

# ACDC-OpFlow Manual V0.1.1

Haixiao Li      Azadeh Kermansaravi      Aleksandra Lekić

May 27, 2025

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Configuration . . . . .	2
1.2.1	Dependencies . . . . .	2
1.2.2	Installation steps . . . . .	3
1.2.3	Code structure . . . . .	4
<b>2</b>	<b>OPF Formulation</b>	<b>5</b>
2.1	Nomenclature . . . . .	5
2.2	Original nonlinear formulation . . . . .	7
2.3	Convex relaxed formulation . . . . .	10
<b>3</b>	<b>Code Implementation</b>	<b>12</b>
3.1	Data preparation . . . . .	12
3.2	Parameter processing . . . . .	17
3.3	Variable definition . . . . .	18
3.4	Constraint construction . . . . .	21
<b>4</b>	<b>Examples</b>	<b>23</b>
4.1	Running a case . . . . .	23
4.2	Benchmark . . . . .	26
4.2.1	Performance comparison . . . . .	26
4.2.2	Discussion . . . . .	27

## 1 Overview

### 1.1 Motivation

AC/VSC-MTDC (Voltage Source Converter-based Multi-terminal DC) grids play a crucial role in modern power systems by enhancing grid stability, flexibility, and grid-friendly interconnection.

VSC-MTDC allows independent active and reactive power control, enabling seamless connection of AC grids in different regions. The scheduling of a power system consisting of multiple interconnected AC grids and a VSC-MTDC grid is significant [1], [2], [3]. As depicted in Fig. 1.1, we need a technique to help enhance the rationality and optimality of scheduling, which involves solving the problems known as the AC/DC optimal power flow (OPF).

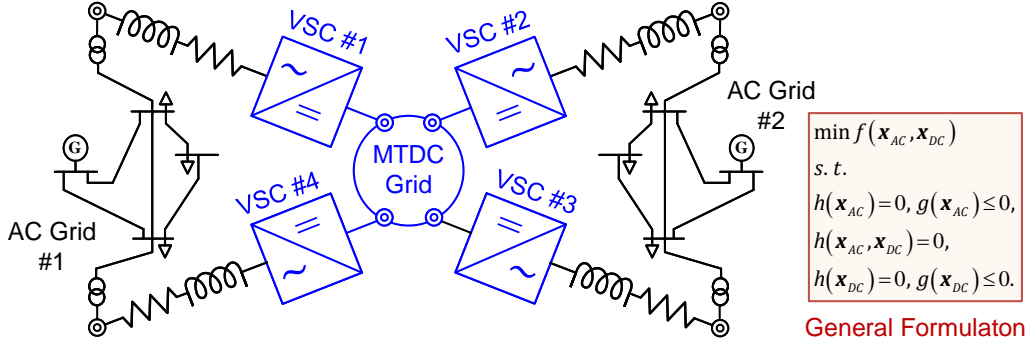


Figure 1.1: Overview of the AC/DC OPF problem

This section provides a concise guide on environment configuration, core code structure, running the code, and accessing results. It serves as a basic introduction to help users to get started with **ACDC-OpFlow**.

## 1.2 Configuration

### 1.2.1 Dependencies

**ACDC-OpFlow** is developed using languages: [Matlab](#), [Julia](#), [Python](#), [C++](#). For each programming environment, we provide recommendations on the dependency versions used in our implementation. While users may install later versions of these dependencies, it is advised to verify compatibility.

- [Matlab r2021b](#) : `Yalmip 20230622`.
- [Julia v1.10](#) : `CSV 0.10.15` `DataFrames 1.7.0` `JuMP 1.23.3` `Gurobi 1.0.0` `Graphs 1.12.1` `GraphPlot 0.6.1` `GLMakie 0.11.4` `ColorSchemes 3.29.0` `Colors 0.12.11`.
- [Python v3.12](#) : `numpy 2.2.1` `pandas 2.2.3` `scipy 1.15.1` `pyomo 6.8.2` `networkx 3.4.2` `matplotlib 3.10.0` `gurobipy 11.0.1`.
- [C++](#) : `Gurobi C++ API` `Eigen3 3.4.0` `matplotlibplusplus 3.4.0`.
- [Gurobi](#) : `Gurobi 9.5.2`.
- [Matpower](#) : `Matpower 7.1`.

### 1.2.2 Installation steps

To install required toolboxes in [Matlab](#) environment, users can download the toolbox directly from the official website. Then, users can add the toolbox folders to Matlab path, using the following commands:

```
>> addpath(genpath('path/to/YALMIP'))
>> ...
>> savepath
```

To install required packages in [Julia](#) environment, users can use the following commands:

```
>> using Pkg
>> Pkg.add("JuMP")
>> Pkg.add("CSV")
>> ...
```

Alternatively, users may use the package mode (triggered by pressing the `]` key in the Julia REPL):

```
>> (@v1.10) pkg> add JuMP
>> (@v1.10) pkg> add CSV
>> ...
```

To install required packages in [Python](#) environment, users can use the following commands if with Anaconda<sup>1</sup>, installing packages into isolated environments:

```
>> conda create -n acdcOpflow-env python=3.12
>> conda activate acdcOpflow-env
>> conda install numpy pandas matplotlib ...
```

Also, users can install packages either globally or within virtual environments:

```
>> pip install numpy pandas matplotlib ...
```

To install required libraries in [C++](#), we recommend using `vcpkg`<sup>2</sup> to simplify the process of acquiring and managing [C++](#) dependencies. The commands are:

```
>> ./vcpkg install eigen3 boost
>> ./vcpkg install matplotlibplusplus
>> ...
```

---

<sup>1</sup><https://www.anaconda.com/>

<sup>2</sup><https://vcpkg.io/en/>

### 1.2.3 Code structure

As presented in Fig. 1.2, under the framework of **ACDC-OpFlow**, every coding language follows a unified workflow: First, we prepare input data related to AC and DC grids, which are all saved in structured comma-separated value (CSV) files. Secondly, extracting key parameter information from .csv data files and handling them to form the specific parameter vectors (matrices). Third, based on predefined parameter vectors (matrices), the AC/DC OPF model is constructed and subsequently solved using the Gurobi solver. Lastly, the optimal power flow results are organized and output in text format, accompanied by a visual presentation to facilitate the observation of the results. For each coding language, the included main functions are listed as Fig. 1.3.

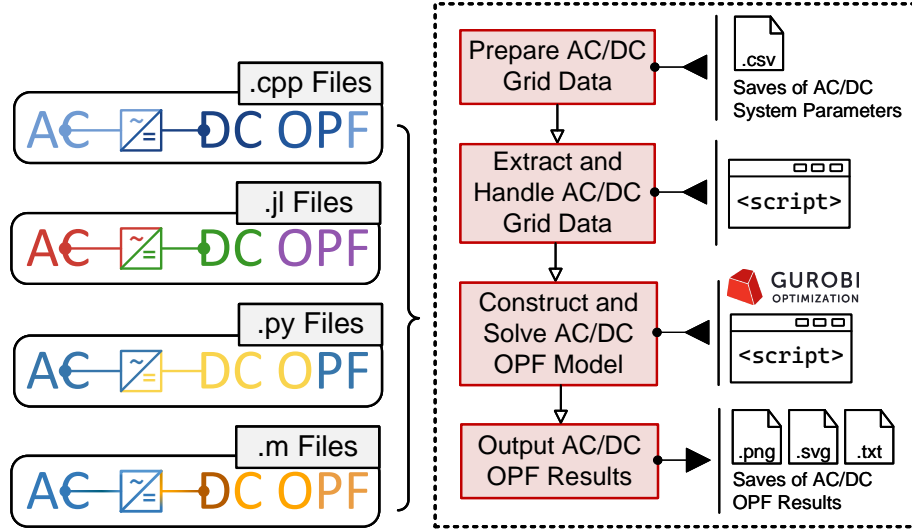


Figure 1.2: Unified workflow in **ACDC-OpFlow**

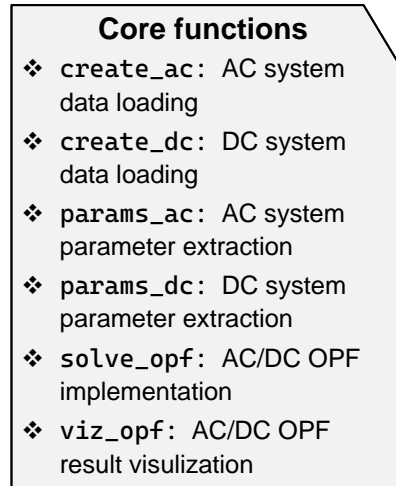


Figure 1.3: Main functions in **ACDC-OpFlow**

## 2 OPF Formulation

### 2.1 Nomenclature

This section introduces the mathematical formulation of the AC/DC OPF used in **ACDC-OpFlow**. The fundamental constraint and objective functions to describe the optimal steady-state behavior for the AC/DC power system are introduced. Symbols appearing in subsequent formulas can be found below.

