# 太阳系典型行星历表的统一化读取与比较

张明睿

**摘要：**

**关键词：**

**DE历表，INPOP历表，EPM历表**

## 引言

大行星历表的构建是一个基本的动力学时空框架工作，深受各个天文与航空航天大国的重视，其用途极为广泛。从早期的观天授时，到近现代的深空探测，都需要太阳、月球和行星的精确空间位置信息作为支撑。

来自不同国家的多个团队都开展过月球和大行星历表的独立研究和构建。目前，世界上使用非常广泛的、精度很高的数值星历表有三个，包括美国喷气推进实验室研发的DE系列行星历表、法国巴黎天文台研发的INPOP系列行星历表以及俄罗斯科学院应用天文研究所研发的EPM系列行星历表。在三种行星历表中，美国的DE历表因开发的时间比较早而被广泛使用，除了用于不同国家的月球深空探测任务之外，还被嵌入各类天体测量、卫星导航等需要高精度计算行星位置的软件中。但是这并不能说明DE历表比INPOP和EPM历表更精确；由于观测数据的公开化，INPOP历表和EPM历表在使用的数据和模型上与DE历表基本相同，只是在处理方法上有些差别，三者的精度是很接近的。由于DE历表进入国内较早，大量航空航天程序均使用DE历表，其接口格式都为DE历表所规定的接口格式，这导致其难以使用INPOP历表和EPM历表。近年来，由于中美关系变化，我们已经不能轻松下载并使用新版DE历表，因此我们需要寻求DE历表的替代品，那就是与之精度相近的INPOP历表和EPM历表。本文尝试统一DE系列行星历表、INPOP行星历表和EPM行星历表的接口格式为最新版本的DE历表——DE436所规定的格式，令以前使用DE历表的程序同样可以很方便地使用INPOP历表和EPM历表正常工作。

## 第一节：DE历表简介

DE436历表是最新版本DE历表，其对之前的DE历表做了精度校正，也加入了之前DE历表所不具备的一些功能。

DE历表主要由四部分构成，其分别为头文件（header文件）、数据文件（asc文件）、测试文件（testpo文件）和程序文件（.f文件）。其中，头文件中包含了DE历表的单位数据长度、起止时间、常数值和插值系数；数据文件包含具体的，记载大行星位置、章动、月球天平动等的数据；测试文件包含了一些参考结果，可以测试生成的二进制历表文件的正确性；程序文件用来生成二进制历表文件，并且对其进行测试。

DE历表的使用方法如下：

1. 访问喷气推进实验室官方网站，下载DE436历表（所有以.436结尾的文件，包括header.436，所有的ascp\*.436和testpo.436）和所需的fortran程序文件（包括asc2eph.f和testeph.f）。
2. 将其中header.436文件和所有ascp\*.436文件拼接成为in文件。
3. 使用asc2eph.f将in文件转换为二进制格式的JPLEPH文件。
4. JPLEPH文件已经可以使用。可以使用testeph.f和testpo.436测试生成的JPLEPH文件。

testeph.f提供许多函数，其中最重要的函数是PLEPH函数，可以用其查询在国际天球参考架下，给定儒略日T时指定天体A相对于指定天体B的相对位置和速度。下表给出testeph.f提供的所有函数和其功能

表 1 testeph.f提供的函数

|  |  |  |
| --- | --- | --- |
| **函数** | **功能** | **说明** |
| PLEPH | 计算天体之间的相对位置和速度 |  |
| READHD | 读取二进制历表的表头 |  |
| INTCHB | 插值函数，计算两条记录中间时刻的天体位置 |  |
| DPLEPH | 可以提供小数的时间 | PLEPH的另一入口 |
| CONST | 读取常数 | PLEPH的另一入口 |
| PL\_UNITS | 确定输出的单位 | PLEPH的另一入口 |

INPOP历表和EPM历表都有和DE历表格式相似的文件，但是其内部存在差别。不同版本的DE历表内部也存在细微差别。为了统一这些差别，我们特此撰写本程序，可以用同样的步骤来针对不同的历表生成可以被testeph.f读取的二进制文件。

## 第二节：统一工作

为了统一三大行星历表的接口为DE历表所规定的格式，首先我们需要了解DE历表的组成。DE历表由表头和若干组（GROUP）组成，按照顺序依次如下表：

表 2 DE历表结构

|  |  |  |
| --- | --- | --- |
| **GROUP编号** | **含义** | **说明** |
| 无 |  | 每一条记录的长度 |
| 1010 | 标题 |  |
| 1030 | 时间轴 | 历表的开始时间、结束时间和两条记录之间的时间差 |
| 1040 | 常数名 |  |
| 1041 | 常数值 |  |
| 1050 | 系数 | 拟合时所用的插值多项式系数 |
| 1070 | 具体记录 | 历表主体 |

