HELMpy
User's Guide

Tulio Molina Juan Ortega

July 13, 2021

Contents

1 Introduction

HELMPY [1] is an open source package of power flow solvers developed on Python 3 [2]. This package contains the Holomorphic Embedding Load-flow Method (HELM) [3] and the Newton-Raphson (NR) algorithm. The intention of HELMPY is to support research, especially on the HELM, and to contribute with the development of open source code related to this subject. The developed code is properly commented and organized so it would be easy to understand and modify. This package was developed by Tulio Molina and Juan José Ortega as a part of their thesis research to obtain the degree of Electrical Engineer at Universidad de los Andes (ULA) in Mérida, Venezuela.

1.1 License and Terms of Use

The code is distributed under the GNU Affero General Public License v3.0 or later [4] (AGPL-3.0-or-later). The full text of the license can be found in the `LICENSE.txt` file.

2 Power Flow

The solution to the power flow problem consists in calculating the voltage at each bus of a power system under steady-state and balanced conditions, and this is carried out by solving a set of nonlinear equations that arises with the well known power balance equations. It is important to remark that the classic single slack bus model for the power flow problem uses a unique bus to account for system losses, on the orther hand, Distributed Slack Bus (DSB) models state that active power losses of the system can be distributed among system's generators in order to obtain a more realistic modeling of power systems. DSB models based on the allocation of participation factors to each generator are widely implemented on the literature, and several approaches have been formulated to calculate these participation factors based on different criteria. In [5], participation factors are calculated depending on each generator's active power output.

Power systems modeling and algorithms implemented in HELMPY for solving the power flow problem are explained below.

2.1 Modeling

The power flow solvers included in HELMPY implement the standard π branch model, which models transmission lines, phase-shifting transformers and off-nominal tap transformers. Shunt admittances for modeling capacitors or reactors banks in parallel to buses are also considered. Besides, reactive power limits at generators are as well taken into account, in exception for the slack's bus generator.

2.2 Algorithms

2.2.1 HELM

The HELM, firstly proposed by Antonio Trias [3], is a power flow method based on the study of complex analysis. The method is fully implemented in HELMPY. The theory, fundamentals and practical implementations used as guide to implement this method are described and presented in [6], [7] and [8]. The method is developed considering two different models for PV buses, as the ones described in [6].

HELM, seen solely as a numeric method, consists in recursively solving a linear matrix equation of the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{B}$ and, after each time this system is solved, performing analytic continuation on a group of power series that represent voltages at buses; this process continues until convergence is reached or divergence is detected. The matrix $\boldsymbol{A}$ remains constant, so it should be factorized. The vector $\boldsymbol{B}$ is computed before each time the system is to be solved. As a result, vector $\boldsymbol{x}$ contains a new coefficient of each voltage power series.

Figure 1 on the following page contains the flowchart that exposes the procedure of the method (solely as a numeric method) and shows the respective and relevant functions and code lines that execute those steps.

2.2.2 Newton-Raphson

The extensible used Newton-Raphson algorithm to solve the power flow problem is implemented using complex numbers in rectangular form for calculations and unknown variables are represented in polar form.

Figure 2 on page 5 contains the flowchart that exposes the procedure of the method (solely as a numeric method) and shows the respective and relevant functions and code lines that execute those steps.

