InterPSS

Network Analysis Function Development Guide

Version: V1.0
Date: 07/01/2016

Document Version History

Version No.	Release Date	Description	Prepared By
1.0.0	7/1/2016	first version	m. zhou

Table of Contents

1. Introduction

This document discuss InterPSS network analysis functionality, including DCLF, Sensitivity analysis, which is designed to be used to develop application such as SCED.

2. DCLF and Sensitivity Analysis

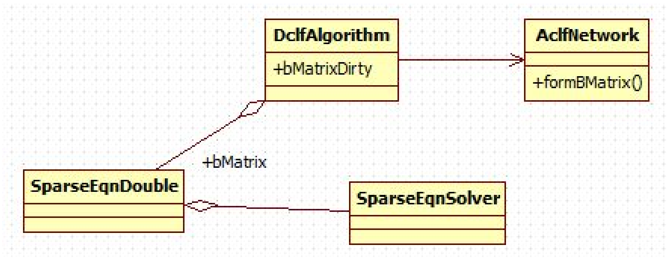
This section discusses InterPSS object models used in Dclf and sensitivity analysis.

DCLF is performed by solving the following eqn:

$$\Delta P = [B]' \Delta\vartheta$$

- The [B]’ matrix is constructed by the IAclfNetFormMatrix.formB1Matrix() method, implemented in the AclfNetworkImpl class

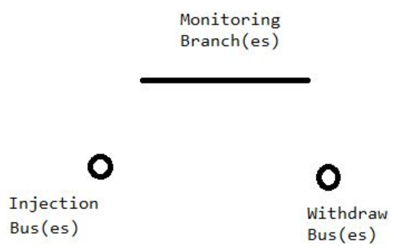
DclfAlgorithm Object Model
The following is a high-level architecture view of the Dclf algorithm class.



- DclfAlgorithm has reference to an AclfNetwork object, which has a formBMatrix() method. B matrix is stored in a SparseEqn object.
- DclfAlgorithm calls the formBMatrix() method to form the matrix and caches the SparseEqn object. There is a bMatrixDirty attribute to track the status of B matrix. If bMatrixDirty is false, DclfAlgorithm will call the method again to refresh the cached SparseEqn object.
- SparseEqn contains a SparseEqnSolver object, which actually performs matrix manipulation, for example, LU decomposition, and caches the factorization table. After the SparseEqn object is LU-factorized, the forward/backward process to solve [A][x] = [b] is multi-thread safe, meaning one can solve the eqn for multiple [b] in parallel.

2.1 Sensitivity Analysis

Three basic concepts are used in sensitivity analysis, as shown in the following figure:



- Injection Bus: one or more injection buses can defined. In the case of multiple injection buses, a set of distribution factors need to be defined.

- Withdraw Bus: one or more withdraw buses can defined. In the case of multiple withdraw buses, a set of distribution factors need to be defined. Withdraw bus is optional. If withdraw bus is not defined, network reference bus is used as the withdraw bus. Network swing bus is by default the reference bus. However, user can change it.
- Monitoring Branch : one or more monitoring branches can be defined for GSF, TPDF calculation.

Sample code to calculate PTDF (power transfer distribution factor) :

```
// create a Dclf algorithm object and run Dclf. The network [B] will be
// created during the DCLF calculation process and cached for later usage.
DclfAlgorithm algo = DclfObjectFactory.createDclfAlgorithm(net);
algo.calculateDclf();

String injectBus = "Bus9822";
String withdrawBus = "Bus9823";
AclfBranch monitorBranch = net.getBranch("Bus8617->Bus8610(1)");
double ptdf = algo.pTransferDistFactor(injectBus, withdrawBus, monitorBranch);
```

2.1.1 Sensitivity Analysis DSL

InterPSS DCLF and Sensitivity analysis DSL (Domain Specific Language) implementation IpssDclf.java wraps the PTDF calculation method and provides a more high-level API abstraction, such as GSF(generator shiting factor), LODF(line outage distribution factor).

- **GSF**

The following are some GSF calculation sample code.

```
// create a AclfNetwork object
AclfNetwork net = ...

// create a DSL object and calculate DCLF
DclfAlgorithmDSL algoDsl = IpssDclf.createDclfAlgorithm(net);
algoDsl.algo().calculateDclf();

// define injection bus and withdraw bus
algoDsl.injectionBusId("Bus2")
        .withdrawBusId("Bus3");

// calculate GSF
double gsf = algoDsl.monitorBranch("Bus2", "Bus3", "1")
                    .genShiftFactor();
```

- **LODF**

The following are some LODF calculation sample code.

