EMTG Tutorial: Config Files

Tim Sullivan [*]

August 4, 2023

Revision Date	**Author**	**Description of Change**
December 2, 2022	Tim Sullivan	Initial revision.
July 31, 2023	Joseph Hauerstein	Conversion to LaTeX.
August 4, 2023	Joseph Hauerstein	Addition of Known Issues section.

[*]Aerospace Engineer, The Aerospace Corporation

Contents

List of Known Issues

List of Acronyms

EMTG	Evolutionary Mission Trajectory Generator
GUI	Graphical User Interface
PEATSA	Python EMTG Automated Trade Study Application
SNOPT	Sparse Nonlinear OPTimizer
NLP	Nonlinear Program
MBH	Monotonic Basin Hopping

1 Introduction

This tutorial provides a brief introduction to the Python EMTG Automated Trade Study Application (PEATSA) used for creating and executing Evolutionary Mission Trajectory Generator (EMTG) trade studies. Example PEATSA configuration and output files for this tutorial are provided in the `EVM_PEATSA.zip` file included with these tutorials and should be referred to for any questions on setup. For additional details, refer to the PyEMTG User Guide included in `<EMTG_root_dir>/PyEMTG/docs/User_Guide/PyEMTG_user_guide.pdf`. Where `<EMTG_root_dir>` is the directory where you installed EMTG.

PEATSA is a set of Python scripts used to create and execute EMTG trade studies. PEATSA operates by taking an EMTG options file, known as the "base case", and varying parameters as defined by the user to create and execute variations of the base case. In this tutorial, you will create a PEATSA trade study which uses the Earth-Venus-Mars (EVM) trajectory first created in the Boundary Types tutorial and varies the launch date and launch vehicle delta-v upper bound to find the combination of these parameters that yields a trajectory that minimizes the required spacecraft delta-v.

Unlike previous tutorials which used PyEMTG in a Windows environment, this tutorial will be conducted in Linux entirely through the terminal and text editor. As a consequence, your Linux environment needs to be set up to execute EMTG. See the documentation in `<EMTG_root_dir>/docs/0_Users/build_system/linux_build_system` for more information.

NOTE: Be careful if copy/pasting apostrophes and quotation marks from this document to a text editor in a Linux environment. The transcription might not work the way you expect. Inspect each copy/pasted character to ensure it is correct, and change it if necessary.

2 Setup

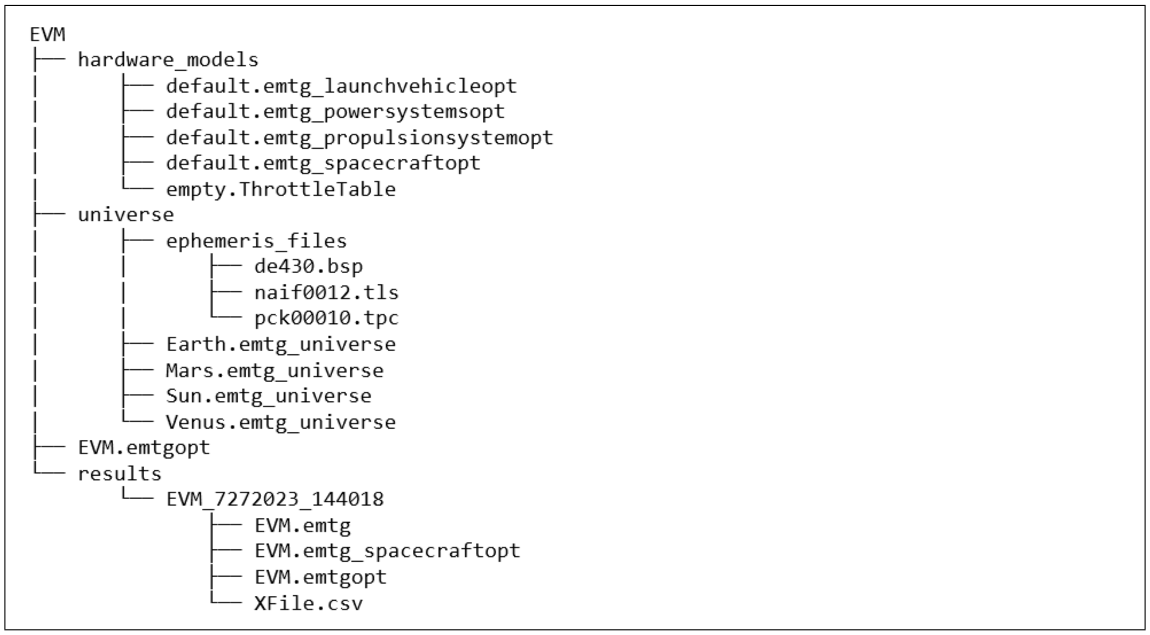
Begin with the Journey Boundaries tutorial directory (`Journey_Boundaries`). Using your Windows environment, open the `EVM.emtgopt` EMTG options file in PyEMTG. From previous tutorials, you should already have a converged solution for this setup. You can verify that the converged solution produces a feasible trajectory by setting "Inner-loop Solver Mode" to "Evaluate trialX" in the "Solver Options" tab. Then, set the converged solution as the initial guess using the "Trial decision vector or initial guess" option and run your options file. You should get a feasible solution. This tutorial will refer to this EMTG options file `EVM.emtgopt` as the PEATSA base case.