不同版本的DE历表，以及INPOP和EPM提供的类似DE格式的历表，其差别主要有以下几点：

1. 每一条记录的长度不同。

2. 常数个数不同。

3. 插值多项式系数个数不同。

为了为不同的历表生成DE历表格式的二进制历表文件，我们需要统一其中的差别。由于fortran语言对文件格式要求非常严格，我们不便于直接修改fortran程序来适应所有情况，因此我们使用C++语言重新编写了asc2eph.f，使用std::cin进行读入，忽略历表内的具体格式，得到了与asc2eph.f功能一样的asc2eph.cpp。该程序按以下步骤工作：

表 3 asc2eph.cpp工作流程

|  |  |
| --- | --- |
| **步骤** | **说明** |
| 读入每一条记录的长度 | 消除空格的差异 |
| 读入历表的标题 | 根据是否读到下一GROUP判断是否读完，消除标题长度的差异 |
| 读入时间轴 | 消除空格的差异 |
| 读入常数名 | 消除空格的差异；消除常数个数的差异 |
| 读入常数值 | 消除空格的差异 |
| 读入插值多项式系数 | 根据是否读到下一GROUP判断是否读完，消除系数个数不同的差异 |
| 将以上读到的内容写入目标文件 | 需要用户自定义目标文件名 |
| 依次读入每一条记录并写入目标文件 |  |

每一条记录的长度需要用户输入，也可以通过用户提供的header文件分析出来，其余的因素均可以通过给定的文件分析出来。具体实现详见asc2eph.cpp。

## 第三节：环境准备

运行本项目之前需要先配置环境并下载程序源代码和所需的历表文件。本项目包含C++、Fortran、Python3三种编程语言，因此需要配置三种编程语言运行所需要的环境。这里我们选用GCC（GNU Compiler Collections）运行C++和Fortran。现分别说明如下（注：后文中出现的`XX`表示需要在命令行窗口中执行XX，并且注意执行时不要键入`）。：

1. 若你使用类Unix操作系统（Mac OS X、Ubuntu、Fedora等），则直接在命令行窗口内依次输入`sudo XXX install gcc`，`sudo XXX install python3`即可完成安装。根据你的操作系统将XXX替换为以下内容：

表 4 类Unix系统安装工具

|  |  |
| --- | --- |
| Mac OS X | brew |
| Ubuntu | apt-get |
| Fedora | yum |

注：若你使用Mac OS X操作系统，你可能需要先下载HomeBrew以使用brew。下载HomeBrew只需要在命令行窗口内输入`/usr/bin/ruby -e "$(curl -fsSL [https://raw.githubusercontent.com/Homebrew/install/master/install)"`](https://raw.githubusercontent.com/Homebrew/install/master/install)%22%60)即可。

2. 若你使用Windows操作系统，则需要手动下载对应文件并配置环境，现分述如下：

2.1 下载GCC

下载MinGW-W64。MinGW-W64是一个运行在Windows上的移植版GCC，其有多个版本。MinGW-W64可以在SourceForge下载，建议选择x86\_64-win32-sjlj-rev0来运行本程序。

2.2 安装GCC

直接解压下载所得到的压缩包即可。（假设你将MinGW-W64解压到了mingw-w64/目录下）。

2.3 下载安装Python3

访问Python官方网站，下载最新版Python3。下载时选择executable installer即可。

2.4 安装Python3

运行下载的可执行程序，全部按照默认的配置进行安装即可（假设你将Python安装到了Python/目录下）。

2.5 配置环境

同时按下<Windows徽标键>和<R>启动运行，在运行窗口中输入sysdm.cpl打开系统属性页面。在系统属性页面中选择“高级”，点击“环境变量”，在“系统变量”中找到Path变量，对其进行编辑。对于Windows10操作系统，直接在弹出窗口中点“新建”，并输入“mingw-w64\bin”；再点击“新建”，并输入“Python”；最后点击“确定”即可。对于其他版本的Windows操作系统，在原有的Path变量的内容最后追加“;mingw-w64\bin;Python”即可。（注：需要将mingw-w64和Python换成你自己安装时文件所在的位置）。

3. 在你的电脑上将环境配置好后，打开新的命令行窗口，分别键入`gcc –version`和`python --version`，若均能输出，说明环境配置完成。

环境配置完成后，需要下载本项目源代码。本项目完整源代码已经托管到github，可以从github上直接下载。若已经安装git，也可以通过在命令行窗口内输入`git clone [https://github.com/frokaikan/bysj.git`](https://github.com/frokaikan/bysj.git%60)来下载。