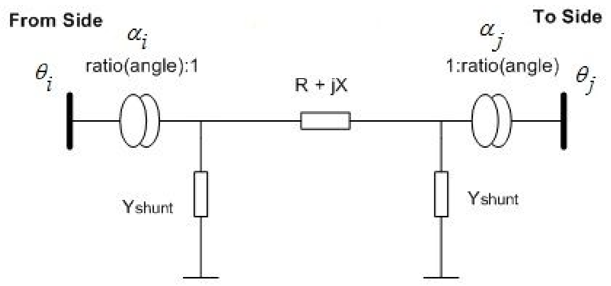
```
// create a AclfNetwork object
AclfNetwork net = ...
```

```
        // create a DSL object and calculate DCLF
        DclfAlgorithmDSL algoDsl = IpssDclf.createDclfAlgorithm(net);
        algoDsl.algo().calculateDclf();

        // define outage branch
        algoDsl.outageBranch("Bus4", "Bus7", "1");

        // calculate LODF
        double lodf = algoDsl.monitorBranch("Bus4", "Bus9", "1")
                              .lineOutageDFactor();
```

More samples could be found in the ipss.pssl.test project.

2.2 PsXfr Model

InterPSS PSXfr model is shown in the following figure:



$$P_i = b_{ij}\sin(\vartheta_i - \vartheta_j - \phi_i + \phi_j)$$
$$= b_{ij}(\vartheta_i - \vartheta_j - \phi_i + \phi_j)$$

Therefore

$$b_{ij}(\vartheta_i - \vartheta_j) = P_i + b_{ij}(\phi_i - \phi_j)$$

Assumption:

$$(\vartheta_i - \vartheta_j - \phi_i + \phi_j) \sim 0.0$$

Therefore, we can use the PsXfr component $b_{ij}(\phi_i - \phi_j)$ to augment bus injection during the Dclf calculation process.

2.2.1 PsXfr Angle Shift Factor

For a PsXfr (i->j), the branch power from bus i to bus j can be expressed as follows:

$$Pij = (\theta i - \phi ij - \theta j) / X$$

And the branch power from bus j to bus i,

$$Pji = (\theta j + \phi ij - \theta i) / X$$

The impact of phase shifting transformer can be modeled as a pair of additional power injections to bus i (ϕij/XT) and to bus j (-ϕij/XT). Therefore we have the following incremental relationship:

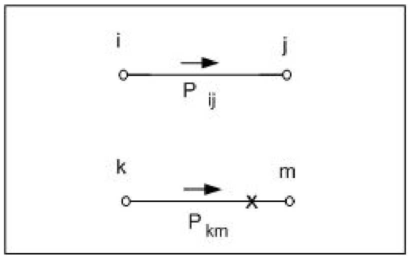
$$\partial P_{km} = GSF_{km,i} \times (\partial \phi_{ij} / X) - GSF_{km,j} \times (\partial \phi_{ij} / X)$$

Or

$$\partial P_{km} / \partial \phi_{ij} = [GSF_{km,i} - GSF_{km,j}] / X$$

2.3 Outage Analysis

In outage analysis, the goal is to calculate power flow increase on a monitoring branch (i->j in figure below), as the result of a branch outage (k->m in the figure below).

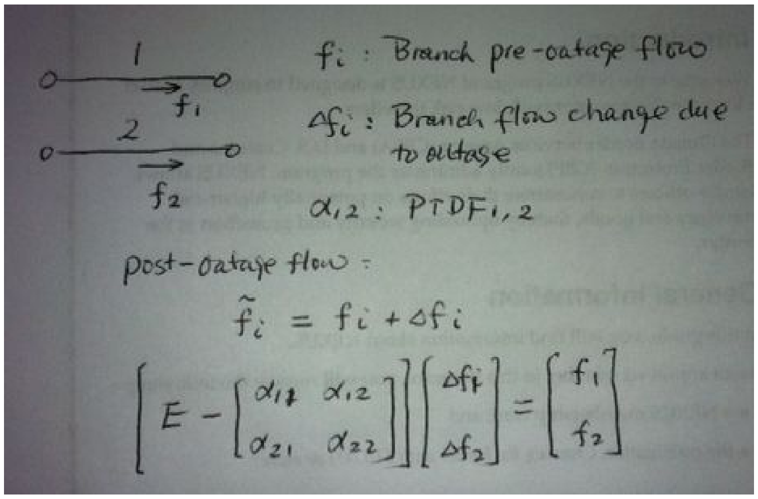


The following are main steps in outage analysis:

1. Run Dclf to calculate pre-outage power flow;
2. Calculate LODF (Line Outage Distribution Factor) of the monitoring branch with regarding to the outage branch;
3. P_{ij} (after outage) = P_{ij} (before outage) + LODF x P_{km} (before outage)

2.3.1 Multi-line Outage

InterPSS multi-line outage is based the algorithm presented in an IEEE paper. The key concept is summarized in the following eqn:



f_i : Branch pre-outage flow

Δf_i : Branch flow change due to outage

α_{12} : PTDF$_{1,2}$

post-outage flow :

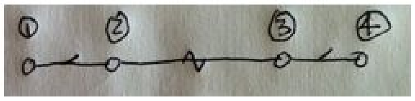
$$\tilde{f}_i = f_i + \Delta f_i$$

$$\left[E - \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} \right] \begin{bmatrix} \Delta f_1 \\ \Delta f_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

In multiple-line outage contingency analysis, if there is islanding, as the result of the outages, the [E-TPTD] matrix might be singular.

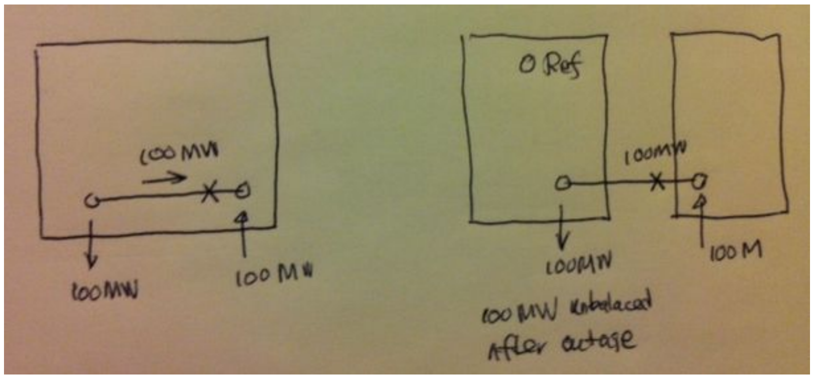
The following rules are applied in building the equivalent outage branches:

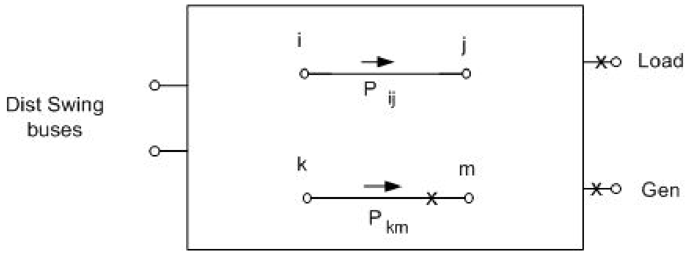
- **Rule-1** : Disable those outage branches connecting between island buses.
- **Rule-2** : If an island has more than one active outage branches, one of the branches needs to be relaxed (disabled).



In the above figure, if we open break 1->2 and 3->4 to open line 2->3, the same result could be achieved mathematically by just opening break 1->2, or break 3->4. Or we can relax an outage branch in the contingency analysis.

2.3.2 Islanding Due to Outage





Outage may result in islanding of some generator buses and/or load buses, as shown in the above figure. The loss of gen/load is usually balanced by a group of distributed generator (Swing) buses. Therefore, if there are islanding gen/load, the following eqn should be used:

$$\begin{aligned} P_{ij}\text{ (after outage)} = {} & P_{ij}\text{ (before outage)} + \\ & \text{LODF} \times [\, P_{km}\text{ (before outage)} + \Delta P_{km}\text{ (loss of gen/load)} \,] + \\ & \sum \text{GSF} \times P\text{ (loss of gen/load)} \end{aligned}$$

Where P (gen/load) is loss of gen/load due to islanding caused by contingency outage.

2.3.3 Branch Closure Outage

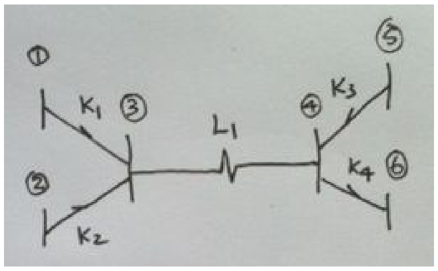
Branch outage could be of type OPEN or CLOSE. A CLOSE branch outage refers to closure of an out-of-service branch in pre-contingency. Branch closure distribution factor formulation can be found Here. A contingency could contain both OPEN and CLOSE branch outages. A generic formulation for branch outage and closure document was created by ISO NE. InterPSS current branch closure outage implementation is based on the document.