#### Indexes and Sets

$(\bar{\cdot})/(\underline{\cdot})$  -The upper/lower bound of the corresponding optimization variables and parameters

$|\cdot|$  -Magnitude

$\mathcal{L}$  -Branch set

$\mathcal{N}$  -Node set

$AC$  -Alternating current

$cost$  -Generation cost

$droop$  -Droop parameters

$gen$  -Generator

$h, i, j$  -System node

$ij$  -System branch

$load$  -Load

$loss$  -Power loss

$MTDC$  -Multi-terminal direct current

$ref$  -Reference value

$VSC$  -Voltage source converter

#### Variables and Parameters Related to the AC Grid

$\theta_{ij}^{AC}$  -Phase difference for AC branch  $ij$

$c_{i,2}^{AC}/c_{i,1}^{AC}/c_{i,0}^{AC}$  -Quadratic/Linear/Constant generation coefficient at AC bus  $i$

$c_{ij}^{AC}/s_{ij}^{AC}$  -Optimization variables linked with the AC nodal voltage

$G_{ij}^{AC}/B_{ij}^{AC}$  -Real/Imaginary part of AC bus admittance matrix

$g_{ij}^{AC}/b_{ij}^{AC}$  -Conductance/Susceptance of AC branch  $ij$

$p_{ij}^{AC}/q_{ij}^{AC}$  -Real/Reactive power for AC branch  $ij$

$p_{i,A2V}^{AC}/q_{i,A2V}^{AC}$  -Real/Reactive power delivered from the AC grid to the VSC station at AC bus  $i$

$p_{i,gen}^{AC}/q_{i,gen}^{AC}$  -Real/Reactive power outputs of generator at AC bus  $i$

$p_{i,load}^{AC}/q_{i,load}^{AC}$  -Real/Reactive power loads at AC bus  $i$

$p_i^{AC}/q_i^{AC}$  -Real/Reactive power injection at AC bus  $i$

$v_i^{AC}$  -AC bus  $i$  voltage

### Variables and Parameters Related to the MTDC Grid

$k_{i,droop}^{MTDC}$  -Droop slope at MTDC bus  $i$

$l_{jh}^{MTDC}$  -Squared current of MTDC branch  $jh$

$p_{i,ref}^{MTDC}$  -Power reference of DC power control/DC droop control at MTDC bus  $i$

$p_{jh}^{MTDC}$  -Power of MTDC branch  $jh$

$u_j^{MTDC}$  -Squared MTDC bus  $j$  voltage

$v_j^{MTDC}$  -MTDC bus  $j$  voltage

$V_{i,ref}^{MTDC}$  -Voltage reference of DC voltage control/DC droop control at MTDC bus  $i$

### Variables and Parameters Related to the VSC Station

$\theta_s^{VSC}/\theta_f^{VSC}/\theta_c^{VSC}$  -VSC bus  $s/f/c$  phase

$a_{loss}^{VSC}/b_{loss}^{VSC}/c_{loss}^{VSC}$  -Constant/Linear/Quadratic coefficient of VSC cpower loss function

$b_f^{VSC}$  -Susceptance for VSC bus  $f$  to ground

$b_{sf}^{VSC}/b_{fc}^{VSC}$  -Susceptance for VSC branch  $sf/fc$

$c_{sf}^{VSC}/s_{sf}^{VSC}/c_{cf}^{VSC}/s_{cf}^{VSC}$  -Optimization variables linked with the VSC nodal voltage

$g_{sf}^{VSC}/g_{fc}^{VSC}$  -Conductance for VSC branch  $sf/fc$

$I_c^{VSC}$  -Phase current at VSC bus  $c$

$l_c^{VSC}$  -Squared phase current at VSC bus  $c$

$p_{loss}^{VSC}$  -VSC power loss

$q_{s,ref}^{VSC}$  -Power reference of reactive power control at VSC bus  $s$

$v_s^{VSC}/v_f^{VSC}/v_c^{VSC}$  -VSC bus  $s/f/c$  voltage

$v_{s,ref}^{VSC}$  -Voltage reference of AC voltage control at VSC bus  $s$

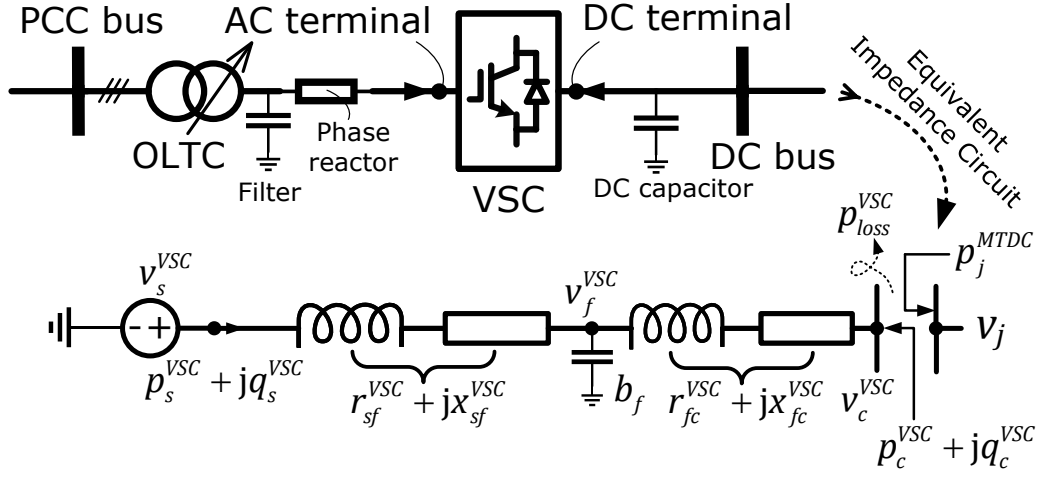


Figure 2.1: Equivalent impedance model of VSC

## 2.2 Original nonlinear formulation

The original AC/DC OPF formulation contains is formulated as a nonlinear programming (NLP) problem due to the presence of several nonlinear constraints, such as nonlinear power flow constraints and quadratic converter loss constraints. Fig. 2.1 is provided to help understanding power flow formulation at the AC/DC coupling points. The explicit formulation of the original AC/DC OPF model is presented as below.

---

Original AC/MTDC OPF Formulation

[NLP]

---

### Variables:

$$\begin{aligned}
 & p_i^{AC}, q_i^{AC}, p_{i,gen}^{AC}, q_{i,gen}^{AC}, p_{i,A2V}^{AC}, q_{i,A2V}^{AC}, \theta_i^{AC}, v_i^{AC} \\
 & p_{ij}^{AC}, q_{ij}^{AC} \\
 & p_j^{MTDC}, v_j^{MTDC} \\
 & p_{jh}^{MTDC} \\
 & p_s^{VSC}, q_s^{VSC}, \theta_s^{VSC}, v_s^{VSC} \\
 & p_c^{VSC}, q_c^{VSC}, \theta_c^{VSC}, v_c^{VSC}, I_c^{VSC} \\
 & q_f^{VSC}, \theta_f^{VSC}, v_f^{VSC}
 \end{aligned}$$

$$\begin{aligned}
 & i \in \mathcal{N}^{AC} \\
 & (i, j) \in \mathcal{L}^{AC} \\
 & j \in \mathcal{N}^{MTDC} \\
 & (j, h) \in \mathcal{L}^{MTDC} \\
 & s \in \mathcal{N}^{VSC} \\
 & c \in \mathcal{N}^{VSC} \\
 & f \in \mathcal{N}^{VSC}
 \end{aligned}$$

**AC constraints:**

$$p_{ij}^{AC} = g_{ij}^{AC} |v_i^{AC}|^2 - |v_i^{AC}| |v_j^{AC}| (g_{ij}^{AC} \cos \theta_{ij}^{AC} + b_{ij}^{AC} \sin \theta_{ij}^{AC}), \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.1a)$$

$$q_{ij}^{AC} = -b_{ij}^{AC} |v_i^{AC}|^2 + |v_i^{AC}| |v_j^{AC}| (b_{ij}^{AC} \cos \theta_{ij}^{AC} - g_{ij}^{AC} \sin \theta_{ij}^{AC}), \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.1b)$$

$$p_i^{AC} = \sum_{(i,j)} p_{ij}^{AC} + \left( \sum_j G_{ij}^{AC} \right) (v_i^{AC})^2, \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.1c)$$

$$q_i^{AC} = \sum_{(i,j)} q_{ij}^{AC} + \left( \sum_j -B_{ij}^{AC} \right) (v_i^{AC})^2, \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.1d)$$

$$p_i^{AC} = p_{i,gen}^{AC} - p_{i,load}^{AC} - p_{i,A2V}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.2a)$$

$$q_i^{AC} = q_{i,gen}^{AC} - q_{i,load}^{AC} - q_{i,A2V}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.2b)$$

$$\underline{p}_{i,gen}^{AC} \leq p_{i,gen}^{AC} \leq \overline{p}_{i,gen}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.3a)$$

$$\underline{q}_{i,gen}^{AC} \leq q_{i,gen}^{AC} \leq \overline{q}_{i,gen}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.3b)$$

$$\underline{|v_i^{AC}|} \leq |v_i^{AC}| \leq \overline{|v_i^{AC}|}, \quad i \in \mathcal{N}^{AC} \quad (2.4)$$

**MTDC constraints:**

$$p_{jh}^{MTDC} = \rho^{MTDC} \cdot v_j^{MTDC} (v_j^{MTDC} - v_h^{MTDC}) y_{jh}^{MTDC}, \quad j, h \in \mathcal{N}^{MTDC}, \quad (j, h) \in \mathcal{L}^{MTDC} \quad (2.5a)$$

$$p_j^{MTDC} = \sum_{(j,h)} p_{jh}^{MTDC}, \quad j, h \in \mathcal{N}^{MTDC}, \quad (j, h) \in \mathcal{L}^{MTDC} \quad (2.5b)$$

$$\underline{v}_j^{MTDC} \leq v_j^{MTDC} \leq \overline{v}_j^{MTDC}, \quad j \in \mathcal{N}^{MTDC} \quad (2.6)$$

$$p_j^{MTDC} = p_{j,ref}^{MTDC}, \quad j \in \mathcal{N}^{MTDC} \quad (2.7a)$$

$$v_j^{MTDC} = v_{j,ref}^{MTDC}, \quad j \in \mathcal{N}^{MTDC} \quad (2.7b)$$

$$p_j^{MTDC} = p_{j,ref}^{MTDC} - \frac{1}{k_{j,droop}^{MTDC}} (v_j^{MTDC} - v_{j,ref}^{MTDC}), \quad j \in \mathcal{N}^{MTDC} \quad (2.7c)$$



