PNNL-31126

ExaGO™ Users Manual

Version 1.5.1

Contents

Chapter 1

Introduction

The Exascale Grid Optimization (ExaGO^{TM}) toolkit is an open source package for solving large-scale power grid optimization problems on parallel and distributed architectures, particularly targeted for exascale machines with heteregenous architectures (GPU). ExaGO^{TM} is written in C/C++ and makes heavy use of the functionality provided by the PETSc[1] library. It uses RAJA [3]and Umpire [4]libraries for execution on the GPU and can make use of several optimization solvers - Ipopt [7], HiOp [6, 5]- for solving the optimization problems.

All ExaGO^{TM} applications use a nonlinear formulation based on full AC optimal power flow. The different applications available with `ExaGO` are listed in Table 1.1

Table 1.1: `ExaGO` applications

Application	**Description**	**Notes**
OPFLOW	AC optimal power flow	
SCOPFLOW	Security-constrained AC optimal power flow	Uses TCOPFLOW for multi-period contingencies
TCOPFLOW	Multi-period AC optimal power flow	
SOPFLOW	Stochastic AC optimal power flow	Uses SCOPFLOW for multi-contingency scenarios

While `ExaGO` is targeted for making use of distributed computing environments and GPUs, its full support is still under development. Table 1.2 lists the architecure support for ExaGO applications.

Table 1.2: `ExaGO` application execution on different hardware

Application	**CPU (serial)**	**CPU (parallel)**	**GPU**
OPFLOW	Y	N	Y
SCOPFLOW	Y	Y	Y
SOPFLOW	Y	Y	Y
TCOPFLOW	Y	N	N

Together, these applications cater to problems spanning the dimensions of security (contingencies), stochasticity, and time as shown in Figure 1.1.

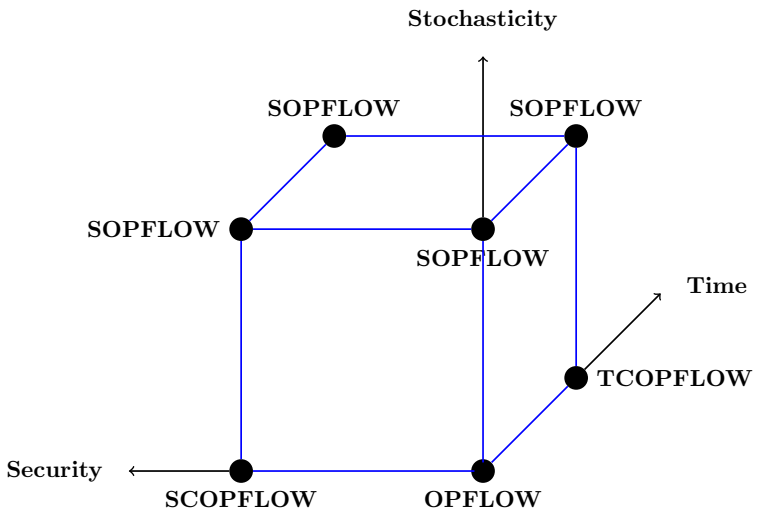

Figure 1.1: ExaGO provides applications along the dimensions of security (contingencies), time, and stochasticity. The label vertices denote different ExaGO applications available.

Chapter 2

Getting Started

2.1 System requirements

ExaGO is currently only built on 64b OSX and Linux machines, compiled with GCC >= 7.3. We build ExaGO on Intel, AMD, and IBM Power9 architectures.

2.2 Prerequisites

This section assumes that you already have the ExaGO source code, and that the environment variable EXAGODIR is the directory of the ExaGO source code. ExaGO may be acquired via the PNNL git repository linked here, like so:

```
> git clone https://gitlab.pnnl.gov/exasgd/frameworks/exago.git exago
> export EXAGODIR=$PWD/exago
```

Paths to installations of third party software in examples are abbreviated with placeholder paths. For example, /path/to/cuda is a placeholder for a path to a valid CUDA Toolkit installation.

2.3 Dependencies

ExaGO at version 1.5.1 has the dependencies listed in table 2.1. The versions of dependencies listed are those we have tested with, though newer version may also be compatible.

Table 2.1: Dependency Table

Dependency	Version Constraints	Mandatory	Notes
PETSc [1]	3.16.x	✓	Core dependency
CMake	3.18	✓	Only a build dependency
MPI	4.1		Only tested with openmpi and spectrummpi
Ipopt [7]	3.12		
HiOp [6, 5]	0.7.1		Prefer dynamically linked
RAJA [3]	0.14.0		
Umpire [2]	6.0.0		Only when RAJA is enabled
MAGMA	2.6.1		Only when GPU acceleration is enabled
CUDA Toolkit	11.4		Only when GPU acceleration is enabled

These may all be toggled via CMake, which will be discussed in the section Building and installation.

2.3.1 Notes on environment modules

Many of the dependencies are available via environment modules on institutional clusters. To get additional information on your institution's clusters, please ask your institution's system administrators. Some end-to-end examples in this document will use system-specific modules and are not expected to expected to run on other clusters.

For example, the modules needed to build and run ExaGO on Newell, an IBM Power9 PNNL cluster, are as follows:

```
> module load gcc/8.5.0
> module load openmpi/4.1.4
> module load cuda/11.4
> module load python/miniconda3.8
> module load cmake/3.19.6
```

2.3.2 Additional Notes on GPU Accelerators

CUDA is currently the only GPU accelerator platform ExaGO fully supports.

2.3.3 Additional Notes on Umpire

Umpire is an implicit dependency of RAJA. If a user enables RAJA, they must also provide a valid installation of Umpire. Additionally, if a user would like to run ExaGO with RAJA and without CUDA, they must provide a CPU-only build of Umpire since an Umpire build with CUDA enabled will link against CUDA.

2.4 Building and installation

2.4.1 Default Build

ExaGO may be built with a standard CMake workflow:

Listing 2.1: Example CMake workflow

```
> cd $EXAGODIR
> export BUILDDIR=$PWD/build INSTALLDIR=$PWD/install
> mkdir $BUILDDIR $INSTALLDIR
> cd $BUILDDIR
> cmake .. -DCMAKE_INSTALL_PREFIX=$INSTALLDIR
> make install
```

The following sections will assume the user is following the basic workflow outlined above.

Note: For changes to the CMake configuration to take effect, the code will have to be reconfigured using cmake and rebuilt using make.

2.4.2 Additional Options

To enable additional options, CMake variables may be defined via CMake command line arguments, ccmake, or cmake-gui. CMake options specific to ExaGO have an EXAGO_ prefix. For example, the following shell commands will build ExaGO with MPI:

```
> cmake .. -DCMAKE_INSTALL_PREFIX=$INSTALLDIR -DEXAGO_ENABLE_MPI=ON
```

ExaGO's CMake configuration will search the usual system locations for an MPI installation.

For dependencies not installed to a system-wide location, users may also directly specify the location of a dependency. For example, this will build ExaGO with IPOPT enabled and installed to a user directory:

```
> cmake .. \
    -DCMAKE_INSTALL_PREFIX=$INSTALLDIR \
    -DEXAGO_ENABLE_IPOPT=ON \
    -DIPOPT_DIR=/path/to/ipopt
```

Notice that the CMake variable IPOPT_DIR does not have an EXAGO_ prefix. This is because the variables specifying locations often belong to external CMake modules. CMake variables indicating installation directories do not have an EXAGO_ prefix.

Some CMake options effect others. This is especially common when the user enables ExaGO's GPU options. For example, if the user enables EXAGO_ENABLE_GPU and EXAGO_ENABLE_RAJA, the user must provide a GPU-enabled RAJA installation. Umpire is also an implicit dependency of RAJA, so if the user enables EXAGO_ENABLE_GPU they must **also** provide a GPU-enabled Umpire installation.

Below is a complete shell session on PNNL's cluster Newell in which a more complicated ExaGO configuration is built, where each dependency installation is explicitly passed to CMake. Environment modules specific to Newell are provided to make the example thorough, even though they are not likely to work on another machine. However, similar modules (with different version numbers) are likely to be available on other platforms.

Listing 2.2: ExaGO build with all options enabled

```
> module load gcc/8.5.0
> module load openmpi/4.1.4
> module load cuda/11.4
> module load python/miniconda3.8
> module load cmake/3.19.6
> git clone https://gitlab.pnnl.gov/exasgd/frameworks/exago.git exago
> export EXAGODIR=$PWD/exago
> cd $EXAGODIR
> export BUILDDIR=$PWD/build INSTALLDIR=$PWD/install
> mkdir $BUILDDIR $INSTALLDIR
> cd $BUILDDIR
> cmake .. \
  -DCMAKE_INSTALL_PREFIX=$INSTALLDIR \
  -DCMAKE_BUILD_TYPE=Debug \
  -DEXAGO_ENABLE_GPU=ON \
  -DEXAGO_ENABLE_HIOP=ON \
  -DEXAGO_ENABLE_IPOPT=ON \
```

```
  -DEXAGO_ENABLE_MPI=ON \
  -DEXAGO_ENABLE_PETSC=ON \
  -DEXAGO_RUN_TESTS=ON \
  -DEXAGO_ENABLE_RAJA=ON \
  -DEXAGO_ENABLE_IPOPT=ON \
  -DIPOPT_DIR=/path/to/ipopt \
  -DRAJA_DIR=/path/to/raja \
  -Dumpire_DIR=/path/to/umpire \
  -DHIOP_DIR=/path/to/hiop \
  -DMAGMA_DIR=/path/to/magma \
  -DPETSC_DIR=/path/to/petsc
> make -j 8 install
> # For the following commands, a job scheduler command may be needed.
> # Run test suite
> make test
> # Run an ExaGO application:
> $INSTALLDIR/bin/opflow
```

2.5 Usage

Each ExaGO application has the following format for execution

```
  ./app <app_options>
```

