Probabilistic Load-Flow Using Analysis Using DPL Scripting Language

González-Longatt, F.; Alhejaj, S.; Marano-Marcolini, A.; Rueda, José L.

DOI
10.1007/978-3-319-50532-9_5

Publication date
2018

Document Version
Final published version

Published in
Advanced Smart Grid Functionalities Based on PowerFactory

Citation (APA)
González-Longatt, F., Alhejaj, S., Marano-Marcolini, A., & Rueda, J. L. (2018). Probabilistic Load-Flow Using Analysis Using DPL Scripting Language. In F. Gonzalez-Longatt, & J. Torres (Eds.), *Advanced Smart Grid Functionalities Based on PowerFactory: Green Energy and Technology* (pp. 93-124). (Green Energy and Technology; No. 1). Springer. https://doi.org/10.1007/978-3-319-50532-9_5

Chapter 5
Probabilistic Load-Flow Using Analysis Using DPL Scripting Language

Francisco Gonzalez-Longatt, S. Alhejaj, A. Marano-Marcolini and José Luis Rueda Torres

Abstract Load-flow analysis is an effective tool that is commonly used to capture the power system operational performance and its state at a certain point in time. Power grid operators use load-flow extensively on a daily basis to plan for day-ahead and dispatch scheduling among many other purposes. Also, it used to plan any grid expansion, alter or modernization. However, due to the deterministic nature and its applicability for only one set of operational data at a certain period, deterministic load-flow reduces the chances for predicting the uncertainty in power system. Researchers usually create a data model using probabilistic analyses techniques to produce a stochastic model that mimics the realistic system data. Combining this model with Monte Carlo methodology leads to form a probabilistic load-flow tool that is more powerful and potent to carry on many uncertainty tasks and other aspects of power system assessment. This chapter presents the

Electronic supplementary material
The online version of this chapter (https://doi.org/10.1007/978-3-319-50532-9_5) contains supplementary material, which is available to authorized users.

The original version of this chapter was revised: ESM files have been included. The erratum to this chapter is available at https://doi.org/10.1007/978-3-319-50532-9_15

F. Gonzalez-Longatt (✉) · S. Alhejaj
School of Electronic, Electrical and Systems Engineering, Loughborough University, Loughborough LE11 3TU, UK
e-mail: fglongatt@fglongatt.org

S. Alhejaj
e-mail: s.m.alhejaj@lboro.ac.uk

A. Marano-Marcolini
Department of Electrical Engineering, Universidad de Sevilla, Camino de los Descubrimientos s/n, Seville, Spain
e-mail: alejandromm@us.es

J. L. Rueda Torres
Department of Electrical Sustainable Energy, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
e-mail: J.L.RuedaTorres@tudelft.nl

F. Gonzalez-Longatt and J. L. Rueda Torres (eds.), *Advanced Smart Grid Functionalities Based on PowerFactory*, Green Energy and Technology,
https://doi.org/10.1007/978-3-319-50532-9_5

DIgSILENT PowerFactory script language (DPL) implementation of a DPL script to *perform probabilistic power* flow (PLF) using Monte Carlo simulations (MCS) to consider the variability of the stochastic variables in the power system during the assessment of the steady-state performance. The developed PLF script takes input data from an external Microsoft Excel file, and then, the DPL can carry on a probabilistic load-flow and export the results using a Microsoft Excel file. The suitability of the implemented DPL is illustrated using the classical IEEE 14 buses.

Keywords DPL script · Simulation · Stochastic model · Probabilistic load-flow Distribution function · Distribution density function · Monte Carlo

5.1 Introduction

The modern power systems are characterized by the introduction of more and more uncontrollable types of generation resources such as renewable generation (wind power, solar power, etc.); it increases the uncertainties which lead the more complex operation and control of the power system.

The increased level of uncertainties in the generation and demand side of the power system requires more efficient tools to be able to capture the variability in the system performance. A deterministic approach such as load-flow analysis is limited to a snapshot of steady-state power system operational data. Therefore, the *deterministic load-flow* (DLF) has an intrinsic limitation because it cannot have represented in a proper way the randomness in the power system.

Usually, a probabilistic approach is used when the deterministic approach is not able to capture the intrinsic variability. The *probabilistic approach* is one of the main three ways to capture and represent the intrinsic uncertainties in the power system beside *fuzzy arithmetic technique* and *interval mathematics*. The advantage and disadvantage of each of these approaches are discussed in [1].

The probabilistic approach of load-flow can be grouped into three approaches: Monte Carlo Simulation (MCS), analytical methods and approximation approaches. Monte Carlo simulation methodology is used widely and almost in all fields of engineering and science to simulate a repetitive process with different input data to produce different output results. In this chapter, a *DIgSILENT PowerFactory script language* (DPL) implementation of a DPL script to *perform probabilistic power* flow (PLF) using MCS is presented. This DPL is designed to consider the variability of the stochastic variables in the power system during the assessment of the steady-state performance. The PLF implemented in this chapter offers several advantages: (i) allow importing and exporting using structured database in the form of Microsoft Excel file (.xlsx), (ii) internal matrixes (*IntMatrix*) are used for communication between subscripts; it offers a fast communication and modular programming, allowing the re-use of the code into other applications.

The chapter starts with an introduction to probabilistic load-flow and its fundamentals, and then, an explanation about the Monte Carlo simulation and the main features of the DIgSILENT Simulation Language are presented. Then, the DPL implementation of the PLF is, and an illustrative example is presented.

5.2 Probabilistic Load-Flow: Fundamentals

The analysis of the steady-state conditions of a power system is one of the most commonly used tools in planning and operation of power systems. The classical load-flow is used to define the steady-state power balance in a power system: the total generation should be equal to the total power demand plus the power losses. It represents the so-called DLF; this tool is used to analyse and assess the planning and to operate the power system on a daily routine.

The DLF uses the known values of electrical power generation and power load demand of a selected network configuration to calculate the system states and power flows [2]. The formulation of the load-flow problem assumes that the data provided is absolutely precise and provides results totally compatible with the given data apart from round-off errors.

The integration of renewable generation units creates several planning and operation challenges. From the load-flow point of view, the random fluctuating character of wind speed causes the wind power plant (WPP) power production to be neither continuous nor slightly controllable.

The DPF analysis has the limitation of ignoring the grid uncertainties: outages, network changes, load variation. As a consequence, the deterministic approach is not sufficient for the analysis of modern power systems, the integration of renewable generation and other sources of randomness. The results of using DPF to attempt the calculation in power system considering uncertainties may lead to very different, even contradictory, or erroneous results, creating massive economic and technical consequences.

As expressed before, in the deterministic load-flow, the output of the model is fully determined by the parameter values and the initial conditions. An alternative approach is the use of stochastic models to represent the load-flow conditions considering uncertainties; this approach considers the inherent randomness; as a consequence, the same set of parameter values and initial conditions will lead to an ensemble of different outputs.

The typical *probabilistic load-flow* (PLF) analysis considers the power generation and grid configurations both to be discrete random variables, while load demand is a continuous random variable [3, 4]. Borkowska and Allan [5] proposed the stochastic load-flow (SLF) in 1974; it provides a full reflection of the influences of many factors' random variations in the power system. The PLF based on those methods directly treats the uncertainty of electric load, generation (especially wind power) and grid parameters. The term SLF is an alternatively used term for PLF and is generally favourable for system operational study that deals with short-term uncertainties [6].

The PLF methods can be included in three sets: (i) *analytical approach,* (ii) *numerical approach* and (iii) *approximate methods*.

Apart from the above grouping, *hybrid methods* uniting more than one of the above methods have attracted additional interest as they can overcome some of the limitations of the individual constituting methods.

The analytical approach analyses a system and its inputs using mathematical expressions, i.e. using convolution techniques, with *probabilistic density functions* (PDF) of stochastic variables of power inputs so that PDFs of stochastic variables of system states and line flows can be obtained. Details of this approach can be found on [5–8]. Analytical methods include techniques such as *convolution method*, *cumulant method* (CM).

The numerical approach and sampling methods include techniques such as MCS, Latin hypercube sampling, uniform design sampling.

5.3 Monte Carlo Simulation Applied to the Load-Flow Problem

In general, the DLF problem consists in finding the zero of a set of nonlinear equations starting defining the power system power balance from an adequate initial guess.

The most general form of the load-flow equations is a set of *differential-algebraic equations* (DAE) in steady state [9]. Then, the formulation of the power flow equations is reduced to the algebraic:

$$\begin{aligned} &\mathbf{g}(\mathbf{x}) = \mathbf{0} \\ &\left[\mathbf{g}^{\mathbf{P}}(\mathbf{x}) \quad \mathbf{g}^{\mathbf{Q}}(\mathbf{x})\right]^{\mathrm{T}} = \mathbf{0} \end{aligned} \tag{5.1}$$

where **g** is the set of algebraic equations that define the power balance at network buses and **x** the state vector. For classical formulation of AC DPF, the *inputs* or known quantities are the injected active powers (P_i) at all busbars (where P and Q or P and V are known) except the slack bus; the injected reactive powers (Q_i) at all load busbars (where P and Q are known); and the voltage magnitude at all generator busbars (where P and V are *outputs* or known).

$$\begin{aligned} P_i &= g_i^P(\delta_1, \delta_2, \ldots, \delta_n, V_1, V_2, \ldots, V_n) \\ Q_i &= g_i^Q(\delta_1, \delta_2, \ldots, \delta_n, V_1, V_2, \ldots, V_n) \end{aligned} \tag{5.2}$$

where $i = 1, 2, \ldots, n$. n represents the number of power buses, nonlinear voltage (V) and phase (δ) relationships. A complete explanation for the classical AC power flow can be found on [10–12].

PLF attempts to obtain PDFs of state vector and line flows of a statistically varying electrical network [13]. P_i and Q_i, are considered by their distributions, usual with binomial repartition with p_i and q_i the probability of up, respectively down state for each unit generation $P_{g,i}$ and the PDF load $P_{L,i}$ is continuous and normal with m (mean) and σ (standard deviation) [14–16]. MCS is a method for iteratively evaluating a deterministic model using sets of random numbers as inputs [17]. Stochastic power flow (SPF) is solved using MSC, which involves repeating

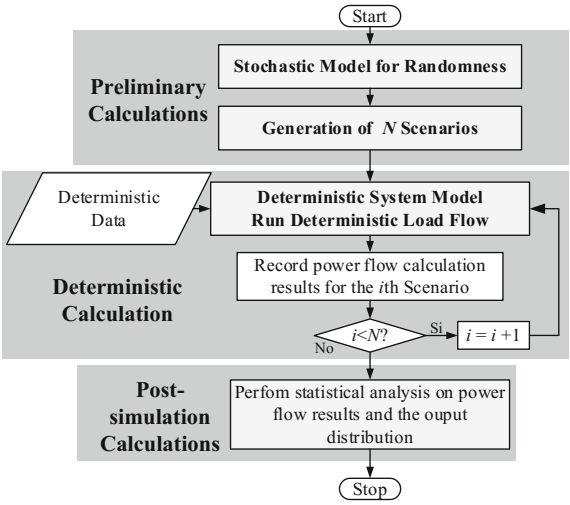


Fig. 5.1 Flowchart of MCS applied to solve PPF

the DLF simulation process using in each simulation a particular set of values of the random variables (loads, conventional and wind generation productions at each node of the considered power system).