**VSC constraints:**

$$p_s^{VSC} = p_{i,A2V}^{AC}, \quad s \in \mathcal{N}^{VSC}, \quad i \in \mathcal{N}^{AC} \quad (2.8a)$$

$$q_s^{VSC} = q_{i,A2V}^{AC}, \quad s \in \mathcal{N}^{VSC}, \quad i \in \mathcal{N}^{AC} \quad (2.8b)$$

$$p_s^{VSC} = |v_s^{VSC}|^2 g_{sf}^{VSC} - |v_s^{VSC}| |v_f^{VSC}| [g_{sf}^{VSC} \cos(\theta_s^{VSC} - \theta_f^{VSC}) + b_{sf}^{VSC} \sin(\theta_s^{VSC} - \theta_f^{VSC})], \quad s, f \in \mathcal{N}^{VSC} \quad (2.9a)$$

$$q_s^{VSC} = -|v_s^{VSC}|^2 b_{sf}^{VSC} - |v_s^{VSC}| |v_f^{VSC}| [g_{sf}^{VSC} \sin(\theta_s^{VSC} - \theta_f^{VSC}) - b_{sf}^{VSC} \cos(\theta_s^{VSC} - \theta_f^{VSC})], \quad s, f \in \mathcal{N}^{VSC} \quad (2.9b)$$

$$p_c^{VSC} = |v_c^{VSC}|^2 g_{fc}^{VSC} + |v_f^{VSC}| |v_c^{VSC}| [g_{fc}^{VSC} \cos(\theta_f^{VSC} - \theta_c^{VSC}) - b_{fc}^{VSC} \sin(\theta_f^{VSC} - \theta_c^{VSC})], \quad f, c \in \mathcal{N}^{VSC} \quad (2.10a)$$

$$q_c^{VSC} = -|v_c^{VSC}|^2 b_{fc}^{VSC} + |v_f^{VSC}| |v_c^{VSC}| [g_{fc}^{VSC} \sin(\theta_f^{VSC} - \theta_c^{VSC}) + b_{fc}^{VSC} \cos(\theta_f^{VSC} - \theta_c^{VSC})], \quad f, c \in \mathcal{N}^{VSC} \quad (2.10b)$$

$$q_f^{VSC} = -|v_f^{VSC}|^2 b_f^{VSC}, \quad f \in \mathcal{N}^{VSC} \quad (2.11)$$

$$p_c^{VSC} + p_{loss}^{VSC} + p_j^{MTDC} = 0, \quad j \in \mathcal{N}^{MTDC}, \quad c \in \mathcal{N}^{VSC} \quad (2.12)$$

$$p_{loss}^{VSC} = a_{loss}^{VSC} + b_{loss}^{VSC} |I_c^{VSC}| + c_{loss}^{VSC} |I_c^{VSC}|^2, \quad c \in \mathcal{N}^{VSC} \quad (2.13a)$$

$$0 \leq |I_c^{VSC}| \leq |\overline{I_c^{VSC}}|, \quad c \in \mathcal{N}^{VSC} \quad (2.13b)$$

$$(v_c^{VSC})^2 \cdot |I_c^{VSC}|^2 = (p_c^{VSC})^2 + (q_c^{VSC})^2, \quad c \in \mathcal{N}^{VSC} \quad (2.14)$$

$$v_s^{VSC} = v_{s,ref}^{VSC}, \quad s \in \mathcal{N}^{VSC} \quad (2.15a)$$

$$q_s^{VSC} = q_{s,ref}^{VSC}, \quad s \in \mathcal{N}^{VSC} \quad (2.15b)$$

**Goal:**

$$\min \sum_i c_{i,2}^{AC} (p_{i,gen}^{AC})^2 + c_{i,1}^{AC} p_{i,gen}^{AC} + c_{i,0}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.16)$$

- **AC Constraints:** ✓(2.1a)-(2.1d) are power flow constraints. ✓(2.2a)-(2.2b) are nodal power balance equations. ✓(2.3a)-(2.3b) are generator power output limits. (2.4) is the voltage magnitude limit.
- **MTDC Constraints:** ✓(2.5a)-(2.5b) are power flow constraints. ✓(2.6) is the voltage magnitude limit. ✓(2.7a)-(2.7c) represents the VSC control mode at DC terminal. Note that only one of them is hold depending on the chosen control model.
- **VSC Constraint:** ✓(2.8a)-(2.8b) are the active and reactive power couplings between the AC grid and VSC. ✓(2.9a)-(2.9b) are the active and reactive power injections from the view of node  $s$ . ✓(2.10a)-(2.10b) are the active and reactive power injections from the view of node  $c$ . ✓(2.11) is the reactive power injection from the view of node  $f$ . If we consider the latest converters with the MMC topology, the filter is not required to install at node  $f$  and  $b_f^{VSC} = 0$ . ✓(2.12) is the active power coupling between the MTDC grid and VSC. ✓(2.13a)-(2.13b) are related to the converter loss function. ✓(2.14) denote the voltage and current relationship from the view of node  $c$ . ✓(2.15a)-(2.15b) represent the VSC control mode at AC PCC. Note that only one of them holds depending on the chosen control model.

- **Goal:** ✓(2.16) represent minimizing power generation costs.

## 2.3 Convex relaxed formulation

The original AC/DC OPF problem is NLP. Such a problem is not favored due to their limited scalability and computational challenges. Give these reasons, we formulate a convex relaxed version of the original AC/DC OPF problem. Our demo code is based on the convex relaxed version as below.

---

Convex Relaxed AC/MTDC OPF Formulation

[SOCP]

---

**Variables:**

$$\begin{array}{ll}
 p_i^{AC}, q_i^{AC}, p_{i,gen}^{AC}, q_{i,gen}^{AC}, p_{i,A2V}^{AC}, q_{i,A2V}^{AC}, s_{ij}^{AC}, c_{ij}^{AC} & i, j \in \mathcal{N}^{AC} \\
 p_{ij}^{AC}, q_{ij}^{AC} & (i, j) \in \mathcal{L}^{AC} \\
 p_j^{MTDC}, v_j^{MTDC} & j \in \mathcal{N}^{MTDC} \\
 p_{jh}^{MTDC}, l_{jh}^{MTDC} & (j, h) \in \mathcal{L}^{MTDC} \\
 p_s^{VSC}, q_s^{VSC} & s \in \mathcal{N}^{MTDC} \\
 p_c^{VSC}, q_c^{VSC}, I_c^{VSC}, l_c^{VSC} & c \in \mathcal{N}^{MTDC} \\
 q_f^{VSC} & f \in \mathcal{N}^{MTDC} \\
 c_{mn}^{VSC}, c_{mn}^{VSC} & m, n \in \{s, f, c\} \quad s, f, c \in \mathcal{N}^{MTDC}
 \end{array}$$

**AC Constraints:**

$$p_{ij}^{AC} = G_{ij} (c_{ii}^{AC} - c_{ij}^{AC}) + B_{ij} s_{ij}^{AC}, \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.17a)$$

$$q_{ij}^{AC} = -B_{ij} (c_{ii}^{AC} - c_{ij}^{AC}) + G_{ij} s_{ij}^{AC}, \quad i, j \in \mathcal{N}^{AC}, \quad (i, j) \in \mathcal{L}^{AC} \quad (2.17b)$$

$$p_i^{AC} = c_{ii}^{AC} G_{ii} + \sum_j (c_{ij}^{AC} G_{ij} - s_{ij}^{AC} B_{ij}), \quad i, j \in \mathcal{N}^{AC} \quad (2.17c)$$

$$q_i^{AC} = -c_{ii}^{AC} B_{ii} - \sum_j (c_{ij}^{AC} B_{ij} + s_{ij}^{AC} G_{ij}), \quad i, j \in \mathcal{N}^{AC} \quad (2.17d)$$

$$c_{ij}^{AC} = c_{ji}^{AC}, \quad i, j \in \mathcal{N}^{AC} \quad (2.17e)$$

$$s_{ij}^{AC} = -s_{ji}^{AC}, \quad i, j \in \mathcal{N}^{AC} \quad (2.17f)$$

$$(c_{ij}^{AC})^2 + (s_{ij}^{AC})^2 \leq c_{ii}^{AC} c_{jj}^{AC}, \quad i, j \in \mathcal{N}^{AC} \quad (2.17g)$$

$$p_i^{AC} = p_{i,gen}^{AC} - p_{i,load}^{AC} - p_{i,A2V}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.18a)$$

$$q_i^{AC} = q_{i,gen}^{AC} - q_{i,load}^{AC} - q_{i,A2V}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.18b)$$

$$\underline{p}_{i,gen}^{AC} \leq p_{i,gen}^{AC} \leq \bar{p}_{i,gen}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.19a)$$

$$\underline{q}_{i,gen}^{AC} \leq q_{i,gen}^{AC} \leq \bar{q}_{i,gen}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.19b)$$

$$|v_i^{AC}|^2 \leq c_{ii}^{AC} \leq \overline{|v_i^{AC}|^2}, \quad i \in \mathcal{N}^{AC} \quad (2.20)$$

**MTDC constraints:**

$$p_j^{MTDC} = \sum_{(j,h)} p_{jh}^{MTDC}, \quad j \in \mathcal{N}^{MTDC}, \quad (j,h) \in \mathcal{L}^{MTDC} \quad (2.21a)$$

$$p_{jh}^{MTDC} + p_{hj}^{MTDC} = r_{jh}^{MTDC} l_{jh}^{MTDC}, \quad (j,h) \in \mathcal{L}^{MTDC} \quad (2.21b)$$

$$u_j^{MTDC} - u_h^{MTDC} = r_{jh}^{MTDC} (p_{jh}^{MTDC} - p_{hj}^{MTDC}), \quad j, h \in \mathcal{N}^{MTDC}, \quad (j,h) \in \mathcal{L}^{MTDC} \quad (2.21c)$$

$$p_{jh}^{MTDC} \leq l_{jh}^{MTDC} u_j^{MTDC} \quad j \in \mathcal{N}^{MTDC}, \quad (j,h) \in \mathcal{L}^{MTDC} \quad (2.21d)$$

$$\underline{u}_j^{MTDC} \leq u_j^{MTDC} \leq \bar{u}_j^{MTDC}, \quad j \in \mathcal{N}^{MTDC} \quad (2.22)$$

$$p_j^{MTDC} = p_{j,ref}^{MTDC}, \quad j \in \mathcal{N}^{MTDC} \quad (2.23a)$$

$$u_j^{MTDC} = (v_{j,ref}^{MTDC})^2, \quad j \in \mathcal{N}^{MTDC} \quad (2.23b)$$

$$p_j^{MTDC} = p_{j,ref}^{MTDC} - \frac{1}{k_{j,droop}^{MTDC}} \left( \frac{1}{2} + \frac{1}{2} u_j^{MTDC} - v_{j,ref}^{MTDC} \right), \quad j \in \mathcal{N}^{MTDC} \quad (2.23c)$$