NOTE: When selecting the initial guess using the "Trial decision vector or initial guess" button, the GUI will not update to indicate the selected initial guess has been set, but the new TrialX will have been set in the options file.

Now, copy the Boundary Types tutorial directory into your Linux environment, including an example solution directory, which should be located in the `results` folder. In this tutorial, the directory will be named `EVM` rather than `Journey_Boundaries`. From this point on, the `EVM` directory will be referred to as `<EVM_dir>`.

You also need to copy the `EVM_universe` directory to the Linux environment, if it is not contained in `Journey_Boundaries`, because you need the EMTG universe files and ephemeris data to execute EMTG. The directory setup is shown in Figure 1.

The next step is to create a Python script used by PEATSA to run the trade study and a trade parameters CSV file which informs PEATSA which parameters will be varied and the values to use.



```
EVM
├── hardware_models
│       ├── default.emtg_launchvehicleopt
│       ├── default.emtg_powersystemsopt
│       ├── default.emtg_propulsionsystemopt
│       ├── default.emtg_spacecraftopt
│       └── empty.ThrottleTable
├── universe
│       ├── ephemeris_files
│       │       ├── de430.bsp
│       │       ├── naif0012.tls
│       │       └── pck00010.tpc
│       ├── Earth.emtg_universe
│       ├── Mars.emtg_universe
│       ├── Sun.emtg_universe
│       └── Venus.emtg_universe
├── EVM.emtgopt
└── results
        └── EVM_7272023_144018
                ├── EVM.emtg
                ├── EVM.emtg_spacecraftopt
                ├── EVM.emtgopt
                └── XFile.csv
```

Figure 1: PEATSA EVM Directory Setup.

2.1 PEATSA Script Generation

EMTG comes with a Python script to generate the initial PEATSA Python file called `PEATSA_script_generator.py`, located in the `<EMTG_root_dir>/PyEMTG/PEATSA` directory. Run the script generator from your `Journey_Boundaries` directory with the command:

```
python <EMTG_root_dir>/PyEMTG/PEATSA/PEATSA_script_generator.py
```

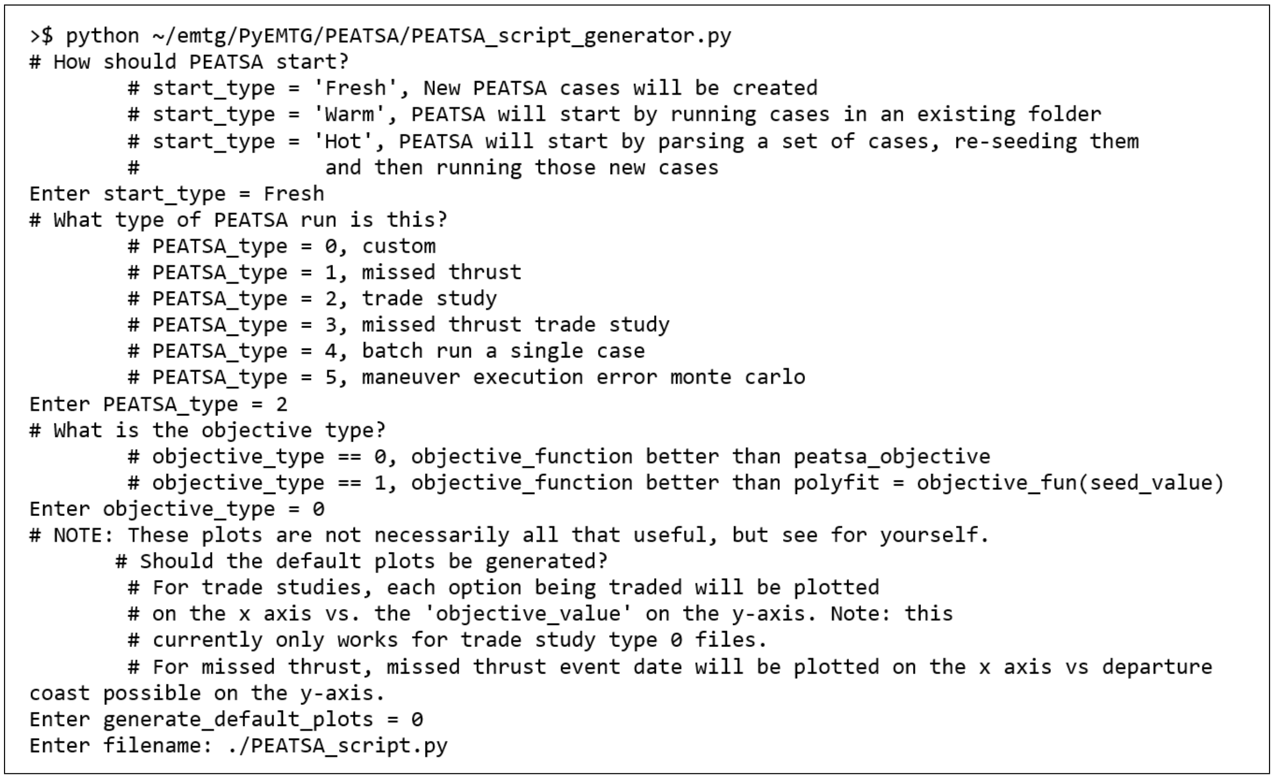
The script generator will begin asking a series of questions in the console and then generate the initial PEATSA Python script. Enter the answers listed below to the PEATSA script generator. The full question and answer sequence is shown in Figure 2.