For OPFLOW for example:

```
./opflow -help
ExaGO 1.4.1 built on Jan 14 2022

Arguments for opflow:
        -help
                Print help message (type: flag)

        -version
                Print version information (type: flag)

        -config
                Print configuration options used to build ExaGO (type:
                   ↪ flag)

        -options_file /path/to/options_file
                Path to options file used to load additional ExaGO
                   ↪ configuration options (type: string)

        -opflow_model (POWER_BALANCE_POLAR|POWER_BALANCE_HIOP|
           ↪ PBPOLRAJAHIOP)
                OPFLOW model name (type: string)
```

```
-opflow_solver (IPOPT|HIOP|HIOPSPARSE)
        OPFLOW solver type (type: string)

-opflow_initialization (OPFLOWINIT_MIDPOINT|OPFLOWINIT_FROMFILE|
   ↪ OPFLOWINIT_ACPF|OPFLOWINIT_FLATSTART)
        Type of OPFLOW initialization (type: string)

-opflow_objective (MIN_GEN_COST|MIN_GENSETPOINT_DEVIATION|NO_OBJ)
        Type of OPFLOW objective (type: string)

-opflow_genbusvoltage (VARIABLE_WITHIN_BOUNDS|FIXED_WITHIN_QBOUNDS
   ↪ |FIXED_AT_SETPOINT)
        Type of OPFLOW gen bus voltage control (type: string)

-opflow_has_gensetpoint 0
        Use set-points for generator real power (type: bool)

-opflow_use_agc 0
        Use automatic generation control (AGC) (type: bool)

-opflow_tolerance 1e-06
        Optimization tolerance (type: real)

-opflow_ignore_lineflow_constraints 0
        Ignore line flow constraints? (type: bool)

-opflow_include_loadloss_variables 0
        Ignore line flow constraints? (type: bool)

-opflow_loadloss_penalty 1000
        Penalty for load loss (type: real)

-opflow_include_powerimbalance_variables 0
        Allow power imbalance? (type: bool)

-opflow_powerimbalance_penalty 10000
        Power imbalance penalty (type: real)

-hiop_compute_mode (auto|cpu|hybrid|gpu)
        Set compute mode for HiOp solver (type: string)

-hiop_verbosity_level 0
        Set verbosity level for HiOp solver, between 0 and 12 (
           ↪ type: int)

-hiop_ipopt_debug 0
        Flag enabling debugging HIOP code with IPOPT (type: bool)
```

ExaGO applications also have an option to print many options ExaGO was configured with, which aid in reproducing build environments that a given ExaGO source tree was built with.

```
./opflow -config
ExaGO version 1.3.0 built on Jan 14 2022
Built with the following command:
$ cmake -B /Users/manc568/workspace/exago/build -S /Users/manc568/
   ↪ workspace/exago \
        -DEXAGO_ENABLE_RAJA:BOOL=OFF \
        -DEXAGO_ENABLE_PYTHON:BOOL=ON \
        -DEXAGO_ENABLE_HIP:BOOL=OFF \
        -DEXAGO_ENABLE_GPU:BOOL=OFF \
        -DEXAGO_ENABLE_HIOP_SPARSE:BOOL=ON \
        -DEXAGO_ENABLE_HIOP:BOOL=ON \
        -DEXAGO_ENABLE_IPOPT:BOOL=ON \
        -DEXAGO_ENABLE_MPI:BOOL=ON \
        -DEXAGO_BUILD_STATIC:BOOL=OFF \
        -DEXAGO_ENABLE_CUDA:BOOL=OFF \
        -DEXAGO_ENABLE_HIOP_DISTRIBUTED:BOOL=OFF \
        -DEXAGO_ENABLE_PETSC:BOOL=ON \
        -DEXAGO_BUILD_SHARED:BOOL=OFF \
        -DIPOPT_INCLUDES:STRING="" \
        -DIPOPT_ROOT_DIR:STRING="" \
        -DHiOp_DIR:STRING="/some/path" \
        -DPETSC_DIR:STRING="/some/path" \
        -DCMAKE_CXX_COMPILER:STRING="/usr/local/bin/g++-11" \
        -DIPOPT_LIBRARIES:STRING="" \
        -DCMAKE_BUILD_TYPE:STRING="Debug" \
        -DPETSC_INCLUDES:STRING="" \
        -DCMAKE_C_COMPILER:STRING="/usr/local/bin/gcc-11" \
        -DCMAKE_INSTALL_PREFIX:STRING="/Users/manc568/workspace/exago/
           ↪ install" \
        -DPETSC_LIBRARIES:STRING="" \
        -DEXAGO_OPTIONS_DIR:STRING="/Users/manc568/workspace/exago/install
           ↪ /share/exago/options"
```

Here, `app_options` are the command line options for the application. Each application has many options through which the input files and the control options can be set for the application. All application options have the form `-app_option_name` followed by the `app_option_value`. For instance,

```
./opflow -netfile case9mod.m -opflow_model POWER_BALANCE_POLAR \
-opflow_solver IPOPT
```

will execute the OPFLOW application using `case9mod.m` input file with the model `POWER_BALANCE_POLAR` and Ipopt [7]solver.

Options can also be passed to each application through an options file `-optionsfile <optionfilename` ↪ , or via command line or through a combination of the command line and options file. The option

specified via command line supersedes that given in in the options file. For example, if `-options_file opflowoptions` specified `-netfile case9mod.m` within its settings:

```
./opflow -options_file opflowoptions # Uses case9mod.m
./opflow -netfile case118.m -options_file opflowoptions # Uses case118.m
```

Chapter 3

Optimal power flow (OPFLOW)

OPFLOW solves the full AC optimal power flow problem and provides various flexible features that can be toggled via run-time options. It has interfaces to different optimization solvers that can be executed on CPUs or on GPUs.

3.1 Formulation

Optimal power flow is a general nonlinear programming problem with the following form

$$\begin{aligned} \text{min. } & f(x) && (3.1)\\ & \text{s.t.} \\ & g(x) = 0 && (3.2)\\ & h(x) \leq 0 && (3.3)\\ & x^{\min} \leq x \leq x^{\max} && (3.4) \end{aligned}$$

Here, x are the decision variables with lower and upper bounds $x^{\min}$ and $x^{\max}$, respectively, $f(x)$ is the objective function, $g(x)$ and $h(x)$ are the equality and inequality constraints, respectively. In the following sections we describe what constitutes these different terms as used by OPFLOW.

3.1.1 Variables and bounds

The different variables used in OPFLOW formulation are described in Table 3.1.

Power imbalance variables are non-physical (slack) variables that measure the violation of power balance at buses. Having these variables (may) help in making the optimization problem easier to solve since they always ensure feasibility of the bus power balance constraints.

3.1.2 Objective Function

The objective function for OPFLOW is given in (3.5)

$$\text{min. } C_{gen}(p^{\text{g}}) + C_{dev}(\Delta p^{\text{g}}) + C_{loss}(\Delta p^{\text{l}}, \Delta q^{\text{l}}) + C_{imb}(\Delta p^{+}, \Delta p^{-}, \Delta q^{+}, \Delta q^{-}) \quad (3.5)$$

Table 3.1: Optimal power flow (OPFLOW) variables

Symbol	Variable	Bounds	Notes
p_j^{g}	Generator real power dispatch	$p_j^{\mathrm{gmin}} \leq p_j^{\mathrm{g}} \leq p_j^{\mathrm{gmax}}$	
q_j^{g}	Generator reactive power dispatch	$q_j^{\mathrm{gmin}} \leq q_j^{\mathrm{g}} \leq q_j^{\mathrm{gmax}}$	
Δp_j^{g}	Generator real power deviation	$-p_j^{\mathrm{r}} \leq \Delta p_j^{\mathrm{g}} \leq p_j^{\mathrm{r}}$	• Only used when `-opflow_has_gensetpoint` or `-opflow_use_agc` option is active • Δp_j^{g} is the deviation from real power generation set-point p_j^{gset}.
p_j^{gset}	Generator real power set-point	$p_j^{\mathrm{gmin}} \leq p_j^{\mathrm{gset}} \leq p_j^{\mathrm{gmax}}$	• Only used when `-opflow_has_gensetpoint` or `-opflow_use_agc` option is active.
ΔP	System power excess/deficit	Unbounded	Only used when `-opflow_use_agc` is active
θ_i	Bus voltage angle	$\text{-}\pi \leq \theta_i \leq \pi$	• Used with power balance polar model (`-opflow_model POWER_BALANCE_POLAR`) • θ_i is unbounded, except reference bus angle θ_i^{ref} which is fixed to 0
v_i	Bus voltage magnitude	$v_i^{\mathrm{min}} \leq v_i \leq v_i^{\mathrm{max}}$	• Used with power balance polar model (`-opflow_model POWER_BALANCE_POLAR`) • $v_i^{\mathrm{min}} = v_i^{\mathrm{max}} = v_i^{\mathrm{set}}$ if fixed generator set point voltage option is active (`-opflow_has_gensetpoint`)
$\Delta p_i^+, \Delta p_i^-$	Bus real power mismatch variables	$0 \leq \Delta p_i^+, \Delta p_i^- \leq \infty$	Used when power mismatch variable option is active (`-opflow_include_powerimbalance_variables1`)
$\Delta q_i^+, \Delta q_i^-$	Bus reactive power mismatch variables	$0 \leq \Delta q_i^+, \Delta q_i^- \leq \infty$	Used when power mismatch variable option is active (`-opflow_include_powerimbalance_variables1`)
Δp_j^{l}	Real power load loss	$0 \leq \Delta p_j^{\mathrm{l}} \leq p_j^{\mathrm{l}}$	Used when load loss variable option is active (`-opflow_include_loadloss_variables1`)
Δq_j^{l}	Reactive power load loss	$0 \leq \Delta q_j^{\mathrm{l}} \leq q_j^{\mathrm{l}}$	Used when load loss variable option is active (`-opflow_include_loadloss_variables1`)

Total generation cost $C_{gen}(p^{\mathrm{g}})$

Needs option `-opflow_objective MIN_GEN_COST`

$$C_{gen}(p^{\mathrm{g}}) = \sum_{j \in J^{\mathrm{gen}}} C_j^{\mathrm{g}}(p_j^{\mathrm{g}}) \tag{3.6}$$

Here, C_j^{g} is a quadratic function of the form $C_j^{\mathrm{g}} = a_j^{\mathrm{g}} p_j^{\mathrm{g}^2} + b_j^{\mathrm{g}} p_j^{\mathrm{g}} + c_j^{\mathrm{g}}$.

Total generation setpoint deviation $C(\Delta p^{\mathrm{g}})$

Needs option `-opflow_objective MIN_GENSETPOINT_DEVIATION`

$$C_{dev}(\Delta p^{\mathrm{g}}) = \sum_{j \in J^{\mathrm{gen}}} (\Delta p_j^{\mathrm{g}^2}) \tag{3.7}$$

This feature is only supported with IPOPT solver.