4. 配置好环境后，需要下载对应的历表。每个历表所需下载的DE格式文件如下表所示：

表 5 所需要的历表文件

|  |  |
| --- | --- |
| DE（以DE436为例） | 1. header.436  2. ascp\*.436  3. testpo.436 |
| INPOP（以INPOP17a为例） | 1. inpop17a\_TDB\_m1000\_p1000\_littleendian.dat或inpop17a\_TDB\_m1000\_p1000\_bigendian.dat  2. inpop17a\_TDB\_m1000\_p1000.header  3. testpo.INPOP17A\_TDB |
| EPM（以EPM17为例） | 1. header.17  2. epm2017\_de\_ascii.txt |

注：

1. INPOP直接提供编译好的二进制历表文件，我们只需对其进行检测。至于littleendian和bigendian应该选择哪一个，可以将命令行窗口cd到本项目所在目录，然后运行`python makeit.py endian`，根据输出即可判断。

2. EPM不提供testpo，因此无法利用testeph.exe（见后文）对其进行检测。

## 第四节：运行程序

当你配置好所需要的环境，并下载好所需要的历表后，你便可以运行本项目。具体步骤可以参阅本项目根目录下的README.md，也可以阅读以下的使用说明。

1. 打开命令行窗口，cd到你的工作目录。不要关闭这个窗口，以下所述所有有关命令行的操作均在这个窗口内执行。

2. 在你的工作目录下建立新目录source/，然后将你的历表文件重命名后移动到该目录下。按照你所选历表的不同，执行如下操作：

表 6 历表文件重命名方式

|  |  |
| --- | --- |
| 所选历表 | 具体操作 |
| DE | 1. 在header文件最后加上后缀.header  2. 在所有asc文件（以asc开头）最后加上后缀.asc。  3. 在testpo文件最后加上后缀.testpo |
| INPOP | 1. 在二进制历表文件最后加上后缀.EPH  2. 在header文件最后加上后缀.header  3. 在testpo文件最后加上后缀.testpo |
| EPM | 1. 在header文件最后加上后缀.header  2. 在asc文件最后加上后缀.asc（EPM历表仅有一个asc历表文件）  3. 在testpo文件最后加上后缀.testpo |

3. 在命令行窗口内键入`python makeit.py make -n MY\_EPH`，将MY\_EPH替换为你想生成的二进制历表文件名称。注意，该名称必须以“EPH”结尾，若使用INPOP历表，则该文件名必须和提供的INPOP历表的文件名相同（使用INPOP历表时可以不写-n MY\_EPH）。如果一切顺利的话，你可以在工作目录下找到output/文件夹，其中包含所生成的二进制历表文件和一些中间文件，如下表：

表 7 本项目输出文件

|  |  |
| --- | --- |
| asc2eph\_struct.o | asc2eph.exe所需中间文件 |
| asc2eph.exe | 将full\_asc.in转换为二进制历表文件的可执行程序（可单独使用） |
| full\_asc.in | 拼接后的纯文本历表文件。会将Fortran格式的浮点数(如1.2D+03)转换为C格式浮点数(如1.2E+03) |
| fsizer3.o | testeph.exe所需中间文件 |
| libeph.o | testeph.exe所需中间文件 |
| testeph.exe | 用来测试所生成二进制历表文件的可执行程序（可单独使用） |
| MY\_EPH | 二进制历表文件（可单独使用） |

注：fsizer3.o和libeph.o已经包含所有需要的查询函数，将其和含有PLEPH等函数的主程序链接在一起，即可生成可以执行的程序。

在执行过程中可能会出现一些错误，现在将目前发现的可能出现的错误列举如下，并给出错误原因：

表 8 可能出现的错误

|  |  |
| --- | --- |
| Invalid ksize | header文件不合法 |
| eph file must ends with EPH. | 用户输入的二进制历表文件名不以EPH结尾 |
| Invalid namfil | 用户输入的二进制历表文件名存在非法字符 |
| You can just set ONE operation in <make> <test> <clean> <endian> | 输入了makeit.py无法识别的命令。makeit.py仅能识别上述四个命令。 |
| Invalid input. Please check "source/" directory. | 未按要求提供输入文件，需要检查source/目录。 |
| Found eph : XXX, but namfil == XXX. | 在测试INPOP历表时，提供的二进制历表文件名和INPOP历表自身的名称不一致 |
| No .testpo file | 在检测历表文件时，source/目录内未找到testpo文件 |
| At most ONE .testpo file | source/目录内找到了多个testpo文件，最多只能提供一个。 |
| Unknwon error | 未知的错误，你可以把你的错误通过issue或邮件告诉我，我会帮你解决问题。 |

