U)}$	$A_{\text{th}}^{(V)}$	$A_{\text{th}}^{(r)}$	N	$\Delta V_{\max}$	$rms(\Delta V)$	$\Delta U_{\max}$	$rms(\Delta U)$	$\Delta r_{\max}$	$rms(\Delta r)$
$30''$	$30''$	100 km	42	$230''$	$48''$	$150''$	$34''$	340 km	86 km
$10''$	$10''$	20 km	69	$100''$	$21''$	$98''$	$19''$	150 km	31 km
$1''$	$1''$	2 km	187	$17''$	$3.0''$	$18''$	$2.4''$	17 km	4.3 km
$0.001''$	$0.001''$	0.1 km	3759	$0.093''$	$0.016''$	$0.058''$	$0.012''$	2.0 km	0.42 km

Not surprisingly, $\Delta q_{\max}$ calculated by (32), (33), (34) are always larger than those estimated by MC, but the estimates for $rms(\Delta q)$ by both methods turn out to be quite close for the cases considered.

Note that the accuracy is estimated by comparing the truncated series to the full series. One might want to compare the truncated series to other data such as JPL's ephemerides. The Monte Carlo subroutine can be modified to make that comparison. In the following, I show results of comparison between the truncated series and JPL's DE406 and DE431 in the time intervals $-50 < T < 10$.

In all cases, $\tau = 50$, $T_1 = -50$, $T_2 = 10$ and parameters fitted to DE405/DE406 are used. The series is now compared to JPL's DE406 using Monte Carlo method with $n = 10000$. The case with $A_{\rm th}^{(U)} = A_{\rm th}^{(V)} = A_{\rm th}^{(r)} = 0$ is the untruncated series.

$A_{\rm th}^{(U)}$	$A_{\rm th}^{(V)}$	$A_{\rm th}^{(r)}$	N	$\Delta V_{\rm max}$	$rms(\Delta V)$	$\Delta U_{\rm max}$	$rms(\Delta U)$	$\Delta r_{\rm max}$	$rms(\Delta r)$
$30''$	$30''$	100 km	42	$230''$	$47''$	$150''$	$34''$	370 km	86 km
$10''$	$10''$	20 km	69	$97''$	$21''$	$100''$	$19''$	150 km	31 km
$1''$	$1''$	2 km	187	$15''$	$2.9''$	$14''$	$2.4''$	17 km	4.3 km
$0.001''$	$0.001''$	0.1 km	3759	$3.5''$	$0.56''$	$0.59''$	$0.10''$	2.6 km	0.46 km
0	0	0	35901	$3.5''$	$0.56''$	$0.59''$	$0.10''$	1.3 km	0.19 km

Same as the previous table but the comparison is to JPL's DE431.

$A_{\rm th}^{(U)}$	$A_{\rm th}^{(V)}$	$A_{\rm th}^{(r)}$	N	$\Delta V_{\rm max}$	$rms(\Delta V)$	$\Delta U_{\rm max}$	$rms(\Delta U)$	$\Delta r_{\rm max}$	$rms(\Delta r)$
$30''$	$30''$	100 km	42	$240''$	$48''$	$150''$	$34''$	340 km	86 km
$10''$	$10''$	20 km	69	$100''$	$21''$	$99''$	$19''$	150 km	31 km
$1''$	$1''$	2 km	187	$19''$	$3.7''$	$18''$	$2.4''$	17 km	4.3 km
$0.001''$	$0.001''$	0.1 km	3759	$12.5''$	$2.5''$	$1.5''$	$0.33''$	3.4 km	0.53 km
0	0	0	35901	$12.5''$	$2.5''$	$1.5''$	$0.33''$	2.6 km	0.33 km

2.3 Saving the Truncated Series

If you are happy with the truncated series, you may want to save the coefficients of the truncated series so that you don't have to recalculate them every time. A subroutine is provided in `ElpMpp_trim.cpp` to output the coefficients to 14 data files:
`output_data_files(const char *dataFileSuffix, Elp_coefs &coefs)`
where `coefs` is the struct that stores the truncated coefficients. All file names have the form `elp_MMMM.[X]dataFileSuffix`, where MMMM is main or pert, [X] can be long, lat, dist, longT0, longT1, longT2, longT3, latT0, latT1, latT2, distT0, distT1, distT2, distT3. `dataFileSuffix` contains characters appended to the file name. It provides a shorthand notation for the truncated series.