**VSC constraints:**

$$p_s^{VSC} = p_{i,A2V}^{AC}, \quad s \in \mathcal{N}^{VSC} \quad (2.24a)$$

$$q_s^{VSC} = q_{i,A2V}^{AC}, \quad s \in \mathcal{N}^{VSC} \quad (2.24b)$$

$$p_s^{VSC} = c_{ss}^{VSC} g_{sf}^{VSC} - c_{sf}^{VSC} g_{sf}^{VSC} + s_{sf}^{VSC} b_{sf}^{VSC}, \quad s, f \in \mathcal{N}^{MTDC} \quad (2.25a)$$

$$q_s^{VSC} = -c_{ss}^{VSC} b_{sf}^{VSC} + c_{sf}^{VSC} b_{sf}^{VSC} + s_{sf}^{VSC} g_{sf}^{VSC}, \quad s, f \in \mathcal{N}^{MTDC} \quad (2.25b)$$

$$c_{sf}^{VSC} = c_{fs}^{VSC}, \quad s, f \in \mathcal{N}^{AC} \quad (2.25c)$$

$$s_{sf}^{VSC} = -s_{fs}^{VSC}, \quad s, f \in \mathcal{N}^{VSC} \quad (2.25d)$$

$$(c_{sf}^{VSC})^2 + (s_{sf}^{VSC})^2 \leq c_{ss}^{VSC} c_{ff}^{VSC}, \quad s, f \in \mathcal{N}^{AC} \quad (2.25e)$$

$$p_c^{VSC} = c_{cc}^{VSC} g_{cf}^{VSC} - c_{cf}^{VSC} g_{cf}^{VSC} + s_{cf}^{VSC} b_{cf}^{VSC}, \quad c, f \in \mathcal{N}^{MTDC} \quad (2.26a)$$

$$q_c^{VSC} = -c_{cc}^{VSC} b_{cf}^{VSC} + c_{cf}^{VSC} b_{cf}^{VSC} + s_{cf}^{VSC} g_{cf}^{VSC}, \quad c, f \in \mathcal{N}^{MTDC} \quad (2.26b)$$

$$c_{cf}^{VSC} = c_{fc}^{VSC}, \quad c, f \in \mathcal{N}^{VSC} \quad (2.26c)$$

$$s_{cf}^{VSC} = -s_{fc}^{VSC}, \quad c, f \in \mathcal{N}^{VSC} \quad (2.26d)$$

$$(c_{cf}^{VSC})^2 + (s_{cf}^{VSC})^2 \leq c_{cc}^{VSC} c_{ff}^{VSC}, \quad c, f \in \mathcal{N}^{VSC} \quad (2.26e)$$

$$q_f^{VSC} = -c_{ff}^{VSC} b_f^{VSC}, \quad f \in \mathcal{N}^{VSC} \quad (2.27)$$

$$p_c^{VSC} + p_{loss}^{VSC} + p_j^{MTDC} = 0, \quad j \in \mathcal{N}^{MTDC}, \quad c \in \mathcal{N}^{VSC} \quad (2.28)$$

$$p_{loss}^{VSC} = a_{loss}^{VSC} + b_{loss}^{VSC} |I_c^{VSC}| + c_{loss}^{VSC} l_c^{VSC}, \quad c \in \mathcal{N}^{VSC} \quad (2.29a)$$

$$0 \leq |I_c^{VSC}| \leq \left| \overline{I_c^{VSC}} \right|, \quad c \in \mathcal{N}^{VSC} \quad (2.29b)$$

$$0 \leq l_c^{VSC} \leq \left| \overline{I_c^{VSC}} \right|^2, \quad c \in \mathcal{N}^{VSC} \quad (2.29c)$$

$$|I_c^{VSC}|^2 \leq l_c^{VSC}, \quad c \in \mathcal{N}^{VSC} \quad (2.30a)$$

$$(p_c^{VSC})^2 + (q_c^{VSC})^2 \leq c_{cc}^{VSC} \cdot l_c^{VSC} \quad c \in \mathcal{N}^{VSC} \quad (2.30b)$$

$$c_{ss}^{VSC} = (v_{s,ref}^{VSC})^2, \quad s \in \mathcal{N}^{VSC} \quad (2.31a)$$

$$q_s^{VSC} = q_{s,ref}^{VSC}, \quad s \in \mathcal{N}^{VSC} \quad (2.31b)$$

Goal:

$$\min \sum_i c_{i,2}^{AC} (p_{i,gen}^{AC})^2 + c_{i,1}^{AC} p_{i,gen}^{AC} + c_{i,0}^{AC}, \quad i \in \mathcal{N}^{AC} \quad (2.32)$$

- 
- **AC Constraints:** ✓(2.17a)-(2.17g) consist of the second-order cone (SOC) relaxed power flow constraints [4]. ✓(2.18a)-(2.18b) are nodal power balance equations. ✓(2.19a)-(2.19b) are generator power output limits. (2.20) is the squared voltage magnitude limit.
  - **MTDC Constraints:** ✓(2.21a)-(2.21d) consist of the SOC relaxed power flow constraints [5]. ✓(2.22) is the squared voltage magnitude limit. ✓(2.23a)-(2.23c) represents the VSC control mode at DC terminal. Especially, the term  $v_{ij}^{MTDC}$  is replaced by the expression with respect to  $u_{ij}^{MTDC}$  in (2.23c), through Talyor expansion. Note that for (2.23a)-(2.23c), only one of them holds depending on the chosen control model.
  - **VSC Constraint:** ✓(2.24)-(2.24b) are the active and reactive power couplings between the AC grid and VSC. ✓(2.25a)-(2.25e) consist of the SOC relaxed active and reactive power injections from the view of node  $s$ . ✓(2.26a)-(2.26e) consist of the SOC relaxed active and reactive power injections from the view of node  $c$ . ✓(2.27) is the reactive power injection from the view of node  $f$ . ✓(2.28) is the active power coupling between the MTDC grid and VSC. ✓(2.29a)-(2.29c) are related to the converter loss function. ✓(2.30a)-(2.30b) consist of the SOC relaxed voltage and current relationship from the view of node  $c$ . ✓(2.31a)-(2.31b) represent the VSC control mode at AC PCC. Note that only one of them is hold depending on the chosen control model.
  - **Goal:** ✓(2.32) represent minimizing power generation costs.

## 3 Code Implementation

### 3.1 Data preparation

The data required for AC/DC OPF includes the fundamental data of the interconnected AC grids and the fundamental data of the MTDC grid, which are listed and shown in Fig. 3.1. All data are stored in .CSV files due to its compatibility with various language environments.

Users can manually fill in power grid parameters into respective .CSV files based on their specific needs. Below we present the details regarding the respective .CSV. The style of the saved data in .CSV is basically consistent with mpc structure used in Matpower<sup>3</sup> and MatACDC<sup>4</sup>.

For the DC grid, the required fundamental data files includes `baseMW_dc.csv`, `pol_dc.csv`, `bus_dc.csv`, `branch_dc.csv`, `branch_dc.csv`. In the file, each column of the data corresponds to a parameter item. A specific rule needs to be followed when saving data into these files. The details about parameter items and descriptions of DC grid data files can be found in Table 1, 2, 3, 4, 5. Note that in `bus_dc.csv` and `branch_dc.csv` files, some parameter items may appear to be meaningless but serve as placeholders to maintain the same format style with the well-known Matpower.

---

<sup>3</sup>MATPOWER Manual

<sup>4</sup>MatACDC Manual

Required data (.CSV)	
Power grids data sheets	
-baseMVA_ac	-baseMW_dc
-bus_ac	-pol_dc
-branch_ac	-bus_dc
-gen_ac	-branch_dc
-gencost_ac	-conv_dc

Figure 3.1: Required power grid data sheet

For the AC grid, the required fundamental data files includes `baseMVA_ac.csv`, `bus_ac.csv`, `branch_ac.csv`, `gen_ac.csv`, `gencost_ac.csv`. The details about parameter items and descriptions of AC grid data files can be found in Table 6, 7, 8, 9, 10. Matpower provides various AC grid case files. Sometimes, users may want to directly select and integrate multiple AC grids without manually modifying data files. To facilitate this process, we provide auxiliary functions `merge_ac` and `save_csv`. User can build up `Matlab` script as below.

```

1 mpc_merged = merge_ac('case9', 'case14'); % merge_ac('case_a', 'case_b', '...')
2 % If no path is provided, saved in the current path.
3 save_csv(mpc_merged, directory);

```

After running the script, users can name the prefix of the series of .CSV filenames related to ac grids and directly type it. Finally, save information is shown in command window, like blow.

```

>> Reordered bus IDs for case9
>> Reordered bus IDs for case14
>> Enter a prefix for the CSV filenames: ac9ac14
>> Merged AC grid data has already been saved to:
>> D:\acdcopf\Tests\ac9ac14_baseMVA_ac.csv
>> D:\acdcopf\Tests\ac9ac14_bus_ac.csv
>> D:\acdcopf\Tests\ac9ac14_gen_ac.csv
>> D:\acdcopf\Tests\ac9ac14_branch_ac.csv
>> D:\acdcopf\Tests\ac9ac14_gencost_ac.csv

```

Column	Parameters	Description [Unit]
1	baseMW	Base power capacity [MW].

Table 1: baseMW\_dc.csv file

Column	Parameters	Description [Unit]
1	pol	Number of poles (1=monopolar grid, 2=bipolar grid) [-].