MSC is used to solve PLF; it involves repeating the DPF calculation process using in each simulation a particular set of values of the random variables, called *simulation scenario*. Depending upon the number of uncertainties and the ranges specified for them, a MSC could involve a large number of scenarios and recalculations before it is complete. Figure 5.1 shows details of the process of solving Probabilistic power flow (PPF) using MTC.

This chapter adopts a *Monte Carlo* (MC) method for the SPF analysis. This technique is used in [18] to solve the SPF problem including wind farms by repeated simulations. The two main features of MCS are as follows: (a) it provides considerably accurate results, but the computation time is consuming for large systems with several uncertain parameters, (b) it can be easily combined with pre-existent DPF programs to create and easy and fast implementation of SPF. In this paper, the approach selected is an SPF based on MSC.

5.4 DIgSILENT Programming Language (DPL)

Before starting to delve into the program and coding,[1] it is worth explaining how *DIgSILENT Programming Language* (DPL) works in the context of the modern programming world. DPL is a scripting language that enables the users to do many

[1]In order to use DPL, users are required to have some programing experience and writing code in some of the modern used programing languages such as C++, Java and/or Python with some good understanding of the main concepts of object-oriented programing (OOP).

automated tasks available in DIgSILENT PowerFactory [19]. Therefore, the users will have full control over the power grid parameters, power system calculations and pre/post processing the information coming from power system studies. Additionally, the DPL is built with the object-orientated concept in mind so that everything that the user can deal with can be considered as a set of classes that have some parameters and functionalities too. The main and principal class is the DPL command class[2] [1].

The probabilistic power flow implemented in this chapter is implemented using the previously mentioned MSC approach. The DPL implementation assumes the simulation scenarios consist of a number of samples generated per each random variable represented in the problem. For simplicity, the implementation assumed the scenarios are provided to the DPL using a database with a well-defined data structure, taking the opportunity of the DPL capability of managing Microsoft Office files, the script imports the scenarios from a Microsoft Excel file and then the results are exported using the same type of files.

5.5 DPL Implementation: Probabilistic Load-Flow

The implementation of the PLF using MSC consists of one main script which is able to call four subscripts. DPL main script is named "**ProbabilisticLF**" and contains the main logic behind the probabilistic load-flow; it involves: (i) importing the simulation scenarios to be used in the MCS, (ii) calculating the deterministic load-flow per each scenario and (iii) export the results of the deterministic load-flow.

The input of the DPL implementation consists of a database containing (*Nsamples*) simulation scenarios including the variables related to the system stochasticity. The implementation of this chapter considers the variability on the active (*P*) and reactive power (*Q*) on loads, generators, wind power plants, PV plants and PHEV.

The implementation of the PLF is created using a modular approach where main activities are directed from the main script, and the subroutines perform very specific actions. A matrix approach is used for communication and data interchange between the subroutines and main script. Matrices (*IntMat*) are considered as an external object of the main script (**StochData.xlsx**), but also the subroutines are considered internal objects; this approach allows the main script and subroutines use the same matrices to exchange data and information.

The main DPL, "**ProbabilisticLF**", includes four subscripts (see Fig. 5.2):

– "**Read_MSExcel**" is designed to open the MS Excel file and read the data contained in the file considering the specific data structure (for more details see

[2]This component is well explained and documented in the DPL manual of the DIgSILENT PowerFactory software.

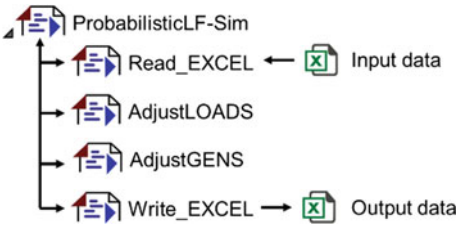


Fig. 5.2 Logical structure of the PobabilisticLF.ComDpl

the file named "**StochData.xlsx**"). When this subscript is executed, twelve matrices (*IntMat*) are loaded with the scenarios data: active power (**DATA_P_ESS, DATA_PG, DATA_PL, DATA_P_PHEV, DATA_P_PV, DATA_P_WPP**) and reactive power (**DATA_Q_ESS, DATA_QG, DATA_QL, DATA_Q_PHEV, DATA_Q_PV, DATA_Q_WPP**).

- "**Read_Matrix**": this subroutine is used to read the input matrix (matrixA) and copies it into the output matrix (matrixB).
- "**Adjust_Loads**": the subroutine is designed to adjust the values of active (*plini*) and reactive power (*qlini*) at the load elements (ElmLod). The script made use of a general selection (Set) to access the specific load where the variability is considered.
- "**Adjust_Gens**": this subroutine is designed to adjust the values of the active (*pgini*) and reactive power (*qgini*) of all the power sources and storage devices in the power network. The subroutine uses a general selection where all the power sources and energy storage equipment are stored: synchronous machines, wind turbines, electric vehicles, PV systems and energy storage. The power network uses *ElmSym* to model the classical synchronous generators and *ElmGenstat*, static generator element for modelling the power converter-based technologies.
- "**Write_Excel**": this subroutine is designed to write the data results of the probabilistic load-flow into a Microsoft Excel file with a very specific data structure.
- The next subsection of this chapter is dedicated to explaining in basic and generic terms the programming aspects of each of the scripts above.

5.6 Main Script: ProbablisticLF

The script named "**ProbabilisticLF**" represents the main program of the PLF implementation, and it is programmed in a very modular way where the subscripts can be systematically called and using matrix (*IntMatrix*) to interchange data and allow the flow of information inside the program.

For simplicity, the main variables involved in the probabilistic load-flow are defined as input parameters of the "**ProbablisticLF**" DPL command. Figure 5.2

shows the main input parameters of the probabilistic simulation including the filename and path of the input and output data, a number of elements to be considered as statistically defined (elements where the active and reactive power will change during the Monte Carlo simulation). Also, the authors have included few configurations related to the load-flow analysis function (*ComLdf*). A flag (*iopt_net*) is used to allow the probabilistic load-flow analysis considered the power flow: (0) balanced, (1) unbalanced or (2) using the DC load-flow method.

The main script **ProbabiliticLF.ComDPL** uses external objects as a way to create an artificial communication between the main script and subscripts (see Fig. 5.3). Figure 5.4 shows the name, object and description of the twelve matrix objects (*IntMat*) used by the probabilistic load-flow implementation main script and its subscripts.

A key aspect of the modular programming approach used in this implementation is the fact that all the objects are inside the main script: (i) subscripts in the form of DPL commands and (ii) matrices (*IntMat*). Because all the objects and subscripts are located inside, the same folder allows a horizontal communication with the matrixes allowing a simple data interchange. All the objects inside the main command DPL, "**ProbabilisticLF**", can be observed suing the button *Contents* of the main command DPL dialogue box. Figure 5.5 shows the contents of the **ProbabilisticLF** command DPL, where the subscripts and matrix objects are shown.

Name: ProbablisitcLF-Sim

General Selection: Study Cases\Base Case\DPL Commands Set

Input parameters:

	Type	Name	Value	Unit	Description
▶ 1	string	OUTPUT_FILENAM	E:\MYPC\My Desktop\Sto		Output Data Filename MS Excel
2	string	INPUT_FILENAME_	E:\MYPC\My Desktop\Sto		Input Data Filename MS Excel
3	int	Nsample	10000		Number of Samples
4	int	PF_Mode	0		Operation mode: 0 Balanced, 1: Unbalanced
5	int	NWPP	1		Number of Wind Power Plants
6	int	NESS	1		Number of Energy Storage Systems
7	int	NPV	2		Number of PV Systems
8	int	NPHEV	1		Number of PHEV
9	int	NGEN	5		Number of Synch Generators
10	int	NLOAD	11		Number of Loads

Fig. 5.3 ProbabilisticLF.ComDpl: lists of the input parameters

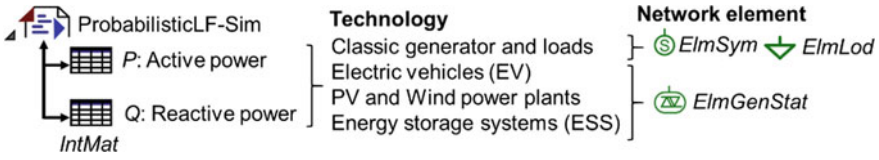


Fig. 5.4 Interaction between ProbabilisticLF.ComDpl and the external object and its interaction with network objects

External Objects:

	Name	object	Description
▶ 1	PESS	PESS	Active power of the Energy Storage System
2	QPV	QPV	Reactive power of the PV systems
3	PPV	PPV	Active power of the PV systems
4	QWPP	QWPP	Reactive power of the Wind Power Plants
5	QPHEV	QPHEV	Reactive power of the PHEV
6	QESS	QESS	Rective power of the Energy Storage Systems
7	PWPP	PWPP	Active power of Wind Power Plants
8	PG	PG	Active power of Synchronous Machines
9	PL	PL	Active power of loads
10	PPHEV	PPHEV	Active power of the PHEV

Fig. 5.5 ProabilisticLF.ComDpl: list of the external objects

In the sequel to this section, a discussion of all tasks is carried through this main script that is illustrated with the relative segment of code.

The DPL script language uses a syntax quite similar to the C++ programming language. This type of language is intuitive, easy to read and easy to learn. The starting point of the main script is the variable definition section; it allows to declare the local variables and initialize them (if needed); see Fig. 5.6.

The next segment of code is used to quantify the type of network elements and the number of them in the active project (see the snippet shown in Fig. 5.7). The script capture for a particular network element, all the object and store them in memory. For example, *AllRelevent()* method is called to get all the load elements *'*. ElmLod'* in the grid and save them in a set of a variable called *'S_load'*. The next programming line, "*n_load = S_load.Count();*" determine the number of objects in the set. Similarly, similar lines of code are written to get the set of all instants of all other network elements and their number count in the active network.

The system information is shown on the output window using the snippet shown in Fig. 5.8.

The main script starts calling the first subscript *READ_EXCEL* in order to import the data from the Microsoft Excel file and save them to into the RAM memory using matrixes. DPL is an object-oriented program, and it requires a precise definition of the variables. The imported data is stored inside ten matrices; five are used for active power and five for reactive power; the size of each matrix is internally initialized using the object method *Init* (see Fig. 5.9); it initializes the matrix with given size and values, regardless of the previous size and data.

