powsybl-open-loadflow

Release 2.4.0-SNAPSHOT

Author name not set

Jul 09, 2026

CONTENTS

CHAPTER

ONE

POWSYBL VS POWSYBL OPEN LOAD FLOW

PowSyBl Open Load Flow provides:

- An open-source implementation of the LoadFlow API from PowSyBl Core, supporting DC and AC calculations.
- An open-source implementation of the SecurityAnalysis API from PowSyBl Core, supporting DC and AC calculations.
- An open-source implementation of the SensitivityAnalysis API from PowSyBl Core, supporting DC and AC calculations.

Most of the code is written in Java. It only relies on native code for the KLU sparse linear solver. Linux, Windows and MacOS are supported. KLU is distributed with license LGPL-2.1+.

1.1 Common features

The AC calculations are based on full Newton-Raphson algorithm. The DC calculations are based on direct current linear approximation. Open Load Flow relies on:

- Fast and robust convergence, based on KLU sparse solver.
- Distributed slack (on generators, on loads, or on conform loads); Manual or automatic slack bus selection as explained *here*.
- Support of generators' active and reactive power limits, including the support of reactive capability curves.
- 5 voltage initialization modes: flat, warm, angles-only based on a DC load flow, magnitude-only initialization based on a specific initializer, or both voltages angle and magnitude initialization based on the two previous methods.
- Support of zero impedance branches, including complex zero impedance subnetworks, particularly important in case of voltage controls and topology changes involved in contingencies or in remedial actions.
- Multiple synchronous component calculation, generally linked to HVDC lines.
- Modeling of secondary voltage control following research of Balthazar Donon, Liège University.
- Support of asymmetrical calculations.
- Implementation of three methods to update the state vector in the Newton-Raphson algorithm: classic, rescaling under maximum voltage change and linear search rescaling.

1.2 About controls

Open Load Flow supports:

- Generator and static var compensator voltage remote control through PQV bus modeling. It supports any kind of shared voltage control between controllers that can be generators, static var compensators, or VSC converter stations.
- Static var compensator local voltage control with a slope (support the powsybl-core extension VoltagePerReactivePowerControl.
- Local and remote phase control: phase tap changers can regulate active power flows or limit currents at given terminals.
- Local and remote voltage control by transformers, including shared controls.
- Local and remote voltage control by shunts, including shared controls.
- Remote reactive power control of a branch by generators, including shared controls.
- Remote reactive power control of a branch by transformers.

Heterogeneous voltage controls management has become a key feature. All well-modeled voltage controls are kept and managed through a priority and a complex management of zero impedance lines. The generators have the first priority, followed by transformers, and then shunts. In a load flow run, in a controlled bus, only the main voltage control of highest priority controls voltage. When incremental outer loops are used, secondary priorities voltage controls can help generators that have reached reactive limits.

1.2.1 Load flow

Modeling and equations

Grid modeling

Open Load Flow computes power flows from IIDM grid model in bus/view topology. From the view, a very simple network, composed of only buses and branches is created. In the graph vision, we rely on a Π model for branches (lines, transformers, boundary lines, etc.):

- R and X are respectively the real part (resistance) and the imaginary part (reactance) of the complex impedance ;
- G_1 and G_2 are the real parts (conductance) on respectively side 1 and side 2 of the branch ;
- B_1 and B_2 are the imaginary parts (susceptance) on respectively side 1 and side 2 of the branch ;
- A_1 is the angle shifting on side 1, before the series impedance. For classical branches, the default value is zero ;
- ρ_1 is the ratio of voltages between side 2 and side 1, before the series impedance. For classical branches, the default value is 1.

As the Π model is created from IIDM grid modeling that locates its ratio and phase tap changers in side 1, A_2 and ρ_2 are always equal to zero and 1. In case of a branch with voltage or phase control, the Π model becomes an array. See below our model:

HVDC line

Open Load Flow also supports networks with HVDC lines (High Voltage Direct Current lines). An HVDC line is connected to the rest of the AC network through HVDC converter stations, that can be either LCC (Line-Commutated Converter) or VSC (Voltage-Source Converter).

DC detailed model

Additionally, Open Load Flow supports AC-DC load flow formulation with detailed model of DC elements.
There is no limitation on the number of synchronous components and DC components in a single connected component, except if specific load flow parameters are used (see *acDcNetwork parameter documentation*)

AC flows computing

AC flows computing in OpenLoadFLow relies on solving a system of non-linear squared equations, where the unknowns are voltage magnitude and phase angle at each bus of the network, implying that there are $2N$ unknowns where N is the number of buses. There are two equations per network bus, resulting in $2N$ equations. The nature of these 2 equations depends on the type of the bus:

- PQ-bus: active and reactive balance are fixed at the bus,
- PV-bus: active balance and voltage magnitude are fixed at the bus.

Moreover, at the slack bus, the active balance equation is removed and replaced by an equation fixing the voltage phase angle at 0.

Let v_i be the unknown voltage magnitude at bus i. Let ϕ_i be the unknown voltage phase angle at bus i. Equation fixing voltage magnitude to a reference (also called target) is simply written $v_i = V_i^{ref}$. Equation fixing voltage phase angle at slack bus i is: $\phi_i = 0$

To build the active and reactive balance equations, Open Load Flow first expresses active and reactive power flowing from a bus to another through a line:

$$p_{i,j} = \rho_i v_i (G_i \rho_i v_i + Y \rho_i v_i \sin(\Xi) - Y \rho_j v_j \sin(\theta))$$

$$q_{i,j} = \rho_i v_i (-B_i \rho_i v_i + Y \rho_i v_i \cos(\Xi) - Y \rho_j v_j \cos(\theta))$$

Where Y is the magnitude of the line complex admittance $\frac{1}{R+jX}$, and Ξ such that: $R + jX = \frac{1}{Y} e^{j(\frac{\pi}{2} - \Xi)}$. θ satisfies: $\theta = \Xi - A_i + A_j - \phi_i + \phi_j$.

Beware that $p_{i,j}$ is the power that goes out from the bus i.

Therefore, active and reactive balance equations are expressed as:

$$P_i^{in} = \sum_{j \in \delta(i)} p_{i,j}$$

$$Q_i^{in} = \sum_{j \in \delta(i)} q_{i,j}$$

where $\delta(i)$ is the set of buses linked to i in the network graph.

The resulting non-linear system of equations is solved by default via the Newton-Raphson algorithm. The underlying principle of the algorithm is the following:

- It starts at a certain point $x_0 = (v_0, \phi_0)$ as an approximate solution to the system of equations;
- Then, in an iterative fashion, it generates a series $x_1, x_2, .., x_k$ of better approximate solutions to the system of equations;
- These iterates x_k are found by solving a system of equations using local Jacobian matrix $J(v, \phi)$ at the previous point x_{k-1};

See **acSolverType** below for more details.

Other regulation modes

PQ-bus and PV-bus are used to model local voltage magnitude or local reactive power controls. Other controls are supported in OpenLoadFLow:

- Remote voltage control for generators, static var compensators and two and three windings transformers with ratio tap changer. Control shared over several controllers buses is supported ;
- Remote reactive power control for generators ;
- For static var compensator with a voltage set point, the support of a voltage per reactive power control, also called slope, that modifies a bit the local voltage at connection bus. We only support a local control.

Remote voltage control

In our explanation, we have two buses. A generator or more is connected to bus b_1, that is called controller bus. The remote bus b_2, where voltage should reach the target, is called controlled bus. The bus b_1 is no longer a PQ-bus and becomes a P-bus: only active power balance is fixed for that bus. Bus b_2 becomes a PQV-bus, where the voltage magnitude is fixed at the value defined by the voltage control. To resume:

- At controller bus b_1:
  - ${b_1}^{in} = \sum_{j \in v(b_1)} p_{b_1,j}$.
- At controlled bus b_2:
  - $P_{b_2}^{in} = \sum_{j \in v(b_2)} p_{b_2,j}$.
  - $Q_{b_2}^{in} = \sum_{j \in v(b_2)} q_{b_2,j}$.
  - $v_{b_2} = V_{b_1}^{c}$.

Remote reactive power control

A bus b_1 has, through a generator, a remote reactive power control on a branch (i, j). This controller bus is treated as a P-bus: only active power balance is fixed for that bus. The reactive power flowing at side i on line (i, j) is fixed by the control (it could be at side j too). To resume:

- At controller bus b_1:
  - $P_{b_1}^{in} = \sum_{j \in v(b_1)} p_{b_1,j}$.
- At controlled branch (i, j):
  - $q_{i,j} = Q_{b_1}^{c}$.

Local voltage control for a static var compensator with a slope

We only support the simple case where:

- Only one generator controlling voltage is connected to a bus. If other generators are present, they should have a local reactive power control ;
- The control is local ;
- No other generators from other controller buses are controlling the bus where the static var compensator is connected. Let's call it b_1.

In that case only, the voltage equation at bus b_1 is replaced with:

$$v_{b_1} + s \cdot q_{svc} = V_{b_1}^{c}$$

Where s is the slope of the static var compensator.

Computing HVDC power flow

LCC converters

LCC converters can be considered as fixed loads in the load flow. Indeed, on one side of the line is the rectifier station, and on the other side of the line is the inverter station. The active power flows from the rectifier station to the inverter station, is fixed, and equals to a target value P (AC side). The active power flow at each station at AC side is given by:

- $P_{rectifier} = P$
- $P_{inverter} = (1 - LossFactor_{inverter}) * ((1 - LossFactor_{rectifier}) * (P - P_{LineLoss}))$

Power flows are in load sign convention, the active power at the rectifier AC terminal is positive and the active power at the inverter AC terminal is negative. The HVDC line losses $P_{LineLoss}$ are described in a dedicated section further below.

The reactive power consumption of each converter on AC side is determined by the configured converter power factor of the converter station in the grid model, representing the ratio between active power P and apparent power S. For each converter, its target reactive power is given by:

- $Q = \mid P * \tan(\cos(powerfactor)) \mid$

Note that LCC converters are always absorbing reactive power.

VSC converters

VSC converters are self-commutated converters that can be assimilated to generators in the loadflow. There can be two main modes to control active power flow through the line:

- In **active power setpoint** mode, as for LCC converters, on one side of the line is the rectifier station, and on the other side of the line is the inverter station. The active power flow from the rectifier station to the inverter station is fixed and equals to a target value P (AC side). The active power flow at each station at AC side is given by:
  - $P_{rectifier} = P$
  - $P_{inverter} = (1 - LossFactor_{inverter}) * ((1 - LossFactor_{rectifier}) * (P - P_{LineLoss}))$
- In **AC emulation** mode, the active power flow between both stations is given by: $P = P_0 + k\ (\phi_1 - \phi_2)$ with ϕ_1 and ϕ_2 being the voltage angles at the bus connection for each converter station, and P_0 and k being fixed parameters for the HVDC line. If P is positive, the converter station 1 is controller, else it is converter station 2. For example, if station 1 is controller, the active power flow at each station is given by the formula below (HVDC line losses are described in the next paragraph):
  - $P_{controller} = P_0 + k\ (\phi_1 - \phi_2)$
  - $P_{noncontroller} = (1 - LossFactor_{noncontroller}) * ((1 - LossFactor_{controller}) * (P_0 + k\ (\phi_1 - \phi_2) - P_{LineLoss}))$

The HVDC line losses are described in a dedicated section further below.

In both control modes (active power setpoint mode or in AC emulation mode), the target value P is bounded by a maximum active power P_{max} that can be either:

- the `maxP` configured for the HVDC line,
- or alternatively separate limit values for both directions using the HVDC operator active power range IIDM extension In AC Emulation, these boundaries are handled through the HVDC AC emulation limit outer loop. After solving the equation system, if the computed active power flow through the HVDC overpasses the limits, saturation is applied by the outer loop. Note that this HVDC AC emulation outer loop is only available in AC calculation, and normal DC calculation (not available in DC Woodbury Security analysis and DC Woodbury Sensitivity Analysis).

The reactive power flow on each side of the line depends on whether voltage regulation of the converters is enabled. If the voltage regulation is enabled, then the VSC converter behaves like a generator regulating the voltage. Otherwise, reactive power of the converter at AC side is given by its reactive power setpoint.

HVDC line losses

In both cases of LCC and VSC, the power flows are impacted by losses of the converter stations. In addition to the converter losses, Joule effect (due to resistance in cable) implies line losses in the HVDC line. The HVDC line losses are calculated assuming nominal DC voltage: $P_{LineLoss} = Ri^2$ with $i = P_1/V$ with R being the HVDC line resistance, P_1 being the active power at the output of the controller station and V being the HVDC nominal voltage (equals 1 per unit).

DC flows computing

The DC flows computing relies on several classical assumptions to build a model where the active power flowing through a line depends linearly on the voltage angles at its ends. In this simple model, reactive power flows and active power losses are totally neglected. The following assumptions are made to ease and speed the computations:

- The voltage magnitude is equal to 1 per unit at each bus,
- The series conductance $G_{i,j}$ of each line (i, j) is neglected, only the series susceptance $B_{i,j}$ is considered,
- The voltage angle difference between two adjacent buses is considered as very small.

Therefore, the power flows from bus i to bus j following the linear expression:

$$P_{i,j} = \frac{\phi_i - \phi_j + A_{i,j}}{X_{i,j}}$$

Where $X_{i,j}$ is the serial reactance of the line (i, j), ϕ_i the voltage angle at bus i and $A_{i,j}$ is the phase angle shifting on side j.

DC flows computing gives a linear grid constraints system. The variables of the system are, for each bus, the voltage angle ϕ. The constraints of the system are the active power balance at each bus, except for the slack bus. The voltage angle at slack bus is set to zero. Therefore, the linear system is composed of N variables and N constraints, where N is the number of buses in the network.

We introduce the linear matrix J of this system that satisfies:

If i is the slack bus,

$$J_{i,i} = 1$$

Else:

$$J_{i,i} = \sum_{j \in \delta(i)} \frac{1}{X_{i,j}}$$

$$J_{i,j} = -\frac{1}{X_{i,j}}, \quad \forall j \in \delta(i)$$

where $\delta(i)$ is the set of buses linked to bus i in the network graph. All other entries of J are zero.