4. 如果需要测试所生成的二进制历表文件，可以在命令行窗口内输入`python makeit.py test`，会自动调用testeph.exe读取source/目录下的xxx.testpo对所生成的二进制历表文件进行测试。本步会将source/目录下的xxx.testpo复制到output/目录下，并将其重命名为TESTPO。

如果运行得到下图结果，



图 1 检测成功样例

并且下图所示输出之后有至少一行内容，



图 2 输出的头部

说明测试通过，生成的二进制历表文件是正确的。

如果运行过程中出现下图结果，



图 3 检测失败条目



图 4 检测失败

说明测试不通过，生成的二进制历表文件存在问题。需要检查source/文件夹中的文件是否符合规范，然后重新生成。

5. 如果需要清除所有所生成的文件，可以在命令行窗口内输入`python makeit.py clean`，会清除所有程序运行过程中生成的文件，将项目恢复为初始状态（不会清除source/文件夹中的内容）。

## 第五节：历表比较

DE历表以儒略日计量时间，为了使用DE历表，我们需要了解如何在公历和儒略日之间进行转换。

儒略日的本质是积日，从公元前4213年1月1日12:00 UTC+0到当前时刻所经过的天数，即为当前时刻对应的儒略日。经过数学推导，我们有：公元Y年M月D日12:00 UTC+0对应的儒略日JD为（下式中括号[]表示向下取整）：

我们计算2000年1月1日至2000年12月31日共366天的12:00 UTC+0时刻（儒略日2451545到2451910）的太阳相对于地球的位置和速度，对三个历表给出的位置矢量（单位：AU）和速度矢量（单位：AU/Day）分别作差，计算其最大模长，并以此衡量三个历表的差异大小。为此，我们在output/目录下编写cmp.cpp文件，调用fsizer3.o和libeph.o计算在该时刻的地日相对位置。由于cmp.cpp文件由C++语言编写，其不能直接与用fortran语言编写的fsizer3.o和libeph.o生成可执行文件，为生成cmp.exe可执行文件，我们把命令行窗口切换到output/文件夹下，并依次执行：

1. `g++ -O3 -std=gnu++14 -o cmp.o -c cmp.cpp`

2. `gfortran cmp.o libeph.o fsizer3.o -lstdc++-6 -o cmp.exe`

即可生成可执行文件cmp.exe（注意：切换历表时需要重新生成cmp.exe）。执行cmp.exe，其会将每日得出的结果写入当前目录下的ans.csv文件。以下六张图片给出三个历表的六个分量的具体值。（六个分量为太阳相对于地球的位置和速度在x，y，z轴上的投影）