The database is imported from the Microsoft Excel file; the file name and path are store in the string named *INPUT_FILENAME_EXCEL,* and it is declared as input parameter of the main script. Then, the filename of the input date is assigned to the *FILENAME* parameter and transferred to the *READ_EXCEL* subscript. Also, the number of samples *Nsample* is also passed to that subscript (see Fig. 5.10). At that point, the subscript *READ_EXCEL* is executed to import all the data records in the excel worksheets to the internal matrices: **DATA_P_ESS, DATA_PG, DATA_PL, DATA_P_PHEV, DATA_P_PV, DATA_P_WPP**) and reactive

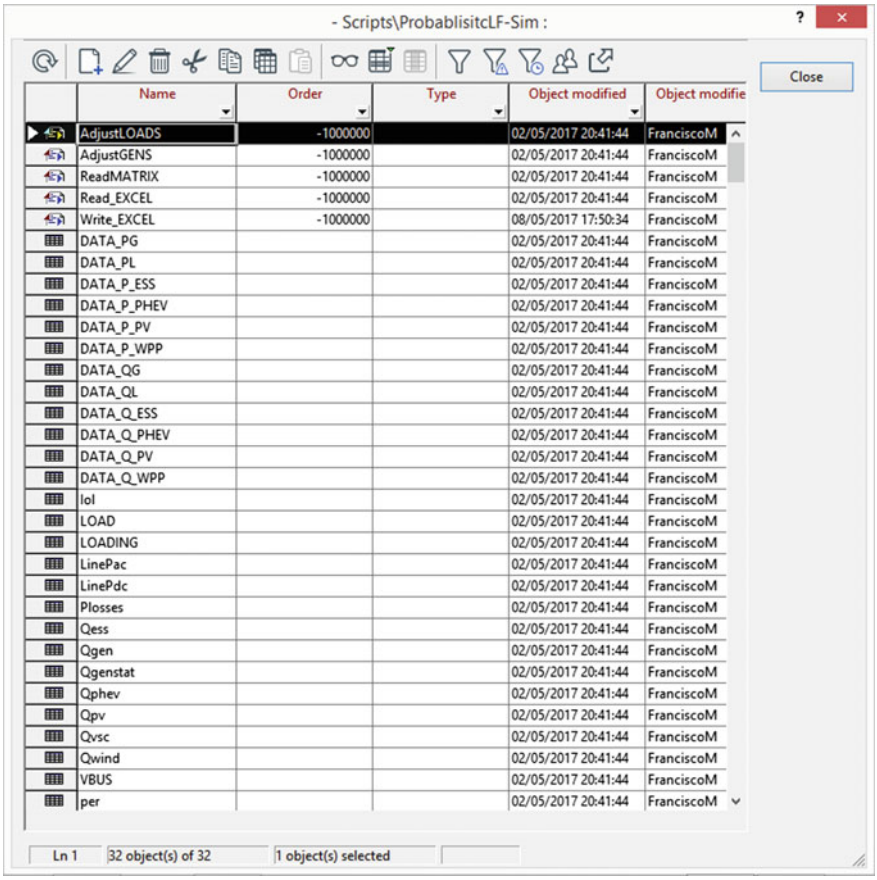


Fig. 5.6 List the objects contained in the main command DPL: ProbabilisticLF.ComDpl

```
! Probabilistic Load Flow (PLF) using Monte Carlo Simulations
! Created by: Dr Francisco Gonzalez-Longatt, April 2015
! Variable Definitions
! Variable Definitions
int       error, iter, ii, n_bus, n_line, n_load;
int       n_sym, n_genstat, n_vscmono, n_wind, n_PV, n_ESS, n_PHEV;
double    M, LO, u1;
double    Time_o, Time_f, Dpl_time;
object    PF, O1, O2, O3, O4, O5, O6, O7, O8, O9, O10;
set       Bus, SB, S_bus, S_line, S_load, S_Sym, S_Genstat, S_VSCmono;
set       sWind, sPV, sESS, sPHEV;
```

Fig. 5.7 Snippet ProbabilisticLF.ComDpl: variable definitions

```
! --------- Get Network Elements--------
! Loads:  ElmLod
S_load = AllRelevant('*.ElmLod');!Returns a set of all available loads in the grid
O1 = S_load.First();              ! Returns the first object in the set
n_load =  S_load.Count();         ! Returns the number of loads in the grid
! Buses: ElmTerm
S_bus = AllRelevant('*.ElmTerm');! Returns a set of (ElmTerm) objects (buses)
O2 = S_bus.First();               ! Returns the first object of set.
n_bus = S_bus.Count();            ! Returns the number of buses
! Lines: ElmLne
S_line = AllRelevant('*.ElmLne');! Returns a set of all (ElmLne) objects (lines)
O3 = S_line.Firstmatch('ElmLne');! Returns the first object in the set
n_line = S_line.Count();          ! Returns the number of lines
! Synchronous Generators: ElmSym
S_Sym = AllRelevant('*.ElmSym'); ! Returns a set of (ElmSym) objects (Synch Mach.)
O4 = S_Sym.Firstmatch('ElmSym'); ! Returns the first objectin the set
n_sym = S_Sym.Count();           ! Returns the number of Synch. Mach. objects
! Wind Power plants: ElemGenstat
sWind = AllRelevant('WF*.ElmGenstat');! Returns a set of all (ElmGenstat) objects
O7=sWind.Firstmatch('WF*.ElmGenstat');! Returns the first object in the set
n_wind =  sWind.Count();          ! Returns the number of objects (wind farms)
! PV Power plants: : ElemGenstat
sPV = AllRelevant('*.ElmPvsys'); ! Returns a set of all (ElmPvsys) objects (PV)
O8 = sPV.Firstmatch('ElmPvsys'); ! Returns the first objects of ElmPvsys.
n_PV =  sPV.Count();              ! Returns the number of PV objects
! ESS –Energy Storage Systems: ElemGenstat
sESS = AllRelevant('Storage.ElmGenstat');! Returns a set of (ElmGenstat) objects
O9 = sESS.First();                ! Returns the first objects Storage.ElmGenstat.
O9.ShowFullName();
n_ESS =  sESS.Count();            ! Returns the first objects Storage.ElmGenstat.
! PHEV –Electric Vehicle: : ElemGenstat
sPHEV = AllRelevant('PHEV.ElmGenstat');! Returns a set (PHEV.ElmGenstat) objects
O10=sPHEV.Firstmatch('PHEV.ElmGenstat');! Returns the first object in this set
n_PHEV =  sPHEV.Count();
```

Fig. 5.8 Snippet ProbabilisticLF.ComDpl: get the network elements

```
printf(' PROBABILISTIC LOAD FLOW \n');
printf(' RUNNING MONTE-CARLO SIMULATIONS ');
printf(' created by Prof. F Gonzalez-Longatt and Samir Alhejaj, Sep 2016');
printf(' ------------------------------------------------ ');
printf(' Number of Buses : %2.2f',n_bus);
printf(' Number of Loads : %2.2f',n_load);
printf(' Number of Lines : %2.2f',n_line);
printf(' Number of Synchronous Generators : %2.2f',n_sym);
printf(' Number of Wind Turbines : %2.2f',n_wind);
printf(' Number of PV Plant : %2.2f',n_PV);
printf(' Number of ESS : %2.2f',n_ESS);
printf(' Number of PHEV : %2.2f',n_PHEV);
```

Fig. 5.9 Snippet ProbabilisticLF.ComDpl: show the power system and network elements information

power (**DATA_Q_ESS, DATA_QG, DATA_QL, DATA_Q_PHEV, DATA_Q_PV, DATA_Q_WPP**). The internal structure and operation of the subscript READ_EXCEL are explained in next subsections.

The main script communicates between subscripts using matrices; as a consequence, the reader must recognize that the imported data from Microsoft Excel is

```
DATA_PL.Init(Nsample,NLOAD);     ! Initialize an internal matrix for Load
DATA_QL.Init(Nsample,NLOAD);
DATA_PG.Init(Nsample,NGEN);      ! Initialize an internal matrix for Syn Gen
DATA_QG.Init(Nsample,NGEN);
DATA_P_WPP.Init(Nsample,NWPP);   ! Initialize an internal matrix for WPP
DATA_Q_WPP.Init(Nsample,NWPP);
DATA_P_PV.Init(Nsample,NPV);     ! Initialize an internal matrix for PVP
DATA_Q_PV.Init(Nsample,NPV);
DATA_P_ESS.Init(Nsample,NESS);   ! Initialize an internal matrix for ESS
DATA_Q_ESS.Init(Nsample,NESS);
DATA_P_PHEV.Init(Nsample,NPHEV); ! Initialize an internal matrix for PHEV
DATA_Q_PHEV.Init(Nsample,NPHEV);
```

Fig. 5.10 Snippet ProbabilisticLF.ComDpl: initialize the values of the internal matrixes

transferred into specific matrices for the internal use inside the main script. Figure 5.11 shows the command lines to copy the data in the "**PG**" matrix into the "**matrixA**" matrix of "**ReadMATRIX**" subscript. By calling "**ReadMATRIX**" subscript and execute it, the data are returned and saved into "**DATA_PG**" matrix. Similarly, the same is executed for all other matrices. The internal structure and operation of the subscript "**ReadMATRIX**" are explained in next subsections.

Now, the main script initiates the Monte Carlo simulation process. Initially, set the start time of the simulation processing time (see Fig. 5.12). It also initializes the "PF" object for load-flow class "*ComLdf*" by calling "*GetCaseObject()*". This object contents all the details that are necessary for running the load-flow analysis.

The Monte Carlo simulation process is a systematic process where the deterministic load-flow is evaluated by each of the scenarios defined in the *Nsamples*. As a consequence, a loop is used to counter "*iter*" the number of the scenarios; the loop "*for*" will complete the calculation of "*Nsample*" defined in the input parameter of the "**ProbablisitcLF-Sim**" DPL command. The simulation process is simple; inside the loop, the value of the active and reactive power of the elements considered as stochastically described is updated, and then, a deterministic load-flow is calculated (see Fig. 5.13).

The subscripts "**AdjustLOADS**" and "**AdjustGENS**" are used to set the active and reactive power of the passive and active network elements. The internal structure and operation of the subscript "**AdjustLOADS**" and "**AdjustGENS**" are explained in next subsections.

A health check is included in the Monte Carlo simulation process. The script executes the load-flow analysis using the following sentence "*error = PF.Execute ()*"; if the load-flow is successful and there is coverage, the execution function will return 0 and if not will return 1. The user can add error handling routine to display an error message in case there is no convergence and load-flow does not complete.

```
! Importing Data From Microsft Excel File (INPUT_FILENAME_EXCEL.xls)
READ_EXCEL:FILENAME = INPUT_FILENAME_EXCEL;
READ_EXCEL:Nsamples = Itermax;
READ_EXCEL.Execute();
```

Fig. 5.11 Snippet ProbabilisticLF.ComDpl: importing data from Microsoft Excel

```
!   ---------- Populate the Inernal Matrices ------------
ReadMATRIX:matrixA = PG;        ! Copy Synch. Gen. active power data into matrixA
ReadMATRIX:matrixB = DATA_PG;   ! Assign matrix DATA_PG to matrixB
ReadMATRIX.Execute();           ! Swap data from PG into DATA_PG
ReadMATRIX:matrixA = QG;        ! Copy Synch. Gen. reactive power data to matrixA
ReadMATRIX:matrixB = DATA_QG;   ! Assign matrix DATA_QG to matrixB
ReadMATRIX.Execute();           ! Swap data from QG into DATA_QG
ReadMATRIX:matrixA = PL;        ! Copy Load active power data to matrixA
ReadMATRIX:matrixB = DATA_PL;   ! Assign matrix DATA_PL to matrixB
ReadMATRIX.Execute();           ! Swap data from PL into DATA_PL
ReadMATRIX:matrixA = QL;        ! Copy Load reactive power data to matrixA
ReadMATRIX:matrixB = DATA_QL;   ! Assign matrix DATA_QL to matrixB
ReadMATRIX.Execute();           ! Swap data from QL into DATA_QL
ReadMATRIX:matrixA = PWPP;      ! Copy WPP active power data to matrixA
ReadMATRIX:matrixB = DATA_P_WPP;! Assign matrix DATA_P_WPP to matrixB
ReadMATRIX.Execute();           ! Swap data from PWPP into DATA_P_WPP
ReadMATRIX:matrixA = QWPP;      ! Copy QWPP reactive power data to matrixA
ReadMATRIX:matrixB = DATA_Q_WPP;! Assign matrix DATA_Q_WPP to matrixB
ReadMATRIX.Execute();           ! Swap data from QWPP into DATA_Q_WPP
ReadMATRIX:matrixA = PPV;       ! Copy PPV active power data to matrixA
ReadMATRIX:matrixB = DATA_P_PV; ! Assign matrix DATA_P_PV to matrixB
ReadMATRIX.Execute();           ! Swap data from PPV into DATA_P_PV
ReadMATRIX:matrixA = QPV;       ! Copy QPV reactive power data to matrixA
ReadMATRIX:matrixB = DATA_Q_PV; ! Assign matrix DATA_Q_PV to matrixB
ReadMATRIX.Execute();           ! Swap data from PG into DATA_Q_PV
ReadMATRIX:matrixA = PESS;      ! Copy PESS active power data to matrixA
ReadMATRIX:matrixB = DATA_P_ESS;! Assign matrix DATA_P_ESS to matrixB
ReadMATRIX.Execute();           ! Swap data from PESS into DATA_P_ESS
ReadMATRIX:matrixA = QESS;      ! Copy QESS reactive power data to matrixA
ReadMATRIX:matrixB = DATA_Q_ESS;! Assign matrix DATA_Q_ESS to matrixB
ReadMATRIX.Execute();           ! Swap data from QESS into DATA_Q_ESS
ReadMATRIX:matrixA = PPHEV;     ! Copy PHEV active power data to matrixA
ReadMATRIX:matrixB = DATA_P_PHEV;! Assign matrix DATA_P_PHEV to matrixB
ReadMATRIX.Execute();           ! Swap data from PG into DATA_PG
ReadMATRIX:matrixA = QPHEV;     ! Copy QPHEV reactive power data to matrixA
ReadMATRIX:matrixB = DATA_Q_PHEV;! Assign matrix DATA_Q_PHEV to matrixB
ReadMATRIX.Execute();
```

