Pyomo-based Three Phase Distribution Network Optimal Power Flow*

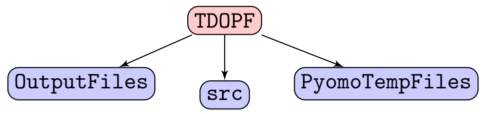
Swastik Sharma†, Swathi Battula‡

1 Introduction

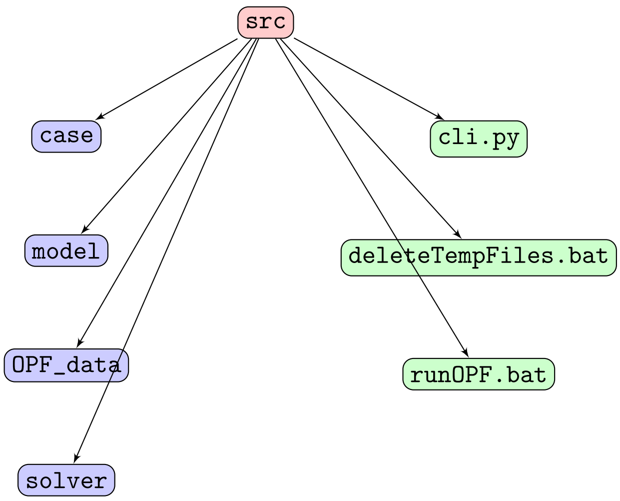
There are very few repositories or open-source programs in Python that can help researchers run OPF on Unbalanced Distribution Networks; this program bridges the gap. It provides a powerful open-source program for that purpose. We use the modelling techniques presented in [citation] to model the OPF. This document will help you understand how to run the Pyomo-based T-DOPF application and provide an in-depth overview of the repository's organisation.

2 Overview of the Repository

The root directory, `TDOPF`, and its containing folders are shown in the following diagram:



1. `src`: This is the brain of the whole application. It contains all the `.py` files and `.bat` files necessary for running the Python program.



   (a) `case`: This folder contains readers to read a matpower based `.m` files. You can safely ignore this if you wish to use `.csv` files directly to input data. The use of `.m` files have not been explored much in this program, but interested people can use these helper files to integrate them.

   (b) `model`: This folder contains all the `.py` files for running the pyomo-based optimisation framework.

   (c) `OPF_data`: This folder contains all the `.csv` or `.xslx` files necessary to run the OPF. The folder contains all the files for an IEEE 123 bus system. Using the same format, one can incorporate any distribution system for analysis.

*Latest Revision: 10^{th} January, 2025

†swastiks21@iitk.ac.in, Dept. of Electrical Engineering, Indian Institute of Technology Kanpur 208016
‡swathi@iitk.ac.in, Dept. of Electrical Engineering, Indian Institute of Technology Kanpur 208016

(d) `solver`: This folder has the `.py` programs to initialise the solver related libraries. If required, the `results.py` program is used to get output and dual variables from the pyomo-optimizer.

(e) `cli.py`: This Python file runs the entire program. It initialises the DATA and OPF model and stores the results in `.txt` format in the output folder.

(f) `deleteTempFiles.bat`: This batch file deletes the pyomo's temporary files in the PyomoTempFiles folder.

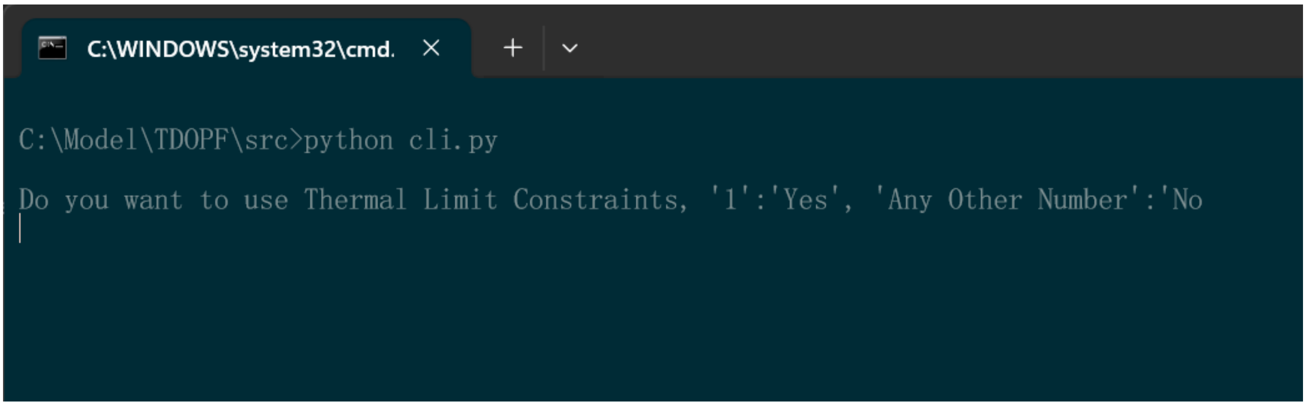
(g) `runOPF.bat`: Although one can run the python program directly through `cmd` using `python cli.py`, this batch file eases the operation as one can directly run this file to run the T-DOPF.

2. `OutputFiles`: This is used to store the results from OPF, such as the voltage magnitude at each node, line power flows, etc.

