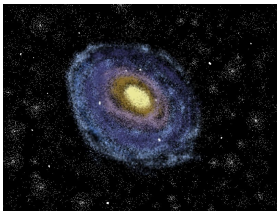

Team Andromeda

User Manual

May 7, 2020

Clients - Dr. Audrey Thirouin and Dr. Will Grundy
Mentor - Isaac Shaffer
Members - Matthew Amato-Yarbrough, Batai Finley, Bradley Kukuk, John Jacobelli, and Jessica Smith

1. Introduction

We are pleased you have chosen licht-cpp for your business needs. Licht-cpp is a powerful API for modeling and visualization of binary asteroid systems that has been custom-designed to meet your needs. Some of the key highlights include:

- A graphical user interface (GUI) that allows easy input for a function that predicts a lightcurve for a single or binary asteroid system
- Renders that are traced by pixel to provide highly accurate light curves
- Uniquely solves precision issues that stem from distance
- Extremely efficient ray-tracer with generated shapes and CPU parallelization
- A nonlinear minimizer (NLM) to find best fitting parameters based on observed data
- The option to display the output from the NLM to a plot, file, or to the terminal
- Ability to create a video of the system's motion
- Three distinct shape types for modeling usage
- An automatic plot of the light curve on the GUI

The purpose of this user manual is to help you, the client, successfully install, administer, and maintain the licht-cpp codebase in your actual business context going forward. Our aim is to make sure that you are able to benefit from our product for many years to come!

2. Installation

2.1 Licht-cpp Installation

As part of final delivery, licht-cpp should have been installed on a platform of your choice. Over time, however, you may want to move to a new platform or re-install the product.

Licht-cpp is a cross-platform solution that requires a C++ compilation environment using CMake. Ultimately, to install licht-cpp the following commands must be run:

- cd licht-cpp/build
- cmake ..
- make

To accomplish this, follow the set-up for a CMake environment for your system below.

Debian-based systems

- sudo apt install cmake

RedHat-based systems

- dnf install clang clang-devel # If you want to use clang
- dnf install libomp-devel # If openmp is not installed
- dnf install cmake

CentOS 6/7

Install g++ 8

- sudo yum install centos-release-scl
- sudo yum-config-manager --enable rhel-server-rhscl-7-rpms
- sudo yum install devtoolset-8

Update terminal

- scl enable devtoolset-8 bash

Install CMake 3.X

- http://jotmynotes.blogspot.com/2016/10/updating-cmake-from-2811-to-362-or.html

Windows

Essentially, Windows needs a MinGW setup that supports pthreads, make, and openmp. To do this, first install the MinGW installation manager (https://osdn.net/dl/mingw/mingw-get-setup.exe). Then install the following packages. Many come with the base system of MinGW and MSYS, but they are pasted here for verbosity:

- mingw-developer-toolkit-bin
- mingw32-autoconf-bin
- mingw32-automake-bin
- mingw32-autotools-bin
- mingw32-base-bin
- mingw32-binutils-bin
- mingw32-gcc-bin
- mingw32-gcc-lic
- mingw32-gcc-g++-bin
- mingw32-gdb-bin
- mingw32-gettext-bin

- mingw32-gettext-dev
- mingw32-libatomic-dll
- mingw32-libexpat-dll
- mingw32-libgcc-dll
- mingw32-libgettextpo-dll
- mingw32-libgmp-dll
- mingw32-libgomp-dll
- mingw32-libiconv-bin
- mingw32-libiconl-dev
- mingw32-libiconv-dll
- mingw32-libintl-dll
- mingw32-libisl-dll
- mingw32-libtldl-dev
- mingw32-libltdl-dll
- mingw32-libmpc-dll
- mingw32-libmpfr-dll
- mingw32-libpthreadgc-dev
- mingw32-libpthreadgc-dll
- mingw32-libquadmath-dll
- mingw32-libssp-dll
- mingw32-libstdc++-dll
- mingw32-libtool-bin
- mingw32-libz-dll
- mingw32-make-bin
- mingw32-mingw-get-bin
- mingw32-mingw-get-gui
- mingw32-mingw-get-lic
- mingw32-mingwrt-dev
- mingw32-mingwrt-dll
- mingw32-pthreads-w32-dev
- mingw32-pthreads-w32-lic
- mingw32-w32apidev
- mingw32-wslfeatures-cfg
- msys-autogen-bin
- msys-base-bin
- msys-bash-bin
- msys-bison-bin
- msys-bsdcpio-bin
- msys-bsdtar-bin
- msys-bzip2-bin
- msys-core-bin

- msys-core-doc
- msys-core-ext
- msys-core-lic
- msys-coreutils-ext
- msys-cvs-bin
- msys-diffstat-bin
- msys-diffutils-bin
- msys-dos2unix-bin
- msys-file-bin
- msys-findutils-bin
- msys-flex-bin
- msys-gawk-bin
- msys-grep-bin
- msys-guile-bin
- msys-gzip-bin
- msys-inetutils-bin
- msys-less-bin
- msys-libarchive-dll
- msys-libbz2-dll
- msys-libcrypt-dll
- msys-libexapt-dll
- msys-libgdbm-dll
- msys-libgmp-dll
- msys-libguile-dll
- msys-libguile-rtm
- msys-libiconv-dll
- msys-libintl-dll
- msys-libltdl-dll
- msys-liblzma-dll
- msys-libmagic-dll
- msys-libminires-dll
- msys-libopenssl-dll
- msys-libopts-dll
- msys-libpopt-dll
- msys-libregex-dll
- msys-libtermcap-dll
- msys-libxml2-dll
- msys-lndir-bin
- msys-m4-bin
- msys-make-bin
- msys-mktemp-bin

- msys-openssh-bin
- msys-openssl-bin
- msys-patch-bin
- msys-perl-bin
- msys-rsync-bin
- msys-sed-bin
- msys-tar-bin
- msys-termcap-bin
- msys-texinfo-bin
- msys-vim-bin
- msys-cz-bin
- msys-zlib-dll

Then install CMake: https://cmake.org/download

Compilation is done via the MSYS shell located at 'C:\MinGW\msys\1.0\msys.bat'. Once in its bash shell, the C:\ drive can be accessed with 'cd /c'. Navigate to the licht-cpp build directory and call 'cmake .. -G "MSYS Makefiles"'. This only has to be configured once.

General Configuration

With the right packages now installed, the build system must be configured.

Set CC/CXX compiler

This step is only sometimes needed, as it depends if the system already has a global compiler variable. The codebase is tested on both g++ and clang++. Either may be used. If not already defined, a common way to add a global variable is to modify ~/.bashrc.

