Oulu - Open-source Geomagnetosphere Propagation Tool (OTSO) User Manual

Nicholas Larsen

October 17, 2022

Abstract

This manual outlines, in detail, all of the necessary information needed by a user to install and use the "Oulu - Open-source Geomagnetosphere Propagation Tool" (OTSO). The various different functions that OTSO provides are explained and an example utilising the tool is provided, showing what the inputs and outputs of OTSO are as well as how to utilise these outputs.

1 Introduction

OTSO is a tool that preforms the numerical integration of the equations of motion for a charged particle within the Earth's magnetic field. This is done to simulate the trajectory of cosmic rays when they encounter the magnetosphere and to determine where said cosmic rays encounter the Earth's surface. OTSO can do this for single cosmic rays to determine the trajectory or for many cosmic rays of varying rigidities in order to determine important parameters, such as cut-off rigidities and asymptotic cones. As such this tool is primarily designed to aid in cosmic ray research and the study of space weather events, namely ground level enhancements.

OTSO is written in two different programming languages, python and fortran. Python is used to enter the input parameters and run the program. The fortran section is responsible for the numerical integration and modelling the Earth's magnetic field. This combination mixes together the user friendly nature of python with the older, more efficient, fortran which has access to many previous created open-access libraries designed for this field.

OTSO is designed to be open-source. The user of OTSO has free reign to edit the program how they see fit in order to achieve their goals. Additions to the base tool can be made by the community which will help develop the tool into a robust geomagnetic computation tool to aid the cosmic ray research community.

As new functions are added I will endeavour to update this manual and the supplied scripts to include information on the new applications as frequently as possible. I, however, implore those who made additional edits that are added to the base tool to provide their own documentation detailing the new features to the repository to simplify this process.

2 Installation Instructions

Please note the following instructions are to get OTSO functioning on a computer that uses the Windows operating system, as this was what the tool was developed on. Other operating systems may have different install procedures or encounter errors in getting OTSO to work.

In order to operate OTSO you will need to have Python installed on your computer alongside the necessary packages. It is recommended that you install the Anaconda program (https://www.anaconda.com/) as this includes all needed packages within it by default. Second, you will also need a working Fortran compiler in order to compile the Fortran section of OTSO. Within the development of the project the gfortran compiler has been used, this can be obtained quite easily online through multiple services (e.g. https://gcc.gnu.org/ and https://www.mingw-w64.org/). We recommend using mingw to install the compiler as it is referred to later on in the setup of OTSO.

Downloading OTSO onto your computer will provide you with a .zip file containing the Fortran library as well as a folder which includes the Python scripts to use the tool. In order to use these python scripts the library must be compiled, organised into a static library, and then compiled again in the Python scripts folder. It is important to note that the following instructions must be followed to recompile the library whenever an edit is made to the Fortran source code.

2.1 The Fortran Library

Within the library folder there will be a large collection of Fortran files with different fortran versions, a side effect of utilising older libraries within OTSO. In order to compile the library you will need to open a command prompt window with the directory of the library folder. After this is done you will need to enter the text:

```
c:\User\OTSO\library> gfortran -c *.f *.f95 -fallow-argument-mismatch
```

This will compile all the .f and newer .f95 files within the folder, creating .o files. Please pay attention that some of the files will have produced .mod files, these are important and should be copied over to the scripts folder. After compiling, the .o files need to be made into a static library. To make the library enter the following line into the command prompt:

```
c:\User\OTSO\library> ar cr OTSOlib.a *.o
```

This will create a static library within the library folder called OTSOlib.a, copy this file, along with the .mod files, to the scripts folder.

2.2 F2PY Compiling

The last step before using the tool is to compile with F2PY within the scripts folder. First, you must find the F2PY script within the anaconda directory and copy the path location. It should appear similar to:

```
c:\User\anaconda3\Scripts\f2py-script.py
```

Once this is done, you must enter the following line into a command prompt with its directory set to the script folder:

```
c:\User\OTSO\scripts>python [YOUR F2PY FILE PATH] -c --fcompiler=gnu95 --compiler=mingw32 -m MiddleMan
Middleman.f95 OTSOlib.a
```

If the compilation was successfully you should see a line at the end of the command prompt that states "Removing build directory". The Middleman term refers to the file that works as the intermediary between the python scripts and fortran library, hence "Middleman". You should now be able to run the Python scripts within the scripts file, allowing you to use OTSO.

