EMTGv9 Linux Build Guide

Noble Hatten*

Revision Date	Author	Description of Change
June 6, 2022	Noble Hatten	Migration of content from Markdown text to LaTeX.
August 8, 2023	Joseph Hauerstein	Created of public release version of Linux install guide.
December 7, 2023	Edwin Dove	Clarified instructions.

*Aerospace Engineer, NASA Goddard Space Flight Center, Flight Dynamics and Mission Design Branch Code 595

Contents

List of Acronyms

EMTG Evolutionary Mission Trajectory Generator

GCC GNU Compiler Collection

GMP GNU Multiple Precision Arithmetic Library

GSAD Ghosh Sparse Automatic Differentation

GSL GNU Scientific Library

GUI Graphical User Interface

ISL GNU Integer Set Library

MPC GNU Multiple Precision Complex Library

MPFR GNU Multiple Precision Floating-point Rounding

PEATSA Python EMTG Automated Trade Study Application

RHEL Red Hat Enterprise Linux

SNOPT Sparse Nonlinear OPTimizer

NLP Nonlinear Program

1 Introduction

The purpose of this document is to describe:

- How to obtain the Evolutionary Mission Trajectory Generator (EMTG) codebase using the public NASA GitHub Repository (Section 4).

- How to obtain the dependencies of EMTG, either by obtaining the dependencies invididually and building from source or by making use of package management tools (Section 3 and 5).
- How to compile EMTG from source on a Linux system (Section 6).

Other notes about the contents of this document:

- When listing Evolutionary Mission Trajectory Generator's (EMTG's) dependencies, specific versions of the dependencies are listed. These dependencies are known to work. The use of other versions of dependencies is not guaranteed to work.
- The instructions in this document are intended to be followed in the order in which they are given.

2 System Requirements

The instructions in this document are intended for a computer running on a 64-bit Linux installation. The instructions have been tested on Red Hat Enterprise Linux (RHEL) 8.8.

3 Obtaining Dependencies Using System Package Management

EMTG relies on external dependencies. This section describes how to obtain (and, when necessary, build) all dependencies with heavy use of package managers. In particular, the `yum` command and the Mamba Python package manager are used to install several dependencies.

3.1 System Package Manager Installations

The RHEL package manager yum is the utilized in this set of instructions for installing dependencies.

1. Install the latest gcc compilers that will be used in future steps by executing the following command:

```
yum install gcc
yum install gfortran
yum install gcc-c++
```

2. Install the following packages using the `yum install <package-name>-<version>` package management command:

- git (v 2.39.3)
- tcsh (v 6.24.10)
- lbzip2 (v 2.5)
- zip (v 3.0)

4 Obtaining the EMTG Git Repository

The latest EMTG source is located at `https://github.com/nasa/EMTG/`. This document will point to a specific EMTG version but users can obtain other versions from the aforementioned github location.

1. Navigate to your user home directory (e.g. /home/username/) in the Terminal window. If downloading to an alternate location, ensure that the directory has no spaces in the file path.
2. Download a zip of EMTG directly by using the following command:

   ```
   curl -LO https://github.com/nasa/EMTG/archive/refs/tags/v9.01.zip &&
   unzip v9.01.zip
   ```

   (a) Alternatively a user can download EMTG by cloning the repository.
   To clone the git repository, navigate to the directory where you want to place the EMTG files, and run the following command:

   ```
   git clone https://github.com/nasa/EMTG.git
   ```

3. Locate the folder that contains the 'EMTG-Config-template.cmake' file.
   For the remainder of this document, this folder will be identified as **<EMTG_root_dir>**
4. Create the **<EMTG_root_dir>**/HardwareModels/ folder
5. Copy the default.emtg_launchvehicleopt and empty.ThrottleTable files from the **<EMTG_root_dir>**/testatron/HardwareModels/ to the **<EMTG_root_dir>**/HardwareModels/ folder
6. Create the **<EMTG_root_dir>**/Universe/ folder
7. Create the **<EMTG_root_dir>**/Universe/ephemeris_files/ folder
8. Copy the *.emtg_universe files from the **<EMTG_root_dir>**/testatron/universe/ folder to the **<EMTG_root_dir>**/Universe/ folder
9. Copy the default.emtg_universe file from the **<EMTG_root_dir>**/PyEMTG/ folder to the **<EMTG_root_dir>**/Universe/ folder and rename to Sun_barycenters.emtg_universe

4.1 Mamba

4.1.1 Purpose

The Mamba Python package manager is used to create a Python environment in which to use the PyEMTG interface and install dependencies. PyEMTG is known to be compatible with Python 3.7. so it is strongly recommended that a Python 3.7 environment be created for PyEMTG. While there are many methods for installing Python, the method supported in this guide is using the Mamba package manager.