Table 2: pol\_dc.csv file

Column	Parameters	Description [Unit]
1	bus_i	Bus number [-].
2	—	N/A.
3	Pd	Active power loads [MW].
4	—	N/A.
5	—	N/A.
6	—	N/A.
7	—	N/A.
8	—	N/A.
9	—	N/A.
10	baseKV	Bus voltage level [kV].
11	—	N/A.
12	Vmax	Maximum bus voltage magnitude [p.u.].
13	Vmin	Minimum bus voltage magnitude [p.u.].

Table 3: bus\_dc.csv file

Column	Parameters	Description [Unit]
1	fbus	From bus index [-].
2	tbus	To bus index [-].
3	r	Branch resistance [p.u.].
4	—	N/A.
5	—	N/A.
6	—	N/A.
7	—	N/A.
8	—	N/A.
9	—	N/A.
10	—	N/A.
11	—	N/A.
12	—	N/A.
13	—	N/A.

Table 4: branch\_dc.csv file

Column	Parameters	Description [Unit]
1	busdc_i	Converter-connected DC bus index [-].
2	busac_i	Converter-connected AC bus index [-].
3	gridac	AC grid number of which converter belongs [-].
4	type_dc	Converter DC-side control modes (1=Constant DC power control, 2=Constant DC voltage control, 3=DC droop control) [-].
5	type_ac	Converter AC-side control modes (1=Constant AC voltage control, 2=Constant reactive power control) [-].
6	P_g	Setting of the active power output of converter DC terminal (applicable to the constant DC power control) [MW].
7	Q_g	Setting of the reactive power output of converter PCC bus (applicable to the constant reactive power control [MVar].
8	Vtar	Setting of the voltage amplitude of converter DC terminal (applicable to the constant DC voltage control) [p.u.].
9	rtf	Resistance of the converter transformer [p.u.].
10	xtf	Reactance of the converter transformer [p.u.].
11	bf	Conductance of the converter filter [p.u.].
12	rc	Resistance of the converter Phase reactor [p.u.].
13	xc	reactance of the converter Phase reactor [p.u.].
14	basekVac	Base voltage of the converter AC side [p.u.].
15	Vmmax	Maximum voltage magnitude of the converter AC side [p.u.].
16	Vmmin	Minimum voltage magnitude of the converter AC side [p.u.].
17	Imax	Maximum current magnitude of the converter [p.u.].
18	status	Converter status (1=in service, 0=out of service) [p.u.].
19	LossA	Constant loss coefficient [MW].
20	LossB	Linear loss coefficient [kV].
21	LossCR	Rectifier quadratic loss coefficient [ $\Omega$ ].
22	LossCI	Inverter quadratic loss coefficient [ $\Omega$ ].

Table 5: conv\_dc.csv file

Column	Parameters	Description [Unit]
1	baseMVA	Base power capacity [MVA].

Table 6: baseMVA\_ac.csv file

Column	Parameters	Description [Unit]
1	bus_i	Bus number [-].
2	type	Bus type (3=Slack, 2=PV, 1=PQ) [-]
3	Pd	Active power loads [MW].
4	Qd	Reactive power loads [MVar].
5	Gs	Shunt conductance (at V=1.0 p.u.) [MW].
6	Bs	Shunt susceptance (at V=1.0 p.u.) [MVar].
7	area	Area number [-].
8	Vm	Reactive power loads [p.u.].
9	Va	Reactive power loads [degrees].
10	baseKV	Bus voltage level [kV].
11	zone	Loss zone [-].
12	Vmax	Maximum bus voltage magnitude [p.u.].
13	Vmin	Minimum bus voltage magnitude [p.u.].

Table 7: bus\_ac.csv file

Column	Parameters	Description [Unit]
1	fbus	From bus index [-].
2	tbus	To bus index [-].
3	r	Branch resistance [p.u.].
4	x	Branch reactance [p.u.].
5	b	Line charging susceptance.
6	rateA	Rate A (MVA limit for normal operation) [MVA].
7	rateB	Rate B (MVA limit for short-term) [MVA].
8	rateC	Rate C (MVA limit for emergency).
9	ratio	Transformer tap ratio (if applicable).
10	angle	Transformer phase shift angle [degrees].
11	status	Line status (1=active, 0=out-of-service).
12	angmin	Minimum voltage angle difference [-].
13	angmax	Maximum voltage angle difference [-].

Table 8: branch\_ac.csv file



Column	Parameters	Description [Unit]
1	bus	Generator's bus number [-].
2	Pg	Active power output [MW].
3	Qg	Reactive power output [MVar].
4	Qmax	Maximum reactive power output [MVar].
5	Qmin	Minimum reactive power output [MVar].
6	Vg	Voltage magnitude setpoint [p.u.].
7	mBase	Total MVA base of generator [MVA].
8	status	Generator status (1=in service, 0=out of service) [-].
9	Pmax	Maximum active power output [MW].
10	Pmin	Minimum active power output [MW].
11	PC1	lower real power output of PQ capability curve [MW].
12	PC2	upper real power output of PQ capability curve [MW].
13	Qc1min	Minimum reactive power output at Pc1 [MVar].
14	Qc1max	Maximum reactive power at PC1 [MVar].
15	Qc2min	Minimum reactive power at PC2 [MVar].
16	Qc2max	Maximum reactive power at PC2 [MVar].
17	ramp_agc	Ramp rate for automatic generation control [MW/min].
18	ramp_10	Ramp rate for 10-minute reserves [MW].
19	ramp_30	Ramp rate for 30-minute reserves [MW].
20	ramp_q	Ramp rate for reactive power [MVar/min].
21	apf	Area participation factor [-].

Table 9: gen\_ac.csv file

Column	Parameters	Description [Unit]
1	model	Cost function type(1=piecewise, 2=polynomial) [-].
2	startup	Startup cost [\$].
3	shutdown	Shutdown cost [\$].
4	n	Number of data points (if piecewise linear) or order of polynomial [\$].
5	cost	Coefficients of polynomial (from highest to lowest order) or piecewise data points.

Table 10: gencost\_ac.csv file

### 3.2 Parameter Processing

Language-specific functions `create_ac` and `create_dc` reads .CSV file data associated with the interconnected AC grids and the MTDC grid. Depending on the programming environment, different handling approaches are used to read and organize these .csv files, as listed in Table 11.

Subsequently, language-specific functions `params_ac` and `params_dc` are further utilized to extract the essential and critical parameters required for constructing the AC/DC OPF model. During this process, different language-specific conventions are adopted to define one-dimensional, two-dimensional, and three-dimensional parameters, which are listed in Table 12.

Table 11: .CSV File Data Handling Approaches Across Programming Languages

Language	Read .csv Files	Transformed Data Structure
Matlab	<code>readmatrix</code> (built-in)	Uses a <code>struct</code> where each field name maps to a numeric matrix.
Python	<code>pandas.read_csv</code>	Uses a <code>dict</code> where each string key maps to a <code>NumPy</code> array or a scalar.
Julia	<code>CSV.File</code> + <code>DataFrame</code>	Uses a <code>Dict{String, Matrix}</code> where each string key maps to a numeric array or a scalar.
Cpp	<code>ifstream</code> + <code>csv_reader</code> (custom)	Uses an <code>unordered_map&lt;std::string, Eigen::MatrixXd&gt;</code> where each string key maps to an Eigen matrix.

Table 12: Parameter Defining Across Programming Languages

Language	1D Parameter ( <code>fbus_dc</code> , <code>nbuses_ac</code> , etc.)	2D Parameter ( <code>bus_dc</code> , <code>pd_ac</code> , etc.)	3D Parameter ( <code>bus_entire_ac</code> , etc.)
Matlab	<code>vector/x(i)</code> , <code>cell/x{n}</code>	<code>matrix/x(i, j)</code> , <code>cell/x{n}(i)</code>	<code>cell/x{n}(i, j)</code>
Python	<code>np.ndarray/x[i]</code> , <code>list/x[n]</code>	<code>np.ndarray/x[i, j]</code> , <code>list/x[n][i]</code>	<code>list/x[n][i, j]</code>
Julia	<code>Vector/x[i](x[n])</code>	<code>Matrix/x[i, j]</code> , <code>Vector/x[n][i]</code>	<code>Vector/x[n][i, j]</code>
Cpp	<code>Eigen::Vector/x(i)</code> , <code>std::vector/x[i]</code>	<code>Eigen::Matrix/x(i, j)</code> , <code>std::vector/x[n](i)</code>	<code>std::vector/x[n](i, j)</code>

★ The mark means `type/syntax`

### 3.3 Variable definition

The core OPF solver can be found in `solve_opf`, where the AC/DC OPF model is formulated and solved. **ACDC-OpFlow** formulates AC/DC OPF model as a SOCP problem, which has been presented in Section 2.3.

In implementations across [Matlab](#), [Python](#), [Julia](#), and [Cpp](#), **ACDC-OpFlow** utilizes Gurobi, a powerful convex optimization solver that supports SOCP solving and offers academic licenses free of charge for research and educational use<sup>5</sup>, as the unified solver backend. To facilitate AC/DC OPF modelling and solver invocation, **ACDC-OpFlow** integrates each language with a widely adopted modelling framework and corresponding solver interface. Table 13 summarizes the dependencies used for optimization in each programming environment.

To further support the optimization modeling, each implementation explicitly defines opti-

<sup>5</sup><https://www.gurobi.com/academia/academic-program-and-licenses/>

<sup>6</sup><https://yalmip.github.io/>

<sup>7</sup><https://www.pyomo.org/>

<sup>8</sup><https://jump.dev/>

<sup>9</sup><https://docs.gurobi.com/projects/optimizer/en/current/reference/cpp.html>

Table 13: Optimization Modeling Dependencies Across Programming Languages

Language	Dependencies
Matlab	<b>YALMIP</b> <sup>6</sup> (modeling + solver interface)
Python	<b>Pyomo</b> <sup>7</sup> (modeling) + <b>gurobipy</b> (solver interface)
Julia	<b>JuMP</b> <sup>8</sup> (modeling) + <b>Gurobi.jl</b> (solver interface)
Cpp	<b>Gurobi C++ API</b> <sup>9</sup> (modeling + solver interface)

mization decision variables in compatible formats with each modeling framework, as shown in Table 14.