3 Functions

- Trajectory
- Asymptotic Cones
- Effective Cutoff Rigidities

There are three main functions within OTSO, these being the computation of: CR trajectories, effective cut-off rigidities, and asymptotic cones. The latter two of these functions are computed in conjunction with each other within the same routine, this saves time as the required computations for one of the results is also needed for the other. There are three other minor functions that OTSO provides. These are the: coordinate transform, magnetic field strength, and planetary cut-off functions. The coordinate transform simply uses the IRBEM library to convert one coordinate system to another; magnetic field strength computes the field strength at a given point under user-inputted conditions, and planetary cut-off computes the effective cut-off rigidity for the entire globe.

3.1 Trajectory

Invoking the trajectory function will make OTSO compute the path of a CR from a given point within the magnetosphere (typically 20km above the Earth's surface) using the 4^{th} order Runge-Kutta numerical integration method until one of three conditions are met. The conditions under which the computation stops are if the CR: encounters the model magnetopause, falls below an altitude of 20km, or travels over 100 Earth radii without meeting the either of the prior two conditions, the trajectory will be considered allowed if the first condition is met and forbidden if either of the latter two are met.

3.2 Cone

As mentioned prior the cone function conducts two of the key computations in tandem, these being the asymptotic cones computation and effective cut-off rigidity calculations. Both of these results require the modelling of cosmic ray trajectories over a range of rigidity values that is defined by the user.

3.2.1 Asymptotic Cone

In order to determine the asymptotic cone of a point on the Earth (typically a neutron monitor location is selected) OTSO will compute the asymptotic latitude (Λ) and longitude (Ψ) of a cosmic ray once the modelling has completed for that rigidity value. The equations for these values in the spherical coordinate system are:

$$\tan\Lambda = \frac{-v_\theta \sin\theta + v_r \cos\theta}{\sqrt{v_\varphi^2 + (v_\theta \cos\theta + v_r \sin\theta)^2}} \tag{1}$$

$$\Psi = \varphi + \arctan\left(\frac{v_\varphi}{v_\theta \cos\theta + v_r \sin\theta}\right) \tag{2}$$

where θ is the co-latitude and φ is the longitude. These calculations are done regardless of where the cosmic ray is at the end of the modelling.

3.2.2 Effective Cut-off Rigidity

While OTSO repeats the same cosmic ray trajectories over the inputted range of rigidities it can encounter a region known as the penumbra. Within this region there is a mixture of allowed and forbidden cosmic ray trajectories, corresponding to being able to and not able to escape the magnetopause respectively. To make sense of the penumbral region an effective cut-off rigidity (R_c) is computed using the last accepted rigidity before the penumbra (R_U), the last accepted rigidity (R_L), and the size of the steps in rigidity over the range being tested (Δ). The equation for (R_c) is as follows:

$$R_c = R_U - \int_{R_U}^{R_L} \Delta R_{(allowed)} \tag{3}$$

4 How to Use

Figure 3 shows the typical layout of the cone script for computing the asymptotic cone and cutoff rigidity for a given location. The script has very clear sections highlighted by commented sections in which the user enters input values for OTSO to use. The following list walks through these sections and explains them separately. It is important to note that other scripts provided with OTSO have a similar layouts, as a result the information provided in this section is applicable to other OTSO functions with minor variations.

4.1 Input

- The stations you wish to test are entered into the list as strings, a separate .csv file containing many of the stations is included with OTSO to reduce the need to look up neutron monitor locations. Additional stations can be added to the list through the AddLocation method (currently commented out). Within this section the start altitude, zenith, and azimuth for the modelled cosmic rays are entered.

- The solar wind conditions present at the time the simulation are inputted in this section. Some other geomagnetic indices are included in this section, however, values are only required for these if later external magnetosphere models are used. It is useful to note that G1, G2, and G3 are special parameters needed for the use of the Tsyganenko 01 and 01S models, a separate script is provided by OTSO to calculate these for the period of time desired.

- IOPT is selected next and relates to the geomagnetic disturbance level (kp index). The IOPT value is typically entered as the kp index value + 1, unless the kp index is greater than or equal to 6 in which case IOPT is always set to 7. This is particularly important for earlier external magnetic field models, such as Tsyganenko 89, it is not needed when using more modern models.