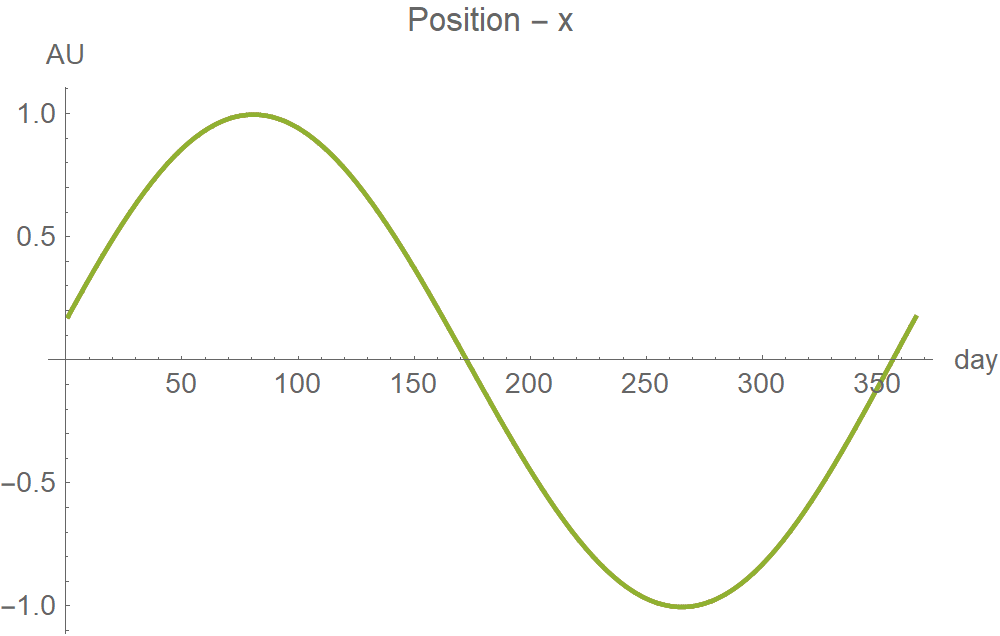


图 5 日-地相对位置，x轴投影

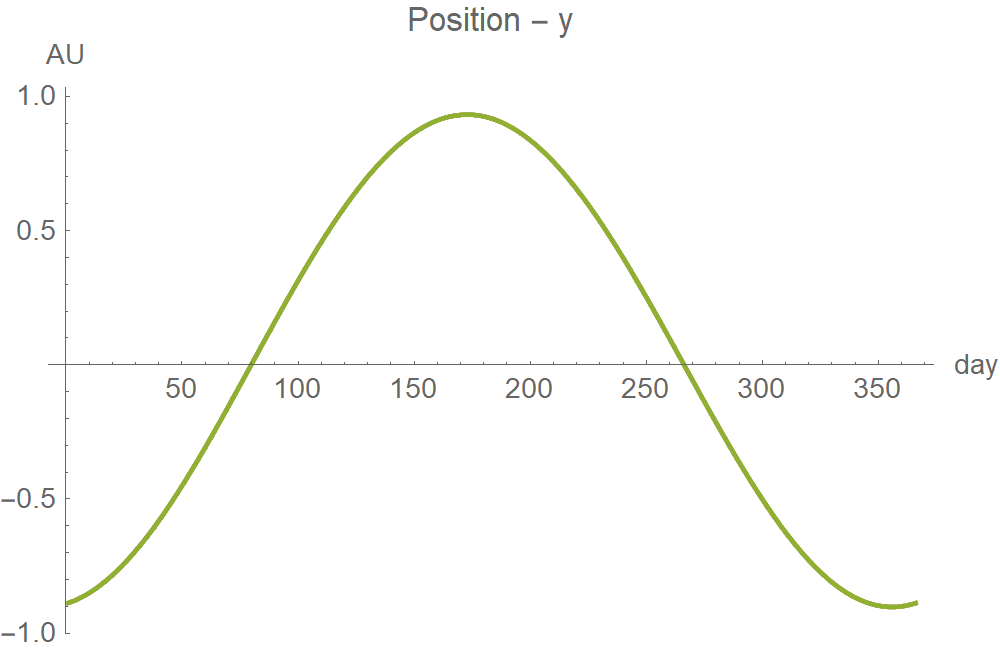


图 6 日-地相对位置，y轴投影

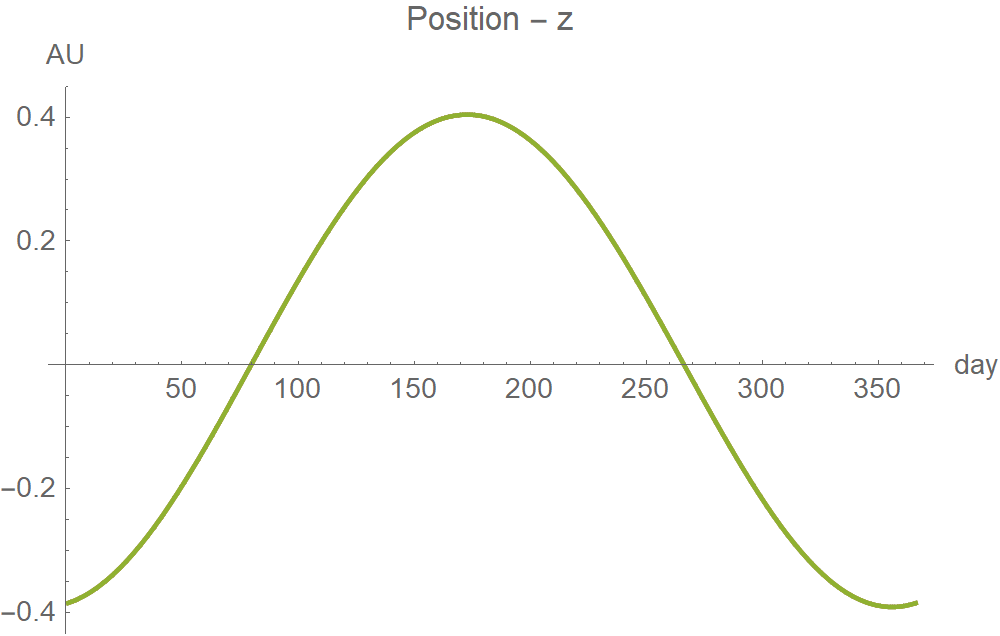


图 7 日-地相对位置，z轴投影

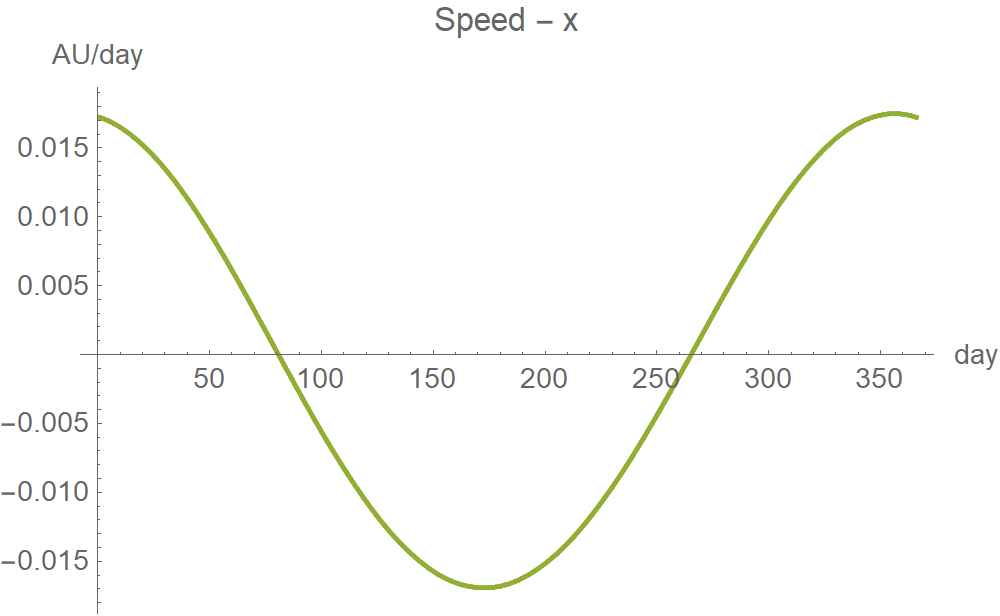


图 8 日-地相对速度，x轴投影

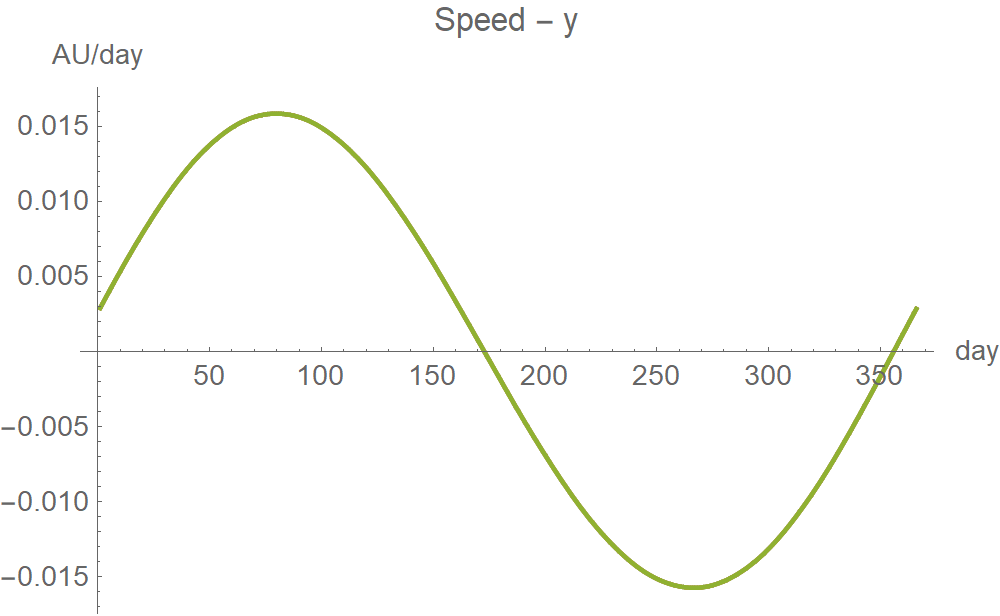


图 9 日-地相对速度，y轴投影

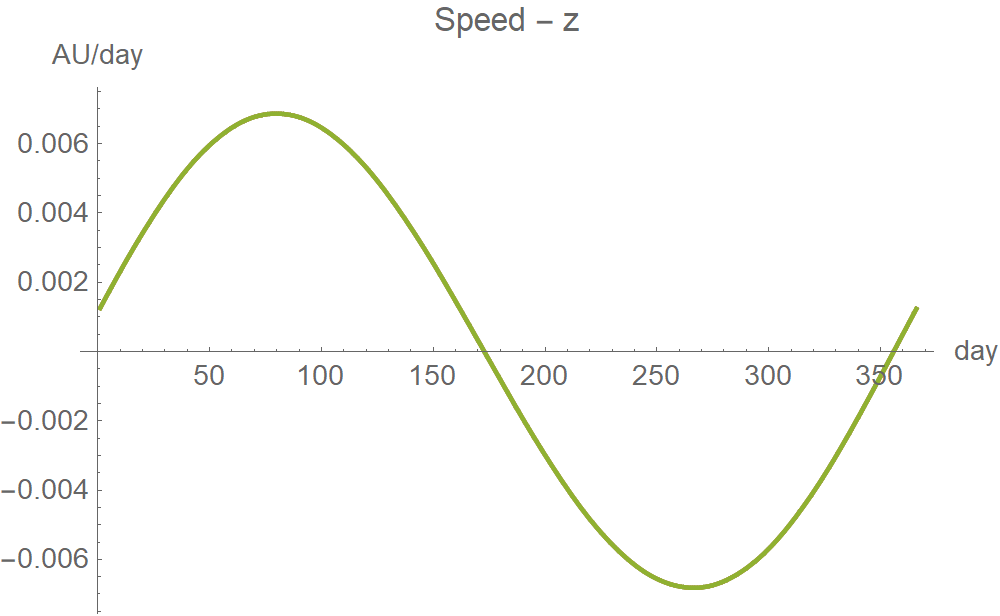


图 10 日-地相对速度，z轴投影

可以发现，六张图都只能看出一条线，这说明三个历表的精度都很高，进一步的，通过比较三个历表生成的ans.csv文件，我们可以得到如下结果：（左下部分给出速度矢量最大差值（单位：AU/day），右上部分给出位置矢量最大差值（单位:AU））

表 9 三个历表的差距

|  |  |  |  |
| --- | --- | --- | --- |
|  | DE436 | INPOP17a | EPM2017 |
| DE436 | ---------- | 1.44568\*10-9 | 1.60312\*10-9 |
| INPOP17a | 1.41421\*10-10 | ---------- | 1.67631\*10-9 |
| EPM2017 | 1.41421\*10-10 | 1.41421\*10-10 | ---------- |

可以发现，三个历表给出的结果只存在极其微小的差距。按照1AU=1.5\*108km计算，三个历表给出的相对位置偏差不超过100m，相对速度偏差不超过10-4m/s。

## 第六节：项目应用

当地球-月球-太阳三者共线，且月球位于地球和太阳中间时，月球便可以遮住一部分太阳，形成日食。如果以地球为原点建立坐标系，则月球的坐标

和太阳的坐标

应该满足