Fig. 5.12 Snippet ProbabilisticLF.ComDpl: populate internal matrices for the Monte Carlo simulation process

```
!    MONTE CARLO SIMULATION PROCESS
!    Initialiting variables for Monte-Carlo simulations
Time_o = GetTime(4);           ! Get System Time
PF = GetCaseObject('*.ComLdf');! Returns first found object of '*.ComLdf' class
                               ! from the currently active study case
PF:iopt_net = PF_Mode;         ! PF_Mode = 1 force unbalanced
                               !  = 0 force balanced
```

Fig. 5.13 Snippet ProbabilisticLF.ComDpl: adjustment and setting of the deterministic load-flow inside the Monte Carlo simulation process

In the end of the load-flow analysis, which will take few seconds, new results values will be saved to each element result object by PowerFactory (see Fig. 5.14).

The deterministic load-flow results are obtained per each simulation scenario; internal matrixes are used to collect the relevant results: bus voltages, lines loading, current in each transmission line and power losses of the transmission lines (internal matrices: *VBUS, LOADING, IOl* and *Plosses*; see Fig. 5.14).

```
! ------------Start Monte Carlo Simulation -------------
for (iter = 1; iter <= Itermax; iter+= 1) {
! BEGIN iter
printf(' ----------------------------------------------');
printf(' Scenario #%d',iter);
! 2.a. ADJUSTING THE STOCHASTIC ELEMENTS
! Adjust Loads ElmLod
AdjustLOADS: iID = iter;
AdjustLOADS: oMatrixP = DATA_PL;
AdjustLOADS: oMatrixQ = DATA_QL;
AdjustLOADS.Execute();
! Adjust Synch Generators ElmSym
AdjustGENS: iID = iter;
AdjustGENS: sGenType= S_Sym;
AdjustGENS: oMatrixP = DATA_PG;
AdjustGENS: oMatrixQ = DATA_QG;
AdjustGENS.Execute();
! Adjust Wind Power plants ElmGenstat
AdjustGENS: iID = iter;
AdjustGENS: sGenType= sWind;
AdjustGENS: oMatrixP = DATA_P_WPP;
AdjustGENS: oMatrixQ = DATA_Q_WPP;
AdjustGENS.Execute();
! Adjust PV power plants ElmGenstat
AdjustGENS: iID = iter;
AdjustGENS: sGenType= sPV;
AdjustGENS: oMatrixP = DATA_P_PV;
AdjustGENS: oMatrixQ = DATA_Q_PV;
AdjustGENS.Execute();
! Adjust ESS ElmGenstat
AdjustGENS: iID = iter;
AdjustGENS: sGenType= sESS;
AdjustGENS: oMatrixP = DATA_P_ESS;
AdjustGENS: oMatrixQ = DATA_Q_ESS;
AdjustGENS.Execute();
! Adjust PHEV ElmGenstat
AdjustGENS: iID = iter;
AdjustGENS: sGenType= sPHEV;
AdjustGENS: oMatrixP = DATA_P_PHEV;
AdjustGENS: oMatrixQ = DATA_Q_PHEV;
AdjustGENS.Execute();
```

Fig. 5.14 Snippet ProbabilisticLF.ComDpl: adjustment in the active and reactive power of the active and passive network elements

The numerical results of the load-flow calculations are exported into Microsoft Excel file; the output data is transferred by using the "**Write_EXCEL**" subscript. The results exporting process start by saving the "Loading" percentage of the transmission lines data to the first worksheet and giving it a name "Loading". Similarly, the buses voltages "Voltages", lines currents "Currents", lines active power "Pij" and power losses "Plosses" are saved (see Fig. 5.15). After the systematic calling of the "**Write_EXCEL**" subscript, the whole set of data is exported into a single Microsoft Excel file and the Monte Carlo simulation process ends.

```
! EXECUTE DETERMINIST LOAD FLOW (ComLdf)
  error = PF.Execute();     ! Executes the command Load Flow
  !  COLLECT ALL RELEVANT DATA
  ! I. Bus Voltages (per unit)  -ElmTerm
  O2 = S_bus.First();       ! Returns the first matching object ElmTerm
  for(ii = 1; ii <= n_bus; ii+= 1) {
  M = O2:m:u1;              ! u1: Magnitude of Terminal Voltages
  VBUS.Set(iter,ii,M);      ! Set the value at position (i,ii) in the matrix
  O2 = S_bus.Next();        ! Returns the next matching object Elmterm
  }
  ! II. Line Loading Conditions (%) _ElmTerm
  O3 = S_line.First();      ! Returns the first matching object ElmTerm
  for(ii=1; ii <= n_line; ii+=1) {
    LO = O3:c:loading;      ! loading: %
    LOADING.Set(iter,ii,LO);
    O3 = S_line.Next(); }
  ! III. Synchronous Generator -ElmSym
  O4 = S_Sym.First();
  for(ii=1;ii<=n_sym;ii+=1) {
    LO = O4:m:Q:bus1;     ¡ Q: reactive power generation
    Qgen.Set(iter,ii,LO);
    O4 = S_Sym.Next(); }
  !  V. Wind Power Plant -ElmGenstat
  O7 = sWind.First();       ! Returns the first objects ElmGenstat
  for(ii=1;ii<=n_wind;ii+=1) {
    LO = O7:m:Q:bus1;       ¡ Q: reactive power generation
    Qwind.Set(iter,ii,LO);
    O7 = sWind.Next(); }
  !  VI. PV Power Plant - ElmGenstat
  O8 = sPV.First();         ! Returns the first objects ElmGenstat
  for(ii=1;ii<=n_PV;ii+=1) {
  LO = O8:m:Q:bus1;         ¡ Q: reactive power generation
    Qwind.Set(iter,ii,LO);
    O8 = sPV.Next();   }
  !  VII. ESS -ElmGenstat
  O9 = sESS.First();        ! Returns the first objects ElmGenstat
  O9.ShowFullName();
  for(ii=1;ii<=n_ESS;ii+=1) {
    LO = O9:m:u:bus1;       ¡ Q: reactive power generation
    Qess.Set(iter,ii,LO);
    O9 = sESS.Next();  }
  !  VIII. PHEV -ElmGenstat
  O10 = sPHEV.First();        ! Returns the first objects ElmVSCmono
  O10.ShowFullName();
  for(ii=1;ii<=n_PHEV;ii+=1) {
    LO = O10:m:u:bus1;
    Qess.Set(iter,ii,LO);
  O10 = sESS.Next(); }
  ! IX. Line Current Iij -ElmLne
  O3 = S_line.First();          ! Returns the first matching object ElmTerm
  for(ii=1;ii<=n_line;ii+=1) {
    LO = O3:m:i:bus1;
    Iol.Set(iter,ii,LO);        ! Save the Loading value to LOADING Matrix
  O3 = S_line.Next(); }
  ! XI. Lines AC Power flow Pij -ElmLne
  O3 = S_line.First();          ! Returns the first matching object ElmTerm
  for(ii=1;ii<=n_line;ii+=1) {
    M = O3:m:P:bus1;
    LinePac.Set(iter,ii,M);       ! Save the Loading value to PinBus Matrix
  O3 = S_line.Next(); }
  ! XII. Active power losses -ElmLne
  O3 = S_line.First();          ! Returns the first matching object ElmTerm
  for(ii=1;ii<=n_line;ii+=1) {
    M = O3:m:Ploss:bus1;
    Plosses.Set(iter,ii,M);       ! Save the Loading value to PinBus Matrix
    O3 = S_line.Next(); }
  } ! END Iter
  !
```

Fig. 5.15 Snippet ProbabilisticLF.ComDpl: executing deterministic load-flow and collecting the numerical results

5.7 DPL Subroutines

The main script **ProbabilisticLF.ComDpl** includes four subscripts (see Fig. 5.1):

- "**Read_Excel**" is designed to open the MS Excel file and read the data contained in the file considering the specific data structure (for more details, see the file named "**StochData.xlsx**").
- "**Read_Matrix**": this subroutine is used to read the input matrix (*matrixA*) and copies it into the output matrix (*matrixB*).
- "**Adjust_Loads**": the subroutine is designed to adjust the values of active (*plini*) and reactive power (*qlini*) at the load elements (*ElmLod*). The script made use of a general selection (*Set*) to access the specific load where the variability is considered.
- "**Adjust_Gens**": this subroutine is designed to adjust the values of the active (*pgini*) and reactive power (*qgini*) of all the power sources and storage devices in the power network.
- "**Write_Excel**": this subroutine is designed to write the data results of the probabilistic load flow into a Microsoft Excel file with a very specific data structure.

The next subsection of this chapter is dedicated to explain in basic and generic terms the programming aspects of each of the aforementioned scripts.

5.7.1 Reading Stochastic Data from Excel File: Read_Excel

DIgSILENT PowerFactory allows the communication with Microsoft Office; as a consequence, the basic features of all spreadsheets in Microsoft Excel can be used to read and write data. The subscript named **Read_EXCEL** is used to load the simulation scenarios (*Nsamples*) used in the probabilistic load-flow; the data records of active and reactive powers for active and passive network components are stored in a structure MS Excel file. Then, the subroutine DPL script **Read_EXCEL** reads the data from the file and stores the read data into internal matrices (*IntMatrix*).

The subscript uses as input parameter a string variable (*FILENAME*) receiving the name and location of the MS Excel file that holds the stochastic data, and an integer variable is also declared to be assigned the number of scenarios to be simulated (*Nsamples*)—see Fig. 5.16. Moreover, numbers of global matrices are defined to hold the data. Such variables can be added by clicking on the "Contents" button from the "Basic Options" tab of the DPL command dialogue (Fig. 5.17).