4.1.2 Download Location

The Mamba release known to be compatible with EMTG can be found at `https://github.com/conda-forge/miniforge/releases/22.9.0-2`. Additional information on obtaining and using Mamba is available at `https://mamba.readthedocs.io/en/latest/#`. To install Mamba, follow the steps below.

4.1.3 Dependency Installation Instructions

1. Navigate to your user home directory (e.g. /home/username/) in the Terminal window.
2. Execute the following command in the terminal to get the installation script for your flavor of Linux:

   ```
   curl -LO "https://github.com/conda-forge/miniforge/releases/download/22.9.0-2
   /Mambaforge-$(uname)-$(uname -m).sh"
   ```

3. Execute the installation script using the following command:

   ```
   bash "Mambaforge-$(uname)-$(uname -m).sh"
   ```

4. Follow the prompts given by the script.
   Once the files have been downloaded and installed, the script will ask you if you want to run `conda init`. Choose to run `conda init`. If you do not, you will have to run the following command to finish setting up Mamba for the current user:
   `eval "$(path/to/conda shell.bash hook)" && conda init`
5. Close and reopen the Terminal window for the conda initialization to finalize.
6. Create a Python environment called "PyEmtgEnv" the following conda command:

   ```
   conda create -n PyEmtgEnv python=3.7
   ```

   - *NOTE: PyEMTG is known to specifically NOT be compatible with Python 3.10 because wxWidgets is not compatible with Python 3.10.*

7. Activate the created Python environment by running the following command:
   `conda activate PyEmtgEnv`

8. Install all the python packages by executing the following commands individually or copying the block of code and pasting into a terminal window:
   (It may take a while for all the packages to be found and installed. The package versions listed are those which EMTG and PyEMTG have been tested.)

```
pip install astropy==4.3.1
pip install spiceypy==5.1.0
pip install jplephem==2.17
pip install matplotlib==3.5.3
pip install numpy==1.21.6
pip install scipy==1.7.3
pip install pandas==1.3.5
```

   - *NOTE: The wxPython package is not listed in the above list even though the PyEMTG Graphical User Interface (GUI) depends on wxPython and wxPython may be installed via* ***pip*** *on Windows. This is because* ***pip*** *may* not *be used to install wxPython on Linux. Installing wxPython on Linux is a more involved procedure, and is beyond the scope of this document at this time, though it may be added at a later date. As a result, these instructions as-is do not allow a user to use the PyEMTG GUI, though other features of PyEMTG are usable.*
   - *See this web page for more details on installing wxPython on Linux:* ***https://wxpython.org/blog/2017-08-17-builds-for-linux-with-pip/index.html***

9. Execute the following command to list the python packages installed and verify they are the versions mentioned in the previous step:

```
pip list
```

5 Manually Obtaining and Setting Up Other Dependencies

This section describes how to obtain (and, when necessary, build) the remaining dependencies.

5.1 Random-Number Utilities

5.1.1 Purpose

EMTG depends on randutils for random number generation.
EMTG is known to work with revision 2 of randutils.

5.1.2 Download Location

The main page for the software distributions is in the following website:
`https://gist.github.com/imneme/540829265469e673d045`

The software revision needed for the EMTG version indicated in this guide can be obtained from the following location:
(In the event the url is no longer active, navigate to the aforementioned software website to find the specific version)
`https://gist.github.com/imneme/540829265469e673d045/8486a610a954a8248c12485fb4cfc390a5f5f854`

5.1.3 Dependency Installation Instructions

1. Navigate to **<EMTG_root_dir>**/src/Math/ directory in the Terminal window
2. Execute the following command to navigate to the Math directory and download randutil:
   `curl -LO https://gist.github.com/imneme/540829265469e673d045/raw/8486a610a954a8248c12485fb4cfc390a5f5f854/randutils.hpp`

5.2 GCC

5.2.1 Purpose

GNU Compiler Collection (GCC) is required for compiling C++ and Fortran code, but it takes a long time to build and requires you to have another C, C++, and Fortran compiler on your machine. Therefore, it is highly recommeneded to simply download GCC using a package manager. GCC version 9.5.0 is known to work with EMTG.