Make

CMake is used to build the code. The make step produces a unit test executable (tester) and a shared library (liblicht-cpp.so). CMake can be used as follows:

- cd licht-cpp/build
- cmake ..
- make # add -jX where X is twice the number of your CPU cores

Run

To run the unit tests, call tester. Otherwise, just interface with liblicht-cpp.so. When calling tester, it is important to be within the build directory, as the file paths within the test are relative to it.

Documentation
To generate documentation, run the below command from within licht-cpp/:

- doxygen Doxyfile # Generated in doc/

It generates five forms of documentation. The most useful form, in our opinion, is HTML. Open doc/html/index.html to browse the documentation of this code base. The large appendix that has been provided in the final report was generated by converting licht-cpp/doc/latex/refman.tex to PDF. This can be done by using a package called pdflatex.

2.2 NLM

Similar to licht-cpp, the NLM is also a solution that can be used on any platform. There are only three requirements associated with using the NLM:

- A C++ compilation environment that utilizes CMake
- Python2.7 and Python2.7-dev, Matplotlib, and Numpy installed
- The compilation of the licht-cpp API

In order to start using the NLM, the user will first have to install Python2.7. This includes the standard Python2.7 library, as well as the Python2.7-dev library. Along with these Python installations, the user will have to install Matplotlib and Numpy.

To install Python2.7 and Python2.7-dev, we recommend downloading the packages listed on the official Python website[1] for Python 2.7.18. The Windows and macOS installers found on the supplied page will install both Python 2.7 and Python2.7-dev. However, to install Python2.7-dev for Debian and Redhat systems, the user will need to run an additional command in the terminal:

- Debian systems: sudo apt-get install python2.7-dev
- Redhat systems: please refer to this official packages website[2]

To install Matplotlib and Numpy, we recommend using pip[3] due to it's versatility and reliability. To install these libraries using pip, you will need to run the following commands in a terminal or command prompt window:

[1] https://www.python.org/downloads/release/python-2718/
[2] https://pkgs.org/download/python2-devel
[3] https://pypi.org/

- Matplotlib[4]: pip install matplotlib
- Numpy[5]: pip install numpy

Once these libraries have been installed, the licht-cpp static library will need to be built. This is done by following the instructions found in section 2.1 for installing the licht-cpp API.

After installing the licht-cpp API, a static library called “liblicht-cpp.a” will be created in the licht-cpp/build directory. This static library is required to run the NLM as it provides the forward model functionality to the NLM.

Afterwards, the user can install the NLM using similar commands, but in a different directory:

- cd licht-cpp/nlm/build
- cmake ..
- make

Documentation

To generate documentation, run the below command from within licht-cpp/nlm/:

- doxygen Doxyfile # Generated in doc/

It generates five forms of documentation. The most useful form, in our opinion, is HTML. Open doc/html/index.html to browse the documentation of this code base.

2.3 GUI Installation

Similar to the licht-cpp library, the GUI is cross platform and available for use on any platform. To compile the GUI , the user must have qmake3.0 or higher on their system. Due to the difference in compilation, the GUI must be compiled after the licht-cpp library is compiled. The GUI is reliant on the static library (.a file) that is created when licht-cpp is compiled.

To compile the GUI:

- Compile forward model
- cd .. (Back into licht-cpp folder)

[4] https://pypi.org/project/matplotlib/
[5] https://pypi.org/project/numpy/

- cd GUI/
- qmake
- make

3. Configuration and Daily Operation

3.1 Licht-cpp

Given that licht-cpp is compiled into a static library that provides a single API function, there is no need for configuration. The only additional configuration is to compile the codebase in “Release” mode to enable some potentially unsafe optimizations.

- This is done by calling “cmake .. -DCMAKE_BUILD_TYPE=Release”.
- To undo this, call “cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo”.

For more information on what is ‘unsafe’ about this, topics to research are ‘-Ofast vs -O3’ and ‘-funsafe-math-optimizations’. In general, if the tests still pass in release mode, the mode is likely safe to use.

Before using the code, the test executable should be run to ensure the calculations are calibrated on the machine. To do this, call “./tester” from within the build directory and ensure all tests pass. If they do not, contact one of the developers, as it likely indicates an implementation difference amongst compilers and needs to be accounted for in the code.

To utilize the API function from the GUI or the NLM, please refer to sections 3.2 and 3.3 below for more information. Additionally, there exists extensive documentation for each of these functionalities that is either automatically generated or written and provided with the codebase.

Some particular aspects to keep in mind are:

- All arrays, except for the facet and vertice arrays, are indexed by the numShapes variable.
    - If numShapes is 1, then the first element (or ‘step’, for arrays such as spin poles) is accessed. If 2, then the first and second elements are accessed. Thus, the minimum dimensionality of these arrays is numShapes * stepSize. That is, more elements can be provided than shapes, but only the numShapes indices are accessed.

    - The API logically cannot ‘default’ a secondary shape’s parameters to the primary shape’s parameters. Thus, the user cannot just leave out the secondary’s element with numShapes = 2. It is undefined behavior to attempt this and there is no consistent implementation that can handle this. It also cannot consistently be tested for from within the C-API. The user is trusted to provide the correct dimensionality and numShapes.
- Wavefront files (.obj extension) are only supported in a very basic format.
    - Only the following lines are supported:
        - v x y z
        - f v1 v2 v3
    - Any other line is ignored. For example:
        - vn ...
    - Any deviations of the supported lines will likely result in a corrupt shape. For example:
        - f v1//vn1 v2//vn2 v3//vn3
        - v x y z w (though this might still work)
    - Thus, the user will need to remove such complications before passing it to the API. Keep in mind that applications like Blender often complicate the format, even if the input was supported by the API.
- vfov, or the vertical field of view in degrees, does not need to be a guessing-game
    - vfov = atan(3*radius / distance) will fill the majority of the frame
        - radius = max radius of both shapes
        - distance = distance to shape of interest, usually primary

The above information was provided from Paired Planet Tech’s User Guide from the first iteration of the project. This was included for clarification about areas of the project Team Andromeda did not work on. For more detail about particular aspects augmented by Team Andromeda, please refer to the following sections.

3.2 Triaxial Ellipsoids

Triaxial ellipsoids are a great addition to the clients’ current software. Running the software with the ellipsoid feature will help create more accurate models of asteroids while maintaining a quick runtime. This part of the document will serve as a guide to using ellipsoids and will detail their use.

Ellipsoids are similar to sphere and faceted objects because the user can specify size, pole orientation, rotation, and Hapke parameters for an ellipsoid object. Though similar, there are some differences from other shapes. This primarily refers to differences between the orient and spin functions utilized by both ellipsoids and

faceted objects. The differences in the orient and spin functions affect how the sizing, orientation, and rotation of the ellipsoid feature is calculated. To further explain this, the following sections specify how to create and use an ellipsoid object in the forward model.