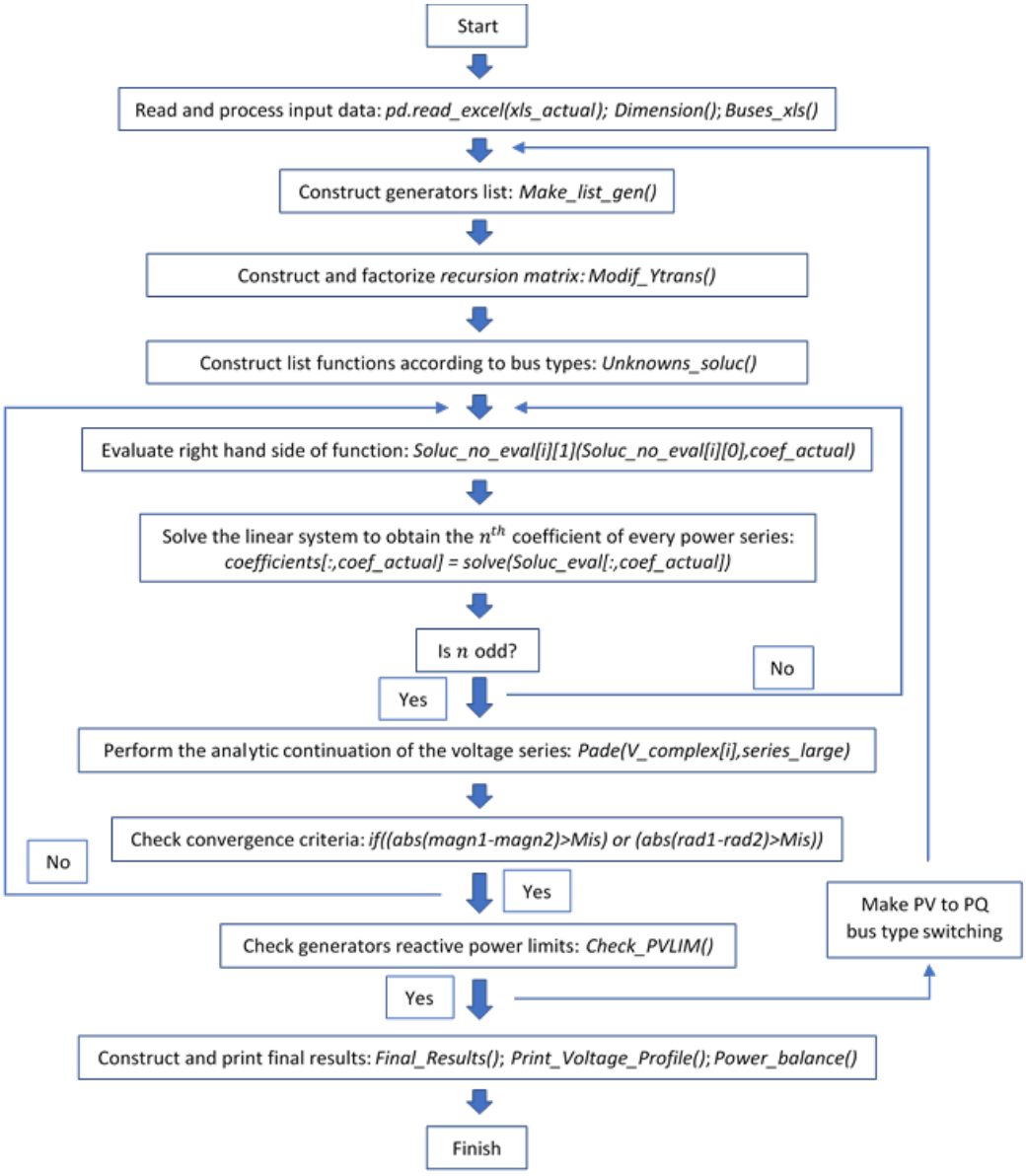

Figure 1: HELM flow chart

2.2.3 Distributed Slack Bus Model

A DSB model for the power flow formulation based on participation factors described in [5] is developed in HELMPY for both HELM and NR methods.