我们利用这一点来用三个历表预测日食。在2009年7月22日，中国南方可以观测到日全食。以上海为例，上海的食甚时刻在9:39:16，时区为UTC+8，换算成儒略日约为2455034.569。我们使用三个历表分别计算此刻月球和太阳相对于地球的位置和速度。为此，我们在output/目录下编写ask.cpp文件，调用fsizer3.o和libeph.o计算在该时刻的地月、地日相对位置。和cmp.cpp类似，由于ask.cpp文件也由C++语言编写，其也不能直接与用fortran语言编写的fsizer3.o和libeph.o生成可执行文件，为生成ask.exe可执行文件，我们把命令行窗口切换到output/文件夹下，并依次执行：

1. `g++ -O3 -std=gnu++14 -o ask.o -c ask.cpp`

2. `gfortran ask.o libeph.o fsizer3.o -lstdc++-6 -o ask.exe`

（即将上一步的cmp替换为ask）即可生成可执行文件ask.exe（注意：切换历表时也需要重新生成ask.exe）。用三个历表分别执行ask.exe，并输入2455034.569，结果如下表：

表 10 日食预测

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 历表 | | 地-日（AU） | 地-月（AU） | 比率 |
| DE | x | -0.4967962790 | -0.0011479920 | 432.7523922806 |
| y | 0.8131564507 | 0.0019209828 | 423.3022962541 |
| z | 0.3525268545 | -0.0033226357 | 420.3969785065 |
| INPOP | x | -0.4967972333 | -0.0011480281 | 432.7396204144 |
| y | 0.8131559545 | 0.0019209664 | 423.3056548287 |
| z | 0.3525266391 | 0.0008385458 | 420.4023665975 |
| EPM | x | -0.4967972348 | -0.0011480281 | 432.7396201743 |