5.2.2 Download Location

The version of GCC known to be compatible with EMTG can be found at `https://sourceware.org/pub/gcc/releases/gcc-9.5.0/`

5.2.3 Dependency Installation Instructions

1. Create a directory to house various multi-user dependencies needed for EMTG:

   ```
   mkdir /Utilities/
   ```

2. Navigate to the Utilities directory by executing the following command:

   ```
   cd /Utilities/
   ```

3. Download the GCC tarball using the following command:

```
curl -LO https://mirrorservice.org/sites/sourceware.org/pub/gcc/releases/gcc-9.5.0/gcc-9.5.0.tar.gz
```

4. Extract the tarball by executing the following command:

```
tar -xzf gcc-9.5.0.tar.gz
```

5. Rename the extracted directory in preparation of an out-of source build:

```
mv gcc-9.5.0 gcc-9.5.0-src
```

6. Download the GCC external prerequisite packages by executing the following commands:

```
cd gcc-9.5.0-src
./contrib/download_prerequisites
```

NOTE: The following prerequisites would have been downloaded if the download_prerequisites script executed successfully:

- GNU Multiple Precision Arithmetic Library (GMP): gmp-6.1.0.tar.bz2
- GNU Multiple Precision Floating-point Rounding (MPFR): mpfr-3.1.4.tar.bz2
- GNU Multiple Precision Complex Library (MPC): mpc-1.0.3.tar.gz
- GNU Integer Set Library (ISL): isl-0.18.tar.bz2

NOTE: If the download_prerequisites script does not work, obtain the prerequisites manually by downloading them from the locations below, placing them in the `gcc-9.5.0-src/` directory, extracting them in that directory, then rerunning the download_prerequisites script mentioned in this step:

- GMP: `https://gcc.gnu.org/pub/gcc/infrastructure/gmp-6.1.0.tar.bz2`
- MPFR: `https://gcc.gnu.org/pub/gcc/infrastructure/mpfr-3.1.4.tar.bz2`
- MPC: `https://gcc.gnu.org/pub/gcc/infrastructure/mpc-1.0.3.tar.gz`
- ISL: `https://gcc.gnu.org/pub/gcc/infrastructure//isl-0.18.tar.bz2`

7. Create a directory for the out-of-source build to reside by executing the following commands:

```
mkdir /Utilities/gcc-9.5.0
cd /Utilities/gcc-9.5.0
```

8. Perform the out-of-source build by executing the following command:

```
../gcc-9.5.0-src/configure --prefix=/Utilities/gcc-9.5.0 --disable-multilib
--program-suffix=-9.5.0 --enable-languages=c,c++,fortran
```

9. Execute the following commands to build the final executable:
   (<number-of-cores-available> is the integer number of cores to be used to perform the build.)

   ```
   make -j <number-of-cores-available>

   make install -j <number-of-cores-available>
   ```

   NOTE: Building GCC from source takes a long time (>1 hour) so be prepared to let the process execute without interruptions.

5.3 CMake

5.3.1 Purpose

EMTG has a CMake-based build system, and is known to be compatible with CMake 3.23.2.

5.3.2 Download Location

The version of CMake known to be compatible with EMTG can be found at `https://github.com/Kitware/CMake/releases/v3.23.2/`.

5.3.3 Depedency Installation Instructions

1. Navigate to the Utilities directory by executing the following command:

   ```
   cd /Utilities/
   ```

2. Download CMake using the following command:

   ```
   curl -LO https://github.com/Kitware/CMake/releases/download/v3.23.2/cmake-
   3.23.2-linux-x86_64.tar.gz
   ```

3. Extract the tarball with the pre-built binaries by using the following command:

   ```
   tar -xzf cmake-3.23.2-linux-x86_64.tar.gz
   ```

4. Rename the cmake directory using the following command:

```
mv /Utilities/cmake-3.23.2-linux-x86_64/ /Utilities/cmake-3.23.2/
```

5.4 Linux Environment Variables

5.4.1 Purpose

In order to make it easier to use EMTG and PyEMTG, users frequently edit their `.bashrc` and/or `.bash_profile` files, which are located in a user's home directory. For example, users might want to create aliases that point to the appropriate versions of compilers and add directories to library search paths. In particular, it is important that the paths to the GCC, Sparse Nonlinear OPTimizer (SNOPT), and Boost libraries are on the `LD_LIBRARY_PATH`. The instructions in this section assume that the user will make use of the bash files.