The right-hand-side b of the system satisfied:

If i is the slack bus,

$$b_i = 0$$

Else:

$$b_i = P_i - \sum_{j \in \delta(i)} \frac{A_{i,j}}{X_{i,j}}$$

where $\delta(i)$ is the set of buses linked to bus i in the network graph.

Where P_i is the injection at bus i.

This linear system is resumed by:

$$J\phi = b$$

The grid constraints system takes as variables the voltage angles. Note that the vector b of right-hand sides is linearly computed from the given injections and phase-shifting angles.

To solve this system, we follow the classic approach of the LU matrices decomposition $J = LU$. Hence, by solving the system using LU decomposition, you can compute the voltage angles by giving as data the injections and the phase-shifting angles.

Area Interchange Control

Area Interchange Control consists in having the Load Flow finding a solution where area interchanges are solved to match the input target interchange values. It is supported for both AC and DC Load Flow computations.

The area interchange control feature is optional, can be activated via the *parameter* `areaInterchangeControl` and is performed by an outer loop.

Area Interchange Control is performed using an outer loop, similar in principle to the traditional `SlackDistribution` outer loop. However unlike the `SlackDistribution` outer loop which distributes imbalance over the entire synchronous component (island), the Area Interchange Control outer loop performs an active power distribution over areas (filtered on areas having their type matching the configured *parameter* `areaInterchangeControlAreaType`), in order to have all areas' active power interchanges matching their target interchanges.

The Area Interchange Control outer loop can handle networks where part (or even all) of the buses are not in an area. For networks that have no areas at all, the behaviour is the same as with the distributed slack outer loop - in such case internally the Area Interchange Control outer loop just triggers the Slack Distribution outer loop logic.

Just like other outer loops, the Area Interchange Control outer loop checks whether area imbalance must be distributed:

- If no, the outer loop is stable
- If yes, the outer loop is unstable and a new Newton-Raphson is triggered

Area Interchange Control - algorithm description

The active power is distributed separately on injections (as configured in the parameter `balanceType`) of each area to compensate the area "total mismatch" that is given by:

$$AreaTotalMismatch = Interchange - InterchangeTarget + SlackInjection$$

Where:

- "Interchange" is the sum of the power flows at the boundaries of the area (load sign convention i.e. counted positive for imports).
- "Interchange Target" is the interchange target parameter of the area.
- "Slack Injection" is the active power mismatch of the slack bus(es) present in the area (see *Slack bus mismatch attribution*).

The outer loop iterates until the absolute value of this mismatch is below the configured *parameter* `areaInterchangePMaxMismatch` for all areas.

When it is the case, "interchange only" mismatch is computed for all areas:

$$InterchangeMismatch = Interchange - InterchangeTarget$$

If the absolute value of this mismatch is below the *parameter* `areaInterchangePMaxMismatch` for all areas and the absolute value of slack bus active power mismatch is below the *parameter* `slackBusPMaxMismatch`, then the outer loop declares a stable status, meaning that the interchanges are correct and the slack bus active power is distributed.

If not, the remaining slack bus mismatch is first distributed over the buses that have no area.

If some slack bus mismatch still remains, it is distributed over all the areas (see *Remaining slack bus mismatch distribution*).

Areas validation

There are some cases where areas are considered invalid and will not be considered for the area interchange control:

- Areas without interchange target
- Areas without boundaries
- Areas that have boundaries in multiple synchronous/connected components. If all the boundaries are in the same component but some buses are in different components, only the part in the component of the boundaries will be considered.

In such cases the involved areas are not considered in the Area Interchange Control outer loop, however other valid areas will still be considered.

Interchange flow calculation

In IIDM each area defines the boundary points to be considered in the interchange. IIDM supports two ways of modeling area boundaries:

- either via an equipment terminal,
- or via a BoundaryLine boundary.

In the BoundaryLine case, the flow at the boundary side is considered as it should be, for both unpaired BoundaryLines and BoundaryLines paired in a TieLine.

Slack bus mismatch attribution

Depending on the location of the slack bus(es), the role of distributing the active power mismatch will be attributed based on the following logic:

- If the slack bus is part of an area: the slack power is attributed to the area (see "total mismatch" calculation in *Algorithm description*). Indeed, in this case the slack injection can be seen as an interchange to 'the void' which must be resolved.
- Slack bus has no area:
  - Connected to other bus(es) without area: treated as the slack mismatch of the buses without area
  - Connected to only buses that have an area:
    - All connected branches are boundaries of those areas: Not attributed to anyone, the mismatch will already be present in the interchange mismatch
    - Some connected branches are not declared as boundaries of the areas: Amount of mismatch to distribute is split equally among the areas (added to their "total mismatch")

Remaining slack bus mismatch distribution

This section covers the case where the "total mismatch" of all areas is in [-`areaInterchangePMaxMismatch`;`areaInterchangePMaxMismatch`], but some slack bus active power mismatch remains (even after trying to distribute on buses with no area). This remaining slack bus active power mismatch will be distributed by all areas, each one will get a share of this mismatch to distribute.

This distribution will affect each area's interchange and will not necessarily make it closer to its target. The distribution factor of each area will be computed in a way that minimizes chances of having the area increase its interchange mismatch up to more than *`areaInterchangePMaxMismatch`* in absolute value. So the factor is proportional to the "margin" of active power that the area can distribute while keeping $-areaInterchangePMaxMismatch < AreaTotalMismatch < areaInterchangePMaxMismatch$.

It is computed like this: $Factor = sign(SlackBusMismatch) * AreaTotalMismatch + areaInterchangePMaxMismatch$ Then factors are normalized to have sum of factors equal to 1.

The distribution is iterative (inside the same outer loop iteration). Each area distributes its share, if some areas cannot fully distribute it, they are excluded from this distribution and the remaining slack is distributed among other areas at next iteration. The distribution iterates until all the mismatch has been distributed and fails if all areas cannot distribute anymore but some mismatch remains.

Zero impedance boundary branches

The following applies when the *`lowImpedanceBranchMode`* is set to `REPLACE_BY_ZERO_IMPEDANCE_LINE`. Currently, computations involving zero-impedance branches used as boundary branches are not supported. However, it is still possible to submit network models that include zero-impedance boundary branches. If a terminal of a zero-impedance branch is designated as a boundary, Open LoadFlow will internally assign the branch an impedance value equal to the *`lowImpedanceThreshold`* parameter.

Fast-Decoupled Algorithm

Fast-Decoupled is an algorithm to solve the inner-loop of the load flow problem, like the Newton-Raphson one. It is activated giving the `FAST_DECOUPLED` value to the *`acSolverType`* parameter. The solved equation system is the same as the one solved by Newton-Raphson method. However, the Jacobian matrix used is decomposed into two smaller matrices, decoupling the active power balance equations from voltage magnitudes variations and reactive power balance equations from voltage phases variations.

Note that "Fast-Decoupled" is the academic name for this algorithm and not a statement about its performances versus the Newton-Raphson algorithm.

Method

The Fast-Decoupled method is composed of two parts, one relative to the decoupling, another relative to the speeding of the calculations.

The decoupling is obtained by dividing the Jacobian matrix into two matrices, one relative to voltage phases variables and active power equations, the other relative to voltage magnitudes variables and reactive power equations. Combining those two smaller matrices, one can obtain an approximation of the real Jacobian, where some terms have been discarded.

By approximating more terms on top of the Jacobian structure simplification, the algorithm provides two Jacobian matrices that are constant, with respect to the multiplication by a diagonal matrix. Thus, the LU decomposition of the two Jacobian matrices is done only once, at the start of the first Fast-Decoupled iteration.

Regarding state vector scaling, the Fast-Decoupled uses both personalized max voltage change and line-search routines. Without these routines, the algorithm struggles to converge on realistic large networks, as it has a simplified vision of the impact of the system variables. Note that *`stateVectorScalingMode`* is not taken into account.

Limitations

The current implemented version cannot compute when one of the following parameter is activated:

- *asymmetrical*,
- hvdcAcEmulation

In case where the user has selected both the Fast-Decoupled algorithm and one of this parameter, an exception is triggered.

Users should notice that default parameters are optimized for the Newton-Raphson algorithm, as it is the default one. When the Fast-Decoupled algorithm is used, we recommend these values for some convergence parameters:

- *maxNewtonRaphsonIterations*: 75,
- *lineSearchStateVectorScalingMaxIteration*: 4,
- *lineSearchStateVectorScalingStepFold*: `3/2 = 1.5`.

AC DC flows computing

AC DC flows computing in OpenLoadFLow is similar to AC flows computing, but with AC and DC equations in the same system. The unknowns are voltage magnitude and phase angle for each AC bus, voltage for each DC bus, and active/reactive power for each voltage source converter. Concerning AC side, the equations are the same as in AC flows computing, concerning DC side, the equations induced by DC components are the followings:

DC bus

At least one DC bus must be connected to the ground in each DC network, its potential is therefore set to 0. Therefore, symmetrical configuration are currently not supported. For the others DC buses, each one introduces an equation of current balance: $\sum_i I_i = 0$ where I_i are the currents going out of the DC bus. These terms are introduced by the DC components connected to the DC bus.

DC Line

Each DC line adds one term in both of its two connected DC buses current balance:

$\sum_i I_i + \frac{V_1 - V_2}{R} = 0$ for dcBus1

$\sum_i I_i - \frac{V_1 - V_2}{R} = 0$ for dcBus2

Line Commutated Converter

Line commutated converters are not supported yet by Open Load Flow.

Voltage source converters

Let consider a network that is composed of one AC network, and one DC network. The voltage source converter is the link between AC and DC networks, it is linked to **one** AC bus at one side, and two DC buses at the other side. Please note that converters with a second optional AC terminal are not supported by Open Load Flow.

The converter can control either the power received by the AC network (`P_PCC` control mode) or the voltage between its two DC buses (`V_DC` control mode). At least one of the voltage source converters of the DC network must be in `V_DC` mode. Otherwise, an exception will be thrown.

In addition to the control modes `P_PCC` and `V_DC`, the voltage source converter can be set in two modes :

- Reactive power control mode, in which it imposes the reactive power received from AC to DC, which is 0 by default. In this case, the AC voltage is not fixed.

- Voltage regulator control mode, in which it imposes the voltage at its AC Bus. In this case the reactive power is not fixed.

Warning: At the moment, the active and reactive power control and the AC voltage control are enforced at the converter AC terminal, not at its PCC terminal.

We note P_{AC} the power flow injected by AC network into the converter. So $P_{AC} > 0$ if the power flows from AC to DC and $P_{AC} < 0$ otherwise.

If the converter is in `P_PCC` control mode, we add an equation to impose P_{AC} :

$P_{AC} = P_{Ref}$

Else the converter is in `V_DC` control mode, and we add an equation to impose the voltage between its two DC buses :

$V_1 - V_2 = V_{Ref}$

Similarly, if the converter controls reactive power, we add an equation to impose Q_{AC} :

$Q_{AC} = Q_{Ref}$

Else the converter controls the AC voltage, and we add an equation to impose V_{AC}:

$V_{AC} = V_{Ref}$

On the AC bus, the active and reactive power injected into the converter is added to its power balance. On the DC side, we introduce the variable I_{Conv} which is the current flowing in the converter from dcBus1 to dcBus2. It is added to the current balances of dcBus1 and dcBus2

$\sum_i I_i + I_{Conv} = 0$ for dcBus1

$\sum_i I_i - I_{Conv} = 0$ for dcBus2

Power Equations

The last equation of converters ensures the conservation of power between AC and DC.

$$P_{DC} + P_{AC} = P_{Loss}$$

with:

- P_{AC} the power injected by the AC network into the converter
- $P_{Loss} >= 0$ the converter losses depending on AC current. Its computation is detailed below.
- $P_{DC} = I_{Conv} * (V_1 - V_2)$ the power injected by the DC network into the converter.

If the converter acts as rectifier, AC injects power in DC, thus $P_{DC} < 0$ and $P_{AC} > 0$, so we have :

$$-|P_{DC}| + P_{AC} = P_{Loss}$$

$$|P_{DC}| = |P_{AC}| - P_{Loss}$$

And if the converter acts as inverter, DC injects power in AC, thus $P_{DC} > 0$ and $P_{AC} < 0$, so we have :

$$P_{DC} - |P_{AC}| = P_{Loss}$$

$$|P_{AC}| = |P_{DC}| - P_{Loss}$$

In both cases, there is a loss of power when passing through the converter.

P_{Loss} is defined as : $P_{Loss} = IdleLoss + SwitchingLoss * |I_{Conv}| + ResistiveLoss * I_{Conv}^2$

Idle loss, switching loss and resistive loss are loss factors that depend on the converter.

Using the previous equation of power conservation between AC and DC, we have $$I_{Conv} * (V_1 - V_2) + P_{AC} = IdleLoss + SwitchingLoss * |I_{Conv}| + ResistiveLoss * I_{Conv}^2$$

Configuration

To use PowSyBl Open Load Flow for all power flow computations, you have to configure the `load-flow` module in your configuration file:

```
load-flow:
  default-impl-name: "OpenLoadFlow"
```

Generic parameters

Generic parameters are documented in PowSyBl Core.

The next section details the parameters that are specific to PowSyBl Open LoadFlow.

Specific parameters

voltageInitModeOverride

Additional voltage init modes of PowSyBl Open Load Flow that are not present in PowSyBl Core `voltageInitMode` LoadFlow Parameter:

- `NONE`: no override
- `VOLTAGE_MAGNITUDE`: specific initializer to initialize voltages magnitudes v, leaving $\theta = 0$. Proven useful for unusual input data with transformers rated voltages very far away from bus nominal voltages.
- `FULL_VOLTAGE`: voltages magnitudes v initialized using `VOLTAGE_MAGNITUDE` initializer, θ initialized using a DC load flow.

The default value is `NONE`.

lowImpedanceBranchMode

The `lowImpedanceBranchMode` parameter defines how to deal with low impedance branches (when Z is less than the per-unit *`lowImpedanceThreshold`*). Possible values are:

- Use `REPLACE_BY_ZERO_IMPEDANCE_LINE` if you want to consider low impedance branches as zero impedance branches.
- Use `REPLACE_BY_MIN_IMPEDANCE_LINE` if you want to consider low impedance branches with a value equal to the `lowImpedanceThreshold`.