- **How should PEATSA start?** "Fresh" – creating new PEATSA cases
- **What type of PEATSA run is this?** "2" – indicates a trade study
- **What is the objective type?** "0" – a new EMTG solution is superior to an old solution if the PEATSA objective value is better than the old value
- **Should the default plots be generated?** "0" – do not generate default plots

  NOTE: Default PEATSA plots are not functional at this time. The current workflow is to import PEATSA data into software like Excel or Python to perform data analysis and create plots.
- **Filename:** "<EVM_dir>/PEATSA_script.py" – include the full path to the mission directory in which the EVM base case was placed

After answering the script generator questions, you should have a `PEATSA_script.py` file in your mission directory. Some additional configuration is required to finish the PEATSA run setup.

First, a note on PyEMTG objects. Many of the options specified in the `PEATSA_script.py` file reference PyEMTG class attributes. Detailed documentation of these options is not yet available. However, users can reference the "MissionObjects.py", "JourneyOptions.py", "MissionEvent.py", and "Mission.py" files in `<EMTG_root_dir>/PyEMTG` when configuring different options. In particular, look at the class variables declared in the constructors of these classes to see what data is accessible and the syntax for accessing it. The table below lists a general guide to some of these objects. MissionOptions class attributes are values that define the options for an EMTG run; a MissionOptions object is created from a .emtgopt file. On the other hand, Mission class attributes refer to the results of an EMTG run; a Mission object is created from a .emtg file. In a PEATSA Python script, the MissionOptions and Mission objects for EMTG cases are accessed using the shorthand "MO" and "M", respectively (see Table 1). While these options correspond to actual Python objects in PyEMTG, they should be entered in the `PEATSA_script.py` file as strings (in quotes), not Python code.



```
>$ python ~/emtg/PyEMTG/PEATSA/PEATSA_script_generator.py
# How should PEATSA start?
        # start_type = 'Fresh', New PEATSA cases will be created
        # start_type = 'Warm', PEATSA will start by running cases in an existing folder
        # start_type = 'Hot', PEATSA will start by parsing a set of cases, re-seeding them
        #               and then running those new cases
Enter start_type = Fresh
# What type of PEATSA run is this?
        # PEATSA_type = 0, custom
        # PEATSA_type = 1, missed thrust
        # PEATSA_type = 2, trade study
        # PEATSA_type = 3, missed thrust trade study
        # PEATSA_type = 4, batch run a single case
        # PEATSA_type = 5, maneuver execution error monte carlo
Enter PEATSA_type = 2
# What is the objective type?
        # objective_type == 0, objective_function better than peatsa_objective
        # objective_type == 1, objective_function better than polyfit = objective_fun(seed_value)
Enter objective_type = 0
# NOTE: These plots are not necessarily all that useful, but see for yourself.
       # Should the default plots be generated?
        # For trade studies, each option being traded will be plotted
        # on the x axis vs. the 'objective_value' on the y-axis. Note: this
        # currently only works for trade study type 0 files.
        # For missed thrust, missed thrust event date will be plotted on the x axis vs departure
coast possible on the y-axis.
Enter generate_default_plots = 0
Enter filename: ./PEATSA_script.py
```

Figure 2: PEATSA Script Generation.

Object	Options/Results Covered
M.	Mission objects (Mission.py) – Results of an EMTG run
MO.	Mission options (MissionOptions.py) – Options used to define an EMTG run
MO.Journeys[i].	Journey Options (JourneyOptions.py) – Options found in the Journey Options panel in PyEMTG
M.Journeys[i].	Journey results (Journey.py) – Results from a specific Journey of an EMTG run
M.Journey[i].missionevents[j].	Events within a Journey (missionEvent.py) – event results from an EMTG solution. A mission event corresponds to a row in a .emtg file

Table 1: PyEMTG Objects.

Brackets are used to index array/list elements such as the list of Journeys (every array/list is 0-indexed because Python is 0-indexed).

PEATSA uses “1” or “0” to indicate True/ON and False/OFF respectively.

Open `PEATSA_script.py` in a text-editor. Make the following changes to the default values set by the script generator:

2.1.1 Path Options

- **run_name:** "PEATSA_run"
  - Each time PEATSA is run, it will save the results in a directory with this name, appending a number if necessary to avoid overwriting previous results.
- **working_directory:** the directory containing the `EVM.emtgopt` file
  - This is the directory in which the run_name folder will be created.
- **nCores:** Set the second element of the tuple to the number of parallel processes you want PEATSA to use when executing EMTG runs.
- **emtg_root_directory:** Full path to the location of the of the EMTG executable.
- **logfile:** Full path and name of file to which PEATSA informational output will be written.

2.1.2 Start Options

- **max_iterations:** 2
  - This will be set to a low number for this tutorial. After each iteration, the objective formula is evaluated for each case and compared to the best objective achieved in any previous iterations. This information is logged and information about the current iteration is used to seed (provide initial guesses) for the next iteration.
- **if_run_cases:** 1
  - This tells PEATSA to execute EMTG cases.

2.1.3 Initial Case Creation Options

- **trade_study_options_files:** ("<EVM_dir>/peatsa_options.csv", 1).
  - The list is populated with 2-element tuples. The first element of each tuple is a string containing the full path to a trade parameter CSV, and the second element is an integer specifying the type of the trade parameter CSV. (See Section 3.1.4. of the PyEMTG User Guide). One tuple is inserted into the list with the options file path (this will be created later) and a 1 to trade all values of each variable against all values of all other variables.