PTDF of a monitoring is defined as the power flow, when 1.0 power is injected at an injection bus and withdrawn from a withdraw bus. In theory, a PTDF of an pre-contingency open branch is equals to zero. However, for estimating branch flow due to opening of other branches on the closure branch, an equivalent PTDF should be used:

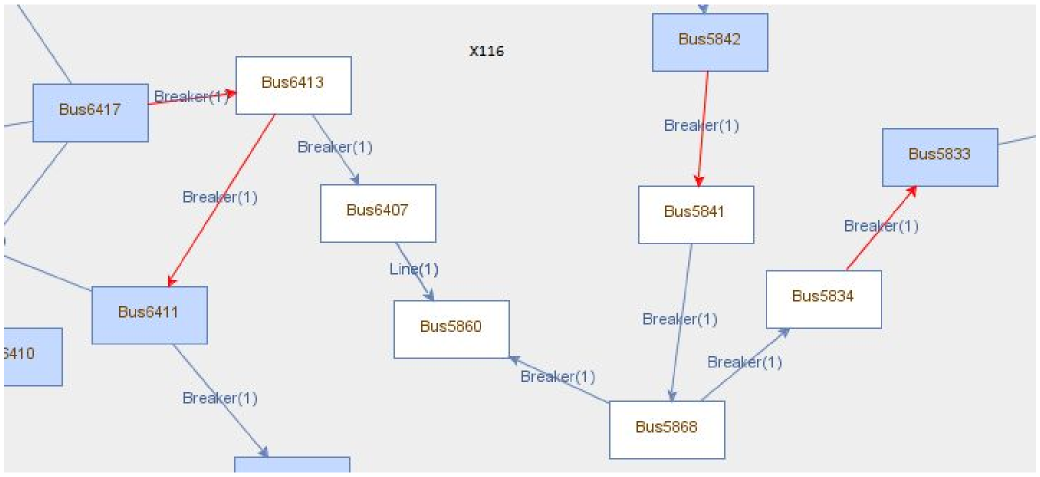
$$PTDF_{ij} = V_{i,mn} - V_{j,mn} / X_{ij}$$

Where, PTDF of branch j->j is expressed by an injection at bus-m and withdraw at bus-n.

Appendix-A Equivalent Outage Branch



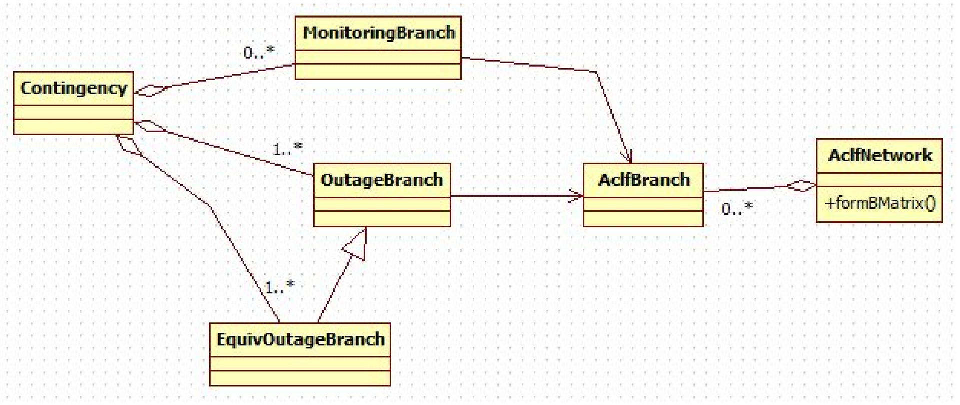
In the above diagram, a contingency to open breakers K_1 - K_4 is intended to disconnect Line L_1. However, from contingency analysis perspective, the CA results of opening K_1 - K_4 and opening Line L_1 are different. When opening L_1, Bus1 and Bus2, Bus5 and Bus6 are connected, while opening K_1 - K_4 would result Bus1 and Bus2, Bus5 and Bus6 disconnected. Therefore, from CA perspective, Bus/Branch model and Node/Breaker model does make difference.



```
Outage of the 4 Breakers : Shifted flow on Bus5869->Bus5842 = -0.4734581
Outage of Line 6407->5860: Shifted flow on Bus5869->Bus5842 = -0.3903599
```