Table 14: Optimization Variable Definition Across Languages

Language	Define Variables
Matlab	<code>sdpvar(...)</code>
Python	<code>Var(..., domain = Reals)</code>
Julia	<code>@variable(model, ...)</code>
Cpp	<code>model.addVar(..., GRB_CONTINUOUS)</code>

Similar to parameter definition, optimization variable definition also conforms to language-specific conventions, and the summary of representative expressions for defining one-dimensional, two-dimensional, and three-dimensional variables in each language environment are listed below:

```

1  %% Variable Defining in Matlab via YALMIP
2
3  % 1D Variable (e.g., vn2_dc)
4  vn2_dc = sdpvar(nbuses_dc, 1);
5  % 2D Variable (e.g., pij_dc)
6  pij_dc = sdpvar(nconvs_dc, nconvs_dc, 'full');
7  % 2D Variable (e.g., vn2_ac)
8  vn2_ac = cell(ngrids, 1);
9  for ng = 1:ngrids
10     vn2_ac{ng} = sdpvar(nbuses_ac{ng}, 1);
11 end
12 % 3D Variable (e.g., pij_ac)
13 pij_ac = cell(ngrids, 1);
14 for ng = 1:ngrids
15     pij_ac{ng} = sdpvar(nbuses_ac{ng}, nbuses_ac{ng}, 'full');
16 end

```

```

1  # Variable definition in Python via Pyomo
2
3  # 1D variable (e.g., vn2_dc)
4  model.vn2_dc = Var(range(nbuses_dc), domain=Reals)

```

```

5 # 2D variable (e.g., pij_dc)
6   model.pij_dc = Var(range(nbuses_dc), range(nbuses_dc), domain=Reals)
7 # 2D variable (e.g., vn2_ac)
8   model.vn2_ac = Var(
9     [(ng, i) for ng in range(ngrids) for i in range(nbuses[ng])],
10    domain=Reals)
11 # 3D variable (e.g., pij_ac)
12   model.pij_ac = Var(
13     [(ng, i, j) for ng in range(ngrids) for i in range(nbuses[ng]) for j in
14       ↪ range(nbuses[ng])],
15    domain=Reals)

```

```

1 # Variable Definition in Julia via JuMP
2
3 # 1D Variable (e.g., vn2_dc)
4   vn2_dc = @variable(model, [1:nbuses_dc])
5 # 2D Variable (e.g., pij_dc)
6   @variable(model, pij_dc[1:nbuses_dc, 1:nbuses_dc])
7 # 2D Variable (e.g., vn2_ac)
8   vn2_ac = Vector{Vector{JuMP.VariableRef}}(undef, ngrids)
9   for ng in 1:ngrids
10     vn2_ac[ng] = @variable(model, [1:nbuses_ac[ng]])
11   end
12 # 3D Variable (e.g., pij_ac)
13   pij_ac = Vector{Matrix{JuMP.VariableRef}}(undef, ngrids)
14   for ng in 1:ngrids
15     pij_ac[ng] = @variable(model, [1:nbuses_ac[ng], 1:nbuses_ac[ng]])
16   end

```

```

1 // Variable Definition in Cpp via Gurobi Cpp API
2
3 // 1D Variable (e.g., vn2_dc)
4   Eigen::Matrix<GRBVar, Eigen::Dynamic, 1> vn2_dc(nbuses_dc);
5   for (int i = 0; i < nbuses_dc; ++i) {
6     vn2_dc(i) = model.addVar(-GRB_INFINITY, GRB_INFINITY, 0.0, GRB_CONTINUOUS);
7   }
8 // 2D Variable (e.g., pij_dc)
9   Eigen::Matrix<GRBVar, Eigen::Dynamic, Eigen::Dynamic> pij_dc(nbuses_dc, nbuses_dc);
10  for (int i = 0; i < nbuses_dc; ++i) {
11    for (int j = 0; j < nbuses_dc; ++j) {
12      pij_dc(i, j) = model.addVar(-GRB_INFINITY, GRB_INFINITY, 0.0,
13        ↪ GRB_CONTINUOUS);
14    }
15  }
16 // 2D Variable (e.g., vn2_ac)
17   std::vector<Eigen::Matrix<GRBVar, Eigen::Dynamic, 1>> vn2_ac(ngrids);
18   vn2_ac[ng].resize(nbuses_ac[ng]);

```

```

18   for (int i = 0; i < nbuses_ac[ng]; ++i) {
19       vn2_ac[ng](i) = model.addVar(-GRB_INFINITY, GRB_INFINITY, 0.0, GRB_CONTINUOUS);
20   }
21   // 3D Variable (e.g., pij_ac)
22   std::vector<Eigen::Matrix<GRBVar, Eigen::Dynamic, Eigen::Dynamic>> pij_ac(ngrids);
23   pij_ac[ng] = Eigen::Matrix<GRBVar, Eigen::Dynamic, Eigen::Dynamic>(nbuses_ac[ng],
24       ↪ nbuses_ac[ng]);
25   for (int i = 0; i < nbuses_ac[ng]; ++i) {
26       for (int j = 0; j < nbuses_ac[ng]; ++j) {
27           pij_ac[ng](i, j) = model.addVar(-GRB_INFINITY, GRB_INFINITY, 0.0,
28               ↪ GRB_CONTINUOUS);
29       }
30   }

```

### 3.4 Constraint construction

Each implementation explicitly adds operational constraints in compatible formats with each modeling framework, as shown in Table 15. Next, we will take the addition of SOC-based DC power flow constraints as an example to present code snippets under different implementations more specifically.

Table 15: Constraint Construcion Across Languages

Language	Constraints Construction
Matlab	Cons = [Cons; expr]
Python	model.Constraint = Constraint(expr)
Julia	@constraint(model, expr)
Cpp	model.addConstr(expr) (for linear constraints), model.addQConstr(expr) (for SOC constraints)

```

1  %% Constraint Construction in Matlab via YALMIP, with vectorized formulation.
2
3  % --- DC Power Flow Constraints (Second-Order Cone Relaxation) ---
4  zij_dc = 1./abs(ybus_dc);
5  zij_dc = zij_dc-diag(diag(zij_dc));
6  zij_dc(isinf(zij_dc)) = 1e4;
7
8  con_dc = [con_dc;
9      pn_dc == sum(pij_dc, 2) * pol_dc;
10     pij_dc + pij_dc' == zij_dc .* lij_dc;
11     repmat(vn2_dc, 1, nbuses_dc) - repmat(vn2_dc', nbuses_dc, 1) == zij_dc .* (pij_dc
12     ↪ - pij_dc');
13     pij_dc.^2 <= lij_dc .* repmat(vn2_dc, 1, nbuses_dc);

```

```

13     vn2_dc >= 0;
14     lij_dc >= 0];

```

```

1  # Constraint Construction in Python via Pyomo, with for-loop formulation.
2
3  model.addconstraints = ConstraintList()
4  # -----
5  # DC Power Flow Constraints (Second-Order Cone Relaxation)
6  # -----
7  zij_dc = 1.0 / np.abs(ybus_dc.toarray())
8  zij_dc = zij_dc - np.diag(np.diag(zij_dc))
9  zij_dc[np.isinf(zij_dc)] = 1e4
10
11 for i in range(nbuses_dc):
12     model.addconstraints.add(model.pn_dc[i] == pol_dc * sum(model.pij_dc[i, j] for j
13     ↪ in range(nbuses_dc)))
14     model.addconstraints.add(model.vn2_dc[i] >= 0)
15     for j in range(nbuses_dc):
16         model.addconstraints.add(model.pij_dc[i, j] + model.pij_dc[j, i] == zij_dc[i,
17         ↪ j] * model.lij_dc[i, j])
18         model.addconstraints.add(model.vn2_dc[i] - model.vn2_dc[j] == zij_dc[i, j] *
19         ↪ (model.pij_dc[i, j] - model.pij_dc[j, i]))
20         model.addconstraints.add(model.pij_dc[i, j]**2 <= model.lij_dc[i, j] *
21         ↪ model.vn2_dc[i])
22         model.addconstraints.add(model.lij_dc[i, j] >= 0)

```

```

1  # Constraint Construction in Julia via JuMP, with vectorized formulation.
2
3  # -----
4  # DC Power Flow Constraints (Second-Order Cone Relaxation)
5  # -----
6  zij_dc = 1.0 ./ abs.(ybus_dc)
7  zij_dc = zij_dc .- Diagonal(diag(zij_dc))
8  zij_dc[isinf.(zij_dc)] .= 1e4
9  @constraint(model, pn_dc .== pol_dc * sum(pij_dc, dims=2))
10 @constraint(model, pij_dc .+ pij_dc' .== zij_dc .* lij_dc)
11 @constraint(model, vn2_dc .- vn2_dc' .== zij_dc .* (pij_dc .- pij_dc'))
12 @constraint(model, pij_dc.^2 .<= lij_dc .* repeat(vn2_dc, 1, nbuses_dc))
13 @constraint(model, vn2_dc .>= 0)
14 @constraint(model, lij_dc .>= 0)

```

```

1  // Constraint Construction in Cpp via Gurobi Cpp API, with for-loop formulation.
2
3  // 1. Constrains for dc power flow - second-order cone relaxation
4  Eigen::MatrixXd y_dc_dense = Eigen::MatrixXd(y_dc);
5  Eigen::MatrixXd zij_dc = y_dc_dense.array().abs().cwiseInverse().matrix();