1. Create or update the `.bash_profile` file in the user's home directory using the following example as a reference:
   NOTE: The contents will vary depending on the user's setup. In this example, it is assumed that GCC, and CMake were installed in `/Utilities`.
   NOTE: In later sections when Boost, and SNOPT are installed verify that the location used matches with the contents of the `.bash_profile` file.
   NOTE: This `.bash_profile` is given at this point in the procedure—that is, after GCC and CMake have been installed but before other dependencies have been installed—to allow the user to place the newly installed GCC and CMake on their `PATH` and GCC on their `LD_LIBRARY_PATH` so that they may be used to build other dependencies.

```
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export CC=/Utilities/gcc-9.5.0/bin/gcc-9.5.0
export CXX=/Utilities/gcc-9.5.0/bin/g++-9.5.0
export FC=/Utilities/gcc-9.5.0/bin/gfortran-9.5.0

PATH=/Utilities/gcc-9.5.0/bin:/Utilities/cmake-3.23.2/bin:$PATH
LD_LIBRARY_PATH=/Utilities/gcc-9.5.0/lib:/Utilities/gcc-9.5.0/lib64:$LD_LIBRARY_PATH
LD_LIBRARY_PATH=/Utilities/snopt-7.6/lib/.libs:$LD_LIBRARY_PATH
LD_LIBRARY_PATH=/Utilities/boost-1.79.0/stage/lib:$LD_LIBRARY_PATH
```

```
export PATH
export LD_LIBRARY_PATH
```

2. Apply the changes to your terminal using the command below:

   ```
   source ~/.bash_profile
   ```

3. Verify that the LD_LIBRARY_PATH variable is properly initialized by executing the following command:

   ```
   echo $LD_LIBRARY_PATH
   ```

4. Close and reopen the Terminal window.

5.5 GSL

5.5.1 Purpose

EMTG depends on GNU Scientific Library (GSL) for cubic-splining utilities. EMTG is known to work with GSL 2.7.0. If you already installed GSL in Section 3, skip this section.

5.5.2 Download Location

There are multiple ways to get GSL, but the method supported by the EMTG build system is to obtain the AMPL version, which has a CMake-based build system. (For this reason, CMake must be installed before GSL.) The version of GSL known to be compatible with EMTG is available at `https://github.com/ampl/gsl/releases/tag/20211111`.

5.5.3 Dependency Installation Instructions

1. Navigate to the Utilities directory by executing the following command:

   ```
   cd /Utilities/
   ```

2. Download AMPL GSL 2.7.0 using the following command:

   ```
   curl -LO https://github.com/ampl/gsl/archive/refs/tags/20211111.tar.gz
   ```

3. Extract the tarball using the following commands:

   ```
   tar -xzf 20211111.tar.gz
   ```

4. Rename the GSL directory and navigate into it using the following commands:

```
mv /Utilities/gsl-20211111/ /Utilities/gsl-2.7.0/
cd gsl-2.7.0
```

5. Create a build directory and navigate into it using the following commands:

```
mkdir build
cd build
```

6. Build the final executable using the following commands:
   NOTE: <number-of-cores-available> is the integer number of cores to be used to perform the build.

```
cmake .. -DNO_AMPL_BINDINGS=1
make -j <number-of-cores-available>
```

5.6 CSPICE

5.6.1 Purpose

EMTG depends on CSPICE for ephemeris-lookup utilities, and is known to work with CSPICE N0067.

5.6.2 Download Location

The version of CSPICE that is known to be compatible with EMTG can be found at `https://naif.jpl.nasa.gov/pub/naif/toolkit/C/`.