When the subscript is called, it prints the file name "*stochData.xlsx*" and its path. In the case of any problem with the excel software or the excel data file, an error message will be displayed as part of the error handling functionality available in DPL. The number of worksheets available in the excel file is read by "*xlGetWorksheetCount()*" method and assigned to an integer variable "*iCount*".

```
! EXPORTING SIMULATION RESULTS TO MICROSOFT EXCEL SHEETS
! 1. Saving LOADING Results
  Write_EXCEL: oResult = LOADING;
  Write_EXCEL: sResult = 'Loading';
  Write_EXCEL:sheetIndex = 1;
  Write_EXCEL.Execute();
! 2. Saving Bus Voltages Results
  Write_EXCEL: oResult = VBUS ;
  Write_EXCEL: sResult = 'Voltages';
  Write_EXCEL:sheetIndex = 2;
  Write_EXCEL.Execute();
! 3. Saving Line current Results
  Write_EXCEL: oResult = Iol ;
  Write_EXCEL: sResult = 'Currents';
  Write_EXCEL:sheetIndex = 3;
  Write_EXCEL.Execute();
! 4. Saving Lines Active Power (AC) Results
  Write_EXCEL: oResult = LinePac ;
  Write_EXCEL: sResult = 'Pij';
  Write_EXCEL:sheetIndex = 4;
  Write_EXCEL.Execute();
! 5. Saving Power Losses Results
  Write_EXCEL: oResult = Plosses ;
  Write_EXCEL: sResult = 'Power losses';
  Write_EXCEL:sheetIndex = 5;
  Write_EXCEL.Execute();
! END OF EXPORTING SIMULATION RESULTS TO MICROSOFT EXCEL SHEETS
  Time_f=GetTime(4);      ! GetSystemTime();
  Dpl_time = Time_f - Time_o;
  printf('Processor Time: %g sec',Dpl_time);
```

Fig. 5.16 Snippet ProbabilisticLF.ComDpl: executing deterministic load-flow and collecting the numerical results

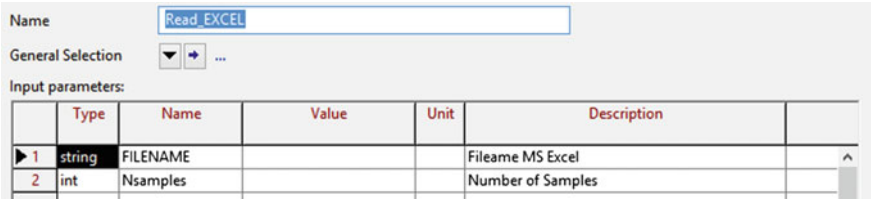


Fig. 5.17 Details of the input parameters of the Read_EXCEL.ComDpl

A loop goes through each worksheet and reads the data from and saves them to the right matrix. For example, the first worksheet data will be read and assigned to "*PG*" matrix by "*Set()*" method and so on for the other matrices. Reading data from excel worksheet is well documented and illustrated by some examples from the knowledge base of the DIgSILENT software website. After the end of importing data process, "*xlTerminate()*" method closes the current instant of excel file (see Figs. 5.18 and 5.19).

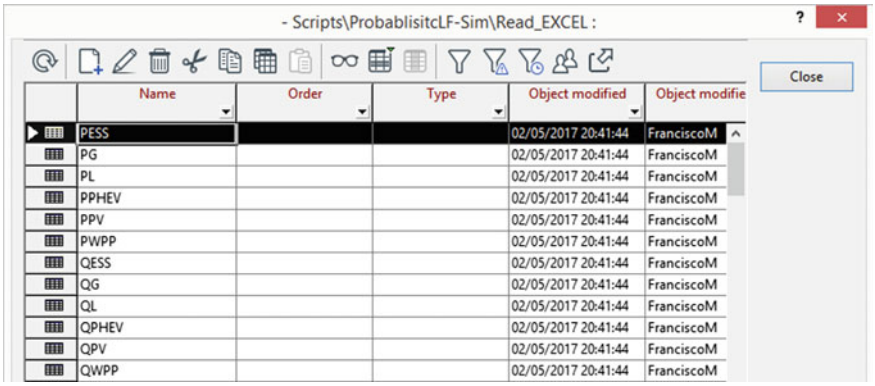


Fig. 5.18 Internal matrices used to store the data read by the subscript Read_EXCEL

5.7.2 *Coping Matrices Content Between Two Subscripts: ReadMATRIX*

The subscript **Read_MATRIX** is designed to copy the data records that are been saved to **READ_EXCEL** subscript matrices into the main program script matrices. The **ReadMATRIX** subscript transforms the local data read from the excel file into a global dataset, and then, it can be used by other subscripts. Even this task seems an additional and not required; the subscript is used here for illustration purposes. Users can still use a different approach or using the passing arguments and getting results through methods that are provided by DPL. The main task here is to swap data from matrix "*matrixA*" to matrix "*matrixB*" and then return the last matrix into the main script (Fig. 5.20).

5.7.3 *Adjusting Loads: Adjust_LOADS*

The subscript named **Adjust_LOADS** is called from the main script in order to set the simulation scenario; it is done by adjusting the active and reactive power in each passive element of the network for the iterations of Monte Carlo simulation process. The network elements are selected by using "*AllRelevant()*" method. A counter "*jj*" is set to the first object "*oObj*" of these elements, and a loop is iterating through each object of this element and assign active power "*plini*" and reactive power "*qlini*" (see Fig. 5.21).

```
! DPL Subscript - Read Data From Excel File
! VARIABLE DEFINITION
int iError, iCount, i, iRow, iCol, iStop, iLstr;
double dValue;
string sString, sString1;
printf('    Loading MS Excel Filename : %s ',FILENAME);
iError = xlStart(); ! Creates a new MS Excel instance
if(iError) {
  Error(' Unable to start MS Excel application '); exit(); }
! Opens an existing workbook = opens xls file
iError = xlOpenWorkbook(FILENAME);
if (iError) {
  Error(' Unable to open Excel file ');
  xlTerminate(); ! Closes currently active MS Excel instance
  exit(); }
iCount = xlGetWorksheetCount();  ! Get number of sheets
printf('    Number of Sheets : %i',iCount);
for(i = 1; i<= iCount; i=i+1) {
  xlActivateWorksheet(i); sString = xlGetWorksheetName(i);
  iRow = 1;   iStop = 0;
  while(iStop = 0) {
    iCol = 1;
    while(1) {
      xlGetValue(iCol, iRow, sString1); xlGetValue(iCol, iRow, dValue);
      iLstr = strlen(sString1); ! Returns the length of a string
      if (iLstr = 0) { ! Stop at empty cell, continue with next row
        if (iCol = 1) {
          iStop = 1;   ! Completely stop if cell in first column is empty
        }
       break;
     }
     if (i=1) {       PG.Set(iRow,iCol,dValue);      }
     if (i=2) {       QG.Set(iRow,iCol,dValue);      }
     if (i=3) {       PL.Set(iRow,iCol,dValue);      }
     if (i=4) {       QL.Set(iRow,iCol,dValue);      }
 ! Adding the power generation sources
     if (i=5) {        PWPP.Set(iRow,iCol,dValue);     }
     if (i=6) {        QWPP.Set(iRow,iCol,dValue);     }
     if (i=7) {        PPV.Set(iRow,iCol,dValue);      }
     if (i=8) {        QPV.Set(iRow,iCol,dValue);      }
     if (i=9) {        PESS.Set(iRow,iCol,dValue);     }
     if (i=10) {       QESS.Set(iRow,iCol,dValue);     }
     if (i=11) {       PPHEV.Set(iRow,iCol,dValue);    }
     if (i=12) {       QPHEV.Set(iRow,iCol,dValue);    }
     iCol = iCol + 1; }
   iRow = iRow + 1; }
}
xlTerminate(); ! Closes currently active MS Excel instance
```

Fig. 5.19 Snippet Read_EXCEL.ComDpl

5.7.4 Adjusting Generators Power: Adjust_GENS

The subscript named **Adjust_GENS** is called from the main script in order to set the simulation scenario; it is done by adjusting the active and reactive power in each active element of the network for the iterations of Monte Carlo simulation process (see Fig. 5.22).

```
! DPL Subscript – Swapping The Contents For Two Matrices
  int   nrowsA, ncolsA,  nrowsB, ncolsB, i1, j1;
  double VALUE;
  printf(' LOADING MATRIXES');
  nrowsA = matrixA.NRow();    ncolsA = matrixA.NCol();
  nrowsB = matrixB.NRow();    ncolsB = matrixB.NCol();
  VALUE = 1;
  for(i1 = 1; i1 <= nrowsA; i1 = i1 + 1) {
    for(j1 = 1; j1 <= ncolsA; j1 = j1 + 1) {
      VALUE = matrixA.Get(i1,j1); matrixB.Set(i1,j1,VALUE);
  }
}
```

Fig. 5.20 Snippet Read_MATRIX.ComDpl

```
  ! DPL Subscript - Adjusting Loads
  int   iN, jj, ii;
  double a, b;
  string sLname;
  object oObj;
  set    sLoads;
  printf(' ADJUSTING LOADS');   printf(' Adjusting Iteration : %i',iID);
  sLoads = AllRelevant('*.ElmLod');
  iN = sLoads.Count();
  printf(' Number of Elements in General Selecion : %i',iN);
  printf(' Full Name of the Objects');
  oObj = sLoads.First();
  jj = 0;
  ! Cycle through the objects in the set and print out the full name
  while(oObj) {
      jj = jj+1;
      a = oMatrixP.Get(iID,jj);
      oObj:plini = a;
      b = oMatrixQ.Get(iID,jj);
      oObj:qlini = b;
      sLname = oObj:loc_name;
      printf(' %i. %s: Pl = %3.3f MW  Ql = %3.3f MVAR ', jj, sLname,a,b);
      oObj = sLoads.Next();
  }
```

Fig. 5.21 Snippet Adjust_LOADS.ComDpl

5.7.5 *Exporting Results Data into Excel File: Write_EXCEL*

The subscript named Write_EXCEL is called from the main script in order to export the numerical results of the Monte Carlo simulation process into a single Microsoft Excel file. This subscript creates a new excel file and gives it uses as a name the input parameter *OUTPUT_FILENAME_EXCEL* entered in the main script. Then, it iterates through each row and column value (using "*mxValue*()" method) in the matrix and writes it (using "*xlSetValue*()" method) to the correspondence cell in the excel worksheet (see details in Fig. 5.23).

```
! DPL Subscript – Adjusting Generation Values
int iN, jj, ii;
double a,b;
string sGname;
object oObj;
printf(' ADJUSTING GENS'); printf(' Row to be adjusted : %i', iID);
iN = sGenType.Count(); printf(' Number of Elements in General Selection : %i',iN);
printf(' Full Name of the Objects');
oObj = sGenType.First();
jj = 0;
! Cycle through the objects in the set and print out the full name
while(oObj) {
              jj = jj+1;
          a = oMatrixP.Get(iID,jj);
          oObj:pgini = a ;
          b = oMatrixQ.Get(iID,jj);
          oObj:qgini = b ;
          sGname = oObj:loc_name;
          printf(' %i. %s: Pg = %3.3f MW  Qg = %3.3f MVAR', jj, sGname, a, b);
          oObj = sGenType.Next();
}
```

Fig. 5.22 Snippet Adjust_GENS.ComDpl

5.8 Simulations and Results

The classical IEEE 14 bus test system is used in this chapter to illustrate the results of the probabilistic load-flow. The IEEE 14 bus test system represents a portion of the American electric power system (in the Midwestern US) as of February, 1962. The test system consists of five synchronous machines; three of which are synchronous compensators used only for reactive power support. There are 11 loads in the system totalling 259 MW and 81.3 Mvar. The data of the IEEE 14 bus test system is publically available at: https://www2.ee.washington.edu/research/pstca/pf14/pg_tca14bus.htm.