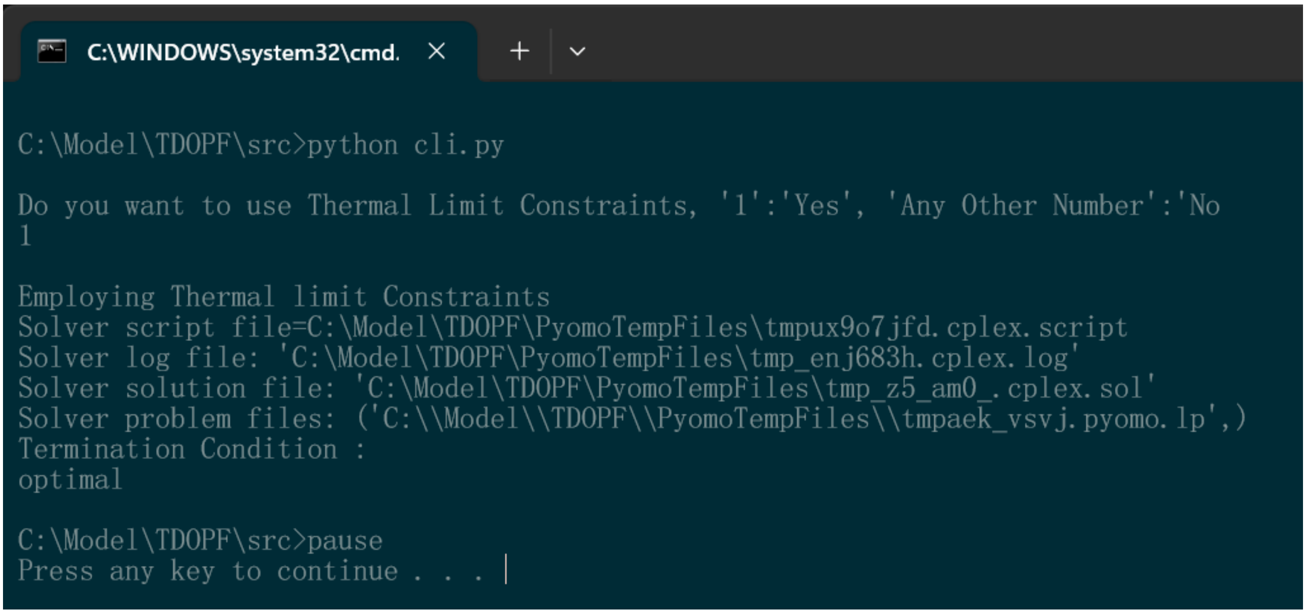
3. `PyomoTempFiles`: This contains the temporary files generated by pyomo.

3 Running the Program:

As said earlier, one can start the program directly by running the `runOPF.bat` file. The user is greeted with the following screen: The thermal line limit constraints have already been added, following the



modelling as laid down in []. The users can use the instructions if they wish to or don't wish to employ the constraints. The thermal limits are calculated using the ampacity of each line configuration, which is already provided in the standard distribution system documentation. If the results are optimal, the user will see the following on the screen:



The results file in `OutputFiles` folder displays all the results of the variables used in the OPF formulation:

Results_2025-01-10-14-23-28.dat

```
THE OPF WAS RUN AT : 2025:01:10 14:23:34 IST +0530

SOLUTION_STATUS
optimal
END_SOLUTION_STATUS

DGResultsforActivePower

DER1
Interval: 1
    ActivePowerGenerated: 0.0 kW at Phase A
    ActivePowerGenerated: 0.0 kW at Phase B
    ActivePowerGenerated: 0.0 kW at Phase C

END_DGResultsforActivePower

DGResultsforReactivePower

DER1
Interval: 1
    ReactivePowerGenerated: 0.0 kVaR at Phase A
    ReactivePowerGenerated: 0.0 kVaR at Phase B
    ReactivePowerGenerated: 0.0 kVaR at Phase C

END_DGResultsforReactivePower

VOLTAGE MAGNITUDES

Phase: A Bus: Bus0 Interval: 1 : 1.03 p.u.
Phase: B Bus: Bus0 Interval: 1 : 1.03 p.u.
Phase: C Bus: Bus0 Interval: 1 : 1.03 p.u.
Phase: A Bus: Bus1 Interval: 1 : 1.02399 p.u.
Phase: B Bus: Bus1 Interval: 1 : 1.02789 p.u.
Phase: C Bus: Bus1 Interval: 1 : 1.02522 p.u.
Phase: A Bus: Bus2 Interval: 1 : NaN
Phase: B Bus: Bus2 Interval: 1 : 1.02777 p.u.
Phase: C Bus: Bus2 Interval: 1 : NaN
```

4 Modifying Input Files and Other Functionalities

All input-related files are in the `OPF_data` folder. You can modify these input files directly to enhance functionalities without altering scripts or code.

1. **Changing load on the system**: If you want to add more load over and above the Spot Loads of the distribution system, there is a Python file to do that, `load_init.py`. You can add more load onto the system, and the corresponding `.csv` files are created in the `load` folder.

2. **Adding DERs:** To add DERs into the system, you can use the `der.csv` file. Since DERs can work both ways, inject or withdraw power, the same file can be used with injections being positive and withdrawal being negative.

3. **Capacitors:** To manage the voltage magnitudes at different buses, the OPF could also be designed to use capacitors. The `CapData.csv` can also specify the location, maximum reactive power per phase, switching amount, and cost.

4. **Changing Base Values or Header Bus:** The operator substation's base values and location are defined in the `Substation.csv` file. Users can modify this file to change the base values according to their system specifications.

5. **Constraints and Objective Functions:** `contraints.py` file in `model` folder contains all the constraints and objective functions required to run the OPF. Users can modify the constraints or objective function here to suit their problem statements.