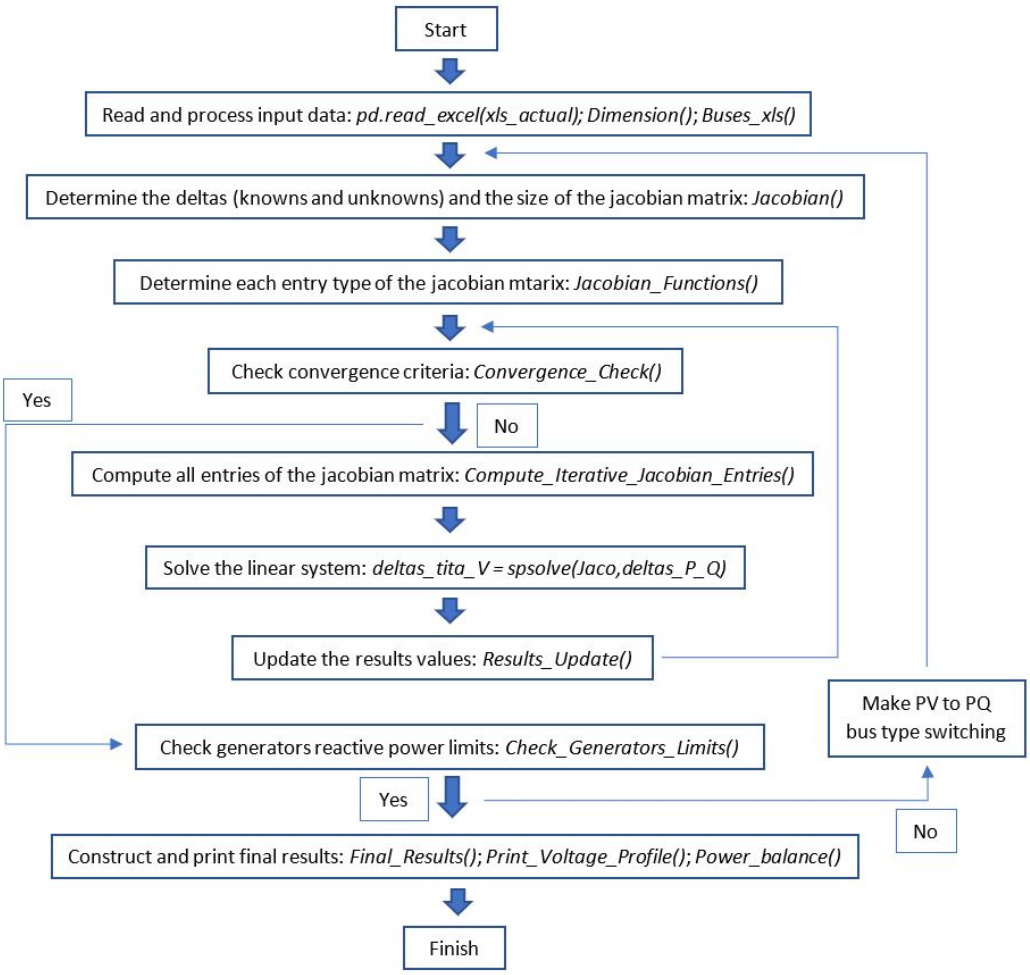

Figure 2: Newton-Raphson flow chart

The DSB model for NR is obtained from [9] and implemented to balanced systems. Furthermore, considering PV bus models defined in [6], four implementations of such DSB model for HELM are developed and denominated as: M1 PV1, M1 PV2, M2 PV1 and M2 PV2, according to the selected model for describing the slack and PV buses. This combinations depends on the chosen arguments when executing the helm function.

2.2.4 Convergence criteria

Considering a maximum mismatch error, the managed convergence criteria for the NR implementation is based on checking active and reactive power mismatch error at each bus, according to the bus type. On the other hand, the managed convergence criteria for implementations based on the HELM consists in checking whether the difference between the magnitudes and phase angles of two consecutive voltages (obtained through analytic continuation of power series) at each bus is less than the mismatch error.