5.6.3 Dependency Installation Instructions

1. Navigate to the Utilities directory by executing the following command:

   ```
   cd /Utilities/
   ```

2. Download CSPICE N0067 using the following command:

   ```
   curl -LO https://naif.jpl.nasa.gov/pub/naif/toolkit/C/PC_Linux_GCC_64bit/packages/cspice.tar.Z
   ```

3. Extract the tarball and navigate into the newly created directory using the following commands:

```
tar -xzf cspice.tar.Z

cd cspice
```

4. Build the final executable using the following command:

```
./makeall.csh
```

   - If you encounter an error when running the `makeall.csh` script, you will have to build each component individually by navigating to each of the subdirectories within the `cspice/src/` directory and running the make script there. For example, the following commands will make the cspice core:

```
cd cspice/src/cspice

./mkprodct.csh
```

5.7 Boost

5.7.1 Purpose

EMTG depends on three components of Boost: filesystem, serialization, and system (and their dependencies). In addition, if a user wishes to build the EMTG PyHardware and Propulator components, then the python component of Boost is required. (Installation of the python component of Boost is described later in this section.) The user may also simply install *all* components of Boost; the only drawback of this approach is that Boost will use more hard drive space. If you already installed Boost in Section 3, skip this section.

5.7.2 Download Location

EMTG is known to work with Boost 1.79.0, which is available at `https://boostorg.jfrog.io/artifactory/main/release/1.79.0/`.

5.7.3 Dependency Installation Instructions

1. Navigate to the user's home directory using the following command:

```
cd
```

2. Create a "user-config.jam" file using the following command:
   NOTE: This file is needed for building the Boost Python library

```
touch ~/user-config.jam
```

3. Open the file in a text editor and modify it so that the contents are:
   NOTE: Replace "/path/to" text with the path to your user mambaforge directory
   NOTE: The white space is important!

```
using python : 3.7 : /path/to/mambaforge/envs/PyEmtgEnv/bin/python3.7 ;
```

   - For additional information, see the Boost documentation at `https://www.boost.org/doc/libs/1_79_0/libs/python/doc/html/building/configuring_boost_build.html`.

4. Save and close the file once the changes are made

5. Navigate to the Utilities directory by executing the following command:

```
cd /Utilities/
```

6. Download Boost 1.79.0 by running the command:

```
curl -LO https://boostorg.jfrog.io/artifactory/main/release/1.79.0/source/boost_1_79_0.tar.gz
```

7. Extract the tarball, rename the directory, and navigate into the new directory using the following commands:

```
tar -xzf boost_1_79_0.tar.gz

mv boost_1_79_0 boost-1.79.0

cd boost-1.79.0
```

8. Execute the bootstrap script that will prepare files for being built using the following command:

```
./bootstrap.sh
```

9. Open the `boost-1.79.0/project-config.jam` file and look for the following lines:
   NOTE: The white space is important!

```
if ! gcc in [ feature.values <toolset> ]
{
   using gcc ;
}
```

10. Update the lines to read as followed to reflect the user's installation of GCC:

```
if ! gcc in [ feature.values <toolset> ]
{
   using gcc : 9.5.0 : /Utilities/gcc-9.5.0/bin/gcc-9.5.0 ;
}
```

11. Build the final executable using the following command:
    NOTE: The –with-python argument is needed for the EMTG PyHardware and Propulator components
    NOTE: The –config argument is to force the build process to use the specific input configuration file in the event an install is being made with root

    ```
    ./b2 --with-filesystem --with-serialization --with-system --with-python
    --config="/path/to/usr/user-config.jam"
    ```

5.8 SNOPT

5.8.1 Purpose

EMTG depends on the commercial Nonlinear Program (NLP) solver package SNOPT to perform gradient-based optimization. EMTG has interfaces known to work for SNOPT versions 7.5, 7.6, and 7.7.

5.8.2 Download Location

For more information on obtaining SNOPT, see `http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm`. The SNOPT source code should be extracted to a folder on your local system, which will be referred to as `<SNOPT_root_dir>`.

5.8.3 Dependency Installation Instructions

Once you have obtained SNOPT, you must modify the `<SNOPT_root_dir>/src/sn87sopt.f` file for it to work correctly with EMTG.