Sizing

The forward model takes in the radii for ellipsoids as a Vector3d type. This means that the input looks like Figure 1. This is referred to as the **b** variable.

```
Vector3d(1.5, 1, 1)
```

Figure 1: Example of input for ellipsoid size

The radii can be thought of as (x, y, z), where the axes are defined using the planes of reference in Figure 2.

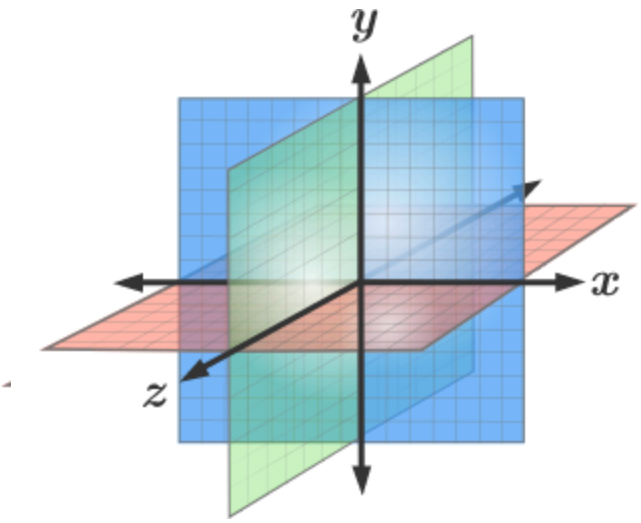

Figure 2: 3D Cartesian Coordinate System

Thus, the x-axis input controls horizontal sizing, the y-axis input controls vertical sizing, and the z-axis input controls "depth" sizing. These axes are relative to the camera. The z-axis referenced in Figure 2 **must** be the smallest axis of the ellipsoid, as rotation happens along this axis. As specified by the clients, the shortest axis will be the axis of rotation for ellipsoids.

Orientation

Orientation uses the equatorial coordinate system. An image of the equatorial coordinate system is shown in Figure 3 to the right.

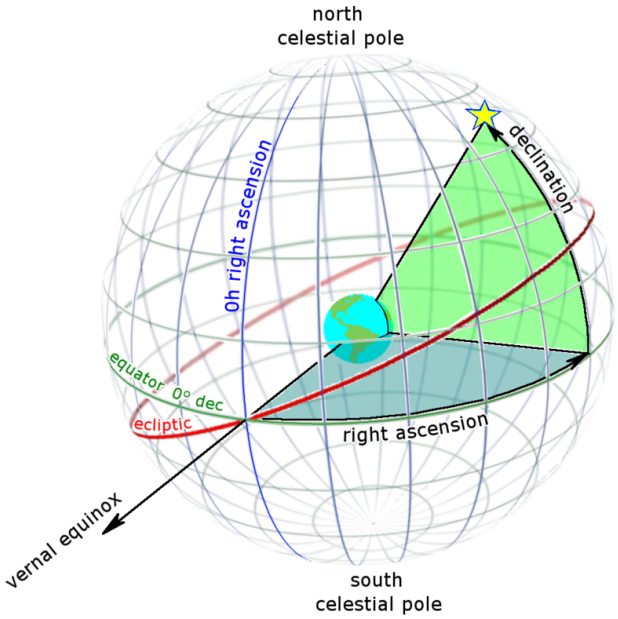

Figure 3: Equatorial Coordinate System

Since equatorial coordinates are used to position the asteroid's "north pole" and not used to simply find a point in space, the vernal equinox in Figure 3 should be thought of as the "north pole". The asteroid's "north pole" would then be positioned via the equatorial coordinate system.

For the ecliptic coordinate system implemented for ellipsoids, orientation is also specified using a Vector3d type, shown below in Figure 4. The spin poles can be defined as follows:

```
std::vector<Vector3d> spinPoles { Vector3d(DEC, RA, 0), Vector3d(DEC, RA, 0) };
```

Figure 4: Declaration of the orientation pole in a forward model test

The spin pole specifies the right ascension and declination values of rotation. **Ellipsoid rotation is defined by radians, not degrees**, so conversion must be accounted for. The first input for spin pole specifies the declination (DEC in Figure 4), while the second input specifies the right ascension (RA in Figure 4). The third value is a filler value due to a Vector3D type being needed for orientation, which requires 3 values. To keep things simple, always use a 0 for the third value. To give a brief overview of how this pole orientation is applied, an example is provided below.

Figure 5 is an image of a single object, which was generated using this website. Traditional ecliptic systems define lambda as longitude and beta as latitude. In our case, longitude is comparable to right ascension and latitude is comparable to declination.

Thus, the right ascension for Roxane is 220° and the declination is -62°. Compared to the images from the website, using right ascension and declination as longitude and latitude respectively provides an accurate orientation for ellipsoids.

To imagine this, think of the asteroid's "north pole" being directly oriented at the camera. First, the "north pole" is rotated clockwise 220°, then downwards 62°.

Again, the rotations must be specified in **radians** to achieve expected rotation. This will allow the ellipsoid to be properly oriented so that rotation can work as intended.

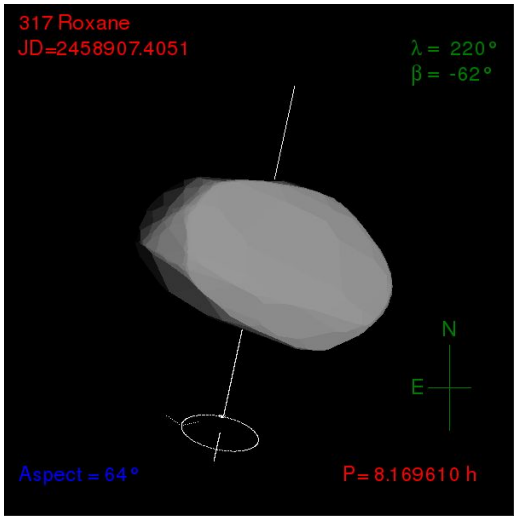

Figure 5: Roxane generated by the website

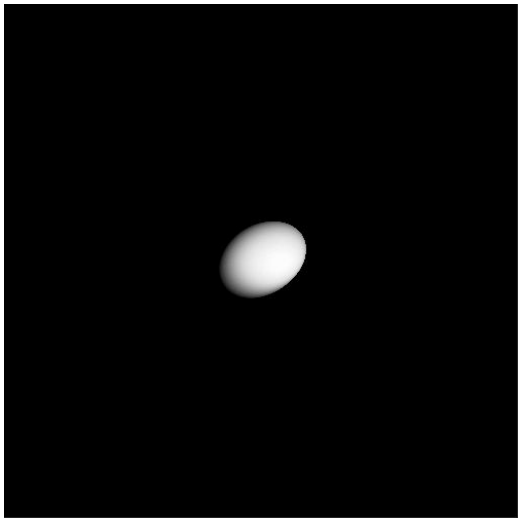

Figure 6: forward model rendered Roxane

Rotation

Ellipsoids rotate about the z-axis shown in Figure 2. An ellipsoid depicted pre-pole orientation is shown in Figure 6. In the forward model, **the ellipsoid rotates about the x-axis** shown in Figure 6, which is the same axis as the z-axis in Figure 2.