3 How to use HELMPY

3.1 Software requirements

HELMPY is written in Python version 3.6.0, therefore, this or newer versions of the aforementioned programming language along with the following libraries must be installed in order to use this power flow solver package:

- `Numpy` [10]
- `Scipy` [11]
- `Pandas` [12]

No installation process other than copying the files of HELMPY folder to the directory where HELMPY is going to be used is needed.

3.2 Input case

For purposes of portability and simplicity HELMPY receives as input case (grid or system) a `.xlsx` file which contains all the needed data to solve a power flow problem.

3.2.1 Obtaining a case from MATPOWER

The format of the input `.xlsx` file has been designed from the format of MATPOWER [13] cases. Therefore, the script `Obtain_case_MATPOWER.m`, which is a template for obtaining the HELMPY input data of the desired grid stored in MATPOWER, is included in the HELMPY folder. By means of this `.m`

script, case9, case118, case1354pegase and case2869pegase stored in MATPOWER have been processed and their respective `.xlsx` files (of the same name) are also included in the data/cases folder of the HELMPY repository.

3.2.2 Creating a case:

The `.xlsx` file for the input case can be created by the user. Such file must contain three sheets, without header, named as follows: *Buses*, *Generator* and *Branches*. According to the sheet, the data that must be stored in each column is presented below:

- *Buses*, each row stands for a bus of the grid:
  - Column A: Bus number (positive integer).
  - Column B: Type of buses. 1 for PQ, 2 for PV, 3 for slack.
  - Column C: Active power demand (MW).
  - Column D: Reactive power demand (MVAr).
  - Column E: Shunt conductance (MW demanded at V = 1.0 p.u.).
  - Column F: Shunt susceptance (MVAr injected at V = 1.0 p.u.).
- *Generators*, each row stands for a generator of the grid:
  - Column A: Bus number (positive integer).
  - Column B: Active power output (MW).
  - Column D: Maximum reactive power output (MVAr).
  - Column E: Minimum reactive power output (MVAr).
  - Column F: Voltage magnitude set point (p.u.).
- *Branches*, each row stands for a branch of the grid:
  - Column A: "From" bus number.
  - Column B: "To" bus number.
  - Column C: Resistance (p.u.).
  - Column D: Reactance (p.u.).
  - Column E: Total line charging susceptance (p.u.).

- Column I: Transformer off nominal turns ratio (taps at “from" bus, impedance at “to" bus).
- Column J: Transformer phase shift angle (degrees), positive for delay.

The remaining columns are not used. The input file must be in the directory where HELMPY is going to be used.

3.3 Functions

As it has been said, HELMPY is able to solve the power flow problem by means of the NR method or by the HELM considering whether PV bus model I or II is used, and also applying the DSB models. These methods are developed to be used as the following functions:

- `nr(GridName, Print_Details=False, Mismatch=1e-4, Results_FileName="", Scale=1, MaxIterations=15, Enforce_Qlimits=True)`
- `nr_ds(GridName, Print_Details=False, Mismatch=1e-4, Results_FileName="", Scale=1, MaxIterations=15, Enforce_Qlimits=True, DSB_model=True)`
- `helm(case, detailed_run_print=False, mismatch=1e-4, scale=1, max_coefficients=100, enforce_Q_limits=True, results_file_name=None, save_results=False, pv_bus_-model=2, DSB_model=False, DSB_model_method=None)`

Where the parameters of the NR functions are explained below:

- `GridName`: String of characters type. This parameter receives the name of the input grid. The `.xlsx` extension must be included in the string of characters.
- `Print_Details`: Boolean type. If `True`, the function prints details of the running power flow problem and the results of it on the Python console, if `False`, it only prints whether convergence was reached or not. This parameter is set as `False` by default.
- `Mismatch`: Float type. It defines the value of the maximum mismatch error. This parameter is set at `1e-4` by default.

- `Results_FileName`: String of characters type. The received string of characters is used for naming the `.txt` and `.xlsx` output files along with other information (used function, `Scale` and `Mismatch`). The name will be the same as the input case file if this parameter is an empty string of characters. It is set as an empty string of characters by default.