```

```

6  zij_dc -= zij_dc.diagonal().asDiagonal();
7  for (int i = 0; i < zij_dc.rows(); ++i) {
8      for (int j = 0; j < zij_dc.cols(); ++j) {
9          if (std::isinf(zij_dc(i, j))) {
10             zij_dc(i, j) = 1e4;
11         }
12     }
13 }
14
15 for (int i = 0; i < nbuses_dc; ++i) {
16     GRBLinExpr power_flow_sum = 0.0;
17     for (int j = 0; j < nbuses_dc; ++j) {
18         power_flow_sum += pij_dc(i, j);
19     }
20     model.addConstr(pn_dc(i) == power_flow_sum * pol_dc);
21 }
22
23 for (int i = 0; i < nbuses_dc; ++i) {
24     for (int j = 0; j < nbuses_dc; ++j) {
25         model.addConstr(pij_dc(i, j) + pij_dc(j, i) == zij_dc(i, j) * lij_dc(i, j));
26         model.addQConstr(pij_dc(i, j) * pij_dc(i, j) <= lij_dc(i, j) * vn2_dc(i));
27         model.addConstr(vn2_dc(i) - vn2_dc(j) == zij_dc(i, j) * (pij_dc(i, j) -
28             ↪ pij_dc(j, i)));
29     }
30 }
31
32 for (int i = 0; i < nbuses_dc; ++i) {
33     model.addConstr(vn2_dc(i) >= 0.0);
34     for (int j = 0; j < nbuses_dc; ++j) {
35         model.addConstr(lij_dc(i, j) >= 0.0);
36     }
37 }

```

## 4 Examples

### 4.1 Running a case

[Matlab](#), [Python](#), [Julia](#), and [C++](#) follows a unified running process. As mentioned earlier, before running the AC/DC OPF, we need to prepare the required .CSV files to save the parameter data related to interconnected AC grids and the MTDC grid. In each programming language, users can execute AC/DC OPF through the code snippet as below.

```

1  solve_opf('mtdc3slack_a', 'ac9ac14',
2           'vscControl', true,
3           'writeTxt', false,
4           'plotResult', true)

```

Noted that in `solve_opf()`, besides attributes `'matdc3slack_a'` and `'ac9ac14'`, which respectively represent the case name of MTDC grid and interconnected AC grids, there are three additional attributes: `'vscControl'`, `'writeTxt'`, and `'plotResult'` need to be set:

- `'vscControl'` is used to determine if the VSC control constraints formulated as (2.23a)-(2.23c) and (2.31a)-(2.31b) are activated. Its default setting is `true` means the VSC control constraints need to be added into the formed AC/DC OPF model, and the control settings are pre-defined in the imported `conv_dc.csv` file. When the VSC control mode is considered to be optimized, it should not be pre-specified. `'vscControl'` should be set to `false`, which means that (2.23a)-(2.23c) and (2.31a)-(2.31b) are excluded from the formed AC/DC OPF model.
- `'writeTxt'` is used to determine whether the printed AC/DC OPF results shown in Fig. 4.1 should be saved as a text file. By default, it is set to `false`, and no `.txt` file will be generated. If set to `true`, a `.txt` file named `opf_result.txt` will be produced in the current working directory. Regardless of whether `'writeTxt'` is set to `true` or `false`, the textual results shown in Fig. 4.1 will always be printed to the terminal window.

Figure 4.1: Printed AC/DC OPF text.

AC Grids Bus Data							
Area #	Branch #	Voltage		Generation		Load	
		Mag [pu]	Ang [deg]	Pg [MW]	Qg [MVar]	Pd [MW]	Qd [MVar]
1	1	1.048	0.000	100.966	119.890	0.000	0.000
1	2	1.100	0.000	150.474	299.987	0.000	0.000
1	3	1.045	0.000	104.361	96.882	0.000	0.000
1	4	1.011	0.000	-	-	0.000	0.000
1	5	1.000	0.000	-	-	90.000	30.000
1	6	1.022	0.000	-	-	0.000	0.000
1	7	1.009	0.000	-	-	100.000	35.000
1	8	1.022	0.000	-	-	0.000	0.000
1	9	0.987	0.000	-	-	125.000	50.000
2	1	1.048	0.000	187.567	10.000	0.000	0.000
2	2	1.025	0.000	35.467	24.251	21.700	12.700
2	3	0.992	0.000	11.719	23.330	94.200	19.000
2	4	0.997	0.000	-	-	47.800	-3.900
2	5	1.000	0.000	-	-	7.600	1.600
2	6	1.041	0.000	0.000	16.485	11.200	7.500
2	7	1.034	0.000	-	-	0.000	0.000
2	8	1.060	0.000	0.000	15.918	0.000	0.000
2	9	1.027	0.000	-	-	29.500	16.600
2	10	1.022	0.000	-	-	9.000	5.800
2	11	1.028	0.000	-	-	3.500	1.800
2	12	1.025	0.000	-	-	6.100	1.600
2	13	1.021	0.000	-	-	13.500	5.800
2	14	1.006	0.000	-	-	14.900	5.000

The total generation cost is \$ 13014.03/MWh(€12050.03/MWh)

AC Grids Branch Data



Area #	Branch #	From Bus#	To Bus#	From Branch Flow		To Branch Flow		Branch Loss
				Pij [MW]	Qij [MVar]	Pij [MW]	Qij [MVar]	Pij_loss [MW]
1	1	1	4	100.966	119.890	-100.966	-12.835	0.000
1	2	4	5	24.744	7.993	-24.632	-7.385	0.112
1	3	5	6	-42.058	-1.973	42.750	4.987	0.691
1	4	3	6	104.361	96.882	-104.361	17.056	0.000
1	5	6	7	61.612	7.580	-61.173	-3.863	0.439
1	6	7	8	-38.827	-12.901	38.967	14.084	0.140
1	7	8	2	-89.193	-4.330	89.193	259.987	0.000
1	8	8	9	50.226	14.014	-49.394	-9.824	0.833
1	9	9	4	-75.606	-16.689	76.221	21.918	0.615
2	1	1	2	121.135	4.877	-118.541	3.044	2.594
2	2	1	5	66.432	10.722	-64.203	-1.523	2.228
2	3	2	3	69.981	5.165	-67.778	4.116	2.203
2	4	2	4	38.953	4.642	-38.101	-2.058	0.851
2	5	2	5	23.375	7.376	-23.049	-6.381	0.326
2	6	3	4	-14.703	2.998	14.856	-2.607	0.153
2	7	4	5	-67.353	15.064	67.993	-13.046	0.640
2	8	4	7	29.925	-15.834	-29.925	20.569	0.000
2	9	4	9	12.873	-5.169	-12.873	6.213	0.000
2	10	5	6	45.496	-7.524	-45.496	27.907	0.000
2	11	6	11	8.435	2.932	-8.365	-2.786	0.070
2	12	6	12	7.611	2.612	-7.537	-2.460	0.073
2	13	6	13	18.250	6.893	-18.018	-6.436	0.232
2	14	7	8	-0.000	-15.521	0.000	15.918	0.000
2	15	7	9	29.925	6.442	-29.925	-5.477	0.000
2	16	9	10	4.166	4.892	-4.154	-4.859	0.012
2	17	9	14	9.131	3.888	-9.013	-3.636	0.119
2	18	10	11	-4.846	-0.941	4.865	0.986	0.019
2	19	12	13	1.437	0.860	-1.431	-0.854	0.006
2	20	13	14	5.949	1.490	-5.887	-1.364	0.062

The total AC network losses is 12.420 MW .

MTDC Bus Data							
Bus DC #	Bus AC #	Area #	DC Voltage Vdc [pu]	DC Power Pdc [MW]	PCC Bus Ps [MW]	Injection Qs [MVar]	Converter loss Conv_Ploss [MW]
1	2	1	1.008	60.000	61.281	40.000	1.210
2	5	1	1.000	-24.443	-23.310	5.158	1.124
3	5	2	0.998	-35.000	-33.837	0.001	1.145

The total converter losses is 3.479 MW

MTDC Branch Data					
Branch #	From Bus#	To Bus#	From Branch Flow Pij [MW]	To Branch Flow Pij [MW]	Branch Loss Pij_loss [MW]
1	1	2	31.986	-31.724	0.262
2	2	3	7.281	-7.268	0.014

3	1	3	28.014	-27.732	0.282
---	---	---	--------	---------	-------

The total DC network losses is 0.557 MW .

Execution time is 2.091s .

- `'plotResult'` is used to determine whether the visualization of AC/DC OPF results, as shown in Fig. 4.2, is enabled. By default, it is set to `true`, and the corresponding plots will be generated automatically. Such visualizations provide intuitive insights into system-wide power flow. In some cases, the system size is relatively large, and the visualization may become cluttered due to the excessive amount of information displayed. Fortunately, the plotting tools used in each programming languages are interactive and support zooming, allowing users to focus on specific regions of interest. Of course, if the user is not interested in visualization, setting `'plotResult'` to `false` will disable plotting outputs.

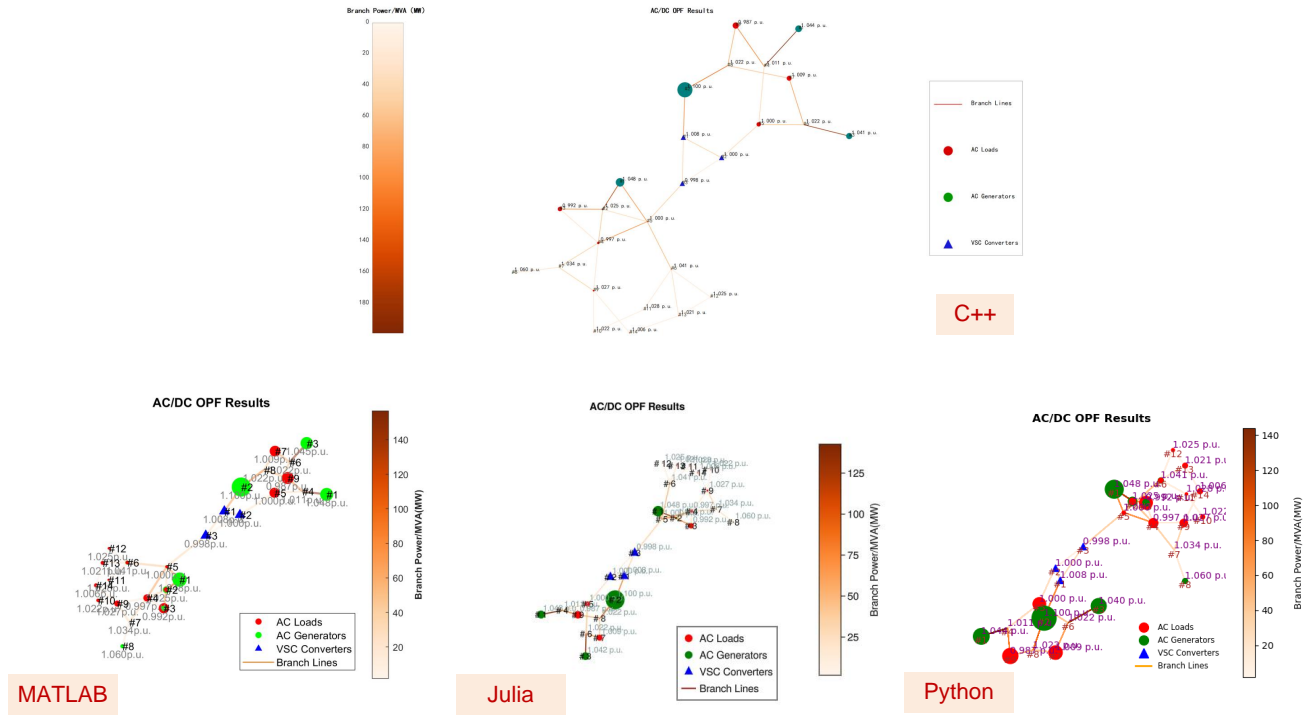


Figure 4.2: Visualization of AC/DC OPF results across programming languages. The size of ● is proportional to the AC load level. The size of ● is proportional to the AC generator power level. The colour intensity of — indicates the branch power level.