The format of the 14 files are almost the same as before. The first line is an integer indicating the number of terms in the series. The three files for the main problem each has 5 columns containing i_1, i_2, i_3, i_4 and $\tilde{A}$. There is no need to calculate $\tilde{A}$ from A and B_i since parameters have been chosen to fit either LLR or DE405/DE406 when constructing the truncated series. The remaining 11 files for perturbations are the same as before: 15 columns containing i_1, i_2, ..., i_{13}, A and ϕ_0.

`ElpMpp_trim.cpp` also has a subroutine that generates a C++ code to compute the truncated series based on the 14 data files created by `output_data_files()`. The subroutine is
`generate_cpp_code(const char* outfile, const char *dataFileSuffix, int corr, double AthU, double AthV, double AthR, double tau, Elp_paras &paras)`
The file name of the C++ code is given by the character array `outfile`.

The file `example_usingElpMpp_trim.cpp` provides an example of using `ElpMpp_trim.cpp` to generate a truncated ELP/MPP02 series, estimate its accuracy, save coefficients to data files and create a C++ code.

3 JavaScript Code Generation

JavaScript is convenient for HTML-based applications, such as my local star charts and equatorial star charts pages. The file `ElpMpp_JavaScript.cpp` contains C++ subroutines that can generate JavaScript functions to compute a truncated ELP/MPP02 series. Instruction for the use of this code is at the beginning of the file. The two main subroutines are

```
generate_javascript_code(const char* outfile, int corr, double AthU, double AthV, double AthR, double tau, Elp_paras &paras, Elp_coefs &coefs, const char* funSuffix)
```

and

```
generate_javascript_code_min(const char* outfile, Elp_paras &paras, Elp_coefs &coefs, const char *funSuffix)
```

If Moon's velocity is also needed, call these two subroutines instead:

```
generate_javascript_code_with_velocity(const char* outfile, int corr, double AthU, double AthV, double AthR, double tau, Elp_paras &paras, Elp_coefs &coefs, const char* funSuffix)
```

and

```
generate_javascript_code_with_velocity_min(const char* outfile, Elp_paras &paras, Elp_coefs &coefs, const char *funSuffix)
```

`outfile` is file name of the js file that will be outputted. Characters in `funSuffix` will be added to every JavaScript function. This is useful to prevent conflicts with code that implements other versions of the truncated series. The subroutine `generate_javascript_code()` generates a human-readable js file. It is intended for your reference only. The subroutine `generate_javascript_code_min()` generates a minified version of the js file, in which comments and unnecessary white spaces are stripped to optimize performance. The js files generated by the two subroutines provide identical functions.

Unlike the C++ code generator, no data files will be outputted. The ELP/MPP02 series are written expliticly in the outputted js file instead of using loops. JavaScript is an interpreted language, not a compiled language. So it cannot take advantage of any loop parallization that may be available through compilers. In addition, in the computation of phases such as in equation (4), many of the i's are zeros. When writing out the series explicitly, all the zero i's are removed to speed up computation. In fact, writing out the series explicitly may also benefit a C++ code, especially when the truncated series contains less than 1000 terms. The subroutines above can be modified easily to turn into a C++ code generator.

This JavaScript calculator uses two js files `ElpMpp02LLR_min.js` and `ElpMpp02DE_min.js` to compute the ELP/MPP02 series. Only Moon's position is calculated here. This calculator uses two js files `ElpMpp02LLRv_min.js` and `ElpMpp02DEv_min.js` to compute both position and velocity. They are minified Javascript files generated using subroutines in `ElpMpp_JavaScript.cpp` by setting $A_{\rm th}^{(V)} = A_{\rm th}^{(U)} = A_{\rm th}^{(r)} = 0$ and using parameters fitted to LLR and DE405/DE406.

The C++ file `example_usingElpMpp_JavaScript.cpp` provides an example of using `ElpMpp_JavaScript.cpp` to generate JavaScript functions to compute a truncated ELP/MPP02 series.