The ellipsoid will rotate about the x-axis shown in Figure 6 if right ascension and declination for the spin pole are both 0°.

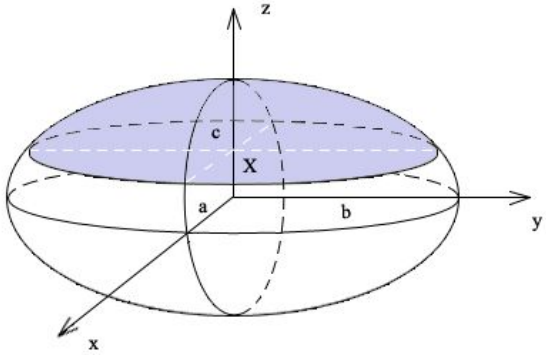

Figure 6: Ellipsoid and its axes prior to pole orientation

Single Asteroid Modeling

An interesting issue was encountered while developing ellipsoids and their use in the forward model. When trying to render a single ellipsoid, an issue occured where the ellipsoid was generated with extremely incorrect lighting and, therefore, produced a faulty light curve. A way to solve this issue was discovered, which was by including a second object. To remedy this, create an object (preferably a sphere as they are the most simplistic) with the radius size of 0. This allows for the forward model to recognize that there is a second object and provide correct lighting for the ellipsoid, while not rendering the second object.

If there are any issues or any additional help is required regarding the triaxial ellipsoid submodule, please refer to the troubleshooting section below. If there are further questions, please refer to the emails listed at the end of the document and feel free to contact either John Jacobelli or Jessica Smith.

3.3 NLM

The NLM module will provide our clients with several functionalities that were not present in this project's previous iteration. The NLM will allow our clients to specify forward model parameter values and choose which ones they want to be estimated. Additionally, data produced by the NLM is capable of being plotted to a graph, stored in a text file and saved. Renders of the binary systems created using estimated parameter values may also be produced using previously implemented forward model functionality. The following section will provide instructions for this module's installation and use.

User Input File

An important aspect to using the NLM is the user input supplied to it and the format in which it is supplied. To increase the ease of use associated with using the NLM, a user input file named "UserInput.txt" is included in the build directory. Figure 7 below illustrates the columns within the file, as well as four examples of what the rows look like.

```
#ID           Parameter                       Fit             Value                          Step-size
# ------------ Define Hapke parameters ------------------------------------------------------------------
0              phaseAnglePrimary               N               0                                    0
1              phaseAngleSecondary             N               0                                    0
2              singleScatteringAlbedoPrimary   N               0.9                                  0
3              singleScatteringAlbedoSecondary N               0.9                                  0
```

Figure 7: Format used by UserInput.txt file

Format

This file is formatted in a specific way in order to be parsed appropriately by the Parser.hpp file within the NLM. The file is formatted in the following way:

Columns:

- ID: the number corresponding to each respective parameter
- Parameter: the parameters that are necessary to run the forward model
- Fit: a letter that is used to determine whether a parameter is being estimated by the forward model
    - ‘N’ - represents a parameter that will not be estimated and will remain constant
    - ‘Y’ - represents a parameter that will be estimated by the NLM
    - ‘U’ - represents a parameter that is unable and should not be attempted to be estimated by the NLM
- Value: the value associated with a parameter
    - Parameters that **are not** being estimated will retain this value throughout the run of the NLM
    - Parameters that **are** being estimated will use the specified value as a starting value and will be adjusted by the NLM
- Step-size: the step-size for parameters that are being estimated
    - If a step-size is specified for parameters that have been chosen **not** to be estimated will be ignored
    - Otherwise, the step-size will be used to increment the value as the NLM determines its initial estimates

Note: The forward model includes parameters that are very sensitive to the values input. As such, when deciding which values and step-sizes to use with the NLM, it is important to use numbers that are appropriate for the parameter.

Rows:

- Parameters: individual values for each forward model parameter that are specified by the user
    - Certain forward model parameters were split into two individual parameters for the UserInput.txt. These parameters usually contain the keywords “Primary” or “Secondary”. For example, the Hapke parameter phaseAngle is split into phaseAnglePrimary and phaseAngleSecondary.
    - For the case of SpinPoles, the keywords “Primary” or “Secondary” are used, as well as “A”, “B”, “C”. The individual characters represent each value that is input for the SpinPole vectors in the form (A, B, C).

- Options: these are options that are specific to the running of the NLM.
    - ObservedData: this is where the user includes the path to the observed data that is being used with the NLM. These files are best stored in the licht-cpp/nlm/build/data directory for ease of input.
    - TerminalOuput, FileOutput, PlotOutput: these options are used to specify how the output from the NLM is displayed or saved to the user's machine.
- Comments: these are allowed in the UserInput.txt file. Any line starting with a '#' will be considered a comment and will be ignored by the parser.

Observed Data File

Observed data must be supplied to the NLM for it to have a set of data to compare predicted light curves against. To do this the use of an observed data file containing a binary systems information is needed. An example observed data file is provided in the NLM data directory.

```
2457463.6332267 22.6429 0.0843  g
2457463.6458417 22.6078 0.0741  g
2457463.6602043 22.6940 0.0807  g
2457463.6746808 22.5956 0.0756  g
```

Figure 8: Format used by the observed date file

Format

This file is formatted in a specific way in order to be parsed appropriately by the FileInput.hpp file within the NLM. The file is formatted in the following way:

Columns:

- Julian dates: the dates at which the corresponding magnitude values are captured
- Magnitude values: the brightness value of an observed binary system
- Error bar values: a value describing the range of error a corresponding magnitude value my vary by
- Label: one or more characters for distinguishing the light curves within the file from each other

NLM Output

When using the NLM there are several means by which output can be accessed by the user. This includes the graphs produced from plotting, the .txt file containing the data for the predicted light curve, and the renders produced when running NLM

with the “renders” parameter set to true. Terminal output is not saved but instead simply displayed to the terminal.

Plot Output

When the “plotOutput” flag in the UserInput.txt file is set to true, graphs containing the predicted and observed light curves plotted will be saved. The graphs will have an x-axis titled time, where the units are in JulianDate, while the y-axis will be plotted in magnitude. Note: the y-axis is inverted. One graph is created for each of the light curves in the observed data file. These graphs will be saved to a folder in the licht-cpp/nlm/build directory called “plot_output”.