Load loss $C(\Delta p^{\mathrm{l}}, \Delta q^{\mathrm{l}})$

This term gets added to the objective when `-opflow_include_loadloss_variables` option is active.

$$C_{loss}(\Delta p^{\mathrm{l}}, \Delta q^{\mathrm{l}}) = \sum_{j \in J^{\mathrm{ld}}} \sigma_j^{\mathrm{l}} (\Delta p_j^{\mathrm{l}} + \Delta q_j^{\mathrm{l}}) \tag{3.8}$$

The load loss penalty σ_j^{l} can be set via the option `-opflow_loadloss_penalty`. The default is \$1000/MW for all loads.

Power imbalance $C_{imb}(\Delta p^+, \Delta p^-, \Delta q^+, \Delta q^-)$

This term gets added to the objective when `-opflow_include_powerimbalance_variables` option is active.

$$C_{imb}(\Delta p^+, \Delta p^-, \Delta q^+, \Delta q^-) = \sum_{i \in J^{bus}} \sigma_i (\Delta p^+ + \Delta p^- + \Delta q^+ + \Delta q^-) \tag{3.9}$$

The power imbalance cost σ_i can be set via the option `-opflow_powerimbalance_penalty`. The default is \$10,000/MW^2 for all buses. Though the power imbalance variables $\Delta p^+, \Delta p^-, \Delta q^+, \Delta q^-$ are slack or non-physical, they can help in solving infeasible cases having no power flow solution, and thus provide a measure of the infeasibility.

3.1.3 Equality constraints

Nodal power balance

The nodal power balance equations for each bus i are given by

$$\sum_{\substack{j\in J^{\text{gen}}\\ A^{\text{g}}_{ij}\neq 0}} p^{\text{g}}_j = p^{\text{sh}}_i + \Delta p^{+}_i - \Delta p^{-}_i + \sum_{\substack{j\in J^{\text{ld}}\\ A^{\text{l}}_{ij}\neq 0}} (p^{\text{l}}_j - \Delta p^{\text{l}}_j) + \sum_{\substack{j\in J^{\text{br}}\\ A^{\text{br}}_{oi}\neq 0}} p^{\text{br}}_{jod} + \sum_{\substack{j\in J^{\text{br}}\\ A^{\text{br}}_{id}\neq 0}} p^{\text{br}}_{jdo} \tag{3.10}$$

$$\sum_{\substack{j\in J^{\text{gen}}\\ A^{\text{g}}_{ij}\neq 0}} q^{\text{g}}_j = q^{\text{sh}}_i + \Delta q^{+}_i - \Delta q^{-}_i + \sum_{\substack{j\in J^{\text{ld}}\\ A^{\text{l}}_{ij}\neq 0}} (q^{\text{l}}_j - \Delta q^{\text{l}}_j) + \sum_{\substack{j\in J^{\text{br}}\\ A^{\text{br}}_{oi}\neq 0}} q^{\text{br}}_{jod} + \sum_{\substack{j\in J^{\text{br}}\\ A^{\text{br}}_{id}\neq 0}} q^{\text{br}}_{jdo} \tag{3.11}$$

(3.12)

where, the real and reactive power shunt consumption is given by (3.13) and (3.14)

The real and reactive power flow p^{br}_{jod}, q^{br}_{jod} for line j from the origin bus o to destination bus d is given by (3.19) – (3.20) and from destination bus d to origin bus o is given by (3.21) – (3.22)

Shunt power

$$p^{\text{sh}}_i = g^{\text{sh}}_i {v_i}^2 \tag{3.13}$$

$$q^{\text{sh}}_i = -b^{\text{sh}}_i {v_i}^2 \tag{3.14}$$

Generator real power output

When using `-opflow_has_gensetpoint`, two extra variables p^{gset}_j and Δp^{g}_j are added for each generator. The generator real power output p^{g}_j is related to the power deviation Δp^{g}_j by the following relations

$$p^{\text{gset}}_j + \Delta p^{\text{g}}_j - p^{\text{g}}_j = 0 \tag{3.15}$$

$$p^{\text{gset}}_j - p^{\text{g}*}_j = 0 \tag{3.16}$$

The second equation sets the generator set-point p^{gset}_j to a fixed value $p^{\text{g}*}_j$. Here, $p^{\text{g}*}_j$ is the set-point for the generator real power output, which can be thought of as an operator set or contractual agreement set-point.

3.1.4 Inequality constraints

MVA flow on branches

MVA flow limits at origin and destination buses for each line.

$${p^{\text{br}}_{jod}}^2 + {q^{\text{br}}_{jod}}^2 \leq {s^{\text{rateA}}_j}^2, \quad j \in J^{\text{br}} \tag{3.17}$$

$${p^{\text{br}}_{jdo}}^2 + {q^{\text{br}}_{jdo}}^2 \leq {s^{\text{rateA}}_j}^2, \quad j \in J^{\text{br}} \tag{3.18}$$

To reduce the number of inequality constraints, only lines that are in service and having MVA A rating s^{rateA}_j less than 10000 MVA are considered.

Branch flows

In polar coordinates, the real and reactive power flow $p^{\text{br}}_{j_{od}}$, $q^{\text{br}}_{j_{od}}$ from bus o to bus d on line j is given by (3.19) – (3.20)

$$p^{\text{br}}_{j_{od}} = g_{oo}v_o^2 + v_o v_d(g_{od}\cos(\theta_o - \theta_d) + b_{od}\sin(\theta_o - \theta_d)) \tag{3.19}$$
$$q^{\text{br}}_{j_{od}} = -b_{oo}v_o^2 + v_o v_d(-b_{od}\cos(\theta_o - \theta_d) + g_{od}\sin(\theta_o - \theta_d)) \tag{3.20}$$

and from bus d to bus o is given by (3.21) – (3.22)

$$p^{\text{br}}_{j_{do}} = g_{dd}v_d^2 + v_d v_o(g_{do}\cos(\theta_d - \theta_o) + b_{do}\sin(\theta_d - \theta_o)) \tag{3.21}$$
$$q^{\text{br}}_{j_{do}} = -b_{dd}v_d^2 + v_d v_o(-b_{do}\cos(\theta_d - \theta_o) + g_{do}\sin(\theta_d - \theta_o)) \tag{3.22}$$

Automatic generation control (AGC)

With `-opflow_use_agc`, two additional constraints are added for each participating generator to enforce the proportional generator redispatch participation as done in automatic generation control (AGC). These two equations are

$$\begin{aligned}\left(\alpha^{\text{g}}_j \Delta P - \Delta p^{\text{g}}_j\right)\left(p^{\text{g}}_j - p^{\text{gmax}}_j\right) \geq 0\\ \left(\Delta p^{\text{g}}_j - \alpha^{\text{g}}_j \Delta P\right)\left(p^{\text{gmin}}_j - p^{\text{g}}_j\right) \geq 0\end{aligned} \tag{3.23}$$

Eq. 3.23 forces the generator set-point deviation to be equal to the generation participation when the generator has head-room available $p^{\text{gmin}}_j \leq p^{\text{g}}_j \leq p^{\text{gmax}}_j$. Here, α^{g}_j is the generator participation factor which is the proportion of the power deficit/excess ΔP that the generator provides.

Generator bus voltage control

When the option `-opflow_genbusvoltage FIXED_WITHIN_QBOUNDS` is used, the generator bus voltage is fixed when the total reactive power generation available at the bus is within bounds. When it reaches its bounds, the voltage varies with the generator reactive power fixed at its bound. To implement this behavior, two inequality constraints are added for each generator bus

$$\begin{aligned}(v^{\text{set}}_i - v_i)(q_i - q^{\text{max}_i}) \geq 0\\ (v_i - v^{\text{set}}_i)(q^{\text{min}_i} - q_i) \geq 0\end{aligned} \tag{3.24}$$

Here, q_i, q^{max_i}, and q^{min_i} are the generated, maximum, and minimum reactive power at the bus, respectively.

3.2 Solvers

OPFLOW can be used with a few different solvers. All the solvers solve the optimization problem via a nonlinear interior-point algorithm.

1. Ipopt [7] is a popular open-source package for solving nonlinear optimization problems. It is the most robust of the solvers implemented for solving OPFLOW. However, it can be run only on a single process and does not have GPU support.
   Option: `-opflow_solver IPOPT -opflow_model POWER_BALANCE_POLAR`

2. HiOp [6, 5] is a high-performance optimization library that implements an interior-point algorithm for solving nonlinear optimization problems. There are two solvers available from the HiOp [6, 5]library: Mixed sparse-dense formulation `-opflow_solver HIOP`, and sparse formulation `-opflow_solver HIOPSPARSE`. The library supports execution both on the CPU and the GPU. Options:
   CPU: `-opflow_solver HIOP -opflow_model POWER_BALANCE_HIOP -hiop_compute_mode CPU`
   GPU: `-opflow_solver HIOP -opflow_model PBPOLRAJAHIOP -hiop_compute_mode GPU`

3.3 Models

A 'model' in ExaGO describes the representation of the underlying physics. All OPFLOW models use the power balance formulation in polar coordinates for the ACOPF equations. The difference between the different models arises from their specific implementation/interface. The different models available for OPFLOW are listed in Table 3.2. As discussed earlier, not every 'model' is compatible with every 'solver'. Table 3.3 lists the solver compatibility for the different models.

Table 3.2: OPFLOW models

Model type	**OPFLOW option (`-opflow_model`)**	**Compatible solvers**	**CPU-GPU**
Power balance with polar coordinates	POWER_BALANCE_POLAR	IPOPT, HIOPSPARSE	CPU
Power balance with polar coordinates used with HIOP	POWER_BALANCE_HIOP	HiOp [6, 5]	CPU/GPU
Power balance with polar coordinates used with HIOP on GPU	PBPOLRAJAHIOP	HiOp [6, 5]	GPU

Table 3.3: OPFLOW Model-solver compatibility

Model Name	Ipopt [7]	HiOp [6, 5]	HIOPSPARSE
POWER_BALANCE_POLAR	✓		✓
POWER_BALANCE_HIOP		✓	
PBPOLRAJAHIOP		✓	

3.3.1 Power balance polar

The power balance polar model (-opflow_model POWER_BALANCE_POLAR) uses the power balance formulation with polar representation for the network voltages. It runs on CPU only and is compatible with Ipopt [7]and sparse HiOp [6, 5]solvers.