1. Navigate to approximately line 2791 for SNOPT 7.7 (line 2709 in SNOPT 7.6) to find the following:

   ```
   primalInf = primalInf/max(xNorm , one)
   ```

2. Comment (Fortran uses the ! character for comments) out the line so it looks like the following:

```
! primalInf = primalInf/max(xNorm , one)
```

Leaving this line uncommented can incorrectly mark certain solutions as feasible in SNOPT.

3. Save and close the file
4. Build the library from the source code
   (a) Perform the steps below for `SNOPT-7.7`
      i. Run the following commands:

```
cd <SNOPT_root_dir>

./configure --with-cpp

make install
```

   (b) Perform the steps below for `SNOPT-7.5` and `SNOPT-7.6`
      i. Run the following commands:

```
cd <SNOPT_root_dir>

./configure --with-cpp

make interface

make install
```

6 Building EMTG

This section describes how to build the EMTG executable after all dependencies have been acquired and set up.

6.1 The EMTG-Config.cmake File

There is a file called EMTG-Config-template.cmake in the EMTG repository root directory. Prior to building EMTG, a copy of this file called EMTG-Config.cmake must be created and the contents of this new file must be edited in order to reflect the locations of dependencies on the user's system.

The template file is heavily commented with specific instructions. In order to set the necessary information to appropriately edit the content of the file, the user must know where they installed CSPICE, SNOPT, Boost, and GSL. If using Ghosh Sparse Automatic Differentation (GSAD), the user must also know the location of their GSAD_2B.h file. With this information, open the EMTG-Config.cmake file in a text editor and set the following variables:

- Set the CSPICE_DIR variable to the full path to the root directory of CSPICE. For example, if CSPICE were placed in /Utilities/cspice, then the user would type in the EMTG-Config.cmake file:

```
set(CSPICE_DIR /Utilities/cspice)
```

- Set the SNOPT_ROOT_DIR variable to the full path to the root directory of SNOPT. For example, if SNOPT were placed in /Utilities/SNOPT-7.7, then the user would type in the EMTG-Config.cmake file:

```
set(SNOPT_ROOT_DIR /Utilities/SNOPT-7.7)
```

- Set the Boost-related variables. Depending on how Boost was installed, it may not be necessary to set all Boost-related variables because CMake can often "find" Boost without user inputs. BOOST_ROOT is the full path to the root directory of Boost. BOOST_INCLUDE_DIR is the full path to the directory that contains the Boost header files. BOOST_LIBRARY_DIRS is the full path to the directory that holds the Boost libraries after they are built. If the user installed Boost manually using the procedure described in Section 5.7, then the BOOST_ROOT, BOOST_INCLUDE_DIR, and BOOST_LIBRARY_DIRS variables should be set. If Boost were placed in /Utilities/boost-1.79.0, then the user would type in the EMTG-Config.cmake file:

```
set(BOOST_ROOT /Utilities/boost-1.79.0)
set(BOOST_INCLUDE_DIR ${BOOST_ROOT}/boost)
set(BOOST_LIBRARY_DIRS ${BOOST_ROOT}/stage/lib)
```

  In this case, all other Boost-related lines of the EMTG-Config.cmake file should be commented out.

- Set the GSL_PATH variable to the full path in which the GSL libraries are located. If GSL were installed from source (Section 5.5), then the GSL_PATH variable must be set to the full path to the GSL build directory. Thus, if GSL were placed in /Utilities/gsl, then the user would type in the EMTG-Config.cmake file:

```
set(GSL_PATH /Utilities/gsl/build)
```

- If using GSAD, set the GSAD_PATH variable to the full path to the directory in which GSAD_2B.h is located. For example, if GSAD_2B.h is located in /Utilities, then the user would type in the EMTG-Config.cmake file:

```
set(GSAD_PATH /Utilities)
```

6.2 Setting CMake Options

The CMake program is used to set additional options for the EMTG build.
The basic process to build EMTG using CMake is:
NOTE: For non-standard cmake options and troubeshooting see Section 6.2.1.

1. Prepare to use CMake to build EMTG:

   ```
   cd <EMTG_root_dir>

   mkdir build

   cd build

   ccmake ..
   ```

   If a user is *re*building EMTG, then the user does not have to recreate the build directory.