File Output

When the “fileOutput” flag in the UserInput.txt file is set to true, a file containing the predicted light curve data will be saved. This file will be formatted to contain three columns: JulianDate, magnitude values, and the label associated with each light curve. This file will be saved to a folder in the licht-cpp/nlm/build directory called “predicted_data_output”.

Terminal Output

When the “terminalOutput” flag in the UserInput.txt file is set to true, printed output detailing the parameters estimated values and corresponding chi square value will be displayed to the users terminal. Each step that the NLM takes in the process of minimization will result in the output of estimated parameter values and the current chi square value at that step.

Renders

When the “renders” flag in the UserInput.txt file is set to true, a series of images for the predicted light curve will be produced. This functionality is extended from the liblicht-cpp.a library. In the case of the NLM, the number of images corresponds to the number of JulianDate times found in the observed data file. The images will be saved to a folder in the licht-cpp/nlm/build directory called “renders”.

Running

When creating the NLM, a main focus was the ease of use associated with using the module. After the cmake and make commands that are necessary to build the NLM have been executed, no other build process is necessary. From this point on, the user may call the NLM from the build directory (licht-cpp/nlm/build) as needed.

General operation:

- The user input file is filled out

- An observed data file is supplied
- Call the NLM using the command ./nlm
- Interpret output

As for the output of the NLM, this is decided based on the optional flags set by the user within the user input file.

Demo

A short demo of entering data into the userInput.txt file, running the NLM and displaying the output of the NLM can be found under the "Demos" section on the team's website[6].

Testing

To run the unit tests, call the tester from the build directory using the command ./tester. Otherwise, just interface with the NLM.

If there are any issues or any additional help is required regarding the NLM module, please refer to the troubleshooting section below. If there are further questions, please refer to the emails listed at the end of the document and feel free to contact either Matthew Amato-Yarbrough or Batai Finley.

3.4 GUI

The GUI was created so that the users would have a more efficient way to run and use the forward model. Due to the large complexities needed for the forward model, there are several requirements and constraints so that the user does not have any interruptions when using the GUI.

When using the GUI for the forward model, there are many specific requirements that must be met when inputting parameters.

Parameter Input Requirements

Files

All files that users would like to use within the forward model must be placed within the GUI/data directory. For file input, the forward model requires that all ephemeris files and observed data files have the extension '".txt". Once the user has placed the external file into the data/ directory, and ensured that it is a ".txt" file, the file will be available for use within the forward model. Please use the following syntax for all file input's within the GUI:

[6] https://www.cefns.nau.edu/capstone/projects/CS/2020/Andromeda-S20/

- “SampleInputFile.txt”