The default value is `REPLACE_BY_ZERO_IMPEDANCE_LINE`.

lowImpedanceThreshold

The `lowImpedanceThreshold` parameter defines in per-unit the threshold used to identify low impedance branches (when Z is less than the `lowImpedanceThreshold` per-unit threshold). The default value is 10^{-8} and it must be greater than `0`.

slackDistributionFailureBehavior

This option defines the behavior in case the slack distribution fails. Available options are:

- `THROW` if you want an exception to be thrown in case of failure
- `FAIL` if you want the OuterLoopStatus to be `FAILED` in case of failure
- `LEAVE_ON_SLACK_BUS` if you want to leave the remaining slack on the slack bus

- DISTRIBUTE_ON_REFERENCE_GENERATOR if you want to put the slack on the reference generator, disregarding active power limits. There must be a reference generator defined, i.e. *referenceBusSelectionMode* must be GENERATOR_REFERENCE_PRIORITY - otherwise this mode falls back to FAIL mode automatically.

The default value is FAIL.

> **Note**
>
> In versions 1.16.0 and before, the default value was LEAVE_ON_SLACK_BUS.

slackBusSelectionMode

The slackBusSelectionMode parameter defines how to select the slack bus. The three options are available through the configuration file:

- FIRST if you want to choose the first bus of all the network buses.
- NAME if you want to choose a specific bus as the slack bus. In that case, the *slackBusesIds* property has to be filled.
- MOST_MESHED if you want to choose the most meshed bus among buses with the highest nominal voltage as the slack bus. This option is typically required for computation with several synchronous components.
- LARGEST_GENERATOR if you want to choose the bus with the highest total generation capacity as the slack bus.

The default value is MOST_MESHED.

Note that if you want to choose the slack buses that are defined inside the network with a slack terminal extension, you have to set the PowSyBl Core readSlackBus LoadFlow Parameter to true. When readSlackBus is set to true, slackBusSelectionMode is still used and serves as a secondary selection criteria:

- for e.g. synchronous components where no slack terminal extension is present.
- for e.g. synchronous components where more than *maxSlackBusCount* slack terminal extensions are present.

mostMeshedSlackBusSelectorMaxNominalVoltagePercentile

This option is used when *slackBusSelectionMode* is set to MOST_MESHED. It sets the maximum nominal voltage percentile. The default value is 95 and it must be inside the interval [0, 100].

maxSlackBusCount

Number of slack buses to be selected. Setting a value above 1 can help convergence on very large networks with large initial imbalances, where it might be difficult to find a single slack with sufficient branches connected and able to absorb or evacuate the slack power. The default value is 1.

slackBusesIds

The slackBusesIds property is a required property if you choose NAME for property *slackBusSelectionMode*. It defines a prioritized list of buses or voltage levels to be chosen for slack bus selection (as an array, or as a comma or semicolon separated string). The default value is an empty list.

slackBusCountryFilter

The slackBusCountryFilter defines a list of countries where slack bus should be selected (as an array, or as a comma or semicolon separated string). Countries are specified by their alpha-2 code (e.g. FR, BE, DE, ...). The default value is an empty list (any country can be used for slack bus selection).

loadPowerFactorConstant

The `loadPowerFactorConstant` property is an optional boolean property. This property is used in the outer loop that distributes slack on loads if:

- the PowSyBl Core `distributedSlack` LoadFlow Parameter is set to `true`
- the PowSyBl Core `balanceType` property is set to `PROPORTIONAL_TO_LOAD` or `PROPORTIONAL_TO_CONFORM_LOAD`.

The default value is `false`.

If prerequisites fulfilled and `loadPowerFactorConstant` property is set to `true`, the distributed slack outer loop adjusts the load P value and adjusts also the load Q value in order to maintain the power factor as a constant value. At the end of the load flow calculation, P and Q at loads terminals are both updated. Note that the power factor of a load is given by this equation:

$$PowerFactor = \frac{P}{\sqrt{P^2 + Q^2}}$$

Maintaining the power factor constant from an updated active power P' means we have to isolate Q' in this equation :

$$\frac{P}{\sqrt{P^2 + Q^2}} = \frac{P'}{\sqrt{P'^2 + Q'^2}}$$

Finally, a simple rule of three is implemented in the outer loop:

$$Q' = \frac{QP'}{P}$$

If `balanceType` equals to `PROPORTIONAL_TO_LOAD`, the power factor remains constant scaling the global $P0$ and $Q0$ of the load. If `balanceType` equals to `PROPORTIONAL_TO_CONFORM_LOAD`, the power factor remains constant scaling only the variable parts. Thus, we fully rely on load detail extension.

In both cases, slack is not distributed to fictitious loads. A load can be fictitious by setting its boolean attribute `isFictitious` or by having a `loadType` equal to `LoadType.FICTITIOUS`.

The default value for `loadPowerFactorConstant` property is `false`.

slackBusPMaxMismatch

When slack distribution is enabled (PowSyBl Core `distributedSlack` LoadFlow Parameter is set to `true`), this is the threshold below which slack power is considered to be distributed. The default value is `1 MW` and it must be greater or equal to `0 MW`.

areaInterchangeControl

The `areaInterchangeControl` parameter defines if the *area interchange control* outer loop is enabled. If set to `true`, the area interchange control outer loop will be used instead of the slack distribution outer loop. The default value is `false`.

areaInterchangeControlAreaType

Defines the `areaType` of the areas on which the *area interchange control* is applied. Only the areas of the input network that have this type will be considered. The default value is `ControlArea`.

areaInterchangePMaxMismatch

Defines the maximum interchange mismatch tolerance for *area interchange control*. The default value is `2 MW` and it must be greater than `0 MW`.

voltageRemoteControl

The `voltageRemoteControl` parameter defines if the remote control for voltage controllers has to be modeled. If set to false, any existing voltage remote control is converted to a local control, using the generator's *equivalentLocalTargetV* if available, or rescaling the target voltage according to the nominal voltage ratio between the remote regulated bus and the equipment terminal bus. The default value is `true`.

voltageRemoteControlRobustMode

When set to true, the algorithm for remote voltage control is more robust to inconsistent voltage targets that might otherwise cause unrealistic voltage exceptions and cause a loadflow failure. The control of unrealistic voltage is performed after the reactive limit outerloop. In addition, groups with unrealistic voltage when entering the reactive limit outerloop, but that do not exceed their reactive diagram are moved PQ and their reactive injection is set to targetQ. With this parameter set to `true` it is in general possible to set the *`minRealisticVoltage` and `maxRealisticVoltage`* values to 0.8 p.u. and 1.2 p.u.

Note that when this option is set to true, instead of failing with an unrealistic voltage at a controller bus, the load flow may produce a solution where a voltage controlled bus does not reach the target voltage, while voltage controllers still have reactive power margin available. The default value is `true`.

voltagePerReactivePowerControl

Whether simulation of static VAR compensators with voltage control enabled and a slope defined should be enabled (See voltage per reactive power control extension). The default value is `false`.

generatorReactivePowerRemoteControl

Whether simulation of generators reactive power remote control should be enabled (See remote reactive power control). The default value is `false`.

disableInconsistentVoltageControl

If multiple generators (or batteries, or VSC converters, or static VAR compensators) are connected to the same bus but either control different buses, either have different target voltages, then their voltage control is disabled if `disableInconsistentVoltageControl` is set to `true`. If the parameter is set to `false`, then only the control of the first generator is kept and applied to all the other generators. The default value is `false`.

secondaryVoltageControl

Whether simulation of secondary voltage control should be enabled. Modeling of secondary voltage control has been designed to provide a fast, static, approximation of the equilibrium state of the generator reactive power alignment process that controls the voltage of a remote pilot point. This reactive power alignment process typically takes several minutes on the network. The default value is `false`.

Please note that the secondaryVoltageControl implementation has the following limitation: Generators that belongs to a secondary voltage control zone should be in local voltage control only. If secondaryVoltageControl is set to `true`, generators that belongs to a secondary voltage control zone and that are configured for remote voltage control are switched to local voltage control with an initial local target equals to remoteTarget / remoteNominalV * localNominalV.

reactiveLimitsMaxPqPvSwitch

When PowSyBl Core `useReactiveLimits` LoadFlow Parameter is set to `true`, this parameter is used to limit the number of times an equipment performing voltage control is switching from PQ to PV type. After this number of PQ/PV type switch, the equipment will not change PV/PQ type anymore. The default value is `3` and it must be greater or equal to `0`.

forceTargetQInReactiveLimits

When PowSyBl Core `useReactiveLimits` LoadFlow Parameter is set to `true`, this parameter is used to prioritize the reactive power limits over the input target Q when target Q is outside these limits. If set to `true`, if any generator has a target Q which is outside its reactive power limits (for its given target P), then its target Q is overridden by the value of the exceeded limit (minQ or maxQ). The default value is `false`.

extrapolateReactiveLimits

When PowSyBl Core `useReactiveLimits` LoadFlow Parameter is set to `true`, this parameter is used for equipment with reactive limits defined by reactive capability curves. If the target P value is outside the reactive capability curve limits (if it is below lowest P value or above highest P value), the behavior depends on the parameter `extrapolateReactiveLimit` :

- If set to `false`: reactive limits at the given target P are defined by the min Q and max Q at the limit of active power of the capability curve.
- If set to `true`: reactive limits at the given target P are defined by the extrapolation of the reactive limits slope of the reactive capability curve (e.g. if the reactive limits decrease when active power decreases, the reactive limits will continue decreasing at the same rate below the minimal active power of the reactive capability curve)

The default value is `false`.

phaseShifterControlMode

- `CONTINUOUS_WITH_DISCRETISATION`: phase shifter control is solved by the AC solver inner-loop.
- `INCREMENTAL`: phase shifter control is solved in the outer-loop

The default value is `CONTINUOUS_WITH_DISCRETISATION`.

transformerVoltageControlMode

This parameter defines which kind of outer loops is used for transformer voltage controls. We have three kinds of outer loops:

- `WITH_GENERATOR_VOLTAGE_CONTROL` means that a continuous voltage control is performed in the same time as the first generator voltage control outerloop. The final transformer ρ is obtained by rounding to the closest tap position and is frozen for all future outerloops. The control deadband is not taken into account.
- `AFTER_GENERATOR_VOLTAGE_CONTROL` means that a continuous voltage control is performed after each generator voltage control outerloop. The final transformer ρ is obtained by rounding to the closest tap position. The control deadband is taken into account. This mode can be further configured with parameters *transformerVoltageControlUseInitialTapPosition* and *generatorVoltageControlMinNominalVoltage*
- `INCREMENTAL_VOLTAGE_CONTROL` means that an incremental voltage control is used. ρ always corresponds to a tap position. Tap changes using sensitivity computations. The control deadband is taken into account. This mode can be further configured with parameter *incrementalTransformerRatioTapControlOuterLoopMaxTapShift*

The default value is `INCREMENTAL_VOLTAGE_CONTROL`.

> **Note**
>
> For OpenLoadFlow 1.16.0 and before, the default value was `WITH_GENERATOR_VOLTAGE_CONTROL`.

transformerReactivePowerControl

This parameter enables the reactive power control of transformer through a dedicated incremental reactive power control outer loop. The default value is `false`.

incrementalTransformerRatioTapControlOuterLoopMaxTapShift

Maximum number of tap position change during a single iteration of the incremental voltage and/or reactive power control outer loop. Applies when *transformerVoltageControlMode* is set to `INCREMENTAL_VOLTAGE_CONTROL` and *transformerReactivePowerControl* is enabled (`true`). The default value is `3`.

shuntVoltageControlMode

This parameter defines which kind of outer loops is used for the shunt voltage control. We have two kinds of outer loops:

- `WITH_GENERATOR_VOLTAGE_CONTROL` means that a continuous voltage control is performed in the same time as the generator voltage control. Susceptance is finally rounded to the closest section for shunt that are controlling voltage. The control deadband is not taken into account.
- `INCREMENTAL_VOLTAGE_CONTROL` means that an incremental voltage control is used. Susceptance always corresponds to a section. Section changes using sensitivity computations. The control deadband is taken into account. This mode can be further configured with parameter *incrementalShuntControlOuterLoopMaxSectionShift*

The default value is `WITH_GENERATOR_VOLTAGE_CONTROL`.

incrementalShuntControlOuterLoopMaxSectionShift

Maximum number of section position change during a single iteration of the incremental shunt voltage control outer loop. Applies when *shuntVoltageControlMode* is set to `INCREMENTAL_VOLTAGE_CONTROL` and when PowSyBl Core `shuntCompensatorVoltageControlOn` LoadFlow Parameter is set to `true`. The default value is `3`.

svcVoltageMonitoring

Whether simulation of static VAR compensators voltage monitoring should be enabled. The default value is `true`.

acSolverType

AC load flow solver engine. Currently, it can be one of:

- `NEWTON_RAPHSON` is the standard Newton-Raphson algorithm for load flow. Solves linear systems via Sparse LU decomposition (by SuiteSparse);
- `NEWTON_KRYLOV` is also the standard Newton-Raphson algorithm for load flow. Solves linear systems via Krylov subspace methods for indefinite non-symmetric matrices (by Kinsol);
- `FAST_DECOUPLED` solves the load flow equation system decoupling angles from magnitudes and active from reactive power. For more information see *Fast-Decoupled Algorithm*.

The default value is `NEWTON_RAPHSON`.

maxOuterLoopIterations

Maximum number of iterations for the AC solver outer loop. The default value is `20` and it must be greater or equal to 1.

newtonRaphsonStoppingCriteriaType

Stopping criteria used for Newton-Raphson and Fast-Decoupled algorithms.

- `UNIFORM_CRITERIA`: stop when quadratic norm of all mismatches vector is below quadratic norm of mismatches of value `newtonRaphsonConvEpsPerEq`. This criteria is defined by the following formula (for n equations):

$$\sqrt{mismatch_1^2 + mismatch_2^2 + ... + mismatch_n^2} < \sqrt{n * newtonRaphsonConvEpsPerEq^2}$$

`newtonRaphsonConvEpsPerEq` defines the corresponding threshold for all equation types, in per-unit. The default value is 10^{-4}p.u. and must be greater than 0.