## 4.2 Benchmark

### 4.2.1 Performance comparison

To compare programming languages, benchmark scripts were developed. The test cases consistently used the same three-terminal MTDC grid case `'mtdc3slack'` a combined with four representative interconnected AC grid cases from small to large scales that are: `'ac9ac14'`, `'ac14ac57'`, `'ac57ac118'`, and `'ac118ac300'`. For all experiments, each was executed five times per test case

and with three warmup trial runs to avoid the effect of just-in-time (JIT) compilation and delayed code execution due to memory caching.

Execution time and memory allocation were stored for each execution, and a post-execution process was carried out to compute median values from the results. In the case of [Julia](#), run time was recorded using the `@elapsed` macro provided in the `BenchmarkTools.jl` package, while memory allocations were tracked at the systems level by observing process monitoring. [Julia](#) utilized a separate, although framework-based dictionary, dependent on grid modeling but also leveraged modular, very high-performance versions of the algorithms. For [Python](#) runtime, `time.time()` was used and memory by using the `psutil` library to return the resident set size (RSS). The [Python](#) benchmark script was made to accommodate missing input files, should there be a case of non-completion.

[C++](#) performance benchmarks used the high-resolution `std::chrono` clock for timing, while memory measurements came from `GetProcessMemoryInfo()` on Windows. [C++](#) was also in an object-oriented context with custom memory stuff to reduce dynamic allocations. [Matlab](#) used `tic` and `toc` to measure execution time and had a custom `et_memory_usage()` method to track memory and invoke garbage collection every run. To make sure memory use was consistent every run, the benchmark had to invoke a pause for a few seconds to get consistent memory states. All implementations run in the same benchmarking environment to give a fair and consistent comparison across languages. The statistics collected, therefore, included median run time, run time variation (min-max), peak memory usage, and relative memory use normalized against the smallest test case as the baseline for each language.

The benchmarking results are summarized in Table 16. For each case, we report the median execution time, run-time range, peak memory usage, and relative memory usage normalized to the smallest test case within each language.

## 4.2.2 Discussion

The results indicate striking variations in execution time and memory consumption between implementations.

### Runtime:

- Across the benchmarked cases, [Julia](#) consistently had the fastest execution times. For example, in the largest case, `'ac118ac300'`, [Julia](#) was the only language to return a solution in seconds (4.446 s), outperforming [C++](#) (38.431 s), [Python](#) (35.976 s), and especially [Matlab](#) (not computed successfully within an hour). [Julia](#)'s position was also aided by the use of JIT compilation, type stability, and fast numerical computing libraries that are heavily optimized.
- [C++](#) had particularly strong and consistent performance, as it was often ranked second in execution time to [Julia](#) with sometimes very small margins. For example, [C++](#) solved `'ac14ac57'` in 1.780 s and `'ac57ac118'` in 3.464 s, indicating an ability to scale with the larger case. While [C++](#) may not be capable of the peak performance that [Julia](#) demonstrated, it still represented a large improvement over [Matlab](#) and [Python](#), in particular, the larger cases.
- [Python](#) had more mixed performance. For some of the smaller cases like `'ac9ac14'`, it performed reasonably well (0.138 s, similar to [C++](#)) but for larger problems like `'ac57ac118'`,

Table 16: Benchmark Results Summary Across Programming Languages

Case	Language	Time (s)	Time Range (s)	Memory (MB)	Rel. Mem.
ac9ac14	Matlab	7.673	7.611–7.732	0.1	1.0x
	Cpp	0.139	0.133–0.142	0.0	0x
	Julia	0.055	0.052–0.072	0.1	1.0x
	Python	0.138	0.127–0.182	1.7	17.0x
ac14ac57	Matlab	157.353	154.715–166.027	5.3	0.14x
	Cpp	1.780	1.478–2.365	0.5	0.013x
	Julia	0.233	0.202–0.940	36.8	1.0x
	Python	1.014	0.997–1.054	2.4	0.07x
ac57ac118	Matlab	177.273	92.259–245.281	14.9	0.12x
	Cpp	3.464	3.299–4.056	0.8	0.006x
	Julia	0.799	0.783–0.821	129.4	1.0x
	Python	5.063	4.977–5.240	2.6	0.02x
ac118ac300	Matlab	-	-	-	-
	Cpp	38.431	37.126–38.924	0.5	0x
	Julia	4.446	4.353–5.343	537.7	1.0x
	Python	35.976	28.344–96.248	1.9	0.003x

★ The case ac118ac300 is not computed successfully in [MATLAB](#)

its performance suffered 5.063 s but still much better than [Matlab](#)). Overall, this suggests that [Python](#) could return competitive performance when appropriately optimized but also suffers from the overheads of being interpreted and dynamically typed languages.

- [Matlab](#) consistently had the slowest execution times, taking 7.673 s for 'ac9ac14' and up to 177.273 s for 'ac57ac118'. Although [Matlab](#) offers a high-level and user-friendly interface for numerical computing, it is not optimized for large-scale, real-time OPF problems, compared to compiled or JIT-compiled languages.

### Memory Usage:

- [Cpp](#) had the lowest memory use in the benchmarks, achieving 0.0MB on 'ac9ac14' and only 0.8 MB on 'ac57ac118'. [Cpp](#)'s efficiency in memory usage can be attributed to its manual control and low overhead, making it a great option for embedded or constrained environments.
- [Matlab](#) also achieved low memory usage of 0.1 MB or small cases, and up to 14.9 MB for 'ac57ac118'. These values show that, while executing slowly, [Matlab](#) uses in-place operations and other efficient numerical operations to limit memory usage.
- [Python](#) had reasonable and moderate memory usage, between 1.7 MB on 'ac9ac14' and 2.6 MB on 'ac57ac118'. However, for the small case calls, its relative memory use was at the high end due likely to either run-time overhead from [Python](#) itself or memory overhead to include references to external libraries (i.e. [numpy](#), or, [pandas](#) etc.).

- In the larger cases, [Julia](#) consumed the most memory during execution. For example, the largest call ('ac57ac118') was finished using 129.4 MB, and the second largest ('ac118ac300') finished with 537.7 MB. This likely reflects its design choice to have more memory used to enable faster execution speed and array "fusion". This is generally fine in high-performance computing, as long as the overall needs of memory consumption remain low and adequate overall.

### Scalability:

- The scalability of [Julia](#) was noted to be very impressive since execution times increased sub-linearly, from 0.055 s for 'ac9ac14' to 0.799 s for 'ac57ac118', and 4.446 s for 'ac118ac300', the largest case. This shows that [Julia](#) is very robust and is quite appropriate for large-scale AC/DC OPF problems or even for real-time applications.
- The scalability of [Cpp](#) was also very strong. Times grew from 0.139 s for 'ac9ac14' to 3.464 s for 'ac57ac118', and 38.431 s for 'ac118ac300'. This projection is enhanced because of the compile-time optimizations and low-level control [Cpp](#) enables to increase speed. However, with this comes a higher level of complexity to develop.
- The scalability for [Python](#) was reasonable. However, there was too much variability in run time as the size of the system increased—the largest case, 'ac57ac118', had running times range from 28.344 s to 96.248 s, which influenced background overhead and behaviour of the number of libraries.
- Beyond time, [Matlab](#) did not scale significantly. It was not able to complete the largest case 'ac118ac300', and the response time for the mid-size cases 'ac14ac57' (157.353 s) and 'ac57ac118' (177.273 s) had large increases in execution time. This demonstrates that [Matlab](#) is not suitable for large-scale optimization still making it a better alternative in prototyping scenarios and for educational purposes.

## References

- [1] J. Beerten, S. Cole, and R. Belmans, “Generalized steady-state vsc mtdc model for sequential ac/dc power flow algorithms,” *IEEE Transactions on Power Systems*, vol. 27, no. 2, pp. 821–829, 2012.
- [2] H. Ergun, J. Dave, D. Van Hertem, and F. Geth, “Optimal power flow for ac–dc grids: Formulation, convex relaxation, linear approximation, and implementation,” *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 2980–2990, 2019.
- [3] D. Van Hertem, O. Gomis-Bellmunt, and J. Liang, *HVDC grids: for offshore and supergrid of the future*. John Wiley & Sons, 2016.
- [4] B. Kocuk, S. S. Dey, and X. A. Sun, “Strong socp relaxations for the optimal power flow problem,” *Operations Research*, vol. 64, no. 6, pp. 1177–1196, 2016.
- [5] L. Gan and S. H. Low, “Optimal power flow in direct current networks,” in *52nd IEEE Conference on Decision and Control*, 2013, pp. 5614–5619.