- `Scale`: Float or integer type. It receives the scaling factor which multiplies active and reactive power load and active power generation of the grid (loadability scaling factor). This parameter is set at 1 by default.

- `MaxIterations`: Integer type. It defines the maximum number of iterations allowed for reaching convergence. The number of iterations is restarted if convergence is reached but reactive power limits are violated. It is set at 15 by default.

- `Enforce_Qlimits`: Boolean type. If `True`, reactive power generation limits are checked at each PV bus after convergence is reached, if `False`, limits are not checked. It is set as `True` by default.

And the parameters of the helm function are explained below:

- `case`: Case data object, contains the grid data.

- `detailed_run_print`: Boolean type. If `True`, the function prints details of the running power flow problem and the results of it on the Python console, if `False`, it only prints whether convergence was reached or not. This parameter is set as `False` by default.

- `mismatch`: Float type. It defines the value of the maximum mismatch error. This parameter is set at `1e-4` by default.

- `scale`: Float or integer type. It receives the scaling factor which multiplies active and reactive power load and active power generation of the grid (loadability scaling factor). This parameter is set at 1 by default.

- `max_coefficients`: Integer type. It defines the maximum number of coefficients to be computed by the algorithm for reaching convergence. The number of coefficients is restarted if convergence is reached but reactive power limits are violated. It is set at 100 by default.

- `enforce_Q_limits`: Boolean type. If `True`, reactive power generation limits are checked at each PV bus after convergence is reached, if `False`, limits are not checked. It is set as `True` by default.

- `results_file_name`: String of characters type. The received string of characters is used for naming the `.txt` and `.xlsx` output files along with other information (used function, `Scale` and `Mismatch`). The name will be the same as the input case file if this parameter is an empty string of characters. It is set as an empty string of characters by default.

- `save_results`: Boolean type. If `True`, a results file is generated and save, if `False`, ir is not. It is set as `False` by default.

- `pv_bus_model`: Integer type. Options available {1,2}. If `1`, the pv bus model 1 is used. If `2`, the pv bus model 2 is used. It is set as `2` by default.

- `DSB_model`: Boolean type. If `True`, the DSB is used, if `False`, the slack bus is set to account for all losses representing in this way the classic bus model. It is set as `True` by default.

- `DSB_model_method`: Integer type. Options available {1,2}. If `1`, the DSB model 1 is used. If `2`, the DSB model 2 is used. It is set as `None` by default. If the DSB_model is set to `True`, DSB_model_method is set to 2.

Each one of these functions returns the resultant voltage profile given in complex rectangular form.

3.4 Output files

In order to show and save the output of each program execution, two files are generated: a `.xlsx` file and a `.txt` file.

The resultant voltage profile, in complex and polar form, is presented in the *Buses* sheet of the resultant `.xlsx` file. On the other hand, the power injection of buses through each branch of the grid is shown in the *Branches* sheet of such file.

Information such as loadability scale factor, convergence time, mismatch error, number of coefficients or number of iterations (according to the used

function), and also a power balance of the problem are shown in the output `.txt` file.

3.5 Running HELMpy

Each one of the aforementioned functions can be imported to the user's Python script by declaring the command:

```
import helmpy
```

The `helmpy.py` script imports the `nr`, `nr_ds` and `helm` functions from the respective `.py` script where each one of them is defined. Therefore, for running HELMpy with a pre-defined input case named `Case.xlsx`, by means of these three functions, a typical Python script would look as follows:

```
import helmpy

voltage_profile_nr = helmpy.nr("Case.xlsx")
grid_data_file_path = '/path/of/case.xlsx'    voltage_-
profile_helm = helmpy.helm(case.case)
```

Where the resulting voltage profile of each function is returned to its corresponding assigned variable.