- `PER_EQUATION_TYPE_CRITERIA`: stop when equation mismatches are below equation type specific thresholds:
  - `maxActivePowerMismatch`: Defines the threshold for active power equations, in MW. The default value is 10^{-2}MW and it must be greater than 0.
  - `maxReactivePowerMismatch`: Defines the threshold for reactive power equations, in MVAr. The default value is 10^{-2}MVAr and it must be greater than 0.
  - `maxVoltageMismatch`: Defines the threshold for voltage equations, in per-unit. The default value is 10^{-4}p.u. and it must be greater than 0.
  - `maxAngleMismatch`: Defines the threshold for angle equations, in radians. The default value is 10^{-5}rad and it must be greater than 0.
  - `maxRatioMismatch`: Defines the threshold for ratio equations, unitless. The default value is 10^{-5} and it must be greater than 0.
  - `maxSusceptanceMismatch`: Defines the threshold for susceptance equations, in per-unit. The default value is 10^{-4}p.u. and it must be greater than 0.

The default value is `UNIFORM_CRITERIA`.

maxNewtonRaphsonIterations

Only applies if *acSolverType* is `NEWTON_RAPHSON` or `FAST_DECOUPLED`. Maximum number of iterations for solver inner loop. The default value is `15` and it must be greater or equal to 1.

maxNewtonKrylovIterations

Only applies if *acSolverType* is `NEWTON_KRYLOV`. Maximum number of iterations for Newton-Raphson inner loop. The default value is `100` and it must be greater or equal to 1.

stateVectorScalingMode

Only applies if *acSolverType* is `NEWTON_RAPHSON`. This parameter 'slows down' the Newton-Raphson by scaling the state vector between iterations. Can help convergence in some cases.

- `NONE`: no scaling is made
- `LINE_SEARCH`: applies a line search strategy
- `MAX_VOLTAGE_CHANGE`: scale by limiting voltage updates to a maximum amplitude p.u. and a maximum angle.

The default value is `NONE`.

lineSearchStateVectorScalingMaxIteration

Only applies:

- If *acSolverType* is NEWTON_RAPHSON and if *stateVectorScalingMode* is LINE_SEARCH,
- Or if *acSolverType* is FAST_DECOUPLED.

Maximum iterations for a vector scaling when applying a line search strategy. The default value is 10 and it must be greater or equal to 1.

lineSearchStateVectorScalingStepFold

Only applies:

- If *acSolverType* is NEWTON_RAPHSON and if *stateVectorScalingMode* is LINE_SEARCH,
- Or if *acSolverType* is FAST_DECOUPLED.

At the iteration i of vector scaling with the line search strategy, with this parameter having the value s , the step size will be $\mu = \frac{1}{s^i}$. The default value is `4/3 = 1.333` and it must be greater than 1.

maxVoltageChangeStateVectorScalingMaxDv

Only applies:

- If *acSolverType* is NEWTON_RAPHSON and if *stateVectorScalingMode* is MAX_VOLTAGE_CHANGE,
- Or if *acSolverType* is FAST_DECOUPLED.

Maximum amplitude p.u. for a voltage change. The default value is `0.1 p.u.` and it must be greater than 0.

maxVoltageChangeStateVectorScalingMaxDphi

Only applies:

- If *acSolverType* is NEWTON_RAPHSON and if *stateVectorScalingMode* is MAX_VOLTAGE_CHANGE,
- Or if *acSolverType* is FAST_DECOUPLED.

Maximum angle for a voltage change. The default value is `0.174533 radians` (10°) and it must be greater than 0.

newtonKrylovLineSearch

Only applies if *acSolverType* is NEWTON_KRYLOV. Activates or deactivates line search for the Newton-Raphson Kinsol solver. The default value is `false`.

plausibleActivePowerLimit

The `plausibleActivePowerLimit` parameter defines a maximal active power limit for generators to be considered as participating elements for:

- slack distribution (if PowSyBl Core balanceType property equals to any of the PROPORTIONAL_TO_GENERATION_<any> types)
- slack selection (if *slackBusSelectionMode* equals to LARGEST_GENERATOR)

The default value is $10000MW$.

> **Note**
>
> In versions 1.16.0 and before the default value was $5000MW$.

minPlausibleTargetVoltage and maxPlausibleTargetVoltage

Equipments with voltage regulation target voltage outside these per-unit thresholds are considered suspect and are discarded from regulation prior to load flow resolution. The default values are `0.8` and `1.2` p.u. and they must be greater or equal to `0`.

minRealisticVoltage and maxRealisticVoltage

These parameters are used to identify if the AC Solver has converged to an unrealistic state.

If there is any bus in a voltage level, in p.u., higher than *minNominalVoltageRealisticVoltageCheck*, with a computed voltage outside these per-unit thresholds, the component solution is deemed unrealistic and its solution status is flagged as failed.

If *voltageRemoteControlRobustMode* is set to true, the check of unrealistic voltage is done after the ReactiveLimits outerloop has been used. In addition, the ReactiveLimits outerloop uses these values as a criteria to block PQ remote controller buses that have an unrealistic voltage and a reactive injection within their reactive diagram.

The default values are `0.5` and `2.0` and they must be greater or equal to `0`.

minNominalVoltageRealisticVoltageCheck

This parameter defines the minimal nominal voltage, in kV, for which a bus outside *minRealisticVoltage and maxRealisticVoltage* will stop the simulation.

Unrealistic voltages -particularly in high-voltage substations- can trigger automated protections or other hazardous phenomena, potentially causing significant impacts to the system. Steady-state simulation may not accurately account for these effects. Therefore, results obtained under such conditions should be interpreted with caution.

A configuration that offers both reliable simulation results and resilience to local observability issues could, for example, use 0.8 and 1.2 p.u. for *minRealisticVoltage and maxRealisticVoltage*, respectively, and 100 kV for `minNominalVoltageRealisticVoltageCheck`.

The default value is 0 kV.

reactiveRangeCheckMode

Open Load Flow discards voltage control for network elements with a too small reactive power range, because in practice a too small reactive power ranger means limited to zero voltage control capability. The involved network element types are:

- Generators
- Batteries
- Voltage Source Converters
- The optional generation part of a Boundary Line
- Static VAR compensators

For a given active power output, the reactive power range is defined as $MaxQ - MinQ$ (always a positive value). The *maximum* and *minimum* reactive range of a network element is:

- for network elements without reactive limits: infinity

- for network elements with reactive limits defined by a pair of min/max values, both minimum and maximum reactive range are equal to $MaxQ - MinQ$
- for network elements with reactive limits defined by a reactive capability curve, the minimum (resp. maximum) reactive range is obtained by finding the curve point having the minimum (resp. maximum) $MaxQ - MinQ$.
- In the case of Static VAR compensators, the reactive power range is derived from maximum and minimum susceptance assuming nominal voltage: $(B_{\text{max}} - B_{\text{min}}) \cdot \text{nominalV}^2$

The `reactiveRangeCheckMode` parameter defines how network elements reactive power range is to be tested. If the test does not pass, meaning the reactive power range is too small, then the voltage control is disabled:

- `MIN_MAX` mode tests if the minimum reactive range is not `0 MVAr` and if the maximum reactive range is above `1 MVAr`.
- `MAX` mode tests if the maximum reactive range is above `1 MVAr`.
- `TARGET_P` tests if the reactive range at $TargetP$ is above `1 MVAr`.

The default value is `MAX`.

reportedFeatures

This parameter allows to define a set of features which should generate additional reports (as an array, or as a comma or semicolon separated string).

This parameter can be used to request details about data inconsistency at network loading through the value :

- `NETWORK_LOADING`: currently, this feature adds information at network loading step, about generators that are discarded from voltage control or active power control

This parameter can also be used to request AC solver iterations report (if *acSolverType* is `NEWTON_RAPHSON` or `FAST_DECOUPLED`) :

- `NEWTON_RAPHSON_LOAD_FLOW`: report AC solver iteration log for load flow calculations.
- `NEWTON_RAPHSON_SECURITY_ANALYSIS`: report AC solver iteration log for security analysis calculations.
- `NEWTON_RAPHSON_SENSITIVITY_ANALYSIS`: report AC solver iteration log for sensitivity analysis calculations.

AC solver iterations report consist in reporting:

- the involved synchronous component
- the involved AC solver outer loop iteration
- for each AC solver inner loop iteration:
  - maximum active power mismatch, the related bus ID with current solved voltage magnitude and angle.
  - maximum reactive power mismatch, the related bus ID with current solved voltage magnitude and angle.
  - maximum voltage control mismatch, the related bus ID with current solved voltage magnitude and angle.
  - the norm of the mismatch vector

The default value is an empty set of features to report.

networkCacheEnabled

This parameter is used to run fast simulations by applying incremental modifications on the network directly to the Open Load Flow internal modeling. The cache mode allows faster runs when modifications on the network are light. Not all modifications types are supported yet, currently supported modifications are:

- target voltage modification

- switch open/close status modification. The switches to be modified must be configured via the *actionableSwitchesIds* property (as an array, or as a comma or semicolon separated string).

The default value is `false`.

actionableSwitchesIds

This parameter list is used if *networkCachedEnabled* is activated. It defines a list of switches that might be modified (as an array, or as a comma or semicolon separated string). When one of the switches changes its status (open/close) and a load flow is run just after, the cache will be used to a faster resolution. Note that in the implementation, all the switches of that list will be considered as retained, leading to a size increase of the Jacobian matrix. The list should have a reasonable size, otherwise the simulation without cache use should be preferred.

alwaysUpdateNetwork

Update the IIDM network state even in case of non-convergence. The default value is `false`.

debugDir

Allows to dump debug files to a specific directory. The default value is undefined (`null`), disabling any debug files writing.

asymmetrical

Allows to run asymmetrical calculations. The default value is `false`.

useActiveLimits

Allows to ignore active power limits during calculations. Active power limits are mainly involved in slack distribution on generators. The default value is `true`.

disableVoltageControlOfGeneratorsOutsideActivePowerLimits

Disables voltage control for generators with `targetP` outside the interval [`minP`, `maxP`]. The default value is `false`.

minNominalVoltageTargetVoltageCheck

This parameter defines the minimal nominal voltage, in kV, for which the plausible voltage target checks are applied.

Above this voltage level, voltage targets that are, in p.u., outside *minPlausibleTargetVoltage and maxPlausibleTargetVoltage* are ignored.

The default value is `20 kV`. It must be greater or equal to `0 kV`.

reactivePowerDispatchMode

This parameter defines how reactive power is split among network elements with controls (voltage or reactive power). It tries to divide reactive power among network elements in the order described below. `reactivePowerDispatchMode` can be one of:

- `Q_EQUAL_PROPORTION`
  1. If all concerned network elements are generators having pre-defined reactive keys via the Coordinated Reactive Control extension, then it splits `Q` proportional to reactive keys
  2. If they don't, but they have plausible reactive limits, split proportionally to the maximum reactive power range (see *reactiveRangeCheckMode* parameter for definition)

3. If they don't, split `Q` equally

- `K_EQUAL_PROPORTION`
  1. If network elements have plausible reactive limits, split `Q` proportionally to `k`, where `k` is defined by $k = \frac{2qToDispatch - qmax1 - qmin1 - qmax2 - qmin2 - ...}{qmax1 - qmin1 + qmax2 - qmin2 + ...}$
  2. If they don't, split `Q` equally

The default value is `Q_EQUAL_PROPORTION`.

outerLoopNames

> **Note**
>
> This is an advanced parameter. Unless you have specific needs, use the default outer loop order built-in PowSyBl Open Load Flow.

This parameter allows to configure both the list of outer loops that can be executed and their explicit execution order. Each outer loop name specified in the list must be unique and match the `NAME` attribute of the respective outer loop.

By default, this parameter is set to `null`, and the activated outer loops are executed in a default order (as defined in `DefaultAcOuterLoopConfig` for AC Load Flow and `DefaultDcOuterLoopConfig` for DC Load Flow).

Here an example with slack distribution and reactive limits consideration, in AC Load Flow, when `outerLoopNames` is **not** used:

- the creation of the `DistributedSlack` *outerloop* is driven by the parameter `distributedSlack` *parameter*.
- the creation of the `ReactiveLimits` *outerloop* is driven by the parameter `useReactiveLimits` *parameter*.
- PowSyBl Open Load Flow default order is to run the `DistributedSlack` outerloop first, and then the `ReactiveLimits` outerloop.

Continuing this example, the outer loops creation and execution order can be modified by setting `outerLoopNames` to value `['ReactiveLimits', 'DistributedSlack']`, in this case PowSyBl Open Load Flow will the `ReactiveLimits` outerloop first, and then the `DistributedSlack` outerloop.

For AC load flow the supported outer loop names, their default execution order, and their corresponding high-level parameter, are:

1. `DistributedSlack` / `AreaInterchangeControl` (parameters: `distributedSlack` / *areaInterchangeControl*)
2. `SecondaryVoltageControl` (parameter: *secondaryVoltageControl*)
3. `VoltageMonitoring` (parameter: *svcVoltageMonitoring*)
4. `ReactiveLimits` (parameter: `useReactiveLimits`)
5. `PhaseControl` / `IncrementalPhaseControl` (parameters: `phaseShifterRegulationOn` and *phaseShifterControlMode*)
6. `SimpleTransformerVoltageControl` / `TransformerVoltageControl` / `IncrementalTransformerVoltageControl` (parameters: `transformerVoltageControlOn` and *transformerVoltageControlMode*)
7. `IncrementalTransformerReactivePowerControl` (parameter: *transformerReactivePowerControl*)
8. `ShuntVoltageControl` / `IncrementalShuntVoltageControl` (parameters: `shuntCompensatorVoltageControlOn` and *shuntVoltageControlMode*)

9. `AutomationSystem` (parameter: *simulateAutomationSystems*)

And for DC load flow:

1. `IncrementalPhaseControl` (parameter: `phaseShifterRegulationOn`)
2. `AreaInterchangeControl` (parameter: *areaInterchangeControl*)

linePerUnitMode

This parameter defines how lines ending in different nominal voltages at both sides are per unit-ed. `linePerUnitMode` can be one of:

- `IMPEDANCE`: handle difference in nominal voltage via impedance correction
- `RATIO`: handle difference in nominal voltage by introducing a ratio

The default value is `IMPEDANCE`.

useLoadModel

When set to `true`, this parameter enables the modeling of the `ZIP` or `EXPONENTIAL` response characteristic of a Load. The default value is `false`.

dcApproximationType

This parameter defines how resistance is neglected compared to inductance in DC approximation. `dcApproximationType` can be one of:

- `IGNORE_R`: consider that r << x
- `IGNORE_G`: consider that g << b

The default value is `IGNORE_R`.

simulateAutomationSystems

Allows to simulate automation systems that are modeled in the network. For the moment, the grid model only supports overload management systems. The default value is `false`.

referenceBusSelectionMode

The reference bus is the bus where the angle is equal to zero. There are several mode of selection:

- `FIRST_SLACK`: the angle reference bus is selected as the first slack bus among potentially multiple slacks (in case *maxSlackBusCount* > 1).
- `GENERATOR_REFERENCE_PRIORITY`: the angle reference bus is selected from generator reference priorities defined via the .