| y | 0.8131559537 | 0.0019209664 | 423.3056551300 |
| z | 0.3525266388 | 0.0008385458 | 420.4023669959 |

我们发现，虽说三个历表在计算地-日，地-月相对位置上存在微小的差异，但三个历表得出的三个坐标比都很接近。换言之，三个历表都指示了此刻很有可能在发生日食。

## 总结

在本项目中，我们统一了DE历表、INPOP历表和EPM历表的接口格式为DE436历表所规定的接口格式，并且可以自动化生成并验证二进制格式的历表。同时，我们对三个历表进行了比较，发现其不存在本质上的差别；也用这三个历表验证了2009年7月22日发生的日全食，发现三个历表均预测此刻很有可能正在发生日全食。这说明DE历表、INPOP历表和EPM历表不存在精度上的差别，INPOP历表和EPM历表均可以作为DE历表的替代品。

## 附录

1. DE历表下载地址：

<ftp://ssd.jpl.nasa.gov/pub/eph/planets/>

2. INPOP历表下载地址：<https://www.imcce.fr/recherche/equipes/asd/inpop/download17a>

3. EPM历表下载地址：

<ftp://ftp.iaaras.ru/pub/epm/>

4. 本项目github网址和zip压缩包下载地址：

<https://github.com/frokaikan/bysj>

https://github.com/frokaikan/bysj/archive/master.zip

5. 本项目说明文件README.md：

# program BINDING\_EPH

author : frokaikan

It's free to use, you can use or change it without my authorization.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## How to use this program?