- This section allows the user specify the type of cosmic ray to be modelled. OTSO provides the option to test cosmic rays that have heavier nuclei. In the base release elements up to beryllium can be tested, however heavier elements can be added by users. The type of cosmic ray is selected via the AtomicNum variable. AntiCheck is a simple binary variable which will decide the charge of the cosmic ray, a value of one will make the cosmic ray its anti-particle counterpart and vice versa for zero (e.g. one = anti-proton, zero = proton).

- The date of that the simulation is to take place is simply entered into the script by using a datetime object, typical within python programs. This is converted into an array which allows fortran to understand the information within the object.

- Magnetic models that are to be used within the computation are selected by entering integer values into the "model" array. The first number in the array refers to the internal magnetic field model being used to represent the Earth's dynamo and the second number corresponds to the external field model that encompasses the effect of magnetospheric currents. The list of available models are shown in Figure 3.

- The start, end, and step values for the rigidities are defined here. The program will go from the start value to the end value in the given step sizes to test cosmic rays of the various rigidities within the given range. When using the trajectory script only one rigidity value is given in this section instead of a range.

- Thanks to the incorporation of the IRBEM library the positional outputs of the computation (OTSO provides the location of the cosmic ray at the end of the simulation for each particle) can be given in a multitude of coordinate systems. In this section the coordinate system is selected. Please refer to the IRBEM library website for further information on the coordinate systems that can be selected https://prbem.github.io/IRBEM/.

- The numerical integration used within OTSO to resolve the closed-form equations of motion that determine the trajectory of the cosmic rays uses a variable time step. The time step is able to grow or be reduced depending on error checks within the program, these detect if the speed of the cosmic ray has grown by an unacceptable amount. The time step has a maximum value determined by the user, preventing the time step growing too large. This maximum value is determined to be a specific percentage of the time taken for the cosmic ray to complete one gyration within the magnetic field at its current location in the magnetic field. Within this section the user can determine the percentage used to determine this maximum time step. The smaller the percentage is set the slower the computation and the more accurate the result will be, however, testing has shown that a good compromise between speed of the computation and result accuracy appears to be a percentage value of 0.15 - 0.4. Values higher than this range start to lead to inaccuracies in exchange for slight increases the computation speed, see Figure 1.

- Most modern external magnetic field models have an associated magnetopause model included within them, enclosing the field model. However, some of the earlier models, namely Tsyganenko 87 and 89, do not have a specific magnetopause model. In the case that the user wishes to use these earlier models a "de-facto" magnetopause boundary must be selected. OTSO currently provides several empirical models to choose from, with the possibility of more being added later. It is useful to note that if using an external field model with a explicitly defined magnetopause

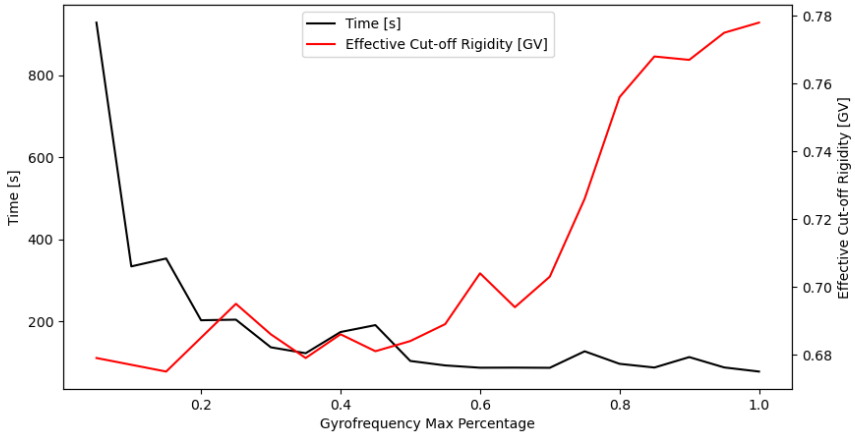

Figure 1: Comparison of computation time and effective cut-off rigidity against the max percentage of the gyrofrequency used within said computation. The OTSO computation done was the same as is shown within the example section, see Figure 3 for the specific input values.

the users choice in this section will be overwritten in favour of the magnetopause model used in the field model selected.