The default value is `FIRST_SLACK`.

writeReferenceTerminals

This parameter allows to write to the IIDM network the Reference Terminals extension containing the generator terminals used as angle reference in the load flow calculation. There is one Terminal created/added in the extension for each calculated Synchronous Component. Works only when *referenceBusSelectionMode* is set to `GENERATOR_REFERENCE_PRIORITY`.

voltageTargetPriorities

When multiple equipment regulate the same bus with different voltage targets, this parameter enables configuring priority to resolve inconsistencies by aligning the voltage targets. Priority is determined by equipment type order; the voltage target of the equipment type listed first takes precedence over those listed later. By default, the order is `["VOLTAGE_SOURCE_CONVERTER", "GENERATOR", "TRANSFORMER", "SHUNT"]`. Note that `"GENERATOR"` indistinctively includes generators, batteries, static var compensators, and VSC HVDC converters.

If the user specifies only a sub-list of priorities, this sub-list is completed by the order defined by default. Thus, if the user specifies only `["TRANSFORMER"]`, it will be completed to `["TRANSFORMER", "VOLTAGE_SOURCE_CONVERTER", "GENERATOR", "SHUNT"]`.

transformerVoltageControlUseInitialTapPosition

This parameter is only used if the transformer voltage control is enabled and of mode `AFTER_GENERATOR_VOLTAGE_CONTROL`. If this parameter is set to `true`, transformers of the same voltage control (meaning controlling the same bus) are merged and the algorithm tries to maintain the initial difference between rhos. If the voltage at controlled bus is closed to the target voltage at initial step of the outer loop, the algorithm blocks transformers at their initial tap positions. The default value of this parameter is `false`.

generatorVoltageControlMinNominalVoltage

This parameter is only used if the transformer voltage control is enabled and of mode `AFTER_GENERATOR_VOLTAGE_CONTROL`. In this mode, at the first step of the outer loop, the transformers that controlled voltage are disabled, only generators are participating to voltage control. At second step, generators controlling voltage at a controlled bus above `generatorVoltageControlMinNominalVoltage` have their voltage control disabled. This parameter overrides an automatic detection of the minimal nominal voltage based on an analysis of nominal voltages controlled by transformers. The default value of this parameter is -1: with this value the parameter is ignored and the outer loop relies only on the automatic detection.

fictitiousGeneratorVoltageControlCheckMode

Specifies the active power checks exemption for fictitious generators voltage control.

PowSyBl Open LoadFlow performs these checks on generators:

- if `targetP` equals zero, voltage control is disabled.
- if parameter *disableVoltageControlOfGeneratorsOutsideActivePowerLimits* is enabled, a generator with a `targetP` lower than `minP` or greater than `maxP`, voltage control is disabled.

The `fictitiousGeneratorVoltageControlCheckMode` option controls whether the above checks must be performed for fictitious generators:

- use mode `FORCED` for an exemption of the two previous checks for fictitious generators.
- Use mode `NORMAL` for no exemption at all, i.e. fictitious generators are processed identically to real generators.

The default mode is `FORCED`.

startWithFrozenACEmulation

If `true`, simulation starts with HVDC links configured in AC emulation frozen at their previous active set point defined by the angles at the HVDC extremities, if defined in the input network. If a solution is found then the simulator continues with the HVDC set to AC emulation mode. Otherwise, the simulation fails.

If `false`, simulation allows HVDC lines to immediately adapt to the new angles.

The value 'true' is typically used to use a load flow to simulate an N-K contingency after a load flow has previously been run.

The default value is `false`. This parameter can be overridden by the security analysis parameter with same name.

generatorsWithZeroMwTargetAreNotStarted

Defines if a generator must be considered as not started when its `targetP` is zero.

When set to `true` (default):

- If the absolute value of `targetP` is zero, the generator is excluded from slack distribution.
- If the absolute value of `targetP` is zero and `minP` is above zero, voltage control is disabled.

When set to `false`:

- Even if `targetP` is zero, the generator may participate in slack distribution.
- Even if `targetP` is zero and `minP` is above zero, voltage control may happen (unless option *`disableVoltageControlOfGeneratorsOutsideActivePowerLimits`* is enabled).

The default value is `true`.

fixVoltageTargets

If true, runs a preprocessing algorithm to identify voltage control settings that may cause convergence issues and removes them automatically from voltage control. This incompatibility is determined by estimating an indicator dv/dz (the difference of target divided by an estimation of the impedance separating the buses). In some situations, in particular when remote voltage control is activated, this could lead to a failure in the Newton-Raphson convergence.

Note that for small impedance values, the indicator value might be affected by the precision limit of double precision computation and, while it always remains high in case of voltage target difference, its precise value may depend on the processor type (for example i686 vs ARM).

> **Note**
>
> In situations where the network configuration can be modified (for example when preparing a forecast network configuration) a preferred solution is to identify problematic voltage settings using the `VoltageTargetChecker.findElementsToDiscardFromVoltageControl` that can be run on a Network. This Java method returns the problematic voltage settings and the reason. Users can then use this result to fix the network data.

The default value is `false`.

acDcNetwork

Defines if the loadflow uses DC detailed equipment and computes an AC DC loadflow.

If `true`, the network supports DC detailed equipments, and the loadflow is computed on the whole connected network, AC and DC sides in the same Jacobian matrix. Currently, the number of synchronous components and DC components is not restricted, except if the `AreaInterchangeControl` outer loop is used. In this case, a connected component shall contain only one synchronous component, but the number of embedded DC components is not restricted.

If `false`, the loadflow is the classic one, without DC detailed components.

The default value is `false`.

Note

With AC-DC network load flow, the following parameters are restricted:

- Generic parameters
  - `dc`: must be set to false.
  - `voltageInitMode` cannot be set to `DC_VALUES`.
  - `componentMode` cannot be set to `MAIN_SYNCHRONOUS`.
- Specific parameters
  - `voltageInitModeOverride` must be set to `NONE`.
  - `acSolverType` must be set to `NEWTON_RAPHSON`.
  - `areaInterchangeControl` must be set to false if a DC component connects two synchronous component.

Moreover, network with the following characteristics are not supported:

- A network containing detailed AC/DC converters with two AC terminals
- A network containing detailed LCC converters

If any of these cases occurs, an exception describing the problem is thrown.

allowNonLinearShuntZeroSection

In IIDM model, Non-Linear Shunt Compensators have an implicit section zero representing the disconnection of the shunt compensator.

This parameter defines whether the implicit section number zero for Non-Linear Shunt Compensators is allowed or not.

If set to `true`, Non-Linear Shunt Compensators can use the implicit section zero.

If set to `false`:

- an input network containing a Non-Linear Shunt Compensator with a zero `sectionCount` results in an exception being thrown.
- voltage regulation by Non-Linear Shunt Compensators will never make use of the section zero, i.e. `solvedSectionCount` will never be zero. To model disconnection, a section with B and G set to zero must be explicitly modeled in the table.

The default value is `true`.

Configuration file example

See below an extract of a config file that could help:

```
open-loadflow-default-parameters:
  lowImpedanceBranchMode: REPLACE_BY_ZERO_IMPEDANCE_LINE
  slackDistributionFailureBehavior: FAIL
  voltageRemoteControl: false
  slackBusSelectionMode: NAME
  slackBusesIds: Bus3_0,Bus5_0
  loadPowerFactorConstant: true
```

The load flow computation implements the Powsybl core loadflow API.

For more details about inputs, outputs, configuration see the Loadflow Powsybl documentation.

1.2.2 Security analysis

Modeling and equations

(work in progress)

Active power slack distribution and Operator Strategies Active Power relative actions

When slack distribution is enabled in the load flow options, then slack distribution is performed in all network simulations of the security analysis, that is:

- in the N ("pre-contingency") simulation, resulting in the creation of the pre-contingency state.
- in the N-1 or N-k ("post-contingency") simulation, performed by applying contingencies on the pre-contingency state, resulting in the creation of the post-contingency state.
- in the Operator Strategies ("curative") simulation, performed by applying actions on the post-contingency state, resulting in the creation of the curative state.

Actions defined in Operators Strategies and involving active power changes (e.g. Load Actions, Generator Actions) can be defined as relative. PowSyBl Open Load Flow considers that the change is relative to the post-contingency state.

For example:

- slack distribution is performed on generators,
- `g1` generator participates in slack distribution,
- An Operator Strategy is defined, where in the occurrence of the `ctg1` contingency `g1` active power output is increased by +20 MW.

Then the following would occur:

- In the N "pre-contingency" state simulation, the generator `g1` solves to a new $initialTargetP(g1, N) = targetP(g1) + distributedSlack(g1, N)$, where:
  - $targetP(g1)$ is `g1` active power setpoint.
  - $distributedSlack(g1, N)$ is `g1` contribution to distributed slack for the balancing of the N state.
- In the N-1 "post-contingency" state simulation of contingency `ctg1`, the generator `g1` solves to a new $initialTargetP(g1, ctg1) = initialTargetP(g1, N) + distributedSlack(g1, cgt1)$, where:
  - $initialTargetP(g1, N)$ is `g1` solved value from above.
  - $distributedSlack(g1, cgt1)$ is `g1` contribution to distributed slack for the balancing of the `cgt1` "post-contingency" state.
- Finally, in the "curative" state of contingency `ctg1`, the generator `g1` will solve to $initialTargetP(g1, ctg1) + 20MW + distributedSlack(g1, curative1)$, where $distributedSlack(g1, curative1)$ is `g1` contribution to distributed slack for the balancing of the "curative" state.

Security analysis on multi components networks

A temporary implementation for multi components analysis has been done to provide a minimal support of this feature. The approach is to execute a security analysis on each separated component and merge the results after all computations are done. However limitations remain until the core security analysis API evolves:

- Convergence status of secondary components are not reported in precontingency status

- Convergence status of contingencies that affect multiple components are not reported (only the first run component's status)
- Actions that change the topology of the network by connecting components not initially connected (using TerminalConnectionAction) are not supported

DC detailed model

Please note that security analysis supports AC-DC networks with DC detailed model only if:

1. The DC components are embedded. Therefore, there is only one synchronous component per connected component.
2. Load flow mode is AC

Please also note that contingencies on DC elements are not supported.

Configuration

To use PowSyBl Open Load Flow for all security analyses, you have to configure the `security-analysis` module in your configuration file:

```
security-analysis:
  default-impl-name: OpenLoadFlow
```

Generic parameters

Generic parameters are documented in PowSyBl Core

The next section details the parameters that are specific to PowSyBl Open Load Flow.

Specific parameters

contingencyPropagation

The `contingencyPropagation` property is applicable only to the portions of the network modeled with a Node/Breaker representation.

Node: In IIDM, the topology modeling style Node/Breaker or Bus/Breaker is defined on a voltage level basis. For most users, all a network is of same topology style, but hybrid representation is also supported.

For Bus/Breaker portions of the network, Security Analysis simulates the outage of only the equipment(s) defined in the contingency.

For Node/Breaker portions of the network:

- when `contingencyPropagation` is set to `false`, Security Analysis will simulate the outage of only the equipment(s) defined in the contingency.
- when `contingencyPropagation` is set to `true` (default) Security Analysis will determine by topological search the switches with type circuit breakers (i.e. capable of opening fault currents) that must be opened to isolate the fault. Depending on the network structure, this could lead to more equipments to be simulated as tripped, because disconnectors and load break switches (i.e., not capable of opening fault currents) are not considered.

The default value is `true`.

createResultExtension

The `createResultExtension` property defines whether Open Load Flow specific results extensions should be created in the security analysis results. Today the available extensions provide information about branches and three-windings transformers voltages (magnitude and angle):

- `OlfBranchResult`
- `OlfThreeWindingsTransformerResult`

The default value is `false`.

threadCount

The `threadCount` property defines the number of threads used to run the security analysis (for both AC and DC). The parallelization is implemented at the contingency level, so the contingency list is split into `threadCount` chunks and each chunk is ran by a different thread.

The thread pool used for getting threads is the one provided by the `ComputationManager` (see `ComputationManager.getExecutor` method). By default, when using the local computation manager, this is the `ForkJoinPool` common pool which is used.

The default value is 1.

dcFastMode

The `dcFastMode` property allows to use fast DC security analysis, based on Woodbury's formula for calculating post-contingency states, when DC mode is activated.

Please note that fast mode has a few limitations:

- Contingencies applied on branches opened on one side are ignored. Also, if a contingency causes the loss of one side of a branch, it is considered completely disabled, and no results are reported for this branch.
- AC emulation of HVDC lines is disabled, as it is not yet supported. Instead, the *active power setpoint* mode is used to control the active power flow through these lines.
- Only PST and topological (except for transformer closing) remedial actions are supported for now.
- Slack relocation following the application of a contingency is not supported. As a result, security analysis is carried out only in slack component, and not necessarily in the largest one.
- Customizing the way contingency imbalances are compensated via `contingencyActivePowerLossDistribution` parameter is not supported.

The default value is `false`.

contingencyActivePowerLossDistribution

> **Note**: This is an advanced parameter. Unless you have specific needs, do not modify this parameter.

Specifies the name of the plugin to be used for compensating any active power injection that has been disconnected by a contingency.