2.1.4 EMTG Control Options

- **killtime:** 100
  - This sets a maximum time, in seconds, that an EMTG instance is allowed to run before PEATSA kills the process. The comments in the PEATSA Python file give best practices for setting this value.
- **MBH_max_run_time:** 60
  - This defines how long in seconds each case is allowed to run in EMTG. Note that this value will be superseded if `MBH_max_run_time` is set in the override_options, described later and in Section 3.1.3 of the PyEMTG User Guide.

The killtime variable is required because Sparse Nonlinear OPTimizer (SNOPT) can encounter an unrecoverable failure that causes an EMTG run to never end, even if `MBH_max_run_time` is exceeded.

2.1.5 Optimization Options

- **objective_formula:** "M.total_deterministic_deltav"
  - This sets the PEATSA objective value; that is, how PEATSA decides if one EMTG run is better than another. The formula must be a function of data that is obtainable from a PyEMTG Mission object and MissionOptions object for a given EMTG run. Note that this objective value does not have to be the same as the objective function for a single EMTG run.
- **max_or_min:** "min"
  - You want to minimize total delta-v.

2.1.6 Sorting Options

- **fingerprint:** ["MO.launch_window_open_date", "MO.Journeys[0].initial_impulse_bounds[1]"]
  - The list of strings in the fingerprint defines a unique EMTG case. A good starting point is to make the fingerprint a list of the trade parameter column headers.
  - These will set in the options CSV file later in the tutorial. The PEATSA run you create will vary the start of the launch window and the upper bounds on the impulse imparted by the launch vehicle to see what impact these have on the spacecraft delta-v.
- **seed_from_cases_that_havent_met_target:** 1
  - This variable defaults to 0 but typically should be 1. When 0/OFF EMTG will not seed new cases from previous runs unless the objective value has been reached. It's common to set the objective value to 0 or a very high number for minimization/maximization

problems, respectively, such as minimizing delta-v or maximizing dry mass. EMTG solutions will rarely achieve these values, meaning previous runs will never be used as seeds for new runs. This is usually not the desired behavior since PEATSA should select previous runs as initial guesses for future runs.

- **only_one_seed_in_all_seed_directions:** 1
  - NOTE: If you have multiple seed criteria, multiple PEATSA runs may be seeded by the same previous run, with each run given the same name. This means that multiple runs will be written to the same file. This causes problems parsing the solution. If this bug has been fixed, the value can be set to 0 or 1 at the user's discretion. If the seed_criteria list (see below) has only a single entry, then setting to 0 or 1 is OK.
- **seed_criteria:** [("MO.launch_window_open_date", -30.1, 30.1, 2), ("MO.Journeys[0].initial_impulse_bounds[1]", -1.1, 1.1, 2)]
  - For case A to be a potential seed for case B, case A's fingerprint must be identical to case B's fingerprint except for the value of the seed criterion element under evaluation. This list specifies the range for a seed criterion in which a previous run is allowed to serve as a seed for a future run.
  - This list is made up of 4-element tuples. The first element is a string that is the criterion to be compared between an EMTG case and a potential seed case. This should be a trade parameter as defined in the trade parameters CSV file. (See Section 3.1.4 of the PyEMTG User Guide.) The second element is a real number that is the maximum negative seed range. In other words, when evaluating a potential seed case, how much less than the current case can the seed criterion be for the seed case to still be considered a potential seed? The third element is a real number that is the maximum positive seed range. In other words, when evaluating a potential seed case, how much greater than the current case can the seed criterion be for the seed case and still be considered a potential seed? The fourth element is an integer and is the seed selection criterion.
  - The values above indicate that a previous run launch window open date can be used as a seed for 30 days in either direction, and that the initial impulse upper bound can be used as a seed for 1.1 km/s above or below the seed run's value. The upper and lower values in the seed criteria were selected to slightly overlap adjacent values set in the options CSV file. The [0] and [1] indicate that parameter is indexing an array or tuple. (In this case, Journeys[0] is indexing into the list of JourneyOptions objects and initial_impulse_bounds[1] is grabbing the upper bound on the initial impulse.
- **override_options:** [("1", "MO.HardwarePath = '<EVM_dir>/hardware_models/' "), ("1", "MO.universe_folder = '<EVM_dir>/universe/' "), ("1", "MO.run_inner_loop = 1"), ("1", "MO.seed_MBH = 1")]
  - This list is used to override EMTG options from the base case. For example, the base case might be generated in Windows using PyEMTG, and this section can be used for a PEATSA run on a Linux server on multiple cores by overriding the paths to the universe and hardware directories. The list should be comprised of two-element tuples where the first element is a logical condition that, when it evaluates to true, indicates the override should be applied. To always apply the override, place a "1" as the first element. The second element is an assignment statement indicating what option to override and the value to assign it.

  - For this tutorial, add override options for the universe and hardware directories because you are running on a different system than the system on which you created the base case, and the base case file paths are not present on the linux system. If your most recent run of your base case set “Inner-loop Solver Mode” to something other than “Monotonic Basin Hopping”, then you will also want to override that (1 corresponds to monotonic basin hopping for `run_inner_loop`). You also want to make sure that seeding MBH is turned on. See Section 3.1.3 of the PyEMTG User Guide for more options.