3.3.2 Power balance with HiOp on CPU

This model (-opflow_model POWER_BALANCE_HIOP) implements the power balance formulation with polar coordinates used with HiOp [6, 5]solver only. The model evaluation is done only on the CPU, but the HiOp [6, 5]solver can be executed either on the CPU (`-hiop_compute_mode CPU`) or GPU (`-hiop _compute_mode HYBRID`) by setting the `-hiop_compute_mode` option appropriately.

3.3.3 Power balance with HiOp on GPU

The PBPOLRAJAHIOP model (-opflow_model PBPOLRAJAHIOP) computes all the model and optimization calculations on the GPU. This model uses RAJA [3]and Umpire [4] libraries to run OPFLOW calculations (objective, constraints, etc.) on the GPU.

3.4 Input and Output

The current `ExaGO` version only supports reading network files in MATPOWER format and can (optionally) write the output back in MATPOWER data file format.

3.5 Usage

```
./opflow -netfile <netfilename>  <opflowoptions>
```

3.6 Options

See table 3.4

Table 3.4: OPFLOW options

Option	**Meaning**	**Values (Default value)**	**Compatibility**
-netfile	Network file name	string < 4096 characters (case9mod.m)	
-print_output	Print output to screen	0 or 1 (0)	All solvers
-save_output	Save output to file	0 or 1 (0)	All solvers
-opflow_output_format	Solution file format	See Table 3.5 (MATPOWER)	All solvers
-opflow_model	Representation of network balance equations and bus voltages	See Table 3.3 (POWER_BALANCE_POLAR)	
-opflow_solver	Optimization solver	See section 3.2	
-opflow_initialization	Type of initialization	See Table 3.6 (MIDPOINT)	All solvers
-opflow_has_gensetpoint	Uses generation set point and activates ramping variables	0 or 1 (0)	All models
-opflow_use_agc	Uses AGC formulation in OPF	0 or 1 (0)	POWER_BALANCE_POLAR only
-opflow_objective	type of objective function	See table 3.8 (MIN_GEN_COST)	All models
-opflow_genbusvoltage	Type of generator bus voltage control	See Table 3.7 (VARIABLE_WITHIN_BOUNDS)	POWER_BALANCE_POLAR only
-opflow_ignore_lineflow_constraints	Ignore line flow constraints	0 or 1 (0)	All models
-opflow_monitor_line_kvlevels	Monitor line flows at these KV levels	comma separated list	All models
-opflow_include_loadloss_variables	Include load loss	0 or 1 (0)	All models
-opflow_include_powerimbalance_variables	Include power imbalance	0 or 1 (0)	All models
-opflow_loadloss_penality	$ penalty for load loss	real (1000)	All models
-opflow_powerimbalance_penalty	$ penalty for power imbalance	real (10000)	All models
-opflow_tolerance	Optimization solver tolerance	real (1e-6)	All solvers

Table 3.5: OPFLOW solution output formats

Format name	**Description**
MATPOWER	Matlab format compatible with MATPOWER
CSV	Custom comma separated variable format
JSON	Javascript object notation, used for visualization
MINIMAL	Simple text file containing minimal information.

Table 3.6: OPFLOW initializations

Initialization type	Meaning
MIDPOINT	Use mid-point of bounds
FROMFILE	Use values from network file
ACPF	Run AC power flow
FLATSTART	Flat-start
DCOPF	Run DC optimal power flow

Table 3.7: OPFLOW generator bus voltage control modes

Voltage control type	Meaning	Compatibility
FIXED_WITHIN_QBOUNDS	Fixed within reactive power bounds	POWER_BALANCE_POLAR only
VARIABLE_WITHIN_BOUNDS	Variable within voltage bounds	All models

Table 3.8: OPFLOW objective function types

Objective function	Meaning	Compatibility
MIN_GEN_COST	Minimize generation cost	All models
MIN_GENSETPOINT_DEVIATION	Minimize deviation (ramp up-down) from generator set-point	POWER_BALANCE_POLAR model only
NO_OBJ	No objective function (only feasibility)	All models

3.7 Examples

Some OPFLOW example runs are provided with some sample output. Options values are the default values in table 3.4 unless otherwise specified. `-print_output 1` is only used in the first example to save space. Sample output is generated by running examples from the installation directory.

Example using the Ipopt [7]solver:

```
input{output/opflow_ipopt_output.tex}
```

Example using the HiOp [6, 5]solver on the CPU with ACPF initialization:

```
input{output/opflow_hiop_output.tex}
```

Chapter 4

Multi-period optimal power flow (TCOPFLOW)

TCOPFLOW solves a full AC multi-period optimal power flow problem with the objective of minimizing the total cost over the given time horizon while adhering to constraints for each period and between consecutive time-periods (ramping constraints).

4.1 Formulation

The multi-period optimal power flow problem is a series of optimal power flow problems coupled via temporal constraints. The generator real power deviation ($p^{g}_{jt} - p^{g}_{jt-\Delta t}$) constrained within the ramp limits form the temporal constraints. An illustration of the temporal constraints is shown in Fig. 4.1 with four time steps. Each time-step t is coupled with its preceding time $t - \Delta t$, where Δt is the time-step where the objective is to find a least cost dispatch for the given time horizon.

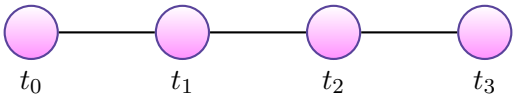

Figure 4.1: Multi-period optimal power flow example with four time-steps. The lines connecting the different time-periods denote the coupling between them.

In general form, the equations for multi-period optimal power flow are given by (4.1) – (4.5). TCOPFLOW solves to minimize the total generation cost $\sum_{t=0}^{N_t-1} f(x_t)$ over the time horizon, where N_t is the number of time-steps. At each time-step, the equality constraints ($g(x_t)$), inequality $h(x_t)$, and the lower/upper limit (x^-,x^+) constraints need to be satisfied. Equation (4.5) represents the coupling between the consecutive time-steps. It is the most common form of coupling that limits the deviation of the real power generation at time t from its preceding time-step $t - \Delta t$ to within its ramping capability Δx_t.

$$\min \sum_{t=0}^{N_t-1} f(x_t) \tag{4.1}$$

s.t.

$$g(x_t) = 0, \qquad t \in [0, N_t - 1] \tag{4.2}$$

$$h(x_t) \leq 0, \qquad t \in [0, N_t - 1] \tag{4.3}$$

$$x^- \leq x_t \leq x^+, \qquad t \in [0, N_t - 1] \tag{4.4}$$

$$-\Delta x_t \leq x_t - x_{t-\Delta t} \leq \Delta x_t, \qquad t \in [1, N_t - 1] \tag{4.5}$$

4.2 Solvers

Currently, `ExaGO` only supports solving TCOPFLOW using Ipopt [7] on on a single rank.

4.3 Input and Output

- **Network file:** The network file describing the network details. Only MATPOWER format files are currently supported.
- **Load data:** One file for load real power and one for reactive power. The files need to be in CSV format. An example of the format for the 9-bus case is here.
- **Wind generation:** The wind generation time-series described in CSV format. See an example of the format here.

If the load data and/or wind generation profiles are not set then a flat profile is assumed, i.e., the load and wind generation for all hours is constant.

The TCOPFLOW output is saved to a directory named `tcopflowout`. This directory contains N_t files, one for each time-step, in MATPOWER data file format.

4.4 Usage

```
input{output/tcopflow_output.tex}
```

Chapter 5

Security-constrained optimal power flow (SCOPFLOW)

SCOPFLOW solves a contingency-constrained optimal power flow problem. The problem is set up as a two-stage optimization problem where the first-stage (base-case) represents the normal operation of the grid and the second-stage comprises N_c contingency scenarios. Each contingency scenario can be single or multi-period.

5.1 Formulation

5.1.1 Single-period

The contingency-constrained optimal power flow (popularly termed as security-constrained optimal power flow (SCOPF) in power system parlance) attempts to find a least cost dispatch for the base case (or no contingency) while ensuring that if any of contingencies do occur then the system will be secure. This is illustrated in Fig. 5.1 for a SCOPF with a base-case c_0 and three contingencies.

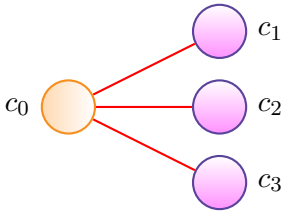

Figure 5.1: Contingency constrained optimal power flow example with three contingencies. c_0 represents the base case (or no contingency case). c_1, c_2, c_3 are the three contingency cases. Each of the contingency states is coupled with the base-case through ramping constraints (denoted by red lines)

In general form, the equations for contingency-constrained optimal power flow are given by (5.1) – (5.5). This is a two-stage stochastic optimization problem where the first stage is the base case c_0 and each of the contingency states $c_i, i \in [1, N_c]$ are second-stage subproblems. SCOPFLOW aims to minimize the objective $\sum_{c=0}^{N_c} f(x_c)$, while adhering to the equality $g_c(x_c)$, inequality $h_c(x_c)$, and the lower/upper bound (x^-,x^+) constraints. Equation (5.5) represents the coupling between the base-case c_0 and each of the contingency states c_i. Equation (5.5) is the most typical form of coupling that limits the deviation of the contingency variables x_c from the base x_0 to within Δx_c. An example of this

constraint could be the allowed real power output deviation for the generators constrained by their ramp limit, which is currently the only constraint SCOPFLOW supports.

$$\min \sum_{c=0}^{N_c} f_c(x_c) \tag{5.1}$$

s.t.

$$g_c(x_c) = 0, \qquad c \in [0, N_c] \tag{5.2}$$

$$h_c(x_c) \leq 0, \qquad c \in [0, N_c] \tag{5.3}$$

$$x^- \leq x_c \leq x^+, \qquad c \in [0, N_c] \tag{5.4}$$

$$-\Delta x_c \leq x_c - x_0 \leq \Delta x_c, \qquad c \in [1, N_c] \tag{5.5}$$

5.1.2 Multiperiod

In the multi-period version, each contingency is comprised of multiple time-periods. The multiple periods have variables and constraints as described in chapter 4. An example of multi-contingency multi-period optimal power flow is illustrated in Fig. 5.2 for multi-period SCOPFLOW with three contingencies c_1, c_2, and c_3 coupled to the base case c_0. Each state is multi-period with two time-periods. Each time-step is coupled with its adjacent one through ramping constraints. We assume that the contingency is incident at the first time-step, i.e. at t_0. This results in the coupling between the contingency cases $c_i, i \in [1, N_c]$ and the base-case c_0 only at time-step t_0.