All files must follow one of the two possible templates:

```
*******************************************************************************
Ephemeris / WWW_USER Tue Feb 26 18:45:41 2019 Pasadena, USA      / Horizons
*******************************************************************************
Target body name: 79360 Sila-Nunam (1997 CS29)    {source: JPL#21}
Center body name: Earth (399)                     {source: DE431}
Center-site name: Lowell Obs.-Discovery Channel Telesc
*******************************************************************************
Start time      : A.D. 2013-Feb-07 00:00:00.0000 UT
Stop  time      : A.D. 2013-Feb-09 00:00:00.0000 UT
Step-size       : 10 minutes
*******************************************************************************
Target pole/equ : No model available
Target radii    : (unavailable)
Center geodetic : 248.577500,34.7442701,2.2444574 {E-lon(deg),Lat(deg),Alt(km)}
Center cylindric: 248.577500,5248.49170,3615.8703 {E-lon(deg),Dxy(km),Dz(km)}
Center pole/equ : High-precision EOP model        {East-longitude positive}
Center radii    : 6378.1 x 6378.1 x 6356.8 km     {Equator, meridian, pole}
Target primary  : Sun
Vis. interferer : MOON (R_eq= 1737.400) km        {source: DE431}
Rel. light bend : Sun, EARTH                      {source: DE431}
Rel. lght bnd GM: 1.3271E+11, 3.9860E+05 km^3/s^2
Small-body perts: Yes                             {source: SB431-N16}
Atmos refraction: NO (AIRLESS)
RA format       : HMS
Time format     : JD
RTS-only print  : NO
EOP file        : eop.190225.p190519
EOP coverage    : DATA-BASED 1962-JAN-20 TO 2019-FEB-25. PREDICTS-> 2019-MAY-18
Units conversion: 1 au= 149597870.700 km, c= 299792.458 km/s, 1 day= 86400.0 s
Table cut-offs 1: Elevation (-90.0deg=NO ),Airmass (>38.000=NO), Daylight (NO )
Table cut-offs 2: Solar elongation (  0.0,180.0=NO ),Local Hour Angle( 0.0=NO )
Table cut-offs 3: RA/DEC angular rate (     0.0=NO )
*******************************************************************************
Initial IAU76/J2000 heliocentric ecliptic osculating elements (au, days, deg.):
  EPOCH=  2455878.5 ! 2011-Nov-13.00 (TDB)         Residual RMS= .22681
   EC= .01024784032430004  QR= 43.44411272689836   TP= 2464244.5684678773
   OM= 304.346207583376    W=  217.2195892913094   IN= 2.243588614484329
  Equivalent ICRF heliocentric equatorial cartesian coordinates (au, au/d):
   X=-2.946605040351360E+01  Y= 2.945530211695031E+01  Z= 1.250246236885602E+01
  VX=-1.916654737250466E-03 VY=-1.595777206663596E-03 VZ=-8.023908465221660E-04
Asteroid physical parameters (km, seconds, rotational period in hours):
   GM= n.a.                RAD= n.a.               ROTPER= 300.24
   H= 5.3                  G= .150                 B-V= n.a.
                           ALBEDO= n.a.            STYP= n.a.
```

```
*******************************************************************************************************************
Date_____JDUT     R.A._(ICRF/J2000.0)_DEC               r         rdot           delta      deldot    S-T-O
*******************************************************************************************************************
$$SOE
2456330.500000000 *   09 05 49.50 +16 13 09.6  43.49328884923  -0.0233310 42.5094499776651   1.4794668   0.0863
2456330.506944444 *   09 05 49.46 +16 13 09.8  43.49328875566  -0.0233310 42.5094559079060   1.4778147   0.0864
2456330.513888889 *   09 05 49.43 +16 13 09.9  43.49328866209  -0.0233311 42.5094618328575   1.4768308   0.0866
2456330.520833333 *   09 05 49.40 +16 13 10.1  43.49328856851  -0.0233311 42.5094677552180   1.4765241   0.0867
2456330.527777778 *r  09 05 49.36 +16 13 10.2  43.49328847494  -0.0233311 42.5094736777193   1.4769023   0.0869
2456330.534722222 *   09 05 49.33 +16 13 10.4  43.49328838136  -0.0233312 42.5094796031219   1.4779717   0.0870
2456330.541666667 *   09 05 49.30 +16 13 10.5  43.49328828779  -0.0233312 42.5094855342090   1.4797376   0.0872
2456330.548611111 C   09 05 49.26 +16 13 10.6  43.49328819421  -0.0233313 42.5094914737817   1.4822037   0.0874
2456330.555555556 C   09 05 49.23 +16 13 10.8  43.49328810063  -0.0233313 42.5094974246532   1.4853722   0.0875
2456330.562500000 N   09 05 49.20 +16 13 10.9  43.49328800706  -0.0233314 42.5095033896441   1.4892444   0.0877
2456330.569444444 N   09 05 49.16 +16 13 11.1  43.49328791348  -0.0233314 42.5095093715761   1.4938200   0.0878
2456330.576388889 N   09 05 49.13 +16 13 11.2  43.49328781991  -0.0233315 42.5095153732674   1.4990972   0.0880
2456330.583333333 A   09 05 49.10 +16 13 11.3  43.49328772633  -0.0233315 42.5095213975268   1.5050732   0.0882
2456330.590277778 A   09 05 49.06 +16 13 11.5  43.49328763275  -0.0233315 42.5095274471486   1.5117435   0.0883
2456330.597222222 A   09 05 49.03 +16 13 11.6  43.49328753918  -0.0233316 42.5095335249070   1.5191027   0.0885
2456330.604166667     09 05 49.00 +16 13 11.8  43.49328744560  -0.0233316 42.5095396335511   1.5271437   0.0886
2456330.611111111     09 05 48.96 +16 13 11.9  43.49328735202  -0.0233317 42.5095457757994   1.5358583   0.0888
2456330.618055556     09 05 48.93 +16 13 12.1  43.49328725845  -0.0233317 42.5095519543346   1.5452368   0.0890
2456330.625000000     09 05 48.90 +16 13 12.2  43.49328716487  -0.0233318 42.5095581717986   1.5552686   0.0891
2456330.631944444     09 05 48.86 +16 13 12.3  43.49328707129  -0.0233318 42.5095644307875   1.5659415   0.0893
2456330.638888889     09 05 48.83 +16 13 12.5  43.49328697771  -0.0233319 42.5095707338462   1.5772421   0.0894
2456330.645833333     09 05 48.80 +16 13 12.6  43.49328688414  -0.0233319 42.5095770834640   1.5891561   0.0896
2456330.652777778     09 05 48.76 +16 13 12.8  43.49328679056  -0.0233320 42.5095834820699   1.6016677   0.0898
2456330.659722222     09 05 48.73 +16 13 12.9  43.49328669698  -0.0233320 42.5095899320273   1.6147602   0.0899
2456330.666666667     09 05 48.70 +16 13 13.1  43.49328660340  -0.0233320 42.5095964356301   1.6284155   0.0901
2456330.673611111     09 05 48.66 +16 13 13.2  43.49328650982  -0.0233321 42.5096029950982   1.6426146   0.0902
2456330.680555556     09 05 48.63 +16 13 13.3  43.49328641624  -0.0233321 42.5096096125730   1.6573376   0.0904
2456330.687500000     09 05 48.60 +16 13 13.5  43.49328632266  -0.0233322 42.5096162901134   1.6725634   0.0906
2456330.694444444     09 05 48.56 +16 13 13.6  43.49328622909  -0.0233322 42.5096230296922   1.6882699   0.0907
2456330.701388889     09 05 48.53 +16 13 13.8  43.49328613551  -0.0233323 42.5096298331918   1.7044342   0.0909
2456330.708333333     09 05 48.50 +16 13 13.9  43.49328604193  -0.0233323 42.5096367024009   1.7210325   0.0910
2456330.715277778     09 05 48.46 +16 13 14.1  43.49328594835  -0.0233324 42.5096436390109   1.7380401   0.0912
2456330.722222222     09 05 48.43 +16 13 14.2  43.49328585477  -0.0233324 42.5096506446129   1.7554317   0.0914
2456330.729166667     09 05 48.39 +16 13 14.3  43.49328576119  -0.0233324 42.5096577206946   1.7731811   0.0915
2456330.736111111     09 05 48.36 +16 13 14.5  43.49328566761  -0.0233325 42.5096648686372   1.7912613   0.0917
2456330.743055556     09 05 48.33 +16 13 14.6  43.49328557403  -0.0233325 42.5096720897130   1.8096450   0.0919
2456330.750000000     09 05 48.29 +16 13 14.8  43.49328548045  -0.0233326 42.5096793850829   1.8283041   0.0920
2456330.756944444     09 05 48.26 +16 13 14.9  43.49328538687  -0.0233326 42.5096867557942   1.8472099   0.0922
2456330.763888889     09 05 48.23 +16 13 15.0  43.49328529328  -0.0233327 42.5096942027787   1.8663335   0.0923
2456330.770833333     09 05 48.19 +16 13 15.2  43.49328519970  -0.0233327 42.5097017268504   1.8856453   0.0925
2456330.777777778     09 05 48.16 +16 13 15.3  43.49328510612  -0.0233328 42.5097093287047   1.9051155   0.0927
2456330.784722222     09 05 48.13 +16 13 15.5  43.49328501254  -0.0233328 42.5097170089166   1.9247140   0.0928
2456330.791666667     09 05 48.09 +16 13 15.6  43.49328491896  -0.0233328 42.5097247679396   1.9444103   0.0930
2456330.798611111     09 05 48.06 +16 13 15.7  43.49328482538  -0.0233329 42.5097326061050   1.9641740   0.0932
2456330.805555556     09 05 48.02 +16 13 15.9  43.49328473180  -0.0233329 42.5097405236212   1.9839742   0.0933
```