2.1.7 Post-Processing Options

- **extra_csv_column_definitions:** [(“C3(km^2/s^2)”, “M.Journeys[0].missionevents[0].C3”), (“Launch Date”, “M.Journeys[0].missionevents[0].GregorianDate”), (“Launch Date”, “M.Journeys[0].missionevents[0].JulianDate”)]
  - After each PEATSA iteration, PEATSA produces a CSV file that summarizes the results of each EMTG run in a single file. This section allows the user to add columns to the IterationX.csv output files produced by PEATSA and is used to show more information about the solutions EMTG finds and/or the options used for an EMTG run. A column is added by adding a two-element tuple to the extra CSV column definitions. The first element is a string defining the column header. The second element is a string that may be evaluated for a given EMTG run. Data must be obtainable from a PyEMTG Mission object or MissionOptions object for a given EMTG run. These objects are accessed via “M.” and “MO.”, respectively.

2.2 PEATSA Trade Parameters File

This tutorial has referred several times to a CSV file that defines trade study parameters. Now you will make that file. This file can be made in Excel and saved as a CSV or any text editor with careful attention to formatting. (Using Excel is usually easier because it handles formatting automatically.) There are three types of trade parameter files that can be used in PEATSA. You will be using “Type 1”, which trades all values of each variable against all values of all other variables. In other words, an EMTG case is created for every permutation of the values listed in rows 4 and below of the CSV (see the PyEMTG User Guide Section 3.1.4 for more information).

The first row of the CSV should contain the full path to the base case directory. The second row should contain the file name to the base case EMTG options file located in the directory in row 1. In other words, when the file name in row 2 is appended to the directory path in row 1, you get the complete path to the base EMTG options file. (The first row entry does not need to end with a file separation character.) The third row should contain the names of the trade parameters used for the PEATSA run. Set these to `MO.launch_window_open_date` and `MO.Journeys[0].initial_impulse_bounds[1]`. The [1] indicates the upper bound value and the [0] indicates this applies to the first Journey. Thus, the trade parameters are the earliest date the spacecraft is allowed to launch, and the maximum impulse allowed to be provided by the launch vehicle.

Rows 4 and below contain the values to be tested for each of the parameters labeled in row 3. For `MO.launch_window_open_date`, set the value in row 4 to 61041, which is the modified Julian date for 1 January 2026. In the rows below, enter additional Julian dates spaced 30 days apart. (Note that this value of 30 days is set in conjunction with the value for the upper bound of "Wait time bounds" for Journey 0 of the base case, which is set to be slightly greater than 30 days.) Thus, you will be attempting to launch in 30-day "bins". For `initial_impulse_bounds[1]`, begin with 5 km/s and increase by 1 km/s in each row. Add as many rows as you like, remembering that the total runs will scale with the product of the number of values in each column. The spacing for both corresponds to the values set in seed criteria.

NOTE: The gap between launch window open dates in this tutorial is 30 days so that a year of launch dates can be examined with a small number of EMTG cases. In practice, however, the gap is usually set to a smaller value, such as 5 days, in order to make "adjacent" cases have similar enough solutions that using a feasible solution for one case as an initial guess for an adjacent case is likely to result in a new feasible case after optimization is performed. In general, proper selection of the distance between adjacent values of trade parameters is important in order for PEATSA to fully take advantage of its seeding capabilities.

If editing this file using a text editor, ensure there are commas between values in each row with no spaces and no empty lines below the trade parameter values. If either of these are present, PEATSA will throw an error at runtime. If using Excel, saving the file as a CSV should result in a correctly formatted options file.

Save your CSV. Remember that the path to and name of the CSV need to be consistent with the value given to `trade_study_options_files` in the PEATSA Python file. An example file is shown in Figure 3.