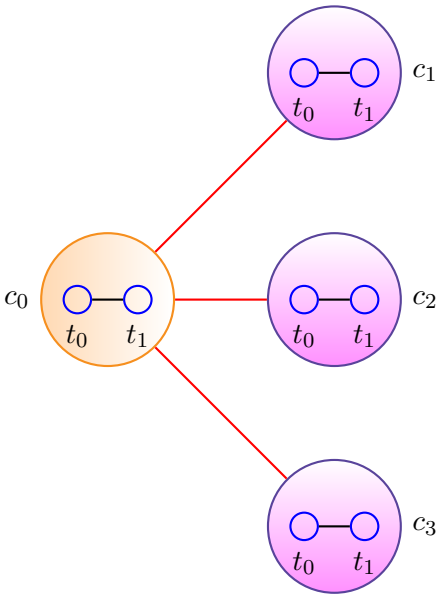

Figure 5.2: Multi-period contingency constrained optimal power flow example with two contingencies c_0 and c_1, each with two time-periods t_0, t_1. State c_0 represents the base case (no contingency) case. The contingency states c_1,c_2,c_3 are coupled with the no-contingency state c_0. The red line denotes the coupling between the contingencies.

The overall objective of this contingency-constrained multi-period optimal power flow is to find a secure dispatch for base-case c_0 while adhering to contingency and temporal constraints. The general formulation of this problem is given in Eqs. (5.6) – (5.12).

$$\min \sum_{c=0}^{N_c} \sum_{t=0}^{N_t-1} f_{ct}(x_{c,t}) \tag{5.6}$$

s.t.

$$g_{ct}(x_{c,t}) = 0, \qquad c \in [0, N_c], t \in [0, N_t - 1] \tag{5.7}$$

$$h_{ct}(x_{c,t}) \leq 0, \qquad c \in [0, N_c], t \in [0, N_t - 1] \tag{5.8}$$

$$x^- \leq x_{c,t} \leq x^+, \qquad c \in [0, N_c], t \in [0, N_t - 1] \tag{5.9}$$

$$-\Delta x_t \leq x_{c,t} - x_{c,t-\Delta t} \leq \Delta x_t, \qquad c \in [0, N_c], t \in [1, N_t - 1] \tag{5.10}$$

$$-\Delta x_c \leq x_{c,0} - x_{0,0} \leq \Delta x_c, \qquad c \in [1, N_c] \tag{5.11}$$

$$\tag{5.12}$$

In this formulation, the objective is to reduce the cost for the base-case time horizon, where $f(x_{0,t})$ is the objective cost for contingency c_0 at time t. Equation (5.12) represents the coupling between the base case c_0 and each contingency c_i at time-step t_0. We use a simple box constraint Δx_c to restrict the deviation of decision variables $x_{c,0}$ from the base-case $x_{0,0}$. The bound Δx_c could represent here, for example, the allowable reserve for each generator.

5.2 Solvers

SCOPFLOW supports solving the optimization problem via Ipopt [7], HiOp [6, 5], or *EMPAR*. Ipopt [7]can solve SCOPFLOW on single rank only. HiOp [6, 5]supports solving the problem in parallel using a primal-decomposition algorithm. With HIOP, one can solve the subproblem either on the CPU or GPU by selecting the appropriate subproblem model and solver (see options table below). However, note that *ExaGO needs to be built with Ipopt [7]even when using HiOp [6, 5]solver.* The *EMPAR* solver does not solve the security-constrained ACOPF problem, it only solves the base-case and the contingencies independently with OPFLOW. It distributes the contingencies to different processes when executed in parallel.

5.3 Input and Output

To execute SCOPFLOW, the following files are required:

- **Network file:** The network file describing the network details. Only MATPOWER format files are currently supported.
- **Contingency file:** The file describing the contingencies. Contingencies can be single or multiple outages. The contingency file needs to be described in PTI format.

If the multi-period option is chosen, then additional files describing the load and wind generation can be (optionally) set.

- **Load data:** One file for load real power and one for reactive power. The files need to be in CSV format. An example of the format for the 9-bus case is here.
- **Wind generation:** The wind generation time-series described in CSV format. See an example of the format here.

The SCOPFLOW output is saved to a directory named `scopflowout`. This directory contains N_c files to save the solution for each contingency in MATPOWER datafile format. Each file has the name `cont_xx` where `xx` is the contingency number.

If the multi-period option is chosen then N_c subdirectories are created (one for each contingency), and each subdirectory contains N_t output files, one for each time-period. The subdirectories have the naming convention `cont_xx` and the output file are named as `t_yy` where `yy` is the time-step number.

5.4 Usage

```
./scopflow -netfile <netfilename> -ctgcfile <ctgcfilename> \
<scopflowoptions> [-scopflow_enable_multiperiod 1]
```

5.5 Options

See table 5.1. In addition, all OPFLOW options in Table 3.4 and TCOPFLOW options in Table **??** can be used.

Table 5.1: SCOPFLOW options

Option	Meaning	Values (Default value)	Compatibility
-netfile	Network file name	string < 4096 characters (case9mod_gen3_wind.m)	
-ctgcfile	Contingency file name	string < 4096 characters (case9.cont)	
-print_output	Print output to screen	0 or 1 (0)	
-save_output	Save output to directory	0 or 1 (0)	Format determined by OPFLOW option.
-scopflow_output_directory	Output directory path	"scopflowout"	
-scopflow_model	SCOPFLOW model type	GENRAMP, GENRAMPT (GENRAMP)	
-scopflow_solver	Set solver for scopflow	IPOPT, HIOP, or EMPAR (IPOPT)	
-scopflow_subproblem_solver	Set solver for subproblem	IPOPT or HIOP (IPOPT)	Only when using HIOP solver for SCOPFLOW
-scopflow_subproblem_model	Set model for subproblem	See OPFLOW chapter	Only when using HIOP solver for SCOPFLOW
-scopflow_Nc	Number of contingencies	int (0. Passing -1 results in all contingencies in the file used)	
-scopflow_mode	Operation mode: Preventive or corrective	0 or 1 (0)	
-scopflow_enable_multiperiod	Multi-period SCOPFLOW	0 or 1 (0)	IPOPT solver only
-scopflow_ploadprofile	Real power load profile	string (load_P.csv)	
-scopflow_qloadprofile	Reactive power load profile	string (load_Q.csv)	
-scopflow_windgenprofile	Wind generation profile	string (case9/scenarios_9bus.csv)	
-scopflow_dT	Length of time-step (minutes)	double (5.0)	
-scopflow_duration	Total duration (hours)	double (0.5)	
-scopflow_tolerance	Optimization solver tolerance	double (1e-6)	

Depending on the value chosen for `-scopflow_mode`, SCOPFLOW can operate either in *preventive* (mode = 0) or *corrective* (mode = 1) mode. In the preventive mode, the PV and PQ generator real power is fixed to its corresponding base-case values. The generators at the reference bus pick up any make-up power required for the contingency. The corrective mode allows deviation of the PV and PQ generator real power from the base-case dispatch constrained by its 30-min. ramp rate capability. The optimization decides the optimal redispatch. One can have AGC control instead of having the generators proportionally share the deficit/excess power by using the option *(*-opflow˙use˙agc).

The option `-scopflow_enable_multiperiod 1` must be used in order to enable any of the options listed in table 5.1 for multiperiod analysis.

5.6 Examples

Some SCOPFLOW example runs are provided with some sample output. Options are the default options given in table 3.4, **??** and 5.1 unless otherwise specified. Sample output is generated by running examples in the installation directory.

Example using the Ipopt [7]solver:

```
input{output/scopflow_ipopt_output.tex}
```

Example using HIOP solver with IPOPT subproblem solver.

```
input{output/scopflow_hiop_output.tex}
```

Chapter 6

Stochastic optimal power flow (SOPFLOW)

SOPFLOW solves a stochastic security-constrained multi-period optimal power flow problem. The problem is set up as a two-stage optimization problem where the first-stage (base-case) represents the normal operation of the grid (or the most likely forecast) and the second-stage comprises N_s scenarios of forecast deviation. Each scenario can have multiple contingencies and each contingency can be multi-period.

6.1 Formulation

An illustration of SOPFLOW is shown in Fig. 6.1 for a case with two scenarios s_0 and s_1 with three contingencies each, and each scenario/contingency with two time-periods. We assume that any contingency is incident at the first time-step, i.e., at t_0.

The full formulation for the stochastic security-constrained multi-period optimal power flow is given in (6.1) – (6.7). In this formulation, the objective is to reduce the expected cost, where $f(x_{s,c,t})$ is the cost for scenario s with contingency c at time t. π_s is the probability of scenario s.

$$\min \sum_{s=0}^{N_s-1} \pi_s \sum_{c=0}^{N_c-1} \sum_{t=0}^{N_t-1} f(x_{s,c,t}) \tag{6.1}$$

s.t.

$$g(x_{s,c,t}) = 0, \qquad s \in [1, N_s - 1], c \in [0, N_c - 1], t \in [0, N_t - 1] \tag{6.2}$$

$$h(x_{s,c,t}) \leq 0, \qquad s \in [1, N_s - 1], c \in [0, N_c - 1], t \in [0, N_t - 1] \tag{6.3}$$

$$x^- \leq x_{s,c,t} \leq x^+, \qquad s \in [1, N_s - 1], c \in [0, N_c - 1], t \in [0, N_t - 1] \tag{6.4}$$

$$-\Delta x_t \leq x_{s,c,t} - x_{s,c,t-\Delta t} \leq \Delta x_t, \qquad s \in [1, N_s - 1], c \in [0, N_c - 1], t \in [1, N_t - 1] \tag{6.5}$$

$$-\Delta x_c \leq x_{s,c,0} - x_{s,0,0} \leq \Delta x_c, \qquad s \in [1, N_s - 1], c \in [1, N_c - 1] \tag{6.6}$$

$$-\Delta x_s \leq x_{s,0,0} - x_{0,0,0} \leq \Delta x_s, \qquad s \in [1, N_s - 1] \tag{6.7}$$

The modeling details used for an optimial power flow problem are also used for a SOPFLOW problem, i.e., each of the circles shown in Fig. 6.1 has the modeling details of an optimal power flow problem (OPFLOW). Incorporating the probabilities π_s for each scenario is not implemented yet which leads to each scenario having an equal probability.