The IEEE 14 bus test system is used in this section, and it has been modelled using DIgSILENT PowerFactory. However, the model used here is customized variation of the original system which is created to integrate new technologies where the presence of uncertainties could be modelled using probabilistic data. New generation technologies, energy storage system and electric vehicles are included. Two wind power plants are added: onshore wind farm (WF1) and offshore wind farm (WF2). A multi-terminal HVDC system (three terminals) is used to connect the WF1 and WF2 into the IEEE 14 bus system. Two photovoltaic plants (PV1 and PV2), electric vehicle (PHEV) and battery energy storage system BESS (battery energy storage system) are added to the network. Table 5.1 lists all the generation elements in the grid with their nominal ratings. Figure 5.23 shows the single line diagram of the customized IEEE-14 test model, and Fig. 5.24 shows details of the HVDC transmission network use to integrate the wind farm power plants. The customized IEEE 14 bus test system is a representative network of the future power system considering several new technologies.

```
! Write2ECEL: Subrutine deseigned to write the results of the probabilitic load flow
! into a Microsoft Excel file.
! VARIABLE DEFINITION
int iError, savingErr, noRows, noCols, row, col, noOfXleWSs, xleWSno;
double mxValue;
string sWsName, sNAME;
int results, ii, n_bus, n_line;
object O2, O3;
set S_bus, S_line;
printf('    Data Transfer to Excel File Started ...');
xlAddWorksheet(sResult); ! Create New Worksheet
! ADDING HEADERS
results = strcmp(sResult,'Voltages');
if (results =0)      {
 S_bus = AllRelevant('*.ElmTerm'); ! Returns a set with calculation relevant objects
 O2 = S_bus.First();               ! Returns the first objects ElmTerm.
 n_bus = S_bus.Count();            ! Returns the number of stored ElmTerm.
 for(ii = 1; ii <= n_bus; ii+= 1) {
    sNAME = O2:loc_name;
    O2 = S_bus.Next();        ! Returns the next matching object Elmterm
    xlSetValue(ii,1,sNAME);
  }
}
if (results <>0) {
  S_line = AllRelevant('*.ElmLne'); ! Returns a set with calculation relevant objects
  O3 = S_line.Firstmatch('ElmLne'); ! Returns the first objects ElmLne.
  n_line = S_line.Count();
  for(ii = 1; ii <= n_line; ii+= 1) {
     sNAME = O3:loc_name;
     O3 = S_line.Next();        ! Returns the next matching object Elmterm
     xlSetValue(ii,1,sNAME);
   }
}
printf('    Excel-Worksheet name: %s', sResult );
!  Reading Data from Matrix and write it to Excel
noRows = oResult.NRow(); noCols = oResult.NCol();
mxValue = 0;
for (row = 1 ; row <= noRows ; row =row+1) {
  for (col = 1 ; col <= noCols ; col= col+1) {
    mxValue = oResult.Get(row,col);
    xlSetValue(col,row+1,mxValue); }
}
```

Fig. 5.23 Snippet Write_EXCEL.ComDpl

Seven load elements are connected to the test system, and Table 5.2 shows the active and reactive power demand of the loads (Fig. 5.25).

5.9 Stochastic Data Modelling

The IEEE 14 bus test system has been customized to include new technologies of generation, energy storage and also transportation (electric vehicles). Those new technologies and the loads are used to include uncertainties in the system in order to demonstrate the use of the probabilistic load-flow. The uncertainties are modelled in this paper using probabilistic models described by a probability distribution.

Table 5.1 Details of the installed capacity of the active elements in the test system

Generator	Type	Bus	Installed capacity (MVA)
G1	Synchronous generator	1	304.00
G2	Synchronous generator	2	304.00
G3	Synchronous generator	3	80.00
WF2	Offshore wind farm	4, 5	6.15
CS6	Synchronous capacitor	6	55.00
CS8	Synchronous capacitor	8	55.00
PHEV	Power hybrid electric vehicle	9	1.00
Storage	Energy storage (battery)	10	5.00
PV2	Photovoltaic plant	11	5.00
PV1	Photovoltaic plant	12	2.50
WF1	Onshore wind farm	14	6.52

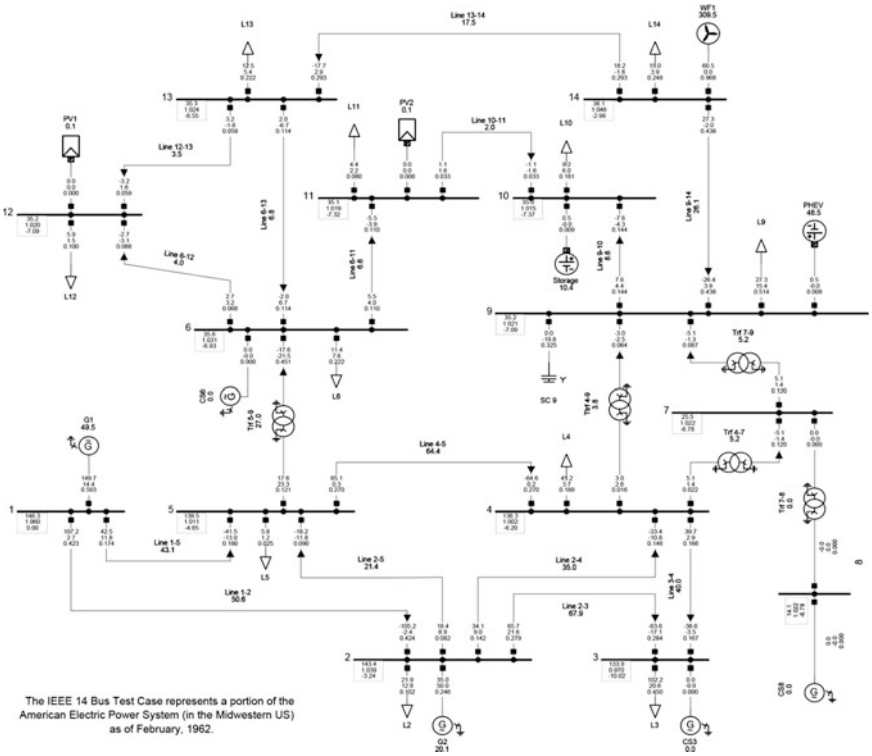


Fig. 5.24 Customized IEEE 14 bus test system including new technologies

Table 5.2 Details of the power consumption on the load connected in the test system

Load	Bus	Active power (MW)	Reactive power (MVar)
L2	2	21.95	12.85
L3	3	102.22	20.62
L4	4	45.14	3.69
L5	5	5.80	1.22
L6	6	11.39	7.63
L9	9	27.34	15.38
L10	11	4.35	2.24
L11	11	4.35	2.24
L12	12	5.88	1.54
L13	13	12.46	5.35
L14	14	14.99	3.86

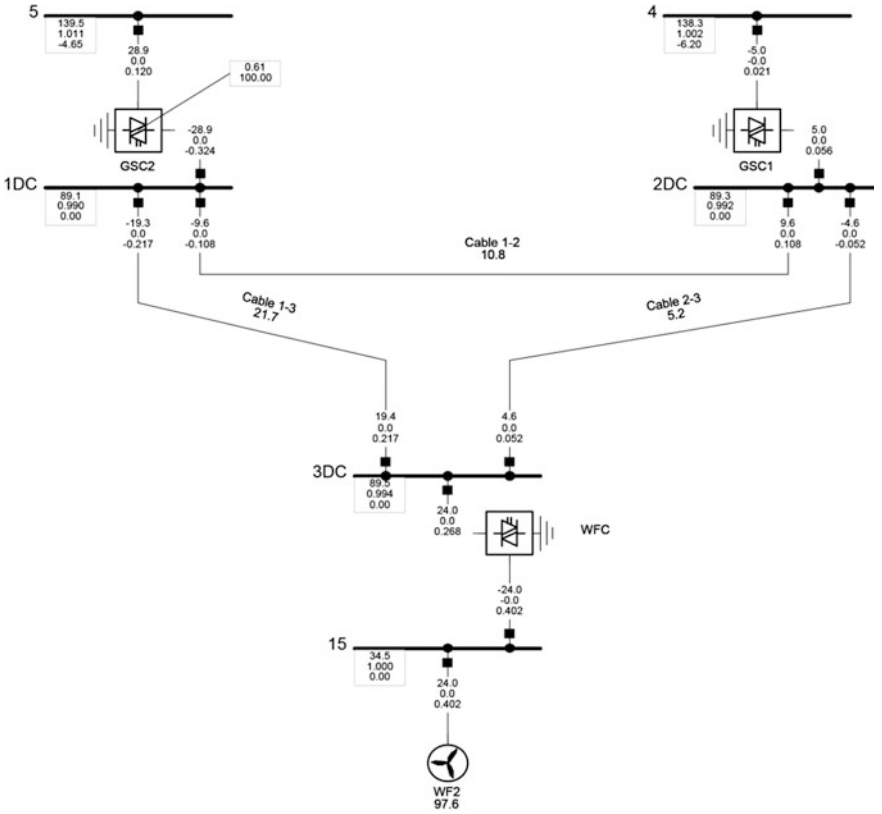


Fig. 5.25 Three-terminal HVDC transmission system used to integrate the wind power plants (WF1 and WF2)

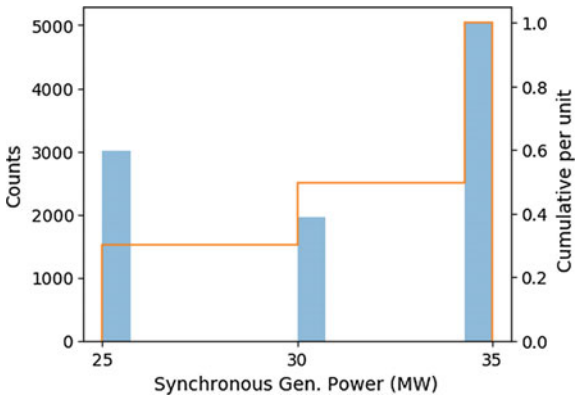


Fig. 5.26 Histogram for synthetic data of G1. Three-state empirical distribution: 25, 30 and 35 MW

A MATLAB program has been used to generate the simulation scenarios considered in this chapter; the program used the well-known inverse transform; it produces samples of a desired probability distribution.

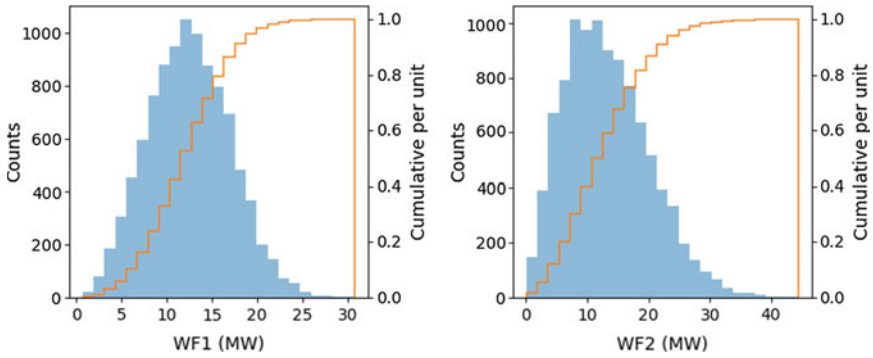
Figure 5.26 shows histograms and the discrete cumulative distribution function (CDF) in percentage representative of the simulation scenarios for the active network element. The histograms have been created using the 10,000 samples created using the specific probability distribution describing the active power of the active network elements (Fig. 5.27).