Figure 9: Formats used by the Sun and Primary ephemeris files

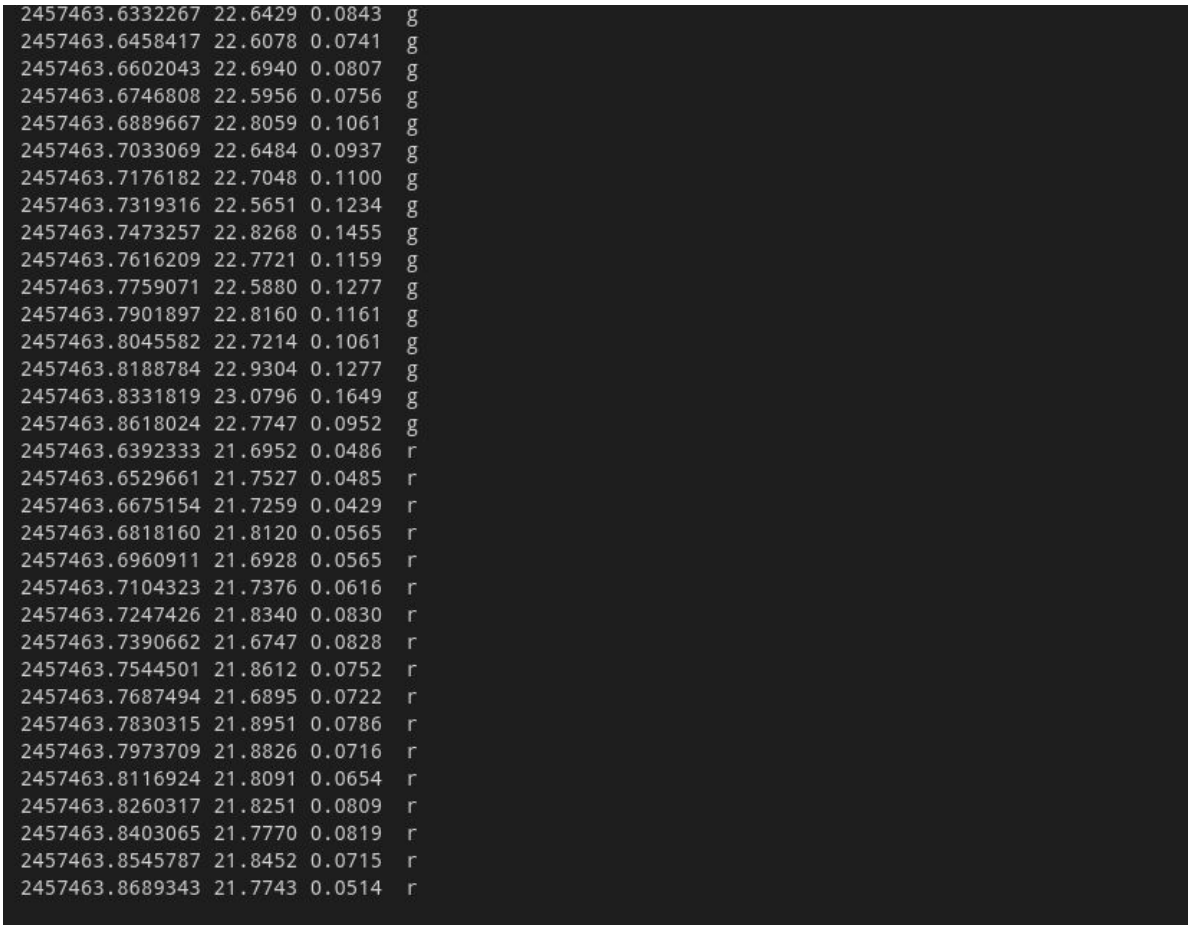

Figure 10: Format used by the observed date file

Vectors

Many of the parameters used within the forward model use vectors as inputs. When inputting any vector values please follow the following syntax:

- “#,#,#”

3D Vectors

Two parameters within the forward model use the type of 3D Vectors. Due to the nature of these values, the values have been split into two separate text inputs, to input these parameters please follow the following syntax:

- B1: “#,#,#”
- B2: “#,#,#”

Settings Requirements

When using the GUI, there are two setting requirements that must be set before any forward model call can begin.

Object(s) Setting

For the objects setting, before use of the forward model, as the user, you must decide whether the call will use one or two objects. If one object is selected several parameters will be unavailable to use.

Hapke Parameter Setting

For the Hapke parameter setting, the user must first decide whether they would like the default Hapke parameters (defined within ForwardModel.cpp) or their own custom Hapke parameters. If the user chooses to use the default parameters, all associated parameters will be locked so the user cannot change these values.

Usage

Using the forward model GUI is a very simple process. Once the code has been compiled using qmake and make(to generate the makefile) the user must then run the executable. To run the executable use the following command from the command line:

- ./ForwardModelUI

After that command has been run, the GUI will be displayed on the screen for usage.

Demo

A short demo of entering data into the GUI, calling the forward model and displaying the output of the GUI can be found under the “Demos” section on the team’s website[7].

If there are any issues or any additional help is required regarding the GUI module, please refer to the troubleshooting section below. If there are further questions, please refer to the emails listed at the end of the document and feel free to contact Bradley Donn Kukuk.

3.5 Video Generator

As the video generator is a small addition, its installation and use will be covered in this section. To use the video generator, FFmpeg must be installed. Since there is a

[7] https://www.cefns.nau.edu/capstone/projects/CS/2020/Andromeda-S20/

wide variety of ways to install this software, we would like to refer the user to view the download page for FFmpeg on their website[8].

The video generator is a script that compiles images that are in the “renders” folder of the forward model, NLM, or GUI. The user simply needs to execute the script from the licht-cpp/script directory using “./videoGenerator” and follow the given prompt. Before running, the user needs to check the renders. The render name will be something similar to NAME_Primary_###.jpeg. The name of the file will be required as input for one of the prompts asked when the script. Moreover, depending on how many integers exist for the “###” space, the user will need to input this number into the script when prompted. For example, an image might be named “SilaNunam_Primary_05.jpeg". Thus, there are 2 integers, and the user would specify the number to be 2. The images were formatted as something similar to “SilaNunam_Primary_001.jpeg”, there would be 3 integers and the user would specify this number when prompted.

If there are any issues or any additional help is required regarding the Video Generator module, please refer to the troubleshooting section below. If there are further questions, please refer to the emails listed at the end of the document and feel free to contact Matthew Amato-Yarbrough.

4. Maintenance

This product is low-maintenance and does not require any specific activities done to ensure its long-term health. The only aspect to note is in the topic of dependencies.

To reduce the required set-up, many dependencies have been stored in the codebase itself. As these dependencies have new releases, the versions in the codebase remain the same. This is beneficial in the fact that updates will not break the code, but disadvantageous when these updates provide performance and stability improvements. Thus, the client may consider manually updating these dependencies once every couple years.

4.1 Licht-cpp Dependencies

To update dependencies, download the new version and replace the old version in the ‘lib’ folder:

- lib/eigen3.3.7 → Eigen version 3.3.7

[8] https://www.ffmpeg.org/download.html

- lib/googletest → GoogleTest version 1.7.0
- lib/idl → IDL header file, no need to update
- lib/tinyjpeg → TinyJPEG library, no need to update

If changing the folder name, such as eigen3.3.7 to eigen3.3.8, the respective directory must be changed within CMakeLists.txt. They are simply stored near the top of the file and are easy to modify.