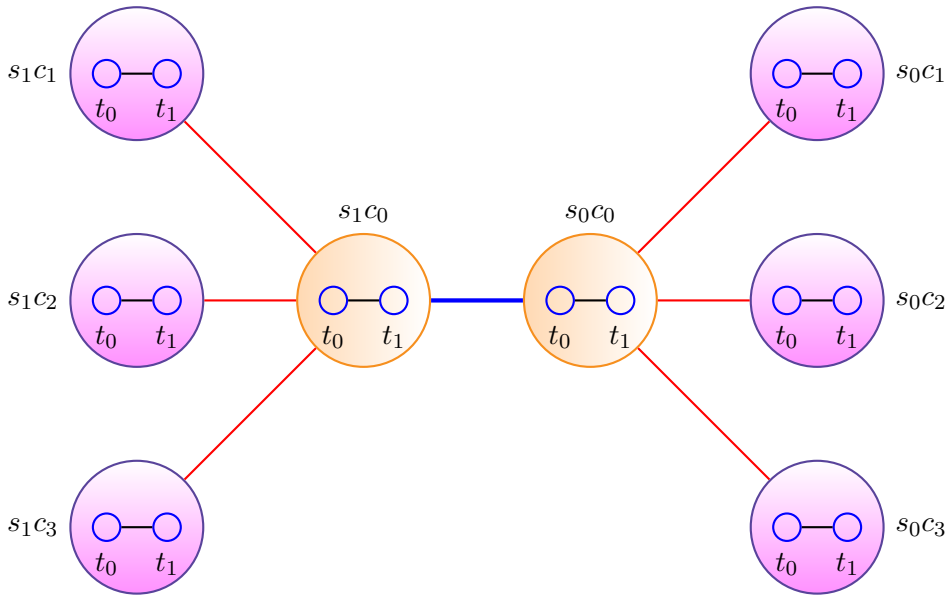

Figure 6.1: Stochastic multi-period contingency constrained structure with two scenarios s_0 and s_1. Each scenario has three contingencies c_1,c_2,and c_3. s_0c_0 and s_1c_0 denote the base-cases for the two scenarios. Each scenario and contingency has two time-periods t_0, and t_2, t_2. The red line denotes the coupling between the contingencies and their respective base-case scenarios.The blue line denotes the coupling between the scenarios

Depending on the options selected, SOPFLOW can be used to solve

- Single-period stochastic optimal power flow : No contingencies or time-periods
- Single-period contingency-constrained stochastic optimal power flow : No time-periods
- Multi-period security-constrained stochastic optimal power flow : Full formulation

Currently, SOPFLOW uses wind power generation as the stochastic variables and each scenario is a realization of the power output from wind generators. A zero fuel cost is used for wind power generation to ensure wind generation would be the dispatched to the given target level (upper limit).

For the contingecy-constrained stochastic optimal power flow, SOPFLOW flattens out the contingencies and scenarios to a two-level formulation. In this formulation, all the scenarios (and their contingencies) are coupled to a base scenario problem.

For contingencies, SOPFLOW supports generation and/or transmission outages. A contingency can have multiple outages, but, it should not cause any islanding. The coupling between the no-contingency and the contingency case for each scenario is also the difference in real power output ($p^{\mathrm{g}}_{jsct} - p^{\mathrm{g}}_{js0t},\ j \in J^{\mathrm{gen}}$) that must be within the 30 minute generator ramp rate. Refer to 5 for details on the contingency modeling.

For multi time-period, we use ramping constraints on the generator real power output between successive time steps.

SOPFLOW can be run in two modes: preventive and corrective. In the preventive mode, generator real power output is fixed to the base-case values for generators at PV bus(es). In this mode, the generators at the reference bus provide/absorb any deficit/surplus power. The corrective mode allows deviation of the PV and PQ generator real power from the base-case dispatch constrained by its 30-min.

ramp rate capability. Note that the preventive/corrective mode is only applied at the first step t_0. In the successive time-steps, the generator dispatch is dictated by the previous step dispatch and the ramp limits.

6.2 Solvers

SOPFLOW supports solving the optimization problem via Ipopt [7], HiOp [6, 5], or *EMPAR*. Ipopt [7]can solve SOPFLOW on single rank only. HiOp [6, 5]supports solving the problem in parallel using a primal-decomposition algorithm. With HIOP, one can solve the subproblem either on the CPU or GPU by selecting the appropriate subproblem model and solver (see options table below). However, note that *ExaGO needs to be built with Ipopt [7]even when using HiOp [6, 5]solver.* The *EMPAR* solver does not solve the stochastic ACOPF problem, but it only solves the base-case and the stochastic scenarios independently with OPFLOW. It distributes the scenarios and contingencies to different processes when executed in parallel. Table 6.1 lists the compatibility of the different solvers for different variations of SOPFLOW .

Table 6.1: SOPFLOW solver compatibility

Solver	Stochastic scenarios	Include contingencies	Include multi-period
IPOPT	Y	Y	Y
HIOP	Y	Y	N
EMPAR	Y	Y	Y

6.3 Input and Output

The following files are needed for executing SOPFLOW.

- **Network file:** The network file describing the network details. Only MATPOWER format files are currently supported.
- **Scenario file:** SOPFLOW only supports reading wind generation scenarios in a CSV format. An example of this format for the 9-bus case is here.
- **Contingency file:** Contingencies can be specified via PTI format file as described in chapter 5. The option `-sopflow_enable_multicontingency` should be set for multi-contingency problems.
- **Load data:** One file for load real power and one for reactive power. The files need to be in CSV format. An example of the format for the 9-bus case is here.

The SOPFLOW output is saved to a directory named `sopflowout`. This directory contains N_s subdirectories to save the solution for each scenario. Each of these subdirectories contain N_c subdirectories, one for each contingency. Each contingency subdirectory has N_t MATPOWER format files to store the output for each time-period for the given contingency and scenario. The subdirectories have the directory name format `scen_x` where x is the scenario number, `cont_y` where y is the contingency number, and the output files have the file name format `t_z` where z is the time-step number.

6.4 Usage

```
./sopflow -netfile <netfilename>  -windgen <wind_scenario_filename> \
[-sopflow_enable_multicontingency 1] <sopflowoptions>
```

6.5 Options

Table 6.2: SOPFLOW options

Option	Meaning	Values (Default value)	Compatibility
-netfile	Network file name	string 4096 characters (case9mod_gen3_wind.m)	
-windgen	Scenario file name	string 4096 characters (case9/10scenarios_9bus.csv)	
-ctgcfile	Contingency file name	string (case9.cont)	
-save_output	Save output to directory	0 or 1 (0)	Format determined by OPFLOW option.
-sopflow_output_directory	Output directory path	"sopflowout"	
-sopflow_mode	Operation mode: Preventive or corrective	0 or 1 (0)	
-sopflow_model	Set model for SOPFLOW	GENRAMP, GENRAMPC (GENRAMP)	
-sopflow_solver	Set solver for SOPFLOW	IPOPT, HIOP, or EMPAR	
-sopflow_subproblem_solver	Set solver for subproblem	IPOPT or HIOP (IPOPT)	Only when using HIOP solver for SOPFLOW
-sopflow_subproblem_model	Set model for subproblem	See OPFLOW chapter	Only when using HIOP solver for SOPFLOW
-sopflow_enable_multicontingency	Multi-contingency SOPFLOW	0 or 1 (0)	
-sopflow_flatten_contingencies	Flatten contingencies for SOPFLOW	0 or 1 (1)	
-sopflow_Ns	Number of scenarios	int (Default 0. Use -1 to select all scenarios from the scenario file)	
-sopflow_Nc	Number of contingencies	int (0. Passing -1 results in all contingencies in the file used)	

6.6 Examples

Some SOPFLOW example runs are provided with some sample output. Options are the default options given in Tables 3.4, **??**, 5.1 and 6.2 unless otherwise specified. Sample output is generated by running examples in the installation directory.

Example using the Ipopt [7]solver:

```
input{output/sopflow_ipopt_output.tex}
```

Example using the *IPOPT* solver with multicontingency enabled:

```
input{output/sopflow_ipopt_mc_output.tex}
```

Chapter 7

Python Bindings

The ExaGO Python bindings are a wrapper around the C++ API to provide an interface to ExaGO in Python. This interface aims to provide equivalent functionality between the ExaGO C++ API and the Python API.

The wrapper uses an object-oriented API slightly different from the C++ API. The C++ API uses the application type in uppercase as the prefix for its methods, where they are native methods in Python. ExaGO Python instances must be destroyed, using del, before calling exago.finalize(), like calling *Destroy() with the C++ API. Failure to do so will cause segmentation faults or other memory errors. In section 7.3, are tables that detail the mappings between functions and objects in the Python vs C++.

7.1 Code Comparison

Below is a comparison of what solving an OPF looks like through the C++ and Python APIs.