Appendix-B Contingency Analysis

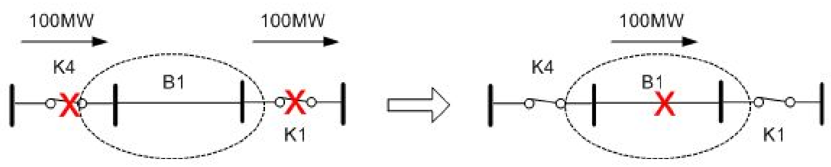
Contingency concept is represented by a Contingency class with the following relationship:



- A contingency contains a set of outage branches. Outage branches are defined in a contingency file associated with the contingency. The following is a sample contingency record:

```
DATA (CONTINGENCY, [CTGLabel,CTGSkip,CTGProc,CTGSolved,LoadMW,CustomString])
{
 "001A" "NO " "NO" "NO" 0.0 "ctg1"
 <SUBDATA CTGElement>
 "BRANCH 8 9 1 OPEN" "" CHECK //line bus8-bus9
 </SUBDATA>
 }
```

- Equivalent outage branch is represented by the EquivOutageBranch class.



As shown in the above figure, opening the two breakers is equivalent to opening the line B1. The equivalent outage branch concept is intended to model the equivalent outage.

Please Note: The equivalent outage branch is currently not fully implemented yet. At the moment, it is used to represent all active outage branches.

- Monitoring branch is for store contingency analysis result, for example, branch shifted flow, branch loading percentage due to contingency outages. Contingency monitoring branch can be pre-defined or constraint branches calculated by specifying branch violation threshold.

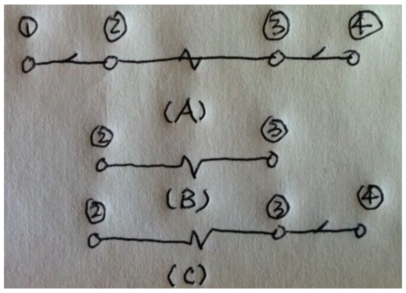
The following is a summary of the contingency analysis implementation.

- **Step-1 :** For a hour, apply economic dispatch result and outage/override schedule for the hour to the network basecase AclfNetwork object, by calling the following method:

```
        model.setHour(hr);
        .
```

- Outage/override schedule needs to be applied first, before applying the ED Gen/Load, since bus gen and/or load might be turned off by the outage/override schedule;
- Both AclfNetwork object and Contingency objects need to be rollback to its basecase state.

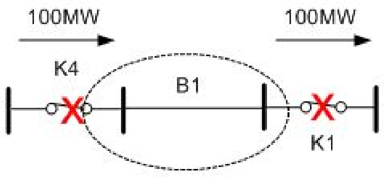
- **Step-2:** (Not implemented currently) Consolidate the basecase network using the small-z branch processing algorithm. During the consolidation process, all contingency outage branches are preserved.



For example, in the above figure, (A) represents un-consolidated basecase network. Breaker 1->2 and breaker 3->4 should be consolidated during the consolidation process, as shown in (B) without considering contingency. However, if breaker 3->4 is a contingency outage branch, it could be preserved during the consolidation process.

- **Step-3:** Identify island buses for a contingency, see Section-3.2.1 for more details. The step should be done before the equivalent outage branch processing.

- **Step-4:** Compute equivalent outage branch.



It is often to open the two breakers to open a line, as shown in the above figure. Opening the two breakers will result in the B1 line and the two terminal buses as an island. For an island with n outage branches, one needs to be relaxed (disabled) to make the [I - PTDF] matrix non-singular, see more discussion in Section-6.2.1.

- **Step-5**: Contingency analysis.
  The following are key steps to performance contingency analysis.

```
// (1) Calculate Dclf
AclfNetwork net = model.getAclfNet();
DclfAlgorithmDSL algoDsl = IpssDclf.createDclfAlgorithm(net, true)
                    .runDclfAnalysis();

// (2) Contingency analysis
ContingencyAnalysisHelper contHelper =
             new ContingencyAnalysisHelper(algoDsl, true /* findContraintBranches */);
```

```
        contHelper.setViolationThreshold(1.0);

        Contingency cont = model.getContingency("Y151");
        contHelper.contAnalysis(cont);

        // (3)Output results
        contHelper.outResult(cont);

        algoDsl.destroy();
```