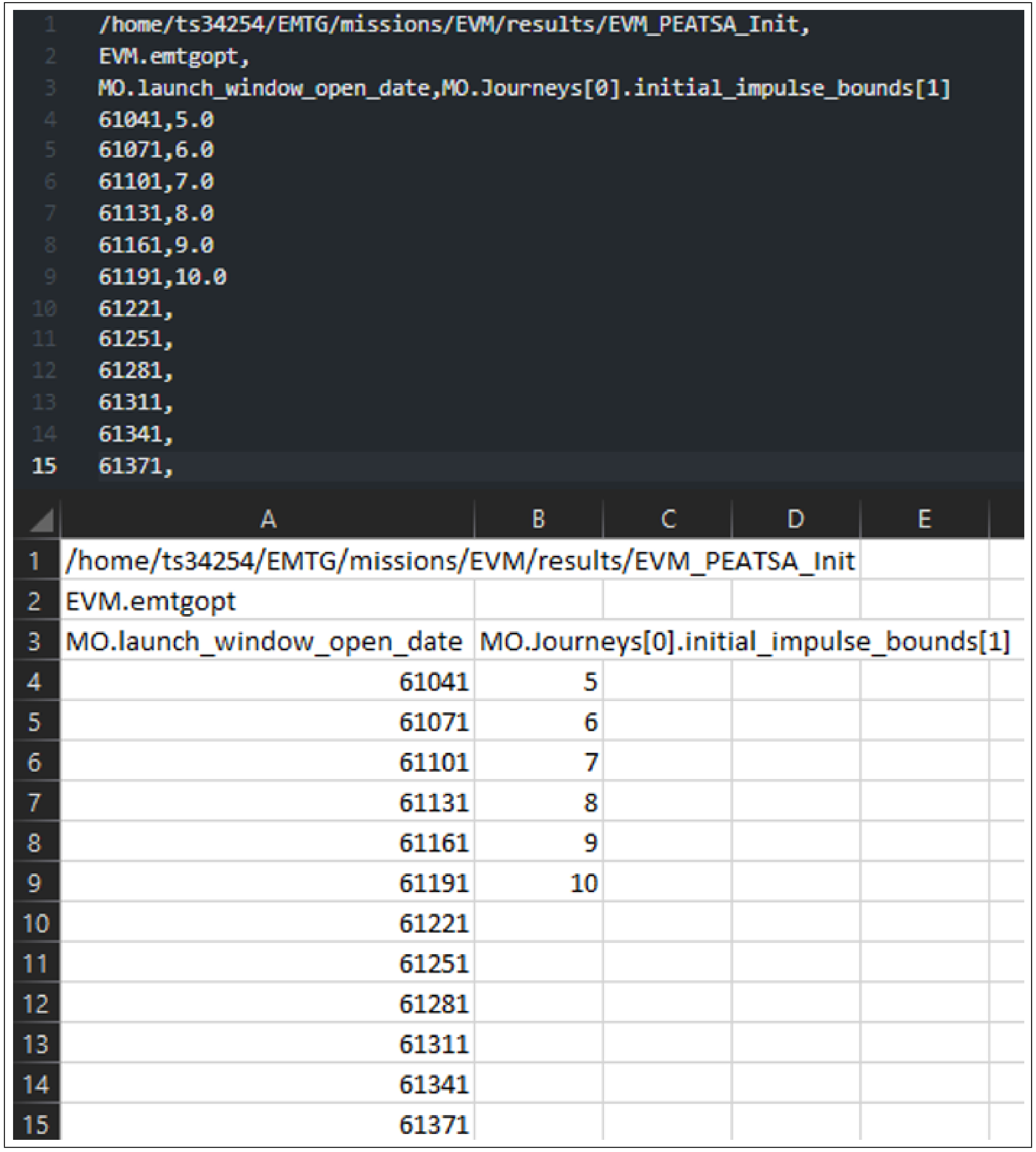
3 Run PEATSA

Once the files required to set up a PEATSA run have been created, the Linux command to execute a PEATSA run is (all on one line):

```
nohup python <EMTG_root_dir>/PyEMTG/PEATA/PEATSA.py <EVM_dir>/PEATSA_script.py >
/path/to/terminal.out 2>&1 &
```

This command will:

- Use Python to execute `PEATSA.py` with `PEATSA_script.py` as a command-line argument.
- Ignore the hang-up signal so that the process does not stop if the user logs out or is disconnected (`nohup`).
- Redirect standard output and standard error to a file named `terminal.out` (`> /path/to/terminal.out 2>&1`). PEATSA writes output to this file, so the user can monitor the progress of a PEATSA run by examining the contents of this file or the file set for the `logfile` variable



```
/home/ts34254/EMTG/missions/EVM/results/EVM_PEATSA_Init,
EVM.emtgopt,
MO.launch_window_open_date,MO.Journeys[0].initial_impulse_bounds[1]
61041,5.0
61071,6.0
61101,7.0
61131,8.0
61161,9.0
61191,10.0
61221,
61251,
61281,
61311,
61341,
61371,
```

	A	B	C	D	E
1	/home/ts34254/EMTG/missions/EVM/results/EVM_PEATSA_Init				
2	EVM.emtgopt				
3	MO.launch_window_open_date	MO.Journeys[0].initial_impulse_bounds[1]			
4	61041	5			
5	61071	6			
6	61101	7			
7	61131	8			
8	61161	9			
9	61191	10			
10	61221				
11	61251				
12	61281				
13	61311				
14	61341				
15	61371				

Figure 3: Trade Parameter CSV File Examples.

in the PEATSA Python setup script. The user can also see if a PEATSA run has died unexpectedly by examining this file. Keep in mind that once PEATSA finds the `logfile`, it will only output to that file.

- Run in the background (`&`).