C++:

```
#include <opflow.h>
#include <exago_config.h>

static char help[] = "User example calling OPFLOW.\n";
static char appname[] = "opflow";

int main(int argc, char** argv) {

  /* Initialize ExaGO application */
  PetscErrorCode ierr;
  OPFLOW opflow;
  MPI.Comm comm = MPI.COMM_WORLD;
  ierr = ExaGOInitialize(comm, &argc, &argv, appname, help);

  /* Create OPFLOW object */
  ierr = OPFLOWCreate(comm, &opflow);
  ExaGOCheckError(ierr);

  /* Read network data */
```

```
  ierr = OPFLOWReadMatPowerData(opflow, "datafiles/case9/case9mod.m");
  ExaGOCheckError(ierr);

  /* Solve */
  ierr = OPFLOWSolve(opflow);
  ExaGOCheckError(ierr);

  /* Print solution */
  ierr = OPFLOWPrintSolution(opflow);
  ExaGOCheckError(ierr);

  /* Destroy OPFLOW object */
  ierr = OPFLOWDestroy(&opflow);
  ExaGOCheckError(ierr);

  /* Clean up resources */
  ExaGOFinalize();

  return 0;
}
```

Python:

```
import exago
exago.initialize("opflow")
opf = exago.OPFLOW()
opf.read_mat_power_data('datafiles/case9/case9mod.m')
opf.solve()
opf.print_solution()
del opf
exago.finalize()
```

7.2 Building with MPI

ExaGO depends on mpi4py when running through the Python interface. When running `ExaGO`, you may need to disable threading through mpi4py before importing exago:

```
import mpi4py.rc
mpi4py.rc.threads = False
from mpi4py import MPI
import exago
comm = MPI.COMM_WORLD
exago.initialize("app", comm)
# ...
exago.finalize()
```

Additionally linting tools may re-order your imports, and so you may need to add appropriate comments (`# noqa`) in order to avoid this:

```
import mpi4py.rc
mpi4py.rc.threads = False
from mpi4py import MPI # noqa
import exago # noqa
```

7.3 Bindings Tables

7.3.1 ExaGO

Table 7.1: ExaGO Python Bindings

C++ API	Python API	Notes
ExaGOInitialize	exago.initialize	
ExaGOFinalize	exago.finalize	
	exago_prefix	Returns the path to the installation directory of ExaGO (for finding mat power data files, etc)
OutputFormat enum	exago.OutputFormat enum	Output format type for functions like save_solution_ Possible values for this enum can be found in 7.4

7.3.2 PFLOW

The functions in the following table could be called by a pflow object (i.e. `pflow = exago.PFLOW(); pflow.solve();`).

Table 7.2: PFLOW Python Bindings

C++ API	Python API	Notes
PFLOW	exago.PFLOW class	
PFLOWReadMatPowerData	read_mat_power_data	
PFLOWSolve	solve	

7.3.3 OPFLOW

The functions in the following table could be called by a opflow object (i.e. `opflow = exago.OPFLOW(); opflow.solve();`).

Table 7.3: OPFLOW Python Bindings

C++ API	Python API	Notes
OPFLOW	exago.OPFLOW class	
OPFLOWObjectiveType enum	exago.OPFLOWObjectiveType enum	More details and possible values for this enum can be found in 7.4
OPFLOWInitializationType enum	exago.OPFLOWInitializationType enum	More details and possible values for this enum can be found in 7.4
OPFLOWGenBusVoltageType enum	exago.OPFLOWGenBusVoltageType enum	More details and possible values for this enum can be found in 7.4
OPFLOWSetObjectiveType	set_objective_type	
OPFLOWSetInitializationType	set_initialization_type	
OPFLOWSetGenBusVoltageType	set_gen_bus_voltage_type	

Table 7.3: OPFLOW Python Bindings (cont.)

C++ API	Python API	Notes
OPFLOWSetModel	set_model	options are "PBPOLRAJAHIOP", "POWER_BALANCE_HIOP", and "POWER_BALANCE_POLAR"
OPFLOWSetSolver	set_solver	options are "IPOPT", "HIOP", and "HIOPSPARSE"
OPFLOWHasGenSetPoint	set_has_gen_set_point	
OPFLOWSetHIOPComputeMode	set_hiop_compute_mode	options are "CPU" or "GPU"
OPFLOWSetHIOPMemSpace	set_hiop_mem_space	options are "DEFAULT", "HOST", "UM", and "DEVICE"
OPFLOWHasLoadLoss	set_has_load_loss	
OPFLOWIgnoreLineflowConstraints	set_ignore_lineflow_constraints	
OPFLOWHasBusPowerImbalance	set_has_bus_power_imbalance	
OPFLOWUseAGC	set_use_agc	
OPFLOWSetHIOPVerbosityLevel	set_hiop_verbosity_level	integer between 0 and 10
OPFLOWSetLoadLossPenalty	set_loadloss_penalty	
OPFLOWSetBusPowerImbalancePenalty	set_bus_power_imbalance_penalty	
OPFLOWSetTolerance	set_tolerance	
OPFLOWSetWeight	set_weight	
PSSetGenPowerLimits	ps_set_gen_power_limits	
OPFLOWGetTolerance	get_tolerance	
OPFLOWGetHIOPComputeMode	get_hiop_compute_mode	
OPFLOWGetHIOPMemSpace	get_hiop_mem_space	
OPFLOWGetModel	get_model	
OPFLOWGetSolver	get_solver	
OPFLOWGetConvergenceStatus	get_convergence_status	
OPFLOWGetObjectiveType	get_objective_type	
OPFLOWGetInitializationType	get_initialization_type	
OPFLOWGetGenBusVoltageType	get_gen_bus_voltage_type	
OPFLOWGetHasGenSetPoint	get_has_gen_set_point	
OPFLOWGetLoadlossPenalty	get_loadloss_penalty	
OPFLOWGetIgnoreLineflowConstraints	get_ignore_lineflow_constraints	
OPFLOWGetHasLoadloss	get_has_loadloss	
OPFLOWGetHasBusPowerImbalance	get_has_bus_power_imbalance	
OPFLOWGetUseAGC	get_use_agc	
OPFLOWGetHIOPVerbosityLevel	get_hiop_verbosity_level	
OPFLOWGetBusPowerImbalancePenalty	get_bus_power_imbalance_penalty	
PSGetGenDispatch	get_gen_dispatch	
OPFLOWGetObjectiveTypes	get_objective_types	
OPFLOWGetInitializationTypes	get_initialization_types	
OPFLOWGetGenBusVoltage	get_gen_bus_voltage	
OPFLOWGetObjective	get_objective	
OPFLOWSolve	solve	
OPFLOWPrintSolution	print_solution	
OPFLOWSaveSolution	save_solution	
OPFLOWReadMatPowerData	read_mat_power_data	
OPFLOWSolutionToPS	solution_to_ps	
OPFLOWSetUpPS	set_up_ps	
OPFLOWSkipOptions	skip_options	
OPFLOWSetLinesMonitored	set_lines_monitored	implemented as two different methods
	set_lines_monitored([...])	Specify a list of line kvlevels (type float) to monitor

Table 7.3: OPFLOW Python Bindings (cont.)

C++ API	Python API	Notes
	`set_lines_monitored(n, "file")`	Read n line kvlevels from a file (n=-1 for all)_

7.3.4 SCOPFLOW

The functions in the following table could be called by a scopflow object (i.e. `scopflow = exago.SCOPFLOW(); scopflow.solve();`)

Table 7.4: SCOPFLOW Python Bindings

C++ API	Python API	Notes
`SCOPFLOW`	`exago.SCOPFLOW class`	
`ContingencyFileInputFormat enum`	`exago.ContingencyFileInputFormat enum`	More details and possible values for this enum can be found in the 7.4. Currently, this is only be used as a Python enum. A string representation is not available.
`SCOPFLOWSetModel`	`set_model`	
`SCOPFLOWSetNetworkData`	`set_network_data`	
`SCOPFLOWSetLoadProfiles`	`set_load_profiles`	
`SCOPFLOWSetNumContingencies`	`set_num_contingencies`	
`SCOPFLOWSetContingencyData`	`set_contingency_data`	
`SCOPFLOWSetPLoadData`	`set_pload_data`	
`SCOPFLOWSetQLoadData`	`set_qload_data`	
`SCOPFLOWSetWindGenProfile`	`set_wind_gen_profile`	
`SCOPFLOWSetTimeStep`	`set_time_step`	
`SCOPFLOWSetDuration`	`set_duration`	
`SCOPFLOWSetTimeStepandDuration`	`set_time_step_and_duration`	
`SCOPFLOWSetTolerance`	`set_tolerance`	
`SCOPFLOWSetVerbosityLevel`	`set_verbosity_level`	
`SCOPFLOWSetComputeMode`	`set_compute_mode`	
`SCOPFLOWSetSolver`	`set_solver`	
`SCOPFLOWSetSubproblemModel`	`set_subproblem_model`	
`SCOPFLOWSetSubproblemSolver`	`set_subproblem_solver`	
`SCOPFLOWSetInitilizationType`	`set_initialization_type`	
`SCOPFLOWSetGenBusVoltageType`	`set_gen_bus_voltage_type`	
`SCOPFLOWEnableMultiPeriod`	`enable_multi_period`	
`SCOPFLOWEnablePowerImbalanceVariables`	`enable_power_imbalance_variables`	
`SCOPFLOWIgnoreLineflowConstraints`	`ignore_lineflow_constraints`	
`SCOPFLOWGetTolerance`	`get_tolerance`	
`SCOPFLOWGetNumIterations`	`get_num_iterations`	
`SCOPFLOWGetConvergenceStatus`	`get_convergence_status`	
`SCOPFLOWGetTotalObjective`	`get_total_objective`	
`SCOPFLOWGetBaseObjective`	`get_base_objective`	
`SCOPFLOWSetUp`	`set_up`	
`SCOPFLOWSolve`	`solve`	
`SCOPFLOWPrintSolution`	`print_solution`	
`SCOPFLOWSaveSolution`	`save_solution`	
`SCOPFLOWSaveSolutionDefault`	`save_solution_default`	
`SCOPFLOWSaveSolutionAll`	`save_solution_all`	
`SCOPFLOWSaveSolutionAllDefault`	`save_solution_all_default`	

7.3.5 SOPFLOW

The functions in the following table could be called by a sopflow object (i.e. `sopflow = exago.SOPFLOW(); sopflow.solve();`)