The variability of the power demands is simulated using a normal (or Gaussian) distribution; the continuous distribution is characterized by two parameters, the mean or expectation of the distribution (μ) and the standard deviation (σ). Details of the Gaussian distribution parameters used for the active power demand in each load are presented in Table 5.3.

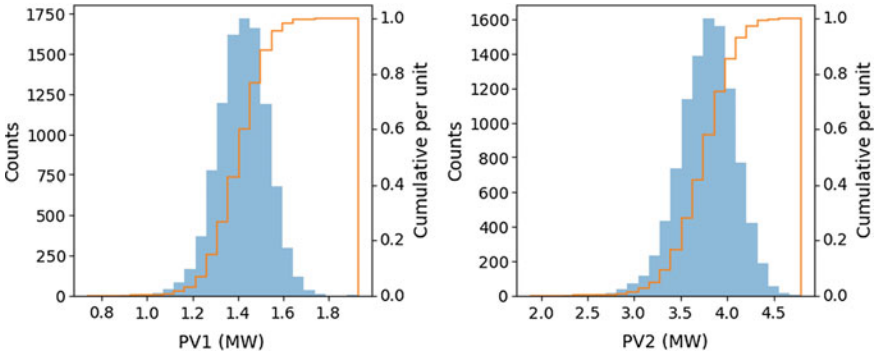
5.10 Results

The 10,000 simulation scenarios are prepared and loaded into a Microsoft Excel file named "*StochData.xlsx*" (4.83 MB). The script named "**ProbabilisticLF**", the main program of the PLF implementation, is called inside DIgSILENT PowerFactory. The main DPL script runs for approximately 65.3 s, and a Microsoft Excel file containing the simulations results of the 10,000 scenarios is produced. The MS Excel file named "*StochResults.xlsx*" contents the simulation results of the bus voltages, loading percentage of the transmission lines, active power losses in each transmission line, transmission lines currents (I_{ij}) and power flows (P_{ij}). The output Microsoft Excel file contents the raw data that should be post-processed in order to create the appropriate probabilistic representation of the variables.

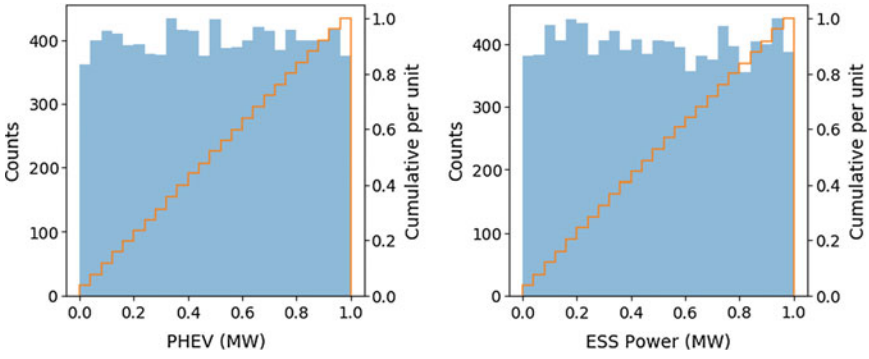
The packages pandas [20] and matplotlib [21] are used in the following to process the generated results. The histogram and empirical cumulative distribution



(a) Wind Farm Power Plants – Weibull distribution (λ,k): WF1 (λ =15 m/s, k =2) and WF2 (λ = 14 m/s, k = 3)



(b) PV Power Plants – Beta distribution (α,β): PV1 (α = 2.3; β = 5.3), PV2 (α = 1.55, β = 4.25)



(c) PHEV charging Station – Beta distribution: PHEV(α = 1.47, β = 5.09)

(d) ESS – Beta distribution: ESS ((α = 0.70, β = 11)

Fig. 5.27 Histograms of synthetic data of all non-CG plants

function are created for the main variables (bus voltages, active power flows, currents and loading percentage of transmission lines and total system power losses). Only the most relevant results are shown here because of space constraints.

Figure 5.28 shows for each bus, the maximum, mean and minimum voltages for the total scenarios are covered in the study. It can be noticed that some buses have an effective voltage control which made this magnitude insensitive to load and generation variations (buses 1 and 15). Other voltages deeply vary depending on the conditions of the considered scenario. The lowest voltage is reached at bus 3 with a value of 0.924 pu, while the highest voltage for all scenarios happens at bus 14 being 1.061 pu.

Figure 5.29 illustrates statistical information concerning the voltage at two loading buses. It is noticed that at bus 14, the range of variation is quite wide, fluctuating from almost 0.93 to 1.06 pu. Instead, at bus 4, the range of voltage variation is considerably narrower, going from 0.99 to 0.935 pu. This is also perceived looking at the σ values indicated for each bus. The wider variation for voltage 14 can be explained by the presence of a wind generator on this bus with a strong variable power injection.

It is worth to mention that the statistical distributions are not following a pure normal (or Gaussian) curve, basically because the different probabilistic distribution functions (three-states, beta, Weibull, normal) used to represent the variability of the active and passive network components (Figs. 5.30 and 5.31).

In Fig. 5.32, the power flows through all the lines and cables, in MW, are depicted providing the maximum, mean and minimum values for all the scenarios. The lines with wider variation are more influenced by the variability of uncontrollable power injections (PV, PHEV, WF). This is noticeable in lines 9–14 and 13–14 which must transfer the variable generation of generator WF1. The histograms, medians and standard deviations of power flows across those lines are shown in Fig. 5.33. As expected, the distributions in both pictures are similar and related to the active power injection by WF1. In fact, when the power injection on

Table 5.3 Details of the Gaussian distribution parameters used for modelling the load demand

Load	Bus	Mean active power (μ in MW)	Standard deviation (% of the μ)
L2	2	21.95	1
L3	3	102.22	25
L4	4	45.14	20
L5	5	5.80	7
L6	6	11.39	15
L9	9	27.34	5
L10	11	4.35	10
L11	11	4.35	2
L12	12	5.88	8
L13	13	12.46	1
L14	14	14.99	4

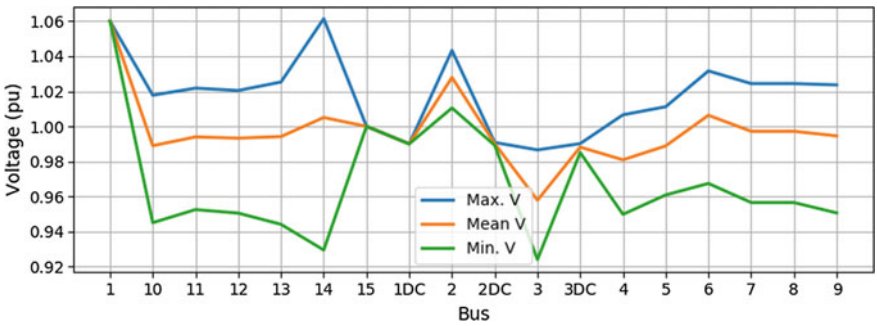


Fig. 5.28 Illustration of voltage variations along the network. The maximum, mean and minimum value at each bus for the 10,000 scenarios are represented

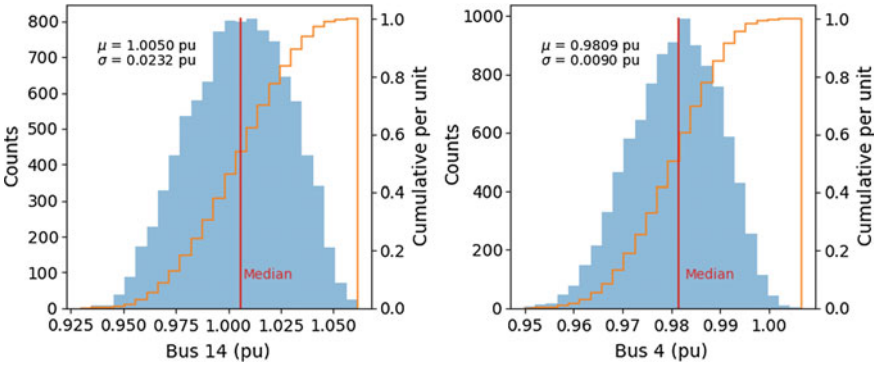


Fig. 5.29 Statistical data related to AC bus voltages. Left: histogram and cumulative distribution of voltage at bus 14. Right: histogram and cumulative distribution at bus 4

bus 14 is low, the power flows from bus 9 to 14 and from 13 to 14 (positive values in Fig. 5.33).

Figure 5.33 displays what happens in the DC part of the system. Here, there are no loads, and the cables are designed to evacuate the power generated by the offshore WF2. This is the reason why power flows are always positive across cable 1–3.

Figure 5.34 depicts the loading of the system branches in all the studied scenarios. It can be noticed that there are no overloads in any scenario and that the most loaded lines are those connecting buses with synchronous generators (line 1–2, line 1–5, line 2–3) (Fig. 5.35).

Finally, Fig. 5.36 shows the histograms of active power losses (MW) and cumulative distribution for the whole system. The mean value is 12.6 MW, but in a few scenarios, the power losses could rise to almost 18 MW, mainly influenced by the generation variability and the path followed by the current to reach the loads.

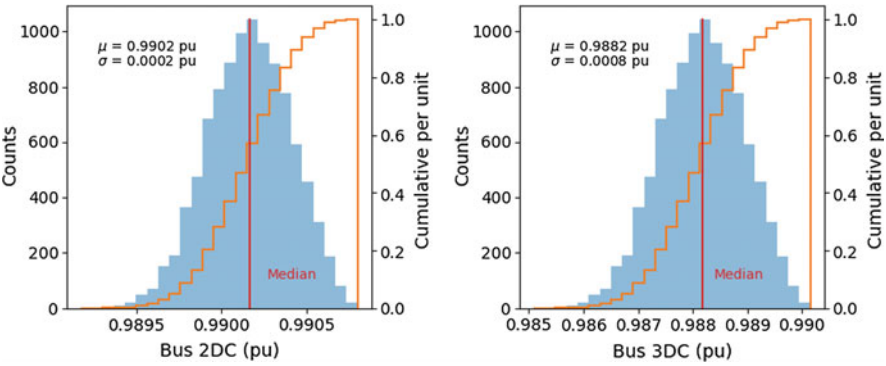


Fig. 5.30 Statistical data related to bus voltage at DC buses. Left: histogram and cumulative distribution of voltage at bus 2DC. Right: histogram and cumulative distribution at bus 3DC

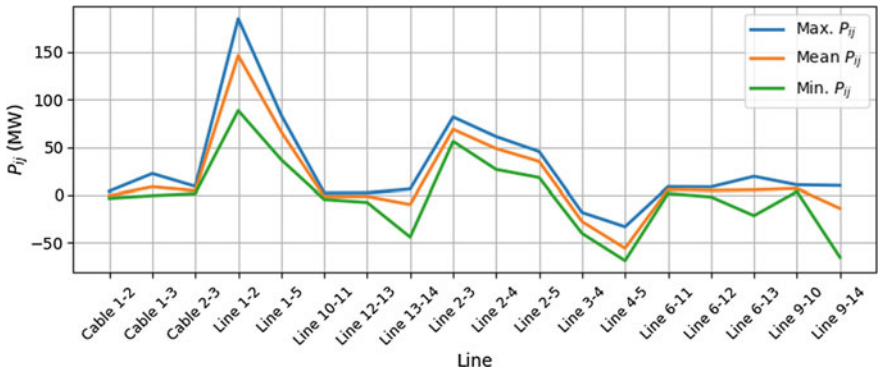


Fig. 5.31 Illustration of lines power flow variations. The maximum, mean and minimum value at each line for the 10,000 scenarios are represented