The default value is `Default`.

The `Default` plugin, when slack distribution or area interchange control is enabled:

- distributes the active power imbalance according to the configured BalanceType
- reports how much active power has been disconnected by the contingency, and how much has been distributed

PowSyBl Open LoadFlow does not provide today additional plugins. To create your own plugin, see the *programming guide*.

startWithFrozenACEmulation

If `true`, contingency simulation starts with HVDC links configured in AC emulation frozen at their previous active set point defined by the angles at the HVDC extremities in the base case. If a solution is found then the simulator continues with the HVDC set to AC emulation mode. Otherwise, the contingency simulation fails.

If `false`, contingency simulation allows HVDC lines to immediately adapt to the new angles.

This parameter overrides the loadflow parameter with the same name.

The default value is `true`

Configuration file example

See below an extract of a config file that could help:

```
open-security-analysis-default-parameters:
  contingencyPropagation: true
  createResultExtension: false
  threadCount: 1
  dcFastMode: false
  contingencyActivePowerLossDistribution: Default
```

Contingency Load Flow Parameters

A specific set of load flow parameters can be configured for each contingency individually.

These parameters correspond directly to the parameters in the `LoadFlowParameters` from powsybl-core API and the *`OpenLoadFlowParameters`* specific parameters:

- `distributedSlack`: Refer to `distributedSlack` in powsybl-core
- `areaInterchangeControl`: Refer to *`areaInterchangeControl` in powsybl-open-loadflow*
- `balanceType`: Refer to `balanceType` in powsybl-core
- `outerLoopNames` : Refer to *`outerLoopNames` in powsybl-open-loadflow*

To customize these parameters for a contingency, add to the `Contingency` object a `ContingencyLoadFlowParameters` extension where you may configure the parameters.

The behavior is as follows:

- When the extension is added: The specified parameters override the corresponding SA input parameters.
- When the extension is absent: The load flow parameters provided in the SA input parameters are applied.

When operator strategies are defined for a contingency, the `scope` attribute of `ContingencyLoadFlowParameters` defines whether the overridden parameters also apply to the operator strategy simulations:

- `CONTINGENCY_AND_OPERATOR_STRATEGY` (default): The overridden load flow parameters are applied to both the contingency itself and any operator associated with it.
- `CONTINGENCY_ONLY`: The overridden load flow parameters are applied only to the contingency and not to the associated operator strategies.

This extension does not override any parameter in case of a sensitivity analysis.

Inputs

To run a PowSyBl Open Load Flow security analysis, you have to precise some inputs that are documented in PowSyBl Core.

Implemented remedial actions

With Open Load Flow only the following remedial actions are currently implemented:

- `LoadAction`
- `SwitchAction`
- `TerminalsConnectionAction` (Only supporting actions on branches and three windings transformers terminals)
- `PhaseTapChangerTapPositionAction`
- `RatioTapChangerTapPositionAction`
- `ShuntCompensatorPositionAction`
- `GeneratorAction`
- `HvdcAction`
- `AreaInterchangeTargetAction`

Note: Some limitations in the use of these actions exist, please read the documentation about the *security analysis specific parameters*.

The security analysis implements the Powsybl core security analysis API.

For more details about inputs, outputs, configuration see the Security analysis Powsybl documentation.

1.2.3 Sensitivity analysis

Getting started

OpenLoadFlow sensitivity calculation engine implements PowSyBl core sensitivity API. Here is an example of Java code calculating generator active power to branch flow active power sensitivity for all lines of the IEEE 14 network:

```
    Network network = IeeeCdfNetworkFactory.create14();
    List<SensitivityFactor> factors = new ArrayList<>();
    for (Generator g : network.getGenerators()) {
        for (Line l : network.getLines()) {
            factors.add(new SensitivityFactor(SensitivityFunctionType.BRANCH_ACTIVE_
↪POWER_1, l.getId(),
                                              SensitivityVariableType.INJECTION_ACTIVE_
↪POWER, g.getId(),
                                              false, ContingencyContext.all()));
        }
    }
    List<Contingency> contingencies = network.getLineStream().map(l -> Contingency.
↪line(l.getId())).collect(Collectors.toList());
    SensitivityAnalysisParameters parameters = new SensitivityAnalysisParameters();
    SensitivityAnalysisResult result = SensitivityAnalysis.run(network, factors,␣
↪contingencies, parameters);
```

As we can see in this example for each sensitivity factor that we want to compute we need to specify the variable type and the function type. Not all combinations of variable and function types are allowed / implemented in OpenLoadFlow. Some are supported only in DC mode, some others only in AC. Here is a table to summarize supported use cases:

Function \ Variable	INJECTION_	INJECTION_	TRANSFORM	BUS_	HVDC_	TRANSFORM	TRANSFORM	TRANSFORM	SHUNT_	BRAN	BRAN	BRAN	SVC_PI[illegible]NT_TA
BRAN	AC + DC	N/A	AC + DC	N/A	AC + DC	AC + DC	AC + DC	AC + DC	AC	AC	AC	AC	AC
BRAN	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
BRAN	N/A	AC	N/A	AC	N/A	N/A	N/A	N/A	AC	AC	AC	AC	AC
BRAN	AC + DC	N/A	AC + DC	N/A	AC + DC	AC + DC	AC + DC	AC + DC	AC	AC	AC	AC	AC
BRAN	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
BRAN	N/A	AC	N/A	AC	N/A	N/A	N/A	N/A	AC	AC	AC	AC	AC
BRAN	AC + DC	N/A	AC + DC	N/A	AC + DC	AC + DC	AC + DC	AC + DC	AC	AC	AC	AC	AC
BRAN	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
BRAN	N/A	AC	N/A	AC	N/A	N/A	N/A	N/A	AC	AC	AC	AC	AC
BUS_V	N/A	AC	N/A	AC	N/A	N/A	N/A	N/A	AC	AC	AC	AC	AC
BUS_R	N/A	AC	N/A	AC	N/A	N/A	N/A	N/A	AC	AC	AC	AC	AC

*HVDC_LINE_ACTIVE_POWER is used to compute the sensitivity according to the active power set point of the HVDC. That is why it is not supported for HVDCs that are in AC emulation mode: an exception is thrown if the user asks a sensitivity according to an HVDC link with AC emulation (and if hvdcAcEmulation parameter is enabled).

Input function types

Branch sensitivity functions :

BRANCH_ACTIVE_POWER_#, BRANCH_REACTIVE_POWER_# or BRANCH_CURRENT_# are associated to branch objects (lines, transformers, boundary lines, tie lines). The # index corresponding to the side of the studied branch.

- If it is a line, a tie line, or a two windings transformer : The side is 1 or 2.
- If it is a three windings transformer : The side is 1, 2 or 3 corresponding to the studied leg of the transformer.
- If it is a boundary line : The side is 1 (network side) or 2 (boundary side). Note that if the boundary line is paired, only side 1 (network side) can be specified, and the sensitivity function is computed at the corresponding tie line side.

Bus sensitivity functions :

BUS_VOLTAGE or BUS_REACTIVE_POWER are associated to the bus of the given network element.

Input variable types

SHUNT_COMPENSATOR_SUSCEPTANCE

`SHUNT_COMPENSATOR_SUSCEPTANCE` computes the sensitivity of a function with respect to the susceptance B (in S) of a shunt compensator. It is associated to a `ShuntCompensator` network element and is only supported in AC mode.

BRANCH_RESISTANCE, BRANCH_REACTANCE and BRANCH_ADMITTANCE

These variables compute the sensitivity of a function with respect to a series parameter of a branch:

- `BRANCH_RESISTANCE`: the series resistance R (in);
- `BRANCH_REACTANCE`: the series reactance X (in);
- `BRANCH_ADMITTANCE`: the series admittance modulus y = 1 / (R^2 + X^2) (in S), perturbed at constant ksi = atan2(R, X) (i.e. R and X scaled together).

They are associated to a branch network element (a line or a two windings transformer) and are only supported in AC mode. The monitored function can be on the same branch (self sensitivity) or on a different one (cross sensitivity).

SVC_PILOT_POINT_TARGET_VOLTAGE

`SVC_PILOT_POINT_TARGET_VOLTAGE` computes the closed-loop sensitivity of a function with respect to the pilot point target voltage (in kV) of a secondary voltage control (SVC) zone. The variable is identified by the SVC control zone name. The computed sensitivity is closed-loop: it accounts for the SVC re-coordination across all zones (controllers adjust their voltage setpoints to keep the pilot voltages on target and to equalize their reactive levels), exactly as done by the secondary voltage control outer loop. It is only supported in AC mode and requires secondary voltage control to be enabled (`OpenLoadFlowParameters.setSecondaryVoltageControl(true)`).

Modeling and equations

DC sensitivity analysis

A DC sensitivity analysis starts from the DC flows computing described in the *load flow section*. Simple sensitivity analyses are supported as:

- How an injection increase of 1 MW will impact the flow of a branch ;
- How a phase shifting of 1° of a phase tap changer will impact the flow of a branch.

This could be done in a set of branches given by the user.

NOTE DC sensitivity analysis can distribute the slack (if the loadflow parameter **distributedSlack** is set to True), but in case of slack distribution failure, is always in mode LEAVE_ON_SLACK bus.

Sensitivities on the base network

Injection increases could be either an active power increase of a generator $targetP$ or an active power increase of a load $P0$. To get the impact of an injection increase in a given bus into a given branch, we compute the right-hand side b corresponding to an injection of 1 MW at this bus and 0 MW elsewhere. The LU decomposition gives the matrices L and U in order to compute the vector θ. Then it is easy to retrieve the active power flow in the branch.

For example, to get the sensitivity from injection increase in bus i to branch (k, l), we compute b satisfying:

$$\begin{aligned} &\texttt{if } n = i: \\ &\qquad b_n = 1 \\ &\texttt{else} \\ &\qquad b_n = 0 \end{aligned} \tag{1.1}$$

Then, we retrieve the sensitivity with θ calculating:

$$s_{i,kl} = \frac{\theta_k - \theta_l}{X_{k,l}}$$

To get the sensitivity from phase-shifting angle in a given branch to a given branch, we compute the right-hand side b corresponding to an increase of 1° for the phase-shifting angle at this branch and 0 elsewhere. The LU decomposition gives the matrices L and U in order to compute the vector θ. Then it is easy to retrieve the active power flow in the branch.

For example, to get the sensitivity from phase-shifting angle increase in branch (i, j) to branch (k, l), we compute b satisfying:

$$\texttt{if } n = i: \quad b_n = -\frac{\pi}{180 X_{i,j}} \tag{1.5}$$

$$\texttt{if } n = j: \quad b_n = \frac{\pi}{180 X_{i,j}} \tag{1.6}$$

$$\texttt{else}: \quad b_n = 0 \tag{1.7}$$

Then, we retrieve the sensitivity with θ calculating:

$$s_{ij,kl} = \frac{\theta_k - \theta_l}{X_{k,l}}$$

In the special case where $(i, j) = (k, l)$, we retrieve the sensitivity calculating:

$$s_{ij,kl} = \frac{\theta_k - \theta_l + \frac{\pi}{180}}{X_{k,l}}$$

Sensitivities in case of slack distribution

If active power mismatch is distributed on loads or on generators, an injection increase at bus b will be distorted by slack distribution. Thus, the sensitivity analysis must include the sensitivity value correction linked to the effects of the participation units to the slack distribution. Let's introduce the list of participating units $(g \in U)$ involved in the slack distribution and their respective participation factor (r_g^c). Note that at this stage, this list of participating elements is computed from the initial network state and not from its state after the DC flows computation. The new sensitivity values are computed from the base case sensitivity values through the following formula:

$$s^c_{b,kl} = s_{b,kl} - \sum_{g \in U} r^c_g s_{g,kl}$$

We only support for the moment balance type `PROPORTIONAL_TO_GENERATION_P_MAX`, `PROPORTIONAL_TO_GENERATION_P` and `PROPORTIONAL_TO_LOAD`.

Contingency management

The contingency management consists in calculating the sensitivity values for post-contingency states of the network. A post-contingency state of the network is the state of the network after an outage (most of the time, the loss of a line). In the particular case of DC flows approximation, the post-contingency sensitivity values can be computed using the pre-contingency sensitivity values and some flow transfer factors. Thus, the same LU decomposition is used both for the pre-contingency analysis and for the post-contingency analysis.

Loss of a single branch

The most frequent event in the network is the loss of a single branch (including the loss of a transformer with or without tap changer).

Let's introduce $s_{b,ij,mk}$ the sensitivity of an increase of 1 MW at bus b on branch (i, j) where the branch (m, k) represents the outage. We want to compute this sensitivity.

We call b^1 the right-hand side vector corresponding to an increase of 1 MW at bus b. We call θ^1 the state vector of voltage angles obtained by solving the equation system on the pre-contingency network that has as right-hand side b^1.

We call b^2 be the right-hand side vector corresponding to an increase of 1 MW at bus m and a decrease of 1 MW at bus k. We call θ^2 the state vector of voltage angles obtained by solving the equation system on the pre-contingency network that has as right-hand side b^2.

Note that both θ^1 and θ^2 are built using the same LU decomposition of the constraints matrix J.

Let $s_{b,ij}$ be the sensitivity of an increase of 1 MW at bus b on branch (i, j) on the pre-contingency network, we recall that:

$$s_{b,ij} = \frac{\theta_i^1 - \theta_j^1}{X_{i,j}}$$

Let $s_{mk,ij}$ be the sensitivity of an increase of 1 MW at bus m and a decrease of 1 MW at bus k, on the pre-contingency network. It can be easily computed through the formula, valid for all buses:

$$s_{mk,ij} = \frac{\theta_i^2 - \theta_j^2}{X_{i,j}}$$

Then, the post-contingency sensitivity $s_{b,ij,mk}$ satisfies:

$$s_{b,ij,mk} = s_{b,ij} + \frac{\theta_m^1 - \theta_k^1}{X_{m,k} - (\theta_m^2 - \theta_k^2)} s_{mk,ij}$$

Loss of more than one branch

Sometimes, an event in the network causes the loss of several buses and branches. Connected to these lost buses, we can have generators or loads.