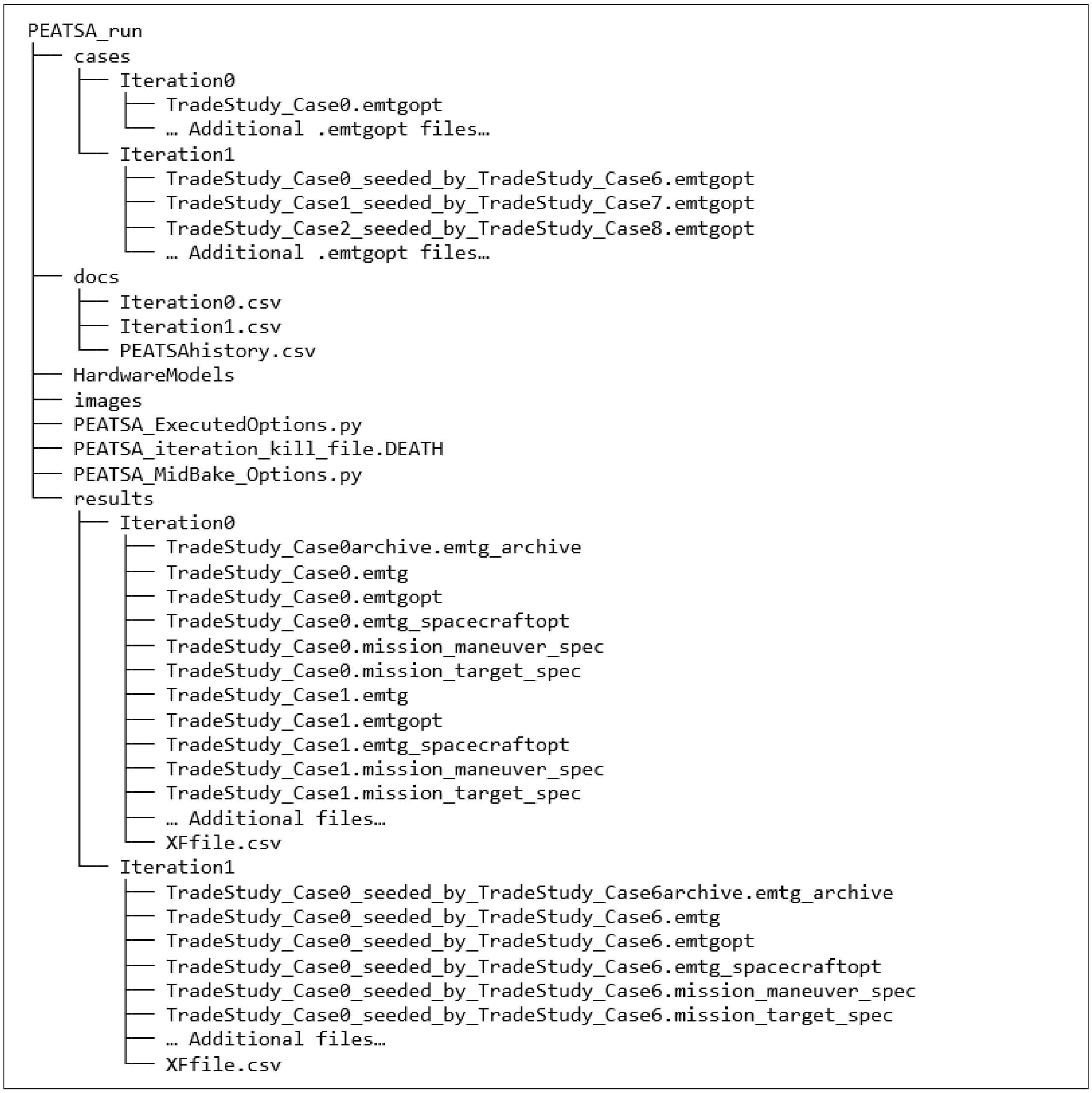
As soon as the PEATSA run is kicked off, the folder set for `run_name` should be created. You can follow the progress of your PEATSA run at its most basic level using the Linux `top` command. You should see python and/or EMTG process(es) being executed if everything is going well. More detailed logging information can be seen by looking at the contents of the `logfile` using the command:

```
tail -f /path/to/logfile
```

4 Post-Process

PEATSA runs produce a lot of files. Figure 4 shows a reduced example of the contents of a PEATSA output directory with `run_name = "PEATSA_run"` after two PEATSA iterations (iteration 0 and iteration 1). Each case that PEATSA creates for PEATSA iteration X results in a new EMTG options file placed in the `cases/IterationX` directory. The normal EMTG output is placed in the `results/IterationX` directory. You'll note that the filenames include which preceding case seeded the current case. A summary of the results for each iteration will be placed in an `IterationX.csv` file in the docs directory. The `PEATSAhistory.csv` file, also in the docs directory, provides basic summary information such as the how many EMTG runs executed, how many converged, how many failed, and how many improved. This can be useful for determining whether a PEATSA run should be stopped, adjusted, and/or restarted. The file is overwritten after each iteration, so it always provides up-to-date information from all the iterations prior the current one. An example is shown in Table 2.

The `IterationX.csv` file contains the seed criteria used for each run, PEATSA objective, additional columns we set above, and other information. There is one row for each unique fingerprint of the trade study, and its contents represent the best solution that PEATSA has found for that fingerprint in any PEATSA iteration up to and including iteration X. The EMTG results themselves are in `PEATSA_run/results/IterationX` and the EMTG options file for each case are in `PEATSA_run/cases/IterationX`.



```
PEATSA_run
├── cases
│   ├── Iteration0
│   │   ├── TradeStudy_Case0.emtgopt
│   │   └── … Additional .emtgopt files…
│   └── Iteration1
│       ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.emtgopt
│       ├── TradeStudy_Case1_seeded_by_TradeStudy_Case7.emtgopt
│       ├── TradeStudy_Case2_seeded_by_TradeStudy_Case8.emtgopt
│       └── … Additional .emtgopt files…
├── docs
│   ├── Iteration0.csv
│   ├── Iteration1.csv
│   └── PEATSAhistory.csv
├── HardwareModels
├── images
├── PEATSA_ExecutedOptions.py
├── PEATSA_iteration_kill_file.DEATH
├── PEATSA_MidBake_Options.py
└── results
    ├── Iteration0
    │   ├── TradeStudy_Case0archive.emtg_archive
    │   ├── TradeStudy_Case0.emtg
    │   ├── TradeStudy_Case0.emtgopt
    │   ├── TradeStudy_Case0.emtg_spacecraftopt
    │   ├── TradeStudy_Case0.mission_maneuver_spec
    │   ├── TradeStudy_Case0.mission_target_spec
    │   ├── TradeStudy_Case1.emtg
    │   ├── TradeStudy_Case1.emtgopt
    │   ├── TradeStudy_Case1.emtg_spacecraftopt
    │   ├── TradeStudy_Case1.mission_maneuver_spec
    │   ├── TradeStudy_Case1.mission_target_spec
    │   ├── … Additional files…
    │   └── XFfile.csv
    └── Iteration1
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6archive.emtg_archive
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.emtg
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.emtgopt
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.emtg_spacecraftopt
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.mission_maneuver_spec
        ├── TradeStudy_Case0_seeded_by_TradeStudy_Case6.mission_target_spec
        ├── … Additional files…
        └── XFfile.csv
```

Figure 4: Example PEATSA Run Directory.