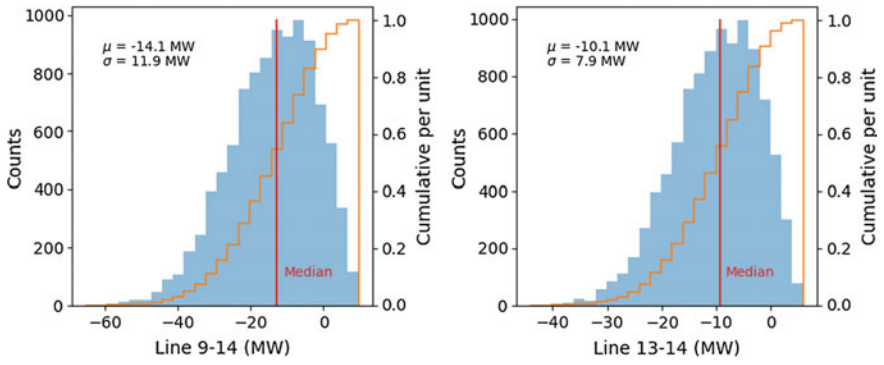


Fig. 5.32 Statistical data related to AC power flows. Left: histogram and cumulative distribution of active power flow through line 9–14 (9 is the sending terminal). Right: histogram and cumulative distribution of active power flow through line 13–14 (13 is sending terminal)

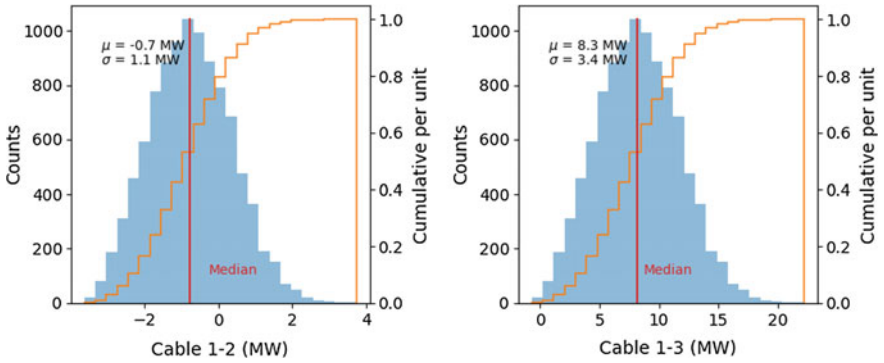


Fig. 5.33 Statistical data related to DC power flows. Left: histogram and cumulative distribution of active power flow through cable 1–2. Right: histogram and cumulative distribution of active power flow through cable 1–3

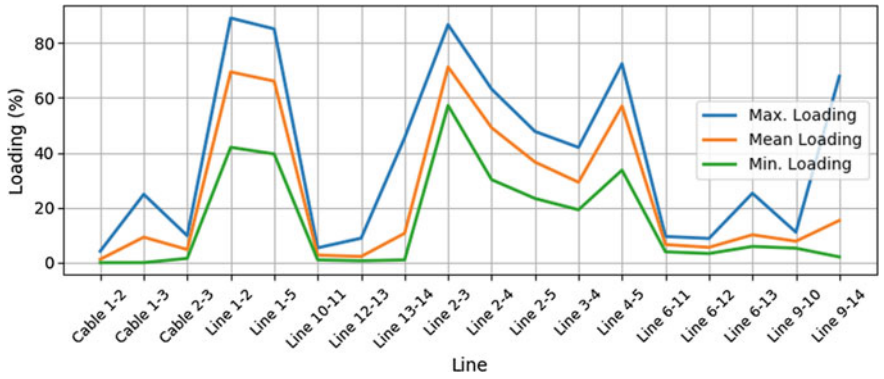


Fig. 5.34 Percentage of loading of system branches. Maximum, mean and minimum values for the 10,000 studied cases are represented

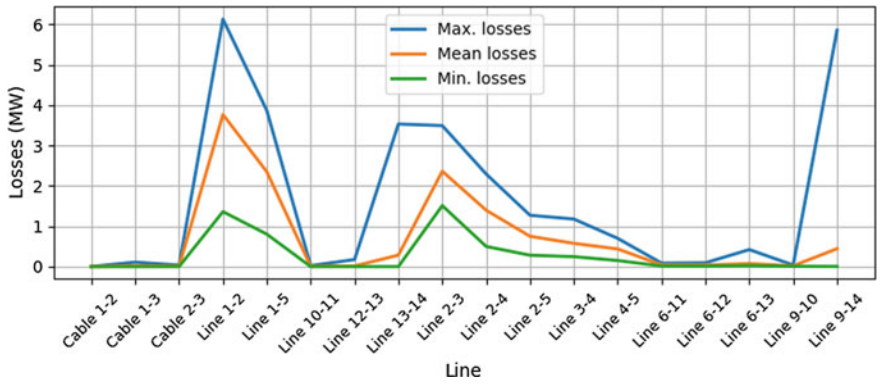


Fig. 5.35 Maximum, mean and minimum losses values (MW) in all branches for the 10,000 studied scenarios

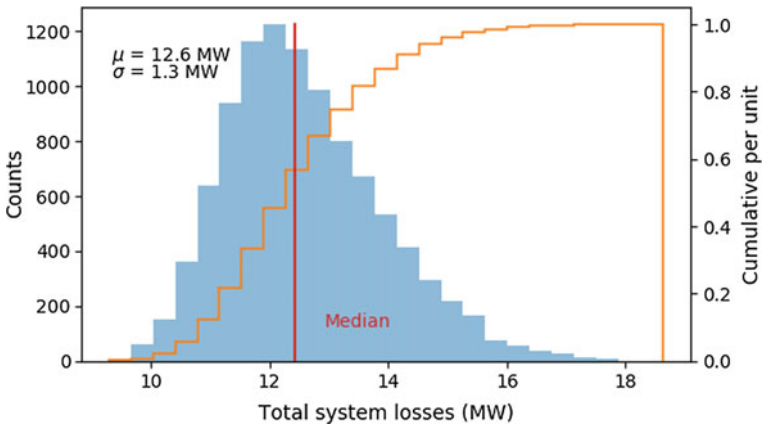


Fig. 5.36 Illustrative histograms of total system active power losses (MW) including cumulative distribution

5.11 Conclusions

Deterministic load-flow analysis is an effective tool that is commonly used to capture the power system operational performance and its state at a certain point of time. However, modern power systems are characterized by an increasing penetration of new technologies adding uncertainties to the design and operation. This chapter presents the *DIgSILENT PowerFactory script language* (DPL) implementation of a DPL script to *perform probabilistic power* flow (PLF) using MCS in order to consider the variability of the stochastic variables in the power system during the assessment of the steady-state performance. The proposed DPL uses as input data the stochastic data coming from an external Microsoft Excel file, and then, the DPL is able to carry on a probabilistic load-flow and exports the results using a Microsoft Excel file. A modular programming approach has been used to provide flexibility and portability of the script. Internal matrices (*IntMatrix*) are used to transfer data between the subscripts; this approach allows to re-use the code in some other application. The suitability of the implemented DPL is illustrated using the classical IEEE 14 buses.

References

1. F. Alvarado, Y. Hu, R. Adapa, Uncertainty in power system modeling and computation, in *Proceedings, 1992 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 754–760 (1992)
2. P. Chen, Z. Chen, B. Bak-Jensen, Probabilistic load flow: a review, in *Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies, 2008. DRPT 2008.* pp. 1586–1591 (2008)

3. G.J. Anders, *Probability Concepts in Electric Power Systems* (Wiley-Interscience, New York, 1990)
4. J. Mur-Amada, A.A. Bayod-Rujula, Wind power variability model Part II—probabilistic power flow, in *9th International Conference on Electrical Power Quality and Utilisation, 2007. EPQU 2007*. pp. 1–6 (2007)
5. B. Borkowska, Probabilistic load flow. IEEE Trans. Power Apparatus Syst. **PAS-93**(3), 752–759 (1974)
6. B.R. Prusty, D. Jena, A critical review on probabilistic load flow studies in uncertainty constrained power systems with photovoltaic generation and a new approach. Renew. Sustain. Energy Rev. **69**, 1286–1302 (2017)
7. R.N. Allan, B. Borkowska, C.H. Grigg, Probabilistic analysis of power flows, in *Proceedings of the Institution of Electrical Engineers*, vol. 121, no. 12, pp. 1551–1556 (1974)
8. A.M. Leite da Silva, S.M.P. Ribeiro, V.L. Arienti, R.N. Allan, M.B. Do Coutto Filho, Probabilistic load flow techniques applied to power system expansion planning. IEEE Trans. Power Syst. 5(4), 1047–1053 (1990)
9. F. Milano, *Power system modelling and scripting*, 1st edn. (Springer, New York, 2010)
10. G.W. Stagg, A.H. El-Abiad, *Computer Methods in Power System Analysis* (McGraw-Hill series in electronic systems) (McGraw-Hill, New York, 1968), 427 p
11. H.E. Brown, *Solution of Large Networks by Matrix Methods*, 2nd edn. (Wiley, New York, Chichester, 1985), pp. xv, 320 p
12. J. Arrillaga, C.P. Arnold, *Computer Analysis of Power Systems* (Wiley, Chichester, England, New York, 1990), pp. xiii, 361 p
13. T.J. Williams, C. Crawford, Probabilistic power flow modeling: renewable energy and PEV grid interactions, presented at the The Canadian Society for Mechanical Engineering Forum 2010, CSME FORUM 2010, Victoria, British Columbia, Canada, 7–9 June 2010
14. Z. Pei, S.T. Lee, Probabilistic load flow computation using the method of combined cumulants and Gram-Charlier expansion. IEEE Trans. Power Syst. **19**(1), 676–682 (2004)
15. Z. Hu, W. Xifan, A probabilistic load flow method considering branch outages. IEEE Trans. Power Syst. **21**(2), 507–514 (2006)
16. F. Coroiu, C. Velicescu, C. Barbulescu, Probabilistic and deterministic load flows methods in power systems reliability estimation, in *EUROCON—International Conference on Computer as a Tool (EUROCON), 2011 IEEE*, pp. 1–4 (2011)
17. M. Aien, R. Ramezani, S.M. Ghavami, Probabilistic load flow considering wind generation uncertainty. ETASR Eng. Technol. Appl. Sci. Res. **1**(4), 126–132 (2011)
18. P. Jorgensen, J.S. Christensen, J.O. Tande, Probabilistic load flow calculation using Monte Carlo techniques for distribution network with wind turbines, in *8th International Conference on Harmonics And Quality of Power, 1998. Proceedings*, vol. 2, pp. 1146–1151 (1998)
19. *DIgSILENT PowerFactory 2016—User Manual* (DIgSILENT GmbH, Gomaringen, 2016)
20. W. McKinney, Data structures for statistical computing in python, in *Proceedings of the 9th Python in Science Conference*, pp. 51–56 (2010)
21. J.D. Hunter, Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**, 90–95 (2007)