- OTSO can produce many files for computations containing many locations. To organise these files the user can input a string that will be added to the name of the produced files as well as providing a name for a folder in which the created files will be stored. Within the created folder a README file is produced that will provide all the information about the computation performed so it is easy to know what the input values were for the files within the folder.
- The final section deals with multi-core processing. The user can enter the amount of cores they wish to split the computations over which reduces the time taken for OTSO to complete the computation for each given location. OTSO will also read the number of cores on the user's computer and prevent too many cores being used as this will cause the program to either fail or take an irrationally long time to complete.

4.2 Output

```
Rigidity(GV),Filter,Latitude,Longitude,X,Y,Z
20.0000,1,41.0066,60.9246,6.27301,10.0045,11.3669
19.9990,1,41.0056,60.9245,6.27195,10.0025,11.3645
19.9980,1,41.0047,60.9243,6.27090,10.0005,11.3622
19.9970,1,41.0037,60.9242,6.26985,9.99846,11.3598
19.9960,1,41.0028,60.9240,6.26879,9.99646,11.3574
19.9950,1,41.0018,60.9239,6.26774,9.99446,11.3550
19.9940,1,41.0008,60.9238,6.26668,9.99246,11.3526
19.9930,1,40.9999,60.9236,6.26563,9.99046,11.3502
19.9920,1,40.9989,60.9235,6.26452,9.98836,11.3477
19.9910,1,40.9980,60.9233,6.26338,9.98621,11.3452
19.9900,1,40.9970,60.9232,6.26225,9.98406,11.3426
```

(a) Top of file

```
0.110000E-1,-1,27.5107,40.1165,0.283900,0.491600,-0.825604
0.100000E-1,-1,48.0840,46.0148,0.284328,0.491168,-0.825710
0.900000E-2,-1,28.2255,24.9671,0.284816,0.490650,-0.825850
0.800000E-2,-1,27.4454,39.7879,0.285347,0.490174,-0.825951
0.700000E-2,-1,42.3870,16.7790,0.285866,0.489673,-0.826069
0.600000E-2,-1,50.0045,23.7422,0.286488,0.489147,-0.826161
0.500000E-2,-1,42.8513,49.2183,0.287241,0.488571,-0.826243
0.400000E-2,-1,26.1784,33.1889,0.288158,0.487898,-0.826322
0.300000E-2,-1,34.5534,17.8472,0.289432,0.487087,-0.826353
0.200000E-2,-1,42.3993,16.5798,0.291646,0.485892,-0.826279
Ru:0.789000,  Rc:0.679000,  Rl:0.508000
```