3.6 Comparing results

The following `.py` scripts found on the test folder can be used to compare voltage profile results obtained from HELMpy and MATPOWER:

- `Compare_Voltages.py`: It compares two voltage profiles obtained from HELMpy. The absolute value of the highest magnitude and phase angle differences between both voltage profiles are printed on the Python console. If the `print_all` variable is set as `True`, the absolute value of the magnitude and phase angle differences at each bus are also going to be printed.

- `Compare_Several_Voltages.py`: It compares one voltage profile (base profile) with a list of voltage profiles obtained from HELMpy. The absolute value of the highest magnitude and phase angle differences

between the base profile and the rest of profiles are printed on the Python console, respectively. If the `print_all` variable is set as `True`, the absolute value of the magnitude and phase angle differences at each bus are also going to be printed.

- `Compare_with_MATPOWER.py`: It compares two voltage profiles, one obtained from HELMPY and the other obtained from MATPOWER. The absolute value of the highest magnitude and phase angle differences between both voltage profiles are printed on the Python console. If the `print_all` variable is set as `True`, the absolute value of the magnitude and phase angle differences at each bus are also going to be printed.

There are `.xlsx` files of already computed results results in the data/results folder of the repository for reference and comparison.

4 Miscellaneous

In order to validate the results obtained from HELMPY, a comparison between these results and the ones obtained from MATPOWER can be made, for such, the `Execute_MATPOWER.m` script is included in the HELMPY folder. This script allows to run a power flow problem with MATPOWER by means of the NR method, where loadability scale factor of the case, tolerance and initial seed of voltages can be modified. The output of `Execute_MATPOWER.m` is a `.xlsx` file containing the resultant voltage profile, which can be used by the `Compare_with_MATPOWER.py` script.

The mentioned `.m` script has been used to execute case9, case118, case1354pegase and case2869pegase using a tolerance of `1e-8`. Their respective `.xlsx` output files are named "`Results MATPOWER case9.xlsx`", "`Results MATPOWER case118.xlsx`", "`Results MATPOWER case1354pegase.xlsx`" and "`Results MATPOWER case2869pegase.xlsx`". These files are included in the `HELMpy` folder.

References

[1] HELMpy, open source package of power flow solvers, including the Holomorphic Embedding Load Flow Method (HELM), developed on Python 3. [Online]. Available: https://github.com/HELMpy/HELMpy

[2] Python programming language. [Online]. Available: https://www.python.org/

[3] A. Trias, "The holomorphic embedding load flow method," in *2012 IEEE Power and Energy Society General Meeting*, July 2012, pp. 1–8.

[4] GNU Affero General Public License v3.0. [Online]. Available: https://opensource.org/licenses/AGPL-3.0

[5] J. Meisel, "System incremental cost calculations using the participation factor load-flow formulation," *IEEE transactions on power systems*, vol. 8, no. 1, pp. 357–363, 1993.

[6] M. K. Subramanian, "Application of holomorphic embedding to the power-flow problem," Master's thesis, Arizona State University, 2014.

[7] S. Rao, Y. Feng, D. J. Tylavsky, and M. K. Subramanian, "The holomorphic embedding method applied to the power-flow problem," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3816–3828, 2016.

[8] A. Trias, "Fundamentals of the holomorphic embedding load-flow method," 2015. [Online]. Available: http://arxiv.org/abs/1509.02421

[9] S. Tong and K. N. Miu, "A network-based distributed slack bus model for dgs in unbalanced power flow studies," *IEEE Transactions on Power Systems*, vol. 20, no. 2, pp. 835–842, 2005.

[10] NumPy, fundamental package for scientific computing with Python. [Online]. Available: http://www.numpy.org/

[11] SciPy, Python-based ecosystem of open-source software. [Online]. Available: https://www.scipy.org/

[12] Pandas, Python Data Analysis Library. [Online]. Available: https://pandas.pydata.org/

[13] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb 2011.