Let's introduce $s_{b,ij,E}$ the sensitivity of an increase of 1 MW at bus b on branch (i, j) when the event E occurs. The event E corresponds to the loss of the n branches indexed by $(m_1, k_1), \cdots, (m_n, k_n)$. We want to compute this sensitivity.

We call b^1 the right-hand side vector corresponding to an increase of 1 MW at bus b. We call θ^1 the state vector of voltage angles obtained by solving the equation system on the pre-contingency network that has as right-hand side b^1.

We call b^{p+1} be the right-hand side vector corresponding to an increase of 1 MW at bus m_p and a decrease of 1 MW at bus k_p. We call θ^{p+1} the state vector of voltage angles obtained by solving the equation system on the pre-contingency network that has as right-hand side b^{p+1}.

Then, the post-contingency sensitivity $s_{b,ij,E}$ satisfies:

$$s_{b,ij,E} = s_{b,ij} + \sum_{p=1}^{n} \alpha_p s_{m_p k_p,ij}$$

Where, valid for all buses:

$$s_{m_p k_p,ij} = \frac{\theta_i^{p+1} - \theta_j^{p+1}}{X_{i,j}}$$

The vector of coefficients α is computed as the solution of a linear system of size n:

$$Mx = c$$

Where M is the $n \times n$ matrix defined by:

$$M_{p,q} = -(\theta^{q+1}_{m_p} - \theta^{q+1}_{k_p}) \quad \text{if } p \neq q, \tag{1.11}$$

$$M_{p,q} = X_{m_p,k_p} - (\theta^{p+1}_{m_p} - \theta^{p+1}_{k_p}) \quad \text{else}$$

And c the vector of size n defined by:

$$c_p = \theta^1_{m_p} - \theta^1_{k_p}$$

Loss of network connectivity management

An event can create one or more new synchronous component. In case of the loss of a single branch (when n = 1), it means, mathematically, that the factor $X_{m,k} - (\theta^2_m - \theta^2_k)$ equals to 0. In case of the loss of more than one branch (when n > 1), it means, mathematically, that the matrix M previously defines is not invertible. In that special configuration, previously described coefficients α cannot be computed and used to assess post-contingency sensitivities. Most of the time, the secondary networks are out of voltage, but it is still possible to get the sensitivity values in the largest network containing the slack bus. In real world, it is like the initial network has lost small parts that not contain the slack bus. Thus, the sensitivity values can still be computed.

A sensitivity involves two equipments in the network: a load or a generator, etc. connected to a bus, or a phase tap changer and a monitored branch. These two equipments should be in the same connected component.

Loss of connectivity by a single branch

This case is quite simple to support. Sensitivities computed on the base network are still valid for the post-contingency network as long as they refer to equipments in the connected component containing the slack bus.

Loss of connectivity by more than one branch

A post-contingency network can be divided into several largest connected components. Let's introduce t the number of those components.

Obviously: $t \leq n + 1$

One of this component contains the slack bus and should be the largest. Only sensitivities involving equipments of that component can be computed. A easy way to compute them is to connect lines that have been disconnected during the contingency. More precisely, we have to find $t - 1$ lines to reconnect in order to obtain a single connected component. As we reconnect exactly $t - 1$ lines and obtain a connected network, we are assured that there are no loop in it.

As the two equipments of the sensitivity lie in the largest connected component of the post-contingency network (containing the slack bus), no flow can pass through the reconnected lines. Then, the sensitivities of the post-contingency network are equal to those of the connected network, where the matrix M is invertible.

How to detect a loss of connectivity?

It might be quite time-consuming to assess the connectivity of the post-contingency network using classic graph theory tools. Here we propose another way to detect a loss of connectivity in the post-contingency network. We need to compute the θ^p vectors as described in the section describing the loss of more than one branch. Our method uses those vectors to detect a loss of connectivity.

First, we define and compute:

$$\forall p \in \{1, \ldots, n\}, \quad \sigma_p = \sum_{q \in \{1,\ldots,n\}} \left| \frac{\theta^{p+1}_{m_q} - \theta^{p+1}_{k_q}}{X_{m_q,k_q}} \right|$$

If we can find $p \in \{1, \ldots, n\}$ such as $\sigma_p \geq 1$, then, there is probably a loss of connectivity in the post-contingency network. Else, we are sure that there is no connectivity loss in the post-contingency network.

We compute sequentially the σ_p coefficients and, as soon as one of them is found greater or equal to 1, we stop the process and a classic graph analysis is performed. Most of the time, connectivity is not lost, especially when we loose a single branch (σ_1 is equal to 1).

Slack distribution in case of connectivity loss

In case of connectivity loss, participating elements that are not connected to slack bus are removed from the list of initial participating elements. Then, the participating factor of remaining elements is increased such as their sum remains equal to one.

Extension to reference flow computations

The methodology described below to access to sensitivity values in a post-contingency network can be applied to compute reference flows in the same post-contingency network.

In that case, the vector of right-hand side b_1 is the injections at network buses. All other data or formula are unchanged. Beware that the system $Mx = c$ whose vector α is solution is modified when b_1 is modified, because right-hand side vector c depends on θ_1. However matrix M only depends on which lines are disconnected by the outage.

In case of an outage causing connectivity loss in the post-contingency network, reference flows can be computed only in the largest connected component containing the slack bus. In that case, the right-hand side injection vector b_1 is modified to take into account only injections in the largest connected part of the post-contingency network (all other injections are set to zero). The same methodology of reconnecting some lines to obtain a connected network is used too.

Return codes in case of uncomputable sensitivities

If an user asks for a sensitivity computation which is not possible to perform, Open Load Flow provides different return codes which are listed below:

- If the user requests a sensitivity involving a variable or a function which does not exist in the network: an error is thrown, Open Load Flow terminates.
- If the user requests a sensitivity involving a variable which does not belong to the main connected component after a connectivity loss: a warning is displayed and the sensitivity is not computed.
- If the user requests a sensitivity involving a function which does not belong to the main connected component of after a connectivity loss: the sensitivity is equal to 0. Note that if both variable and function do not belong to the main connected component (where we have the slack bus), the priority is given to the variable: a warning is displayed and the sensitivity is not computed.

AC sensitivity analysis

An AC sensitivity analysis starts from the AC flows computing described in the *load flow section*. Simple sensitivity analyses are supported as:

- How an active injection increase of 1 MW will impact the flow of a branch ;
- How an active injection increase of 1 MW will impact the current of a branch ;
- How a phase shifting of 1° of a phase tap changer will impact the flow of a branch ;
- How a phase shifting of 1° of a phase tap changer will impact the current of a branch.

This could be done in a set of branches given by the user.

Every sensitivity computation is done using the following formula:

$$S_{\eta,p}(v,\phi) = g_p^T(v,\phi) + G_{v,\phi}(v,\phi)^T J(v,\phi)^{-1} f_p(v,\phi)$$

where:

- η is the list of values measured by the sensitivity (i.e. flow on a line or current on a line) ;
- p is the parameter whose variation is studied by the sensitivity ;
- $S_{\eta,p}(v,\phi)$ is a row-vector of sensitivities on values η according to variation of parameter p, at the point (v,ϕ) ;
- $g_p(v,\phi)$ is the gradient of sensitivities according to the parameter p, at the point (v,ϕ) ;
- $G_{v,\phi}(v,\phi)$ is the matrix whose each column is the gradient of a sensitivity according to state variables v and ϕ, at the point (v,ϕ) ;
- $J(v,\phi)$ is the jacobian matrix of power flow equations system at the point (v,ϕ) ;
- $f_p(v,\phi)$ is the gradient of power flow equations according to the parameter p, at the point (v,ϕ).

Giving a list of sensitivities η and a parameter p, an AC sensitivity analysis is done following the steps:

1. Extract from a power flow computation state variables (v,ϕ) ;
2. Compute vector $f_p(v,\phi)$;
3. Compute jacobian matrix $J(v,\phi)$;
4. Compute the LU decomposition of the jacobian ;
5. Compute $J(v,\phi)^{-1} f_p(v,\phi)$ with the LU decomposition ;
6. Compute $G_{v,\phi}(v,\phi)$;
7. Compute $g_p(v,\phi)$;
8. Compute $S_{\eta,p}(v,\phi)$ using the formula.

Following sections detail how $f_p(v,\phi)$, $G_{v,\phi}(v,\phi)$ and $g_p(v,\phi)$ are computed.

Computation of vector $f_p(v,\phi)$

Vector $f_p(v,\phi)$ is the gradient of power flow equations according to the parameter p, at the point (v,ϕ). Its computation depends on if p is the active injection at a bus or the phase shift of a phase tap changer.

Case 1: p is the active injection at bus i

In this case, all entries of vector $f_p(v,\phi)$ are equal to zero, except one which corresponds to the active power flow balance equation at bus i:

$$\texttt{if } k \texttt{ is the active power flow balance at bus } i: \quad (f_p(v,\phi))_k = 1, \tag{1.13}$$

$$\texttt{else}: \quad (f_p(v,\phi))_k = 0 \tag{1.14}$$

Case 2: p is the phase shift of the phase tap changer on side i of line (i,j)

In this case, all entries of vector $f_p(v,\phi)$ are equal to zero, except the following ones:

- active power flow balance equation at both buses i and j,
- reactive power flow balance equation at bus i, only if it is a PQ bus,
- reactive power flow balance equation at bus j, only if it is a PQ bus.

$$\texttt{let } \alpha = -\rho_i v_i Y \rho_j v_j \cos(\theta) \frac{\pi}{180}, \tag{1.16}$$

(1.17)

$$\texttt{let } \beta = \rho_i v_i Y \rho_j v_j \sin(\theta) \frac{\pi}{180} \tag{1.18}$$

$$\texttt{if } k \texttt{ is the active power flow balance at bus } i: \quad (f_p(v,\phi))_k = \alpha, \tag{1.19}$$

(1.20)

$$\texttt{if } k \texttt{ is the active power flow balance at bus } j: \quad (f_p(v,\phi))_k = -\alpha, \tag{1.21}$$

(1.22)

$$\texttt{if } k \texttt{ is the reactive power flow balance at bus } i: \quad (f_p(v,\phi))_k \tag{1.23}$$

(1.24)

$$\texttt{if } k \texttt{ is the reactive power flow balance at bus } j: \quad (f_p(v,\phi))_k = \tag{1.25}$$

(1.26)

$$\texttt{else}: \quad (f_p(v,\phi))_k \tag{1.27}$$

Computation of matrix $G_{v,\phi}(v,\phi)$

The matrix $G_{v,\phi}(v,\phi)$ is the matrix which column is the gradient of a sensitivity according to state variables v and ϕ, at the point (v,ϕ). It is computed column by column. The computation of column l depends on if it is relative to the sensitivity of the power or the current flowing from i to j.

Case 1: Column l is relative to the sensitivity of the power flowing from i to j

In this case, all entries of column l of $G_{v,\phi}(v,\phi)$ are equal to zero, except for those corresponding to the voltage magnitudes and angles at both buses i and j. Let's introduce the three following derivatives:

$$\frac{dp}{dv_i} = \rho_i(2G\rho_i v_i + 2Y\rho_i v_i \sin(\Xi) - Y\rho_j v_j \sin(\theta)) \tag{1.28}$$

(1.29)

$$\frac{dp}{dv_j} = -\rho_i v_i Y \rho_j \tag{1.30}$$

(1.31)

$$\frac{dp}{d\phi_i} = -\rho_i v_i Y \rho_j v_j \tag{1.32}$$

Note that p always corresponds to side 1 of the branch, it is an arbitrary choice. And note that $\frac{dp}{d\phi_j}$ equals to $-\frac{dp}{d\phi_i}$.

$$\texttt{if } k \texttt{ is the voltage magnitude at bus } i: \quad (G_{v,\phi})_{k,l} = \frac{dp}{dv_i}, \tag{1.33}$$

(1.34)

$$\texttt{if } k \texttt{ is the voltage magnitude at bus } j: \quad (G_{v,\phi})_{k,l} = \frac{dp}{dv_j} \tag{1.35}$$

(1.36)

$$\texttt{if } k \texttt{ is the voltage angle at bus } i: \quad (G_{v,\phi})_{k,l} = \frac{dp}{d\phi_i} \tag{1.37}$$

(1.38)

$$\texttt{if } k \texttt{ is the voltage angle at bus } j: \quad (G_{v,\phi})_{k,l} = -\frac{dp}{d\phi_i} \tag{1.39}$$

(1.40)

$$\texttt{else}: \quad (G_{v,\phi})_{k,l} \tag{1.41}$$

Case 2: Column l is relative to the sensitivity of the current flowing from i to j

As previous case, all entries of column l of $G_{v,\phi}(v,\phi)$ equal zero, except these corresponding to the voltage magnitudes and angles at both buses i and j. Let's introduce $Re(I)$ and $Im(I)$ respectively the real and imaginary parts of the complex current flowing from i to j:

$$N_f = \frac{1000}{\sqrt{3}}, \tag{1.42}$$

(1.43)

$$w_i = (1.44)$$

(1.45)

$$w_j = Y (1.46)$$

(1.47)

$$\gamma = \Xi - A_i + A_j \quad (1.48)$$

(1.49)

$$Re(I) = N_f \rho_i (w_i (G_i \cos(\phi_i) - B_i \sin(\phi_i) + Y \sin(\Xi + \phi_i)) - w_j \sin(\quad (1.50)$$

(1.51)

$$Im(I) = N_f \rho_i (w_i (G_i \sin(\phi_i) + B_i \cos(\phi_i) - Y \cos(\Xi + \phi_i)) + w_j \cos(\quad (1.52)$$

(1.53)

$$|I| = \sqrt{Re(I)^2 + Im(\quad (1.54)$$

Which derivatives are the following ones:

$$\frac{dRe(I)}{dv_i} = N_f \rho_i^2 (G_i \cos(\phi_i) - B_i \sin(\phi_i) + Y \sin(\Xi + \phi_i)), \tag{1.55}$$

(1.56)

$$\frac{dRe(I)}{dv_j} = -N_f \rho_i Y \rho_j \sin(\quad (1.57)$$

(1.58)

$$\frac{dRe(I)}{d\phi_i} = N_f \rho_i w_i (-G_i \sin(\phi_i) - B_i \cos(\phi_i) + Y \cos(\Xi + \quad (1.59)$$