(b) Bottom of file

Figure 2: The output file for the cone function has seven columns, shown clearly on the left. Rigidity, Filter (1 = allowed, 0 and -1 = forbidden. 0 = exceeded travel distance without escaping, -1 = encountered the Earth), Latitude (asymptotic), Longitude (asymptotic), and X, Y, and Z (location the computation ended in the user inputted coordinate system). The bottom of the file on the right shows that underneath all of the rigidity computations the upper, lower, and effective cut-off rigidities (Ru, Rl, and Rc respectively) are given.

```
if __name__ == '__main__':
#################################################################################################
# Picking the stations to be tested.
# Additional stations can be added via the .AddLocation("Name",Latitude,Longitude) function

 List = [
 "Oulu"]
 Alt = 20.0
 Zenith = 0
 Azimuth = 0
 CreateStations = Stations(List, Alt, Zenith, Azimuth)
 #CreateStations.AddLocation("Anomoly", -32, 222)

 UsedStationstemp = CreateStations.GetStations()

 temp = list(UsedStationstemp)
 UsedStations = random.shuffle(temp)
 UsedStations = temp

#################################################################################################
# Solar Wind Conditions
 Vx = -500 #[km/s]
 Vy = 0.0 #[km/s]
 Vz = 0.0 #[km/s]
 By = 5 #[nT]
 Bz = -5 #[nT]
 Density = 1.0 #[cm^-2]
 Dst = 0 #[nT]

# G1 and G2 are only needed for TSY01 use (use external TSY01_Constants tool to calculate them)
# or enter the average solar wind speed, By, and Bz over the 1 hour period prior to event
 G1 = 0
 G2 = 0
 G3 = 0



 WindCreate = SolarWind(Vx, Vy, Vz, By, Bz, Density, Dst, G1, G2, G3)
 WindArray = WindCreate.GetWind()
#################################################################################################
# IOPT is Picked depending on the Kp index at the time picked
# Take IOPT = kp + 1
# Exception if Kp>=6 IOPT = 7
 IOPT = 3
#################################################################################################
# Choose the atomic number of particle you want to test e.g 0 = electron, 1 = proton, 2 = helium
# Choose if you want to reverse the charge 0 = particle, 1 = anti-particle
 AtomicNum = 1
 AntiCheck = 1
#################################################################################################
# Enter date to be investigated as a datetime under EventDate
 EventDate = datetime(2006, 12, 13, 3, 00, 00)
 DateCreate = Date(EventDate)
 DateArray = DateCreate.GetDate()
#################################################################################################
# Pick the magnetosphere models that you want to use.
# Internal: 1 = IGRF, 2 = Dipole (First Number in model array)
# External: 0 = No External Field 1 = TSY87(short), 2 = TSY87(long), 3 = TSY89, 4 = TSY96, 5 = TSY01,
# 6 = TSY01(Storm) (Second Number in model Array)
 model = np.array([1,3])
#################################################################################################
# Pick the start and end rigidity for the computation, as well as the step
 StartRigidity = 20
 EndRigidity = 0
 RigidityStep = 0.001
 RigidityArray = [StartRigidity, EndRigidity, RigidityStep]
#################################################################################################
# Pick the coordinate system desired for the output
# GDZ, GEO, GSM , GSE, SM, GEI, MAG, SPH (GEO in spherical), RLL
 CoordinateSystem  = "GEO"
#################################################################################################
# Pick the maximum percentage of the particles gyrofrequency that can be used as the max time step in the
# computation
 MaxStepPercent = 0.15
#################################################################################################
# Choose model magnetopause
# 0 = 25Re Sphere, 1 = Aberrated Formisano, 2 = Sibeck, 3 = Kobel
 Magnetopause = 3
#################################################################################################
# Choose name of folder that output files will be sent to. Folder created in current directory
 FolderName = "Speed Folder 1"
 FileName = "1"
 current_directory = os.getcwd()
 final_directory = os.path.join(current_directory, FolderName)
 if not os.path.exists(final_directory):
   os.makedirs(final_directory)

 FileDescriptors = [FileName, FolderName, final_directory]
#################################################################################################
# Select number of cores for multicore processing
 CoreNum = 1
 UsedCores = Cores(UsedStations, CoreNum)
 CoreList = UsedCores.getCoreList()
 Positionlists = UsedCores.getPositions()
 ChildProcesses = []
#################################################################################################
```

Figure 3: An example of inputs that might be given to investigate the Oulu neutron monitor station using the Cone script.

5 Example

Values used within Figure 3 are used to calculate the effective cut-off rigidity and asymptotic cone for the Oulu neutron monitor. The results of this are shown within Figures 4 and 5.