4.2 NLM Dependencies

To update dependencies, download the new version and replace the old version in the 'lib' folder:

- lib/matplotlibcpp/matplotlibcpp.h → matplotlibcpp header file, no need to update
- lib/googletest → GoogleTest version 1.7.0
- lib/amoebaNumericalRecipes/amoeba.h → Numerical Recipes header file, no need to update
- lib/amoebaNumericalRecipes/nr3.h → Numerical Recipes header file, no need to update

4.3 GUI Dependencies

All dependencies needed for the forward model GUI are as followed:

- qmake 3.0 or higher
- qt version 5 or higher
- compiled ForwardModel library (liblicht-cpp.a)

5. Troubleshooting

5.1 Forward Model

There are generally three types of problems that can arise:

- Build problems
- Input problems
- Logic problems

First and foremost, ensure the code is running correctly on your system by running the “tester”executable that comes from the compilation step. Ensure all tests pass before using the code. Build problems would occur during the set-up step. CMake is good in providing error messages about dependencies, while the actual make process can have some cryptic messages. More often than not, any build errors have messages that can be resolved by searching for a solution online. If not, the best course of action is to contact Zach, as he wrote the build system. There are many parameters involved in a build system and, without any experience debugging it, it is not suggested to try troubleshooting.

When the code does not work properly or crashes, the most common cause is invalid IDL parameters. C++ is highly sensitive to the types provided by IDL. For instance, if a double is expected and a float is given, the input is invalid. Before attempting to debug the software, triple-check that every parameter is the correct type. Many times we have encountered that a simple typo caused the crash, such as not including a “D” after a number or using the wrong type of slashes for file paths. After that, there is a ‘debug’ parameter that prints out much of the input that the forward model received. It also prints out at which stage the code failed. Enable it by setting the value to 1. Additionally, exceptions are thrown for specific cases of bad input. For instance, if an invalid ephemeris file path was given, an exception is thrown stating that. Use the output to determine if your input was interpreted correctly and to determine what section of code it fails in.

One huge caveat is that IDL does not seem to support the exceptions thrown from within C++. At least, this is true for version 7.0. Thus, everything stated in the above paragraph would likely be false for an IDL user. To account for this, an additional error-handling layer has been implemented. Any exceptions thrown in C++ that are not segmentation faults are caught and stored in variables allocated in IDL. In this manner, IDL users can still see the errors. More documentation lies within the demo script.

If IDL still crashes, i.e. if a segmentation fault is thrown, the most effective troubleshooting is using a C++ debugger. The most common tools for this are GDB and/or valgrind. The easiest way to debug is to create a test in “test/TestForwardModel.cpp” and recreate the IDL parameters by mimicking one of the existing tests. Re-compile the code (if a new test file was added, also call CMake) and call the tester executable. At this point, the user must be comfortable debugging C++ code. If not, the best course of action is to contact one of the developers with a bug report.

5.2 Triaxial Ellipsoids

As referred to in section 3.1, generating a single asteroid using an ellipsoid will produce incorrect lighting and, therefore, produce a faulty light curve. A way to solve this issue was discovered, which was by including a second object. To remedy this, create an object (preferably a sphere as they are the most simplistic) with the radius size of 0. This allows for the forward model to recognize that there is a second object and provide correct lighting for the ellipsoid, while not rendering the second object.

5.3 NLM

When using the NLM there are typically two types of problems that can arise:

- Build Problems
- User Input Problems
- Indefinite minimization

Build Problems

There are only two issues that can occur when trying to build the NLM. Either the user does not possess the necessary CMake version, Python version or both needed to build the NLM. To make sure this issue does not occur the installation of CMake 3.4 or higher is necessary. Additionally, Python 2.7 and the corresponding developer edition is necessary to compile the NLM.

User Input Problem

Several issues can arise when the user inputs improper values into the UserInput.txt files. Commonly, these issues stem from supplying the ephemerisPrimaryFile, ephemerisSunFile, and observedDataFilePath parameters with incorrect data. This problem can occur in a variety of ways such as providing the incorrect file path, a path to a file that does not exist or a path to a misnamed file or directory. With this in mind, checking that the paths and names of the files being supplied are correct is necessary . It should also be noted that when inputting the name of the file to be used for any of these parameters that the ".txt" extension is needed as well. This is to prevent the input of files that can not be parsed by the NLM or the forward model.

Indefinite Minimization

After running the NLM, users may experience the NLM continuously estimating the parameters for long periods of time. In some instances this process may be indefinite as the NLM continuously explores the parameters space for better

parameter estimates. To prevent this from occurring, users will have to tinker with the values they supply the UserInput.txt file. Specifically, the values of the tolerance level and the parameters that they have chosen to estimate.

5.4 GUI

While the GUI is built to handle most errors with detailed descriptions on how to fix these issues, there are a few issues known that are not covered by error handling. They are as follows:

- Unable to read external .txt file
- GUI timeout

Unable to Read External .txt File

There are a few reasons as to why this error may arise, and solving this issue is fairly simple.

- File was not placed within the GUI/data/ directory
    - If the file is not placed within the GUI/data/ directory, the GUI will not be able to find or read in the file. Simply place the file within the directory and the issue will be solved
- File type is not .txt
    - The only allowed external file type that can be passed to the forward model is a ".txt" file. If you are using any other file type, please try and convert that file to a .txt and the forward model will be able to accept the file.
- "Unable to read file" error
    - If this file occurs, the file that was input does not meet the specified layout for the forward model call. Please refer to the templates used above.

GUI Timeout

For the forward model GUI to time out or "get stuck" the user must have input a value that is not excessively large, or excessively small. The GUI is not stuck, it is running the program with the parameters requested. If the forward model call lasts longer than five minutes, the forward model GUI will shut down due to a timeout functionality. This time functionality was created to preserve the users hardware as some forward model calls may run for several hours with excessive data.

6. Conclusion

Team Andromeda is glad to have continued the development of such an interesting project. We hope the solution serves you well for many years to come.

With best wishes from the Team Andromeda development team,
Matthew Amato-Yarbrough (mba75@nau.edu)
Batai Finley (baf238@nau.edu)
Bradley Donn Kukuk (bdkukuk@zohomail.com)
John Jacobelli (johnpjacobelli@gmail.com)
Jessica Smith (j.l.smith.software@gmail.com)

While we are moving on to professional careers, we would be happy to answer any questions that arise about the codebase.

For further questions regarding the initial development of licht-cpp, please contact the original development team Paired Planet Technologies:
Zach Kramer (kramer.zachary.nau@gmail.com)
Brian Donnelly (bridonnelly17@gmail.com)