(1.60)

$$\frac{dRe(I)}{d\phi_j} = -N_f \rho_i w_j \cos(\quad (1.61)$$

(1.62)

$$\frac{dIm(I)}{dv_i} = N_f \rho_i^2 (G_i \sin(\phi_i) + B_i \cos(\phi_i) - Y \cos(\Xi + \quad (1.63)$$

(1.64)

$$\frac{dIm(I)}{dv_j} = N_f \rho_i Y \rho_j \cos(\quad (1.65)$$

(1.66)

$$\frac{dIm(I)}{d\phi_i} = N_f \rho_i w_i (G_i \cos(\phi_i) - B_i \sin(\phi_i) + Y \sin(\Xi + \quad (1.67)$$

(1.68)

$$\frac{dIm(I)}{d\phi_j} = -N_f \rho_i w_j \sin(\quad (1.69)$$

We obtain:

$$\texttt{if } k \texttt{ is the voltage magnitude at bus } i: \quad (G_{v,\phi})_{k,l} = \frac{Re(I)\frac{dRe(I)}{dv_i} + Im(I)\frac{dIm(I)}{dv_i}}{|I|}, \tag{1.70}$$

(1.71)

$$\texttt{if } k \texttt{ is the voltage magnitude at bus } j: \quad (G_{v,\phi})_{k,l} = \frac{Re(I)\frac{dRe(I)}{dv_j} + Im(I)\frac{dIm(I)}{dv_j}}{|I|}, \tag{1.72}$$

(1.73)

$$\texttt{if } k \texttt{ is the voltage angle at bus } i: \quad (G_{v,\phi})_{k,l} = \frac{Re(I)\frac{dRe(I)}{d\phi_i} + Im(I)\frac{dIm(I)}{d\phi_i}}{|I|}, \tag{1.74}$$

(1.75)

$$\texttt{if } k \texttt{ is the voltage angle at bus } j: \quad (G_{v,\phi})_{k,l} = \frac{Re(I)\frac{dRe(I)}{d\phi_j} + Im(I)\frac{dIm(I)}{d\phi_j}}{|I|}, \tag{1.76}$$

(1.77)

$$\texttt{else}: \quad (G_{v,\phi})_{k,l} = 0. \tag{1.78}$$

Computation of vector $g_p(v, \phi)$

Vector $g_p(v, \phi)$ is the gradient of the sensitivities according to the parameter p, at the point (v, ϕ). First, its computation depends on if p is the active injection at a bus or the phase shift of a phase tap changer. Secondly it depends on if the sensitivity function is the power or the current flowing through a line (i, j).

In the case of an injection as parameter p, vector $g_p(v, \phi)$ equals zero.

In the case of a phase shift of a phase tap changer as parameter p, a component of $g_p(v, \phi)$ is non zero if and only if it is relative to a sensitivity on the branch (i, j) where lies the phase tap changer. In this case, the value of the component is given by:

$$\rho_i v_i Y \rho_j v_j \texttt{cos}(\theta)\frac{\pi}{180}, \quad \texttt{if the sensitivity is on the power flow}, \tag{1.79}$$

(1.80)

$$-\frac{Re(I)\frac{dRe(I)}{d\phi_j} + Im(I)\frac{dIm(I)}{d\phi_j}}{|I|}\frac{\pi}{180}, \quad \texttt{if the sensitivity is on the current flow}. \tag{1.81}$$

Note that the two formulas above are valid when the phase tap changer is on side i of the monitored line (i, j). The opposite values are taken when the equipment is on side j of the line.

Sensitivities in case of slack distribution

Computations of sensitivities in case of a slack distribution work in the same way as in the *DC sensitivity analysis*.

Let's introduce the list of participating elements $(g \in U)$ and their respective participation factor (r_g^c). We recall the formula:

$S_{\eta,p}^c = S_{\eta,p} - \sum_{g \in U} r_g^c S_{\eta,g}$.$

Contingency management

Contrary to *DC sensitivity analysis*, computations of sensitivities in case of contingencies are performed by restarting the sequence of computations based on the equation system of the post-contingency network and not more on the equation system of the pre-contingency network.

Sensitivity involving voltage magnitudes

Open Load Flow allows to compute the sensitivity from an increase of voltage magnitude at a PV-bus to the voltage magnitude at a PQ-bus. This is done using the same formula previously introduced:

$S_{\eta,p}(v,\phi) = g_p^T(v,\phi) + G_{v,\phi}(v,\phi)^T J(v,\phi)^{-1} f_p(v,\phi)$.

Where:

- η is the voltage magnitude at a PQ-bus named i,
- p is the voltage magnitude at a PV-bus named b,
- $f_p(v,\phi)$ is a vector composed of null entries except for the one relative to the voltage magnitude at PV-bus b, that equals 1,
- $G_{v,\phi}(v,\phi)$ is a vector composed of null entries except for the one relative to the voltage magnitude at PQ-bus i, that equals 1,
- $g_p(v,\phi)$ is null.

DC detailed model

Please note that AC-DC networks with DC detailed model are not supported for sensitivity analysis

Configuration

To use PowSyBl Open Load Flow for all sensitivity analyses, you have to configure the `sensitivity-analysis` module in your configuration file:

```
sensitivity-analysis:
  default-impl-name: OpenLoadFlow
```

Generic parameters

Generic parameters are documented in PowSyBl Core

The next section details the parameters that are specific to PowSyBl Open LoadFLow.

Specific parameters

debugDir

Allows to dump debug files to a specific directory. The default value is undefined (`null`), disabling any debug files writing.

startWithFrozenACEmulation

If `true`, contingence simulation starts with HVDC link configured in AC emulation frozen at their previous active set point defined by the angles at the HVDC extremities in the base case. If a solution is found then the simulator continues with the HVDC set to AC emulation mode. Otherwise, the contingence simulation fails.

If `false`, contingence simulation allows HVDC lines to immediately adapt to the new angles.

The default value is `true`

threadCount

The `threadCount` property defines the number of threads used to run an AC sensitivity analysis (multi-threading is currently not supported for DC sensitivity analysis). The parallelization is implemented at the contingency level, so the contingency list is split into `threadCount` chunks and each chunk is ran by a different thread.

The thread pool used for getting threads is the one provided by the `ComputationManager` (see `ComputationManager.getExecutor` method). By default, when using the local computation manager, this is the `ForkJoinPool` common pool which is used.

The default value is 1.

Configuration file example

See below an extract of a config file that could help:

```
open-sensitivityanalysis-default-parameters:
  debugDir: /path/to/debug/dir
```

The sensitivity analysis implements the Powsybl core sensitivity analysis API.

For more details about inputs, outputs, configuration see the Sensitivity analysis Powsybl documentation.

1.2.4 Advanced Programming Guide

This section describes advanced programming topics.

These topics apply only to Open LoadFlow in Java. In particular, this section does not apply to PyPowSyBl.

PowSyBl at large is build with a modular architecture. PowSyBl Open LoadFlow is no different and implements the same principles.

Via this modular design some PowSyBl Open LoadFlow features can be modified or enhanced using plug-ins, without requiring to build a customized/forked version of PowSyBl Open LoadFlow.

This section details PowSyBl Open LoadFlow plug-in capabilities and how to use them.

LfNetwork Loader post-processors

Any simulation performed by PowSyBl Open LoadFlow starts by loading the IIDM Grid Model into PowSyBl Open LoadFlow representation: the `LfNetwork` .

By providing an `LfNetworkLoaderPostProcessor` plug-in, it is possible to further alter the `LfNetwork`. The plug-in is called by PowSyBl Open LoadFlow network loader:

- at each bus, branch, injection, and area creation
- and when the `LfNetwork` loading is complete

In each callback a reference to the original IIDM object is provided, as well as the corresponding object in `LfNetwork`.

Below an example empty post-processor, doing nothing:

```
package org.example.powsybl.plugins;

import com.google.auto.service.AutoService;
import com.powsybl.openloadflow.network.*;

@AutoService(LfNetworkLoaderPostProcessor.class)
public class MyLfNetworkLoaderPostProcessor implements LfNetworkLoaderPostProcessor {
```

(continues on next page)

(continued from previous page)

```
    @Override
    public String getName() {
        return "my-example-plugin";
    }

    @Override
    public LoadingPolicy getLoadingPolicy() {
        return LoadingPolicy.ALWAYS;
    }

    @Override
    public void onBusAdded(Object element, LfBus lfBus) {
        // implement me
    }

    @Override
    public void onBranchAdded(Object element, LfBranch lfBranch) {
        // implement me
    }

    @Override
    public void onInjectionAdded(Object element, LfBus lfBus) {
        // implement me
    }

    @Override
    public void onAreaAdded(Object element, LfArea lfArea) {
        // implement me
    }

    @Override
    public void onLfNetworkLoaded(Object element, LfNetwork lfNetwork) {
        // implement me
    }
}
```

For example, if you have a need to copy some extra data from IIDM elements to LfNetwork elements `PropertyBag` , you may proceed as follows:

```
    @Override
    public void onBranchAdded(Object element, LfBranch lfBranch) {
        if (element instanceof Identifiable<?> identifiable) {
            lfBranch.setProperty("name-or-id", identifiable.getNameOrId());
        }
    }
```

The above example stores the IIDM branch name in the LfBranch as a property having key `name-or-id`.

Note that `PropertyBag`-s are not limited to storing strings: you can attach a complex object behind a property. One practical use case of doing so is to attach to the LfNetwork specific information needed to run custom Outer-Loops. Refer to *Outer-Loop Configuration*.

Note that besides attaching properties, manipulating the created LfNetwork elements attributes directly is another

perfectly valid use case for the `LfNetworkLoaderPostProcessor`. Just remember: with great power comes great responsibilities.

External AC Solvers

PowSyBl Open LoadFlow provides out-of-the box two AC Solvers:

- Newton-Raphson
- Newton-Krylov

Other AC solvers can be plugged into PowSyBl Open LoadFlow with the following interfaces that you would need to implement:

- `AcSolver`: the solver itself
- `AcSolverParameters`: any additional parameter that you need specifically for your solver
- `AcSolverFactory`: Responsible for creating `AcSolver` instances and `AcSolverParameters`. Provide your own implementation of `AcSolverFactory` and make it available to the Java ServiceLoader. The `getName()` method should provide the plugin name - which can then be used in the `acSolverType` *Load Flow parameter*

PowSyBl Open LoadFlow uses the same plugin mechanism internally. For more details you may have a look at:

- `NewtonRaphson` / `NewtonRaphsonParameters` / `NewtonRaphsonFactory`
- `NewtonKrylov` / `NewtonKrylovParameters` / `NewtonKrylovFactory`

Load Flow Outer-Loops configuration

PowSyBl Open LoadFlow provides out of the box two mechanisms for configuring Outer-Loops via *Load Flow parameters*:

- Option 1: using "basic" parameters: This is the standard usage. For example setting `useReactiveLimits` will configure an OuterLoop dedicated to handling reactive power limits of generators, static VAR compensators, etc…
- Option 2: using the more advanced `outerLoopNames` parameter, allowing to tune not only the outer-loops creation, but also specifying a specific order for OuterLoops execution.

Refer to the *`outerLoopNames` Load Flow parameter documentation* for more details about these two standard ways of configuring Outer Loops.

For even more advanced usage, it is possible to plug your own Outer-Loops configuration into PowSyBl Open LoadFlow. To do so, you will have to provide via the Java Service Loader your own implementation of the following interfaces, defined in package `com.powsybl.openloadflow.lf.outerloop.config`:

- `AcOuterLoopConfig` for AC LoadFlow
- `DcOuterLoopConfig` for DC LoadFlow

In fact, PowSyBl Open LoadFlow internally is using these interfaces for the two options described at the beginning of this chapter:

- if `outerLoopNames` parameter is blank, PowSyBl Open LoadFlow uses its own internal implementation called `DefaultAcOuterLoopConfig` / `DefaultDcOuterLoopConfig`
- if `outerLoopNames` parameter is used, PowSyBl Open LoadFlow uses its own internal implementation called `ExplicitAcOuterLoopConfig` / `ExplicitDcOuterLoopConfig`

If an alternative implementation of `AcOuterLoopConfig` / `DcOuterLoopConfig`is provided via the Java Service Loader, it will be used as a replacement of the Open LoadFlow implementations above. The use cases typically are:

- customizing the creation and/or ordering of the PowSyBl Open LoadFlow provided Outer-Loops

- creating new Outer-Loop types/implementations in your own private code, by implementing new `AcOuterLoop` / `DcOuterLoop`, then instantiating these Outer-Loops in your custom Outer-Loops configuration.

Some tips:

- instead of re-creating your own Outer-Loop configuration implementation from scratch, you could instead benefit from existing classes and extend them as needed, e.g. for AC Load Flow Outer-Loops configuration you may find it easier to extend either `AbstractAcOuterLoopConfig` or `DefaultAcOuterLoopConfig`.
- If you are creating new Outer-Loop types/implementations, typically you are going to need more data from the network. This is a typical use case of the *LfNetwork Loader PostProcessors*

Contingency Active Power Loss Distribution

This plugin applies only to Security Analysis.

The way active power imbalances created by contingencies (disconnection of loads, of generators, ...) are compensated can be tailored to specific needs through a plugin.

The plugin to be used can be specified in the `contingencyActivePowerLossDistribution` security analysis *parameter*.

PowSyBl Open LoadFlow provides only a `Default` plugin, but other plugins can be added.

To add another plugin, you will need to code (in Java) an implementation of the `ContingencyActivePowerLossDistribution` interface and make this implementation available to the Java ServiceLoader (e.g. using Google's AutoService):

- the `getName()` method should provide the plugin name - which can then be used in the `contingencyActivePowerLossDistribution` security analysis parameter
- the `run(...)` method will be called by the security analysis engine for each contingency and should provide the logic. This method must return the amount of distributed active power (in per-unit) and has access to:
  - the network
  - the contingency in open-loadflow representation, including among others information about disconnected network elements, and how much active power has been lost.
  - the contingency definition
  - the security analysis parameters
  - the contingency load flow parameters overrides if any (See *Contingency Load Flow Parameters*)
  - the contingency report node - so that the plugin may add any report message needed.