2. The `ccmake ..` command brings up the CMake "GUI"

3. Press `c` to Configure.

4. Select the options you want for each "Name" in the "Value" column. For the build described in this document, make sure that the following options are set:

   - Set SNOPT_MINGW_DLL `OFF`.
   - Set CMAKE_BUILD_TYPE `Release`.
   - Set BACKGROUND_MODE `ON`.

5. Press `t` (Toggle advanced mode) to see the advanced settings.

6. Go down to the options below and set as indicated:
   NOTE: In order to build Propulator and PyHardware utilities, the user must be sure that the Boost python library was built. (See Section 5.7.)

   - Set "BUILD_PROPULATOR" `ON`
   - Set "BUILD_PYHARDWARE" `ON`
   - Set "BUILT_IN_THRUSTERS" `ON`
   - Set "PROBEENTRYPHASE" `ON`

7. Press `c` to Configure again.

8. Set the python library paths to those for the PyEmtgEnv set in Section 4.1, instead of the default Mamba python libraries. The following example will provide guidance on how to configure the python library paths:

   - Set "PYTHON_EXECUTABLE" `/path/to/mambaforge/envs/PyEmtgEnv/bin/python3.7`

- Set "PYTHON_INCLUDE_DIR" `/path/to/mambaforge/envs/PyEmtgEnv/include/python3.7m`
- Set "PYTHON_LIBRARY" `/path/to/mambaforge/envs/PyEmtgEnv/lib/libpython3.7m.so`

9. Press `c` to Configure again.
10. Press `g` to Generate.
    NOTE: The "GUI" exits automatically if everything works as expected.
    NOTE: The EMTG executable is placed in `<EMTG_root_dir>/build/src`.

```
make -j <number-of-cores-available>

make install
```

6.2.1 Non-Standard CMake Options

The CMake options described in this section are relevant to developers and testers only and are not relevant for normal day-to-day use of EMTG. As with the other CMake options, after all CMake options have been set, the user must configure and generate the project in CMake, then build.

The EMTG Mission Testbed is primarily used when developing new code to check the accuracy of Jacobians against GSAD. In order to use the Mission Testbed with GSAD, the user must have set the GSAD_PATH variable in the EMTG-Config.cmake file, as described in Section 6.1. Then, in the CMake options, the user must:

- Set "RUN_MISSION_TESTBED" `ON`.
- Set "BUILD_EMTG_TESTBED" `ON`.
- Set "USE_AD_INSTRUMENTATION" `ON`.

Note that you may have to press `t` (Toggle advanced mode) to see some settings.

The EMTG Acceleration Model Testbed is used to print to file all acceleration model contributions for debugging and for comparing the Jacobians of the acceleration model against GSAD. In order to use the Acceleration Model Testbed with GSAD, the user must have set the GSAD_PATH variable in the EMTG-Config.cmake file, as described in Section 6.1. Then, in the CMake options, the user must:

- Set "RUN_ACCELERATION_MODEL_TESTBED" `ON`.
- Set "BUILD_EMTG_TESTBED" `ON`.

- Set "USE_AD_INSTRUMENTATION" `ON`.

When building EMTG with Propulator and PyHardware, there may be an error where the compiler cannot find `pyconfig.h`. In this case, the user must locate pyconfig.h on their machine (likely in the `/path/to/mambaforge/envs/PyEmtgEnv/include` directory) and copy the file to the `/path/to/mambaforge/envs/PyEmtgEnv/include/python3.7m` directory. This can be done using the command:

```
cp /path/to/mambaforge/envs/PyEmtgEnv/include/pyconfig.h /path/to/mambaforge/
envs/PyEmtgEnv/include/python3.7m
```

7 Executing EMTG Without PyEMTG

EMTG is executed by invoking the EMTGv9 executable with the name of an EMTG options file ("*.emtgopt") as the only command-line argument. For example:

```
cd <EMTG_root_dir>/build/src

./EMTGv9 /path/to/mission_name.emtgopt
```

8 PyEMTG

PyEMTG is a set of Python scripts that provide capabilities such as a GUI for EMTG and automated trade study utilities (i.e., Python EMTG Automated Trade Study Application (PEATSA)). Full PyEMTG documentation is contained in the EMTG repository's /PyEMTG/docs/ directory. In that directory, see PyEMTG_docs_readme.pdf for an overview of available PyEMTG documentation, which includes a reference to the PyEMTG User's Guide.