+ You need :

+ Linux :

+ GCC (GNU Compiler Collections)

+ Python3

+ Windows :

+ MinGW (Minimum GCC for Windows)

+ Python3

+ You should : ("xxx" can be any name)

1. mkdir "source/" and put all your file to "source/" directory.

2. Rename your header file to "xxx.header".

3. choose ONE of the following three steps :

+ Rename all your ascii file to "xxx.asc", and let them sorted in lexicographical order. (If you have DE format ascii file)

This program will automatically change fortran-style float number (e.g. 1.2D+03) to C-style float number (e.g. 1.2E+03)

+ Rename your BINDED ascii file to "xxx.in". (If you have binded your .header file and .asc files)

+ Rename your binary file to "JPLEPH". (If you have DE format binary file)

4. (Just for test) Rename your testpo file to "xxx.testpo". If you don't have testpo file, ignore this step.

5. type `python makeit.py make -k <ksize> -n <namfil>` to generate EPH file.

6. find your JPLEPH file at "output/" directory. The outputs are :

+ asc2eph.exe :: change your .in file to EPH binary file.

+ "namfil" :: the binary EPH file.

+ libeph.o :: the Object file contains all JPL functions (except fsizer3).

+ fsizer3.o :: the Object file contains function fsizer3.

+ testeph.exe :: test your JPLEPH file.

7. (Just for test) type `python makeit.py test` to run testeph.

If you have any problem, please send an e-mail to "541240857@qq.com". I'll reply you ASAP.

Thanks for using this program.

6. 本项目asc2eph.cpp：

<https://github.com/frokaikan/bysj/blob/master/asc2eph.cpp>

7. 如何打开命令行窗口:

8. gcc官方网站下载页：

<https://gcc.gnu.org/install/binaries.html>

9. MinGW-W64 x86\_64-win32-sjlj-rev0下载地址：

<https://jaist.dl.sourceforge.net/project/mingw-w64/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/8.1.0/threads-win32/sjlj/x86_64-8.1.0-release-win32-sjlj-rt_v6-rev0.7z>

10. 测试程序cmp.cpp:

#include <cstdio>

extern "C" void pleph\_(double\*, int\*, int\*, double(\*)[6]);

extern "C" void pl\_units\_(bool\*, bool\*, bool\*);

double pos[6];

void output(FILE\* stream)

{

std::fprintf(stream, "%0.10lf,%0.10lf,%0.10lf,,%0.10lf,%0.10lf,%0.10lf\n",

pos[0],

pos[1],

pos[2],

pos[3],

pos[4],

pos[5]

);

}

void query(double time, int p1, int p2)

{

pleph\_(&time, &p1, &p2, &pos);

}

int main()

{

FILE\* csv = fopen("out.csv", "w");

bool t = true, f = false;

pl\_units\_(&t, &t, &f);

for (double tx = 2451545; tx <= 2451910+1e-9; ++tx)

{

query(tx, 11, 3);

output(csv);

}

}

11. 查询程序ask.cpp:

#include <cstdio>

#include <vector>

extern "C" void pleph\_(double\*, int\*, int\*, double(\*)[6]);

extern "C" void pl\_units\_(bool\*, bool\*, bool\*);

double pos[6];

void print()

{

std::printf("Position :: %20.10lf %20.10lf %20.10lf\nSpeed :: %20.10lf %20.10lf %20.10lf\n",

pos[0],

pos[1],

pos[2],

pos[3],

pos[4],

pos[5]

);

}

void query(double time, int p1, int p2)

{

pleph\_(&time, &p1, &p2, &pos);

print();

}

std::vector<double> save\_pos()

{

std::vector<double> ret(6);

std::copy(pos, pos+6, ret.begin());

return ret;

}

int main()

{

bool t = true, f = false;

pl\_units\_(&t, &t, &f);

double dt;

std::scanf("%lf", &dt);

query(dt, 11, 3);

auto v1 = save\_pos();

query(dt, 10, 3);

auto v2 = save\_pos();

for (int i=0; i<6; ++i)

std::printf("part %d : %20.10f\n", i, v1[i]/v2[i]);

}

## 参考文献