Table 7.5: SOPFLOW Python Bindings

C++ API	Python API	Notes
SCOPFLOW	exago.SCOPFLOW class	
ScenarioFileInputFormat enum	ScenarioFileInputFormat enum	More details and possible values for this enum can be found in 7.4. Currently, this is only be used as a Python enum. A string representation is not available.
ScenarioUncertaintyType enum	ScenarioUncertaintyType enum	More details and possible values for this enum can be found in the 7.4. Currently, this is only be used as a Python enum. A string representation is not available.
SOPFLOWSetModel	set_model	
SOPFLOWSetNetworkData	set_network_data	
SOPFLOWSetContingencyData	set_contingency_data	
SOPFLOWSetNumContingencies	set_num_contingencies	
SOPFLOWSetScenarioData	set_scenario_data	
SOPFLOWSetNumScenarios	set_num_scenarios	
SOPFLOWSetWindGenProfile	set_wind_gen_profile	
SOPFLOWSetTimeStepandDuration	set_time_step_and_duration	
SOPFLOWSetTolerance	set_tolerance	
SOPFLOWSetSubproblemVerbosityLevel	set_subproblem_verbosity_level	
SOPFLOWSetSubproblemComputeMode	set_subproblem_compute_mode	
SOPFLOWSetSubproblemModel	set_subproblem_model	
SOPFLOWSetSubproblemSolver	set_subproblem_solver	
SOPFLOWSetSolver	set_solver	
SOPFLOWSetInitializationType	set_initialization_type	
SOPFLOWSetGenBusVoltageType	set_gen_bus_voltage_type	
SOPFLOWSetLoadProfiles	set_load_profiles	
SOPFLOWSetLoadProfiles	set_ignore_lineflow_constraints	
SOPFLOWEnableMultiContingency	enable_multi_contingency	
SOPFLOWFlattenContingencies	flatten_contingencies	
SOPFLOWGetNumScenarios	get_num_scenarios	
SOPFLOWGetNumIterations	get_num_iterations	
SOPFLOWGetConvergenceStatus	get_convergence_status	
SOPFLOWGetTotalObjective	get_total_objective	
SOPFLOWGetConvergenceStatus	get_converged_status	
SOPFLOWGetTolerance	get_tolerance	
SOPFLOWSetUp	setup	
SOPFLOWSolve	solve	
SOPFLOWPrintSolution	print_solution	
SOPFLOWSaveSolution	save_solution	
SOPFLOWSaveSolutionAll	save_solution_all	

7.4 Enums

The enums in ExaGO serve as integer mapped values for certain settings within the application. Several type settings, OPFLOWObjectiveType, OPFLOWInitializationType, and OPFLOWGenBusVolt-

ageType, are specific to OPFLOW.

Instances can be constructed directly through the ExaGO library (i.e. `exago.OutputFormat.CSV`). The possible values for the current enums are as follows:

Possible Values		
OPFLOWObjectiveType	**OPFLOWInitializationType**	**OPFLOWGenBusVoltageType**
- MIN_GEN_COST - MIN_GENSETPOINT_DEVIATION - NO_OBJ	- OPFLOWINIT_FROMFILE - OPFLOWINIT_MIDPOINT - OPFLOWINIT_ACPF - OPFLOW_FLATSTART	- VARIABLE_WITHIN_BOUNDS - FIXED_WITHIN_QBOUNDS - FIXED_AT_SETPOINT

Possible Values			
OutputFormat	**ContingencyFileInputFormat**	**ScenarioFileInputFormat**	**ScenarioUncertaintyType**
- CSV - MATPOWER - JSON - MINIMAL	- NATIVE - PSSE	- NATIVE_SINGLEPERIOD - NATIVE_MULTIPERIOD	- NONE - WIND - LOAD

Note: getters for the possible values of ContingencyFileInputeFormat, ScenarioFileInputFormat, ScenarioUncertaintyType are not currently available.

Instances can be constructed directly through the exago library (i.e. `exago.OPFLOWObjectiveType.MIN_GEN_COST` or `exago.MIN_GEN_COST`)

Setter functions for these OPFLOW Type configurations can take an integer, an instance of the exago enum, or a string that describes the enum (i.e. 'MIN_GEN_COST'). The rest currently only accept an instance of the exago enum. The possible values for these OPFLOW Type enums can be retrieved through opflow.get_xxx_types() (i.e. opflow.get_gen_bus_voltage_types()). The opflow.get_xxx_types functions yield a list of the enum values.

The next section covers some code examples on how to get and use these values.

7.4.1 Code Examples

Set with a string

```
>>> opflow.set_initialization_type('OPFLOWINIT_FROMFILE')
>>> opflow.get_initialization_type()
OPFLOWInitializationType.OPFLOWINIT_FROMFILE
```

Set with an integer

```
>>> opflow.set_initialization_type(1)
>>> opflow.get_initialization_type()
OPFLOWInitializationType.OPFLOWINIT_FROMFILE
```

Set with an enum instance

```
>>> opflow.set_initialization_type(exago.OPFLOWInitializationType.
   ↪ OPFLOWINIT_FROMFILE)
>>> opflow.get_initialization_type()
OPFLOWInitializationType.OPFLOWINIT_FROMFILE
```

Set via getter function

```
>>> types = opflow.get_objective_types()
[<OPFLOWObjectiveType.MIN_GEN_COST: 0>, <OPFLOWObjectiveType.
   ↪ MIN_GENSETPOINT_DEVIATION: 1>, <OPFLOWObjectiveType.NO_OBJ: 2>]
>>> opflow.set_objective_type(types[0])
>>> opflow.get_objective_type()
OPFLOWObjectiveType.MIN_GEN_COST
```

7.5 History

ExaGO pre v1.1 had optional Python bindings, implemented with ctypes, that could be enabled. HiOp [6, 5], a critical dependency, updated to Umpire v6 when GPU and RAJA options were enabled. Because Umpire after v6 ships with CUDA device code, a final device link step was required for any other libraries or executables. This drastically complicated the ctypes Python bindings, which relied on calling out to the shared library directly.

ExaGO v1.2.1 was the first version to ship Python bindings that used Pybind11, which creates a shared library directly importable from Python, which simplified the user experience and made it possible to call ExaGO from Python when only static libraries are generated.

Appendices

Appendix A

Symbol reference

Units of measurement are given in Table (A.1), indices and index sets in Table (A.2), subsets in Table (A.3), special set elements in Table (A.4), and real-valued parameters in Table (A.5).

Table A.1: Units of measurement

Symbol	Description
1	dimensionless. Dimensionless real number quantities are indicated by a unit of 1.
USD	US dollar. Cost, penalty, and objective values are expressed in USD.
h	hour. Time is expressed in h.
pu	per unit. Voltage magnitude is expressed in a per unit system under given base values, and the unit is denoted by pu
rad	radian. Voltage angles are expressed in rad.
MW	megawatt. Real power is expressed in MW.
MVar	megavolt-ampere-reactive. Reactive power is expressed in MVar.
MVA	megavolt-ampere. Apparent power is expressed in MVA.
MW at 1 pu	megawatt at unit voltage. Conductance is expressed in MW at 1 pu, meaning the conductance is such as to yield a real power flow equal to the indicated amount when the voltage is equal to 1 pu
MVar at 1 pu	megavolt-ampere-reactive at unit voltage. Susceptance is expressed in MVar at 1 pu, meaning the susceptance is such as to yield a reactive power flow equal to the indicated amount when the voltage is equal to 1 pu

Table A.2: Index sets

Symbol	Description
$a \in A$	areas
$i \in I$	buses

Table A.2: Continued

Symbol	Description
$j \in J$	bus-connected grid components, i.e. loads, shunts, generators, stochastic resources, branches
$k \in K$	security contingencies, i.e. NERC $(n-1)$- or $(n-k)$-style contingencies, different from the severe event we are modeling
$s \in S$	stochastic scenarios
$t \in T$	time periods

Table A.3: Subsets

Symbol	Description
$J^{\text{gen}} \subset J$	generators
$J^{\text{ld}} \subset J$	loads
$J^{\text{br}} \subset J$	branches, i.e. lines, transformers
$J^{\text{sh}} \subset J$	shunts
$J_i^{\text{o}} \subset J$	branches with origin bus at bus i
$J_i^{\text{d}} \subset J$	branches with destination bus at bus i

Table A.4: Special set elements

Symbol	Description
$a_i \in A$	area of bus i
$i_j^{\text{d}} \in I$	destination bus of branch $j \in J$
$i_j^{\text{o}} \in I$	origin bus of branch $j \in J^{\text{br}}$

Table A.5: Real-valued parameters

Symbol	Description
$b_j^{\max}$	maximum susceptance for shunt $j \in J^{\text{sh}}$ (MVar at 1 pu)

Table A.5: Continued

Symbol	Description
$b_j^{\min}$	minimum susceptance for shunt $j \in J^{\text{sh}}$ (MVar at 1 pu)
b_j^{ch}	charging susceptance for branch $j \in J^{\text{br}}$ (MVar at 1 pu)
b_j^{s}	series susceptance for branch $j \in J^{\text{br}}$ (MVar at 1 pu)
p_j^{gset}	real power set point of generator $j \in J^{\text{gen}}$ (MW)
p_j^{gmax}	maximum real power output for generator $j \in J^{\text{gen}}$ (MW)
p_j^{gmin}	minimum real power output for generator $j \in J^{\text{gen}}$ (MW)
p_j^{r}	maximum ramp rate for generator $j \in J^{\text{gen}}$ (MW/h)
q_j^{gmax}	maximum reactive power output for generator $j \in J^{\text{gen}}$ (MVar)
q_j^{gmin}	minimum reactive power output for generator $j \in J^{\text{gen}}$ (MVar)
$v_i^{\max}$	maximum voltage magnitude for bus $i \in I$ (pu)
$v_i^{\min}$	minimum voltage magnitude for bus $i \in I$ (pu)
π_s	probability of scenario s (1)

Bibliography

[1] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.12, Argonne National Laboratory, 2019.

[2] D. Beckingsale, M. Mcfadden, J. Dahm, R. Pankajakshan, and R. Hornung. Umpire: Application-focused management and coordination of complex hierarchical memory. *IBM Journal of Research and Development*, 2019.

[3] David A Beckingsale, Jason Burmark, Rich Hornung, Holger Jones, William Killian, Adam J Kunen, Olga Pearce, Peter Robinson, Brian S Ryujin, and Thomas RW Scogland. Raja: Portable performance for large-scale scientific applications. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 71–81. IEEE, 2019.

[4] David A Beckingsale, Marty J Mcfadden, Johann PS Dahm, Ramesh Pankajakshan, and Richard D Hornung. Umpire: Application-focused management and coordination of complex hierarchical memory. *IBM Journal of Research and Development*, 64(3/4):00–1, 2019.

[5] C. G. Petra, I.Aravena Solis, N. Gawande, V. Amatya, A. Li, J. Li, S. Abhyankar, S. Peles, M. Schanen, K. Kim, A. Maldonado, M. Anitescu. ExaSGD - Optimization grid dynamics at Exascale, 2020.

[6] Cosmin Petra. HiOP User Guide version 0.3, 2017.

[7] Andreas Waechter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.