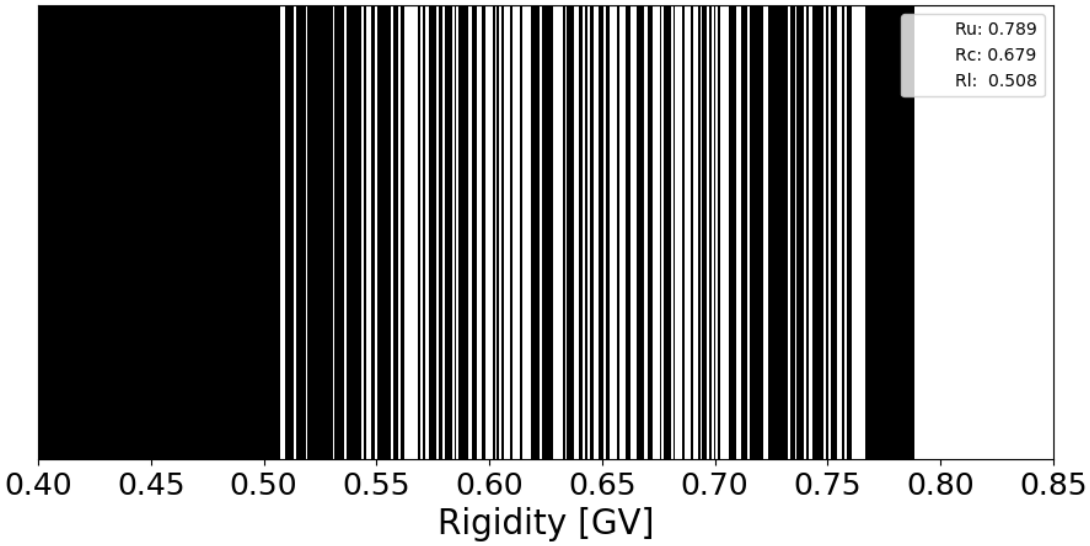

Figure 4: Allowed (white) and forbidden (black) rigidities for a CR arriving at the Oulu neutron monitor station using the input values given in Figure 3. The upper, lower, and effective rigidities are shown as Ru, Rl, and Rc respectively.

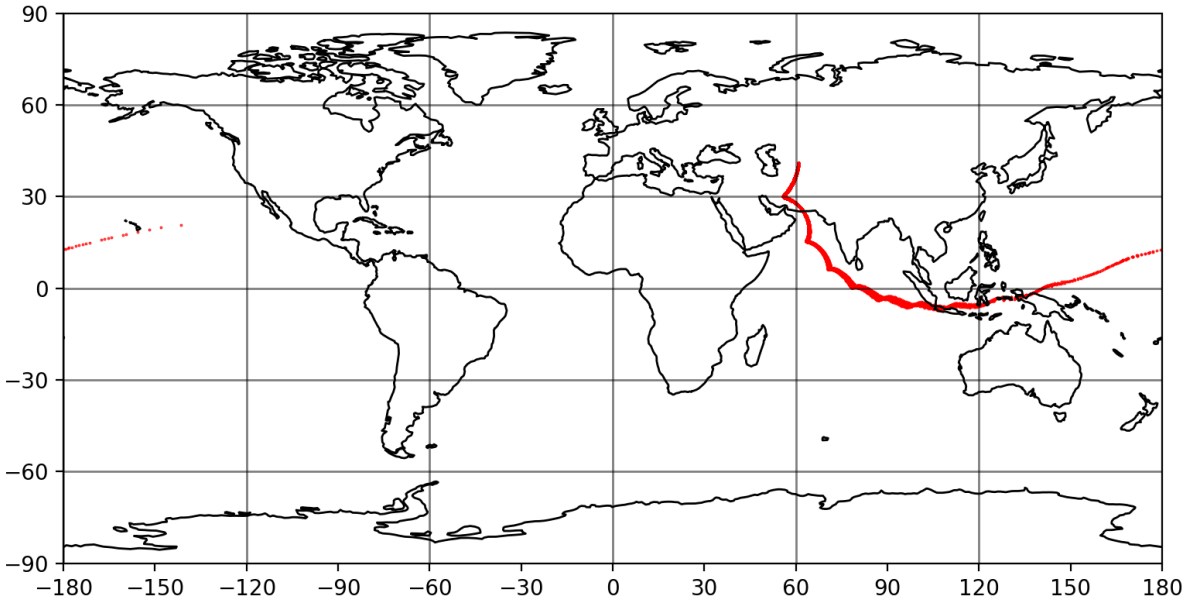

Figure 5: Asymptotic cone for the Oulu neutron monitor station using the input values given in Figure 3.

6 Editing

As mentioned before in the installing section, if the user decides to make changes within the fortran section of the code the same steps must be undertaken as when first compiling the code. The fortran section of the code uses pointers for key parts of the computation that may want to be edited, this prevents IF ELSE bottlenecks in the code. These pointers apply to the magnetic field models and magnetopause models. If the user wishes to add a model to these sections it is quite straight forward to add in new models to these sections, due to the modular design. As long as the new model inputs are the same as prior models one need only copy and paste a pointer assigning function and change the function name. Unfortunately a limitation of fortran is that when using pointers the inputs must be the same for all the models, to combat this the user should alter the module files to include any extra parameters needed so that the model can access any additional values needed that aren't provided in the function input. An example of this is the solar wind module file that contains many variables that apply to specific magnetic field models.