Iteration	0	1
Cases Created	72	72
Cases Run	72	72
Ran Unseeded	72	0
Cases Finished	72	71
Iteration Cases Converged	72	71
PEATSA Cases Converged	72	72
First Convergence this Iteration	72	0
Improvement this Iteration	72	39
Number of PEATSA Cases That Converged on First NLP Solve This Iteration	36	71
Average Number of Solution Attempts for all Cases This Iteration	546.89	533.75
Average Solution Attempt Number of BEst Feasible Solution Amoung All Feasible Cases This Iteration	277.21	201.75
Times Using Seed Before Improvement	72	6

Table 2: Example `PEATSAhistory.csv` Contents.

The `IterationX.csv` files are especially useful for exploring and presenting the trade space. Add as many custom columns as necessary to understand the advantages and disadvantages of different mission configurations using the `extra_csv_column_definitions` list. For our Earth-Venus-Mars case, Figure 5 can be created using the iterations file, looking at the spacecraft delta-v required for different combinations of launch vehicle C3 (characteristic energy, the square of V-infinity which we traded in this PEATSA run) and launch date. Be careful – Rows of the `IterationX.csv` file whose `Filename` column starts with FAILURE did not converge and should be filtered out of any analyses.

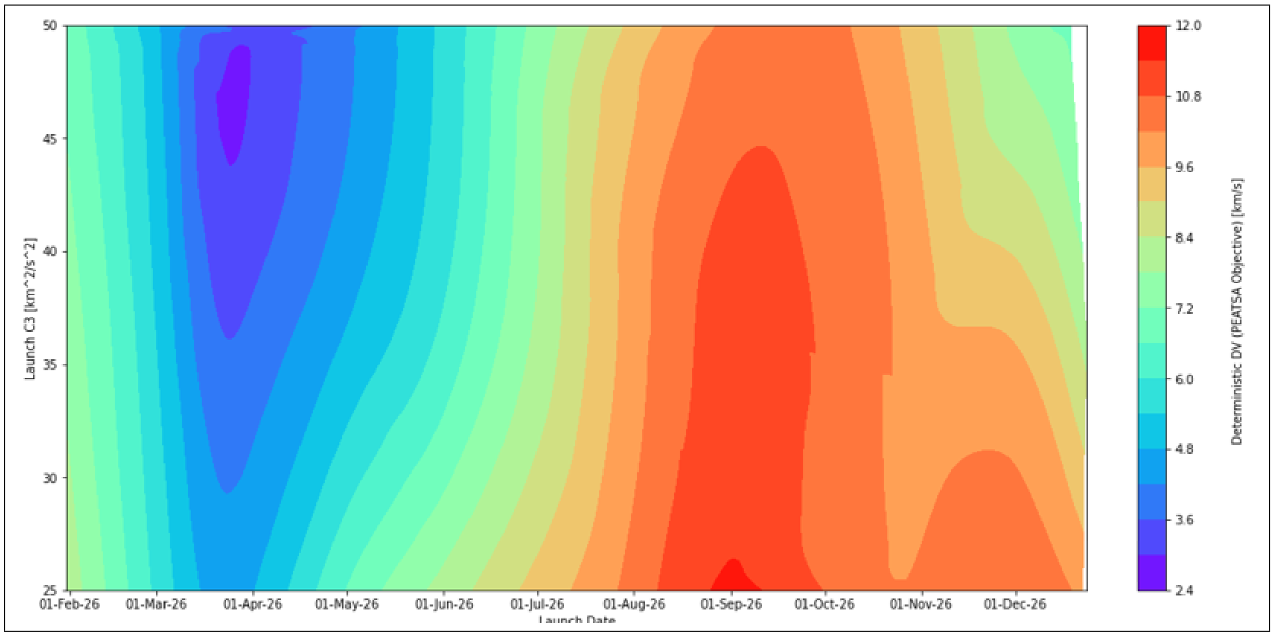


Figure 5: PEATSA Run Results.

4.1 Best Results

PEATSA includes another script, `<EMTG_root_dir>/PyEMTG/PEATSA/grab_best_peatsa_results.py`, which will fetch the feasible case with the best PEATSA objective for each fingerprint and place them in a user-defined folder. The command syntax for this script is:

```
python <EMTG_root_dir>/PyEMTG/PEATSA/grab_best_peatsa_results.py /path/to/PEATSA
_run/ /path/to/best_peatsa
```

Where `/path/to/PEATSA_run` is the directory from which to extract results and `/path/to/best_peatsa` is the destination for the best feasible cases. These results can be used to seed a new PEATSA case and reduce disk usage by filtering out "bad" cases, but note that iteration CSV files are not saved by this command, so be careful deleting old run directories. Note that the script does not create `/path/to/best_peatsa`; you must create it ahead of time.

This concludes the introductory tutorial on PEATSA! PEATSA contains capabilities beyond what was discussed. Refer to the PyEMTG User Guide, the comments in the PEATSA script created by the script generator, and the PyEMTG options files listed in Section 2.1 to learn more about PEATSA's capabilities.